ТОШКЕНТ АХБОРОТ ТЕХНОЛОГИЯЛАРИ УНИВЕРСИТЕТИ
АХБОРОТ ТЕХНОЛОГИЯЛАРИ УНИВЕРСИТЕТИ

ИНФОРМАТИКА КАФЕДРАСИ

C++ ТИЛИ

услубий қўлланма

ТОШКЕНТ 2011

Муаллиф: Ш. А. Назиров., Р. В. Қобулов**.** «Объектга йўналтирилган дастурлаш тиллари» фанидан ўқув қўлланма. /ТУИТ. 260 б. Тошкент, 2011.

Қўлланма мақсади – назарий билимларни мустахкамлаш ва объектга йўналтирилган дастурлар яратиш ва жорий этиш амалий кўникмаларини хосил қилишдан иборат.

Қўлланма синфлар, ворислик, амалларни қўшимча юклаш, истиснолардан фойдаланиб объектли дастурлашга бағишланган. Бундан ташқари стандарт оқимлар библиотекасидан фойдаланиб дастурлашга бағишланган мавзулар хам берилган. Қўлланма касб хунар коллежлари ва олий ўқув юртлари профессор – ўқитувчилари ва талабалари учун мўлжалланган.

Тошкент Давлат Университети ўқув – услубий кенгаши томонидан тасдиқланган.

Тақризчилар:

ЎзМУ доценти,
ф-м.ф.н. А. Хайдаров

«Математика ва информацион
технологиялари» институти
катта илмий ходими,
ф-м.ф.н., доцент Ф. Н. Нуралиев

К И Р И Ш

Кейинги йилларда амалий дастурчиларга жуда кўп интеграцион дастур тузиш муҳитлари таклиф этилаяпти. Бу муҳитлар у ёки бу имкониятлари билан бир–биридан фарқ қилади. Аксарият дастурлаштириш муҳитларининг фундаментал асоси C++ тилига бориб тақалади. C++ тили Б.Страуструп томонидан яратилган. Америка миллий стандартлар институти (American National Standards Institute – ANSI) раҳбарлиги остидаги Стандартларни аккредитивлаш комитети (Accredited Standards Committee) C++ тилининг халқаро стандартини тузди.

C++ стандарти айни вақтда ISO – International Standards Organization (Стандартлаш бўйича ҳалқаро ташқилот) стандарти деб ҳам номланади.

Вақт ўтиши билан дастурчилар олдига қуйилган масалалар ўзгариб боряпти. Бундан йигирма йил олдин дастурлар катта ҳажмдаги маълумотларни қайта ишлаш учун тузилар эди. Бунда дастурни ёзувчи ҳам, унинг фойдаланувчиси ҳам компьютер соҳасидаги билимлар бўйича профессионал бўлиши талаб этиларди. Ҳозирда эса кўпгина ўзгаришлар рўй берди. Компьютер билан кўпроқ унинг аппарат ва дастурий таъминоти ҳақида тушунчаларга эга бўлмаган кишилар ишлашяпти. Компьютер одамлар томонидан уни чуқур ўрганиш воситаси эмас, кўпроқ ўзларининг олдиларига қўйилган, ўзларининг ишларига тегишли бўлган муаммоларини ечиш инструменти бўлиб қолди.

Дастурлашга талабни ўзгариши нафақат тилларнинг ўзгаришига балки уни ёзиш технологиясини ҳам ўзгаришига олиб келди. Дастурлаш эволюцияси тарихида кўпгина босқичлар бўлишига қарамай биз бу қўлланмада процедурали дастурлашдан объектларга мўлжалланган дастурлашга ўтишни қараймиз.

Қўлланмада асосий эътибор дастурлар тузиш усулларига қаратилган бўлиб, келтирилган материаллар кетма-кетликда берилган, унинг ёрдамида ўқувчи компьютерда тез мустақил ҳолда дастур тузиш имконига эга бўлади ва замонавий объектга йўналтирилган дастурлаш технологиялари билан танишади. Қўлланма икки қисмдан иборат бўлиб, биринчи қисм 1- 8 боблардан иборат бўлиб C++ тилида структурали дастурлашга бағишланган. Иккинчи қисм 9- 17 боблардан иборат бўлиб C++ тилида объектли дастурлашга бағишланган. Хар бир қисмнинг охирги икки боби динамик хотира билан ишлаш ва библиотекалар яратишга бағишланган.

Ўйлаймизки қўлланма билан танишган касб-ҳунар коллежлари ва академик лицей ўқувчилари, Олий ўқув юртлари талабалари, магистрлари ва аспирантлари компьютерда C++ тилида ўз дастурларини яратишга киришади.

1 боб. Тил лексик асослари

1.1. Алфавит ва хизматчи сўзлар

Алфавит. C++ алфавитига қуйидаги символлар киради.

- Катта ва кичик лотин алфавити харфлари (`A,B,..,Z,a,b,…,z`)
- Рақамлар: `0,1,2,3,4,5,6,7,8,9`
- Махсус символлар: `" , {} | [] () + - / % \ ; ' . : ? < = > _ ! & * # ~ ^`
- Кўринмайдиган символлар ("умумлашган бўшлик символлари"). Лексемаларни ўзаро ажратиш учун ишлатиладиган символлар (мисол учун бўшлик, табуляция, янги қаторга ўтиш белгилари).

Изохларда, сатрларда ва символли константаларда бошқа литераллар, масалан рус харфлари ишлатилиши мумкин.

C++ тилида олти хил турдаги лексемалар ишлатилади: эркин танланадиган ва ишлатиладиган идентификаторлар, хизматчи сўзлар, константалар(константа сатрлар), амаллар(амаллар белгилари), ажратувчи белгилар.

Идентификатор. Идентификаторлар лотин харфлари,остки чизиқ белгиси ва сонлар кетма кетлигидан иборат бўлади. Идентификатор лотин харфидан ёки остки чизиш белгисидан бошланиши лозим.

Мисол учун:

```
A1, _MAX,  adress_01, RIM, rim
```

Катта ва кичик харфлар фарқланади, шунинг учун охирги икки идентификатор бир биридан фарқ қилади.

Borland компиляторларидан фойдаланилганда номнинг биринчи 32 харфи ,баъзи компиляторларда 8 та харфи инобатга олинади. Бу холда `NUMBER_OF_TEST` ва `NUMBER_OF_ROOM` идентификаторлари бир биридан фарқ қилмайди.

Хизматчи сўзлар. Тилда ишлатилувчи яъни дастурчи томонидан ўзгарувчилар номлари сифатида ишлатиш мумкин бўлмаган идентификаторлар хизматчи сўзлар ёки калит сўзлар дейилади. Компиляторнинг техник документациясида барча заҳираланган сўзларнинг рўйхати туради.

Ўзгарувчиларни таърифлаш. C++ тилида ўзгарувчини аниқлаш учун компьютерга унинг типи (масалан, `int`, `char` ёки `float`) хамда исми хақида ҳақида маълумот берилади. Бу ахборот асосида компиляторга ўзгарувчи учун қанча жой ажратиш лозим ва бу ўзгарувчида қандай турдаги

қиймат сақланиши мумкинлиги ҳақида маълумот аниқ бўлади. Ўзгарувчи номи идентификатор бўлиб, хизматчи сўзлардан фарқли бўлиши керак.

Ҳар бир ячейка бир байт ўлчовга эга. Агар ўзгарувчи учун кўрсатилган тип 4 байтни талаб қилса, унинг учун тўртта ячейка ажратилади. Айнан ўзгарувчини типига мувофиқ равишда компилятор бу ўзгарувчи учун қанча жой ажратиш кераклигини аниқлайди.

Компютерда қийматларни ифодалаш учун битлар ва байтлар қўлланилади ва хотира байтларда ҳисобланади.

1.2. Ўзгармаслар

Ўзгармаслар турлари. Ўзгарувчилар каби ўзгармаслар ҳам маълумотларни сақлаш учун мўлжалланган хотира ячейкаларини ўзида ифодалайди. Ўзгарувчилардан фарқли равишда улар дастурни бажарилиши жараёнида қиймати ўзгармайди. Ўзгармас эълон қилиниши билан унга қиймат бериш лозим, кейинчалик бу қийматни ўзгартириб бўлмайди.

C++ тилида икки турдаги, литерал ва номланган ўзгармаслар аниқланган.

Литералли ўзгармаслар тўғридан-тўғри дастурга киритилади. Масалан:

```
int myAge =39;
```

Бу ифодада `MyAge` `int` типидаги ўзгарувчи, 39 сони эса литерал ўзгармасдир.

Белгили ўзгармаслар. Белгили ўзгармаслар одатда бир байт жойни эгаллайди ва бу 256 хил белгини сақлаш учун етарлидир. `Char` типи қийматларини 0..255 сонлар тўпламига ёки `ASCII` белгилар тўпламига интерпретация қилиш мумкин.

`ASCII` белгилари деганда компьютерларда қўлланиладиган стандарт белгилар тўплами тушунилади. `ASCII` - бу `American Standard Code for Information Interchange` (Американинг ахборот алмашиниши учун стандарт коди) деган маънони англатади.

Махсус белгилар. C++ компиляторида текстларни форматловчи бир нечта махсус белгилардан фойдаланилади. (Улардан энг кўп тарқалгани жадвалда келтирилган). Бу белгиларни дастурда ишлатишда «тескари слеш»дан фойдаланамиз. Тескари слешдан кейин бошқарувчи белги ёзилади. Масалан, табуляция белгисини дастурга қўйиш учун қуйидагича ёзувни ёзиш керак.

```
char tab ='\t';
```

Бу мисолдаги `char` типидаги ўзгарувчи `\t` қийматини қабул қилади. Махсус белгилар ахборотларни экранга, файлга ва бошқа чиқариш қурилмаларига чиқаришда форматлаш учун қўлланилади.

Махсус `'\'` символидан бошланган символлар эскейп символлар дейилади.Символли константа қиймати символнинг компьютерда кабул қилинган сонли кодига тенгдир.

ESC (эскейп) символлар жадвали:

Езилиши	Ички коди	Символи(номи)	Маъноси
\a	0x07	bel (audible bell)	Товуш сигнали
\b	0x08	bs (bascspase)	Бир кадам кайтиш
\f	0x0C	ff (form feed)	Сахифани утказиш
\n	0x0A	lf (line feed)	Каторни утказиш
\r	0x0D	cr (carriage return)	Кареткани кайтариш
\t	0x09	ht (horizontal tab)	Горизонтал табуляция
\v	0x0B	vt (vertical tab)	Вертикал табуляция
\\	0x5C	\ (bacslash)	Тескари чизик
\'	0x27	‘ (single out)	Апостриф (оддий қавс)
\''	0x22	" (double quote)	Иккилик қавс
\?	0x3F	? (question mark)	Савол белгиси
\000	000	ихтиёрий (octal number)	Символ саккизлик коди
\xhh	0xhh	ихтиёрий (hex number)	Символ унолтилик коди

Маълумотларнинг бутун сон тури. Бутун сонлар ўнлик, саккизлик ёки ўн олтилик саноқ системаларида берилиши мумкин.

Ўнлик саноқ системасида бутун сонлар 0-9 рақамлари кетма кетлигидан иборат бўлиб, биринчи раками 0 бўлиши керак эмас.

Саккизлик саноқ системасида бутун сонлар 0 билан бошланувчи 0-7 рақамларидан иборат кетма кетликдир.

Ўн олтилик саноқ системасида бутун сон `0x` ёки `0X` билан бошланувчи 0-9 ракамлари ва `a-f` ёки `A-F` харфларидан иборат кетма кетликдир.

Масалан 15 ва 22 ўнлик сонлари саккизликда 017 ва 026, ўн олтиликда 0xF ва 0x16 шаклда тасвирланади.

Маълумоларнинг узун бутун сон тури.

Охирига `l` ёки `L` харфлари қўйилган ўнлик, саккизлик ёки ўн олтилик бутун сон.

Маълумотларнинг ишорасиз (`unsigned`) бутун сон тури:

Охирига `u` ёки `U` харфлари қўйилган ўнлик, саккизлик ёки ўн олтилик оддий ёки узун бутун сон.

Маълумотларнинг хақиқий сон тури. Маълумотларнинг хақиқий сон тури олти кисмдан иборат бўлиши мумкин: бутун кисм, нуқта, каср кисм, `e` ёки `E` белгиси, ўнлик даража , `F` ёки `f` суффикслари.

Масалан : `66.  .0  .12  3.14F  1.12e-12`

Маълумоларнинг узун хақиқий сон тури :

Охирига `L` ёки `l` суффикслари куйилган хақиқий сон.

Масалан: `2E+6L;`

Мантиқий константа. Мантиқий константалар `true`(рост) ва `false`(ёлғон) қийматлардан иборат. Ички кўриниши `false` – 0, ихтиёрий бошқа қиймат `true` деб қаралади.

Сатрли константа. Сатрли константалар C++ тили константаларига кирмайди, балки лексемалари алохида типи хисобланади. Шунинг учун адабиётда сатрли константалар сатрли лексемалар деб хам аталади..

Сатрли константа бу иккилик қавсларга олинган ихтиёрий символлар кетма кетлигидир. Мисол учун " Мен сатрли константаман".

Сатрлар орасига эскейп символлар хам кириши мумкин. Бу символлар олдига \ белгиси қўйилади. Мисол учун :

"\n Бу сатр \n уч каторга \n жойлашади".

Сатр символлари хотирада кетма кет жойлаштирилади ва хар бир сатрли константа охирига автоматик равишда компилятор томонидан `'\0'` символи қўшилади. Шундай сатрнинг хотирадаги хажми символлар сони+1 байтга тенгдир.

Кетма кет келган ва бўшлик, табуляция ёки сатр охири белгиси билан ажратилган сатрлар компиляция даврида битта сатрга айлантирилади. Мисол учун:

"Салом" "Тошкент "

сатрлари битта сатр деб қаралади.

"Салом Тошкент"

Бу қоидага бир неча каторга ёзилган сатрлар хам бўйсинади. Мисол учун :

"Ўзбекистонга "

"бахор "

"келди"

қаторлари битта қаторга мос:

"Ўзбекистонга бахор келди"

Агар сатрда `'\'` белгиси учраса ва бу белгидан сўнг то `'\n'` сатр охири белгисигача бўшлик белгиси келса бу бўшлик белгилари `'\'` ва `'\n'` белгиси билан бирга сатрдан ўчирилади. Сатрнинг ўзи кейинги сатрда келган сатр билан қўшилади.

"Ўзбекистонга \ "

" бахор\

" келди"

қаторлари битта қаторга мос:

"Узбекистонга бахор келди"

Номланган ўзгармаслар. Белгили ўзгармас – бу номга эга бўлган ўзгармасдир. C++ тилида белгили ўзгармасни аниқлашнинг икки усули мавжуд:

1. `#define` директиваси ёрдамида ўзгармасни аниқлаш.
2. `const` калитли сўзи орқали ўзгармасни аниқлаш.

Анъанавий усул ҳисобланган `#define` директиваси орқали ўзгармасни аниқлашни қуйидаги мисолда кўришимиз мумкин.

```
#define StudentsPerClass 15
```

Бу ҳолда `StudentsPerClass` ўзгармас ҳеч қандай типга тегишли бўлмайди.

Препроцессор `StudentsPerClass` сўзига дуч келганида уни `15` литералига алмаштиради.

C++ тилида `#define` директивасидан ташқари ўзгармасни аниқлашнинг нисбатан қулайроқ бўлган янги усули ҳам мавжуд:

```
const unsigned short int StudentsPerClass=15
```

Бу мисолда ҳам белгили константа `StudentsPerClass` номи билан аниқланаяпти ва унга `unsigned short int` типи бериляпти. Бу усул бир қанча имкониятларга эга бўлиб у сизнинг дастурингизни кейинги ҳимоясини енгиллаштиради. Бу ўзгармасни олдингисидан энг муҳим афзаллиги унинг типга эгалигидир.

Номланган константалар қуйидаги шаклда киритилади:

`const` тип константа_номи=константа_қиймати.

Мисол учун:

```
const double EULER=2.718282;
const long M=99999999;
const R=765;
```

Охирги мисолда константа типи кўрсатилмаган, бу константа `int` типига тегишли деб хисобланади.

Белгили ўзгармасларни литерал ўзгармасларга нисбатан ишлатиш қулайроқдир. Чунки агарда бир хил номли литералли ўзгарувчини қийматини ўзгартирмоқчи бўлсангиз бутун дастур бўйича уни ўзгартиришга тўғри келади, белгили ўзгармасларни эса фақатгина бирининг қийматини ўзгартириш етарли.

Тўплам ўзгармаслари. Бундай ўзгармасларни ҳосил қилиш учун янги берилган маълумотлар типлари тузилади ва ундан сўнг бу типга тегишли

ўзгармасли қийматлар тўплами билан чегараланган ўзгарувчилар аниқланади.

Сановчи константалар `enum` хизматчи сўзи ёрдамида киритилиб, `int` типидаги сонларга қулай сўзларни мос қўйиш учун ишлатилади.

Мисол учун:

```
enum{one=1,two=2,three=3};
```

Агар сон қийматлари кўрсатилмаган бўлса энг чапки сузга 0 қиймати берилиб колганларига тартиб бўйича усувчи сонлар мос куйилади:

```
enum{zero,one,two};
```

Бу мисолда автоматик равишда константалар қуйидаги қийматларни кабул килади:

```
zero=0, one=1, two=2;
```

Константалар аралаш кўринишда киритилиши хам мумкин:

```
Enum(zero,one,for=4,five,seeks}.
```

Бу мисолда автоматик равишда константалар қуйидаги қийматларни кабул килади:

```
zero=0, one=1, for=4;five=5,seeks=6;
```

Саноқли типларни ҳосил қилиш учун `enum` калитли сўзи ва ундан кейин тип номи ҳамда фигурали қавс ичида вергуллар билан ажратилган ўзгармас қийматлари рўйхати ишлатилади. Масалан, `RANG` номли саноқли тип деб эълон қилайлик ва унинг учун 5 та `QIZIL, KUK, YASHIL, OQ, QORA` қийматларини аниқлайлик.

```
enum RANG { QIZIL, KUK, YASHIL, OQ, QORA };
```

Бунда ифода иккита ишни бажаради:

1. `RANG` номли янги саноқли тип ҳосил қилади;
2. Қуйидаги белгили ўзгармасларни аниқлайди.

0 қиймат билан `QIZIL;`

1 қиймат билан `KUK;`

2 қиймат билан `YASHIL` ва ҳоказо;

Ҳар бир саноқли ўзгармас бирор бир аниқланган бутун қийматга мос келади.

Бошланғич ҳолатда ўзгармасларга 0 дан бошлаб қиймат берилади. Лекин, ихтиёрий ўзгармасга бошқа қийматни ўзлаштириш ҳам мумкин. Бунда уларга қиймат бериш ўсиш тартибида бўлиши лозим. Масалан,

```
enum RANG{QIZIL=100,KUK=200,YASHIL=300,OQ,QORA=500};
```

кўринишда саноқли типни аниқласак `QIZIL` ўзгармаси 100 га, `KUK` – 200 га, `Yashil` – 300 га, `OQ` –301 га, `QORA` – 500 га тенг бўлади.

Яна бир мисол:

```
enum BOOLEAN {NO, YES};
```

Константалар қийматлари:

```
NO=0, YES=1;
```

1.3. Амаллар

Ўзлаштириш амали. Ўзлаштириш амали (=) ўзидан чап томонда турган операнд қийматини тенглик белгисидан ўнг томондагиларни ҳисобланган қийматига алмаштиради. Масалан,

```
x = a+b;
```

ифодаси x операндга a ва в ўзгарувчиларни қийматларини қўшишдан ҳосил бўлган натижани ўзлаштиради.

Ўзлаштириш амалиидан чапда жойлашган операнд адресли операнд ёки l–қиймат дейилади. Ўзлаштириш амалидан ўнгда жойлашган операнд операцион операнд ёки r–қиймат дейилади.

Ўзгармаслар фақатгина r–қиймат бўлиши мумкин ва ҳеч қачон адресли операнд бўла олмайди, чунки дастурнинг бажарилиши жараёнида ўзгармас қийматини ўзгартириб бўлмайди.

```
35 = x //
```

нотўғри!

l–қиймат эса r–қиймат бўлиши мумкин.

Қиймат бериш амали = бинар амал бўлиб чап операнди одатда ўзгарувчи ўнг операнди одатда ифодага тенг бўлади. Мисол учун

```
z=4.7+3.34
```

Бу қиймати 8.04 га тенг ифодадир. Бу қиймат Z ўзгарувчига хам берилади.

Бу ифода охирига нуқта вергуль; белгиси қўйилганда операторга айланади.

```
z=4.7+3.34
```

Битта ифодада бир неча қиймат бериш амаллари қўлланилиши мумкин. Мисол учун:

```
c=y=f=4.2+2.8;
```

Математик амаллар. C++ тилида 5 та асосий математик амаллар қўлланилади: қўшиш (+), айириш (-), кўпайтириш (*), бутун сонга бўлиш (/) ва модул бўйича бўлиш (%)(қолдиқни олиш). Амаллар одатда унар яъни битта операндга қўлланиладиган амалларга ва бинар яъни икки операндга қўлланиладиган амалларга ажратилади.

Бинар амаллар аддитив яъни + қўшиш ва – айириш амалларига , хамда мультипликатив яъни * кўпайтириш, / бўлиш ва % модуль олиш амалларига ажратилади.

Бутун сонга бўлиш одатдаги бўлишдан фарқ қилади. Бутун сонга бўлишдан ҳосил бўлган бўлинманинг фақатгина бутун қисми олинади.

Бутун сонни бутун сонга бўлганда натижа бутун сонгача яхлитланади. Мисол учун 20/3=6; (-20)/3=-6; 20/(-3)=-6.

Модуль амали бутун сонни бутун сонга бўлишдан хосил бўладиган қолдиққа тенгдир. Агар модуль амали мусбат операндларга қўлланилса, натижа хам мусбат бўлади, акс холда натижа ишораси компиляторга боғликдир.

Масалан, 21 сонини 4 га бўлсак 5 сони ва 1 қолдиқ ҳосил бўлади. 5 бутун сонга бўлишни қиймати, 1 эса қолдиқни олиш қиймати ҳисобланади.

Инкремент ва декремент. Дастурларда ўзгарувчига `1` ни қўшиш ва айириш амаллари жуда кўп ҳолларда учрайди. `C++` тилида қийматни `1` га ошириш инкремент, `1` га камайтириш эса декремент дейилади. Бу амаллар учун махсус операторлар мавжуддир.

Инкремент оператори `(++)` ўзгарувчи қийматини `1` га оширади, декремент оператори `(--)` эса ўзгарувчи қийматини `1` га камайтиради. Масалан, `c` ўзгарувчисига `1` қийматни қўшмоқчи бўлсак қуйидаги ифодани ёзишимиз лозим.

```
c++ //c ўзгарувчи қийматини 1 га оширдик.
```

Бу ифодани қуйидагича ёзишимиз мумкин эди.

```
c=c+1;
```

Бу ифода ўз навбатида қуйидаги ифодага тенг кучли:

```
c+=1;
```

Префикс ва постфикс. Инкремент оператори ҳам, декремент оператори ҳам икки вариантда ишлайди: префиксли ва постфиксли. Префиксли вариантда улар ўзгарувчидан олдин `(++Age)`, постфиксли вариантда эса ўзгарувчидан кейин `(Age++)` ёзилади.

Оддий ифодаларда бу вариантларни қўлланишида фарқ катта эмас, лекин бир ўзгарувчига бошқа ўзгарувчининг қийматини ўзлаштиришда уларнинг қўлланилиши бошқача характерга эга. Префексли оператор қиймат ўзлаштирилгунча, постфиксли оператор эса қиймат ўзлаштирилгандан кейин бажарилади. Мисол учун `i` қиймати 2 га тенг бўлсин, у холда `3+(++i)` ифода қиймати 6 га, `3+i++` ифода қиймати 5 га тенг бўлади. Иккала холда хам `i` қиймати 3 га тенг бўлади.

Амаллар устиворлиги. Мураккаб ифодаларда қайси амал биринчи навбатда бажарилади, қўшишми ёки кўпайтиришми? Масалан:

```
x=5+3*8;
```

ифодада агарда биринчи қўшиш бажарилса натижа `64` га, агарда кўпайтириш биринчи бажарилса натижа `29` га тенг бўлади.

Ҳар бир оператор приоритет қийматига эга. Кўпайтириш қўшишга нисбатан юқорироқ приоритетга эга. Шунинг учун бу ифода қиймати `29` га тенг бўлади.

Агарда иккита математик ифоданинг приоритети тенг бўлса, улар чапдан ўнгга қараб кетма – кет бажарилади.

Демак

```
x=5+3+8*9+6*4
```

ифодада биринчи кўпайтириш амаллари чапдан ўнгга қараб бажарилади `8*9=72` ва `6*4=24`. Кейин бу ифода соддароқ кўриниш ҳосил қилади.

```
x=5+3+72+24
```

Энди қўшишни ҳам худди шундай чапдан унга қараб бажарамиз:

```
5+3=8; 8+72= 80; 80+24=104;
```

Лекин, барча амаллар ҳам бу тартибга амал қилмайди. Масалан, ўзлаштириш амалии ўнгдан чапга қараб бажарилади.

Аддитив амалларининг устиворлиги мультипликатив амалларининг устиворлигидан пастрокдир.

Унар амалларнинг устиворлиги бинар амаллардан юқоридир.

Мураккаб қиймат бериш амали. C++ тилида мураккаб қиймат бериш амали мавжуд бўлиб, умумий кўриниши қуйидагичадир:

Ўзгарувчи_номи амал= ифода;

Бу ерда амал қуйидаги амаллардан бири * , / , % , + , - , & ,^ , | , << , >>.

Мисол учун:

`x+=4` ифода `x=x+4` ифодага эквивалентдир;

`x*=a` ифода `x=x*a` ифодага эквивалентдир;

`x/=a+b` ифода `x=x/(a+b)` ифодага эквивалентдир;

`x>>=4` ифода `x=x>>4` ифодага эквивалентдир;

Имло белгилари амал сифатида. C++ тилида баъзи бир имло белгилари хам амал сифатида ишлатилиши мумкин. Бу белгилардан оддий () ва квадрат [] қавслардир. Оддий қавслар бинар амал деб қаралиб ифодаларда ёки функцияга мурожаат қилишда фойдаланилади. Функцияга мурожаат қилиш қуйидаги шаклда амлга оширилади:

<функция номи> (<аргументлар рўйхати>). Мисол учун `sin(x)` ёки `max(a,b)`.

Мураккаб ифодаларни тузишда ички қавслардан фойдаланилади. Масалан, сизга секундларнинг умумий сони кейин эса барча қаралаётган

одамлар сони, ундан кейин эса уларнинг кўпайтмасини ҳисоблаш керак бўлсин.

```
TotalPersonSeconds=((NumMinutesToThink+
NumMinutesToType)*60*(PeopleInTheOffice+
PeopleOnVocation ))
```

Бу ифода қуйидагича бажарилади. Олдин `NumMinutesToThink` ўзгарувчисининг қиймати `NumMinutesToType` ўзгарувчиси қийматига қўшилади. Кейин эса ҳосил қилинган йиғинди 60 га кўпайтирилади. Бундан кейин `PeopleInTheOffice` ўзгарувчи қиймати `PeopleOnVocation` қийматига қўшилади. Кейин эса секундлар сони кишилар сонига кўпайтирилади.

Квадрат қавслардан массивларга мурожаат килишда фойдаланилади. Бу мурожаат қуйидагича амалга оширилади:

<массив номи>[<индекс>]. Мисол учун `a[5]` ёки `b[n][m]`.

Вергуль символини ажратувчи белги деб хам караш мумкин амал сифатида хам караш мумкин. Вергуль билан ажратилган амаллар кетма-кетлиги бир амал деб қаралиб, чапдан ўнгга хисобланади ва охирги ифода қиймати натижа деб каралади. Мисол учун:

`d=4,d+2` амали натижаси 8 га тенг.

Шартли амал. Шартли амал тернар амал дейилади ва учта операнддан иборат бўлади:

<1-ифода>?<2-ифода>:<3-ифода>

Шартли амал бажарилганда аввал 1- ифода хисобланади. Агар 1-ифода қиймати 0 дан фарқли бўлса 2- ифода хисобланади ва қиймати натижа сифатида кабул килинади, акс холда 3-ифода хисобланади ва қиймати натижа сифатида кабул килинади.

Мисол учун модульни хисоблаш: `x<0?-x:x` ёки иккита сон кичигини хисоблаш

```
a<b?a:b .
```

Шуни айтиш лозимки шартли ифодадан хар қандай ифода сифатида фойдаланиш мумкин. Агар `F FLOAT` типга,а `N – INT` типга тегишли бўлса ,

```
(N > 0) ? F : N
```

ифода `N` мусбат ёки манфийлигидан катъи назар `DOUBLE` типига тегишли бўлади.

Шартли ифодада биринчи ифодани қавсга олиш шарт эмас.

Мантиқий амаллар. Дастурлашда бир эмас балки бир нечта шартли ифодаларни текшириш зарурияти жуда кўп учрайди. Масалан, `x` ўзгарувчиси `y` ўзгарувчисидан, `y` эса ўз навбатида `z` ўзгарувчисидан каттами шарти бунга мисол бўла олади. Бизнинг дастуримиз мос амални бажаришдан олдин бу иккала шарт рост ёки ёлғонлигини текшириши лозим.

Қуйидаги мантиқ асосида юқори даражада ташқил қилинган сигнализация системасини тасаввур қилинг. Агарда эшикда сигнализация

ўрнатилган бўлса ВА кун вақти кеч соат олти ВА бугун байрам ЁКИ дам олиш куни БЎЛМАСА полиция чақирилсин. Барча шартларни текшириш учун C++ тилининг учта мантиқий оператори ишлатилади. Улар жадвалда келтирилган

Амал	Белги	Мисол
ВА	&&	1ифода && 2ифода
ЁКИ	||	1ифода||2ифода
ИНКОР	!	!ифода

Мантиқий амаллар || (дизъюнкция); && (конъюнкция); !(инкор) амаллари деб аталади. Мантикий амалларни бутун сонларга қўллаш мумкин. Бу амалларнинг натижалари қуйидагича аниқланади:

x||y амали 1 га тенг агар x>0 ёки y>0 бўлса, аксинча 0 га тенг

x&&y амали 1 га тенг агар x>0 ва y>0 бўлса, аксинча 0 га тенг

!x амали 1 га тенг агар x>0 бўлса, аксинча 0 га тенг

Бу мисолларда амаллар устиворлиги ошиб бориш тартибида берилгандир.

Инкор ! амали унар қолганлари бинар амаллардир.

Мантиқий кўпайтириш амали. Мантиқий кўпайтириш амали иккита ифодани ҳисоблайди, агар иккала ифода true қиймат қайтарса ВА оператори ҳам true қиймат қайтарди. Агарда сизнинг қорнингиз очлиги рост бўлса ВА сизда пул борлиги ҳам рост бўлса сиз супермаркетга боришингиз ва у ердан ўзингизга тушлик қилиш учун бирор бир нарса харид қилишингиз мумкин. Ёки яна бир мисол, масалан,

```
(x==5)&&(y==5)
```

мантиқий ифодаси агарда x ва y ўзгарувчиларини иккаласининг ҳам қийматлари 5 га тенг бўлсагина true қиймат қайтаради. Бу ифода агарда ўзгарувчилардан бирортаси 5 га тенг бўлмаган қиймат қабул қилса false қийматини қайтаради. Мантиқий кўпайтириш оператори фақатгина ўзининг иккала ифодаси ҳам рост бўлсагина true қиймат қайтаради.

Мантиқий кўпайтириш амали && белги орқали белгиланади.

Мантиқий қўшиш амали. Мантиқий қўшиш амали ҳам иккита ифода орқали ҳисобланади. Агарда улардан бирортаси рост бўлса мантиқий қўшиш амали true қиймат қайтаради. Агарда сизда пул ЁКИ кредит карточкаси бўлса, сиз счётни тўлай оласиз. Бу ҳолда иккита шартнинг бирданига бажарилиши: пулга ҳам ва кредит карточкасига ҳам эга бўлишингиз шарт эмас. Сизга улардан бирини бажарилиши етарли. Бу операторга оид яна бир мисолни қараймиз. Масалан,

```
(x==5)||(y==5)
```

ифодаси ёки x ўзгарувчи қиймати, ёки y ўзгарувчи қиймати, ёки иккала ўзгарувчининг қиймати ҳам 5 га тенг бўлса рост қиймат қайтаради.

Мантиқий инкор амали. Мантиқий инкор оператори текширилаётган ифода ёлғон бўлса true қиймат қайтаради. Агарда текширилаётган ифода рост бўлса инкор оператори false қиймат қайтаради. Масалан, (!(x==5)) ифодасининг қиймати, агарда x ўзгарувчиси 5 га тенг бўлмаса true қиймат қайтаради. Бу ифодани бошқача ҳам ёзиш мумкин:

```
(x!=5)
```

Муносабат амаллари. Бундай амаллар иккита қийматни тенг ёки тенг эмаслигини аниқлаш учун ишлатилади. Таққослаш ифодаси доимо true (рост) ёки false (ёлғон) қиймат қайтаради. Муносабат амаллари арифметик типдаги операндларга қўлланилса қийматлари 1 га тенг агар нисбат бажарилса ва аксинча 0 га тенгдир.

Муносабат амалларининг қўлланилишига оид мисол жадвалда келтирилган.

Номи	Амал	Мисол	Қайтарадиган қиймат
Тенглик	==	100==50 50==50	false true
Тенг эмас	!=	100!=50 50!=50	true false
Ката	>	100>50 50>50	true false
Катта ёки тенг	>=	100>=50 50>=50	true true
Кичик	<	100<50 50<50	true false
Кичик ёки тенг	<=	100<=50 50<=50	true true

Катта >, кичик <, катта ёки тенг >=, кичик ёки тенг <= амалларининг устиворлиги бир хилдир.

Тенг == ва тенг эмас != амалларининг устиворлиги ўзаро тенг ва колган нисбат амалларидан пастдир.

Разрядли амаллар. Разрядли амаллар натижаси бутун сонларни иккилик кўринишларининг хар бир разрядига мос мантикий амалларни қўллашдан хосил бўлади. Масалан 5 коди 101 га тенг ва 6 коди 110 га тенг.

6&5 қиймати 4 га яъни 100 га тенг.
6|5 қиймати 7 га яъни 111 га тенг.
6^5 қиймати 3 га яъни 011 га тенг.
~6 қиймати 4 га яъни 010 га тенг.

Бу мисолларда амаллар устиворлиги ошиб бориши тартибида берилгандир.

Бу амаллардан ташқари M<<N чапга разрядли силжитиш ва M>>N ўнгга разрядли силжитиш амаллари қўлланилади. Силжитиш M бутун соннинг разрядли кўринишига қўлланилади. N нечта позицияга силжитиш кераклигини кўрсатади.

Чапга N позицияга суриш бу операнд қийматини иккининг N чи даражасига кўпайтиришга мос келади. Мисол учун 5<<2=20. Бу операторнинг битли кўриниши: 101<<2=10100.

Агар операнд мусбат бўлса N позицияга унгга суриш чап операндни иккининг N чи даражасига бўлиб каср кисмини ташлаб юборишга мосдир. Мисол учун 5>>2=1. Бу операторнинг битли кўриниши 101>>2=001=1. Агарда операнд қиймати манфий бўлса икки вариант мавжуддир: арифметик силжитишда бўшатилаётган разрядлар ишора разряди қиймати билан тўлдирилади, мантикий силжитишда бўшатилаётган разрядлар нуллар билан тўлдирилади.

Разрядли инкор амали унар қолган амаллар бинар амалларга киради.

Разрядли суриш амалларининг устиворлиги ўзаро тенг, разрядли инкор амалидан паст, колган разрядли амаллардан юқоридир.

Амаллар устиворлиги жадвали

Ранг	Амаллар	Йуналиш
1	() [] -> :: .	Чапдан унгга
2	! ~ + - ++ -- & * (тип) sizeof new delete тип()	Унгдан чапга
3	. * ->*	Чапдан унгга
4	* / % (мультипликатив бинар амаллар)	Чапдан унгга
5	+ - (аддитив бинар амаллар)	Чапдан унгга
6	<< >>	Чапдан унгга
7	< <= >= >	Чапдан унгга
8	= !=	Чапдан унгга
9	&	Чапдан унгга
10	^	Чапдан унгга
11	|	Чапдан унгга
12	&&	Чапдан унгга
13	||	Чапдан унгга
14	?:(шартли амал)	Унгдан чапга
15	= *= /= %= += -= &= ^= |= <<= >>=	Унгдан чапга
16	, (вергуль амали)	Чапдан унгга

1.4. C++ тилида дастур тузилиши

C++ тилида оддий дастур. C++ тилида тузилган дастур объектлар, функциялар, ўзгарувчилар ва бошқа элементлардан ташқил топади.

C++ тилида ёзилган дастур таркибини ўрганиш учун оддийгина SALOM.CPP дастури билан танишамиз. Бу дастур кичик бўлишига қарамасдан бизда қизиқиш уйғотувчи бир нечта элементдан иборатдир.

SALOM.CPP дастури мисолида C++ тилида тузилган дастур қисмларини намойиш қилиш

```
#include <iostream.h>
int main( )
{
 cout << "Salom!\n";
 return 0;
}
```

Натижа:

```
Salom!
```

ТАҲЛИЛ

1 – сатрда iostream файли жорий файлга бириктиляпти. Дастурда биринчи фунта (#) белгиси жойлашган. У препроцессорга сигнал узатади.

2 Компиляторнинг ҳар сафар ишга туширилишида препроцессор ҳам ишга туширилади. У дастурдаги фунта (#) белгиси билан бошланувчи қаторларни ўқийди.

include - препроцессорнинг командаси бўлиб, у қуйидагича таржима қилинади: «Бу командани ортидан файл номи келади. Ушбу номдаги файлни топиш ва файлдаги мазмунни дастурнинг жорий кисмига ёзиш лозим».

Бурчакли қавс ичидаги файлни мос файллар жойлаштирилган барча папкалардан излаш лозимлигини кўрсатади. Агарда компилятор тўғри созланган бўлса бурчакли кавслар iostream файлини сизнинг компиляторингиз учун файлларни ўзида сақловчи папкадан излаши кераклигини кўрсатади. Iostream (input-output stream – киритиш–чиқариш оқими) файлида экранга маълумотларни чиқариш жараёнини таъминлайдиган cout объекти аниқланган. Биринчи қатор бажарилгандан сўнг iostream файли жорий дастурга худди унинг мазмунини қўл билан ёзганимиздек бириктирилади. Препроцессор компилятордан кейин юкланади ва фунт (#) белгии билан бошланувчи барча қаторларни бажаради, дастур кодларини компиляцияга тайёрлайди.

Дастурнинг асосий коди main() функциясини чақириш билан бошланади. C++ тилидаги ҳар бир дастур main() фукциясини ўзида

сақлайди. Функция бу бир ёки бир неча амални бажарувчи дастур блокидир. Одатда функциялар бошқа функциялар орқали чақирилади, лекин `main()` функцияси алоҳида хусусиятга эга бўлиб у дастур ишга туширилиши билан автоматик тарзда чақирилади.

`main()` функциясини бошқа функциялар каби қайтарадиган қиймати типини эълон қилиш лозим. `SALOM.cpp` дастурида `main()` функцияси `int` (`integer` – бутун сўзидан олинган) типли қиймат қайтаради, яъни бу функция ишини тугатгандан сўнг операцион системага бутун сонли қиймат қайтаради. Операцион системага қиймат қайтариш унчалик муҳим эмас, умуман система бу қийматдан фойдаланмайди, лекин C++ тили стандарти `main()` функцияси барча қоидаларга мувофиқ эълон қилинишини талаб қилади.

Айрим компиляторлар `main()` функциясини `void` типидаги қиймат қайтарадиган қилиб эълон қилиш имконини беради. C++ да бундан фойдаланмаслик керак, чунки ҳозирда бундай услуб эскирган. `main()` функциясини `int` типини қайтарадиган қилиб аниқлаш лозим ва бунинг ҳисобига функциянинг охирига `return 0` ифодаси ёзилади.

Барча функциялар очилувчи фигурали қавс (`{`) билан бошланади ва (`}`) ёпилувчи қавс билан тугайди. Фигурали қавсларни ичида жойлашган барча сатрлар функция танаси деб айтилади.

Бизнинг оддий дастуримизда `cout` объекти экранга маълумотни чиқариш учун қўлланилади. `cin` ва `cout` объектлари мос равишда маълумотларни киритиш (масалан, клавиатура орқали) ва уларни чиқариш (экранга чиқариш) учун қўлланилади.

`cout` объекти ҳақида қисқача маълумот. Кейинги мавзуларда сиз `cout` объектини қандай ишлатиш лозимлигини билиб оласиз. Ҳозир эса у ҳақида қисқача маълумот берамиз. Экранга маълумотни чиқариш учун *`cout`* сўзини, ундан сўнг чиқариш операторини (`<<`) киритиш лозим. C++ компилятори (`<<`) белгисини битта оператор деб қарайди. Қуйидаги дастурни таҳлил қиламиз.

`cout` объектини қўлланилишига мисол

```
#include <iostream>
using namespace std;
int main()
{
cout << "Bu son 5 ga teng:" << 5<< "\n";
cout <<"endl operatori ekranda yangi";
cout << " satrga o`tish amalini bajaradi";
cout <<endl;
cout << "Bu katta son:\t"<< 70000 <<endl;
```

```
    cout << "Bu 5 va 8 sonlarining yig`indisi:"<<"\t"<<
8+5 << endl;
    cout << "Bu kasr son:\t"<<(float)5/8<< endl;
    cout << "Bu esa juda katta son:"<<" \t";
    cout << (double) 7000*7000 << endl;return 0;
    };
```

Натижа:

```
   Bu son 5 ga teng: 5
   endl operatori ekranda yangi satrga o`tish amalini
bajaradi
   Bu katta son: 70000
   Bu 5 va 8 sonlarining yig`indisi: 13
   Bu kasr son:  0.625
   Bu esa juda katta son: 4.9e+07
```

Айрим компиляторларда cout объектидан кейин математик операцияларни бажариш учун фигурали қавсларни ишлатиш талаб қилинади.

`endl` оператори `end line` (сатр охири) деган сўздан олинган бўлиб «энд-эл» деб ўқилади.

Изоҳлар. Сиз дастур ёзаётган вақтингизда нима иш қилмоқчи эканлигингиз доимо аниқ бўлади. Лекин бир ойдан сўнг бу дастурга қайтиш лозим бўлса дастурга тегишли деталлар ва уларнинг вазифалари нимадан иборат эканлигини билмаслигингиз мумкин.

Дастурни бутунлай хотирангиздан ўчириб юбормаслик ва бошқаларга ҳам тушунарли бўлиши учун изоҳлардан фойдаланиш лозим. Изоҳлар компилятор томонидан тушириб қолдириладиган дастурнинг алоҳида сатрида ёки бутун бир блокида қўлланилади. Қуйидаги листингни кўриб чиқамиз.

`Salom.cpp` дастури мисолида изоҳларни намойиш қилиш.

```
#include <iostream>
using namespace std;
Int main()
{
cout << "Salom!\n";
/* бу изох токи изохнинг
охирини кўрсатувчи белги, яъни  юлдузча
ва слэш белгиси учрамагунча давом этади */
cout << "Bu kommentariy tugadi\n";
// бу изох сатрни охирида тугайди.
```

```
// Иккита  слэщдан сўнг хеч кандай текст
// булмаслиги мумкин.
return 0;
}
```

Натижа:

```
Salom
Bu kommentariy tugadi
```

Ифода. C++ тилида ифодалар бирор бир ҳисоблаш натижасини қайтарувчи бошқа ифодалар кетма-кетлигини бошқаради ёки ҳеч нима қилмайди (нол ифодалар).

C++ тилида барча ифодалар нуқтали вергул билан якунланади. Ифодага мисол қилиб ўзлаштириш амалини олиш мумкин.

```
x=a+b;
```

Алгебрадан фарқли равишда бу ифода `x a+в` га тенг эканлигини англатмайди. Бу ифодани қуйидагича тушуниш керак:

`a` ва `в` ўзгарувчиларни қийматларини йиғиб натижасини `x` ўзгарувчига берамиз ёки `x` ўзгарувчига `a+в` қийматни ўзлаштирамиз. Бу ифода бирданига иккита амални бажаради, йиғиндини ҳисоблайди ва натижани ўзлаштиради. Ифодадан сўнг нуқтали вергул қўйилади. `(=)` оператори ўзидан чап томондаги операндга ўнг томондаги операндлар устида бажарилган амаллар натижасини ўзлаштиради.

Бўш жой (пробел) белгиси. Бўш жой белгиларига нафақат пробел, балки янги сатрга ўтиш ва табуляция белгилари ҳам киради. Юқорида келтирилган ифодани қуйидагича ҳам ёзиш мумкин:

```
x       = a
+         b    ;
```

Бу вариантда келтирилган ифода кўримсиз ва тушунарсиз бўлса ҳам тўғридир.

Бўш жой белгилари дастурнинг ўқилишлилигини таъминлайди.

Қўшма операторлар. Бир неча операторлар { ва } фигурали қавслар ёрдамида қўшма операторларга ёки блокларга бирлаштирилиши мумкин. Блок ёки қўшма оператор синтаксис жихатдан битта операторга эквивалентдир. Блокнинг қўшма операторлардан фарқи шундаки блокда объектлар таърифлари мавжуд бўлиши мумкин.

Қуйидаги дастур қисми қўшма оператор:

```
{
n++;
summa+=(float)n;
}
```

Бу фрагмент бўлса

блок:

```
{
int n=0;
n++;
summa+=(float)n;
}
```

САВОЛЛАР

1. Нима учун литералли ўзгармасга нисбатан белгили ўзгармасни ишлатиш яхшироқ?
2. const калитли сўзини #define директиваси ўрнига қўллашни афзаллиги нимада?
3. `#include` директиваси қандай вазифани бажаради.
4. `main()` функциясининг ўзига хос хусусияти нимадан иборат?
5. Қандай изоҳ турларини биласиз ? Улар нима билан фарқ қилади?
6. Изоҳлар бир неча қаторда ёзилиши мумкинми?
7. x=5+7 ёзуви ифода бўла оладими? Унинг қиймати нечага тенг?
8. 201/4 ифоданинг қиймати нечага тенг?
9. 201%4 ифода қиймати нечага тенг?
10. x = 3 ва x = =3 ифодаларнинг фарқи нимадан иборат?

МИСОЛЛАР

1. Математик амаллардан фойдаланишни кўрсатувчи дастур тузинг. Бу дастурда қуйидаги операторлар қатнашиши мумкин:

```
cout<<"5+7="<<5+7<<endl;
```

2. Мантиқий амаллардан фойдаланишни кўрсатувчи дастур тузинг. Бу дастурда қуйидаги операторлар қатнашиши мумкин:

```
cout<<"true and alse="<<1&&0<<endl;
```

3. Нисбат амаллардан фойдаланишни кўрсатувчи дастур тузинг.

4. Муносабат амаллардан фойдаланишни кўрсатувчи дастур тузинг.

5. Сон абсолют қийматини шартли амал ёрдамида хисобловчи дастур тузинг.

2 боб. Дастурларнинг таркибий қисмлари

2.1. Ўзгарувчилар

Ўзгарувчилар турлари. Дастур ўзи ишлатадиган маълумотларни сақлаш имкониятига эга бўлиши лозим. Бунинг учун ўзгарувчилар ва ўзгармаслардан фойдаланилади.

C++ тилида ўзгарувчилар маълумотни сақлаш учун қўлланилади. Ўзгарувчининг дастурда фойдаланиш мумкин бўлган қандайдир қийматларни сақлайдиган компьютер хотирасидаги ячейка кўринишда ифодалаш мумкин.

Компютер хотирасини ячейкалардан иборат қатор сифатида қараш мумкин. Барча ячейкалар кетма – кет номерланган. Бу номерлар ячейканинг адреси деб аталади. Ўзгарувчилар бирор – бир қийматни сақлаш учун бир ёки бир нечта ячейкаларни банд қилади.

Ўзгарувчининг номини (масалан, `MyVariable`) хотира ячейкаси адреси ёзилган ёзув деб қараш мумкин.

Масалан `MyVariable` ўзгарувчиси 102 – адресдаги ячейкадан бошлаб сақланади. Ўзининг ўлчовига мувофиқ `MyVariable` ўзгарувчиси хотирадан бир ёки бир неча ячейкани банд қилиши мумкин.

Ўзгарувчиларнинг қуйидаги типлари мавжуддир:
bool – мантиқий;
char – битта символ;
long char – узун символ;
int – бутун сон;
short ёки **short int** – киска бутун сон
long ёки **long int** – узун бутун сон
float хақиқий сон;
long float ёки **double** – иккиланган хақиқий сон
long double – узун иккиланган хақиқий сон

Бутун сонлар ўлчами. Бир хил типдаги ўзгарувчилар учун турли компютерларда хотирадан турли ҳажмдаги жой ажратилиши мумкин. Лекин, битта компютерда бир хил типдаги иккита ўзгарувчи бир хил миқдорда жой эгаллайди.

`char` типли ўзгарувчи бир байт ҳажмни эгаллайди. Кўпгина комьпютерларда `short int` (қисқа бутун) типи икки байт, `long int` типи эса 4 байт жой эгаллайди. Бутун қийматлар ўлчовини компьютер системаси ва ишлатиладиган компилятор аниқлайди. 32 – разрядли компьютерларда бутун ўзгарувчилар 4 байт жой эгаллайди. Қуйидаги дастур сизнинг компютерингиздаги типларнинг ўлчовини аниқлаб беради.

Таянч типлар учун компютер хотирасидан ажратиладиган байтларни аниқлаш.

```
#include <iostream>
using namespace std;
int main()
{
cout << "int tipining o`lchami: \t"
<<sizeof(int)<<" bayt."<<endl;
cout<< "short int  tipining o`lchami:\t"
<< sizeof(short)<< "bayt."<<endl;
cout<<"long int tipining o`lchami:\t"
<< sizeof(long)<<"bayt."<<endl;
cout<< "char tipining o`lchami:\t"
<< sizeof(char)<<"bayt."<<endl;
return 0;
};
```

Натижа:

```
int tipining o`lchami: 4 bayt.
short int  tipining o`lchami:2 bayt.
long int tipining o`lchami: 4 bayt.;
char tipining o`lchami:1 bayt;
```

Ўзгарувчига қиймат бериш. Ўзгарувчиларни дастурнинг ихтиёрий кисмида таърифлаш ёки кайта таърифлаш мумкин.

Мисол учун:

```
int a, b1, ac; ёки
int a;
int b1;
 int ac;
```

Ўзгарувчилар таърифланганда уларнинг қийматлари аниқланмаган бўлади. Лекин ўзгарувчиларни таърифлашда инициализация яъни бошлангич қийматларини кўрсатиш мумкин.

Мисол учун:

```
int i=0;
char c='k';
```

Ўзгарувчиларга қиймат бериш учун ўзлаштириш оператори қўлланилади. Масалан, `Width` ўзгарувчисига 5 қийматни бериш учун қуйидагиларни ёзиш лозим:

```
unsigned short Width;
Width = 5;
```

Бу иккала сатрни `Width` ўзгарувчисини аниқлаш жараёнида биргаликда ёзиш мумкин.

```
unsigned short Wigth = 5;
```

Бир неча ўзгарувчиларни аниқлаш вақтида ҳам уларга қиймат бериш мумкин:

```
long width = 5,  length = 7;
```

Бу мисолда `long` типидаги `width` ўзгарувчиси 5 қийматни, шу типдаги `length` ўзгарувчиси эса 7 қийматни қабул қилди. Қуйидаги дастурда ўзгарувчиларни аниқлашга оид мисолни қараймиз.

Ўзгарувчиларнинг қўлланиши.

```
#include <iostream>
using namespace std;
int main()
{
int Buyi=5, Eni=10,Yuzasi;
cout << "Bo'yi:" << Buyi <<  " \n";
cout << "Eni:" << Eni << endl;
Yuzasi= Buyi*Eni;
cout << "Yuzasi:" << Yuzasi << endl;
return 0;
}
```

Натижа:

```
Bo`yi: 5
Eni: 10
Yuzasi: 50
```

Ишорали ва ишорасиз типлар. Дастурда қўлланиладиган бутун сонли типлар ишорали ва ишорасиз бўлиши мумкин. Баъзан ўзгарувчи учун фақатгина мусбат сонни қўллаш фойдали бўлади. `Unsingned` калитли сўзисиз келтирилган бутун сонли типлар (`short` ва `long`) ишорали ҳисобланади. Ишорали бутун сонлар манфий ва мусбат бўлиши мумкин. Ишорасиз сонлар эса доимо мусбат бўлади. Ишорасиз бутун сонлар устида амаллар `mod` 2^n арифметикасига асослангандир. Бу ерда `n` сони `int` типи хотирада эгалловчи разрядлар сонидир. Агар ишорасиз `k` сони узунлиги `int` сони разрядлар сонидан узун булса, бу сон қиймати `k mod` 2^n га тенг бўлади. Ишорасиз `k` сон учун `-k` амали $2^n - k$ формула асосида хисобланади. Ишорали яъни `signed` типидаги сонларнинг энг катта разряди

сон ишорасини кўрсатиш учун ишлатилса `unsigned` (ишорасиз) типдаги сонларда бу разряд сонни тасвирлаш учун ишлатилади.

Ишорасиз бутун сонларни айиришда, агарда натижа манфий сон бўлса ғайриоддий натижа беради. Буни қуйида кўришимиз мумкин.

Айириш натижасида бутун сонни тўлиб қолишига мисол

```
#include <iostream>
using namespace std;
int main()
{
unsigned int ayirma;
unsigned int kattaSon = 100;
unsigned int kichikSon = 50;
ayirma = kattaSon - kichikSon;
cout << "Ayirma:"<< ayirma<< " ga teng\n";
ayirma = kichikSon - kattaSon;
cout << "Ayirma:"<< ayirma<< " ga teng\n";
return 0;
}
```

Натижа:

```
Ayirma: 50 ga teng
Ayirma: 4294967246 ga teng
```

`typedef` калитли сўзи. `unsigned short int` каби калит сўзларни кўп марталаб дастурда ёзилиши зерикарли ва диққатвозлик талаб қилганлиги учун C++ тилида бундай типларни `typedef` калитли сўзи ёрдамида псевдонимини (тахаллусини) тузиш имконияти берилган. `typedef` сўзи типни аниқлаш маъносини билдиради.

Псевдоним тузишда типнинг номи янги тузиладиган тип номидан фарқли бўлиши лозим. Бунда биринчи `typedef` калитли сўзи, кейин мавжуд тип номи, ундан сўнг эса янги ном ёзилади. Масалан:

```
typedef unsigned short int  ushort
```

Бу сатрдан сўнг `ushort` номли янги тип ҳосил бўлади ва у қаерда `unsigned short int` типидаги ўзгарувчини аниқлаш лозим бўлса, шу жойда ишлатилади.

Мисол учун янги COD типини киритиш:

```
typedef unsigned char COD;
COD simbol;
```

`typedef` оператори орқали типларнинг аниқланиши

```
#include <iostream>
using namespace std;
typedef  unsigned short int ushort;
int main()
{
ushort Buyi = 5;
ushort Eni = 10;
ushort Yuzasi = Buyi* Eni;
cout << "Yuzasi:" << Yuzasi << endl;
}
```

Натижа:

```
Yuzasi: 50
```

Типлар билан ишловчи амаллар. Типларни ўзгартириш амали қуйидаги кўринишга эга:

(тип_номи) операнд;

Бу амал операндлар қийматини кўрсатилган типга келтириш учун ишлатилади. Операнд сифатида костанта, ўзгарувчи ёки қавсларга олинга ифода келиши мумкин. Мисол учун `(long)6` амали константа қийматини ўзгартирмаган холда оператив хотирада эгаллаган байтлар сонини оширади. Бу мисолда константа типи ўзгармаган булса, `(double)6` ёки `(float)6` амали константа ички кўринишини хам ўзгартиради. Катта бутун сонлар хақиқий типга келтирилганда соннинг аниқлиги йуколиши мумкин.

`sizeof` амали операнд сифатида кўрсатилган объектнинг байтларда хотирадаги хажмини хисоблаш учун ишлатилади. Бу амалнинг икки кўриниши мавжуд:

`sizeof` ифода

`sizeof` (тип)

Мисол учун:

```
sizeof 3.14=8
sizeof 3.14f=4
sizeof 3.14L=10
sizeof(char)=1
sizeof(double)=8.
```

Типларни келтириш. Бинар арифметик амаллар бажарилганда типларни келтириш қуйидаги қоидалар асосида амалга оширилади:

`short` ва `char` типлари `int` типига келтирилади;

Агар операндлар бири `long` типига тегишли бўлса иккинчи операнд хам `long` типига келтирилади ва натижа хам `long` типига тегишли бўлади;

Агар операндлар бири `float` типига тегишли бўлса иккинчи операнд хам `float` типига келтирилади ва натижа хам `float` типига тегишли бўлади;

Агар операндлар бири `double` типига тегишли бўлса иккинчи операнд хам `double` типига келтирилади ва натижа хам `double` типига тегишли бўлади;

Агар операндлар бири `long double` типига тегишли бўлса иккинчи операнд хам `long double` типига келтирилади ва натижа хам `long double` типига тегишли бўлади.

2.2. Шартли оператор

if оператори. Одатда дастур сатрма–сатр тартиб билан бажарилади. `If` оператори шартни текшириш (масалан, икки ўзгарувчи тенгми) ва унинг натижасига боғлиқ равишда дастурни бажарилиш тартибини ўзгартириш имконини беради. `If` операторининг оддий шакли қуйидаги кўринишдадир:

```
if (шарт)
   ифода;
```

Қавс ичидаги шарт ихтиёрий ифода бўлиши мумкин. Агарда бу ифода `false` қийматини қайтарса ундан кейинги ифода ёки блок тушириб қолдирилади. Агарда шарт `true` қиймат қайтарса навбатдаги ифода бажарилади. Қуйидаги мисолни қараймиз:

```
If (kattaSon>kichikSon)
kattaSon=kichikSon;
```

Бу ерда `kattaSon` ва `kichikSon` ўзгарувчилари таққосланаяпти. Агарда `kattaSon` ўзгарувчиси қиймати катта бўлса, бу навбатдаги қаторда унга қиймат сифатида `kichikSon` ўзгарувчисининг қиймати ўзлаштирилади.

`if` операторида фигурали қавс ичига олинган ифодалар блокини ҳам ишлатиш мумкин.

```
if (шарт)
{
   3 - ифода
   4 - ифода
   5 - ифода
 }
```

Қуйида ифодалар блокининг қўлланилишига оид мисол келтирилган

```
if(kattaSon>kichikSon)
{  kattaSon=kichikSon
    cout<<"kattaSon:"<<kattaSon << "/n";
    cout<<"kichikSon:"<<kichikSon<< "/n";
```

```
}
```

Бу ҳолда `kattaSon` ўзгарувчисига нафақат `kichikSon` ўзгарувчиси ўзлаштирилаяпти, балки экранга бу ҳақида ахборот ҳам чиқарилаяпти.

Муносабат операторининг қўлланилиши орқали тармоқланишга мисол

```
#include <iostream>
using namespace std;
int main( )
{
int BuxoroGol, PaxtakorGol;
 cout<<"Buxoro komandasi kiritgan to`plar"<< "sonini
yozing:";
 cin >> BuxoroGol;
 cout<<"Paxtakor komandasi kiritgan";
 cout<< "to`plar sonini yozing:";
 cin >> PaxtakorGol;
cout << "\n";

if ( BuxoroGol>PaxtakorGol)
cout << "Yashasin Buxoro!\n";

if (BuxoroGol < PaxtakorGol)
{
 cout << "Yashasin PaxtakorGol \n";
 cout << "Bugun Toshkentda bayram!\n";
};
if (BuxoroGol==PaxtakorGol)
{
  cout << "Durrangmi? Yo-oq? Bo`lishi"<<
  cout<<" mumkin emas \n";
  cout <<"Paxtakorning kiritgan to`plari";
  cout<< "haqida ma`lumotni qaytadan yozing\n";
};
cin >> PaxtakorGol;
if (BuxoroGol>PaxtakorGol)
{
cout<<"Buxoro yutishini oldindan bilgan";
cout<<" edim! Shuning uchun qayta so`radim\n";
cout<< "Yashasin Buxoro!";
}
if (BuxoroGol<PaxtakorGol)
{
cout<<"Paxtakor yutishini oldindan bilgan";
```

```
cout<<" edim! Shuning uchun qayta so`radim\n";
cout<< "Yashasin Paxtakor!";
cout << "Bugun Toshkentda bayram!\n";
}
if (BuxoroGol==Paxta
korGol)
cout<<"Qoyil! Haqiqatan ham during ekan\n";
cout<<"\n Ma`lumotingiz uchun rahmat\n";
return 0;
}
```

Натижа:

```
Buxoro komandasi kiritgan to`plar sonini yozing:3
Paxtakor komandasi kiritgan to`plar sonini yozing:3
Durrangmi? Yo-oq? Bo`lishi mumkin emas

Paxtakorning kiritgan to`plari haqida ma`lumotni
qaytadan yozing: 2

Buxoro yutishini oldindan bilgan edim! Shuning uchun
qayta so`radim
Yashasin Buxoro!
```

else калит сўзи. Дастурларда кўп ҳолларда бирор бир шартнинг бажарилишига (яъни бу шарт `true` қиймат қайтарса) боғлиқ равишда бир блок, унинг бажарилмаслигига асосан эса (яъни бу шарт `false` қиймат қайтарса) бошқа бир блокнинг бажарилиши талаб қилинади. 4.3. – листингда биринчи текшириш (`BuxoroGol>PaxtakorGol`) `true` қиймат қайтарса экранда бир хабар, `false` қийматида эса бошқа бир хабар чиқарилади.

Бундай масалаларни юқорида кўрсатилган усул, яъни қатор шартларни текшириш учун бир нечта `if` операторини қўллаш орқали ҳал қилиш мумкин, лекин бу тушуниш учун бироз мураккаброқдир. Дастурнинг соддалигини таъминлаш учун `else` калитли сўзидан фойдаланиш мумкин.

```
if (шарт)
 Ифода;
else
 Ифода;
```

`else` калитли сўзининг ишлатилиши.

```
#include <iostream>
using namespace std;
int main()
{
```

```
int BirinchiSon, IkkinchiSon;
cout << "Katta sonni kiriting:";
cin >> BirinchiSon;
cout<<"\n Kichik sonni kiriting:";
cin >> IkkinchiSon;
if (BirinchiSon > IkkinchiSon)
cout << "\n Rahmat! \n";
else
cout << "\n Ikkinchisi katta son-ku!";
return 0;
}
```

Натижа:

```
Katta sonni kiriting: 10
Kichik sonni kiriting: 12
Ikkinchisi katta son - ku!
```

Санoқли ўзгармасни қўлланиши

```
#include <iostream>
using namespace std;
int main( )
{
enum Kunlar{Dushanba, Seshanba, Chorshanba,
Payshanba, Juma, Shanba, Yakshanba};
int tanlash;
cout << "Kun nomerini kiriting (0-6):";
cin>>tanlash;
if (tanlash==Yakshanba || tanlash == Shanba)
cout<<"\nBugun siz uchun dam olish kuni!"
<<endl;
else
cout << "\n Bugun siz uchun ish kuni.\n";
return 0;
};
```

Натижа:

```
 Kun nomerini kiriting(0-6): 6
 Bugun siz uchun dam olish kuni!
```

if оператори орқали мураккаб конструкцияларни ҳосил қилиш. if-else конструкциясида ифодалар блокида ихтиёрий операторларни ишлатишда ҳеч қандай чегара йўқ. Шу жумладан, ифодалар блоки ичида яна if-else операторларини ишлатиш мумкин. Бу ҳолда бир нечта if операторидан иборат ичма – ич конструкция ҳосил бўлади.

```
if (1-шарт)
{
  if (2-шарт)
   1-ифода
  else
  {
    if (3-шарт)
     2-ифода
    else
     3-ифода
  }   }
  else

 4-ифода;
```

Ушбу бир нечта if операторидан ташқил топган конструкция қуйидаги тартибда ишлайди: агарда 1-шарт ва 2-шарт рост бўлса 1-ифода бажарилади. Агарда 1-шарт рост ва 2-шарт ёлғон натижа қайтарса, у ҳолда 3-шарт текширилади ва агарда бу шарт рост бўлса 2-ифода, ёлғон бўлса эса 3-ифода бажарилади. Ва энг охири, агарда 1-шарт ёлғон бўлса 4-ифода бажарилади. Бундай мураккаб конструкцияга мисол қуйида келтирилган.

if оператори ички бўлган мураккаб конструкция

```
#include <iostream>
using namespace std;
{
// Иккита сон киритамиз.Уларни BirinchiSon
//ва IkkinchiSon ўзгарувчиларига beramiz
//Агарда KattaSon Қиймати KichikSon
// Қийматидан катта бўлса катта сон
//кичигига колдиксиз бўлинишини текшира-
// миз. Агарда у колдиксиз бўлинса улар
//тенг ёки тенг эмаслигини текширамиз.
int BirinchiSon, IkkinchiSon;
cout<<"Ikkita son kiriting.\n Birinchisi: ";
cin >> BirinchiSon;
cout << "\n Ikkinchisi:";
```

```
cin >> IkkinchiSon;
cout << "\n\n";

if (BirinchiSon>=IkkinchiSon)
if ( (BirinchiSon%IkkinchiSon)==0)
{
if (BirinchiSon==IkkinchiSon)
cout<< "Ular bir – biriga teng!\n";
else
cout<< "Birinchi son ikkinchisiga"<<"karrali!\n";
}
else
cout<<"Ikkinchi son katta!\n";

return 0;
}
```

Натижа:

```
Ikkita son kiriting
Birinchisi:      9
Ikkinchisi:    3
Birinchi son ikkinchisiga karrali!
```

2.3. Функциялар

Функция тушунчаси. Объектларга мўлжалланган дастурлашда асосий эътибор функцияга эмас, балки объектга қаратилган бўлсада дастурларда функция марказий компонентлигича қолди. Функция бу маъносига кўра дастур ости бўлиб, у маълумотларни ўзгартириши ва бирор бир қиймат қайтариши мумкин. C++ да ҳар бир дастур ҳеч бўлмаганда битта `main()` функциясига эга бўлади. `main()` функцияси дастур ишга туширилиши билан операцион система томонидан автоматик чақирилади. Бошқа функциялар эса у томонидан чақирилиши мумкин.

Ҳар бир функция ўзининг номига эгадир. Қачонки, дастурда бу ном учраса бошқарув шу функция танасига ўтади. Бу жараён функцияни чақирилиши (ёки функцияга мурожаат қилиш) деб айтилади. Функция ишини тугатгандан сўнг дастур ўз ишини функция чақирилган қаторнинг кейингисидан бошлаб давом эттиради.

Қачонки, дастур бирор бир хизмат кўрсатувчи амалларни бажарилишига эҳтиёж сезса керакли функцияни чақиради. Бу операция бажарилгандан кейин эса дастур ўз ишини функция чақирилган жойдан бошлаб давом эттиради. Бу ғоя қуйидаги дастурда намойиш этилган.

Функцияни чақирилишига мисол

```
#include <iostream>
```

```
using namespace std;
//NamoyishFunktsiyasi  функцияси экранга
// ахборий маълумот чикаради.
void NamoyishFunktsiyasi()
{
cout<< "NamoyishFunktsiyasi chaqirildi\n";
}
//main() функцияси олдин ахборот чикаради ва
// NamoyishFunktsiyasi функциясини чакиради
// Кейин яна намойиш функциясини чакиради
int main()
{
 cout << "Bu main() funktsiyasi \n";
NamoyishFunktsiyasi();
cout << "main() funktsiyasiga qaytildi\n";
return 0;
}
```

Натижа:

```
Bu main() funktsiyasi
Namoyish funktsiyasi chaqirildi
main() funktsiyasiga qaytildi
```

Функцияларнинг қўлланилиши. Функциялар ё `void` типидаги, ё бошқа бирор бир типдаги қиймат қайтаради. Иккита бутун сонни қўшиб, уларнинг йиғиндисини қайтарадиган функция бутун қиймат қайтарувчи дейилади. Фақатгина қандайдир амалларни бажариб, ҳеч қандай қиймат қайтармайдиган функциянинг қайтарувчи типи `void` деб эълон қилинади.

Функция сарлавҳа ва танадан иборатдир. Функция сарлавҳасида унинг қайтарадиган типи, номи ва параметрлари аниқланади. Параметрлар функцияга қиймат узатиш учун ишлатилади. Масалан, агар функция икки сонни қўшишга мўлжалланган бўлса у ҳолда бу сонларни функцияга параметр қилиб бериш керак. Бундай функциянинг сарлавҳаси қуйидагича бўлади:

```
int Qushish(int a,int b)
```

Параметр – бу функцияга узатиладиган қиймат типини эълон қилишдир. Функция чақирилганда унга узатиладиган қиймат аргумент деб айтилади. Кўпчилик дастурчилар бу иккала тушунчани синоним сифатида қарашади. Баъзилар эса бу терминларни аралаштиришни нопрофессионаллик деб ҳисоблайди. Мавзуларда иккала терминни бир хил маънода келган.

Функция танаси очилувчи фигурали қавс билан бошланади ва у бир неча қатордан иборат бўлиши мумкин. (Функция танасида ҳеч қандай сатр бўлмаслиги ҳам мумкин). Сатрлардан кейин эса ёпилувчи фигурали қавс келади. Функциянинг вазифаси унинг сатрларида берилган дастурий кодлар

билан аниқланади. Функция дастурга `return` оператори орқали қиймат қайтаради. Бу оператор функциядан чиқиш маъносини ҳам англатади. Агарда функцияга чиқиш операторини (`return`) қўймасак функция сатрларини тугаши билан у автоматик `void` типидаги қийматни қайтаради. Функция қайтарадиган тип унинг сарлавҳасида кўрсатилган тип билан бир хил бўлиши лозим.

Қуйидаги дастурда иккита бутун сонли параметрни қабул қилувчи ва бутун сонли қиймат қайтарувчи функция намойиш қилинган.

Оддий функцияни ишлатишга мисол

```
#include <iostream>
using namespace std;
int Qushish(int x,int y)
{
 cout<<x<<" va "<<y<<"sonlarini Qushish()"
 << "funktsiyasi orqali qushdik\n";
 return (x + y) ;
}
 int main()
{
 cout <<"Biz main() funktsiyasidamiz!";
 int a, b, c;
 cout << "Ikkita son kiriting:";
 cin >> a;
 cin >> b;
cout <<"\nQushish()funktsiyasi chaqirildi"<<endl;
 c = Qushish(a,b);
 cout <<"\n main() funktsiyasiga qaytildi."<<endl;
 cout << "c ning qiymati " <<c<<" ga teng" << endl;
 cout << " \n Ishni tugatish ... \n";
 return 0;
}
```

Натижа:

```
  Biz main() funktsiyasidamiz!
  Ikkita son kiriting: 5 3
  Qushish()funktsiyasi chaqirildi
  5  va  3  sonlarini  Qushish()funktsiyasi  orqali
 qushdik
  main() funktsiyasiga qaytildi.
  c ning qiymati 8 ga teng
  Ishni tugatish …
```

ТАҲЛИЛ

`Qushish()` функцияси иккита бутун сонли параметрни қабул қилади ва бутун сонли қиймат қайтаради. Дастурнинг экранга биринчи хабарни

чиқаради. Кейин фойдаланувчига иккита сонни киритиш ҳақида хабар берилади. Фойдаланувчи бўшлик белгиси (пробел) орқали ажратган ҳолда иккита сон киритади. Кейин эса `Enter` тугмасини босади. Бош `main()` функцияси `Qushish()` функциясига фойдаланувчи томонидан киритилган иккита сонни параметр сифатида узатади.

Дастур бошқаруви `Qushish()` функциясига ўтади. `a` ва `b` параметрлар экранга чиқарилади ва қўшилади. Функция натижасини қайтаради ва ўз ишини якунлайди.

Хотиранинг тақсимланиши. Дастур ишлай бошлаши билан операцион система (Dos ёки Microsoft Windows) компиляторнинг талабига мувофиқ хотира соҳасидан жой ажратади. C++ дастурчиси глобал номлар фазоси, эркин тақсимланувчи хотира, регистр, сегментли хотира ва стек тушунчаларини билиши лозим.

Глобал номлар фазосида глобал ўзгарувчилар сақланади. Глобал номлар фазоси ва эркин тақсимланувчи хотира ҳақида кейинги мавзуларда батафсилроқ тўхталади.

Регистр хотиранинг махсус соҳаси бўлиб, унинг асосий вазифаси ички ёрдамчи функцияларни ишлашини ташқил этишдир.

Дастурнинг ўзи дастур операторлари иккилик форматда сақланиши учун ажратилган компьютер хотирасида сақланади.

Стек – бу дастурда функция чақирилганда ундаги маълумотлар учун талаб қилинадиган хотиранинг махсус соҳасидир. Унинг стек деб аталишига сабаб «`охирги келган – биринчи кетади`» принципи асосида ишлашидир. Ҳақиқатан ҳам, функцияларни чақирилиши худди шу принцип асосида амалга оширилади. Агарда бир функция иккинчисини чақирса, иккинчи функция ўз ишини тугатгандан сўнг биринчи функция ўз ишини якунлайди, яъни охирги чақирилган функция биринчи ишини тугатади.

2.4. Функцияларни таърифлаш ва бажариш

Функциянинг таърифи. Функция таърифида функция номи, типи ва формал параметрлар рўйхати кўрсатилади. Формал параметрлар номларидан ташқари типлари хам кўрсатилиши шарт. Формал параметрлар рўйхати функция сигнатураси деб хам аталади.

Функция таърифи умумий кўриниши қуйидагичадир:

Функция типи функция номи(формал_параметрлар_таърифи)

Формал параметрларга таъриф берилганда уларнинга бошлангич қийматлари хам кўрсатилиши мумкин.

Функция кайтарувчи ифода қиймати функция танасида **return** <ифода> ; оператори оркали кўрсатилади.

```
    int  Yuzaint uzunlik,int kenglik)
{ - очилувчи фигурали кавс.
```

```
//  funktsiya tanasi
  return (uzunlik*kenglik);
} - ёпилувчи фигурали кавс.
```

Функциянинг бажарилиши. Функция чақирилганда унда кўрсатилган амаллар очилувчи фигурали қавсдан ({) кейинги биринчи ифодадан бошлаб бажарилади. Функция танасида `if` шартли операторидан фойдаланиб тармоқланишни ҳам амалга ошириш мумкин.

Функция ўз танасида бошқа функцияларни ва ҳатто ўз – ўзини ҳам чақириши мумкин.

Қайтариладиган қийматлар, параметрлар ва аргументлар. Функция бирор бир қиймат қайтариши мумкин. Функцияга мурожаат қилингандан сўнг у қандайдир амалларни бажаради, кейин эса у ўз ишининг натижаси сифатида бирор бир қиймат қайтаради. Бу `кайтариладиган қиймат` деб аталади ва бу қийматнинг типи олдиндан эълон қилиниши лозим. Қуйидаги ёзувда `myFunction` функцияси бутун сонли қиймат қайтаради.

```
int myFunction()
```

Функцияга ҳам ўз навбатида бирор бир қиймат узатиш мумкин. Узатиладиган қийматлар функциянинг параметрлари деб айтилади.

```
int myFunction (int Par, float  ParFloat);
```

Бу функция нафақат бутун сон қайтаради, балки параметр сифатида бутун ва ҳақиқий сонли қийматларни қабул қилади.

Параметрда функция чақирилганда унга узатиладиган қиймат типи аниқланиши лозим. Функцияга узатиладиган ҳақиқий қийматлар аргументлар деб айтилади.

```
int theValueReturned=myFunction(5,6,7);
```

Бу ерда `theValueReturned` номли бутун сонли ўзгарувчига аргумент сифатида 5, 6 ва 7 қийматлар берилган `myFunction` функциясининг қайтарадиган қиймати ўзлаштирилаяпти. Аргумент типлари эълон қилинган параметр типлари билан мос келиши лозим.

```
#include <iostream>
using namespace std;
float  min(float a=0.0, float b)
{
if (a<b) return a;
return b;
}
int main()
{
cout<<min(5,-6);
}
```

Функция таърифида формал параметрлар инициализация килиниши, яъни бошланғич қийматлар кўрсатилиши мумкин.

Функцияга мурожаат қилинганда бирор хақиқий параметр кўрсатилмаса, унинг ўрнига мос формал параметр таърифида кўрсатилган бошлангич қиймат ишлатилади.

Мисол учун:

```
include  <iostream.h>
float  min(float a=0.0, float b)
{
if (a<b) return a;
return b;
}
int main()
{
int y=6,z; z=min(y);
cout<<z;
}
```

Агар функция хеч кандай қиймат кайтармаса унинг типи `void` деб кўрсатилади.

Мисол учун:

```
void print()
{
cout<<"\n Salom! ";
};
```

Бу функцияга `print();` шаклида мурожжат килиш экранга `Salom!` Езилишига олиб келади.

САВОЛЛАР

1. Бутун сонли ва ҳақиқий типларни қандай фарқи бор?
2. Unsigned типининг хоссаларини кўрсатинг.
3. Типларни келтириш қоидалари.
4. unsigned short int ва long int типларининг ўзаро фарқи нимада?
5. Дастур ишига “яхши” ва “ёмон” номланган ўзгарувчи қандай таъсир қилади?
6. Биринчи қайси функция бажарилади?
7. Символли киритиш функциялари.
8. Шартли оператор умумий кўриниши.
9. Қуйидаги ифодалар хисобланшггандан сўнг `i` ва `j` ўзгарувчилариннинг қиймати нечага тенг бўлади:

```
i=5,i+=6, j=i++;
```

10.Агарда MyAge, a ва в ўзгарувчиларининг типлари int бўлса уларнинг қийматлари қуйидаги ифодалар бажарилгандан сўнг нечага тенг бўлади.

```
a.MyAge = 39;
b.a=MyAge++
c.b=++MyAge
```

МАСАЛАЛАР

1. Хамма типлар хажмларини экранга чиқарувчи дастур яратинг ва ишлатинг.

2. Дастурда иккита номланган константа киритиб, йиғиндиси ва кўпайтмасини чиқаринг.

3. Учбурчак периметри ва юзасини хисоблаш функцияларини тузиб дастурда фойдаланинг.

3. Шартли оператор ёрдамида учта сон максимумини хисобловчи функция яратинг ва дастурда фойдалнинг.

4. Шартли оператор ёрдамида учта сон минимумини хисобловчи функция яратинг ва дастурда фойдалнинг.

3. Операторлар

3.1. Шартли цикллар

Цикларни ташқил этиш. Ҳар қандай дастурнинг структураси тармоқланиш ва цикллар тўпламининг комбинациясидан иборат бўлади. Қатор масалаларни ечиш учун кўпинча битта амални бир неча маротаба бажариш талаб қилинади. Амалиётда бу рекурсиялар ва итератив алгоритмлар ёрдамида амалга оширилади. Итератив жараёнлар – бу операциялар кетма-кетлигини зарурий сонда такрорланишидир.

`while` оператори орқали цикларни ташқил этиш. `while` оператори ёрдамида цикларни ташқил этишда операциялар кетма-кетлиги циклнинг давом этиш шарти «тўғри» бўлса ёки 0 дан фарқли бўлсагина унинг навбатдаги операциялари амалга оширилади.

while оператори қуйидаги умумий кўринишга эгадир:

while(ифода) Оператор

Агар дастурда `while (1);` сатр куйилса бу дастур хеч качон тугамайди.

Қуйидаги дастурда `counter` ўзгарувчиси қиймати токи 5 га тенг бўлгунга қадар ошиб борар эди.

`while` оператори ёрдамида циклни ташқил этиш

```
#include <iostream>
using namespace std;
int main()
{
int counter=0; //Бирламчи қийматни ўзлаштириш
while(counter<5)//Цикл шартини текшириш
{
counter ++;
cout << "counter :"<< counter << ". \n" ;
}
cout<<"Tsikl tugadi.Counter:"<<counter<<".\n";
return  0;
}
```

Натижа:

```
 counter : 1
 counter : 2
 counter : 3
 counter : 4
 counter : 5
```

```
Tsikl tugadi.Counter: 5.
```

while оператори орқали мураккаб конструкцияларни тузиш. while оператори шартида мураккаб мантиқий ифодаларни ҳам қўллаш мумкин. Бундай ифодаларни қўллашда && (мантиқий кўпайтириш), || (мантиқий қўшиш) , ҳамда !(мантиқий ИНКОР) каби операциялардан фойдаланилади. Қуйида while оператори конструкциясида мураккаброқ шартларни қўйилишига мисол келтирилган .

while конструкциясидаги мураккаб шартлар.

```
#include <iostream>
using namespace std;
int main()
{
unsigned short kichik;
 unsigned long  katta;
 const unsigned short   MaxKichik=65535;
 cout  << "Kichik sonni kiriting:";
   cin  >> kichik;
   cout << "Katta sonni kiriting:";
   cin  >> katta;
   cout  << "kichik son:" << kichik << "…";
   //Хар бир итерацияда учта шарт текширилади.
   while (kichik<katta && katta>0 &&
         kichik< MaxKichik )
   {
    if(kichik%5000==0) //Хар 5000 сатрдан
     //кейин нуқта чикарилади
    cout<<"." ;
kichik++;
katta-=2 ;
}
cout<<"\n kichik son:"<<kichik<<" katta son:"
<<katta << endl ;
return 0 ;
}
```

Натижа:

```
Kichik sonni kirit : 2
Katta  sonni kirit : 100000
Kichik son : 2 ……
Kichik son :33335    katta son  : 33334
```

ТАҲЛИЛ

Дастур қуйидаги мантиқий ўйинни ифодалайди. Олдин иккита сон – `kichik` ва `katta` киритилади. Ундан сўнг токи улар бир бирига тенг бўлмагунча, яъни «учрашмагунча» кичик сон бирга оширилади, каттаси эса иккига камайтирилади. Ўйинни мақсади қийматлар «учрашадиган» сонни топишдир.

Қийматлар киритилгандан сўнг циклни давом эттиришнинг қуйидаги учта шарти текширилади:

`kichik` ўзгарувчиси қиймати `katta` ўзгарувчиси қийматидан ошмаслиги.

`katta` ўзгарувчиси қиймати манфий ва нолга тенг эмаслиги

`kichik` ўзгарувчиси қиймати `MaxKichik` қийматидан ошиб кетмаслиги

Сўнгра `kichik` сони 5000 га бўлингандаги қолдиқ ҳисобланади. Агарда `kichik` 5000 га қолдиқсиз бўлинса бу операциянинг бажарилиши натижаси 0 га тенг бўлади. Бу ҳолатда ҳисоблаш жараёнини визуал ифодаси сифатида экранга нуқта чиқарилади. Кейин эса `kichik` қиймати биттага оширилади, `katta` қиймати эса 2 тага камайтирилади. Цикл агарда текшириш шарти таркибидаги бирорта шарт бажарилмаса тўхтатилади.

`do`...`while` конструкцияси ёрдамида цикл ташқил этиш. Айрим ҳолларда `while` оператори ёрдамида цикларни ташқил этишда унинг танасидаги амаллар умуман бажарилмаслиги мумкин. Чунки циклни давом этиш шарти ҳар бир итерациядан олдин текширилади. Агарда бошланғич берилган шарт тўғри бўлмаса цикл танасининг бирорта оператори ҳам бажарилмайди.

`while` цикли танасидаги амаллар бажарилмай қолиши

```
#include <iostream>
using namespace std;
int main()
{
int counter;
cout << "How many hellos ?:";
cin >> counter;
while (counter>0 )
{
cout <<  "Hello ! \n";
counter-- ;
}
cout<<"Counter is OutPut ;" << counter ;
return  0;
}
```

Натижа:

```
How many   hellos ? : 2
Hello !
Hello !
counter is OutPut : 0
How many  hellos ?  : 0
counter is OutPut : 0
```

do…while конструкциясининг қўлланилиши. do-while оператори умумий кўриниши қуйидагича:

do
Оператор
while(ифода)

do…while конструкциясида цикл шарти унинг танасидаги операциялар бир марта бажарилгандан сўнг текширилади. Бу цикл операторларини ҳеч бўлмаганда бир марта бажарилишини кафолатлайди. То шарти «ёлғон» бўлмагунча ёки ифода қиймати 0 бўлмагунча цикл кайтарилади.

Қуйида олдинги дастурда келтирилган вариантнинг бир оз ўзгартирилган шакли, яъни while оператори ўрнига do…while конструкцияси қўлланган шакли келтирилган .

do...while конструкциясининг қўлланилиши

```
#include <iostream>
using namespace std;
int main()
{
int counter;
cout<<"How many hellos ?";
cin >>counter;
do
{
cout << "hello \n";
counter--;
}
while(counter>0);
cout << "Counter is :" << counter <<endl;
return 0 ;
}
```

Натижа:

```
how many hellos ? 2
hello
hello
Counter is : 0
How many hellos ? 0
Hello
Counter is: - 1
```

3.2. Параметрик цикл операторлари

`for` оператори. for оператори умумий кўриниши қуйидагича:

for(1-ифода;2- ифода; 3-ифода)
Оператор
Бу оператор қуйидаги операторга мосдир.

1-ифода;
while(2-ифода) {
оператор
3-ифода
}

`while` оператори ёрдамида цикларни ташқил этишда 3 та зарурий амаллар: цикл ўзгарувчисига бошланғич қиймат бериш, ҳар бир итерацияда циклни давом этиш шарти бажарилишини текшириш ва цикл ўзгарувчиси қийматини ўзгартиришни бажаришимиз керак.

`while` операторининг ишлатилишига яна бир мисол.

```
#include <iostream>
using namespace std;
int main()
{
int counter=0;
while (counter <5)
{
counter++ ;
cout << "Looping!";
}
cout << "\n Counter:" << counter << "\n";
return 0;
}
```

Натижа:

```
  Looping!  Looping!  Looping!  Looping!  Looping!
```

```
Counter: 5
```

`for` оператори циклни ишлаши учун зарур бўладиган учта операцияни ўзида бирлаштиради. Бу операцияларни қисқача қуйидагича характерлаш мумкин: бошланғич қийматни ўзлаштириш, шартни текшириш, цикл счётчигини қийматини ошириш. `for` оператори ифодасидаги қавснинг ичида шу уччала операцияни амалга оширувчи ифодалар ёзилади. Қавс ичидаги ифодалар нуқтали вергул орқали ажратилади.

`for` циклининг биринчи ифодаси цикл счётчигига бошланғич қийматни ўзлаштиради. Счётчик – тўғридан–тўғри `for` циклида эълон қилинадиган ва қиймат ўзлаштириладиган бутун сонли ўзгарувчидир. C++ да бу ўринда счётчикка қиймат берадиган ихтиёрий ифода ёзилишига имкон берилган. `for` циклининг иккинчи параметрида циклни давом этиш шарти аниқланади. Бу шарт `while` конструкциясининг шарти бажарадиган вазифани амалга оширади. Учинчи параметрда эса цикл счётчиги қийматини ўзгартирувчи (оширувчи ёки камайтирувчи) ифода ёзилади.

`for` циклининг қўлланилишига мисол.

```
#include <iostream>
using namespace std;
int main()
{
int counter;
for (counter=0 ; counter<5; counter++ )
cout<< "Looping!";
cout<< "\n Counter:" << counter<< ".\n";
return 0;
}
```

Натижа:

```
Looping!  Looping!  Looping!  Looping!  Looping!
Counter: 5
```

`for` оператори учун мураккаб ифодаларни берилиши. `for` цикли дастурлашнинг кучли ва қулай инструментидир. `for` операторида циклни ўзаро боғлиқ бўлмаган параметрлар (бошланғич қиймат ўзлаштириш, бажарилиш шарти ва қадам) ни қўлланилиши цикл ишини бошқаришда жуда яхши имкониятларни очиб беради.

`for` цикли қуйидаги кетма–кетликда ишлайди.

3. Цикл счетчигига бошланғич қиймат ўзлаштирилади.
4. Циклни давом этиш шартидаги ифода қиймати ҳисобланади.

5. Агарда шарт ифодаси `true` қиймат қайтарса олдин цикл танаси бажарилади, кейин эса цикл счётчиги устида берилган амаллар бажарилади.

Ҳар бир итерацияда 2 – ва 3 – қадамлар такрорланади.

Циклда бир нечта счётчикни қўлланилиши. `for` циклининг синтаксиси унда бир нечта ўзгарувчи - счетчикни қўлланилишига, циклни давом этишини мураккаб шартларини текширишга ва цикл счётчиклари устида кетма-кет бир нечта операцияни бажарилишига имкон беради.

Агарда бир нечта счётчикка қиймат ўзлаштирилса ёки улар ўртасида бир нечта операция бажарилса, бу ифодалар вергул билан ажратилган ҳолда кетма – кет ёзилади.

`for` циклида бир нечта счётчикни қўлланилиши

```
#include <iostream>
using namespace std;
int main()
{
for (int i=0, j=0; i<3; i++, j++)
cout<< "i:" <<i<< "j:" <<j<< endl;
return 0;
}
```

Натижа:

```
i: 0      j: 0
i: 1      j: 1
i: 2      j: 2
```

`for` циклида нол параметрларни ишлатилиши. `for` циклининг ихтиёрий параметри тушириб қолдирилиши мумкин. Нол параметрлар `for` циклини бошқа параметрларидан нуқтали вергул (;) билан ажратилади. Агарда `for` циклини 1 – ва 3 – параметрларини тушириб қолдирсак, у худди `while` операторидек қўлланилади.

for циклининг нол параметрлари

```
#include <iostream>
using namespace std;
int main()
{
int counter=0;
for ( ; counter<5 ; )
{
counter++;
cout <<" Looping!";
```

```
}
cout<< "\n Counter:" << counter<< ".\n";
return 0;
}
```

Натижа:

```
 Looping!  Looping!  Looping!  Looping!  Looping!
 Counter: 5
```

ТАҲЛИЛ

Бу циклни бажарилиши `while` циклини бажарилишига ўхшаш тарзда амалга оширилади. Аввал `counter` ўзгарувчисига қиймат ўзлаштирилаяпти. `for` циклида эса параметр сифатида фақатгина циклни давом этиш шартини текшириш ифодаси ишлатилган. Цикл ўзгарувчиси устида операция ҳам тушириб қолдирилган. Бу ҳолда ушбу циклни қуйидагича ифодалаш мумкин:

```
while(counter<5)
```

`for` циклининг танаси бўш бўлган ҳолда қўлланилиши. Циклда `for` оператори орқали унинг танасига ҳеч қандай оператор ёзмасдан туриб ҳам бирор бир амални бемалол бажариш мумкин. Бунда цикл танаси бўш сатрдан иборат бўлади.

For циклининг танаси бўш бўлган ҳолда қўлланилиши.

```
#include <iostream>
using namespace std;
int main()
{
for (int i=0; i<5; cout<< "i" <<i++<<endl);
return 0;
}
```

Натижа:

```
i:  0
i:  1
i:  2
i:  3
i:  4
```

Ички цикллар. Бошқа циклнинг ичида ташқил этилган цикл ички цикл деб айтилади. Бу ҳолда ички цикл ташқи циклни ҳар бир итерациясида тўлиқ бажарилади. Қуйида матрица элементларини ички цикл орқали тўлдирилиши намойиш қилинган.

Ички цикллар.

```
#include <iostream>
using namespace std;
int main()
{
int rows, columns;
char theChar;
cout << "How many rovs?";
cin >> rows;
cout << "How many columns?";
cin >> columns;
cout << "What character?";
cin>>theChar;
for ( int i=0;  i<rows;  i++)
{
for (int j=0; j<columns; j++)
cout << theChar;
cout<< "\n";
}
return 0;
}
```

Натижа:

```
How many rows? 4
How many columns? 12
What character? x
x x x x x x x x x x x x
x x x x x x x x x x x x
x x x x x x x x x x x x
x x x x x x x x x x x x
```

for цикли счётчигининг кўриниш соҳаси. ANSI нинг янги стандарти бўйича циклда эълон қилинган ўзгарувчининг кўриниш соҳаси фақат цикл ичидангина иборат. Лекин кўпгина компиляторлар эски стандартларни ҳам қўллаб – кувватлайдилар. Қуйида келтирилган дастур кодини киритиб ўзингизнинг компиляторингиз янги стандартга мос келиш- келмаслигини текширишингиз мумкин.

```
#include <iostream>
using namespace std;
int main()
 {
for( int i = 0; i<5; i++ )
{
cout << " i: " << i <<  endl;
}
i=7; // i кўриниш сохаси чегарасидан ташқарида
return 0;
}
```

Агарда бу дастур хатоликсиз компиляция қилинса демак у ANSI нинг янги стандартини қўллаб - қувватламайди. Янги стандартга мувофиқ компиляторлар i=7 ифода учун хатолик хақида хабар бериши керак. Дастур кодига озгина ўзгартириш киритилганда сўнг дастур барча компиляторлар учун хатоликсиз ишлайди.

```
#include <iostream>
using namespace std;
int main ()
int i;
for ( int i = 0; i<5; i++ )
{
cout << " i: " << i <<  endl ;
}
i=7; //Энди i барча компиляторлар томонидан
//хатоликсиз кабул килиниши мумкин.
return 0;
}
```

3.3. Ўтиш операторлари

Ўтиш break ва continue операторлари. Кўпинча циклнинг навбатдаги итерациясига цикл танасидаги бошқа операторлар (навбатдаги операторлар) бажарилмасдан туриб ўтиш зарурияти туғилади. Бундай ҳолатларда continue оператори қўлланилади. Бундан ташқари, циклни бажарилиши шарти қаноатлантирилганда ҳам, қатор ҳолларда ундан чиқиб кетиш зарурияти пайдо бўлади. Бу ҳолда эса break оператори ишлатилади. Бундай операторларни қўлланилишига қуйидаги мисолда келтирилган. Бу мисол бизга олдинги листингдан таниш бўлган ўйиннинг бироз мураккаблаштирилган вариантидир. Бу ерда кичик ва катта қийматлардан ташқари қадам ва мақсадли катталикни киритиш талаб этилади. Агарда кичик сон қиймати қадам ўзгарувчисига (qadam) каррали бўлмаса katta

қиймати иккига камайтирилади. Қачонки, kichik ўзгарувчиси қиймати katta ўзгарувчиси қийматидан катта бўлса ўйин тугатилади. Агарда katta ўзгарувчиси қиймати мақсадли катталик (maqsad) нинг қиймати билан устма – уст тушса ўйин бекор қилинганлиги ҳақида хабар экранга чиқарилади.

break ва continue операторларининг қўлланилиши .

```
#include <iostream>
using namespace std;
int main()
{
unsigned short  kichik ;
 unsigned long   katta;
 unsigned long  qadam;
 unsigned long  maqsad ;
 const  unsigned  short   MaxKichik = 65535;
 cout<< "Kichik nomerni kiriting:";
 cin >>kichik ;
 cout<< "Katta nomerni kiriting :";
 cin >>katta ;
 cout<<"Qadam qiymatini kiriting:";
 cin >>qadam;
 cout<<"Maqsadli kattalik qiymatini kiriting:";
 cin >> maqsad;
 cout <<"\n";
while(kichik<katta && katta>0 &&
kichik<MaxKichik)
  {
    kichik++ ;
    if(kichik%qadam==0)
     {
      cout <<"qadam:" << kichik <<endl ;
      continue ;
     }

if(katta==maqsad) //максадли нуқтага
// тенглигини текшириш
{
cout <<"Maqsadga erishildi!";
break;
}
katta -= 2;
}
cout<<"\n Kichik son:"<< kichik;
cout<<" katta son:"<< katta << endl ;
```

```
return 0;
}
```

Натижа:

```
Kichik sonni kiriting: 2
Katta sonni kiriting: 20
Qadam qiymatini kiriting: 4
Maqsadli kattalik qiymatini kiriting: 6
Qadam :4
Qadam: 8

Kichik son : 10   Katta son:8
```

while(true) конструкциясини қўлланилиши. Циклнинг навбатдаги итерациясига ўтишда шарт сифатида C++ тилида синтаксиси бўйича тўғри бўлган ихтиёрий ифода қатнашиши мумкин. Бунда шарт «тўғри» бўлса цикл бажарилаверади. Чексиз цикларни ташқил этиш учун шарт сифатид

а `true` мантиқий ўзгармас қўлланилади.

while операторини қўллашга оид яна бир мисол.

```
#include <iostream>
using namespace std;
int main()
{
int counter = 0;
while(true)
{
counter++;
if(counter>10)
break;
}
cout<<"counter:"<<counter << "\n ";
return 0 ;
}
```

Натижа:

```
Counter:  11
```

for Операторини параметрсиз қўлланиши. Қаралган мисолимиз яна бир марта C++ тилида бир хил масалани бир неча усул билан ҳал қилиш имконияти борлигини кўрсатади. Бундан ташқари, `for` циклининг 3 та параметрини ҳам тушириб қолдириш ва циклни `break` ва `continue` операторларини қўллаш орқали бошқариш мумкин. `for` конструкциясини параметрларсиз қўлланилишига мисол қуйида кўрсатилган.

for операторларини параметрларсиз қўлланилиши

```
#include <iostream>
using namespace std;
int main()
{
int counter=0;
int max;
cout<< "How many hellos?";
cin>> max;
for( ; ; )
{
if (counter <max)
{
cout << "Hello! \n";
counter++;
}
else
break;
}
return 0;
}
```

Натижа:

```
How many hellos? 3
Hello!
Hello!
Hello!
```

3.4. Танлаш оператори

switch операторы. Олдинги мавзуларда биз `if` ва `if/else` операторлари билан танишган эдик. Лекин айрим масалаларни ечишда `if` оператори ичида кўп сондаги `if` операторларини қўллашга тўғри келади. Бу эса дастурни ёзишни ҳам, уни тушинишни ҳам мураккаблаштириб юборади. Бундай муаммони ечиш учун `C++` тилида `switch` оператори қўлланилади. Бу операторнинг `if` операторидан асосий фарқи шуки, унда бир йўла бир нечта шарт текширилади.

Switch операторининг қўлланилиши. `switch` операторининг қўлланилиш синтаксиси қуйидагича:

```
switch(ифода)
{
case 1-нчи Қиймат: ифода;
case 2-нчи Қиймат: ифода;
…
```

```
case n-нчи қиймат: ифода;
default : ифода;
}
```

switch оператори орқали дастурнинг тармоқланиши бир неча мумкин бўлган қийматларни қайтарувчи ифоданинг натижаси асосида ташқил этилади. switch операторидаги қавс ичида берилган ифоданинг қайтарган қиймати case операторидан кейинда кўрсатилган қиймат билан солиштирилади. Ифоданинг қиймати билан case операторидан кейинги қиймат мос келса танланган case операторидан кейинги барча сатрлар бажарилади. Бунда амалларни бажарилиши break операторигача давом этади.

Агарда case операторлари қийматидан бирортаси ҳам қайтарилган қийматга мос келмаса default операторидан кейинги дастур сатрлари бажарилади. Агарда бу оператор мавжуд бўлмаса бошқарув switch блоки танасидан чиқади ва кейинги дастур сатрларига берилади.

switch операторидан кейинги қавс ичида тилнинг конструкцияси нуқтаи–назаридан тўғри бўлган ихтиёрий ифодани ишлатиш мумкин. Оператор идентификатори ўрнида ҳам ихтиёрий оператор ёки ифода, ҳамда оператор ва ифодаларнинг кетма-кетлигини ишлатиш мумкин. Лекин бу ерда мантиқий операциялар ёки таққослаш ифодаларини ишлатиш мумкин эмас.

1-мисол.

```
switch(choice)
{
case 0:
          cout<< "zero!"<< endl;
          break;
case 1:
          cout<< "one!"<< endl;
          break;
case 2:
          cout<< "two!" <<endl;
          break;
default:
          cout<< "default!"<<endl;
}
```

2-мисол

```
switch (choice)
{
case 0:
case 1:
case 2:
        cout< "Less than 3! "<< endl;
```

```
            break;
    case 3:
            cout<< "Equals 3! " << endl;
            break;
    default:
            cout<< "Greater than3 ! " << endl;}
```

Оператор ёки ифодалардан кейин `break` оператори қўлланилмаса жорий `case` операторидан кейинги `case` блокидаги барча ифодалар бажарилади. Кўп ҳолларда бундай ситуацияда хатолик рўй беради. Шунинг учун, `break` операторини тушириб қолдирсангиз бу амални тавсифловчи мос изоҳни(коментарийни) ёзишни унутманг.

`switch` операторининг қўлланилиши

```
#include <iostream>
using namespace std;
int main()
{
unsigned short int  number;
cout<< "Enter   a  number  between 1 and 5:";
cin>>number;
switch (number)
{
case 0:   cout << "Small";
            break;
case 5:   cout<< "Good job! \n";
case 4:   cout << "Nice Pick! \n";
case 3:   cout<< "Excellent! \n";
case 2:   cout << "Master full! \n";
case 1:   cout << "Incredible! \n";
            break;
default:  cout << "Big! \n";
            break;
}
cout<< "\n\n";return 0;
}
```

Натижа:

```
Enter a number between 1 and 5:  3
Excellent!
Masterful!
Ineredible!
Enter a number between 1 and 5: 8
Big!
```

ТАҲЛИЛ

Дастур олдин сон киритишни сўрайди. Кейин эса киритилган сон `switch` оператори орқали текширилади. Агарда 0 киритилган бўлса унга мувофиқ равишда экранга `kichik son` хабари чиқарилади. Ва ундан кейин ёзилган `break` оператори `switch` конструкциясини бажарилишини якунлайди. Агарда 5 сони киритилса мувофиқ хабар чиқарилади. Ундан кейинги токи `break` сўзигача барча сатрлар кетма – кет бажарилади.

Агарда дастурга 5 дан катта сон киритилса `default` бажарилади, яъни экранга `Big` хабари чиқарилади.

`switch` оператори ёрдамида меню командаларини бажариш. `for(;;)` оператори орқали чексиз цикларни ташқил қилинишини олдинги мавзуларда кўриб чиққан эдик. Агарда бундай цикллар бажарилишини `break` оператори орқали тўхтатмасак у чексиз марта ишлайди. Бу турдаги цикллар меню командалари билан ишлашда қулайлик туғдиради. Бунда фойдаланувчи таклиф этилган командалардан бирини танлайди, кейин эса аниқланган амал бажарилади ва яна менюга қайтилади.

Чексиз цикларда ҳеч қандай шарт мавжуд эмас. Шунинг учун бундай циклдан фақатгина `break` оператори орқали чиқилади.

Чексиз циклга мисол

```
#include <iostream>
using namespace std;
void DoTaskOne()
{
cout<<"Salom"<<endl;
};
void DoTaskMany(int n)
{
for(int i=0;i<n;i++) cout<<"Salom"<<endl;
};
int main ( )
{
bool exit=false;
 for (; ;)
 {
 int choice;
 cin>>choice;
 switch (choice)
 {
case (0): exit=true;
          break;
case (1): DoTaskOne ( );
           break;
case (2): DoTaskMany (2);
```

```
            break;
case (3): DoTaskMany (3);
}
if (exit) break;
}
return 0;
}
```

3.5. Шартсиз ўтиш оператори

goto оператори тарихи. Дастурлашни илк даврларида кичикроқ ҳажмдаги ва етарлича содда дастурлар ишлатилар эди. Бундай дастурларда цикллар нишонлардан, операторлар ва командалар кетма – кетлигидан ҳамда ўтиш операторидан иборат эди .

C++ тилида нишон деб орқасидан икки нуқта (:) ёзиладиган идентификаторга айтилади. Нишон доимо бошқарув ўтиши лозим бўлган оператордан олдин ўрнатилади. Керакли нишонга ўтиш учун goto оператори қўлланилади.

Бунда калит сўздан кейин нишон номи ёзилади. Ўтиш операторининг кўриниши:

goto <идентификатор>. Бу оператор идентификатор билан белгиланган операторга ўтиш кераклигини

кўрсатади.

Мисол учун `goto A1;…; A1:y=5;`

goto оператори ёрдамида цикл ташқил етиш .

```
#include <iostream>
using namespace std;
 int main()
{
int counter=0; // счётчикни инициализация килиш
loop:
counter ++;   //  циклни бошланиши
cout<<"counter: " <<counter<< "\n";
if(counter <5) //Қийматни текшириш
goto loop;  //цикл бошига кайтиш
cout<<"Tsikl tugadi.counter:"<<counter<<endl;
return 0;
}
```

Натижа:

```
 counter : 1
 counter : 2
```

```
counter : 3
counter : 4
counter : 5
Tsikl tugadi.Counter: 5.
```

Нима учун goto операторини ишлатмаслик керак. goto оператори орқали дастурнинг ихтиёрий нуқтасига бориш мумкин. Лекин goto операторининг тартибсиз қўлланилиши бу дастурни умуман тушунарсиз бўлишига олиб келади. Шунинг учун охирги 20 йилликда бутун жаҳон бўйича дастурлашни ўрганувчиларга қўйидаги фикр таъкидланиб келинмокда “Ҳеч қачон goto операторини ишлатманг”.

goto операторининг ўрнини бир мунча мукаммалроқ структурага эга бўлган конструкциялар эгаллади. Булар for, while ва do…while операторлари бўлиб, улар goto операторига нисбатан кўпрок имкониятларга эгадир. Лекин дастурлашда ҳар қандай инструмент тўғри қўлланилгандагина фойдали бўлиши ҳисобга олиниб ANSI комитети C++ тилида goto операторини қолдиришга қарор қилди. Албатта, бу билан қуйидаги ҳазил фикр ҳам пайдо бўлди: “Болалар! Бу оператордан уй шароитида фойдаланиш зарарсиздир’’.

САВОЛЛАР

1. While операторининг умумий кўриниши.
2. While операторида мураккаб шарт қандай берилади?
3. Do while оператори While операторидан қандай фарқ қилади?
4. For операторининг умумий кўриниши.
5. Ички цикллар қандай ташқил этилади?
6. Switch операторининг умумий кўриниши.
7. Сирпаниш деб нимага айтилади?
8. Continue операторидан нима учун фойдаланилади?
9. Break операторидан нима учун фойдаланилади?
10. Goto оператори бошқаришни қаерга узатади?

МИСОЛЛАР

1. Карра жадвалини чиқарувчи дастур тузинг.
2. Берилган eps аниқликда умумий хади 1/n! бўлган кетма кетлик йиғиндисини хисобловчи дастур тузинг.
3. Умумий хади x/n! бўлган кетма кетлик n та хади йиғиндисини хисобловчи дастур тузинг.
4. Киритилган n та сон қатъий ўсувчи эканлигини текширувчи дастур яратинг.
5. Киритилган n символдан нечтаси унли харф эканлигини switch оператори ёрдамида хисобловчи дастур тузинг.

4 боб. Функциялар хоссалари

4.1. Функция прототипи

Функцияни эълон қилиш ва аниқланиши. Дастурда функцияни қўллаш учун, олдин уни эълон қилиш, кейин эса аниқлаш лозим. Функцияни эълон қилишда компиляторга унинг номи, қайтарадиган қийматлари ва параметрлари ҳақида хабар берилади. Функцияни аниқланишидан компилятор унинг қандай ишлаши ҳақида маълумот олади. Дастурдаги бирор функцияни олдиндан эълон қилмасдан туриб чақириш мумкин эмас. Функцияни эълон қилиниши унинг прототипини (тимсолини) ҳосил қилиш деб аталади.

Функцияни эълон қилиш. Функцияни эълон қилишнинг уч хил усули мавжуд:

- Функция прототипи файлга ёзилади, кейин эса у `#include` ифодаси қўлланилиб керакли дастурга қўшиб қўйилади.
- Функция ишлатиладиган файлга унинг прототиплари ёзилади.
- Функция уни чақирувчи ихтиёрий функциядан олдин ёзилади ва бу ҳолда функция эълон қилиниши билан бир вақтда аниқланади.

Функцияни прототипини тузмасдан туриб ҳам уни ишлатишдан олдин эълон килиш мумкин. Лекин, дастурлашнинг бундай услуби қуйидаги учта сабабга кўра яхши ҳисобланмайди.

Биринчидан, функцияни файлда кўрсатилган тартибда ёзиш, уни дастур ишлатилишида ўзгартириш жараёнини мураккаблаштиради.

Иккинчидан, қуйидаги кўп учрайдиган ҳолатни амалга ошириш имконияти мавжуд эмас.

`A()` функция `B()` функцияни чақирсин. Худди шунингдек, дастурнинг бирор бир қисмида `B()` функция `A()` функцияни чақирсин. У ҳолда биз `A()` функцияни `B()` функция аниқланмасдан туриб ишлата олмаймиз.

Бу ҳолда ҳеч бўлмаганда битта функция олдиндан эълон қилиниши лозим.

Учинчидан, функциянинг прототиплари дастурни текшириш жараёнида жуда яхши ишлатилади. Агарда функция прототипи аниқланган бўлса унга мувофиқ функция аниқланган параметрини қабул қилади ёки аниқланган бирор бир қиймат қайтаради. Дастурда эълон қилинган прототипга мувофиқ бўлмаган функцияни ишлатишга уринсак компилятор бу хатоликни компиляция жараёнини ўзидаёқ аниқлайди ва дастур ишлашида турли нохуш хатоликларни рўй беришининг олдини олади.

Прототип таърифи. Кўпгина ички қурилган функцияларнинг прототиплари дастурга `#include` калит сўзи ёрдамида қўшиладиган файл-сарлавҳасида ёзилади. Фойдаланувчи томонидан тузиладиган функциялар

учун эса уларнинг мос прототипларини дастурга қўшиш дастурчи томонидан бажарилиши лозим.

Функциянинг прототипи нуқтали вергул орқали тугайдиган функцияни қайтарадиган қиймати ва сигнатурасидан иборатдир. Функцияни сигнатураси деб унинг номи ва параметрлар рўйхати тушинилади.

Формал параметрлар рўйхати барча параметрлар ва уларнинг типларини ифодалайди.

Функциянинг прототипи ҳамда аниқланишидаги унинг қайтарадиган қиймати типи ва сигнатураси мос бўлиши лозим. Агарда бундай мутаносиблик бўлмаса компилятор хатолик ҳақида хабар беради. Функция прототипида параметр номларисиз типларни кўрсатилиши етарлидир. Масалан, қуйида келтирилган мисол тўғридир:

```
long Area(int, int)
```

Бу прототип иккита бутун сонли параметрни қабул қилиб, `long` типидаги қиймат қайтарадиган `Area()` номли функцияни эълон қилади. Прототипнинг бундай ёзилиши унчалик яхши вариант эмас. Прототипга параметрларнинг номларини қўшилиши уни тушунарлироқ бўлишини таъминлайди.

Ҳар бир функциянинг қайтарадиган қиймати типи аниқланган бўлади. Агарда у очиқ аниқланмаган бўлса автоматик равишда `int` типини қабул қилади.

Агар программада функция таърифи мурожаатдан кейин берилса, ёки функция бошқа файлда жойлашган бўлса, мурожжатдан олдин шу функциянинг прототипи жойлашган бўлиши керак.

Функцияни эълон қилиниши, аниқланиши ва ишлатилиши

```
#include <iostream>
using namespace std;
//функция прототипи
int Yuza(int uzunlik, int kenglik);
int main()
{
int YerUzunligi;
int YerKengligi;
int YerMaydoni;
cout<< "\n Yerning uzunligi necha metr? ";
cin >> YerUzunligi;
cout<< "\n Yerning kengligi necha metr? ";
cin >> YerKengligi;
YerMaydoni=Yuza(YerUzunligi,YerKengligi);
cout << "\n Yer maydoni yuzasi ";
cout<<YerMaydoni;
cout << " kvadrat metr \n";
return 0;
}
```

```
int Yuza(int YerUzunligi,int YerKengligi)
{
return YerUzunligi* YerKengligi;
}
```

Натижа:

```
Yerning uzunligi necha metr? 200
Yerning kengligi necha metr? 100
Yer maydoni yuzasi 20000 kvadrat metr
```

4.2. Функциялар ва ўзгарувчилар

Локал ўзгарувчилар. Функцияга қийматлар узатиш билан бирга унинг танасида ўзгарувчиларни эълон қилиш ҳам мумкин. Бу локал ўзгарувчилар орқали амалга оширилади. Қачонки дастурни бажарилиши функциядан асосий қисмга қайтса, бу функциядаги локал ўзгарувчилар хотирадан ўчирилади.

Локал ўзгарувчилар худди бошқа ўзгарувчилар каби аниқланади. Функцияга бериладиган параметрларни ҳам локал ўзгарувчилар деб аташ мумкин ва уларни функция танасида аниқланган ўзгарувчилар каби ишлатиш мумкин.

Функция локал ўзгарувчилари ва параметрларининг қўлланилиши

```
#include <iostream>
using namespace std;
float Almashtirish(float);
int main()
{
  float TempFer;
float TempCel;
cout <<" Feringait bo`yicha temperaturani";
cout<< "kiriting:";
cin >> TempFer;
TempCel = Almashtirish(TempFer);
cout << "\n Bu temperatura selziy shkalasi";
cout<< "bo`yicha: ";
cout << TempCel << endl;
return 0 ;
}
float Almashtirish(float TempFer)
{
float TempCel;
TempCel=((TempFer-32)*5)/9;
return TempCel;
};
```

Натижа:

```
Feringait bo`yicha temperaturani kiriting:  50
Bu temperatura selziy shkalasi bo`yicha:    10
```

Глобал ўзгарувчилар. `main()` функциясида аниқланган ўзгарувчилар дастурдаги барча функциялар учун мурожаат қилишга имконли ва кўриниш соҳасига эга ҳисобланади. Бундай ўзгарувчилар дастурдаги функциялар учун глобал ўзгарувчилар дейилади.

Глобал ўзгарувчи номи билан функция ичида номлари устма-уст тушадиган локал ўзгарувчилар фақатгина жорий функциянинг ичидагина глобал ўзгарувчининг қийматини ўзгартиради. Лекин глобал ўзгарувчи функция ўз ишини тугатгач у чақирилишидан олдинги қийматини сақлаб қолади, яъни функция танасида эълон қилинган локал ўзгарувчи функциянинг ичида глобал ўзгарувчини яширади холос. Бунда локал ўзгарувчи алоҳида ҳосил қилинади ва функция ишлаш вақтида глобал ва локал ўзгарувчиларнинг номлари бир хил бўлса фақатгина локал ўзгарувчи устида амаллар бажарилади. Глобал ўзгарувчи эса функциянинг бажарилиши давомида олдинги қийматини сақлаб туради.

Глобал ва локал ўзгарувчиларнинг қўлланиши

```
#include <iostream>
using namespace std;
void MeningFunktsiyam() ; // прототип
int x = 5, y = 7; // глобал ўзгарувчилар
int main()
{
cout << "main()dagi x ning qiymati:"<<x<<"\n";
cout<<"main()dagi y ning qiymati y:"<<y<<"\n";
MeningFunktsiyam();
cout << "MeningFunktsiyam()funktsiyasi"<<"ishini
tugatdi! \n";
cout<<"main()dagi x ning qiymati:"<<x<<"\n";
cout<<"main()dagi y ning qiymati:"<<y<<"\n";
 return 0;
 }
void MeningFunktsiyam()
{
int y = 10;
cout<<"MeningFunktsiyam()dagi"<<"x:" << x <<"\n";
cout<<"MeningFunktsiyam()dagi"<<"y:"<<y<<"\n";
}
```

Натижа:

```
main()dagi x ning qiymati: 5
main()dagi y ning qiymati: 7

MeningFunktsiyam()dagi x: 5
MeningFunktsiyam()dagi y: 10

MeningFunktsiyam()funktsiyasiishini tugatdi!

main()dagi x ning qiymati: 5
main()dagi y ning qiymati: 7
```

Локал ўзгарувчилар ҳақида батафсилроқ маълумот. Функцияни ичида аниқланган ўзгарувчилар локал кўриниш соҳасига эга дейилади. Юқорида айтиб ўтилганидек, бу ўзгарувчиларни фақатгина функциянинг ичидагина қўллаш мумкинлигини англатади. C++да ўзгарувчиларни нафақат дастурнинг бошида балки ихтиёрий жойда аниқлаш мумкин. Агарда ўзгарувчи функция танасидаги бирор бир блок ичида аниқланган бўлса, бу ўзгарувчи фақатгина шу блок ичидагина таъсирга эга бўлиб бутун функциянинг ичида кўриниш соҳасига эга бўлмайди.

Локал ўзгарувчини кўриниш соҳаси

```
#include <iostream>
using namespace std;
void MeningFunktsiyam();
int main()
{
 int  x=5;
cout<<"\n main()dagi x ning qiymati:"<<x;
MeningFunktsiyam();
cout<<"\n main()dagi x ning qiymati:"<< x;
 return 0;
}
void MeningFunktsiyam()

{
int x = 8;
cout<<"\n MeningFunktsiyam()dagi";
cout<<"local x ning qiymati: "<<x<<endl;
{
cout <<"\n MeningFunktsiyam() ";
cout<<"funktsiyasi blokidagi x ning qiymati";
cout<< " x:"<<x;
int x = 9;
cout<<"\n Blok ichida aniqlangan";
cout<< "x ning qiymati:"<<x;
```

```
}
cout<<"\n MeningFunktsiyam()dagi";
cout<< "blockdan tashqarisida x ning qiymati:";
cout<<x<<endl;
}
```

Натижа:

```
main()dagi x ning qiymati: 5
MeningFunktsiyam()dagi local x ning qiymati: 8
MeningFunktsiyam() funktsiyasi blokidagi x ning
qiymati: 8
Blok ichida aniqlangan x ning qiymati: 9
MeningFunktsiyam()dagi blockdan tashqarisida x ning
qiymati: 8
main()dagi x ning qiymati: 5
```

4.3. Функциядан фойдаланиш хусуиятлари

Функцияда ишлатиладиган операторлар. Функцияда ишлатиладиган операторлар сони ва турига ҳеч қандай чегара йўқ. Функция ичида исталганча бошқа функцияларни чақириш мумкин бўлсада, лекин функцияларни аниқлаш мумкин эмас.

C++ тилида функциянинг ҳажмига ҳеч қандай талаб қўйилмаса ҳам уни кичикроқ тарзда тузган маъқулдир. Ҳар бир функция тушуниш осон бўладиган битта масалани бажариши лозим. Агарда функция ҳажми катталашаётганлигини сезсангиз, бу функцияни бир нечта кичикроқ функцияларга ажратиш ҳақида ўйлашингиз керак.

Функция аргументлари. Функциянинг аргументлари турли типда бўлиши мумкин. Шунингдек, аргумент сифатида C++ тилидаги бирор бир қиймат қайтарадиган ўзгармаслар, математик ва мантиқий ифодалар ва бошқа ихтиёрий функцияларни бериш мумкин.

Мисол сифатида бизда бирор бир қиймат қайтарувчи `double()`, `triple()`, `square()` ва `cube()` функциялари берилган бўлсин. Биз қуйидаги ифодани ёзишимиз мумкин:

```
Answer=(double(triple(square(my Value))))
```

Бу ифодада `myValue` ўзгарувчисини қабул қилинади ва у `cube()` функциясига аргумент сифатида узатилади.

`cube()` функцияси қайтарган қиймат эса `square()` функциясига аргумент сифатида узатилади. `square()` функцияси қиймат қайтаргандан кейин, бунинг қиймати ўз навбатида `triple()` фукнциясига аргумент сифатида берилади. `triple()` функциясининг қиймати эса `double()` функциясига аргумент қилиб берилади ва у `Answer` ўзгарувчисига ўзлаштирилади.

Параметрлар бу локал ўзгарувчилардир. Функцияга узатилган ҳар бир аргумент, бу функцияга нисбатан локал муносабатда бўлади. Функцияни бажарилиши жараёнида аргументлар устида бажарилган ўзгартиришлар функцияга қиймат сифатида берилган ўзгарувчиларга таъсир қилмайди.

Функцияга параметрларни қиймат сифатида узатиш

```
#include <iostream>
using namespace std;
void Almashtirish(int x, int y);
int main()
{
int x = 5,  y =10;
cout << "Main(). Almashtirishdan oldin,x:";
cout<<x <<"  y:"  << y <<"\n";
Almashtirish(x,y);
cout<<"Main(). Almashtirishdan keyin, x:";
cout<<x<< "y:" <<y<< "\n";
return 0;
}
void Almashtirish(int x, int y)
{
int temp;
cout<<"Almashtirish().Almashtirishdan ";
cout<<"oldin, x: "<<x<<" y: "<<y<< "\n";
temp = x;
x = y;
y = temp;
cout<<"Almashtirish().Almashtirishdan ";
cout<<"keyin, x: "<<x<<" y: "<<y<< "\n";
}
```

Натижа:

```
Main(). Almashtirishdan oldin,x: 5 y:10
Almashtirish().Almashtirishdan oldin, x:5 y:10
Almashtirish().Almashtirishdan keyin, x:10 y:5
Main(). Almashtirishdan keyin, x:5 y:10 x:
```

Функциянинг қайтарадиган қийматлари. Функция ё бирор бир реал қийматни, ё компиляторга ҳеч қандай қиймат қайтарилмаслиги ҳақида хабар берувчи `void` типидаги қийматни қайтаради.

Функцияни қиймат қайтариши учун `return` калитли сўзидан фойдаланилади. Бунда олдин `return` калитли сўзи, кейин эса қайтариладиган қиймат ёзилади. Қиймат сифатида эса ўзгармаслар каби бутун бир ифодаларни ҳам бериш мумкин. Масалан:

```
return 5 ;
return (x > 5) ;
return (MyFunction()) ;
```

`MyFunction()` функцияси бирор бир қиймат қайтаришидан келиб чиқсак, юқоридаги барча ифодалар тўғри келтирилган. `return(x>5)` ифодаси эса `x 5` дан катта бўлса `true`, кичик ёки тенг бўлса `false` мантиқий қийматини қайтаради.

Агарда функцияда `return` калит сўзи учраса ундан кейинги ифода бажарилади ва унинг натижаси функция чақирилган жойга узатилади. `return` оператори бажарилгандан кейин дастур функция чақирилган сатрдан кейинги ифодага ўтади. `return` калитли сўзидан кейинги функция танасидаги операторлар бажарилмайди.

Функция бир нечта `return` операторларини ўзида сақлаши мумкин.

Бир нечта `return` операторини қўлланилиши

```
#include <iostream>
using namespace std;
int IkkigaKupaytirish(int KupaytSon);
int main()
{
int natija=0;
int input;
cout << "Ikkiga ko`paytiriladigan sonni";
cout<< "kiriting(0 dan 10000 gacha):";
cin >> input;
cout << "\n IkkigaKupaytirish() funktsiyasi";
cout<< "chaqirilishidan oldin\n";
cout<<"Kiritilgan qiymat:" <<input;
cout<<"Ikkilangani:"<<natija<< "\n";
natija = IkkigaKupaytirish(input);
cout<<"\nIkkigaKupaytirish() funktsiyasidan";
cout<<"qaytgandan so`ng...\n";
cout<<"Kiritilgan qiymat:" <<input;
cout<<"Ikkilangani:"<<natija<< "\n";
return 0;
}
int IkkigaKupaytirish(int original)
{
if (original <= 10000)
return original*2;
else
return -1;
cout<< "Siz bu satrga o`ta olmaysiz!\n";
}
```

Натижа:

```
Ikkiga ko`paytiriladigan sonni kiriting (0 dan 10000
gacha ): 9000
IkkigaKupaytirish()  funktsiyasi  chaqirilishidan
oldin
Kiritilgan qiymat: 9000 Ikkilangani: 0

Ikkiga_kupaytirish() funktsiyasidan qaytgandan so`ng

Kiritilgan qiymat:9000 Ikkilangani: 18000

Ikkiga ko`paytiriladigan sonni kiriting (0 dan 10000
gacha ): 11000
IkkigaKupaytirish()  funktsiyasi  chaqirilishidan
oldin
Kiritilgan qiymat: 11000 Ikkilangani: 0

Ikkiga_kupaytirish() funktsiyasidan qaytgandan so`ng

Kiritilgan qiymat:11000 Ikkilangani: -1
```

Инлайн функциялар. Дастурда функция таърифланганда компилятор функция кодини бир марта машина кодига ўтказади ва дастурга маълумотларни стекка жойловчи инструкциялар қўшади. Аргументларни стека қўшиш, функцияга ўтиш ва қайтиш машина вақтини олади.

C++ компилятори *inline*, сўзини учратса бажарилувчи файлга хар бир функцияга мурожаат ўрнига функция операторларини қўяди. Шундай қилиб дастур эффективлигини ошириш мумкиндир.

Мисол:

```
#include <iostream>

using namespace std;
inline int  min(float a, float b)
{
if(a<b) return a; else return b;
}
int main()
{
int x=5,y=6,z;
z=min(x,y);
cout<<"z="<<z;
return 0;
}
```

```
Натижа
Z=5
```

BC++ компиляторида инлайн функция дастурга жойлаштирилиши учун рекурсив булмаслиги керак ва `for, while, do,switch, goto` операторлари ишлатилмаган бўлиши керак.

4.4. Функцияга параметрлар узатиш

Қиймат бўйича узатиш. Параметрлар қийматини функцияга узатишнинг иккита усули мавжуд: адрес бўйича ва қиймати бўйича.

C++ тилида чақирилган функция чақирувчи функциядаги ўзгарувчи қийматини ўзгартира олмайди. У фақат ўзининг вақтинчалик нусхасини ўзгартириши мумкин холос.

Чақираётган ва чақирилаётган функциялар параметрлар орқали ахборот алмашади

Қиймати бўйича узатишда қуйидаги амаллар бажарилади:

1. фактик параметрлар ўрнида турган ифодалар қийматлари ҳисобланади;
2. функциянинг формал параметрлари учун стекда хотира ажратилади;
3. хисобланган қийматлар формал параметр учун ажратилган адрес бўйича ёзилади, бунда турларнинг ўзаро мувофиқлиги текширилади ҳамда, зарурат туғилганда, улар қайта ўзгартирилади.

Мисол:

```
  double square(double a, double b, double c);
{
//funktsiya a, b, c uzunlikdagi tomonlarga ega bo‘lgan uchburchak maydonini qaytarib beradi.
  double s, r=(a+b+c)/2;
return s=sqtr(p*(p-a)*(p-b)*(p-c));//Geron formulasi
}
```

Шундай қилиб, стекка фактик параметрларнинг нусхалари киритилади ва функция операторлари ушбу нусхалар билан иш олиб боради. Фактик параметрларнинг ўзига функциянинг кириш ҳуқуқи йўқ, демак уларни ўзгартириш имкони ҳам йўқ.

Қиймат бўйича чақириш кулайлик туғдиради. Чунки функцияларда камроқ ўзгарувчиларни ишлатишга имкон беради. Мисол учун шу хусусиятни акс эттирувчи POWER функцияси вариантини келтирамиз:

```
int power(x,n)
{
int x,n;
int p;
```

```
for (p = 1; n > 0; --n)
p = p * x;
return (p);
}
```

Аргумент N вақтинчалик ўзгарувчи сифатида ишлатилади. Ундан то қиймати 0 булмагунча бир айрилади. N функция ичида ўзгариши функцияга мурожжат қилинган бошлангич қийматига таъсир қилмайди.

Адрес бўйича узатиш. Адрес бўйича узатишда стекка параметрлар адресларининг нусхалари киритилади, демакки, функцияда фактик параметр жойлаштирилган хотира уясига кириш ҳуқуқи пайдо бўлади ва функция бу параметрни ўзгартириши мумкин.

Адрес бўйича узатиш учун иловалар қўлланиши мумкин. Илова бўйича узатишда функцияга чақириш пайтида кўрсатилган параметр адреси узатилади, функция ичида эса параметрга барча мурожаатларнинг сезилмаган ҳолда номлари бекор қилинади.

Мисол учун туртбурчак юзи ва периметрини берилган томонлари бўйича хисоблаш функциясини қуйидагича тасвирлаш мумкин.

```
void pr(float a,float b, float& s, float& p)
{
p=2(a+b);
s= a*b;
}
```

Бу функцияга қуйидагича мурожаат килиниши мумкин `pr(a,b,&p,&s)`. Функцияга p ва s ўзгарувчиларнинг адреслари узатилади. Функция танасида шу адреслар бўйича `2*(a+b)` ва `a*b` қийматлар ёзилади.

Кейинги мисолда берилган икки ўзгарувчининг қийматларини ўзаро алмаштириш функциясидан фойдаланилади:

```
#include <iostream>
using namespace std;
 void change(int &a,int &b)
 {
 int r=a; a=b;b=r;
 };

 int main()
 {
 int x=1,y=5;
 change(x,y);
 cout<<"x="<<x<<" y="<<y;
return 0;
  }
```

```
Натижа
x=5 y=1
```

4.5. Функцияларни қўшимча юклаш

Бир хил номли ҳар хил функциялар. C++ тилида бир номдаги бир нечта функцияларни тузиш имконияти мавжуддир. Бу *бир хил номдаги хар хил функциялар* дейилади. Қайта юкланувчи функциялар бир – бирлари билан параметрлари рўйхати билан фарқ қилиши лозим. Бунда параметрлар ёки турли сонда аниқланиши, ёки типлари билан фарқланиши керак. Қуйидаги мисолларни кўриб чиқамиз:

```
int myFunction( int, int )
int myFunction( long, long)
int myFunction( long )
```

`myFunction()` функцияси учта турли параметрлар рўйхати билан хар хил номда аниқланяпти. Функциянинг биринчи ва иккинчи версиялари параметрлар типи билан, учинчиси эса параметрлар сони билан фарқ қилаяпти.

Ушбу функцияларнинг қайтарадиган қийматларининг типлари бир хил ёки турлича бўлиши мумкин. Лекин, параметрлари рўйхати бир хил бўлган иккита функция турли типдаги қиймат қайтарадиган қилиб аниқланса дастурни компиляция қилиш жараёнида хатолик юз беради.

Функцияларни бир хил ном билан аниқланиши уларнинг полиморфизми деб ҳам аталади. Полиморфизм сўзи `поли` *(*гр. `poly`*)* - кўп, `морфе` *(*гр. `morphë`*)* - шакл сўзидан олинган бўлиб, кўпшакллилик деган маънони англатади.

Функциянинг полиморфизми деганда дастурда бир нечта турли вазифаларни бажарувчи бир хил номдаги функциялар бўлиши тушунилади. Параметрлари сони ва типини ўзгартириш орқали бир нечта бир хил номдаги функцияларни аниқлаш мумкин. Бундай ҳолда функцияни чақиришда ҳеч қандай ноаниқлик бўлмайди, керакли функция параметрига мувофиқ тарзда аниқланади.

Мисол учун хар хил ўзгарувчиларни кўпайтириш учун қуйидаги функциялар киритилган бўлсин:

```
Double multy(double x) {return x*x*x;}
Double multy(double x, double y) {return x*y*y;}
Double multy(double x,double y, double z) {return
x*y*z;}
```

Қуйидаги мурожаатларнинг хар бири тўғри бажарилади:

```
multy(0.4)
multy(4.0,12.3)
multy(0.1,1.2,6.4);
```

Фараз қиламиз, сиз ихтиёрий берилган қийматни иккига кўпайтирадиган функция ёзмоқчисиз. Бу функцияга `int, long, float` ёки `double` типидаги қийматларни узатиш имконияти бўлишини ҳоҳладингиз. Бир хил номли хар хил функцияларсиз буни бажариш учун тўртта турли номдаги функцияларни ҳосил қилишингиз керак бўлади:

```
int DoubleInt(int);
long DoubleLong(long);
float DoubleFloat(float);
double DoubleDouble(double);
```

Бир хил номли функциялар ёрдамида эса уларни ўрнига ушбу функцияларни аниқлашимиз мумкин.

```
int Double(int);
long Double(long);
float Double (float);
double Double(Double);
```

Қайта юкланган функциялар чақирилганда компилятор айнан қайси вариантдаги функцияни ишлатиш кераклигини автоматик тарзда узатилаётган параметрларнинг типларига мувофиқ аниқлайди.

Функцияларнинг полиморфизми

```
#include <iostream>
using namespace std;
int Doubled(int);
long Doubled(long);
float Doubled(float);
double Doubled(double);
int main()
{
int myInt = 6500;
long myLong = 65000;
float myFloat=6.5;
double myDouble = 6.5e20;
int doubledInt;
long doubledLong;
float doubledFloat;
double doubledDouble;
cout<<"myInt: "<< myInt<< "\n";
cout<<"myLong: "<< myLong<< "\n";
cout<<"myFloat: "<< myFloat<< "\n";
cout<<"myDouble: "<< myDouble<< "\n";
doubledInt=Doubled(myInt);
doubledLong=Doubled(myLong);
doubledFloat=Doubled(myFloat);
doubledDouble=Doubled(myDouble);
cout<<"doubledInt:"<<doubledInt<<"\n";
```

```
cout<<"doubledLong:"<<doubledLong<<"\n";
cout<<"doubledFloat:"<<doubledFloat<<"\n";
cout<<"doubledDouble:"<<doubledDouble<<"\n";
return 0;
}
int Doubled(int original)
{
return 2*original;
}
long Doubled(long  original)
{
return 2*original;
}
float Doubled(float original)
{
return 2*original;
}
double Doubled(double  original)
{
return 2*original;
}
```

Натижа:

```
MyInt:       6500
MyLong:      65000
MyFloat:         6.5
MyDouble:    6.5 e+20
DoubledInt: 13000
DoubledLong: 130000
DoubledFloat: 13
DoubledDouble: 13e+21
```

Функцияларни қўшимча юклаш қоидалари.

1) Функциялар битта кўриниш сохасига тегишли бўлиши лозим.

2) Функциялар параметрлари кўзда тутилган қийматларга эга бўлиши мумкин, лекин хар хил функциялардаги бир хил параметрлар бир хил қийматга эга бўлиши керак.

3) Агар функциялар параметрлари таърифи фақат const модификатори ёки илова мавжудлиги билан фарқ килса, бу фукнкцияларни қўшимча юклаш мумкин эмас.

Масалан компилятор `int&f1(int&,const int&){. . .}` ва `int f1(int,int){. . .}` – функцияларни ажрата олмайди.

4.6. Рекурсия

Рекурсия. Функция ўз – ўзини чақириши рекурсия дейилади. У икки хил – тўғри рекурсия ва билвосита рекурсия бўлиши мумкин. Агарда функция ўзини ўзи чақирса бу тўғри рекурсия дейилади. Агарда функция бошқа бир функцияни чақирса ва у функция ўз навбатида биринчисини чақириши эса билвосита рекурсия дейилади.

Айрим муаммолар рекурсия ёрдамида осонроқ ечилади. Агарда маълумотлар устида бирор процедура ишлатилса ва унинг натижаси устида ҳам худди шу процедура бажарилса, бундай ҳолларда рекурсияни ишлатиш лозим. Рекурсия икки хил натижа билан якунланади: у ёки охир – оқибат албатта тугайди ва бирор натижа қайтаради, иккинчиси ҳеч қачон тугалланмайди ва хатолик қайтаради.

Мисол учун факториални хисоблаш функциясини келтирамиз:

```
long  fact(int k)
{
if (k<0) return 0;
if (k==0) return 1;
return k*fact(k-1);
}
```

Манфий аргумент учун функция 0 қиймат қайтаради. Параметр 0 га тенг бўлса функция 1 қиймат қайтаради. Акс холда параметр қиймат бирга камайтирилган холда функциянинг ўзи чақирилади ва узатилган параметрга кўпайтирилади. Функциянинг ўз ўзини чақириш формал параметр қиймати 0 га тенг бўлганда тўхтатилади.

Агарда функция ўзини ўзи чақирса, бу функцияни нусхаси чақирилишини айтиб ўтиш лозим. Рекурсия ёрдамида ечиладиган муаммо сифатида Фибоначчи қаторини кўрсатиш мумкин.

`1, 1, 2, 3, 5, 8, 13, 21, 34 …`

Бу қаторнинг ҳар бир сони (иккинчисидан кейин) ўзидан олдинги икки соннинг йиғиндисига тенг. Масала қуйидагидан иборат: Фибоначчи қаторини 12 – аъзоси нечага тенглигини аниқланг. Бу муаммонинг ечишнинг усулларидан бири қаторни батафсил таҳлил қилишдан иборатдир. Қаторнинг олдинги икки сони бирга тенг. Ҳар бир навбатдаги сон олдинги иккита соннинг йиғиндисига тенг. Яъни, ўн еттинчи сон ўн олтинчи ва ўн бешинчи сонлар йиғиндисига тенг. Умумий ҳолда `n` – соннинг йиғиндиси `(n-2)`- ва `(n-1)` – сонларнинг йиғиндисига тенг (`n>2` ҳол учун).

Рекурсив функциялар учун рекурсияни тўхтатиш шартини бериш зарур. Фибоначчи қатори учун тўхтатиш шарти `n<3` шартидир.

Бунда қуйидаги алгоритм қўлланилади:

1. Фойдаланувчидан Фибоначчи қаторининг қайси аъзосини ҳисоблашини кўрсатишини сўраймиз.

2. `fib()` функциясини чақирамиз ва унга фойдаланувчи томонидан берилган Фибоначчи қатори аъзоси тартиб номерини аргумент сифатида узатамиз.
3. `fib()` функцияси `(n)` аргументни таҳлил қилади. Агарда `n<3` бўлса, функция 1 қиймат қайтаради; акс ҳолда `fib()` функцияси ўзини ўзи чақиради ва унга аргумент сифатида `n-2` қийматни беради, кейин яна ўз–ўзини чақиради ва бу функцияга `n-1` ни қиймат сифатида беради. Бундан кейин эса уларнинг йиғиндисини қиймат сифатида узатади.

Агарда `fib(1)` функцияси чақирилса у `1` қийматни қайтаради. Агарда `fib(2)` функцияси чақирилса у ҳам `1` қийматни қайтаради. Агарда `fib(3)` функцияси чақирилса у `fib(1)` ва `fib(2)` функцияларини йиғиндисини қайтаради. `fib(2)` ва `fib(1)` функциялари `1` қийматни қайтарганлиги сабабли `fib(3)` функцияси қиймати `2` га тенг бўлади.

Агарда `fib(4)` функцияси чақирилса, бунда у `fib(3)` ва `fib(2)` функциялари йиғиндисини қиймат сифатида қайтаради. `fib(3)` функцияси 2 ва `fib(2)` функцияси 1 қиймат қайтаришидан `fib(4)` функцияси 3 ни қиймат сифатида қайтариши келиб чиқади. Демак, Фибоначчи қаторининг тўртинчи аъзоси 3 га тенг экан.

Юқорида, тавсифланган усул бу масалани ечишда унчалик самарали усул эмас. `fib(20)` функцияси чақирилса, у томонидан `fib()` функцияси 13529 марта чақирилади. Бундай ҳолларда эҳтиёт бўлиш лозим. Агарда Фибоначчи қаторининг жуда катта номерини берсак хотира етишмаслиги мумкин. `Fib()` функциясини ҳар сафар чақирилишида хотирадан маълум бир соҳа заҳираланади. Функциядан қайтиш билан хотира бўшатилади. Лекин, рекурсив тарзда функция чақирилса хотирадан унинг учун янги жой ажратилади ва токи ички функция ўз ишини тугатмагунча уни чақирган функция эгаллаган жой бўшатилмайди. Шунинг учун хотирани етишмай қолиш хавфи бор.

Фибоначчи қатори аъзосининг қийматини топиш учун рекурсияни қўлланилишига мисол

```
#include <iostream>
using namespace std;
int fib(int n);
int main( )
{
int n, javob;
cout << "Izlanayotgan nomerni kiriting:";
cin >> n;
cout << "\n\n";
javob=fib(n);
```

```
cout<< "Fibonachchi qatorining"<< n;
cout<<"nomeri qiymati " <<javob<<" ga teng \n";
return 0;
}
int fib(int n)
{
if (n <3)
{
return (1);
}
else
{
return(fib(n-2)+fib(n-1));
 }
}
```

Натижа:

```
Izlanayotgan nomerni kiriting:: 6
Fibonachchi qatorining 6 nomeri qiymati 8 ga teng
```

Айрим компиляторлар cout объекти қатнашган ифодаларда операторларни қуллашда баъзи қийинчиликлар пайдо бўлади. Агарда компилятор чиқариш операторидаги ифода ҳақида огоҳлантириш берса айириш операциясини қавс ичига олинг.

Рекурсияга мисол сифатида сонни сатр шаклида чикариш масаласини кўриб чиқамиз. Сон рақамлари тескари тартибда хосил бўлади. Биринчи усулда ракамларни массивда саклаб сўнгра тескари тартибда чикаришдир.

Рекурсив усулда функция хар бир чақирикда бош рақамлардан нусха олш учун ўз ўзига мурожаат қилади, сўнгра охирги рақамни босиб чиқаради.

```
printd( int n)
{
int i;
if (n < 0)
{
cout<< '-';
n = -n;
}
if ((i = n/10) != 0)
printd(i);
cout<< n % 10;
}
```

PRINTD(123) чакирикда биринчи функция PRINTD N = 123 қийматга эга. У 12 қийматни иккинчи PRINTD га узатади, бошкариш узига кайтганда 3 ни чикаради.

САВОЛЛАР

1. Нима учун барча ўзгарувчиларни глобал деб эълон қилиш мақсадга мувофиқ эмас.
2. Нима учун функцияга аргумент сифатида узатилган ўзгарувчилар қиймати функция танасида ўзгартирилса дастурнинг асосий кодидаги шу ўзгарувчи қийматига акслантирилмайди.
3. Функция прототипини эълон қилиш ва функцияни аниқлаш ўртасида қандай фарқ бор?
4. Функция прототипини кўрсатишда, аниқлашда ва чақиришда унинг параметрлари номлари устма – уст тушиши керакми?
5. Агарда функция ҳеч қандай қиймат қайтармаса уни қандай эълон қилиш керак?
6. Агарда функцияни эълон қилишда қайтарадиган типи аниқланмаса, у бошланғич равишда қандай тип қайтаради.
7. Локал ўзгарувчи нима?
8. Кўриниш соҳаси нима?
9. Бир хил номли хар хил функциялар қандай аниқланади?
10. Рекурсия нима?

МИСОЛЛАР

1. Убурчак юзасини, хамда тўғри тўртбурчак юзасини хисобловчи бир хил номли функциялар тузинг ва дастурда фойдаланинг.

2. Икки энг катта умумий бўлувчисини(ЭКУБ) хисобловчи функция тузинг ва дастурда фойдаланинг.

3. Рекурсия ёрдамида берилган бутун сон рақамлари суммасини хисобловчи функция тузинг ва дастурда фойдаланинг.

4. Рекурсия ёрдамида Паскаль учбурчагини хисобловчи функция тузинг. Бу функция ёрдамида учбурчакни экранга чиқапувчи функция тузиб дастурда фойдаланинг.

5. Учта сон максимумини хисобловчи жойлаштирилувчи `( inline)` функция тузинг ва дастурда фойдаланинг. Худди шу дастурни функция чақириғи ўрнида танаси жойлаштирилган вариантини яратинг.

5 боб. Массивлар ва сатрлар

5.1. Бир ўлчовли массивлар

Массив тушунчаси. Массив бу бир типли номерланган маълумотлар жамланмасидир. Массив индексли ўзгарувчи тушунчасига мос келади. Массив таърифланганда типи, номи ва индекслар чегараси кўрсатилади. Масалан `type` туридаги `length` та элементдан иборат `a` номли массив шундай эълон қилинади:

```
type a[length];
```

Бу махсус `a[0], a[1], ..., a[length -1]` номларга эга бўлган `type` туридаги ўзгарувчиларнинг эълон қилинишига тўғри келади.

Массивнинг ҳар бир элементи ўз рақамига - индексга эга. Массивнинг x-нчи элементига мурожаат индекслаш операцияси ёрдамида амалга оширилади:

```
int x=...;                          //butun sonli indeks
TYPE value=a[x];                 //ch-nchi elementni
o‘qish
a[x]=value;                        //x-yxb elementga
yozish
```

Индекс сифатида бутун тур қийматини қайтарадиган ҳар қандай ифода қўлланиши мумкин: `char, short, int, long`. C++ да массив элементларининг индекслари 0 дан бошланади (1 дан эмас), `length` элементдан иборат бўлган массивнинг охирги элементининг индекси эса - бу `length -1` (`length` эмас). Массивнинг `int z[3]` шаклдаги таърифи, `int` типига тегишли `z[0],z[1],z[2]` элементлардан иборат массивни аниқлайди.

Массив чегарасидан ташқарига чиқиш (яъни мавжуд бўлмаган элементни ўқиш/ёзишга уриниш) дастур бажарилишида кутилмаган натижаларга олиб келиши мумкин. Шуни таъкидлаб ўтамизки, бу энг кўп тарқалган хатолардан биридир.

Агар массив инициализация қилинганда элементлар чегараси кўрсатилган бўлса , рўйхатдаги элементлар сони бу чегарадан кам бўлиши мумкин, лекин ортиқ бўлиши мумкин эмас.

Мисол учун `int a[5]={2,-2}`. Бу холда `a[0]` ва `a[1]` қийматлари аниқланган бўлиб, мос холда 2 ва –2 га тенг. Агар массив узунлигига қараганда камроқ элемент берилган бўлса, қолган элементлар 0 ҳисобланади:

```
int a10[10]={1, 2, 3, 4};          //va 6 ta nol
```

Агар номланган массивнинг тавсифида унинг ўлчамлари кўрсатилмаган бўлса, компилятор томонидан массив чегараси автоматик аниқланади:

```
int a3[]={1, 2, 3};
```

Массивда мусбат элеменлар сони ва суммасини хисоблаш.

```
#include <iostream>
using namespace std;
int main()
int s=0,k=0;
int  x[]={-1,2,5,-4,8,9};
for(int i=0; i<6; i++)
{
if (x[i]<=0) continue;
k++;
s+=x[i];
};
cout<<k<<endl;
cout<<s;
return 0;
};
```

Массивнинг энг катта, энг кичик элементи ва ўрта қийматини аниқлаш:

```
#include <iostream>
using namespace std;
int main()
{
int i,j,n;
int min,max,s;
float a,b,d,x[100];
while(1)
{
cout<<("\n n=");
cin>>n;
if ( n>0 && n<=100 ) break;
cout<<"\n Hato  0<n<101 bo'lishi kerak";
}
cout<<"\n elementlar qiymatlarini kiriting:\n";
for (i=0;i<n;i++)
{
cout<<"x["<<i<<"]=";
cin>>x[i];
}
```

```
max=x[0];
min=x[0];
for (s=0,i=0;i<n;i++)
{
s+=x[i];
if (max<x[i]) max=x[i];
if (min>x[i]) min=x[i];
};
s/=n;
cout<<"\n max="<<max;
cout<<"\n min="<<min;
cout<<"\n o'rta qiymat="<<s;
return 0;
}
```

Массивларни навларга ажратиш. Навларга ажратиш - бу берилган кўплаб объектларни бирон-бир белгиланган тартибда қайтадан гуруҳлаш жараёни.

Массивларнинг навларга ажратилиши тез бажарилишига кўра фарқланади. Навларга ажратишнинг `n*n` та қиёслашни талаб қилган оддий усули ва `n*log(n)` та қиёслашни талаб қилган тез усули мавжуд. Оддий усуллар навларга ажратиш тамойилларини тушунтиришда қулай ҳисобланади, чунки содда ва калта алгоритмларга эга. Мураккаблаштирилган усуллар камроқ сонли операцияларни талаб қилади, бироқ операцияларнинг ўзи мураккаброқ, шунинг учун унча катта бўлмаган массивлар учун оддий усуллар кўпроқ самара беради.

Оддий усуллар учта асосий категорияга бўлинади:

- оддий киритиш усули билан навларга ажратиш;
- оддий ажратиш усули билан навларга ажратиш;
- оддий алмаштириш усули билан навларга ажратиш

Оддий киритиш усули билан навларга ажратиш

Массив элементлари аввалдан тайёр берилган ва дастлабки кетма-кетликларга бўлинади. `i=2` дан бошлаб, ҳар бир қадамда дастлабки кетма-кетликдан `i`-нчи элемент чиқариб олинади ҳамда тайёр кетма-кетликнинг керакли ўрнига киритиб қўйилади. Кейин `i` биттага кўпаяди ва ҳ.к.

Керакли жойни излаш жараёнида, кўпроқ ўнгдан битта позициядан танлаб олинган элементни узатиш амалга оширилади, яъни танлаб олинган элемент, `j:=i-1` дан бошлаб, навларга ажратиб бўлинган қисмнинг навбатдаги элементи билан қиёсланади. Агар танлаб олинган элемент `a[i]` дан катта бўлса, уни навларга ажратиш қисмига қўшадилар, акс ҳолда `a[j]` битта позицияга сурилади, танлаб олинган элементни эса навларга ажратилган кетма-кетликнинг навбатдаги элементи билан қиёслайдилар.

Тўғри келадиган жойни қидириш жараёни иккита турлича шарт билан тугалланади:

- агар a[j]>a[i] элементи топилган бўлса;
- агар тайёр кетма-кетликнинг чап учига етилган бўлса.

```
int i, j, x;
for(i=1; i<n; i++)
{
x=[i];//   kiritib   qo‘ishimiz   lozim   bo‘lgan elementni esda saqlab qolamiz
j=i-1;
while(x<a[j]&&j>=0)//to‘g‘ri    keladigan    joyni qidirish
}
a[j+1]=a[j] /o‘nga surilish
j--;
}
a[j+1]=x;//elementni kiritish
}
```

Оддий танлаш усули билан навларга ажратиш

Массивнинг минимал элементи танланади ҳамда массивнинг биринчи элементи билан жой алмаштирилади. Кейин жараён қолган элементлар билан такрорланади ва ҳ.к.

```
int i,min,n_min,j;
for(i=0;i<n-1;i++)
{
min=a[i];n_min=i; //minimal qiymatni qidirish
for(j=i+1;j<n;j++)
if(a[j]<min){min=a[j];n+min=j;}
a[n_min]=a[i];//almashtirish
a[i]=min;
}
```

Оддий алмаштириш усули билан навларга ажратиш

Элементлар жуфтлари охиргисидан бошлаб қиёсланади ва ўрин алмашинади. Натижада массивнинг энг кичик элементи унинг энг чапки элементига айланади. Жараён массивнинг қолган элементлари билан давом еттирилади.

```
for(int i=1;i<n;i++)
for(int j=n-1;j>=i;j—
if(a[j]<a[j-1])
```

```
{int r=a[j];a[j]=a[j-1];a[j-1]=r;}
}
```

Бир ўлчамли массивларни функция параметрлари сифатида узатиш. Массивдан функция параметри сифатида фойлаланганда, функциянинг биринчи элементига кўрсаткич узатилади, яъни массив ҳамма вақт адрес бўйича узатилади. Бунда массивдаги элементларнинг миқдори ҳақидаги ахборот йўқотилади, шунинг учун массивнинг ўлчамлари ҳақидаги маълумотни алоҳида параметр сифатида узатиш керак.

Мисол:

Массив барча элементлари чиқарилсин:

```
#include <iostream>
using namespace std;
int form(int a[100])
{
int n;

cout<<"\nEnter n:";
cin>>n;
for(int i=0;i<n;i++)
a[i]=rand()%100;
return n;
}
void print(int a[100],int n)
{
for(int i=0;i<n;i++)
cout<<a[i]<<" ";
cout<<"\n";
}
 int main()
 {
 int a[100];
 int n;
 n=form(a);
 print(a,n);
 return 0;
 }
```

Функцияга массив бошланиши учун кўрсаткич узатилгани туфайли (адрес бўйича узатиш), функция танасининг операторлари ҳисобига массив ўзгариши мумкин.

Функцияларда бир ўлчовли сонли массивлар аргумент сифатида ишлатилганда уларнинг чегарасини кўрсатиш шарт эмас.

Мисол:

Массивдан барча жуфт элементлар чиқарилсин:

```
#include <iostream>
using namespace std;
void form(int a[],int n)
{
    for(int i=0;i<n;i++)
 a[i]=rand()%100;
 }
void print(int a[],int n)
{
 for(int i=0;i<n;i++)
     cout<<a[i]<<" ";
 cout<<"\n";
}
int Dell(int a[],int n)
{
 int j=0,i,b[100];
 for(i=0;i<n;i++)
     if(a[i]%2!=0)
     {
         b[j]=a[i];j++;
     }
     for(i=0;i<n;i++)a[i]=b[i];
return j;
}
  int main()
  {
  int a[100];
  int n;
  cout<<"\nEnter n:";
  cin>>n;
  form(a,n);
  print(a,n);
  n=Dell(a,n);
      print(a,n);
  return 0;
  }
```

Функцияларда бир ўлчовли сонли массивлар аргумент сифатида ишлатилганда уларнинг чегарасини кўрсатиш шарт эмас. Мисол тарикасида `n` ўлчовли векторлар билан боғлиқ функцияларни таърифлашни кўриб чиқамиз.

Векторнинг узунлигини аниқлаш функцияси:

```
float mod_vec(int n,float x[])
{
float a=0;
for (int i=0; i<n; i++)
 a+=x[i]*x[i];
return sqrt(double(a));
}
```

Функцияларда бир ўлчовли массивлар кайтарилувчи қийматлар сифатида хам келиши мумкин. Мисол учун икки вектор суммасини хисобловчи функция процедурани кўрамиз:

```
void sum_vec(int n, float a,float b, float c)
{
for(int i=0;i<n;i++,c[i]=a[i]+b[i]);
}
```

Бу функцияга қуйидагича мурожаат килиш мумкин:

```
float a[]={1,-1.5,-2};
b[]={-5.2,1.3,-4},c[3];
sum_vec(3,a,b,c);
```

5.2. Кўп ўлчовли массивлар

Кўп ўлчовли массивлар таърифи. Икки ўлчовли массивлар математикада матрица ёки жадвал тушунчасига мос келади. Жадвалларнинг инциализация қилиш қоидаси, икки ўлчовли массивнинг элементлари массивлардан иборат бўлган бир ўлчовли массив таърифига асослангандир.

Мисол учун икки қатор ва уч устундан иборат бўлган хақиқий типга тегишли `d` массив бошланғич қийматлари қуйидагича кўрсатилиши мумкин:

```
float d[2][3]={(1,-2.5,10),(-5.3,2,14)};
```

Бу ёзув қуйидаги қиймат бериш операторларига мосдир:

```
d[0][0]=1;d[0][1]=-2.5;d[0][2]=10;
d[1][0]=-5.3;d[1][1]=2;d[1][2]=14;
```

Бу қийматларни битта рўйхат билан хосил қилиш мумкин:

```
float d[2][3]={1,-2.5,10,-5.3,2,14};
```

Инициализация ёрдамида бошланғич қийматлар аниқланганда массивнинг хамма элементларига қиймат бериш шарт эмас.

Мисол учун: `int x[3][3]={(1,-2,3),(1,2),(-4)}.`

Бу ёзув қуйидаги қиймат бериш операторларига мосдир:

```
x[0][0]=1;x[0][1]=-2;x[0][2]=3;
x[1][0]=-1;x[1][1]=2;x[2][0]=-4;
```

Инициализация ёрдамида бошланғич қийматлар аниқланганда массивнинг биринчи индекси чегараси кўрсатилиши шарт эмас, лекин қолган индекслар чегаралари кўрсатилиши шарт.

Мисол учун:

```
double x[][2]={(1.1,1.5),(-1.6,2.5),(3,-4)}
```

Бу мисолда автоматик равишда каторлар сони учга тенг деб олинади.

Қуйидаги кўрадиган мисолимизда жадвал киритилиб хар бир каторнинг максимал элементи аникланади:

```
#include <iostream>
using namespace std;
int main()
{
double a[4][3];
double max;
int i,j;
for(i=0;i<4;i++)
{
for(j=0;j<3;j++)
{
cout<<"a["<<i<<"]["<<j<<"]=";
cin>>a[i][j];
}
cout<<'\n';
};
for(i=0;i<4;i++)
{
max=a[i][0];
for(j=0;j<3;j++)

if (max<a[i][j]) max=a[i][j];
cout<<max<<endl;
}
return 0;
}
```

Функцияга кўп ўлчамли массивларни узатиш. Кўп ўлчамли массивларни функцияга узатишда барча ўлчамлар параметрлар сифатида узатилиши керак. C++ да кўп ўлчамли массивлар аниқланиши бўйича мавжуд эмас. Агар биз бир нечта индексга эга бўлган массивни тавсифласак (масалан, `int mas [3][4]`), бу дегани, биз бир ўлчамли `mas` массивини тавсифладик, бир ўлчамли `int [4]` массивлар эса унинг элементларидир

Мисол: Квадрат матрицани узатиш (транспортировка қилиш)

Агар `void transp(int a[][],int n){.....}` функциясининг сарлавҳасини аниқласак, бу ҳолда биз функцияга номаълум ўлчамдаги массивни узатишни хоҳлаган бўлиб қоламиз. Аниқланишига кўра массив бир ўлчамли бўлиши керак, ҳамда унинг элементлари бир хил узўнликда бўлиши керак. Массивни узатишда унинг элементларининг ўлчамлари ҳақида ҳам бирон нарса дейилмаган, шунинг учун компилятор хато чиқариб беради.

Бу муаммонинг энг содда ечими функцияни қуйидагича аниқлашдир:

`void transp(int a[][4],int n){.....}`, бу ҳолда ҳар бир сатр ўлчами 4 бўлади, массив кўрсаткичларининг ўлчами эса ҳисоблаб чиқарилади.

```
#include <iostream>
using namespace std;
const int N=4;//global ўзгарувчи
void input_array(int a[][N],int n)
{
int i,j;
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
cout<<"a["<<i<<"]["<<j<<"]=";
cin>>a[i][j];
}
cout<<'\n';
}
}
void print_array(int a[][N],int n)
{
int i,j;
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
cout<<"a["<<i<<"]["<<j<<"]=";
cout<<a[i][j]<<" ";
}
cout<<'\n';
}
}
void transp(int a[][N],int n)
{
int r;
for(int i=0;i<n;i++)
for(int j=0;j<n;j++)
if(i<j)
{
r=a[i][j];a[i][j]=a[j][i];a[j][i]=r;
}
}

 int main()
```

```
 {
int mas[N][N];
int n;
cout<<'\n'<<" n="; cin>>n;
input_array(mas,n);
transp(mas,n);
print_array(mas,n);return 0;
 }
```

Мисол тарикасида уч ўлчовли квадрат матрицани уч ўлчовли векторга кўпайтириш функциясини кўриб чиқамиз:

```
void umn_vec( float a[3][3],float b[3], float c[3])
{
for(int i=0; i<3; i++)
{
c[i]=0;
for(int j=0; j<3; j++)
c[i]+=a[i][j]*b[j];
};
}
```

Хар хил чегарали жадваллар билан функциялардан фойдаланишнинг бир йули бу олдиндан киритилувчи константалардан фойдаланишдир.

5.3. Белгили ахборот ва сатрлар

Сатрлар. C++ да белгили маълумотлар учун `char` тури қабул қилинган. Белгили ахборотни тақдим этишда белгилар, символли ўзгарувчилар ва матний константалар қабул қилинган.

Мисоллар:

```
const char c='c';//belgi - bir baytni egallaydi, uning qiymati o‘zgarmaydi

char a,b;//belgili o‘zgaruvchilar, bir baytdan joy egallaydi, qiymatlari o‘zgaradi.
```

C++ даги сатр - бу нул-белги - \0 (нул-терминатор)- билан тугалланувчи белгилар массиви. Нул-терминаторнинг ҳолатига қараб сатрнинг амалдаги узунлиги аниқланади. Бундай массивдаги элементлар сони, сатр тасвирига қараганда, биттага кўп.

Символли массивлар қуйидагича инициализация килинади:

`char capital[]="TASHKENT";` Бу холда автоматик равишда массив элементлари сони аниқланади ва массив охирига сатр кўчириш `'\0'` символи қўшилади.

Юқоридаги инициализацияни қуйидагича амалга ошириш мумкин:

```
char capital[]={'T','A','S','H','K','E','N','T','\0'};
```

Бу холда сўз охирида `'\0'` символи аниқ кўрсатилиши шарт.

Қиймат бериш оператори ёрдамида сатрга қиймат бериш мумкин эмас. Сатрни массивга ёки киритиш пайтида ёки номлантириш ёрдамида жойлаштириш мумкин.

Мисол:

```
#include <iostream>
using namespace std;
int main()
{
char s1[10]="string1";
int k=sizeof(s1);
cout<<s1<<"\t"<<k<<endl;
char s2[]="string2";
k=sizeof(s2);
cout<<s2<<"\t"<<k<<endl;
char s3[]={'s','t','r','i','n','g','3','\0'};
k=sizeof(s3);
cout<<s3<<"\t"<<k<<endl;
char *s4="string4";//satr ko'rsatkichi, uni
o'zgartirib bo'lmaydi
k=sizeof(s4);
cout<<s4<<"\t"<<k<<endl;
return 0;
}
```

Натижа:

```
string1 10
string2 8
string3 8
string4 4
```

Кейинги мисолимизда киритилган сўздан берилган харф олиб ташлаш дастури берилган:

```
#include <iostream>
using namespace std;
int main()
```

```
{
char s[100];
cin>> s;
int i, j;
for ( i = j = 0; s[i] != '\0'; i++)
if ( s[i] != 'c' )
s[j++] = s[i];
s[j] = '\0';
cout<< s;
return 0;
}
```

Хар гал `'c'` дан фарқли символ учраганда , у `j` позицияга ёзилади ва фақат шундан сўнг `j` қиймати 1 га ошади. Бу қуйидаги ёзувга эквивалент:

```
if ( s[i] != c )
s[j] = s[i];
j++;
```

Функциялар ва сатрлар. Функцияларда сатрлар ишлатилганда уларнинг чегарасини кўрсатиш шарт эмас. Сатрларнинг узунлигини хисоблаш `len` функциясии қуйидагича таърифлаш мумкин:

```
int len(char c[])
{ int m=0;
for(m=0;c[m]!='0';m++);
return m;
};
```

Шу функциядан фойдаланилган дастурни келтирамиз:

```
#include <iostream>
using namespace std;
int len(char c[])
{ int m=0;
while(c[m++]);
return m-1;
};
int main()
{
char e[]="Pro Tempore!";
cout<<"\n"<<len(e);return 0;
};
```

Бу функциянинг стандарт варианти `strlen` деб аталади ва бу функциядан фойдаланиш учун `string.h` сарлавхали файлидан фойдаланиш лозим.

Сатрдан нусха олиш strcpy функциясини C++ тилида қуйидагича таърифлаш мумкин:

```
#include <iostream>
using namespace std;
void strlen(char s1[], char s2[])
{
int i=0;
while(s2[i]!='\0') s1[i++]=s2[i];
s1[i]=s2[i];
}
int main()
{
char s1[]="aaa";
char s2[]="ddd";
strcpy(s1,s2);
cout<< s1;
return 0;
}
```

Натижа:

```
ddd
```

Берилган сатрни тескарига айлантирувчи функция:

```
reverse(char s[])
{
int c, i, j;
for(i = 0, j = strlen(s) - 1; i < j; i++, j--)
c = s[i];
s[i] = s[j];
s[j] = c;
}
```

Кейинги мисолимизда T каторни S катор охирига уловчи STRCAT(S,T) функциясини кўриб чиқамиз:

```
strcat(char s[], t[])
 {
int i, j;
i = j = 0;
while (s[i] != '\0')
i++;
while((s[i++] = t[j++]) != '\0')
}
```

5.4. Сўзлар массивлари

Сўзлар массивини киритиш. C++ тилида сўзлар массивлари икки ўлчовли символли массивлар сифатида таърифланади. Мисол учун:

```
char name[4][5].
```

Бу таъриф ёрдамида хар бири 5 та харфдан иборат бўлган 4 та сўзли массив киритилади. Сўзлар массивлари қуйидагича инициализация қилиниши мумкин:

```
char Name[3][8]={ "Анвар","Миркомил","Юсуф"}.
```

Бу таърифда хар бир сўз учун хотирадан 8 байт жой ажратилади ва хар бир сўз охирига ' \0' белгиси куйилади.

Сўзлар массивлари инициализация қилинганда сўзлар сони кўрсатилмаслиги мумкин. Бу холда сўзлар сони автоматик аниқланади:

```
char comp[][9]={ "компьютер","принтер","картридж"}.
```

Қуйидаги дастурда берилган харф билан бошланувчи сўзлар рўйхати босиб чиқарилади:

```
#include <iostream>
using namespace std;
int main()
{
char a[10][10];
char c;
for (int i=0;i<10;i++) cin>>a[i];
cin>>c;
for (int i=0;i<10;i++)
if (a[i][0]==c) cout<<a[i]<<endl;
return 0;
}
```

Қуйидаги дастурда фан номи, талабалар рўйхати ва уларнинг бахолари киритилади. Дастур бажарилганда икки олган талабалар рўйхати босиб чиқарилади:

```
#include <iostream>
using namespace std;
int main()
{
char a[10][10];
char s[10];
int k[10];
cin>>s;
for (int i=0;i<10;i++)
{
cin>>a[i];
cin>>k[i];
```

```
};
for (int i=0;i<10;i++)
if (k[i]==2) cout<<a[i]<<endl;
return 0;
}
```

Функциялар ва сўзлар массивлар. Сатрли массивлар функция аргументи сифатида ишлатилганда сатрларнинг умумий узунлиги аниқ кўрсатилиши шартдир.

Мисол тариқасида ихтиёрий сондаги сатрлар массивини алфавит бўйича тартиблаш функциясидан фойдаланилган дастурни кўриб чиқамиз:

```
#include <iostream>
using namespace std;
#define m 10
void sort(int n, char a[][m])
{
char c;
int i,j,l;
for (i=0;i<n;i++)
for (j=i+1;j<m;j++)
if  (a[i][0]<a[j][0] )
for(l=0;l<m;l++
)
{
c=a[i][l];
a[i][l]=a[j][l];
a[j][l]=c;
};
};
int main()
{
char aa[][m]={"Alimov","Dadashev","Boboev"};
 sort(3,aa);
for(int i=0; i<3;i++) cout<<a[i]<<endl;
}
```

5.6. Сатр мураккаб тип сифатида

`String` типи. Сатрлар билн ишлаш учун стандарт библиотекага кирувчи `string` мураккаб туридан фойдаланиш қулайдир.

Бу типдан фойдаланиш учун қуйидаги сарлавхали файлни улаш лозим:

```
#include <string>
```

Сатрларни таърифлашга мисоллар:

```
string st( "БАХО \n" ); //символлар сатри билан  инициаллаш
```

`string st2;` // бўш сатр

`string st3( st );` шу типдаги ўзгарувчи билан инициаллаш

Сатрлар устида амаллар. Сатрлар устида қуйидаги амаллар аниқланган:

- қиймат бериш (=);
- конкатенация ёки сатрларни улаш (+);
- қиймат бериб қўшиш амали (+=)
- икки амал эквивалентликни текшириш учун (==) ва (!=);
- индекс олиш ([]).
- солиштириш амаллари(<, <=,>, >=);

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
string s1="ABC";
cout<<s1[0]<<endl;
string s2="DEF";
string s3;
s1+=s2;
cout<<s1<<endl;
if(s1<=s2) s3="Yes";else s3="No";
cout<<s3;
return 0;
}
```

Натижа:

```
A
ABCDEF
Yes
```

Усуллар. Сатр узунлигини аниқлаш учун `size()` функциясидан фойдаланилади(узунлик тугалловчи символни хисобга олмайди).

`cout << "узунлик "<< st << ": " << st.size();`

Махсус `empty()` усули агар сатр бўш бўлса `true` қайтаради, акс холда `false` қайтаради:

`if ( st.empty() )` // тўғри: бўш

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
```

```
string s1="Hello";
string s2;
cout<<s1.size()<<endl;
cout<<s2.size()<<endl;
if(s2.empty()) cout<<"Yes";else cout<<"No";
return 0;
}
```

Натижа:

```
5
0
Yes
```

```
Киритилган сатрда 'a' харфи сонини хисоблаш
#include <iostream>
#include <string>
using namespace std;
int main()
{
int k=0;
string s;
cin>>s;
if(!s.empty())
{
for(unsigned i=0;i<s.size();i++)
if( s[i]=='a') k++;
};
cout<<k<<endl;
return 0;
}
for(int i=0;i<3;i++)
cout<<'\n'<<aa[i];
return 0;
}
```

Бу мисолда функция спецификациясида кўрсатиш шарт бўлган сатрлар массивининг иккинчи параметрини `m` константа орқали киритдик.

САВОЛЛАР

1. Сатр символли массивдан қандай фарқ қилади?
2. Бир ўлчовли массивларни инициализация қилиш усулларини кўрсатинг.
3. Кўп ўлчовли массив таърифи хусусиятларини келтиринг.
4. Кўп ўлчовли масивлар инициализацияси хусусиятлари.
5. Сатрларни инициализация қилиш усулларини кўрсатинг.

6. Сўзлар массиви қандай киритилади?

7. Қандай қилиб бир ўлчовли массивлар формал параметрлар сифатида ишлатилиши мумкин?

8. Қандай қилиб кўп ўлчовли массивлар формал параметрлар сифатида ишлатилиши мумкин?

9. Сатр `string` объект таърифлаш усуллари.

10. Сатр `string` объект усуллари.

МАСАЛАЛАР

1. Берилган массив қатъий камаювчи эканлигини текширувчи функция тузинг ва дастурда фойдаланинг.

2. Матрицани векторга кўпайтириш функциясини тузинг ва дастурда фойдаланинг.

3. Берилган сатр телефон рақами эканлигини аниқловчи функция тузинг ва дастурда фойдаланинг.

4. Берилган сатр ўзгарувчи эканлиги текширкувчи функция тузинг ва дастурда фойдданинг. Сатр `string` объектидан фойдаланинг.

5. Берилган жумладан энг қисқа сўз ажратиб олувчи функция тузинг ва дастурда фойдданинг. Сатр `string` объектидан фойдаланинг.

6 боб. Структуралар

6.1. Структура таърифи

Ўзгарувчиларнинг қўшимча турлари. Структура бу турли типдаги маълумотларнинг бирлаштирилган типдир. Структура хар хил типдаги элементлар-компоненталардан иборат бўлади.

Структуралар қуйидагича таърифланиши мумкин:

struct струтурали_тип_номи
{Элементлар_таърифлари}

Масалан:

```
struct Date
{
int year;
char  month, day;
};
```

Дастурда тузилма туридаги ўзгарувчи қуйидаги шаклда киритилади:

Тузилма_номи идентификаторларнинг_рўйхати;

Масалан:

```
Date s1, s2;
```

Мисол учун:

```
struct complex
{
double real;
double imag;
}
```

Бу мисолда комплекс сонни тасвирловчи структурали тип `complex` киритилган бўлиб, комплекс сон хақиқий кисмини тасвирловчи `real` ва мавхум қисмини тасвирловчи `imag` компоненталаридан иборатдир.

Конкрет структуралар бу холда қуйидагича тасвирланади:

```
complex sigma,alfa;
```

Қуйидаги мисолда каср сонни тасвирловчи `numerator` –суръат ва `denominator`-махраж компоненталаридан иборат структура таърифи келтирилган.

```
struct  fraction;
{
int numerator;
int denominator;
}
```

Бу холда конкрет структуралар қуйидагича тасвирланиши мумкин:

```
fraction beta;
```

Структуралар таърифланганда конкрет структуралар рўйхатини киритиш мумкин:

struct струтурали_тип_номи
{Элементлар_таърифлари}
Конкрет_структуралар_рўйхати.
Мисол:

```
struct student
{
char name[15];
char surname[20];
int year;
} student_1, student_2, student_3;
```

Бу холда `student` структурали тип билан бирга учта конкрет структура киритилади. Бу структуралар студент исми (`name[15]`), фамилияси (`surname[20]`), тугилган йилидан (`year`) иборат.

Структурали тип таърифланганда тип номи кўрсатилмай, конкрет стьруктуралар рўйхати кўрсатилиши мумкин:

struct
{Элементлар_таърифлари}
Конкрет_структуралар_рўйхати.

Қуйидаги таъриф ёрдамида учта конкрет структура киритилади, лекин структурали тип киритилмайди.

```
struct
{
char processor [10];
int frequency;
int memory;
int disk;
} IBM_486, IBM_386, Compaq;
```

Структураларга мурожаат. Конкрет структуралар таърифланганда массивлар каби инициализация килиниши мумкин. Масалан

```
complex sigma ={1.3;12.6};
goods coats={"пиджак",40000,7.5,220, "12.01.97");
```

Бир хил типдаги структураларга қиймат бериш амалини қўллаш мумкин:

```
Complex alfa; alfa=sigma;
```

Лекин структуралар учун солиштириш амаллари аникланмаган.

Структуралар элементларига қуйидагича мурожаат килиш мумкин:

Структура номи.элемент_номи.

' Нуқта амали' структура элементига мурожаат килиш амали дейилади. Бу амал қавс амаллари билан бирга энг юқори устиворликка эгадир.

Мисол:

```
Complex alfa={1.2,-4.5},betta={5.6,-7.8),sigma;
Sigma.real=alfa.real+betta.real;
Sigma.imag=alfa.imag+betta.imag;
```

Конкрет структуралар элементлари дастурда алохида киритилиши ва чиқарилиши зарурдир. Қуйидаги мисолда хизматчи структураси киритилади:

```
#include <iostream>
using namespace std;
struct employee
{
char name [64];
long employee_id;
float salary;
char phone[10];
int office_number;
} worker;
void show_emploee(employee worker)
    {
    cout << "Ismi: " << worker.name << endl;
    cout << "Telefon: " << worker.phone << endl;
    cout << "Nomer: " << worker.employee_id << endl;
    cout << "Oylik: " << worker.salary << endl;
    cout << "Ofis: " << worker.office_number << endl;
    };
int main()
{
worker.employee_id = 12345;
    worker.salary = 25000.00;
    worker.office_number = 102;
    cout<<"\n ismi:"; cin>> worker.name;
    cout<<"\n telefon:"; cin>>worker.phone;
cout<<endl;
show_emploee(worker);
return 0;
}
```

6.2. Структуралар ва массивлар

Массивлар структуралар элементлари сифатида. Массивларни структуралар элементи сифатида ишлатилиши хеч кандай кийинчилик туғдирмайди. Биз юқорида символли массивлардан фойдаланишни кўрдик.

Қуйидаги мисолда фазода берилган нуқтавий жисмни тасвирловчи компоненталари жисм массаси ва координаталаридан иборат структура киритилган бўлиб, нуқтанинг координаталар марказигача бўлган масофаси хисобланган.

```
#include <iostream>
using namespace std;
 #include <math.h>
struct
 {
double mass;
float coord[3];
} point={12.3,{1.0,2.0,-3.0}};
int main()
{
int i;
float s=0.0;
for (i=0;i<3; i++)
s+=point.coord[i]*point.coord[i];
cout<<"\n masofa="<<sqrt(s);return 0;
}
```

Бу мисолда `point` структураси номсиз структурали тип оркали аниқланган бўлиб, қийматлари инициализация ёрдамида аникланади.

Структуралар массивлари. Структуралар массивлари оддий масивлар каби тасвирланади. Юқорида киритилган структурали типлар асосида қуйидаги структуралар массивларини киритиш мумкин:

```
Struct goods list[100];
complex set [80];
```

Бу таърифларда `list` ва `set` структуралар номлари эмас, элементлари структуралардан иборат массивлар номларидир. Конкрет структуралар номлари бўлиб `set[0],set[1]` ва хоказолар хизмат қилади. Конкрет структуралар элементларига қуйидагича мурожаат қилинади: `set[0].real`- `set` массиви биринчи элементининг `real` номли компонентасига мурожаат.

Қуйидаги мисолда нуқтавий жисмларни тасвирловчи структуралар массиви киритилади ва бу нуқталар системаси учун оғирлик маркази координаталари (x_c, y_c, z_c) хисобланади. Бу координаталар қуйидаги формулалар асосида хисобланади:

$m=\sum m_i;\quad x_c=(\sum x_i m_i)/m;\quad y_c=(\sum y_i m_i)/m;\quad z_c=(\sum z_i m_i)/m;$

```
#include <iostream>
using namespace std;
```

```
struct particle
{
double mass;
double coord [3];
};
int main()
{
struct particle mass_point[]={ 20.0,
{2.0,4.0,6.0},40.0, {6.0,-2.0,6.0}, 10.0,
{1.0,3.0,2.0}};
int N;
struct particle center ={ 0.0, {0.0,0.0,0.0}};
int I;
N=sizeof(mass_point)/sizeof(mass_point[0]);
for (I=0;I<N;I++)
{
center.mass+=mass_point[I].mass;
center.coord[0]+= mass_point[I].coord[0]*
mass_point[I].mass;
center.coord[1]+= mass_point[I].coord[1]*
mass_point[I].mass;
center.coord[2]+= mass_point[I].coord[2]*
mass_point[I].mass;
}
cout<<"\n Massa markazi koordinatalari:";
for (I=0;I<3;I++)
{
center.coord[I]/=center.mass;
cout<<"\n Koordinata "<<(I+1)<<center.coord[I];
}
return 0;
}
```

6.3. Структура хоссалари

Структуралар ва функциялар. Структуралар функциялар аргументлари сифатида ёки функция қайтарувчи қиймат келиши мумкин. Бундан ташқари иккала холда хам структурага кўрсаткичлардан фойдаланиш мумкиндир.

Мисол учун комплекс сон модулини хисоблаш дастурини келтирамиз:

```
Double modul(complex a)
{return sqrt(a.real*a.real+a.imag*a.imag}
```

Икки комплекс сон йиғиндисини хисоблаш функцияси:

```
complex add(complex a, complex b)
```

```
{ complex c;
c.real=a.real+b.real;
c.imag=a.imag+b.imag;
return c;
}
```

Структуралар учун хотирадан жой ажратиш. Структура учун ажратилган жой хажмини қуйидаги амаллар ёрдамида аниқлаш мумкин:

Sizeof (структурали_тип_номи);
Sizeof (структура_номи);
Sizeof структура_номи.

Охирги холда структура номи ифода деб каралади. Ифоданинг типи аниқланиб, хажми хисобланади.

Мисол учун:
Sizeof (struct goods)
Sizeof (tea)
Sizeof coat

Мураккаб типлар яъни массивлар ва структурали типлар учун хотирага талаб уларнинг таърифига боғликдир. Масалан double array[10] таъриф хотирадан 10*sizeof байт жой ажратилишига олиб келади.

```
Struct mixture
{
int ii;
long ll;
char cc[8];
};
```

Бу таъриф хар бир Struct mixture типидаги объект хотирада `sizeof(int)+sizeof(long)+8*sizeof(char)` байт жой эгаллашини кўрсатади. Объъект аниқ хажмини қуйидаги амал хисоблайди:

```
Sizeof(struct mixture)
```

Хотирани текислаш. Структурали тип киритилиши бу тип учун хотирадан жой ажратилишига олиб келмайди. Хар бир конкрет структура (объект) таърифланганда, шу объект учун элементлар типларига қараб хотирадан жой ажратилади. Хотирадан жой зич ажратилганда структура учун ажратилган жой хажми хар бир элемент учун зарур бўлган хотира хажмлари йигиндисига тенг бўлади. Шу билан бирга хотирадан жой зич ажратилмаслиги хам мумкин яъни элементлар орасида бўш жойлар хам колиши мумкин. Бу бўш жой кейинги элементни хотира кисмларининг кабул қилинган чегаралари бўйича текислаш учун колдирилади. Мисол учун бутун

типдаги элементлар жуфт адреслардан бошланса, бу элементларга мурожаат тезроқ амалга оширилади.

```
#include <iostream>
using namespace std;
struct Foo
{
char c;
int i;
char s;
};
int main ()
{
cout<<"Foo hajmi="<<sizeof(Foo);
return 0;
}
```

Натижа:

```
Foo hajmi=12
```

Конкрет структураларни жойлашувига баъзи компиляторларда `#pragma` препроцессор директиваси ёрдамида таъсир ўтказиш мумкин. Бу директива қуйидаги шаклда:

```
Pragma pack(n)
```

Бу ерда `n` қиймати 1,2 ёки 4 га тенг бўлиши мумкин.

`Pack(1)` – элементларни байт чегаралари бўйича текислаш;

`Pack(2)` – элементларни сўзлар чегаралари бўйича текислаш;

`Pack(4)` – элементларни иккиланган сўзлар чегаралари бўйича

Масалан

```
#include <iostream>
#pragma pack(1)

using namespace std;

struct Foo
{
char c;
int i;
char s;
};
int main ()
{
cout<<"Foo hajmi="<<sizeof(Foo);
return 0;
}
```

Натижа:

```
Foo hajmi=6
```

6.4. Бирлашмалар

Структураларга якин тушунча бу бирлашма тушунчасидир. Бирлашмалар union хизматчи сузи ёрдамида киритилади. Мисол учун

```
union
{
long h;
int i,j;
char c[4]
}UNI;
```

Бирлашмаларнинг асосий хусусият шундаки унинг хамма элементлари бир хил бошлангич адресга эга бўлади.

Бирлашмаларнинг асосий авфзалликларидан бири хотира бирор кисми қийматини хар хил типдаги қиймат шаклида караш мумкиндир.

Мисол учун қуйидагича бирлашма

```
union
{
float f;
unsigned long k;
char h[4];
}fl;
```

Хотирага `fl.f=2.718` хақиқий сон юборсак унинг ички кўриниши кодини `fl.l` ёрдамида кўришимиз, ёки алохида байтлардаги қийматларни `fl.h[0]; fl.h[1]` ва хоказо ёрдамида куришимиз мумкин.

Қуйидаги дастур ёрдамида бирлашма хусусиятини текшириш мумкин:

```
#include <iostream>

using namespace std;

enum paytype{CARD,CHECK};
struct{
paytype ptype;
union{
 char card[4];
 long check;
 };
}info;

int main()
{
info.ptype=CHECK;
info.check=77;
switch (info.ptype)
```

```
    {
    case CARD:cout<<"\nKarta bilan
to'lash:"<<info.card;break;
    case CHECK:cout<<"\nChek bilan
to'lash:"<<info.check;break;
    }
    return 0;
    }
    Натижа
    Chek bilan to'lash:77
```

Разрядли майдонлар. Разрядли майдонлар структуралар ва бирлашмалар майдонларининг хусусий холидир. Разрядли майдон таърифланганда унинг узунлиги битларда кўрсатилади (бутун мусбат константа).

Мисол:

```
#include <iostream>
using namespace std;

struct
{
int a:8;
int b:6;
} xx={64,64};

int main()
{
cout<<xx.a<<endl;
cout<<xx.b<<endl;
return 0;
}
Натижа
64
0
```

Разрядли майдонлар ихтиёрий бутун типга тегишли бўлиши мумкин. Разрядли майдонлар адресини олиш мумкин эмас. Хотирада разрядли майдонларни жойлаштириш компилятор ва аппаратурага боғлиқ.

САВОЛЛАР

1. Структура умумий кўриниши.
2. Структуралар қандай инициализация қилинади?
3. Структура элементларига қандай мурожаат қилинади?
4. Структурага кўрсаткич таърифи.
5. Структура хажми қандай ўлчанади?
6. Структуралар таркибидаги массивлар элементларига қандай мурожаат қилинади?
7. Структуралар массиви қандай инициализация қилинади?

МАСАЛАЛАР

1. Абитуриент (исми, туғилган йили, йиғилган балл, аттестат ўрта бали) структурасини яратинг. Структура типидаги массив яратинг.

Кўрсатилган рақамли элементни олиб ташланг ва кўрсатилган фамилияли элементдан сўнг элемент қўшинг.

2. Ходим (исми, лавозими, туғилган йили, ойлиги).
структурасини яратинг. Структура типидаги массив яратинг.

Кўрсатилган фамилияли элементни олиб ташланг ва кўрсатилган рақамли элементдан сўнг элемент қўшинг.

3. Мамлакат (номи, пойтахт, ахоли сони, эгаллаган майдони).
структурасини яратинг. Структура типидаги массив яратинг.

Кўрсатилган ахоли сонидан кичик бўлган элементни олиб ташланг ва кўрсатилган номга эга бўлган элементдан сўнг элемент қўшинг.

4. Давлат (номи, давлат тили, пул бирлиги, валюта курси).
структурасини яратинг. Структура типидаги массив яратинг.

Кўрсатилган номга эга бўлган элементни ўчиринг ва файл охирига иккита элемент қўшинг.

5. Инсон (исми, яшаш манзили, телефон рақами, ёши).
структурасини яратинг. Структура типидаги массив яратинг.

Кўрсатилган ёшга эга бўлган элементни ўчиринг ва берилган телефон рақамидаги элементдан олдин элемент қўшинг.

6 боб. Кўрсаткичлар, массивлар, функциялар

7.1. Кўрсаткич ва иловалар

Кўрсаткичлар. Кўрсаткич - хотира уясининг уникал адресини сақлайдиган ўзгарувчи. Кўрсаткич оператив хотирадаги бирон-бир ўзгарувчи мавжуд бўлиши мумкин бўлган бирон-бир жойни белгилайди. Кўрсаткичларнинг қийматларини ўзгартириш, турли вариантларда қўллаш мумкинки, бу дастурнинг мослашувчанлигини оширади.

Кўрсаткич одатда типга эга бўлиб қуйидагича эълон қилинади:

<турнинг номи>*<кўрсаткичнинг номи>=<дастлабки қиймат>

Мисол учун:

```
int *pr;
char *alfa;
```

Бу холда кўрсаткичлар ноаниқ қийматга эга бўлади. Кўрсаткичлар таърифланганда уларнинг типлари кўрсатилиши шарт. Кўрсаткичларни инициализация килиш яъни бошлангич қийматларини киритиш мумкин. Маълум турдаги бирон-бир ўзгарувчи адреси ёки NULL қиймат дастлабки қиймат бўлиши мумкин. Кўрсаткичларга бошлангич махсус NULL қиймати берилса бундай кўрсаткич бўш кўрсаткич деб аталади.

Бирон-бир ўзгарувчи адресини олиш ҳамда уни кўрсаткичга қиймат сифатида бериш учун «&» оператори қўлланади.

Мисол:

```
int I=100;
int*p=&I;
unsigned longint *ul=NULL;
```

Мисол:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
int i=123;
int *p=&i;
cout<<"\n i="<<i<<" p="<<p;
int j=456;
p=&j;
cout<<"\n j="<<i<<" p="<<p;
return 0;
}
```

```
Kelib chiqadi:
i=123 p=0012FF60
j=456 p=0012FF48
```

Тескари оператор - «*» бўлиб, кўрсаткичда сақланаётган адрес бўйича уя қийматига мурожаат қилиш имконини беради.

Мисол:

```
int I=100;
int*p=&I
int J=*p;
```

Мисол:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
int i=123;
int *si=&i;
cout<<"\n i="<<i<<" *si="<<*si;
i=456;
cout<<"\n i="<<i<<" *si="<<*si;
*si=0;
cout<<"\n i="<<i<<" *si="<<*si;
return 0;
}
```

```
Kelib chiqadi:
i=123 *si=123
i=456 *si=456
i=0 *si=0
```

Илова тушунчаси. Илова (ссылка) – илова таърифирда кўрсатилган объект номининг синоними.

Иловани эълон қилиш шакли

тур & исм =исм_объект;

Мисоллар:

```
int x; // o‘zgaruvchining aniqlash
int& sx=x; //x o‘zgaruvchiga iqtibosni aniqlash
const char & CR='\n'; //konstantaga iqtibosni aniqlash
```

Иловалар билан ишлаш қоидалари.

1) Ўзгарувчи илова, агар у функция параметри бўлмаса, `extern` сифатида тавсифланмаган бўлса ёки синф майдонига илова қилмаса, ўзига тавсиф берилаётганда очиқ-ойдин номланиши керак.

2) Номлангандан сўнг, иловага бошқа қиймат берилиши мумкин эмас.

3) Иловаларга кўрсаткичлар, иловалар массивлари ва иловаларга иловалар бўлиши мумкин эмас.

4) Илова устида ўтказилган операция ўзи илова қилаётган қийматнинг ўзгаришига олиб келади

Мисол:

```
#include <iostream>
using namespace std;
  int main()
  {
   int i=123;
   int &si=i;
   cout<<"\n i="<<i<<" si="<<si;
   i=456;
   cout<<"\n i="<<i<<" si="<<si;
   si=0;
  cout<<"\n i="<<i<<" si="<<si;
  return 0;
  }
```

```
Kelib chiqadi:
i=123 si=123
i=456 si=456
i=0 si=0
```

7.2. Кўрсаткичлар хоссалари

Кўрсаткичлар устида ўтказиладиган операциялар. Кўрсаткичлар устида унар операциялар бажариш мумкин: инкремент ва декремент ++ ва -- операцияларини бажаришда, кўрсаткич қиймати кўрсаткич мурожаат қилган тур узунлигига кўпаяди ёки камаяди.

Мисол:

```
int*ptr, a[10];
ptr=&a[5];
ptr++;      */ = a[6]*/ elementining adresiga
ptr--;      */ = a[5]*/ elementining adresiga
```

Қўшиш ва айириш бинар операцияларида кўрсаткич ва `int` турининг қиймати иштирок этиши мумкин. Бу операция натижасида кўрсаткич

қиймати дастлабкисидан кўрсатилган элементлар сонига кўпроқ ёки камроқ бўлади.

Мисол:

```
int*ptr1, *ptr2, a[10];
int i=2;
ptr1=a+(i+4);      /* = a[6]*/ elementining
adresiga
ptr2=ptr1-i;         /* = a[4]*/ elementining
adresiga
```

Айириш операциясида битта турга мансуб бўлган иккита кўрсаткич иштирок этиши мумкин. Операция натижаси `int` турига эга ҳамда камаювчи ва айирувчи ўртасидаги дастлабки тур элементларининг сонига тенг, бундан ташқари агар биринчи адрес кичикроқ бўлса, у ҳолда натижа манфий қийматга эга бўлади.

Мисол:

```
int *ptr1, *ptr2, a[10];
int i;
ptr1=a+4;
ptr2=a+9;
i=ptr1-ptr2; /*=5 */
i=ptr1-ptr2; /*=-5 */
```

Бир турга тааллуқли бўлган иккита кўрсаткич қийматларини ==, !=, <, <=, >, >= амаллари ёрдамида ўзаро қиёслаш мумкин. Бунда кўрсаткичларнинг қийматлари шунчаки бутун сонлар сифатида олиб қаралади, қиёслаш натижаси эса 0 (ёлғон) ёки 1 (рост) га тенг бўлади.

Мисол:

```
int *ptr1, *ptr2, a[10];
ptr1=a+5;
ptr2=a+7;
if(ptr1>ptr2) a[3]=4;
```

Бу мисолда `ptr1` нинг қиймати `ptr2` нинг қийматидан камроқ, шунинг учун `a[3]=4` оператори бажарилмай қолади.

Константа кўрсаткич ва константага кўрсаткичлар. Константа кўрсаткич қуйидагича таърифланади:

<тип>* const<кўрсаткич номи>=<константа ифода>

Мисол учун: `char* const key_byte=(char*)0x0417.`

Бу мисолда константа кўрсаткич клавиатура холатини кўрсатувчи байт билан боглангандир.

Константа кўрсаткич қийматини ўзгартириш мумкин эмас лекин * амали ёрдамида хотирадаги маълумот қийматини ўзгартириш мумкин. Мисол учун `*key_byte='Ё'` амали 1047(0x0417) адрес қиймати билан бирга клавиатура холатини хам ўзгартиради.

Константага кўрсаткич қуйидагича таърифланади:

<тип>`const`*<кўрсаткич номи>=<константа ифода>.

Мисол учун `const int zero=0; int const* p=&zero;`

Бу кўрсаткичга * амалини қўллаш мумкин эмас, лекин кўрсаткичнинг қийматини ўзгартириш мумкин. Қиймати ўзгармайдиган константага кўрсаткичлар қуйидагича киритилади:

<тип>`const*` const<кўрсаткич номи>=<константа ифода>.

Мисол учун

```
const float pi=3.141593; float  const* const pp=&pi;
```

Турлаштирилмаган кўрсаткич. Турлаштирилмаган (типиклаштирилмаган) кўрсаткич `void` турга эга бўлиб, ихтиёрий турдаги ўзгарувчи адреси қиймат сифатида берилиши мумкин.

Махсус `void` типидаги кўрсаткичлар аждодий кўрсаткичлар деб аталиб хар хил типдаги объектлар билан боғланиш учун ишлатилади.

Мисол учун:

```
int I=77;
float Euler=2.18282;
void *vp;
Vp=&I;
cout<< (*(int*)vp;
Vp=&Euler;
cout<< (*(float*)vp;
```

Қуйидаги операторлар кетма кетлиги хатоликка олиб келади:

```
void *vp;int *ip; ip=vp;
```

Бу хатолик сабаби битта объектга хар хил типдаги кўрсаткичлар билан мурожаат килиш мумкин эмас.

Кўрсаткчлар функция параметри сифатида. Кўрсаткичлар ёрдамида параметр қийматини ўзгартириш мумкин.

Мисол учун туртбурчак юзи ва периметрини берилган томонлари бўйича хисоблаш функциясини қуйидагича тасвирлаш мумкин.

```
void pr(float a,float b, float* s, float* p)
{
*p=2(a+b);
*s= a*b;
}
```

Бу функцияга қуйидагича мурожаат килиниши мумкин `pr(a,b,&p,&s)`. Функцияга p ва s ўзгарувчиларнинг адреслари узатилади. Функция танасида шу адреслар бўйича `2*(a+b)` ва `a*b` қийматлар ёзилади.

Кейинги мисолда берилган икки ўзгарувчининг қийматларини ўзаро алмаштириш функциясидан фойдаланилади:

```
#include <iostream>
using namespace std;
  void Change(int*a,int*b)//manzil bo'yicha uzatish
  {
  int r=*a; *a=*b;*b=r;
  }
  int main()
  {
  int x=1,y=5;
  Change(&x,&y);
  cout<<"x="<<x<<"y="<<y;
  return 0;
  }
  Натижа
  x=5y=1 kelib chiqadi.
```

Кўрсаткичлар ўрнига иловалардан фойдаланиш дастурнинг ўқилишини яхшилайди, чунки бу ўринда адрес бўйича қиймат олиш операциясини қўлламаса ҳам бўлади. Қиймат бўйича узатиш ўрнига иловалардан фойдаланиш самаралироқ ҳамдир, чунки параметрларни нусхалашни талаб қилмайди. Агар функция ичида параметрнинг ўзгаришини тақиқлаш лозим бўлиб қолса, бу ҳолда `const` модификатори қўлланади. Бу модификаторни функцияда ўзгариши кўзда тутилмаган барча параметрлар олдидан қўйиш тафсия қилинади (ундаги қайси параметрлар ўзгарадию, қайсилари ўзгармаслиги сарлавҳадан кўриниб туради).

7.3. Кўрсаткичлар ва массивлар

Кўрсаткич ва массив номи. Массивлар номи дастурда константа кўрсаткичдир. Шунинг учун хам `int z[4]` массив элементига `*(z+4)` шаклда мурожаат килиш мумкин.

Массивлар билан ишланганда кавсларсиз ишлаш мумкин.

Қуйидаги мисолда хар бир харф алохида босиб чикарилади:

```
#include <iostream>
using namespace std;
int main()
{
char x[]="DIXI";
int i=0;
```

```
while (*(x+i)!='\0')
cout<<'\n'<<*(x+i++);
return 0;
}
```

Компилятор x[i] қийматни хисоблаганда (x+i)*sizeof(<>) формула бўйича адресни хисоблаб шу адресдаги қийматни олади.

Агар массив <тип> x[n][k][m] шаклда таърифланган бўлса, x[i][j][l] қийматни хисобланганда (x+k*j+l) *sizeof(<>) формула бўйича адресни хисоблаб шу адресдаги қийматни олади.

Массивларни функциялар параметрлари сифатида. Массивлар функцияга туридаги бир ўлчамли массивлар сифатида ёки кўрсаткичлар сифатида узатилиши мумкин. Масалан сатрлар функцияга char туридаги бир ўлчамли массивлар сифатида ёки char* туридаги кўрсаткичлар сифатида узатилиши мумкин. Оддий массивлардан фарқли ўлароқ, функцияда сатр узунлиги кўрсатилмайди, чунки сатр охирида сатр охири \0 белгиси бор.

Мисол: Берилган белгини сатрда қидириш функцияси

```
int  find(char *s,char c)
{
for (int i=0;i<strlen(s);i++)
if(s[i]==c) return i;
return -1
}
```

Массив функция қиймати сифатида. Массив қиймат қайтарувчи функция таърифи:

```
float *sum_vec(int n,float a,float b)
{
float d[n];
for(int i=0;i<n;i++,d[i]=a[i]+b[i]);
return d;
}
```

Бу функцияга қуйидагича мурожаат килиш мумкин:

```
float a[]={1,-1.5,-2},b[]={-5.2,1.3,-4};
float c[]=sum_vec(3,a,b);
```

Кўп ўлчамли массивлар ва кўрсаткичлар. C++ да массивнинг энг умумий тушунчаси - бу кўрсаткичдир, бунда ҳар хил турдаги кўрстакич бўлиши мумкин, яъни массив ҳар қандай турдаги элементларга, шу

жумладан, массив бўлиши мумкин бўлган кўрсаткичларга ҳам эга бўлиши мумкин. Ўз таркибида бошқа массивларга ҳам эга бўлган массив кўп ўлчамли ҳисобланади.

Бундай массивларни эълон қилишда компютер хотирасида бир нечта турли хилдаги объект яратилади. Масалан, `int app[4][3]`

```
App
↓
app[0]→ app[0][0]     app[0][1]     app[0][2]
app[1]→ app[1][0]     app[1][1]     app[1][2]
app[2]→ app[2][0]     app[2][1]     app[2][2]
app[3]→ app[3][0]     app[3][1]     app[3][2]
```

Шундай қилиб, `app[4][3]` нинг эълон қилиниши дастурда учта турли хилдаги объектларни юзага келтиради: `app` идентификаторли кўрсаткични, тўртта кўрсаткич дан иборат номсиз массивни ва `int` туридаги ўн иккита сондан иборат номсиз массивни. Номсиз массивларга кириш ҳуқуқига эга бўлиш учун app кўрсаткичли адресли ифодалар қўлланади. Кўрсаткичлар массиви элементларига кириш ҳуқуқи `app[2]` ёки `*(app+2)` шаклидаги индексли ифоданинг биттасини кўрсатиш орқали амалга оширилади. `int` туридаги икки ўлчамли сонлар массивига кириш учун `app[1][2]` шаклидаги иккита индексли ифода ёки унга эквивалент бўлган `*(*(app+1)+2)` ва `(*(app+1))[2]` шаклидаги ифодалар қўлланиши керак.

7.4. Динамик массивлар

Кўрсаткичлар массивлари. Кўрсаткичлар массивлари қуйидагича таърифланади

<тип> *<ном>[<сон>]

Мисол учун `int *pt[6]` таъриф `int` типидаги объектларга олти элементли массивни киритади.

Кўрсаткичлар массивлари сатрлар масссивларини тасвирлаш учун қулайдир.

Мисол учун фамилиялар рўйхатини киритиш учун икки ўлчовли массивдан фойдалани керак.

```
char fam[][20]={ "Олимов","Рахимов","Эргашев"}
```

Хотирада 60 элементдан иборат бўлади, чунки хар бир фамилия 20 гача 0 лар билан тўлдирилади.

Кўрсаткичлар массиви ёрдамида бу массивни қуйидагича таърифлаш мумкин.

```
char *pf[]=  { "Олимов","Рахимов","Эргашев"}.
```

Бу холда рўйхат хотирада 23 элементдан иборат бўлади, чунки хар бир фамилия охирига 0 белгиси куйилади

Хар хил чегарали жадваллар билан функциялардан фойдаланишнинг бир йули бу олдиндан киритилувчи константалардан фойдаланишдир. Лекин асосий йули кўрсаткичлар массивларидан фойдаланиш.

Қуйидаги мисолда кўрсаткичлар массиви ёрдамида сўзлар массиви тартибланади:

```
#include <iostream>
using namespace std;
void sort(int n, char* a[])
{
char* c;
int i,j;
for (i=0;i<n;i++)
for (j=i+1;j<n;j++)
if  (a[i][0]<a[j][0] )
{
c=a[i];a[i]=a[j];a[j]=c;
};
};
int main()
{
char* pa[10]={"Alimov","Dadashev","Boboev"};
sort(3,pa);
for(int i=0;i<3;i++) cout<<'\n'<<pa[i];
return 0;
}
```

Бу мисолда шунга эътибор бериш керакки сатрли массивнинг тартибланмаган эълементларини алмаштириш учун қўшимча цикл киритилган эди. Кўрсаткичлар массивининг элементлари адреслардан иборат бўлгани учун, қўшимча циклга хожат қолмади.

Бир ўлчовли динамик массивлар. С++тилида ўзгарувчилар ё статик тарзда - компиляция пайтида, ёки стандарт кутубхонадан функцияларни чақириб олиш йўли билан динамик тарзда - дастурни бажариш пайтида жойлаштирилиши мумкин. Асосий фарқ ушбу усулларни қўллашда кўринади - уларнинг самарадорлиги ва мослашувчанлигида. Статик жойлаштириш самаралироқ, чунки бунда хотирани ажратиш дастур бажарилишидан олдин содир бўлади. Бироқ бу усулнинг мослашувчанлиги анча паст, чунки бунда биз жойлаштирилаётган объектнинг тури ва ўлчамларини аввалдан билишимиз керак бўлади. Масалан, матний файлнинг ичидагисини сатрларнинг статик массивида жойлаштириш қийин: аввалдан унинг ўлчамларини билиш керак бўлади. Номаълум сонли элементларни олдиндан сақлаш ва ишлов бериш керак бўлган масалалар одатда хотиранинг динамик ажратилишини талаб қилади.

Хотирани динамик ва статик ажратиш ўртасидаги асосий фарқлар қуйидагича:

- статик объектлар номланган ўзгарувчилар билан белгиланади, ҳамда ушбу объектлар ўртасидаги амаллар тўғридан-тўғри, уларнинг номларидан фойдаланган ҳолда, амалга оширилади. Динамик объектлар ўз шахсий отларига эга бўлмайди, ва улар устидаги амаллар билвосита, кўрсаткичлар ёрдамида, амалга оширилади;
- статик объектлар учун хотирани ажратиш ва бўшатиш компилятор томонидан автоматик тарзда амалга оширилади. Дастурчи бу ҳақда ўзи қайғуриши керак эмас. Статик объектлар учун хотирани ажратиш ва бўшатиш тўлалигича дастурчи зиммасига юклатилади. Бу анчайин қийин масала ва уни ечишда хатога йўл қўйиш осон.

Динамик тарзда ажратилаётган хотира устида турли хатти-ҳаракатларни амалга ошириш учун `new` ва `delete` операторлари хизмат қилади.

Маълум бир турдаги элементлардан ташкил топган берилган ўлчамлардаги массивга хотира ажратиш учун new операторидан фойдаланиш лозим:

```
int *pia=new int[4];
```

Бу мисолда хотира `int` туридаги тўртта элементдан иборат массивга хотира ажратилади. Афсуски, `new` операторининг бу шакли массив элементларини номлантириш (инициаллаштириш) имконини бермайди.

Динамик массивни бўшатиш учун delete операторидан фойдаланиш лозим:

```
delete[] pia;
```

Агар ажратилган хотирани бўшатиш эсдан чиққудек бўлса, бу хотира бекордан-бекорга сарфлана бошлайди, фойдаланилмай қолади, бироқ, агар унинг кўрсаткичи ўз қийматини ўзгартирган бўлса, уни тизимга қайтариш мумкин эмас. Бу ҳодиса хотиранинг йўқотилиши (утечка памяти) деган махсус ном билан аталади. Пировард натижада дастур хотира етишмагани туфайли авария ҳолатида тугалланади (агар у анча вақт ишлайверса).

Икки ўлчовли динамик массивлар. Матрицани шакллантиришда олдин бир ўлчовли массивларга кўрсатувчи кўрсаткичлар массиви учун хотира ажратилади, кейин эса параметрли циклда бир ўлчовли массивларга хотира ажратилади.

Мисол:

```
int n;
cin>>n;
double *matr[100];
for (i=0;i<n;i++)  matr[i]=new int[n];
```

Хотирани бўшатиш учун бир ўлчовли массивларни бўшаттирувчи циклни бажариш зарур.

```
for(int i=0;i<n;i++)
delete matr[i];
```

Қуйидаги мисолда матрицани транспонирлаш функцияси ишлатилади.

```
#include <iostream>
using namespace std;
void trans(int n,double *p[])
{
double x;
for (int i=0;i<n-1;i++)
for (int j=i+1;j<n;j++)
{
x=p[i][j];
p[i][j]=p[j][i];
p[j][i]=x;
}
};
void inp_matr (int n,double *p[])
{
double x;
for (int i=0;i<n;i++)
for (int j=0;j<n;j++)
{
cin>>x;
p[j][i]=x;
}
};
void print_matr (int n,double *p[])
{
for (int i=0;i<n;i++)
{
cout<<endl;
for (int j=0;j<n;j++)
cout<<p[j][i]<<" ";
}
};
void creat_matr (int n,double *p[])
{
for (int i=0;i<n;i++)
p[i]=new double[n];
};

void delete_matr(int n,double *p[])
```

```
{
for(int i=0;i<n;i++)
delete p[i];
//delete [] p;
};

int main()
{
double *ptr[100];
int n=3;
creat_matr(n,ptr);
inp_matr(n,ptr);
trans(n,ptr);
print_matr(n,ptr);
return 0;
}
```

7.5. Функцияга кўрсаткич

Функцияларни чақиришда фойдаланиш. C++ тили синтаксисига кўра функцияга кўрсаткич функция адресини акс эттирувчи ўзгарувчи ёки ифодадир. Функцияга кўрсаткич бажарилувчи қиймати функция кодининг биринчи байти адресидир. Функцияга кўрсаткичлар устида арифметик амаллар бажариш мумкин эмас. Энг кенг қўлланувчи функцияга константа кўрсаткич функциянинг номидир. Функцияга ўзгарувчи кўрсаткич функция таърифи ва прототипидан алохида киритилади. Функцияга ўзгарувчи кўрсаткич қуйидагича тасвирланади:

<функция типи> (* кўрсаткич номи)(параметрлар спецификацияси).

Мисол учун `int (*point) (void)`.

Бу таърифда қавслар мухим ахамиятга эга, чунки қавслар ёзилмаса бу таъриф параметрсиз функция прототипи деб каралади. Функцияга ўзгарувчи кўрсаткич қийматлари сифатида, бир хил типга эга бўлган хар хил функциялар адресларини берилиши мумкин.

Қиймати бирор функция адресига тенг бўлган функцияга ўзгарувчи кўрсаткич шу функцияга мурожаат килиш учун ишлатилиши мумкин.

Қуйидаги мисода функцияга уч хил мурожаат қилиш кўрсатилган:

```
#include <iostream>
using namespace std;
void f1()
{
cout<<"\nf1 функция бажарилди";
};
void f2()
{
```

```
cout<<"\nf2 функция бажарилди";
};

int main()
{
void (*ptr) (void);
ptr=f1;
(*ptr)();
ptr=f1;
(*ptr)();
return 0;
}
```

Дастурда функцияга костанта кўрсаткич яъни номлари орқали ва ўзгарувчи кўрсаткичлар ёрдамида мурожаат қилишнинг хамма усуллари кўрсатилгандир. Шуни таъкидлаш лозимки адрес олиш * амали қўлланилганда қавслар ишлатиш шартдир.

Функцияга ўзгарувчи кўрсаткич таърифланганда инциализация қилиш, яъни бошланғич қиймат сифатида ўзгарувчи кўрсаткич билан бир хил типга эга бўлган функция адресини кўрсатиш мумкин. Мисол учун:

```
int fic (char);
int (*pfic) (char)=fic;
```

Функцияларга иловалар. Функияга кўрсаткич қандай аниқланса функцияга илова ҳам худди шундай аниқланади:

функция_тури(&илова_номи)(параметрлар)номлантирувчи_ифода;

Мисол:

```
int(&fret)(float,int)=f;// илови аниқлаш
```

Функция номини параметрларсиз ва қавсларсиз қўллаш функция адреси сифатида қабул қилинади. Функцияга илова функция номининг синоними бўлади. Функцияга илованинг қийматини ўзгартириб бўлмайди, шунинг учун кўп ўринда иловага кўрсаткичлар эмас, функцияга кўрсаткичлар қўлланади.

Мисол:

```
#include <iostream>
using namespace std;
void f(char c)
{
cout<<"\n"<<c;
}
int main()
```

```
{
void(*pf)(char);//funktsiya ko‘rsatkichi
void(&rf)(char)=f;//iqtibos ko‘rsatkichi
f('A');//nomi bo‘yicha chaqirish
pf=f;//ko‘rsatkich funktsiyaga ko‘yiladi
(*pf)('B');//ko‘rsatkich yordamida chaqirish
rf('S');//iqtibos bo‘yicha chaqirish
return 0;
}
```

Функцияга кўрсаткичлар параметр сифатида. Функцияга кўрсаткичларларни функцияларга параметр сифатида узатиш мумкин.

Бундай узатиш номи олдиндан белгиланмаган функциялар билан ишлашга имкон беради. Мисол учун тўртбурчаклар усули ёрдамида интеграл хисоблаш функциясидан фойдаланилган дастурни қараб чиқамиз. Дастурда $x/(x^2+1)^2$ функцияси интеграли -1 ва 2 ораликда, $4*\cos^2 x$ функцияси интеграли 0 ва ½ ораликда хисобланади.

```
#include <math.h>
#include <iostream>
using namespace std;
double ratio(double x)
{
double z;
z=x*x+1;
return x/(z*z);
};
double cos(double v)
{
double w;
w=cos(v);
return  4*w*w;
};
double rectangle(double (*pf)(double),double
a,double b)
{
int n=20;
int i;
double h,s=0.0;
h=(b-a)/n;
for (i=0;i<n;i++)
s+=pf(a+h/2+i*h);
return h*s;
};
int main()
{
```

```
double a,b,c;
a=-1;
b=2.0;
c=rectangle(ratio,a,b);
cout<<c;
c=rectangle(cos,0.0,0.5);
cout<<c;
return 0;
}
```

Бу дастурда интегрални хисоблаш `rectangle` функциясини чақириш оркали бажарилади.

Бу функция қуйидаги параметрларга эга: `pf` – `double` типли параметрга эга ва шу типдаги қиймат кайтарувчифункцияга кўрсаткич, a ва b параметрлари интеграллаш чегараларидир. Интегралланувчи функциялар қийматлари `ratio()` ва `cos4_2()` функцияларни чақириш орқали хисобланади. Дастурда `rectangle()` функцияси икки марта чақирилиб, биринчи холда `pf` кўрсаткич қиймати `ratio()` функцияси адресига тенг, иккинчи холда `cos4_2()` функцияси адресига тенгдир.

Дастур бажарилиши натижалари:

0.149847

1.841559

САВОЛЛАР

1. Кўрсаткич таърифини келтиринг.
2. Илова кўрсаткичдан қандай фарқ қилади?
3. Кўрсаткичлар билан боғлиқ амалларни келтиринг
4. Кўрсаткич ва массив номи орасида қандай фарқ бор?
5. Кўрсаткичлар массиви қандай таъриф қилинади?
6. Амаллар new ва delete нима учун ишлатилади?
7. Динамик массивлар оддий массивлардан қандай фарқ қилади?
8. Динамик массивларни хосил қилиш усулларини кўрсатинг.
9. Функцияга кўрсаткич таърифи умумий кўриниши.
10. Функцияга кўрсаткич ва функцияга илова орасида қандай фарқ мавжуд?

МИСОЛЛАР

1. Берилган сатр ўзгарувчи эканлигини аниқловчи функция тузинг ва дастурда фойдаланинг. Функция танасида фақат кўрсаткичлар устида амаллардан фойдаланинг.

3. Матрицани векторга кўпайтириш функциясини яратиб дастурда фойдаланинг. Матрица кўрсаткичлар массиви сифатида киритилсин.

4. Динамик равишда ўқувчилар фамилиялари ва бахолари массивларини хосил қилувчи функциялар яратинг. Хамма аълочилар фамилияларини чиқарувчи функция тузиб дастурда фойдаланинг.

5. Учбурчак динамик массив ёрдамида Паскаль учбурчагини хисобловчи функция тузинг. Бу функциядан ташқари учбурчакни экранга чиқапувчи функция тузиб дастурда фойдаланинг.

2. Дихотомия усули ёрдамида f(x)=0 тенгламани ечиш учун функция тузинг.Функцияга кўрсаткич параметр сифатида узатилсин.

8 боб. Препроцессор воситалари

8.1. Препроцессор тушунчаси

Дастур матни ва препроцессор. C++ тилида матнли файл шаклида таерланган дастур учта қайта ишлаш босқичларидан ўтади.

Матнни препроцессор директивалари асосида ўзгартилиши. Бу жараён натижаси яна матнли файл бўлиб препроцессор томонидан бажарилади.

Компиляция. Бу жараён натижаси машина кодига ўтказилган объектли файл бўлиб, компилятор томонидан бажарилади.

Боғлаш. Бу жараён натижаси тўла машина кодига ўтказилган бажарилувчи файл бўлиб, боғлагич(компоновщик) томонидан бажарилади.

Препроцессор вазифаси дастур матнини препроцессор директивалари асосида ўзгартиришдир. Маалан `define` директиваси дастурда бир жумлани иккинчи жумла билан алмаштириш учун ишлатилади. Бу директива умумий кўриниши қуйидагича:

`#define` <алмаштирувчи ифода> <алмашинувчи ифода>

Бу директива бажарилганда дастур матнидаги алмаштирувчи ифодалар алмашинувчи ифодаларга алмаштирилади.

Агар дастурда қуйидаги матн мавжуд бўлсин:

```
double mix=EULER
d=alfa*EULER
```

Препроцессор бу матнда хар бир EULER константани унинг қиймати билан алмаштиради, ва натижада қуйидаги матн хосил бўлади.

```
double mix=2.718282
d=alfa*2.718282
```

`include` директиваси икки кўринишда ишлатилиши мумкин.

`#include` *`файл номи`* директиваси дастурнинг шу директива урнига қайси матнли файлларни қўшиш кераклигини кўрсатади.

`#include <`*`файл номи`*`>` директиваси дастурга компилятор стандарт библиотекаларига мос келувчи сарлавхали файллар матнларини қўшиш учун мулжаллангандир. Бу файлларда функция прототипи, типлар, ўзгарувчилар, константалар таърифлари ёзилган бўлади. Функция прототипи функция кайтарувчи тип, функция номи ва функцияга узатилуквчи типлардан иборат бўлади. Мисол учун `cos` функцияси прототипи қуйидагича ёзилиши мумкин: `double cos(double ).` Агар функция номидан олдин `void` типи кўрсатилган бўлса бу функция хеч кандай қиймат кайтармаслигини кўрсатади.Шуни таъкидлаш лозимки бу директива дастурга стандарт библиотека қўшилишига олиб келмайди. Стандарт функцияларнинг кодлари боғлаш яъни алоқаларни тахрирлаш босқичида, компиляция босқичидан сўнг амалга оширилади.

Компиляция босқичида синтаксис хатолар текширилади ва дастурда бундай хатолар мавжуд бўлмаса, стандарт функциялар кодларисиз машина кодига утказилади.

Сарлавхали файлларни дастурнинг ихтиёрий жойида улаш мумкин бўлса хам, бу файллар одатда дастур бошида қўшиш лозимдир. Шунинг учун бу файлларга сарлавхали файл (`header file`) номи берилгандир.

Дастур матнини компиляция қилиш. Дастур кодини бажарилувчи файлга ўтказиш учун компиляторлар қўлланилади. Компилятор қандай чақирилади ва унга дастур коди жойлашган жойи ҳақида қандай хабар қилинади, бу конкрет компиляторга боғликдир. Бу маълумотлар компиляторнинг документациясида берилган бўлади.

Дастур коди компиляция қилиниши натижасида объектли файл ҳосил қилинади. Бу файл одатда `.obj` кенгайтмали бўлади. Лекин бу ҳали бажарилувчи файл дегани эмас. Объектли файлни бажарилувчи файлга ўгириш учун йиғувчи дастур қўлланилади.

Йиғувчи дастур ёрдамида бажарилувчи файлни ҳосил қилиш. C++ тилида дастурлар одатда бир ёки бир нечта объектли файллар ёки библиотекаларни компоновка қилиш ёрдамида ҳосил қилинади. `Библиотека` деб бир ёки бир нечта компоновка қилинувчи файллар тўпламига айтилади. C++ нинг барча компиляторлари дастурга қўшиш мумкин бўлган функциялар (ёки процедуралар) ва синфлардан иборат библиотека ҳосил қила олади. `Функция` – бу айрим хизматчи амалларни, масалан икки сонни қўшиб, натижасини экранга чиқаришни бажарувчи дастур блокидир. `Синф` сифатида маълумотлар тўплами ва уларга боғланган функцияларни қараш мумкин. Функциялар ва синфлар ҳақидаги маълумотлар кейинги мавзуларда батафсил берилган.

Демак, бажарилувчи файлни ҳосил қилиш учун қуйида келтирилган амалларни бажариш лозим:

`.cpp` кенгайтмали дастур коди ҳосил қилинади;

Дастур кодини компиляция қилиш орқали `.obj` кенгайтмали объектли файл тузилади;

Бажарилувчи файлни ҳосил қилиш мақсадида `.obj` кенгайтмали файли зарурий библиотекалар орқали компоновка қилинади.

Файллардан матнлар қўшиш. Файлдан матн кушиш учун уч шаклга эга булган # include оператори кулланилади:

```
# include <файл номи>
# include "файл номи"
# include макрос номи
```

Макрос номи #define директиваси орқали киритилган препроцессор идентификатори ёки макрос бўлиши мумкин.

Агар биринчи шакл қўлланилса препроцессор қўшилаётган файлни стандарт библиотекалардан излайди.

Агар иккинчи шакл қўлланилса препроцессор фойдаланувчининг жорий каталогини кўриб чиқади ва бу каталогда файл топилмаса стандарт системали каталогларга мурожаат килади.

Агар программада бир неча функциялардан фойдаланилса ,функциялар таърифи ,танаси билан бирга алохида файлларда сақлаш қулайдир. Хамма функциялар танасига ва main() функцияси танасига чақирилаётган функциялар прототиплари жойлаштирилса, программа танасида функцияларни ихтиёрий жойлаштириш мумкин. Бу холда программа фақат процессор командаларидан хам иборат бўлиши мумкин.

Шартли директивалар. Шартли директива қуйидаги кўринишга эгадир:

```
#if  бутун сонли ифода.
текст_1
#else
текст_2
#endif
#else текст_2 кисми ишлатилиши шарт эмас.
```

Директива бажарилганда #if дан сўнг ёзилган бутун сонли ифода қиймати хисобланади. Агар бу қиймат 0 дан катта бўлса текст_1 компиляция қилинаётган матнга қўшилади, аксинча текст_2 қўшилади. Агар #else директиваси ва текст_2 мавжуд бўлмаса бу директива ўтказиб юборилади.

#ifdef *идентификатор*

директивасида #define директиваси ёрдамида идентификатор аниқланганлиги текширилади. Агар идентификатор аниқланган бўлса текст_1 бажарилади.

#ifndef *идентификатор*

директивасида аксинча шарт рост хисобланади агар идентификатор аниқланмаган бўлса.

Дастурга улаш мўлжалланган файлларнинг хар бирига битта файл уланиш мўлжалланган бўлса, бу файл бир неча марта дастурга уланиб қолади. Бу қайта уланишни олдини олиш учун стандарт файллар юқорида кўрилган директивалар ёрдамида химоя қилингандир. Бу химоя усули қуйидагича бўлиши мумкин.

```
/* filename Номли файл */
/* FILENAME аникланганлигини текшириш */
# indef FILE_NAME
… /* Уланаетган файл тексти
    /* Таъриф
#define FILE_NAME
```

```
#endif
```

Тармоқланувчи шартли директивалар яратиш учун қуйидаги директива киритилган:

`#elif` *`бутун_сонли_ифода`*

Бу директива ишлатилган текст структураси:

```
#if шарт
текст
#elif 1_ифода
1_текст
#elif 2_ифода
2_текст
…
#else
текст
#endif
```

Препроцесссор аввал `#if` директивасидаги шартни текширади. Агар шарт 0 га тенг бўлса 1_ифода хисобланади агар у хам 0 бўлса 2_ифодани хисоблайди ва хоказо.

Агар хамма ифодалар 0 бўлса `else` учун кўрсатилган текст уланади. Агар бирор ифода 0 дан катта бўлса шу директивада кўрсатилган текст уланади.

8.2. Макрослар

Макрос тушунчаси. Макрос таърифига кўра бир жумлани иккинчи жумла билан алмаштиришдир. Макрослар кўпинча макротаъриф деб хам аталади. Энг содда макротаъриф

`# define` *идентификатор алмаштирувчи сатр*.

Бу директива ёрдамида фойдаланувчи асосий типлар учун янги номлар киритиши мумкин.

Масалан: `# define real long double`

Дастур матнида `long double` типидаги ўзгарувчиларни `real` сифатида таърифлаш мумкин.

Параметрли макротаърифлардан фойдаланиш янада кенгроқ имкониятлар яратади:

`# define` ном (параметрлар рўйхати) алмаштирилувчи_қатор

Бу ерда ном – макрос номи.

Параметрлар рўйхати – вергул билан ажратилган идентификаторлар рўйхати.

Макротаърифнинг классик мисоли :

```
# define max (a,b)  (a<b ? b:a)
```

Бу макросдан фойдаланганда компилятор `max (a,b)` ифодани `x<a ? y:x)` ифода билан алмаштиради.

Яна бир мисол:

```
# define ABS(x) (x<0 ? -(x):x)
```

Мисол учун дастурдаги `ABS(E-Z)` ифода `(E-Z<0 ? (E-Z):E-Z)` ифода билан алмаштирилади.

Макросларнинг функциядан фарқи. Функция дастурда битта нусхада бўлса, макрос хосил қилувчи матнлар макрос хар гал чақирилганда дастурга жойлаштирилади. Функция параметрлар спецификациясида кўрсатилган типлар учун ишлатилади ва конкрет типдаги қиймат қайтаради. Макрос хар қандай типдаги параметрлар билан ишлайди. Хосил қилинаётган қиймат типи фақат параметрлар типлари ва ифодаларга боғлиқ.

Макрожойлашлардан тўғри фойдаланиш учун алмаштирилувчи сатрни қавсга олиш фойдалидир.

Функциянинг хақиқий параметрлари бу ифодалардир, макрос аргументлари бўлса вергул билан ажратилган лексемалардир. Аргументларга макро-кенгайтиришлар қўлланмайди.

Шуни кўрсатиш лозимки C++ тилида макрослар жуда кам ишлатилади.

8.3. Хотира синфлари

Автоматик хотира объектлари. *Блок* деб функция танаси ёки фигурали қавслар ичига олинган таърифлар ва операторлар кетма кетлигига айтилади.

Автоматик хотира объектлари фақат ўзи аниқланган блок ичида мавжуд бўлади. Блокдан чиқишда объектлар учун ажратилган хотира қисми бўшатилади, яъни объектлар йўқолади. Шундай қилиб автоматик хотира хар доим ички хотирадир, яъни бу хотирага ўзи аниқланган блокда мурожаат қилиш мумкин. Автоматик хотира объектлари `auto` ёки `register` сўзлари ёрдамида таърифланади.

Агар махсус кўрсатилмаган бўлса ўзгарувчи хар доим автоматик хотира турига тегишли деб хисобланади.

Статик хотира объектлари. *Статик хотира* объектлари блокдан чиқилгандан сўнг хам мавжуд бўлиб колаверади. Статик хотира объектлари `static` хизматчи сўзи ёрдамида таърифланади.

Мисол:

```
#include <iostream>
using namespace std;
void autofunc(void)
{
int K=1;
cout<<" K="<<K;
K++;
return;
}
int main()
{
int i;
for (i=0;i<5;i++)
autofunc();
return 0;
}
```

Бу дастур бажарилиши натижаси:
K=1 K=1 K=1 K=1 K=1

Шу дастурнинг иккинчи кўринишида K ўзгарувчи статик ўзгарувчи сифатида таърифланади:

```
#include <iostream>
using namespace std;
void autofunc(void)
{
static int K=1;
cout<<" K="<<K;
K++;
return;
}
int main()
{
int i;
for (i=0;i<5;i++)
autofunc();
return 0;
}
```

Бу дастур бажарилиши натижаси:
K=1 K=2 K=3 K=4 K=5

Бу мисолда K ўзгарувчи фақат бир марта инициализация қилинади ва унинг қиймати навбатдаги мурожаатгача сақланади.

Глобал объектлар. *Глобал* объектлар деб дастурда хамма блоклар учун умумий бўлган объектларга айтилади. Хар бир блок ичида глобал объектга мурожжат қилиш мумкиндир.

Мисол:

```
#include <iostream>
using namespace std;
int N=5;
void func(void)
{
cout<<"\tN="<<N;
N--;
return;
}

int main()
{
for (int i=0;i<5;i++)
{
func();
N+=2;
}
return 0;
}
```

Дастур бажарилиши натижаси:

N=5 N=6 N=7 N=8 N=9

`N` ўзгарувчиси `main()` ва `func()` функциялари ташқарисида аниқланган ва бу функцияларга нисбатан глобалдир. Хар бир `func()` чақирилганда унинг қиймати 1 га камаяди, асосий дастурда бўлса 2 тага ошади. Натижада `N` қиймати 1 га ошиб боради.

Энди дастурни ўзгартирамиз:

```
#include <iostream>
using namespace std;
int N=5;
void func(void)
{
cout<<"\tN="<<N;
N--;
return;
}

int main()
```

```
{
int N;
for (N=0;N<5;N++)
{
func();
}
return 0;
}
```

Дастур бажарилиши натижаси:

N=5 N=4 N=3 N=2 N=1

`N` ўзгарувчиси `main()` функциясида автоматик ўзгарувчи сифатида таърифланган ва у глобал N ўзгарувчига таъсир қилмайди. Яъни глобал `N` ўзгарувчи `main()` функцисида кўринмайди.

Ташки объектлар. Дастур бир неча матнли файлларда жойлашган бўлиши мумкин. Дастур ўз навбатида функциялардан иборатдир. Хамма функциялар ташқи хисобланади. Хатто хар хил файлларда жойлашган функциялар хам хар хил номларга эга бўлиши лозимдир. Функциялардан ташқари дастурларда ташқи объектлар ўзгарувчилар, кўрсаткичлар, массивлар ишлатилиши мумкин.

Ташки объектлар хамма функцияларда хам кўринмаслиги мумкин. Агар объект файл бошида таърифланган бўлса у шу файлдаги хамма функцияларда кўринади.

Агар ташки объектга шу объект таърифланган блокдан юқорида ёки бошқа файлда жойлашган блокдан мурожжат қилиниши керак бўлса, бу объект `extern` хизматчи сўзи ёрдамида таърифланши лозимдир. Шуни айтиш лозимки `extern` хизматчи сўзи ёрдамида таърифланганда инициализация қилиш ёки чегараларни кўрсатиш мумкин эмас.

```
extern double summa[];
extern char D_phil [];
extern long M;
```

Мисол учун VAL ва SP ўзгарувчилари битта файлда, бу ўзгарувчиларга мурожаат килквчи PUSH, POP и CLEAR функциялари бошқа файлда таърифланган булсин. Бу холда бу файллар орасидаги боғлиқликни таъминлаш учун қуйидаги таърифлар лозимдир:

1 файлда:

```
INT SP = 0;  /* STACK POINTER */
DOUBLE VAL[MAXVAL]; /* VALUE STACK */
```

2 файлда:

```
EXTERN INT SP;
EXTERN DOUBLE VAL[];
```

DOUBLE PUSH(F) ...
DOUBLE POP() ...
CLEAR() ...

Динамик хотира. Динамик хотира бу дастур бажарилиши жараёнида ажратиладиган хотирадир. Динамик хотира `new` оператори орқали ажратилгандан сўнг то `delete` оператори томонидан бушатилмагунча сақланади.

Қуйидаги мисолда якка объектга хотира ажратилади:

```
int*pint=new int(1024)
```

Бу ерда `new` оператори `int` туридаги номсиз объектга хотирани ажратиб беради, уни 1024 қиймати билан номлантиради (инициаллаштиради) ҳамда яратилган объект адресини қайтариб беради. Бу адрес `pint` кўрсаткичига жойлаштирилади. Ушбу номсиз объект устидаги барча хатти-ҳаракатлар шу кўрсаткич билан ишлаш орқали амалга оширилади, чунки динамик объект билан тўғридан-тўғри иш олиб бориш (манипуляциялар ўтказиш) мумкин эмас.

Агар динамик хотира дастур бажарилиши тугагунча бўшатилмаган бўлса, автоматик равишда дастур тугаганда бўшатилади. Шунга қарамай ажратилган хотирани дастурда махсус бўшатиш дастурлашнинг сифатини оширади.

Динамик объект керак бўлмай қолганда, унга ажратилган хотирани тўғридан-тўғри бўшатиш `delete` оператори ёрдамида амалга оширилади:

```
delete pint;
```

Дастур бажарилиш давомида ажратилган хотира қисмига мурожаат имконияти шу қисмга адресловчи кўрсаткичга боғлиқдир. Шундай қилиб, бирор блокда ажратилаётган динамик хотира билан ишлашнинг қуйидаги учта варианти мавжуддир:

- Кўрсаткич автоматик хотира турига кирувчи локал объект. Бу холда ажратилдган хотирага блокдан ташқарида мурожаат қилиб бўлмайди, шунинг учун блокдан чиқишда бу хотирани бўшатиш лозимдир.
- Кўрсаткич автоматик хотира турига кирувчи локал статик объект. Блокда бир марта ажратилган динамик хотирага, блокка хар бир қайта кирилганда кўрсаткич оркали мурожаат қилиш мумкиндир. Хотирани блокдан фойдаланиб бўлгандан сўнг бўшатиш лозимдир.
- Кўрсаткич блокка нисбатан глобал объектдир. Динамик хотирага кўрсаткич кўринувчи хамма блоклардан мурожаат қилиш мумкин.Хотирани фақат ундан фойдаланиб бўлгандан сўнг бўшатиш лозимдир.

Иккинчи вариантга мисол келтирамиз, бу мисолда динамик хотира объекти ички статик кўрсаткич билан боғлиқдир:

```
#include <iostream>
```

```
using namespace std;
void dynamo(void)
{
static char *uc=NULL;
if (uc==NULL)
{
uc=new char(1);
*uc='A';
}
cout<<*uc;
(*uc)++;
return;
};

int main()
{
int i;
for (i=0;i<5;i++)
dynamo();
return 0;
}
```

Дастур бажарилиши натижаси:
A B C D E

Бу дастурнинг камчилиги ажратилган хотира бўшатилмаслигидир.

Кейинги дастурда динамик хотирага кўрсаткич глобал объектдир:

```
#include <iostream>
using namespace std;
char *uk=NULL;
void dynam1(void)
{
cout<<*uk;
(*uk)++;
return;
};

int main()
{
int i;
uk=new char(1);
*uk='A';
for (i=0;i<5;i++)
{
```

```
dynam1();
(*uk)++;
}
delete uk;

return 0;
}
```

Дастур бажарилиши натижаси:

```
A C E G I
```

Динамик объект асосий дастурда яратилиб, `uk` кўрсаткич билан боғликдир. Дастурда бошлангич `'A'` қийматга эга бўлади. Кўрсаткич глобал бўлгани учун динамик объектга `main()` ва `dynam1()` функцияларида мурожаат қилиш мумкин.

Динамик хотирага ажратилгандан сўнг шу объект билан боғлиқ кўрсаткич ташки объект сифатида таърифланган ихтиёрий блокда мурожаат қилиш мумкин.

8.4. Бош функция параметрлри

Main функцияси параметрлари. Хар қандай дастур қуйидагича сарлавхага эга бўлиши лозимдир.

```
int main (int argc, char*argv[ ] ,char*envp[ ])
```

`argv` – сатрларга кўрсаткичлар массиви.

`argc`– `int` типидаги параметр `argv` массивидаги элементлар сонини белгилайди.

`envp`- хар бири мухит ўзгарувчиларининг бирини таърифловчи сатрларга кўрсаткичлар массиви.

Мухит дейилганда `main()` функциясини ишга туширган операцион тизим тушунилади.

Асосий `main()` функцияси параметрлари вазифаси бажарилаётган дастур билан операцион тизим аниқроғи дастурни ишга туширган команда қатори билан алоқани таъминлашдан иборатдир.

Агар `main()` функцияси ичида функцияни ишга туширган команда қаторидаги маълумотга эхтиёж бўлмаса , параметрлар ташлаб кетилади.

`Argv [ ]` массиви индекси 0 дан бошланади. Бу элемент дастурнинг номи билан бирга тўла йулни кўрсатади.

Мисол учун дастур номи `Example` бўлиб у команда қаторидаги қуйидаги сатр орқали ишга туширилаётган бўлсин.

```
C:\ CATALOG\ Example.exe
```

Бу холда argc қиймати 1 га тенг бўлади, Argv[0] эса қуйидаги сатрни кўрсатади:

```
"C:\ CATALOG\ Example.exe"
```

Мисол учун команда каторидан хамма маълумотни экранга сўзма-сўз чиқарувчи дастурни кўрамиз.

```
#include <iostream>
using namespace std;
int main(int argc, char* argv[ ])
{
int i;
for (i=0; i<argc; i++)
cout<<"\n"<<argv[i];
return 0;
}
```

Дастур қуйидаги команда катори оркали ишга туширилсин

```
C:\VVP\ tert66.exe 11 22 33
```

Дастур бажарилиши натижаси:

```
Argv[0] - C:\ VVP\ tert 66.exe
Argv[1] - 11;
Argv[2] - 22;
Argv[3] - 33;
```

Одатда `argv` кўрсаткичлар массиви `NULL` қиймат билан тугайди. Бу хоссадан фойдаланиб юқоридаги массивни биринчи параметрсиз ёзиш мумкин:

```
#include <iostream.h>
int main(int argc, char* argv[ ])
{
int i;
for (i=0; argv[i]!=NULL; i++)
cout<<"\n"<<argv[i];
return 0;

}
```

Учинчи параметр `envp` вазифаси дастурга мухит хакидаги маълумотни узатиш учун ишлатилади. Бу параметр имкониятларини қуйидаги дастурда кўрамиз.

```
#include <iostream>
using namespace std;
int main(int argc, char* argv[ ], char*envp[ ])
```

```
{
int n;
cout<<"\n dastur parametrlar soni"<<argc-1;
for (n=0; n<argc; n++)
cout<<"\n"<<n<<"chi parametr:"<<argv[n];
for (n=0; envp[n]; n++)
cout<<"\n"<<envp[n];
return 0;
}
```

8.5. Параметрлар сони ўзгарувчи булган функциялар.

Ихтиёрий сонли параметрлар. C++ тилида параметрлар сони белгиланмаган функиялар яратиш мумкиндир. Параметрлар сони ва тури функцияга мурожаат қилинганда, хақиқий параметрлар сони ва тури асосида аниқланади. Параметрлар сони ўзгарувчан бўлган функция жуда бўлмаса битта параметрга эга бўлиши керак.

Параметрлар сони ўзгаручан функция таърифи:

```
<тип><исм>(<ошкор параметрлар>, . . . )
```

Вергульдан сўнг уч нуқта кейинги параметрлар сони ва типи ихтиёрий бўлиши мумкинлигини кўрсатади. Рўйхат боши ва охирини белгилаш учун икки усул мавжуд:

1) рўйхат тугаши белгиси маълум;
2) узатилаётган параметрлар сони маълум.

Мисол:

```
#include <iostream>
using namespace std;
int sum(int n, ...)
{
int total = 0;
int arg;
int*pa=&n;
for(int i=0;i<n;i++)
{
pa++;
arg=*pa;
total += arg;
};
return total;
};

int main()
{
int k=9;
```

```
int m=sum(4,1,k,k,1);
cout<<m;
return 0;
}
```

Ихтиёрий сонли параметрлар учун макровоситалар. Параметрлар сони ўзгарувчи функциялар яратиш қулай усули stdarg.h файлида сақланувчи макрокотаърифлардан фойдаланишдир. Бу макротаърифлар ихтиёрий сонли параметрларга эга бўлган функциялар яратишнинг қулай ва содда воситалари бўлиб қуйидаги кўринишга эгадирлар:

void **va_start**(va_list param,охирги аниқ параметр)- рўйхатнинг биринчи элементи билан богланиш ;

<тип> va_arg(va_list param, <тип>) – рўйхатнинг кейинги элементини укиш.

void **va_end**(va_list param) – рўйхат тугагандан сўнг чақириладиган команда.

Бу макросларда биринчи параметрлар stdarg.h файлида аниқланган махсус va_list типига тегишли бўлиши лозимдир. Бу тип оркали кўрсаткич хоссасига эга бўлган объект эълон қилинади. Функция танасида албатта va_list типидаги объект таърифланган бўлиши лозим. Мисол учун

```
va_list factor;
```

Бу объект va_start() функцияси ёрдамида ноъмалум узунликдаги рўйхатнинг биринчи элементи билан боғланади. Бунинг учун макроснинг иккинчи параметри сифатида охирги таърифланган параметр ишлатилади:

```
va_start(factor, охирги_аниқ параметр);
```

Бу функция factor кўрсаткичига охирги аниқ параметри адресини қиймат сифатида беради.

Номаълум рўйхат биринчи параметр типини функцияга узатиш ва кўрсаткични шу параметрга тўғрилаш учун қуйидаги мурожаатдан фойдаланиш лозим:

```
va_arg(factor,<тип>);
```

Кейинги параметр типини функцияга узатиш ва кўрсаткични шу параметрга тўғрилаш учун яна va_arg() макросига мурожаат қилинади:

```
va_arg(factor,<тип_1>);
```

Ихтиёрий сонли параметрли функциядан тўғри қайтиш учун va_end макросидан фойдаланилади:

```
va_end(factor);
```

Бу макрос параметрлар рўйхати тугагандан чақирилади. Шуни кўрсатиш лозимки va_start() макроси чақирилмасдан олдин va_end() макроси қайта чақирилиши мумкин эмас.

Мисол:

```
#include <iostream>
using namespace std;
#include <stdarg.h>
void miniprint(char *format,...)
{
va_list ap;
char *p;
int ii;
double dd;
va_start(ap,format);
for (p=format;*p;p++)
{
if (*p!='%')
{
cout<<*p;
continue;
}
switch(*++p)
{
case 'd':ii=va_arg(ap,int);
cout<<ii;
break;
case 'f':dd=va_arg(ap,double);
cout<<dd;
break;
default:cout<<*p;
}
}
va_end(ap);
}

int main()
{
int k=154;
double e=2.718282;
miniprint("\n k=%d,\t e=%f",k,e);
return 0;
}
```

САВОЛЛАР

1. Компилятор ва препроцессорнинг фарқи нимадан иборат?
2. Шартли директивалар нима учун ишлатилади?
3. Хотира синфларини кўрсатинг.
4. Ташқи объектлар қандай эълон қилинади?

5. Таъриф билан эълон орасида қандай фарқ бор?
6. Динамик хотира билан ишлаш қандай вариантлари мавжуд?
7. Динамик ўзгарувчилар оддий ўзгарувчилардан қандай фарқ қилади?
8. Бош функция параметрлари.
9. Нима учун параметрлари сони ўзгарувчан функция жуда бўлмаса битта
10. параметрга эга бўлиши керак
11. Параметрлар сони ўзгарувчи функциялар яратиш учун қандай макротаърифлардан фойдаланиш қулай?

МИСОЛЛАР

1. Иккита функция яратиб икки файлга ёзинг. Иккала файлни дастурга улаб функцияларни ишлатинг.
2. Икки сон минимумини хисобловчи макрос яратинг ва дастурда фойдаланинг.
3. Берилган кетма кетлик қатъий камаювчи эканлигини текширувчи параметрлар сони ўзгарувчан функция тузинг ва дастурда фойдаланинг.
4. Параметрлари сони ўзгарувчан функция ёрдамида Матрицани векторга кўпайтириг функциясини яратиб дастурда фойдаланинг. Матрица кўрсаткичлар массиви сифатида киритилсин.
5. Хамма хотира синфларига тегишли ўзгарувчилар қатнашган дастур тузинг.

9 боб. Объектга мўлжалланган дастурлаш асослари

9.1. Объектга мўлжалланган ёндошув тарихи

Объектга мўлжалланган ёндошув (ОМЙО) бир кунда ўйлаб топилган эмас. Унинг пайдо бўлиши дастурий таъминотнинг табиий ривожидаги навбатдаги поғона холос. Вақт ўтиши билан қайси услублар ишлаш учун қулай-у, кайсиниси ноқулай эканини аниқлаш осон бўлиб борди. ОМЙО энг муваффақиятли, вақт синовидан ўтган услубларни ўзида самарали мужассам этади.

Дастлаб дастурлаш анчайин бошқотирма ихтиро бўлиб, у дастурчиларга дастурларни коммутация блоки орқали компьютернинг асосий хотирасига тўғридан-тўғри киритиш имконии берди. Дастурлар машина тилларида иккилик тасаввурда ёзилар эди. Дастурларни машина тилида ёзишда тез-тез хатоларга йўл қўйилар эди, энг устига уларни тузилмалаштириш имкони бўлмагани туфайли, кодни кузатиб бориш амалда деярли мумкин бўлмаган хол эди. Бундан ташқари, машина кодларидаги дастур тушуниш учун ғоят мураккаб эди.

Вақт ўтиши билан компьютерлар тобора кенгроқ қўллана бошлади хамда юқорироқ даражадаги процедура тиллари пайдо бўлди. Буларнинг дастлабкиси ФОРТРАН тили эди. Бироқ объектга мўлжалланган ёндошув ривожига асосий таъсирни кейинроқ пайдо бўлган, масалан, АЛГОЛ каби процедура тиллари кўрсатди.

Процедуравий, структуравий ва объектларга мўлжалланган дастурлаш

Шу вақтгача дастурлар берилган маълумотлар устида бирор бир амал бажарувчи процедуралар кетма-кетлигидан иборат эди. Процедура ёки функция ҳам ўзида аниқланган кетма-кет бажарилувчи командалар тўпламидан иборатдир. Бунда берилган маълумотларга мурожаатлар процедураларга ажратилган ҳолда амалга оширилади.

Процедура тиллари дастурчига ахборотга ишлов бериш дастурини пастроқ даражадаги бир нечта процедурага бўлиб ташлаш имконини беради. Пастроқ даражадаги бундай процедуралар дастурнинг умумий тузилмасини белгилаб беради. Ушбу процедураларга изчил мурожаатлар процедуралардан ташкил топган дастурларнинг бажарилишини бошқаради.

Дастурлашнинг бу янги парадигмаси машина тилида дастурлаш парадигмасига нисбатан анча илғор бўлиб, унга тузилмалаштиришнинг асосий воситаси бўлган процедуралар қўшилган эди, Майдароқ функцияларни нафақат тушуниш, балки созлаш хам осонроқ кечади.

Структуравий дастурлашнинг асосий ғояси «бўлакла ва ҳукмронлик қил» принципига бутунлай мос келади. Компьютер дастурини масалалар тўпламидан иборат деб қараймиз. Оддий тавсифлаш учун мураккаб бўлган ихтиёрий масалани бир нечта нисбатан кичикроқ бўлган таркибий масалаларга ажратамиз ва бўлинишни токи масалалар тушуниш учун етарли даражада оддий бўлгунча давом эттирамиз.

Мисол сифатида компания хизматчиларининг ўртача иш ҳақини ҳисоблашни оламиз. Бу масала содда эмас. Уни қатор қисм масалаларга бўламиз:

1. Ҳар бир хизматчининг ойлик маоши қанчалигини аниқлаймиз.
2. Компаниянинг ходимлари сонини аниқлаймиз.
3. Барча иш ҳақларини йиғамиз.
4. Ҳосил бўлган йиғиндини компания ходимлари сонига бўламиз.

Ходимларнинг ойлик маошлари йиғиндисини ҳисоблаш жараёнини ҳам бир неча босқичларга ажратиш мумкин.

2. Ҳар бир ходим ҳақидаги ёзувни ўқиймиз.
3. Иш хақи тўғрисидаги маълумотни оламиз.
4. Иш ҳақи қийматини йиғиндига қўшамиз.
5. Кейинги ходим ҳақидаги ёзувни ўқиймиз.

Ўз навбатида, ҳар бир ходим ҳақидаги ёзувни ўқиш жараёнини ҳам нисбатан кичикроқ қисм операцияларга ажратиш мумкин:

1. Хизматчи файлини очамиз.
2. Керакли ёзувга ўтамиз.
3. Маълумотларни дискдан ўқиймиз.

Структуравий дастурлаш мураккаб масалаларни ечишда етарлича мувафаққиятли услуб бўлиб қолди. Лекин, 1980 – йиллар охирларида Структуравий дастурлашнинг ҳам айрим камчиликлари кўзга ташланди.

Биринчидан, берилган маълумотлар (масалан, ходимлар ҳақидаги ёзув) ва улар устидаги амаллар (излаш, таҳрирлаш) бажарилишини бир бутун тарзда ташқил этилишидек табиий жараён реализация қилинмаган эди. Аксинча, процедуравий дастурлаш берилганлар структурасини бу маълумотлар устида амаллар бажарадиган функцияларга ажратган эди.

Иккинчидан, дастурчилар доимий тарзда эски муаммоларнинг янги ечимларини ихтиро қилар эдилар. Бу ситуация кўпинча велосипедни қайта ихтиро қилиш ҳам деб айтилади. Кўплаб дастурларда такрорланувчи блокларни кўп марталаб қўллаш имкониятига бўлган ҳоҳиш табиийдир. Буни радио ишлаб чиқарувчи томонидан приёмникни йиғишга ўхшатиш мумкин. Конструктор ҳар сафар диод ва транзисторни ихтиро қилмайди. У оддийгина – олдин тайёрланган радио деталларидан фойдаланади холос. Дастурий таъминотни ишлаб чиқувчилар учун эса бундай имконият кўп йиллар мобайнида йўқ эди.

Бошқа томондан, процедурали дастурлаш коддан такроран фойдаланиш имконини чеклаб қўяди. Бунинг устига дастурчилар тез-тез «макарон» дастурлар хам ёзиб туришганки, бу дастурларни бажариш ликопдаги спагетти уйумини ажратишга ўхшаб кетар эди. Ва, нихоят, шу нарса аниқ бўлдики, процедурали дастурлаш усуллари билан дастурларни ишлаб чиқишда диққатни маълумотларга қаратишнинг ўзи муаммоларни келтириб чиқарар экан. Чунки маълумотлар ва процедура ажралган, маълумотлар инкапсулланмаган. Бу нимага олиб келади? Шунга олиб келадики, хар бир процедура маълумотларни нима қилиш кераклигини ва улар қаерда жойлашганини билмоғи лозим бўлади. Агар процедура ўзини ёмон тутса-йу,

маълумотлар устидан нотўғри амалларни бажарса, у маълумотларни бузиб қўйиши мумкин. Хар бир процедура маълумотларга кириш усулларини дастурлаши лозим бўлганлиги туфайли, маълумотлар тақдимотининг ўзгариши дастурнинг ушбу кириш амалга оширилайотган барча ўринларининг ўзгаришига олиб келар эди. Шундай қилиб, хатто энг кичик тўғрилаш хам бутун дастурда қатор ўзгаришлар содир бўлишига олиб келар эди.

Модулли дастурлашда, масалан, Модула2 каби тилда процедурали дастурлашда топилган айрим камчиликларни бартараф этишга уриниб кўрилди. Модулли дастурлаш дастурни бир неча таркибий бўлакларга, ёки, бошқача қилиб айтганда, модулларга бўлиб ташлайди. Агар процедурали дастурлаш маълумотлар ва процессларни бўлиб ташласа, модулли дастурлаш, ундан фарқли ўлароқ, уларни бирлаштиради. Модул маълумотларнинг ўзидан хамда маълумотларга ишлов берадиган процедуралардан иборат. Дастурнинг бошқа қисмларига модулдан фойдаланиш керак бўлиб қолса, улар модул интерфейсига мурожаат этиб қўйақолади. Модуллар барча ички ахборотни дастурнинг бошқа қисмларида йаширади.

Бироқ модулли дастурлаш хам камчиликлардан холи эмас. Модуллар кенгаймас бўлади, бу дегани кодга бевосита киришсиз хамда уни тўғридан-тўғри ўзгартирмай туриб модулни қадамма-қадам узгартириш мумкин эмас. Бундан ташқари, битта модулни ишлаб чиқишда, унинг функцияларини бошқасига ўтказмай (делегат қилмай) туриб бошқасидан фойдаланиб бўлмайди. яна гарчи модулда турни белгилаб бўлса-да, бир модул бошқасида белгиланган турдан фойдалана олмайди.

Модулли ва процедурали дастурлаш тилларида тузилмалаштирилган ва тузилмалаштирилмаган маълумотлар ўз «тур»ига эга. Бироқ турни кенгайтириш усули, агар «агрегатлаш» деб аталувчи усул ёрдамида бошқа турларни яратишни хисобга олмаганда, мавжуд эмас.

Ва, нихоят, модулли дастурлаш - бу яна процедурага мўлжалланган гибридли схема бўлиб, унга амал қилишда дастур бир неча процедураларга бўлинади. Бироқ эндиликда процедуралар ишлов берилмаган маълумотлар устида амалларни бажармайди, балки модулларни бошқаради.

Амалиётга дўстона фойдаланувчи интерфейслари, рамкали ойна, меню ва экранларни тадбиқ этилиши дастурлашда янги услубни келтириб чиқарди. Дастурларни кетма-кет бошидан охиригача эмас, балки унинг алоҳида блоклари бажарилиши талаб қилинадиган бўлди. Бирор бир аниқланган ҳодиса юз берганда дастур унга мос шаклда таъсир кўрсатиши лозим. Масалан, бир кнопка босилганда фақатгина унга бириктирилган амаллар бажарилади. Бундай услубда дастурлар анча интерактив бўлиши лозим. Буни уларни ишлаб чиқишда ҳисобга олиш лозим.

Объектга мўлжалланган дастурлаш бу талабларга тўла жавоб беради. Бунда дастурий компонентларни кўп марталаб қўллаш ва берилганларни манипуляция қилувчи усуллар билан бирлаштириш имконияти мавжуд.

Объектга мўлжалланган дастурлашнинг асосий мақсади берилганлар ва улар устида амал бажарувчи процедураларни ягона объект деб қарашдан иборатдир.

9.2. Объектга мўлжалланган ёндошувнинг афзалликлари ва мақсадлари

ОМЙО дастурий таъминотни ишлаб чиқишда олтида асосий мақсадни кўзлайди. ОМЙО парадигмасига мувофиқ ишлаб чиқилган дастурий таъминот қуйидаги хусусийатларга эга бўлмоғи лозим:

1. табиийлик;
2. ишончлилик;
3. қайта қўлланиш имконияти;
4. кузатиб боришда қулайлик;
5. такомиллашишга қодирлик;
6. янги версийаларни даврий чиқаришнинг қулайлиги.

Табиийлик

ОМЙО ёрдамида табиий дастурий таъминот яратилади. Табиий дастурлар тушунарлироқ бўлади. Дастурлашда «массив» ёки «хотира сохаси» каби атамалардан фойдаланиш ўрнига, ечилаётган масала мансуб бўлган соха атамаларидан фойдаланиш мумкин. Ишлаб чиқилаётган дастурни компьютер тилига мослаш ўрнига, ОМЙО аниқ бир соханинг атамаларидан фойдаланиш имконини беради.

Ишончлилик

Яхши дастурий таъминот бошқа хар қандай махсулотлар, масалан, музлатгич ёки телевизорлар каби ишончли бўлмоғи лозим.

Пухта ишлаб чиқилган ва тартиб билан ёзилган объектга мўлжалланган дастур ишончли бўлади. Объектларнинг модулли табиати дастур қисмларидан бирида, унинг бошқа қисмларига тегмаган холда, ўзгартишлар амалга ошириш имконини беради. Объект тушунчаси туфайли, ахборотга ушбу ахборот керак бўлган шахслар эгалик қилади, масъулият эса берилган функцияларни бажарувчилар зиммасига юклатилади.

Қайта қўлланиш имконияти

Қурувчи уй қуришга киришар эакан, хар гал ғиштларнинг янги турини ихтиро қилмайди. Радиомухандис янги схемани яратишда, хар гал резисторларнинг янги турини ўйлаб топмайди. Унда нима учун дастурчи «Ғилдирак ихтиро қилавериши керак»? Масала ўз ечимини топган экан, бу ечимдан кўп марталаб фойдаланиш лозим.

Малакали ишлаб чиқилган объектга мўлжалланган синфларни бемалол такроран ишлатиш мумкин. Худди модуллар каби, объектларни хам турли дастурларда такроран қўллаш мумкин. Модулли дастурлашдан фарқли ўлароқ, ОМЙО мавжуд объектларни кенгайтириш учун ворисликдан, созланаётган кодни ёзиш учун эса полиморфизмдан фойдаланиш имконини беради.

Кузатиб боришда қулайлик

Дастурий махсулотнинг иш бериш даври унинг ишлаб чиқилиши билан тугамайди. Дастурни ишлатиш жарайонида *кузатиб бориш* деб номланувчи тиргак керак. Дастурга сарфланган 60 фоиздан 80 фоизгача вақт кузатиб боришга кетади. Ишлаб чиқиш эса иш бериш циклининг 20 фоизинигина ташкил этади.

Пухта ишланган объектга мўлжалланган дастур ишлатишда қулай бўлади. Хатони бартараф этиш учун, фақат битта ўринга тўғрилаш киритиш кифоя қилади. Чунки ишлатишдаги ўзгаришлар тиниқ, бошқа барча объектлар такомиллаштириш афзалликларидан автоматик равишда фойдалана бошлайди. Ўзининг табиийлиги туфайли дастур матни бошқа ишлаб чиқувчилар учун тушунарли бўлмоғи лозим.

Кенгайишга қодирлик

Фойдаланувчилар дастурни кузатиб бориш пайтида тез-тез тизимга янги функцияларни қўшишни илтимос қиладилар. Объектлар кутубхонасини тузишнинг ўзида хам ушбу объектларнинг функцияларини кенгайтиришга тўғри келади.

Дастурий таъминот статик (қотиб қолган) эмас. Дастурий таъминот фойдали бўлиб қолиши учун, унинг имкониятларини муттасил кенгайтириб бориш лозим. ОМЙО да дастурни кенгайтириш усуллари кўп. Ворислик, полиморфизм, қайта аниқлаш, вакиллик хамда ишлаб чиқиш жарайонида фойдаланиш мумкин бўлган кўплаб бошқа шаблонлар шулар жумласидандир.

Янги версияларнинг даврий чиқарилиши

Замонавий дастурий махсулотнинг иш бериш даври кўп холларда хафталар билан ўлчанади. ОМЙО туфайли дастурларни ишлаб чиқиш даврини қисқартиришга эришилди, чунки дастурлар анча ишончли бўлиб бормоқда, кенгайиши осонроқ хамда такроран қўлланиши мумкин.

Дастурий таъминотнинг табиийлиги мураккаб тизимларнинг ишлаб чиқилишини осонлаштиради. Хар қандай ишланма хафсала билан ёндошувни талаб қилади, шунинг учун табиийлик дастурий таъминотнинг ишлаб чиқиш даврларини қисқартириш имконини беради, чунки бутун диққат-эътиборни ечилайотган масалага жалб қилдиради.

Дастур қатор объектларга бўлингач, хар бир алохида дастур қисмини бошқалари билан параллел равишда ишлаб чиқиш мумкин бўлади. Бир нечта ишлаб чиқувчи синфларни бир-бирларидан мустақил равишда ишлаб чиқиши мумкин бўлади. Ишлаб чиқишдаги бундай параллеллик ишлаб чиқиш вақтини қисқартиради.

9.3. C++ тили ва объектларга мўлжалланган дастурлаш

C++ тили объектга мўлжалланган дастурлаш принципларини қўллаб қувватлайди. Бу принциплар қуйидагилардир:

- Инкапсуляция
- Меросхўрлик
- Полиморфизм

Инкапсуляция

Инкапсуляциялаш - маълумотларнинг ва шу маълумотлар устида иш олиб борадиган кодларнинг битта объектда бирлаштирилиши. ОМД атамачилигида маълумотлар объект маълумотлари аъзолари (data members)деб, кодлар объектли методлар ёки функция-аъзолар (methods, member functions) деб аталади.

Инкапсуляция ёрдамида берилганларни яшириш таъминланади. Бу жуда яхши характеристика бўлиб фойдаланувчи ўзи ишлатаётган объектнинг ички ишлари ҳақида умуман ўйламайди. Ҳақиқатан ҳам, холодильникни ишлатишда рефрижекторни ишлаш принципини билиш шарт эмас. Яхши ишлаб чиқилган дастур объектини қўллашда унинг ички ўзгарувчиларининг ўзаро муносабати ҳақида қайғуриш зарур эмас.

C++ тилида инкапсуляция принципи синф деб аталувчи ностандарт типларни(фойдаланувчи типларини) ҳосил қилиш орқали ҳимоя қилинади.

Синф - бу махсус турлар бўлиб, ўзида майдон, усуллар ва хоссаларни мужассамлаштиради.

Синф мураккаб структура бўлиб, маълумотлар таърифларидан ташқари, процедура ва функциялар таърифларини ўз ичига олади.

Синф жисмоний мохиятга эга эмас, тузилманинг эълон қилиниши унинг энг яқин аналогиясидир. Синф объектни яратиш учун қўлланган дагина, хотира ажралиб чиқади. Бу жараён хам синф нусхаси (class intsance) ни яратиш деб аталади.

Тўғри аниқланган синф объектини бутун дастурий модул сифатида ишлатиш мумкин. Ҳақиқий синфнинг барча ички ишлари яширин бўлиши лозим. Тўғри аниқланган синфнинг фойдаланувчилари унинг қандай ишлашини билиши шарт эмас, улар синф қандай вазифани бажаришини билсалар етарлидир.

Айнан инкапсуляциялаш туфайли мустақиллик даражаси ортади, чунки ички деталлар интерфейс ортида йаширинган бўлади.

Инкапсуляциялаш модулликнинг объектга мўлжалланган тавсифидир. Инкапсуляциялаш ёрдамида дастурий таъминотни маълум функцияларни бажарувчи модулларга бўлиб ташлаш мумкин. Бу функцияларни амалга ошириш деталлари эса ташқи оламдан йаширин холда бўлади.

Мохиятан *инкапсуляциялаш* атамаси «герметик беркитилган; ташқи таъсирлардан химояланган дастур қисми» деган маънони билдиради.

Агар бирон-бир дастурий объектга инкапсуляциялаш қўлланган бўлса, у холда бу объект қора қути сифатида олиб қаралади. Сиз қора қути нима қилаётганини унинг ташқи интерфейсини кўриб турганингиз учунгина билишингиз мумкин. Қора қути бирон нарса қилишга мажбурлаш учун, унга хабар йубориш керак. Қора қути ичида нима содир бўлаётгани ахамийатли эмас, қора қути йуборилган хабарга адекват (мос равишда) муносабатда бўлиши мухимрокдир.

Интерфейс ташқи олам билан тузилган ўзига хос битим бўлиб, унда ташқи объектлар ушбу объектга қандай талаблар йубориши мумкинлиги кўрсатилган бўлади. Интерфейс - объетни бошқариш пулти.

Шуни таъкидлаб ўтамизки, «Casio» соатининг суюқ кристалли дисплейи ушбу объектнинг маълумотлар аъзоси бўлади, бошқариш тугмачалири эса объектли методлар бўлади. Соат тугмачаларини босиб, дисплейда вақтни ўрнатиш ишларини олиб бориш мумкин, яъни ОМД атамаларини қўллайдиган бўлсак, методлар, маълумотлар аъзоларини ўзгартириб, объект холатини модификация қилади.

Агарда муҳандис ишлаб чиқариш жараёнида резисторни қўлласа, у буни янгидан ихтиро қилмайди, омборга (магазинга) бориб мос параметрларга мувофиқ керакли детални танлайди. Бу ҳолда муҳандис жорий резистор қандай тузилганлигига эътиборини қаратмайди, резистор фақатгина завод характеристикаларига мувофиқ ишласа етарлидир. Айнан шу ташқи конструкцияда қўлланиладиган яширинлик ёки объектни яширинлиги ёки автономлиги хоссаси инкапсуляция дейилади.

Яна бир марта такрорлаш жоизки, резисторни самарали қўллаш учун унинг ишлаш принципи ва ички қурилмалари ҳақидаги маълумотларни билиш умуман шарт эмас. Резисторнинг барча хусусиятлари инкапсуляция қилинган, яъни яширилган. Резистор фақатгина ўз функциясини бажариши етарлидир.

Инкапсуляциялаш нима учун керак?

Инкапсуляциялашдан тўғри фойдаланиш туфайли объектлар билан ўзгартириладиган компонентлар (таркибий қисмлар) дек муомала қилиш мумкин. Бошқа объект сизнинг объектингиздан фойдалана олиши учун, у сизнинг объектингизнинг оммавий интерфейсидан қандай фодаланиш кераклигини билиши кифойа. Бундай мустақиллик учта мухим афзалликка эга.

- Мустақиллик туфайли, объектдан такроран фойдаланиш мумкин. Инкапсуляциялаш пухта амалга оширилган бўлса, объектлар маълум бир программага боғланиб қолган бўлмайди. Улардан имкони бўлган хамма ерда

фойдаланиш мумкин бўлади. Объектдан бошқа бирон ўринда фойдаланиш учун, унинг интерфейсидан фойдаланиб қўйа қолиш кифойа.

- Инкапсуляциялаш туфайли, объектда бошқа объектлар учун кўринмас бўлган ўзгаришларни амалга ошириш мумкин. Агар интерфейс ўзгартирилмаса, барча ўзгаришлар объектдан фойдаланайотганлар учун кўринмас бўлади. Инкапсуляциялаш компонентни яхшилаш, амалга ошириш самарадорлигини таъминлаш, хатоларни бартараф этиш имконини беради, яна буларнинг хаммаси дастурнинг бошқа объектларига таъсир кўрсатмайди. Объектдан фойдаланувчилар уларда амалга оширилайотган барча ўзгаришлардан автоматик тарзда йутадилар.

- Химояланган объектдан фойдаланишда объект ва дастурнинг бошқа қисми ўртасида бирон-бир кўзда тутилмаган ўзаро алоқалар бўлиши мумкин эмас. Агар объект бошқалардан ажратилган бўлса, бу холда у дастурнинг бошқа қисми билан фақат ўз интерфейси орқали алоқага киришиши мумкин.

Шундай қилиб, инкапсуляциялаш ёрдамида модулли дастурларни яратиш мумкин. Самарали инкапсуляциялашнинг учта ўзига хос белгиси қўйидагича:

-абстракция;

-жорий қилишнинг беркитилганлиги;

-масъулиятнинг бўлинганлиги.

9.4. Меросийлик

Ворислик бу мавжуд синфларга янги майдонлар, хоссалар ва усуллар қўшиш ёрдамида янги синфлар ҳосил қилиш имкониятини беради. Янги ҳосил қилинган авлод синф асос яъни аждод синф хоссалари ва усулларига ворислик қилади.

C++ тили ҳам шундай меросхўрликни ҳимоя қилади. Бу янги берилганлар типи (синф), олдиндан мавжуд бўлган синфни кенгайтиришдан ҳосил бўлади. Бунда янги синф олдинги синфнинг меросхўри деб аталади.

`Acme Motors` компанияси инженерлари янги автомобил конструкциясини яратишга аҳд қилишса, улар иккита вариантдан бирини танлашлари лозим. Биринчиси, автомобилнинг конструкциясини бошидан бошлаб янгидан ихтиро қилиш, иккинчиси эса мавжуд `Star` моделини ўзгартиришдир. `Star` модели қарийб идеал, фақатгина унга турбокомпрессор ва олти тезланишли узатма қўшиш лозим. Бош муҳандисиккинчи вариантни танлади. Яъни нолдан бошлаб қуришни эмас, балки `Star` автомобилига озгина ўзгартириш қилиш орқали яратишни танлади. Уни янги имкониятлар билан ривожлантирмоқчи бўлди. Шунинг учун, янги моделни `Quasar` деб номлашни таклиф қилди. `Quasar-Star` моделига янги деталларни қўшиш орқали яратилган.

Ворислик

Ворислик мавжуд бўлган синфнинг таърифи асосидаёқ янги синфни яратиш имконини беради. Янги синф бошқаси асосида яратилгач, унинг таърифи автоматик тарзда мавжуд синфнинг барча хусусийатлари, хулқ-атвори ва жорий қилинишига ворислик қилади. Аввал мавжуд бўлган синф интерфейсининг барча методлари ва хусусийатлари автоматик тарзда ворис интерфейсида пайдо бўлади. Ворислик ворис синфида бирон-бир жихатдан тўғри келмаган хулқ-атворни аввалдан кўра билиш имконини беради. Бундай фойдали хусусият дастурий таъминотни талабларнинг ўзгаришига мослаштириш имконини беради. Агар ўзгартиришлар киритишга эхтиёж туғилса, бу холда эски синф функцияларига ворислик қилувчи янги синф ёзиб қўйа қолинади. Кейин ўзгартирилиши лозим бўлган функцияларга қайтадан таъриф берилади хамда янги функциялар қўшилади. Бундай ўрнига ўрин қўйишнинг мазмуни шундан иборатки, у дастлабки синф таърифини ўзгартирмай туриб, объект ишини ўзгартириш имконини беради. Ахир бу холда қайта тест синовларидан пухта ўтказилган асосий синфларга тегмаса хам бўлади-да.

Агар сиз кўп марталаб қўллаш ёки бошқа бирон мақсадларга кўра ворисликни қўллашга ахд қилсангиз, аввал хар гал қаранг - меросхўр-синф билан ворисликни бераётган синфнинг турлари ўзаро мос келадими.

“has”(эгалик) ва “is a”(бир хиллик) муносабатлари

Одатда синфларни лойиҳалашда савол келиб чиқади, синфларни ўзаро муносабатини қандай қуриш керак бўлади.Иккита оддий синфларга мисол кўрамиз – Square ва Rectangle, улар квадрат ва тўғритўртбурчаклардир. Шуниси тушунарлики бу синфлар ворислик боғланишида бўлади, лекин иккита синфдан қайси бири аждод синф бўлади. Яна иккита синфга мисол – Car ва Person, яъни машина ва инсон. Бу синфлар билан Person_of_Car яъни машина эгаси синфи қандай алоқада бўлиши мумкин? Бу икки синф билан ворислик боғланишида бўлиши мумкинми? Синфларни лойиҳалаш билан боғлиқ бу саволларга жавоб топиш учун шуни назарда тутиш керакки, “мижоз-етказувчи” боғланиши “эга” (“has”) боғланишини, ворислик боғланиши эса “бир хил” (“is a”) боғланиши тушунчаларини ифодалайди. Square ва Rectangle синфлари мисоли тушунарли, ҳар бир объект квадрат тўғритўртбурчакдир, шунинг учун бу синфлар ўртасида ворислик боғланиши ифодаланади, ва Rectangle синфи ота-оналар синфини ифодалайди. Square синфи унинг ўғлидир. Машина эгаси машинага эга ва инсондир. Шунинг учун Person_of_Car синфи Car синфнинг мижози бўлиб ҳисобланади ва Person синфнинг ворисидир.

Ворислик табақаланиши қандайдир маъно касб этиши учун аждодлар устидан қандай амаллар бажарилган бўлса, авлодлар устидан хам шундай амаллар бажарилиш имконияти бўлиши лозим. Меросхўр синфга функцияларни кенгайтириш ва янгиларини қўшиш учун рухсат берилади. Аммо унга функцияларни чиқариб ташлашга рухсат йўқ.

Ворислик ёрдамида қурилган синф методлар ва хусусийатларнинг учта кўринишига эга бўлиши мумкин:

- Ўрнига ўрин қўйиш (алмаштириш): янги синф аждодларининг методи ёки хусусийатини шунчаки ўзлаштириб олмайди, балки унга янги таъриф хам беради;
- Янги: янги синф бутунлай янги методлар ёки хусусийатларни қўшади;
- Рекурсив: янги синф ўз аждодлари методлари ёки хусусийатларини тўғридан-тўғри олиб қўйа қолади.

Объектга мўлжалланган тилларнинг кўпчилиги таърифни маълумот узатилган объектдан қидирадилар. Агар у ердан таъриф топишнинг иложи бўлмаса, бирон таъриф топилмагунча, қидирув табақалар бўйича юқорига кўтарилаверади. Маълумотни бошқариш айнан шундай амалга оширилади хамда айнан шу туфайли ўрингa ўрин қўйиш жарайони иш кўрсатади.

Ворис синфлар химояланган кириш даражасига эга бўлган методлар ва хусусийатларга кириш хуқуқини олишлари мумкин. Базавий синфда фақат авлодлар фойдаланиши мумкинлиги аниқ бўлган методларгагина химояланган кириш даражасини беринг. Бошқа холларда хусусий ёки оммавий кириш даражасидан фойдаланиш лозим. Бундай ёндошув барча синфларга, шу жумладан, тармоқ синфларга хам кириш хуқуқи берилганидан кўра, мустахкамроқ конструкцияни яратиш имконини беради.

Ворислик турлари

Ворислик уч асосий холларда қўлланади:

1.кўп марталаб фойдаланишда;

2.ажралиб туриш учун;

3.турларни алмаштириш учун.

Ворисликнинг айрим турларидан фойдаланиш бошқаларидан кўра афзалроқ хисобланади. Ворислик янги синфга эски синфнинг амалда қўлланишидан кўп марталаб фойдаланиш имконини беради. Кодни қирқиб ташлаш ёки киритиш ўрнига, ворислик кодга автоматик тарзда киришни таъминлайди, яни кодга киришда, у янги синфнинг бир қисмидек олиб қаралади. Кўп марталаб қўллаш учун ворисликдан фойдаланар экансиз, сиз мерос қилиб олинган реализация (жорий қилиниш) билан боғлиқ бўласиз. Ворисликнинг бу турини эхтиёткорлик билан қўллаш лозим. Фарқлаш учун ворислик фақат авлод-синф ва аждод-синф ўртасидаги фарқларни дастурлаш имконини беради. Фарқларни дастурлаш ғоят қудратли воситадир. Кодлаш хажмининг кичиклиги ва коднинг осон бошқарилиши лойиха ишланмасини осонлаштиради. Бу холда код сатрларини камроқ ёзишга тўғри келадики, бу қўшиладиган хатолар миқдорини хам камайтиради.

Алмаштириш имконияти - ОМЙО да мухим тушунчалардан бири. Меросхўр синфга унинг аждоди бўлмиш синфга йубориладиган хабарларни йубориш мумкин бўлгани учун, уларнинг хар иккаласига бир хил муносабатда бўлиш мумкин. Айнан шунинг учун меросхўр синфни яратишда хулқ-атворни чиқариб ташлаш мумкин эмас. Алмаштириш имкониятини

қўллаб, дастурга хар қандай тармоқ турларни қўшиш мумкин. Агар дастурда аждод қўлланган бўлса, бу холда у янги объектлардан қандай фодаланишни билади.

9.5. Полиморфизм

C++ тили бир хил номдаги функция турли объект томонидан ишлатилганда турли амалларни бажариши имкониятини таъминлайди. Бу функция ва синфнинг полиморфлиги деб номланади. Поли – кўп, морфе – шакл деган маънони англатади. Полиморфизм – бу шаклнинг кўп хиллигидир. Бу тушунчалар билан кейинчалик батафсил танишамиз.

Агар инкапсуляциялаш ва ворисликни ОМЙО нинг фойдали воситалари сифатида олиб қараш мумкин бўлса, полиморфизм - энг универсал ва радикал воситадир. Полиморфизм инкапсуляциялаш ва ворислик билан чамбарчас боғлиқ, боз устига, полиморфизмсиз ОМЙО самарали бўлолмайди. Полиморфизм - ОМЙО парадигмасида марказий тушунчадир. Полиморфизмни эгалламай туриб, ОМЙО дан самарали фойдаланиш мумкин эмас.

Полиморфизм шундай холатки, бунда қандайдир битта нарса кўп шаклларга эга бўлади. Дастурлаш тилида «кўп шакллар» дейилганда, битта ном автоматик механизм томонидан танлаб олинган турли кодларнинг номидан иш кўриши тушунилади. Шундай қилиб, полиморфизм ёрдамида битта ном турли хулқ-атворни билдириши мумкин.

Ворислик полиморфизмнинг айрим турларидан фойдаланиш учун зарурдир. Айнан ўриндошлик имконияти мавжуд бўлгани учун, полиморфизмдан фойдаланиш мумкин бўлади. Полиморфизм ёрдамида тизимга тўғри келган пайтда қўшимча функцияларни қўшиш мумкин. Дастурни ёзиш пайтида хатто тахмин қилинмаган функтсионаллик билан янги синфларни қўшиш мумкин, бунинг устига буларнинг хаммасини дастлабки дастурни ўзгартирмай туриб хам амалга ошириш мумкин. янги талабларга осонгина мослаша оладиган дастурий восита деганда, мана шулар тушунилади.

Полиморфизмнинг учта асосий тури мавжуд:

-Қўшилиш полиморфизми

-Параметрик полиморфизм

-Қўшимча юкланиш .

Қўшилиш полиморфизмини баъзида соф полиморфизм деб хам атайдилар. Қўшилиш полиморфизми шунинг билан қизиқарлики, унинг туфайли тармоқ синф нушалари ўзини турлича тутиши мумкин. Қўшилиш полиморфизмидан фойдаланиб, янги тармоқ синфларни киритган холда, тизимнинг хулқ-атворини ўзгартириш мумкин. Унинг бош афзаллиги шундаки, дастлабки дастурни ўзгартирмай туриб, янги хулқ-атворни яратиш мумкин.

Айнан полиморфизм туфайли жорий қилишдан такроран фодаланишни ворислик билан айнанлаштириш керак эмас. Бунинг ўрнига ворисликдан аввалам бор ўзаро алмашиниш муносабатлари ёрдамида

полиморф хулқ-атворга эришиш учун фойдаланиш лозим. Агар ўзаро алмашиниш муносбатлари тўғри белгиланса, бунинг ортидан албатта такроран қўллаш чиқиб келади. Қўшилиш полиморфизмидан фойдаланиб, базавий синфдан, хар қандай авлоддан, шунингдек базавий синф қўллайдиган методлардан такроран фойдаланиш мумкин.

Параметрик полиморфизмдан фойдаланиб, турдош методлар ва турдош (универсал) турлар яратиш мумкин. Турдош методлар ва турлар далилларнинг кўплаб турлари билан ишлай оладиган дастурни ёзиш имконини беради. Агар қўшилиш полиморфизмидан фойдаланиш объектни идрок этишга таъсир кўрсатса, параметрик полиморфизмдан фойдаланиш қўлланайотган методларга таъсир кўрсатади. Парметрик полиморфизм ёрдамида, параметр турини бажарилиш вақтигача эълон қилмай туриб, турдош методлар яратиш мумкин. Методларнинг параметрик параметрлари бўлганидек, турларнинг ўзи хам параметрик бўлиши мумкин. Бироқ полиморфизмнинг бундай тури барча тилларда хам учрайвермайди (C++да мавжуд).

Қўшимча юкланиш ёрдамида битта ном турлича методларни билдириши мумкин. Бунда методлар фақат миқдорлари ва параметр турлари билан фарқланади. Метод ўз далиллари (аргументлари) га боғлиқ бўлмаганда, ортиқча юкланиш фойдалидир. Метод ўзига хос параметрлар турлари билан чекланмайди, балки хар хил турдаги параметрларга нисбатан хам қўлланади. Масалан `max` методини кўриб чиқайлик. Максимал - турдош тушунча бўлиб, у иккита муаййан параметрларни қабул қилиб, уларнинг қайси бири каттароқ эканини маълум қилади. Таъриф бутун сонлар ёки сузувчи нуқтали сонлар қийосланишига қараб ўзгармайди.

Полиморфизмдан самарали фойдаланиш сари қўйилган биринчи қадам бу инкапсуляциялаш ва ворисликдан самарали фойдаланишдир. Инкапсуллашсиз дастур осонгина синфларнинг жорий қилинишига боғлиқ бўлиб қолиши мумкин. Агар дастур синфларнинг жорий қилиниш аспектларидан бирига боғлиқ бўлиб қолса, тармоқ синфда бу жорийни тўғрилаш мумкин бўлмайди.

Ворислик - қўшилиш полиморфизмининг мухим таркибий қисми. Хамма вақт базавий синфга имкон даражада яқинлаштирилган даражада дастурлашга уринган холда, ўринбосарлик муносабатларини ўрнатишга харакат қилиш керак. Бундай усул дастурда ишлов берилайотган объектлар турлари миқдорини оширади.

Пухта ўйлаб ишлаб чиқилган табақаланиш ўринбосарлик муносабатларини ўрнатишга ёрдам беради. Умумий қисмларни абстракт синфларга олиб чиқиш керак хамда объектларни шундай дастурлаш керакки, бунда объектларнинг ихтисослаштирилган нушалари эмас, балки уларнинг ўзлари дастурлаштирилсин. Бу кейинчалик хар қандай ворис синфни дастурда қўллаш имконини беради.

Агар тил воситалари билан интерфейс ва жорий қилинишни тўлиқ ажратиш мумкин бўлса, у холда одатда мана шу воситалардан фойдаланиш керак, ворисликдан эмас. Интерфейс ва жорий қилинишни аниқ ажратиб,

ўринбосарлик имкониятларини ошириш ва шунинг билан полиморфизмдан фойдаланишнинг янги имкониятларини очиб бериш мумкин.

Бироқ кўп ўринларда тажрибасиз лойихачилар полиморфизмни кучайтириш мақсадида хулқ-атворни жуда баланд табақавий даражага олиб чиқишга уринадилар. Бу холда хар қандай авлод хам бу хулқ-атворни ушлаб тура олади. Шуни эсдан чиқармаслик керакки, авлодлар ўз аждодларининг функцияларини чиқариб ташлай олмайдилар. Дастурни янада полиморф қилиш мақсадида пухта режалаштирилган ворислик табақаларини бузиш йарамайди.

Акселераторни босилишида `Star` моделига нисбатан янги яратилган `Quasar` моделида бошқачароқ амаллар бажарилиши мумкин. `Quasar` моделида двигателга ёқилғини сепувчи инжектор системаси ва `Star` моделидаги корбюратор ўрнига турбокомпрессор ўрнатилган бўлиши мумкин. Лекин фойдаланувчи бу фарқларни билиши шарт эмас. У рулга ўтиргач оддийгина акселаторни босади ва автомобилнинг мос реакциясини кутади.

9.6. Компонентлар

C++ компонентлар(components) тушунчасини хам киритади. Компонентлар - махсус синфлар бўлиб, уларнинг хусусиятлари объектлар атрибутларини ташкил қилади, уларнинг методлари эса компонентли синфларнинг тегишли нушалари уцидаги операцияларни амалга оширади. Метод тушунчаси одатда компонентли синфлар таркибида қўлланади ва ташқи томондан оддий синфнинг функция-аъзо атамасидан фарқ қилмайди. C++ тили компонентларнинг тури ва функционал хулқ-атворини нафақат методлар ёрдамида, балки компонентлар синфларигагина хос бўлган хусусиятлар воситасида хам манипуляция қилиш имконини беради.

- Компонентлар хусусиятлари (propertion) бу маъулмотлар аъзоларининг кенгайишидир. Гарчи улар маълумотларни ўз холларича сақламасалар-да, бироқ объект маълумотлари аъзоларига кириш хуқуқини таъминлайдилар. Хусусиятларни эълон қилишда C++ _property калит-сўздан фойдаланади. Воқеалар (events) ёрдамида компонента ўзига қандайдир аввалдан белгиланган таъсир кўрсатилганини фойдаланувчига маълум қилади. C++ тилида ишлаб чиқилаётган дастурлардаги методлар асосан маълум воқеаларнинг юзага келишида дастур реакциясини уларга нисабатан ишга соладиган воқеаларнинг қайта ишлагичлари *(events handlers)*да қўлланади. Windows операция тизимидаги воқеалар ва маълумотлардаги қандайдир ўхшашликни пайқаб олиш қийин эмас. Бу ердаги оддий типик воқеалар клавиатурадаги тугмача ёки клавишаларни босишдан иборат. Компоненталар ўз хусусиятлари, методлари ва воқеаларини инкапсулалайдилар.

Бир қарашда, компоненталар C++тилининг бошқа объектли синфларидан, бир қатор хусусиятларни хисобга олмаганда, хеч бир фарқ қилмайди. Бу хусусиятлар орасида хозирча қуйидагиларни кўрсатиб ўтамиз:

- Компоненталарнинг кўпчилиги интерфейснинг фойдаланувчи билан бошқариш элементи бўлиб, уларнинг айримлари ғоят мураккаб хулқ-атворга эга.
- Барча компоненталар битта умумий аждод-синф (TСомпонент) нинг бевосита ёки билвосита авлодларидирлар.
- Компоненталар одатда бевосита қўлланади, яъни уларнинг хусусиятлари устида иш олиб борилади (манипуляциялар ўтказилади); уларнинг ўзлари янги тармоқ синфлар (синфчалар) қуриш учун базавий синфлар сифатида хизмат қила олмайди.
- Компоненталар фақат **new** оператори ёрдамида уюм (*heap*) нинг динамик хотирасида жойлаштирилади, оддий синфлар объектларида бўлганидек, стекда эмас.
- Копмоненталар хусусиятлари RTTI - динамик турлар идентификациясини ўз ичига олади.

10 боб. Таянч синфлар

10.1. Синф таърифи

Синф-структура тушунчаси кенгайтмаси сифатида. Синфларни энг содда холда қуйидагича тасвирлаш мумкин:

Синф-калити Синф-сони {компоненталар рўйхати}

Синф компоненталари содда холда типланган маълумотлар ва функциялардан иборат бўлади. Фигурали кавсларга олинган компоненталар рўйхати Синф танаси деб аталади. Синфга тегишли функциялар компонента-функциялар ёки синф функциялари деб аталади.

Синф калити сифатида `Struct` хизматчи сўзи ишлатилиши мумкин. Масалан қуйидаги конструкция комплекс сон синфини киритади.

```
struct complex
{
double real;
double imag;
void define (double re=0.0, double im=0.0)
{
real=re; imag=im;
}
void display (void)
{
cout<=”real=”<<real;
cout<=”imag=”<<imag;
}
};
```

Структурадан бу синфнинг фарқи шуки компонента маълумотлардан (`real, imag`) ташқари иккита компонента функция (`define() ва display ()`) киритилган.

Бу киритилган синф ўзгарувчилар типи деб каралиши мумкин. Бу типлар ердамида конкрет объектларни қуйидагича тасвирлаш мумкин:

Мисол учун:

```
complex x,y;
complex dim[8];
```

Синфга тегишли объектлар қуйидагича тасвирланади;

Синф-номи.объект-номи

Дастурда объект компонентасига қуйидагича мурожат килиш мумкин:

Синф-номи.объект-номи :: компонента-номи ёки соддарок холда

Объект-номи. Элемент-номи

Мисол учун:

```
x.real=1.24;
x.imag=0.0;
dim[3]. Real=0.25;
dim[3]. Imag=0.0;
```

Синфга тегишли функцияларга қуйидагича мурожат килинади:
объект-номи. функция-номи

Мисол учун:

`X. define(0.9)` **(**Бу холда real=0.9 ва imag=0.0)

`X. define(4.3,20.0)` **(**Бу холда комплекс сон 4.3+i*20.0)

`Display` функцияси экранда комплекс сон қийматларини тасвирлайди.

Компанента ўзгарувчилар ва компанента функциялар.Синф компанента ўзгарувчилари сифатида ўзгарувчилар , массивлар, кўрсаткичлар ишлатилиши мумкин. Элементлар таърифланганда инициализация килиш мумкин эмас. Бунинг сабаби шуки синф учун хотирадан жой ажратилмайди. Компанента элементларига компанента функциялар оркали мурожат қилинганда фақат номлари ишлатилади Синфдан ташқарида синф элементларига эмас объект элементларига мурожат килиш мумкин . Бу мурожат икки хил бўлиши мумкиндир.

Объект- номи . Элемент - номи.

Синф элементлари синфга тегишли функцияларида ишлатилишидан олдин таърифланган бўлиши шарт эмас. Худди шундай бир функциядан хали таърифи берилмаган иккинчи функцияга мурожаат килиш мумкин.

Компоненталарга мурожаат хукуклари. Компоненталарга мурожаат хукуки мурожаат спецификаторлари ёрдамида бошкарилади. Бу спецификаторлар:

Protected – химояланган;
Private – хусусий;
Public – умумий;

Химояланган компаненталардан синфлар иерархияси қурилганда фойдаланилади. Оддий холда `Protected` спецификатори `Private` спецификаторига эквивалентдир. Умумий яъни `Public` типидаги компоненталарга дастурнинг ихтиёрий жойида мурожаат килиниши мумкин.

Хусусий яъни `Private` типидаги компоненталарга синф ташқарисидан мурожаат қилиш мумкин эмас. Агар синфлар `Struct` хизматчи сўзи билан киритилган бўлса, унинг хамма компоненталари умумий `Public` бўлади,

лекин бу хукукни мурожаат спецификаторлари ёрдамида ўзгартириш мумкин.

Агар синф `Class` хизматчи сўзи оркали таърифланган бўлса, унинг хамма компоненталари хусусий бўлади. Лекин бу хукукни мурожаат спецификаторлари ёрдамида ўзгартириш мумкиндир.

Бу спецификатор ёрдамида синфлар умумий холда куйидагича таърифланади:

```
class class_name
{
int data_member; // Маълумот-элемент
void show_member(int); // Функция-элемент
};
```

Синф таърифлангандан сўнг, шу синф типидаги ўзгарувчиларни(объектларни) куйидагича таърифлаш мумкин:

```
class_name object_one, object_two, object_three;
```

Куйидаги мисолда *employee*, синфи киритилгандир:

```
class employee
{
   public:
   long employee_id;
   float salary;
   void show_employee(void)
{
  cout<<"Nomer: "<<employee_id<<endl;
  cout<<"Maosh: "<<salary<<endl;
};
};
```

Бу синф икки ўзгарувчи ва битта функция-элементга эга.

Куйидаги дастур икки `employee` объектини яратади. Нукта оператордан фойдаланиб маълумот элементларга киймат берилади сўнгра *show_employee* элементидапн фойдаланиб хизматчи хакидаги маълумот экранга чикарилади:

```
#include <iostream>
using namespace std;
class employee
{
   public:
   long employee_id;
   float salary;
   void show_employee(void)
```

```
{
  cout<<"Nomer: "<<employee_id<<endl;
  cout<<"Maosh: "<<salary<<endl;
};
};
int main()
{
employee worker, boss;
worker.employee_id = 12345;
worker.salary = 25000;
boss.employee_id = 101;
boss.salary = 101101.00;
cout<<"\n"<<"ishchi"<<endl;
worker.show_employee();
cout<<"\n"<<"boss"<<endl;
boss.show_employee();
return 0;
}
```

10.2. Синф компонента функциялари

Компонента функция таърифи. Компонента функция албатта синф танасида таърифланган бўлиши лозим. Глобал функциялардан фарқли компонента функция синфнинг хамма компоненталарига мурожат қилиши мумкин. Функциянинг фақат прототипи эмас тўла таърифи синф танасида жойлашган бўлса, бу функция жойлаштирувчи (`inline`) функция хисобланади. Маълумки `inline` функцияларда цикллар, калит бўйича ўтиш оператори ишлатилиши мумкин эмас. Бундан ташқари бундай функциялар рекурсив функция бўлолмайди. Бу чегараларни енгиш учун синф танасига фақат функция прототипи жойлаштирилиб, функциянинг тўла таърифи синф ташқарисида дастурга кирувчи бошқа функциялар билан бирга берилади. Компонента функцияни синф ташқарисида таърифланганда, қайси синфга тегишли эканлигини қуйидаги шаклда кўрсатилади:

Синф-номи :: Компонента функция-номи

Синф танасига компонента функция прототипи қуйидаги шаклда жойлаштирилади:

Тип функция-номи(формал-параметрлар-таърифи)

Синф ташқарисида функция қуйидаги шаклда таърифланади:

Тип синф-номи :: функция-номи(формал-параметрлар-спецификацияси) { функция танаси };

Олдинги мисолдаги *employee* синфида функция синф ичида таърифланган. Бундай функция жойланувчи *(inline) функция* деб қаралади.

Функцияни синф ташқарисида таърифлаб синф ичига функция прототипини жойлаштириш мумкин. Синф таърифи бу холда қуйидаги кўринишда бўлади:

```
class employee
{
   public:
   long employee_id;
   float salary;
   void show_employee(void);
};
```

Хар хил функциялар бир хил номли функциялардан фойдаланиши мумкин бўлгани учун функция номи синф номи ва глобал рухсат оператори белгиси (::) қўйилиши лозим.

```
void employee::show_employee(void)
{
  cout<<"Nomer: "<<employee_id<<endl;
  cout<<"Maosh: "<<salary<<endl;
};
```

Функция синф ташқарисида таърифланган бўлса уларни `inline` функция сифатида қараш учун функция таърифида `inline` сўзи аниқ кўрсатилган бўлиши керак.

Қуйидаги дастур *show_employee* функцияси таърифини синф ташқарисига жойлаштиради ва `inline` сўзи аник кўрсатилади:

```
#include <iostream>
using namespace std;
class employee
{
   public:
   long employee_id;
   float salary;
   void show_employee(void);
};
inline void employee::show_employee(void)
{
  cout<<"Nomer: "<<employee_id<<endl;
  cout<<"Maosh: "<<salary<<endl;
};

int main()
{

employee worker, boss;
worker.employee_id = 12345;
worker.salary = 25000;
```

```
boss.employee_id = 101;
boss.salary = 101101.00;
cout<<"\n"<<"ishchi"<<endl;
worker.show_employee();
cout<<"\n"<<"boss"<<endl;
boss.show_employee();
return 0;
}
```

10.3. Конструктор ва деструктор

Конструкторлар. Конструкторлар бу синф компонента функциялари бўлиб, объектларни автоматик инициализация қилиш учун ишлатилади.

Конструкторлар кўриниши қуйидагича бўлиши мумкин:

Синф номи (формал параметрлар рўйхати)
{конструктор танаси}

Бу компонента функция номи синф номи билан бир хил бўлиши лозим.

Мисол учун complex синфи учун конструкторни қуйидагича киритиш мумкин :

```
complex (double re = 0.0; double im = 0.0 )
 {real=re; imag=im;}
```

Конструкторлар учун кайтарилувчи типлар, хатто void типи хам кўрсатилмайди. Дастурчи томонидан кўрсатилмаган холда хам объект яратилганда конструктор автоматик равишда чақирилади.

Масалан объект `complex cc;` шаклида аниқланган бўлса, конструктор автоматик чақирилиб

`real` ва `imag` параметрлари автоматик равишда 0.0 қийматларига эга бўлади.

Кўзда тутилган холда параметрсиз конструктор ва қуйидаги типдаги нусха олиш конструкторлари яратилади: T :: T (const T&)

Мисол учун

```
class F
{...
          public : F(const T&)
     ...
 }
```

Синфда бир нечта конструкторлар бўлиши мумкин, лекин уларнинг фақат биттасида параметрлар қийматлари олдиндан кўрсатилган бўлиши керак.

Конструктор адресини хисоблаш мумкин эмас. Конструктор параметри сифатида ўз синфининг номини ишлатиш мумкин эмас, лекин бу номга кўрсаткичдан фойдаланиш мумкин.

Конструкторни оддий компонента функция сифатида чакириб бўлмайди. Конструкторни икки хил шаклда чақириш мумкин :

Синф_номи .Объект_номи (конструктор_хақиқий_параметлари)
Синф_номи (конструктор_хақиқий_параметлари)

Биринчи шакл ишлатилганда хақиқий параметрлар рўйхати буш булмаслиги лозим. Бу шаклдан янги объект таърифланганда фойдаланилади:

```
complex SS(10.3; 0.22)
// real=10.3; SS.imag= 0.22;
complex EE (2.3)
// EE . real= 2.3;
EE.imag= 0.0;
complex D()  // хато
```

Конструкторни иккинчи шаклда чақириш номсиз объект яратилишига олиб келади. Бу номсиз объектдан ифодаларда фойдаланиш мумкин.

Мисол учун :

```
complex ZZ= complex (4.0;5.0);
```

Бу таъриф оркали `ZZ` объект яратилиб, унга номсиз объект қийматлари(`real= 4.0; imag= 5.0`) берилади;

Конструктор номи синф номи билан бир хил бўлиши лозимдир. Мисол учун сиз `employee` синфдан фойдалансангиз, конструктор хам employee номга эга бўлади. Агар дастурда конструктор таърифи берилган булса объект яратилганда автоматик чақирилади. Қуйидаги дастурда `employee` номли синф киритилгандир:

```
class employee
{
public:
employee(long, float);
void show_employee(void);
private:
long employee_id;
float salary;
};
```

Конструктор таърифи:

```
employee::employee(long empl_id, float sal)
{
employee_id = empl_id;
if (salary < 50000.0)
salary = sal;
else
salary = 0.0;
}
```

Шу синфдан фойдаланилган дастур:

```
#include <iostream>
using namespace std;
class employee
{
public:
employee(long, float);
void show_employee(void);
private:
long employee_id;
float salary;
};
employee::employee(long empl_id, float sal)
{
employee_id = empl_id;
if (salary < 50000.0)
salary = sal;
else
salary = 0.0;
}
void employee::show_employee(void)
{
cout << "Nomer: " << employee_id << endl;
cout << "Maosh: " << salary << endl;
}
int main()
{
employee worker(101, 10101.0);
cout<<"ishchi"<<endl;
worker.show_employee();
return 0;
}
```

Конструктордан фойдаланиб объект таърифиланганда параметр узатиш мумкин: `employee worker(101, 10101.0);`

Агар дастурда `employee` типидаги объектлар мавжуд бўлса хар бирини қуйидагича инициализация қилиш мумкин

```
employee worker(101, 10101.0);
employee secretary(57, 20000.0);
employee manager(1022, 30000.0);
```

Сатрли майдонга мисол. Кейинги мисолда сатрли майдон `string` типидаги ўзгарувчи сифатида берилади.

```
#include <iostream>
#include <string>

using namespace std;

class employee
{
public:
employee(string , long, float);
void show_employee(void);
int change_salary(float) ;
long get_id(void);
private:
string name;
long employee_id;
float salary;
};
employee::employee(string name, long employee_id,
float salary)
{
employee::name= name;
employee::employee_id = employee_id;
if (salary < 50000.0)
employee::salary = salary;
else
employee::salary = 0.0;
}
void employee::show_employee(void)
{
cout << "Ism: " << name << endl;
cout << "Nomer: " << employee_id << endl;
cout << "Maosh: " << salary << endl;
}
int main()
{
```

```
employee worker("Happy Jamsa", 101, 10101.0);
worker.show_employee();
return 0;
}
```

Натижа:

```
Ism: Happy Jamsa
Nomer: 101
Maosh: 10101
```

Конструкторлар ва кўзда тутилган қийматлар. Конструкторларда кўзда тутилган қийматлардан хам фойдаланиш мумкиндир. Мисол учун қуйидаги конструктор `employee` маоши қийматини дастурда кўрсатилмаган бўлса 10000.0 тенг қилиб олади:

```
employee::employee(long empl_id, float sal = 100.00)
{
employee_id = empl_id;
if (salary < 50000.0)
salary = sal;
else
salary = 0.0;
}
```

Конструкторларни қўшимча юклаш. C++ тилида конструкторларни хам қўшимча юклаш мумкиндир. Қуйидаги дастурда конструктор `employee` қўшимча юклангандир. Биринчи конструктор, дастур хизматчи, номер ва ойлиги кўрсатилишини талаб қилади. Иккинчи конструктор ойликни киритилишини сўрайди. Синф таърифи ичида иккала конструктор прототипи кўрсатилиши лозим:

```
#include <iostream>
using namespace std;
class employee
{
public:
employee(long, float);
employee(long);
void show_employee(void);
private:
long employee_id;
float salary;
};
employee::employee(long employee_id, float salary)
{
```

```
employee::employee_id = employee_id;
if (salary < 50000.0) employee::salary = salary;
else
employee::salary = 0.0;
}
employee::employee(long employee_id)
{
employee::employee_id = employee_id;
do
{
cout << "Maosh kiriting $50000 dan kichik: ";
cin >> employee::salary;
}
while (salary >= 50000.0);
}
void employee::show_employee(void)
{
cout << "Nomer: " << employee_id << endl;
cout << "Maosh: " << salary << endl;
}
int main()
{
cout<<"ishchi"<<endl;
employee worker(101, 10101.0);
worker.show_employee();
cout<<"manager"<<endl;
employee manager(102);
manager.show_employee();
return 0;
}
```

Объектлар массивлари. Объектлар массиви таърифлаш учун синф кўзда тутилган (параметрсиз) конструкторга эга бўлиши керак.

Объектлар массиви кўзда тутилган конструктор томонидан, ёки хар бир элемент учун конструктор чақириш йўли билан инициализация қилиниши мумкин.

`class complex a[20];` //кўзда тутилган параметрсиз конструкторни чақириш

`class complex b[2]={complex(10),complex (100)};`
//ошкор чақириш

Қуйидаги мисолда *player*, синфи киритилади. Дастурда синф функцияси *show_player* ва конструктор ташқарисида таърифланади. Сўнгра *player* типидаги икки массив яратилиб, хар бири хакидаги маълумот экранга чикарилади

```
#include <iostream>
#include <string>
using namespace std;
class player
{
public:
player();
player (string name,int weight, int age);
void show_player (void);
private:
string name;
int weight;
int age;
};
player::player()
{
name="";
weight = 0;
age = 0;
};
player::player(string name,int weight, int age)
{
player::name=name;
player::weight = weight;
player::age = age;
};
void player::show_player (void)
{
cout<<"Ism: " << name << endl;
cout<<"Vazn: " << weight << endl;
cout<<"Yosh: " << age << endl;
}
class array_player
{ public:
void show_array(player a[],int n)
{ for(int i=0;i<n;i++)
{a[i].show_player();cout<<endl;}}
void input_array(player a[],int n)
{string name;int weight,age;
for(int i=0;i<n;i++)
{cin>>name>>weight>>age;
a[i]=player(name,weight,age);
}
}
};
```

```
int main()
{array_player arr;
Player happy[]={player("Olimov",58,24),
player("Alimov",72,35)};
arr.show_array(happy,2);
player matt[2];
arr.input_array(matt,2);
arr.show_array(matt,2);
return 0;
}
```

Инициализаторлар рўйхати. Конструктор ёрдамида объект маълумотларни инициялизациялашни иккита усули мавжуд.

Биринчи усулда параметрлар қийматлари конструктор танасига узатилади. Икинчи усулда эса ушбу синфдаги инициализаторлар рўйхатидан фойдаланиш назарда тутилган. Бу рўйхат параметрлар рўйхати ва конструктор танаси орасига жойлашади. Рўйхатдаги хар бир инициализатор конкрет аниқ компонентага боғлиқ ва қуйидаги кўринишга эга:

<ном> (<ифода>)

Мисол:

```
#include <iostream>
using namespace std;
class A
{
int i;
char c;
public:
A(int ii, char t):i(ii){c=t;};
void show()
{
cout<<"i="<<i<<"\nc="<<c;
}
};
int main()
{
A ss(5,'a');
ss.show();
return 0;
}
```

Деструкторлар. Синфнинг бирор объекти учун ажратилган хотира объект йукотилгандан сўнг бўшатилиши лозимдир.

Синфларнинг махсус компоненталари деструкторлар, бу вазифани автоматик бажариш имконини яратади.

Деструкторни стандарт шакли қуйидагича :

~ синф_номи ( ) {деструктор танаси}

Деструктор параметри ёки кайтарилувчи қийматга эга бўлиши мумкин эмас (хатто `void` типидаги).

Агар синфда ошкор деструктор мавжуд бўлмаса, кўзда тутилган деструктор чақирилади.

Дастур объектни ўчирганда деструктор автоматик чақирилади.

Мисол:

```
#include <iostream>
using namespace std;
class Person
{
public:
Person ()
{
cout<<"Yaratidi"<<endl;
}
~Person ()
{
cout<<"O'chirldi"<<endl;
}
};
int main()
{
{
Person work;
}
int kk;cin>>kk;
return 0;
}
```

Натижа

```
Yaratidi
O'chirldi
```

10.4. Статик элементлар ва функциялар

Маълумотлар элементидан биргаликда фойдаланиш. Одатда, маълум синф объектлари яратилаётганда, ҳар бир объект ўз-ўзининг маълумотлар элементлари тўпламини олади. Бироқ шундай ҳоллар ҳам юзага келадики, унда бир хил синфлар объектларига бир ёки бир нечта маълумотлар элементларидан (статик маълумотлар элементларидан) биргаликда фойдаланиш керак бўлиб қолади. Бундай ҳолларда маълумотлар элементлари

умумий ёки жузъий деб эълон қилинади, кейин эса тур олдидан, қуйида кўрсатилганидек, `static` калит-сўз келади:

```
private;
static int shared_value;
```

Синф эълон қилингач, элементни синфдан ташқаридаги глобал ўзгарувчи сифатида эълон қилиш керак. Бу қуйида шундай кўрсатилган:

```
int class_name::shared_value;
```

Навбатдаги дастур `book_series` синфини аниқлайди. Бу синф (серия)нинг барча объектлари (китоблари) учун бир хилда бўлган `page_count` элементидан биргаликда фойдаланади. Агар дастур ушбу элемент қийматини ўзгартирса, бу ўзгариш шу ондаёқ барча синф объектларида ўз аксини топади:

```
#include <iostream>
using namespace std;
    class book_series
    {
    public:
       book_series(float);
       void show_book(void);
       void set_pages(int) ;
    private:
       static int page_count;
       float price;
    };
    int book_series::page_count;
    void book_series::set_pages(int pages)
    {
       page_count = pages;
    }
    book_series::book_series(float price)
    {
       book_series::price = price;
    }
    void book_series:: show_book (void)
    {
       cout << "Narx: " << price << endl;
       cout << "Betlar: " << page_count << endl;
    }
    int main()
    {
       book_series programming(213.95);
       book_series word(19.95);
```

```
    word.set_pages(256);
    programming.show_book ();
    word.show_book() ;
    cout << endl << "page_count ning o'zgarishi " <<
endl;
    programming.set_pages(512);
    programming.show_book();
    word.show_book();
return 0;
}
```

Кўриниб турганидек, синф *page_count* ни *static int* сифатида эълон қилади. Синфни аниқлагандан сўнг, дастур шу вақтнинг ўзида `page_count` элементини глобал ўзгарувчи сифатида эълон қилади. Дастур `page_count` элементини ўзгартирганда, ўзгариш шу вақтнинг ўзидаёқ `book_series` синфининг барча объектларида намоён бўлади.

Агар объектлар мавжуд бўлмаса, public static атрибутли элементлардан фойдаланиш. Синф элементини статис каби эълон қилишда бу элемент ушбу синфнинг барча объектлари томонидан биргаликда қўлланади. Бироқ шундай вазиятлар юз бериши мумкинки, дастур ҳали объектни яратганича йўқ, аммо у элементдан фойдаланиши керак. Элементдан фойдаланиш учун дастур уни `public` ва `static` сифатида эълон қилиши керак. Масалан, қуйидаги дастурда, хатто `book_series` синфидаги объектлар мавжуд бўлмаса ҳам, бу синфнинг `page_count` элементидан фойдаланилади:

```
#include <iostream>
using namespace std;
class book_series
{
public:
static int page_count;
private:
float price;
};
int book_series::page_count;
int main()
{
book_series::page_count = 256;
cout << "page_count ning joriy qiymati " <<
book_series::page_count <<"ga teng"<<endl;
return 0;
}
```

Бу ўринда, синф page_count синфи элементини public сифатида эълон қилгани учун, хатто агар book_series синфидаги объектлар мавжуд бўлмаса ҳам, дастур синфнинг ушбу элементига мурожаат қилиши мумкин.

Статик функция -элементлардан фойдаланиш. Аввалги дастур маълумотлар *статик* элементларининг қўлланишини кўрсатиб берган эди. C++худди шундай усул билан *статик* функция-элементлар (усуллар)ни аниқлаш имконини беради. Агар *статик* усул яратилаётган бўлса, дастур бундай усулни, хатто унинг объектлари яратилмаган ҳолда ҳам, чақириб олиши мумкин. Масалан, агар синф синфдан ташқари маълумотлар учун қўлланиши мумкин бўлган усулга эга бўлса, сиз бу усулни статик қила олишингиз мумкин бўларди. Функциядан фойдаланиш учун дастур уни public ва static сифатида эълон қилиши керак. Масалан, қуйидаги дастурда, хатто book_series синфидаги объектлар мавжуд бўлмаса ҳам, бу синфнинг show_count() усулидан фойдаланилади:

```
#include <iostream>
using namespace std;
class book_series
{
public:
   static int show_count() { return page_count;};
private:
   float price;
   static int page_count;
};
int book_series::page_count=256;
int main()
{
cout << "page_count ning joriy qiymati " <<
book_series::show_count() <<"ga teng"<< endl;

return 0;
}
```

САВОЛЛАР

1. Очиқ (public) ва ёпиқ (private) ўзгарувчи – аъзолар орасида қандай фарқ бор?
2. Синфнинг функция аъзолари қачон ёпиқ бўлиши лозим?
3. Синфнинг функция аъзолари қачон очиқ бўлиши лозим?
4. Агар синф class сўзи ёрдамида таърифланган бўлса кўзда тутилган бўйича компонеталари қандай мурожаат хукуқига эга бўлади

5. Қайси холда синф усуллари жойлаштирилувчи функция хисобланади?
6. Агарда синфнинг иккита объектини эълон қилсак, уларнинг ўзгарувчи аъзолари қиймати турлича бўлиши мумкинми?
7. Конструкторлар хоссаларини кўрсатинг.
8. Синф объектини ҳосил қилишда қандай функция чақирилади?
9. Синф объекти учун ажратилган хотира майдонини тозалашда қандай функция чақирилади.
10. Статик майдонлар хусусий бўлиши мумкинми?

МАСАЛАЛАР

1. Талаба синфини яратинг. Синфда параметрсиз ва параметрли конструктор, киритиш ва чиқариш усуллари яратилсин.

2. Учта томони билан берилган Учбурчак синфини яратинг ва дастурда қўлланг. Синфда юза ва периметрни хисоблаш усуллари бўлсин. Конструкторда берилган уч томон хақиқатан учбурчак ташкил қилиши текширилсин.

3. Криптографик синф яратинг. Синфда Цезарь усули асосида шифрлаш ва дешифрлаш усуллари мавжуд бўлсин. Калит конструкторда киритилсин.

4. Талаба синфини ва дастурда қўлланг. Синфда параметрсиз ва параметрли конструктор, киритиш ва чиқариш усуллари яратилсин. Бундан ташқари талаба номери статик майдони ва шу майдон қийматини чиқарувчи усул яратинг.

5. Футболист(исм, ёш, амплуа, голлар сони) синфини яратинг. Синфда конструктор ва деструктор яратинг.

11 боб. Синфлар орасида муносабатлар

11.1. Синф дўстлари

Синф дўстлари таърифи. Синфнинг компанеталарига мурожат қилишнинг яна бир усули дўстона функциялардан фойдаланишдир. Синфнинг дўстона функцияси деб шу синфга тегишли бўлмаган лекин шу синфнинг химояланган компанентларига мурожат килиш хукуқига эга бўлган функцияларга айтилади. Функция дўстона бўлиши учун синф танасида `friend` спецификатори билан таърифланиши лозим.

Дўстона функцияни таърифлаш:

friend <функция прототипи >

Дўстона функциялардан фойдаланиш хусусиятлари қуйидагилардир:

Дўстона функция мурожат қилинганда `this` кўрсаткичига эга бўлмайди.

Синф объектлари дўстона функцияга параметрлари оркали узатилиши лозим.

Дўстона функция синф компанентаси бўлмагани учун унга танлов амалини қўллаб бўлмайди:

Синф объекти . функция номи

ва Обектга_кўрсаткич-функция номи.

Дўстона функцияга мурожат спецификаторлари (`public, protected, private`) қўлланмайди.

Дўстона функция прототипининг синф усулида жойлаштирилиши фарқи йук.

Дўстона функциялар механизми синфлар орасидаги алоқани соддалаштиришга имкон беради. Синфлардан беркитилган компанеталарига мурожаат қилиш учунгина киритилган функцияларни олиб ташлаш мумкин.

Мисол тариқасида “сохадаги нуқта” ва “сохадаги чизиқ” синфлари учун дўстона функцияни караб чиқамиз. Сохадаги нуқта синфига, (x,y) координаталарини аниқловчи компанеталар киради. Сохадаги чизиқ синфининг компанеталари чизиқнинг умумий тенгламаси `A*x+B*y+C=0` тенгламаси коэффицентлари `A,B,C`.

Қуйидаги дастурда иккала синф учун дўстона бўлган нуқтадан чизиққача масофани хисоблашга имкон берадиган функция киритилган.

```
#include <iostream.h>
class line;
class point
{
float x,y ;
public :
point(float xn=0, float yn=0)
{
```

```
x=xn; y=yn;
}
friend float masofa(point, line);
};
class line
{
float A,B,C;
public:
line(float a, float b,float c)
{
A=a; B=b; C=c;
}
friend float masofa(point, line);
};
float masofa(point P, line L)
{
return L.A*P.x+L.B*P.y+L.C;
};

int main()
{
point P(16.0,12.3);
line L(10.0,-42.3,24.0);
cout << "\n P nuqtasi L chiziqdan cheklanishi: ";
cout << masofa(P,L);
return 0;
}
```

```
Дастур бажарилиши натижаси
P nuqtasi L chiziqdan cheklanishi: -336.29009
```

Бир синф иккинчи синфга дўстона бўлиши мумкин. Бу холда синфнинг хамма компонента функциялари бошқа синфга дўстона бўлади. дўстона синф ўзга синф танасидан ташқари таърифланган бўлиши лозим.

Дўстона синфларни таърифлаш:

friend <синф номи >

Қуйидаги мисолда `book` синфи `librarian` синфини ўзига дўстона синф деб белгилаган:

```
class book
{
public:
   book(string , string , string );
   void show_book(void);
   friend librarian;
private:
```

```
    string title;
    string author;
    string catalog;
};
```

Шунинг учун `librarian` синф объектлари `book` синфнинг хусусий элементларига, нуқта операторидан фойдаланган холда, тўғридан тўғри мурожаат этиши мумкин:

```
#include <iostream>
#include <string>

using namespace std;
class librarian;
class book
{
public:
    book(string , string , string );
    void show_book(void);
    friend librarian;
private:
    string title;
    string author;
    string catalog;
};


    book::book(string title, string author, string
catalog)
    {
    book::title=title;
    book::author=author;
    book::catalog=catalog;
    }
    void book::show_book(void)
    {
    cout << "Nomi: " << title << endl;
    cout << "Muallif: " << author << endl;
    cout << "Katalog: " << catalog << endl;
    }
    class librarian
    {
    public:
    void change_catalog(book &, string );
    string get_catalog(book);
```

```
};
void librarian::change_catalog(book& this_book,
string new_catalog)
{
   this_book.catalog=new_catalog;
}
string librarian::get_catalog(book this_book)
{
   string temp_catalog;
   temp_catalog=this_book.catalog;
   return(temp_catalog) ;
}
int main()
{
   book programming( "C++ tilida dasturlashni
o'rganamiz", "Jamsa", "P101");
   librarian library;
   programming.show_book();
   library.change_catalog(programming, "Engil C++
101");
   programming.show_book();

return 0;
}
```

Кўриб турганимиздек дастур `librarian` синфининг `change_catalog` функциясига book объектини адрес орқали бермоқда. Бу функция синфнинг `book` элементини ўзгартиргани учун, дастур параметрни адрес орқали узатиши ва ундан сўнг ушбу синф элементига мурожат учун кўрсаткич ишлатмоғи лозим. `book` синфи аниқланишидан `friend` оператори ўчириб юборилса C++ компилятори хар гал `book` синфи хусусий маълумотларига мурожатда синтаксик хато хақида хабар чиқаради

Дўстлар сонини чегаралаш. Агарда бир нечта синф функцияларига бошқа синфнинг хусусий маълумотларига мурожаат қилиш керак бўлса, у холда C++ дўстона синфнинг фақатгина белгиланган функциялари хусусий элементларга мурожаат этишига имконият беради. Масалан, фақатгина `change_catalog` ва `get_catalog` функцияларга `book` синфнинг хусусий элементларига мурожаат керак. Қуйида кўрсатилгандек, `book` синфнинг ичида фақатгина шу функцияларда хусусий функцияларга мурожаат чегарасини қўйиши лозим:

```
class book
{
public:
   book(string , string , string );
```

```
    void show_book (void);
    friend string librarian::get_catalog(book);
    friend void librarian::change_catalog( book& ,
string );
private:
    string title;
    string author;
    string catalog;
};
```

Кўриб турганимиздек `friend` операторлари хусусий элементларга мурожат қилувчи хамма дўст функцияларини тўлиқ прототипларини ўз ичига олади.

Агар дастур бир синфдан бошқасига мурожаат қилса ва синфлар аниқланиш тартиби нотўғри бўлса синтаксик хатога дуч келиш мумкин. Бизнинг холда `book` синфи `librarian` синфида эълон қилинган функциялар прототипларига мурожат қилмоқда. Шунинг учун `librarian` синфи аниқланиши `book` синфи аниқланишидан олдин келиши керак, бироқ `librarian` синфи `book` синфига мурожат қилмоқда:

```
class librarian
{
public:
    void change_catalog(book& , string );
    string get_catalog(book);
};
```

`book` синфи аниқланишини `librarian` синфи аниқланишидан олдин қўйиб бўлмагани учун. C++ `book` синфини эълон қилиш имконини беради ва шу билан у компиляторга бундай синф борлиги хақида хабар беради ва кейинроқ ўзи хам аниқланади. Қуйида буни қандай амалга ошириш келтирилган:

```
class book; // синф элон килиниши
```

Қуйидаги дастурда `librarian` синфининг айрим усулларигага `book` синфининг хусусий элементларига мурожаат қилиш имконияти берилган. Синфлар тартибига ахамият беринг:

```
#include <iostream>
#include <string>

using namespace std;
class book;
class librarian
{
```

```
    public:
       void change_catalog(book& , string );
       string get_catalog(book);
    };
    class book
    {
    public:
       book(string , string , string );
       void show_book (void);
       friend string librarian::get_catalog(book);
       friend void librarian::change_catalog( book& ,
string );
    private:
       string title;
       string author;
       string catalog;
    };
    book::book(string title, string author, string
catalog)
    {
       book::title= title;
       book::author= author;
       book::catalog= catalog;
    }
    void book::show_book(void)
    {
       cout << "Nomi: " << title << endl;
       cout << "Muallif: " << author << endl;
       cout << "Katalog: " << catalog << endl;
    }
    void librarian::change_catalog(book& this_book,
string new_catalog)
    {
       this_book.catalog= new_catalog;
    }
    string librarian::get_catalog(book this_book)
    {
       string temp_catalog;
       temp_catalog=this_book.catalog;
       return(temp_catalog);
    }
    int main()
    {
       book programming( " C++ tilida dasturlashni
o'rganamiz ", "Jamsa", "P101");
```

```
    librarian library;
    programming.show_book();
    library.change_catalog(programming, "Engil C++
101");
    programming.show_book();
return 0;
}
```

11.2. Синфларни бошқа синфлардан ташқил топиши

Объект майдон сифатида. Мураккаб синфларни ҳосил қилишда олдин уни ташқил этувчи оддийроқ синфларни эълон қилиб, кейин эса уларни бирлаштириш орқали синфни ҳосил қилиш мақсадга мувофиқдир. Масалан, ғилдирак синфи, мотор синфи, узатиш коробкаси синфи ва бошқа синфларни ҳосил қилиб, кейин эса уларни бирлаштириш орқали автомобил синфини қуриш олдимизга турган масалани ечишни анча осонлаштиради.

Иккинчи мисолни кўриб чиқамиз. Тўғри тўртбурчак чизиқлардан ташқил топган. Чизиқ эса икки нуқта орқали аниқланади. Ҳар бир нуқта `x` ва `y` координаталар ёрдамида аниқланади. Қуйидаги дастурда тўртбурчак синфи кўрсатилган. Тўғри тўртбурчак диагонал бўйича икки нуқта ва икки томон ёрдамида аниқланади. Шунинг учун олдин ҳар бир нуқтанинг `x, y` координаталарини сақлаш учун нуқта синфи эълон қилинган.

Нуқта ва тўғритўртбурчакнинг эълон қилиниши

```
#include <iostream>
using namespace std;
class Point
{
public:
Point(int x1=0,int y1=0)
{
x=x1;y=y1;
}
int  GetX() const {return x;}
int  GetY() const {return y;}
private:
int x;
int y;
};

class Rectangle
{
public:
Rectangle(int x1,int y1,int x2,int y2):
```

```
p1(x1,y1),p2(x2,y2)
{
a=x2-x1;
b=y2-y1;
};
Rectangle(Point a1,Point a2):p1(a1),p2(a2)
{
a=a2.GetX() - a1.GetX();
b=a2.GetY() - a1.GetY();
};
int Per() { return 2*(a+b); }
int Sq()  {return a*b; }
private:
Point p1,p2;
int a,b;
};

int main()
{
Rectangle X(10, 20, 50, 80);
cout << "Perimetr="<<X.Per()<<endl;
cout << "Yuza="<<X.Sq()<<endl;
cout<<endl;
Point a(2,4);Point b(5,6);
Rectangle Y(a,b);
cout << "Perimetr="<<Y.Per()<<endl;
cout<<"Yuza="<<Y.Sq();
return 0;
}
```

Натижа:

```
Perimetr=200
Yuza=2400

Perimetr=10
Yuza=6
```

11.3. Локал синфлар

Синф блок ичида, масалан функция томонида тарифланиши мумкин. Бундай синф модель синф деб аталади. Локал синф компонентларига шу синф тарифланган блок ёки функция ташқарисида мурожат килиш мумкин эмас. Локал синф статик компонентларга эга бўлиши мумкин эмас. Локал синф ичида шу синф аниқланган сонига тегишли номлари; статик (`static`) ўзгарувчилар; ташқи (`extern`) ўзгарувчилар ва ташқи функциялардан

фойдаланиш мумкин. Автоматик хотира турига тегишли ўзгарувчилардан фойдаланиш мумкин эмас. Локал синфлар компонент функциялари фақат жойлашинувчи (inline) функция бўлиши мумкин.

Қуйидаги мисолда моддий нуқта синфи яратилиб, унинг ичида нуқта синфига таъриф берилган ва нуқта синфи объекти майдон сифатида келган:

```
#include <iostream>
using namespace std;
class FPoint
{
public:
//Nuqta sinfi
class Point
{
public:
Point(int x1=0,int y1=0)
{
x=x1;y=y1;
}
int  GetX() const {return x;}
int  GetY() const {return y;}
private:
int x;
int y;
};
//

FPoint(int x1,int y1,double w1):p(x1,y1),w(w1){};
void show()
{
cout<<"koordinata x="<<p.GetX()<<endl;
cout<<"koordinata y="<<p.GetY()<<endl;
cout<<"massa w="<<w;
}
private:
Point p;
double w;
};

int main()
{
cout<<"fizik nuqta"<<endl;
FPoint X(1, 2, 5.5);
X.show();
cout<<"\n\noddiy nuqta"<<endl;
```

```
FPoint::Point Y(2,3);
cout<<"koordinata x="<<Y.GetX()<<endl;
cout<<"koordinata y="<<Y.GetY()<<endl;
return 0;
}
```

Натижа:

```
fizik nuqta
koordinata x=1
koordinata y=2
massa w=5.5

oddiy nuqta
koordinata x=2
koordinata y=3
```

Оддий нуқта синфи моддий нуқта синфининг умумий секциясига жойлаштирилган.

Дастурда моддий нуқта синфи объектидан ташқари нуқта синфи объекти хам яратилган. Лекин бу синф номи олдида моддий нуқта номи ва квалификация оператори жойлаштирилган.

Кейинги мисолимизда нуқта синфи таърифи моддий нуқта синфининг хусусий элементлари секциясига жойлаштирилган:

```
#include <iostream>
using namespace std;
class FPoint
{
public:

FPoint(int x1,int y1,double w1):p(x1,y1),w(w1){};
void show()
{
cout<<"koordinata x="<<p.x<<endl;
cout<<"koordinata y="<<p.y<<endl;
cout<<"massa w="<<w;
}
private:

//Nuqta sinfi
class Point
{
public:
Point(int x1=0,int y1=0)
{
x=x1;y=y1;
```

```
}
int x;
int y;
};
//
Point p;
double w;
};

int main()
{
cout<<"fizik nuqta"<<endl;
FPoint X(1, 2, 5.5);
X.show();
int kk;cin>>kk;
return 0;
}
```

Натижа:

```
fizik nuqta
koordinata x=1
koordinata y=2
massa w=5.5
```

Бу мисолда нуқта синфи хамма элементлари умумий, лекин дастурда бу синфдан фойдаланиб бўлмайди.

САВОЛЛАР

1. Дўстона функция аниқланиш шаклини кўрсатинг.
2. Дўстона синфлар аниқланиш шаклини кўрсатинг.
3. Синф дўстларидан нима учун фойдаланилади?
4. Дўстона синфни эълон қилиш синфнинг қайси секциясида эканлиги ахамиятлими?
5. Синфлар қандай қилиб бошқа синфлардан ташқил топиши мумкин?
6. Агар бир синф объекти бошқа синф майдони бўлса биринчи синф майдонларига қандай мурожаат қилинади?
7. Агар бир синф объекти бошқа синф майдони бўлса биринчи синф конструктори қандай чақирилади?
8. Локал синф деб қандай синфга айтилади?
9. Локал синфлар объектлари қандай аниқланади?
10. Локал синф компоненталарига шу синф тарифланган блок ёки функция ташқарисида мурожат килиш мумкинми?

МАСАЛАЛАР

1. РЕКОРД синфини яратинг. Бу синфга дўстона синф яратинг. Дўстона синф усуллари маълум шартга мос объектларни чиқарсин. Дастурда статик объектлар массиви яратинг. Дўстона синф объекти ёрдамида шарьга мос массив элементларини чиқаринг.

2. АВТОБУС синфини яратинг. Бу синфга дўстона синф яратинг. Дўстона синф усуллари маълум шартга мос объектларни чиқарсин. Дастурда статик объектлар массиви яратинг. Дўстона синф объекти ёрдамида шартга мос массив элементларини чиқаринг.

3. САНА синфини яратинг. Бу синф объектидан майдон сифатида фойдаланиб МАШҒУЛОТ синфини яратинг

4. Нуқта синфини яратинг. Бу синф асосида учбурчак синфини яратинг ва дстурда қўлланг. Бу синфда периметр, юзани хисоблаш ва учбурчакни чизиш усуллари мавжуд бўлсин.

5. Юқорида кўрсатилган мисолда нуқта синфини учбурчак синфи ичида локал синф сифатида жойлаштиринг.

12 боб. Ворислик

12.1. Содда ворислик

Ҳосила синфларни эълон қилиш. C++ тили ўзининг барча аждодларининг хусусиятлари, маълумотлари, усуллари ва воқеаларини мерос қилиб оладиган ҳосила синфини эълон қилиш имкониятини беради. Хосила синфда, шунингдек янги тавсифларни эълон қилиш ҳамда мерос сифатида олинаётган айрим функцияларни қўшимча юклаш мумкин.

Ворислик асос синф кодидан ҳосила синф нусхаларида такроран фойдаланиш имконини беради. Такроран қўллаш концепцияси жонли табиатда ўз параллелига эга: ДНК ни асос материал сифатида олиб қараш мумкинки, у ҳар бир яратилган мавжудотдан ўзининг шахсий турини қайта ишлаб чиқиш учун такроран фойдаланади.

Хосила синфни эълон қилишнинг умумлашган синтаксисини кўриб чиқамиз. Секцияларни санаб ўтиш тартиби ҳимоя имтиёзларини энг чекланганларидан, то энг оммавийларига қараб кенгайиб боришига мос келади:

class className: [<кириш ҳуқуқини берувчи спецификатор>] parent Class {
<Дўстонасинфларни эълон қилиш>
private:
<хусусий маълумотлар аъзолари>
<хусусий конструкторлар>
<хусусий усуллар>
protected:
<ҳимояланган маълумотлар аъзолари>
<ҳимояланган конструкторлар>
<ҳимояланган усуллар>
public:
<оммавий хусусиятлар>
<оммавий маълумотлар аъзолари>
<оммавий конструкторлар>
<оммавий деструктор>
<оммавий усуллар>

Химояланган (**protected**) компоненталар синф ичида ва хосила синфларда мурожаат хуқуқига эга.

Синф ўзининг базавий синфидан юзага келаётганида, унинг барча номлари ҳосила синфда автоматик тарзда яширин `private` бўлиб қолади. Аммо уни, базавий синфнинг қуйидаги кириш спецификаторларини кўрсатган ҳолда, осонгина ўзгартириш мумкин:

■ **private**. Базавий синфнинг мерос бўлиб ўтаётган (яъни ҳимояланган ва оммавий) номлари ҳосила синф нусхаларида кириб бўлмайдиган бўлиб қолади.

■ **protected**. Мерос бўлиб ўтаётган (яъни химояланган ва умумий) элементлар **protected** яъни химояланган мурожаат хуқуқига эга бўлади.

■ **public**. Базавий синф ва унинг аждодларининг номлари ҳосила синф нусхаларида қириб бўладиган бўлади, барча ҳимояланган номлар эса ҳимояланган бўлиб қолаверади.

Базавий синф имкониятларини *кенгайтирадиган* синфларни юзага келтириш мумкин: бу йўл сиз учун ғоят қулай, аммо озгина ишлашни талаб қилган функцияга эга. Ҳосила синфда керакли функцияни янгидан яратиш вақтни бекорга сарфлаш билан баробар. Бунинг ўрнига базавий синфда коддан такроран фодуланиш керак: бунда у талаб қилинган даражада кенгайтирилиши мумкин. Ўосила синфда сизни қониқтирмайдиган базавий синф функциясини қайта аниқланг холос.

Худди шундай йўл билан базавий синф имкониятларини *чеклайдиган* синфларни юзага келтириш мумкин: Бу йўл сиз учун ғоят қулай, аммо ниманидир ортиқча қилади.

Қуйидаги мисолда тавсифларни кенгайтириш ва чеклаш усулларининг қўлланишини кўриб чиқамиз:

```
#include <iostream.h>
    class Base
    {
    public:
    void display(){ cout << "Salom Dunyo.  \n"; }
    };
    class Derived : public Base
    {
    public:
    void display()
    {
    // асос синф display() функциясини бажаради
    Base::display();
    cout <<"Hayr Duno. \n";
    }
    };
int main()
{
    Derived d;
    d.display();    //    хосила    синф    display()
функциясини бажаради
return 0;
}
```

Натижа:

```
Salom Dunyo.
Hayr Duno.
```

Бу мисолнинг `Base:: display()` қаторида асос синф `display()` усулига ошкор мурожаат қилинади. Лекин бу йўл билан хусусий элементларга мурожаат қилиб бўлмайди.

Агар асос синф параметрли конструкторга эга бўлса хосила синф хам албатта параметрли контрукторга эга бўлиши шарт ва бу конструктор инициализаторлар рўйхатида ёки танасида асос синф конструкторини чақириши лозим. Масалан:

```
class Base
{
int a;
public:
Base(int a1):  a(a1){}
};
class Derived : public Base
{
int b;
public:
Derived(int a1, int b1) : Base(a1) ,b(b1){}
};
```

Бу мисолда хосила синф конструкторида асос синф конструктори чақирилади сўнгра хосила синф параметрлари инициализация қилинади.

Параметрлар конструктор танасида инициализация қилиниши мумкин.

Масалан:

```
class Base
{
int a;
public:
Base(int a1)  {a=a1;};
};
class Derived : public Base
{
int b;
public:
Derived(int a, int b) {Base(a1); b=b1;}
};
```

Объектлар яратилганда аввал асос синф конструктори чақирилади, сўнгра хосила синф конструктори. Деструкторлар тескари тартибда чақирилади.

Масалан:

```
#include <iostream.h>
```

```
class Base
{
public:
Base() { cout<<" Base sinfi konstruktori \n"; }
~Base() { cout<<" Base sinfi destruktori \n"; }
};
class Derived : public Base
{
public:
Derived() { cout<<" Derived sinfi konstruktori
\n"; }
~Derived() { cout<<" Derived sinfi destruktori
\n"; }
};
int main ()
{
Derived d;
return 0;
}
```

Натижа:

```
Base sinfi konstruktori
Derived sinfi konstruktori
Derived sinfi destruktori
Base sinfi destruktori
```

Қуйида келтирилган дастурда иккита оддий геометрик объект - доира ва цилиндрнинг синфлар табақаланиши эълон қилинган.

Дастур шундай тузилганки, бунда доиранинг `r` - радиуси ва силиндрнинг `h` - баландлиги ўзгарувчиларининг ички қийматлари яратилаётган объектлар параметрларини аниқлаши керак. `Circle` базавий синфи доирани моделлаштиради, `Cylinder` ҳосила синфи эса цилиндрни моделлаштиради.

```
#include <iostream.h>
#include <math.h>
const double pi = 4 * atan(1);
class Circle {
protected:
double r;
public:
Circle (double rVal =0) : r(rVal) {}
void setRadius(double rVal) { r = rVal; };
double getRadius() { return r; };
double Area() { return pi*r*r; };
void showData();
```

```
};
class Cylinder : public Circle {
protected:
double h;
public:
Cylinder(double hVal = 0, double rVal = 0)
: h(hVal), Circle(rVal) { };
void setHeight(double hVal) { h = hVal; }
double getHeight() { return h; };
double Area() { return 2*Circle::Area()+2*pi*r*h; };
void showData();
};
void Circle::showData()
{
cout <<"Doira radiusi = "<<getRadius()<<endl;
cout<<"Aylana maydoni = "<<Area()<<endl;
};
void Cylinder::showData()
{
cout << "Asos radiusi = "<< getRadius()<<endl;
cout<< "TSilindr balandligi = "<< getHeight()<<
endl;
cout<<"YUza maydoni = "<<Area ()<< endl;
};
int main()
{
Circle circle(2) ;
Cylinder cylinder(10, 1);
circle.showData ();
cylinder.showData();
return 0;
}
```

`Circle` синфининг эълони `r` маълумотларининг ягона аъзоси, конструктор ва қатор усуллардан иборат. Объектни яратишда конструктор `r` маълумотлар аъзосини доира радиусининг бошланғич қиймати билан номлайди (инициаллаштиради). Конструкторнинг янги синтаксисини кўрсатиб ўтамиз: чақиришда у базавий конструктор синфига, шунингдек икки нуқтадан кейин кўрсатилган ҳар қандай маълумотлар аъзосига мурожаат қилиши мумкин. Бизнинг мисолимизда `r` маълумотлари аъзоси унга `rVal` параметри билан мурожаат қилиш орқали «яратилади» ва нулли қиймат билан номланади (инициаллаштирилади).

`setRadius` усули `r` маълумотлар аъзоси қийматини белгилайди, `getRadius` усули эса уни қайтаради. `Area` усули доира майдонини

қайтаради. showData усули айлана радиуси ва доира майдонининг қийматларини чиқариб беради.

Circle синфининг ҳосиласи деб эълон қилинган Cylinder синфи h - ягона маълумотлар аъзоси, конструктор ва қатор усуллардан иборат. Бу синф r маълумотлар аъзосини цилиндр асоси радиусини сақлаш учун, ҳамда setRadius ва getRadius усулларини мерос қилиб олади. Объектни яратишда конструктор r ва h маълумотлар аъзоларини бошланғич қийматлар билан номлайди. Конструкторнинг янги синтаксисини кўрсатиб ўтамиз: бизнинг ҳолатда h маълумотлар аъзоси hVal аргументининг қиймати билан номланади (инициаллаштирилади), r маълумотлар аъзоси эса rVal аргументига эга бўлган базавий синф конструкторини чақириш билан номланади.

setHeight функцияси h маълумотлар аъзоси қийматини белгилайди, getHeight эса қайтаради. Circle::Area функцияси базавий синфдан мерос олинган функцияни, цилиндр юзаси майдонини қайтариш учун, ортиқча юклайди. showData функцияси эса цилиндр асосининг радиуси, баландлиги ва юзасининг майдони қийматларини чиқариб беради.

main функцияси Circle синфига мансуб 2 радиусли Circle айланасини ҳамда баландлиги 10, асосининг радиуси 1 бўлган Cylinder синфига мансуб cylinder цилиндрни яратади, кейин яратилган объектларнинг параметрларини чиқариш учун showData га мурожаат қилади.

```
Айлана радиуси=2 Доира майдони=12.566
Асос радиуси=1 силиндр баландлиги=10 Юзасининг майдони=69.115
```

12.2. Кўпликдаги ворислик

Кўплик ворислик таърифи. Кўплик ворислик деб синф таърифида бир неча синф асос синф сифатида кўрсатиш имконига айтилади.

Масалан:

```
class Basis1
{ int a,b;
public:
Basis1(int x,int y){a=x;b=y;}
};
class Basis2
{ int c;
  char*s;
public:
Basis2(int x,char*y){c=x; b=new char[strlen(y)+1];}
~Basis2(){delete[]s;}
};
class Inherit:public Basis1,public Basis2
```

```
    {int sum;
    public:
    Inherit(int x,int y, int z, char*d, int
k):Basis1(x,y),Basis2(z,d){sum=k;}
    };
```

Куйидаги дастурда `computer` синфини яратиш учун, `computer_screen` ва `mother_board` асос синфларидан фойдаланилади:

```
    #include <iostream>
    #include <string>

    using namespace std;
    class computer_screen
    {
    public:
       computer_screen(string , long, int, int);
       void show_screen(void);
    private:
       string type;
       long colors;
       int x_resolution;
       int y_resolution;
    };
    computer_screen::computer_screen(string type, long
colors, int x_res, int y_res)
    {
       computer_screen::type= type;
       computer_screen::colors = colors;
       computer_screen::x_resolution = x_res;
       computer_screen::y_resolution = y_res;
    }
    void computer_screen::show_screen(void)
    {
     cout << "Ekran tipi: " << type << endl;
     cout << "Rang: " << colors << endl;
     cout << "Kattaligi: " << x_resolution << " ga " <<
y_resolution << endl;
    }
    class mother_board
    {
    public:
       mother_board(int, int, int);
       void show_mother_board(void);
```

```
    private:
       int processor;
       int speed;
       int RAM;
    };
    mother_board::mother_board(int processor, int speed,
int RAM)
    {
       mother_board::processor = processor;
       mother_board::speed = speed;
       mother_board::RAM = RAM;
    }
    void mother_board::show_mother_board(void)
    {
     cout << "Processor: " << processor << endl;
     cout << "CHastota: " << speed << "MGC" << endl;
     cout << "OZU: " << RAM << " MBayt" << endl;
    }
    class computer : public computer_screen, public
mother_board
    {
    public:
       computer(string , int, float, string , long, int,
int, int, int, int);
       void show_computer(void);
    private:
       string name;
       int hard_disk;
       float floppy;
    };
    computer::computer(string name, int hard_disk, float
floppy, string screen, long colors, int x_res, int
y_res, int processor, int speed, int RAM) :
computer_screen(screen, colors, x_res, y_res),
mother_board(processor, speed, RAM)
    {
       computer::name= name;
       computer::hard_disk = hard_disk;
       computer::floppy = floppy;
    }
    void computer::show_computer(void)
    {
       cout << "Tip: " << name << endl;
       cout << "Qattiq disk: " << hard_disk << "MBayt"
<< endl;
```

```
    cout << "Yumshoq disk: " << floppy << "MBayt" <<
endl;
    show_mother_board();
    show_screen();
}
int main()
{
computer my_pc("Compaq", 212, 1.44, "SVGA",
16000000, 640, 480, 486, 66, 8);
my_pc.show_computer();
return 0;
}
```

Бу мисолда `computer` синфи конструктори `mother_board` и `computer_screen` конструкторларини чақирилади:

```
computer::computer(string name, int hard_disk, float
floppy, string screen, long colors, int x_res, int
y_res, int processor, int speed, int RAM) :
computer_screen(screen, colors, x_res, y_res),
mother_board(processor, speed, RAM)
```

12.3. Полиморф усуллар

Синфларда полиморфизм. Полиморфизм юқорида айтилганидек юнонча сўз бўлиб, иккита ўзакдан - поли (кўп) ва морфос (шакл) дан иборат ҳамда кўп шакллиликни билдиради. Полиморфизм - бу турдош объектлар (яъни битта аждод ҳосиласи бўлган синфларга мансуб объектлар) нинг дастур бажарилиш вақтида вазиятга қараб, ўзларини турлича тута олиш хусусияти. Объектга мўлжалланган ёндошув доирасида дастурчи объект хулқ-атворига фақат билвосита таъсир кўрсатиши, яъни дастурга киритилаётган усулларни ўзгартириши ҳамда авлодларга ўз аждодларида йўқ бўлган ўзига хос хусусиятларни қўшиши мумкин.

Усулни ўзгартириш учун уни авлодда қўшимча юклаш керак, яъни авлодда битта номдаги усулни эълон қилиш ва унда керакли хатти-ҳаракатларни ишга солиш керак. Натижада аждод-объект ва авлод-объектда битта номдаги иккита усул амал қилади. Бунда ушбу усулларнинг кодлари турлича ишга туширилади ва, демакки, объектлар турлича хатти-ҳаракат кўрсатади. Масалан, геометрик шакллар турдош синфларининг табақаланишида (нуқта, тўғри чизиқ, квадрат, тўғрибурчак, доира, еллипс ва ҳ.к.) ҳар бир синф `Draw` усулига эга бўлиб, у ушбу шаклни чизиб бериш талаби қўйилган воқеа-ҳодисага тегишли жавоб берилиши учун масъулдир.

Полиморфизм туфайли, авлодлар битта воқеага ўзига хос тарзда муносабат билдириш учун, ўз аждодларининг умумий усулларини қўшимча юклашлари мумкин.

Виртуал функциялар. ОМД да полиморфизмга фақат юқорида тавсифи берилган ворислик ва аждод усулини қўшимча юклатиш механизми воситасида эмас, балки *виртуаллаш* воситасида ҳам эришиладики, у аждод функцияларга ўз авлодлари функцияларига мурожаат қилиш имконини беради. Полиморфизм синф архитектураси орқали ишга туширилади, бироқ фақат аъзо-функциялар полиморф бўлишлари мумкин.

C++да полиморф функция битта номдаги эҳтимолий функциялардан бирига фақат бажарилиш пайтида, яъни унга синфнинг конкрет объекти узатилаётган пайтда боғлаб қўйилади. Бошқача қилиб айтганда, дастлабки дастур матнида функциянинг чақирилиши фақат таҳминан белгиланади, айнан қандай функция чақирилаётгани аниқ кўрсатилмайди. Бу жараён кечиккан боғланиш деб ном олган. Навбатдаги мисол оддий аъзо-функцияларнинг полиморф бўлмаганлиги нимага олиб келиши мумкинлигини кўрсатади.

```
#include <iostream>
#include <string>

using namespace std;
class Parent
{
public:
void F1() { cout<<"I am Parent"<<endl; };
void F2(int n) {
for(int i=0;i<n;i++) F1();
};
};
class Child : public Parent
{
public:
void F1() { cout<<"I am Parent"<<endl; }
};
int main() {
Child child;
child.F2(3);

int kk;cin>>kk;
return 0;
}
```

Натижа:

```
I am Parent
I am Parent
I am Parent
```

Parent синфи F1 ва F2 аъзо-функцияларга эга, бунда F2 ни F1 чақиради. Parent синфининг ҳосиласи бўлган Child синфига F2 функцияси ворисликка ўтади, бироқ F1 функцияси қайта таърифланади. Компилятор ворисликка ўтган F2 функцияни Parent::F1 функцияси билан боғлаб трансляция қилиб юборади.

C++кечиккан боғланишни функция бажарилиш пайтида аниқлайди ҳамда функцияларни *виртуаллаш* воситасида уларда полиморфликни таъминлайди. Аждод ва авлод синфларда виртуал функцияларни эълон қилиш синтаксисини умумлаштирадиган мисолни кўриб чиқамиз:

```
class className1 {
//Boshqa a’zo-funktsiyalar
virtual return Type functionName(<parametrlar ro‘yxati>);
}
class className2 : public className1 {
//Boshqa a’zo-funktsiyalar
virtual return Type functionName(<>);
}
```

Parent ва Child синфлари объектларида F1 функцияси полиморфлигини таъминлаш учун, уни виртуал деб эълон қилиш зарур. Қуйида дастурнинг модификацияланган матни келитирилади:

```
#include<iostream.h>
class Parent
{
public:
virtual void F1() { cout<<”I am Parent<<endl; };
void F2(int n) {
for(int i=0;i<n;i++) F1();
};
};
class Child : public Parent
{
public:
void F1() { cout<<”I am Parent<<endl; }
};
int main() {
Child child;
child.F2(3);
```

```
return 0;
}
```

Натижа:

```
I am Child
I am Child
I am Child
```

Мана энди дастур кутилаётган натижани чиқариб беради. Компилятор `Child.F2(3)` ифодасини мерос қилиб олинган `Parent::F2` функцияси мурожаатига трансляция қилиб юборади, бу функция эса, ўз навбатида, `Child::F1` авлодининг қайта аниқланган виртуал функциясини чақириб олади.

Агар функция базавий синфда виртуал деб эълон қилинган бўлса, уни фақат ҳосила синфларда қайта аниқлаш мумкин, бунда параметрлар рўйхати аввалгидек қолиши зарур. Агар ҳосила синфнинг виртуал функцияси параметрлар рўйхатини ўзгартирган бўлса, бу ҳолда унинг базавий синфдаги (ҳамда унинг барча аждодларидаги) версияси кириб бўлмас бўлиб қолади. Бошида бундай вазият боши берк кўчага кириб қолгандек кўриниши мумкин, - амалда ортиқча юкланиш механизмини қўллаб-қувватламайдиган ОМД тилларида шундай бўлади ҳам. C++бу муаммони виртуал функциялардан эмас, балки худди шу номли, фақат бошқа параметр рўйхатига эга бўлган ортиқча юкланган функциялардан фойдаланган ҳолда хал қилади.

Виртуал деб эълон қилинган функция, ҳосила синфларда **виртуал** калит-сўз билан эълон қилингани ёки қилинмаганидан қатъи назар, барча ҳосила синфларда виртуал ҳисобланади.

Вирутал функциялардан берилган синф объектларининг ўзига хос хулқ-атворини ишга солиш учун фойдаланинг. Барча усулларингизни виртуал деб эълон қилманг, - бу уларни чақиришда қўшимча ҳисоблаш сарфларига олиб келади. Ҳамма вақт деструкторларни виртуал деб эълон қилинг. Бу синфлар табақаланишида объектларни йўқ қилишда полиморф хулқ-атворни таъминлайди.

Полиморф объект-телефоннинг яратилиши. Айтайлик, сизнинг бошлиқларингиз сизга объект-телефонингиз дискли, тугмачали ёки тўловли телефонлардан бирини танлаб олиб, эмуляция қила олиши керак деди. Бошқача қилиб айтганда, объект-телефон битта қўнғироқ учун тугмачали аппарат сифатида, бошқа қўнғироқ учун тўловли телефон сифатида ва ҳ.к. ишлаши мумкин эди. Яъни бир қўнғироқдан иккинчисига сизнинг объект-телефонингиз ўз шаклини ўзгартириши лозим бўлади.

Турли синфларга мансуб бу телефонларда фақат битта фарқланувчи функция мавжуд - бу `dial` усули. Полиморф объектни яратиш учун, сиз аввал базавий синф функцияларини, уларнинг прототиплари олдидан виртуал калит-сўзни қўйган ҳолда, аниқлайсиз. Бу базавий синф функциялари ҳосила синфлар функцияларидан шунинг билан фарқланадики, улар *виртуал*дирлар.

Кейин дастурда параметри базавий синф объектига илова глобал функция тузилади. Функция танасда `dial` усулига мурожаат қилинади:

```
    void dial_phone(phone& this_phone,string
this_number)
    { this_phone.dial(this_number);};
```

Объект шаклини ўзгартириш учун, сиз, қуйида кўрсатилганидек, ушбу функцияга ҳосила синф объектини параметр сифатида узатасиз:

```
    dial_phone(home_phone,"303-555-1212");
```

Функцияда келган (`phone&`) белгиси турларга келтиришга имкон бериб, бир турдаги ўзгарувчи (`touch_tone`) адресини бошқа турдаги ўзгарувчи (`phone`)га бериш зарурлигини маълум қилади. Дастур `dial_phone` функциясига турли объектлар адресини тақдим қилиши мумкин экан, демакки, функция полиморф бўлиши мумкин. Навбатдаги дастурда бу усулдан объект-телефон яратиш учун фойдаланади. Дастур ишга туширилгач, `poly_phone` объекти ўз шаклини дискли телефондан тугмачалисига, кейин эса тўловлисига ўзгартиради:

```
    #include <iostream>
    #include <string>

    using namespace std;
    class phone
    {
    public:
       virtual void dial(string number) { cout << "Raqam
to'plami " << number << endl; }
       void answer(void) { cout << "Javobni kutish" <<
endl; }
       void hangup(void) { cout << "Qo'ng'iroq
bajarildi-trubkani qo'yish" << endl; }
       void ring(void) { cout << "Qo'ng'iroq,
qo'ng'iroq, qo'ng'iroq" << endl; }
       phone(string number) { phone::number= number; };
    protected:
       string number;
    };
    class touch_tone : public phone
    {
    public:
```

```
    void dial(string number) { cout << "Pik Pik Raqam
to'plami " << number << endl; }
    touch_tone(string number) : phone(number) { }
};
class pay_phone: public phone
{
public:
    void dial(string number) { cout << "Iltimos
to'lang " << amount << " sent" << endl;
    cout << "Raqam to'plami " << number << endl; }
    pay_phone(string number, int amount) :
phone(number) { pay_phone::amount = amount; }
private:
    int amount;
};

void dial_phone(phone& this_phone,string
this_number)
{ this_phone.dial(this_number);};

int main()
{
    pay_phone city_phone("702-555-1212", 25);
    touch_tone home_phone("555-1212");
    phone rotary("201-555-1212") ;
        // Ob'ekt diskli telefonga aylantirilsin
    dial_phone(rotary,"818-555-1212");
        // Ob'ekt shakli tugmachali telefonga
o'zgartirilsin
    dial_phone(home_phone,"303-555-1212");
        // Ob'ekt shakli to'lovli telefonga
o'zgartirilsin
    dial_phone(city_phone,"212-555-1212");

return 0;
}
```

Агар ушбу дастур компиляция қилиниб ишга туширилса, дисплей экранида қуйидаги ёзув пайдо бўлади:

```
S:\> POLYMORP <ENTER>
Raqam to‘plami 818-555-1212
Pik Pik Raqam to‘plami 303-555-1212
Iltimos to‘lang 25 sent
```

poly_phone объекти дастур бажарилиши давомида ўз шаклини ўзгартириб турар экан, у полиморф бўлади.

12.4. Абстракт синфлар

Абстракт синф таърифи. Хеч бўлмаса битта соф (бўш) виртуал функцияга эга бўлган синф абстракт синф дейилади.

Қуйидаги эълонга эга бўлган компонентали функция соф виртуал функция дейилади:

virtual <тип> <функция_номи>
(<формал_параметрлар_рўйхати>) = 0;

Бу ёзувда «= 0» конструкция «соф спецификатор» дейилади. Соф виртуал функция таърифига мисол:

```
virtual void fpure (void) = 0;
```

Соф виртуал функция хеч нарса қилмайди ва уни чақириб бўлмайди. Унинг қўлланилиши – хосила синфларда унинг ўрнини эгалловчи функциялар учун асос бўлиш. Абстракт синф эса хосила синф учун асосий (базавий) синф сифатида ишлатилиши мумкин.Агар соф виртуал функция хосила синфда тўлиқ таърифланмаса, у хосила синфда хам соф виртуал бўлиб қолади, натижада хосила синф хам абстракт синф бўлади.

Абстракт синфни фақат бошқа синф аждоди сифатида ишлатиш мумкин:

Баъзи синфлар абстракт тушунчаларни ифодалайди ва улар учун объект яратиб бўлмайди. Бундай синфлар бирор хосила синфда маънога эга бўлади:

масалан,

```
class Abstract
{
public:
virtual void draw() = 0;
};
class Derived : public Abstract
{
public:
void draw() { cout « "Salom.";}
};
int main( void )
{
Derived d;
Abstract a;
return 0;
}
```

Агар соф виртуал функция хосила синфда тўлиқ таърифланмаса, у хосила синфда хам соф виртуал бўлиб қолади, натижада хосила синф хам абстракт синф бўлади.

```
class Abstract
{
virtual int f() =0;
virtual float g(float) = 0;
};
class Derived : class Abstract
{
int f();
};
int main (void)
{
Abstract a; //hato
Derived d; // hato
}
```

Абстракт синфлар реализация деталларини аниқлаштирмасдан фақат интерфейсни кўрсатиш учун ишлатилади. Масалан операцион тизимда қурилма драйвери абстракт синф сифатида берилиши мумкин:

```
class character_device {
public:
   virtual int open() = 0;
   virtual int close(const char*) = 0;
   virtual int read(const char*, int) =0;
   virtual int write(const char*, int) = 0;
   virtual int ioctl(int ...) = 0;
   // ...
};
```

Драйверлар `character_device` синфининг аждодлари сифатида киритилиши мумкин.

Абстракт синф туридаги ўзгарувчи яратиб бўлмайди, лекин абстракт синф туридаги кўрсаткич яратиш мумкин. Бу кўрсаткичга абстракт синф абстракт бўлмаган турли авлодлари адресини қиймат сифатида бериб, абстракт усулга мос турли усулларни чақириш мумкин.

Масалан юқоридаги полиморф телефон қуйидагича таърифланиш мумкин:

```
#include <iostream>
#include <string>

using namespace std;
class abstract_phone
```

```
{
public:
   virtual void dial(string number)=0;
   void answer(void) { cout << "Javobni kutish" <<
endl; }
   void hangup(void) { cout << "Qo'ng'iroq
bajarildi-trubkani qo'yish" << endl; }
   void ring(void) { cout << "Qo'ng'iroq, qo'
ng'iroq, qo'ng'iroq" << endl; }
   abstract_phone(string number) {
abstract_phone::number= number; };
protected:
   string number;
};
class phone:public abstract_phone
{
public:
   virtual void dial(string number) { cout << "Raqam
to'plami " << number << endl; }
   phone(string number): abstract_phone(number) {};
};
class touch_tone : public abstract_phone
{
public:
   void dial(string number) { cout << "Pik Pik Raqam
to'plami " << number << endl; }
   touch_tone(string number) :
abstract_phone(number) { }
};
class pay_phone: public abstract_phone
{
public:
   void dial(string number) { cout << "Iltimos
to'lang " << amount << " sent" << endl;
   cout << "Raqam to'plami " << number << endl; }
   pay_phone(string number, int amount) :
abstract_phone(number) { pay_phone::amount = amount; }
private:
   int amount;
};
void dial_phone(abstract_phone& this_phone,string
this_number)
{ this_phone.dial(this_number);};
int main()
{
```

```
    pay_phone city_phone("702-555-1212", 25);
    touch_tone home_phone("555-1212");
    phone rotary("201-555-1212") ;
       // Ob'ekt diskli telefonga aylantirilsin
    dial_phone(rotary,"818-555-1212");
       // Ob'ekt shakli tugmachali telefonga
o'zgartirilsin
    dial_phone(home_phone,"303-555-1212");
       // Ob'ekt shakli to'lovli telefonga
o'zgartirilsin
    dial_phone(city_phone,"212-555-1212");
  return 0;
  }
```

САВОЛЛАР

1. Ворислик нима учун керак?
2. Химояланган protected ва хусуий private хуқуқлари орасида қандай фарқ бор?
3. Нима учун аввал аждод синф конструкторлари чақирилиб, сўнгра авлод синф конструктори чақирилади?
4. Нима учун деструкторлар конструкторларга нисбатан тескари тартибда чақирилади?
5. Ворисликда аждод синф спецификатори сифатида protected кўрсатилиши мумкинми?
6. Синфлар библиотекасини қуришда ворисликдан қандай фойдаланилади?
7. Хусусий деб эълон қилинган компоненталарга бошқа синф усуллари орқали мурожаат қилиш мумкинми?
8. Полиморфизм деб нимага айтилади?
9. Усулни виртуал деб эълон қилиш қандай имкон яратади?
10. Абстракт синф деб қандай синфга айтилади?

МАСАЛАЛАР

1. Ворисликдан фойдаланиб Артист ва Цирк артисти синфини яратинг.
2. Ворисликдан фойдаланиб Нуқта, Тўртбурчак ва Тўғри Тўртбурчак синфларини яратинг.
3. Ворислик ва абстракт синфдан фойдаланиб детал, радиодетал, конденсатор синфини яратинг.
4. Ворисликдан ва абстракт синфдан фойдаланиб синов, имтиҳон, битирув имтиҳони синфларини яратинг.
5. Ворисликдан ва абстракт синфдан фойдаланиб нашр, китоб, ўқув қўлланма синфларини яратинг.

13 боб. Синфларда полиморфизм

13.1. Стандарт амалларини қўшимча юклаш

С ++ тилининг ажойиб хусусиятларидан бири стандарт амалларни янги маълумотлар турларига қўллаш имкониятидир. Масалан сатрлани улашни `S1=S2` кўринишда белгилаш анча кулайдир. Бу амални символли массивларга ёки сатрли константаларга қўллашнинг иложи йўқ. Лекин `S1` ва `S2` ни бирор синф объектлари сифатида тавсифланса улар учун `'+'` амалини киритиш мумкин бўлади. Амални маълумотларнинг янги типига қўллаш учун дастурчи `"` операция – функция `"` деб аталувчи махсус функцияни киритиши лозим. Операция – функция таърифи қуйидагича.

Кайтарилувчи_маълумот_типи `operator` операция_белгиси
(операция_функция_параметрлари_спецификацияси)
{ операция_функция_танаси_операторлари }

Керак бўлганда бу функция оператор прототипини киритиш мумкин.

Кайтарилувчи_маълумот_типи `operator` операция_белгиси
{ операция_функция_параметрлари_спецификацияси)

Мисол учун * амални бирор T синфга тегишли объектларга қўллаш учун қуйидагича функция эълон килиши мумкин:

```
T operator * ( Tx,Ty )
```

Бу усулда таърифланган операция қўшимча юкланган (инглизчасига- `overload`) деб аталади. Агар `T` синф учун юқорида келтирилган турдаги функция эълон қилинган булса, `A*B` ифода `operator (A,B)` сифатида каралади.

Синф объектларига функция-операторни қўллаш учун операция-функция ёки синф компанента функцияси ёки дўстона функция бўлиши, ёки параметрлардан бирортаси синф типига эга бўлиши керак.

➢ Амаллар кенгайтирилганда улар учун хар хил типлар комбинациясини олдиндан назарда тутиш лозим. Лекин операция-функцияларга мурожат қилинганда стандарт типлар алмашинувчи қоидалари ишлатилади, шунинг учун типларнинг хамма комбинацияларини хисобга олиш зарурати йук. Кўпгина холларда бинар амаллар учун қуйидаги холларни хисобга олиш етарлидир.

```
# стандарт тип, синф
# синф, стандарт тип
# синф, синф
```

Масалан: Complex синфи учун қуйидаги дўстона операция-функцияларни киритиш мумкин:

```
complex operator+ (Complex x, Complex y)
{
return (Complex(x.real+y.real, x.imag+y.imag());
}
```

```
Complex operator+ (double x, Complex y)
{
return (Complex(x+y.real, y.imag());
}

Complex operator+(Complex x, double y}
{
return (Complex (x.real+y, x.imag ());
}
```

Қуйидаги дастурда ифодалар қўлланган:

```
#include <iostream>
using namespace std;
class Complex
{
double re,im;
public:
Complex(){};
Complex(double re1,double im1)
{
re=re1;im=im1;
}
void show()
{
 cout<<"re="<<re<<" im="<<im<<endl;
};
friend Complex operator+ (Complex a, Complex b);
friend Complex operator+ (double x, Complex b);
friend Complex operator+(Complex a, double y);
};
Complex operator+ (Complex a, Complex b)
{
return (Complex(a.re+b.re, a.im+b.im));
};
Complex operator+ (double x, Complex b)
{
return (Complex(x+b.re, b.im));
}
Complex operator+(Complex a, double y)
{
return (Complex (a.re+y, a.im));
};
```

```
int main ( )
{
 Complex CC (1.0, 2.0);
 Complex EE(3.0,4.0);
 Complex DD=CC+EE;
 DD.show();
 EE=EE+2.0;
 EE.show();
return 0;
}
```

Натижа:

```
re=4 im=6
re=5 im=4
```

Стандарт типларни синф объектига келтириш вазифасини конструкторга топшириш мумкин. Масалан `Complex` синфига қуйидаги конструкторни қўйиш хамма ёрдамчи функциялардан халос бўлиш имконини беради:

```
Complex (double x)
{
real=x; imag=0.0;
}
```

Бу холда қуйидаги прототипга эга бўлган дўстона операция функциядан фойдаланиш етарли.

```
friend Complex operator+ (Complex, Complex);
```

Синфга конструктор қўшиш ўрнига ягона конструкторга иккинчи параметр қийматини киритиш етарли:

```
Complex (double re, double im=0.0)
{
real=re; imag=im;
}
```

Иккинчи имконият синф компанента функциялардан фойдаланишдир. Хар қандай бирор амал синфга тегишли статик компанента операция-функция ёрдамида қайта юкланиши мумкин. Бу холда битта параметрга эга бўлиб, сарлавхаси қуйидаги кўринишда бўлади:

```
 T operator & (T X)
```

Бу ерда Т-синф, &-операция.

Операция функцияни оддий синф компонента функцияси сифатида чақириш мумкин.

Масалан:

```
#include <iostream>
using namespace std;
```

```
class Complex
{
double re,im;
public:
Complex(){};
Complex(double re1,double im1=0)
{
re=re1;im=im1;
}
void show()
{
 cout<<"re="<<re<<" im="<<im<<endl;
};
Complex operator+ (Complex a);
Complex operator+=(Complex b);
};
Complex Complex:: operator+ (Complex a)
{
return (Complex(a.re+re, a.im+im));
};
Complex Complex:: operator+=(Complex b)
{
re+=b.re;im+=b.im;
return (Complex(re, im));
}

int main ( )
{
 Complex CC (1.0, 2.0);
 Complex EE(3.0,4.0);
 Complex DD=CC+EE;
 DD.show();
 EE+=2.0;
 EE.show();
return 0;
}
```

Натижа:

```
re=4 im=6
re=5 im=4
```

Биз синф амалини глобал операция-функция ва компанента операция-функция ёрдамида кайта юклашни кўриб чикдик. Энди унар амални синф дўстона функцияси ёрдамида қайта юклашни кўриб чиқамиз.

“N ўлчовли фазо радиус-вектори “ синфини киритамиз ва бу синф учун '- '-унар операция функцияни киритамиз. Бу операция вектор йуналишини тескарисига ўзгартиради.

```
#include <iostream>
using namespace std;
const int MAX=100;
class vector
{
int N;
double x[MAX];
friend vector& operator - (vector &);
public:
vector (int n, double xn[]);
void display ( );
};
vector::vector (int n, double xn[])
{
N=n;
for ( int i=0; i<N; i++ ) x[i]=xn[i];
}
void vector:: display ( )
{
cout<<"\n Vector koordinatalari :";
for ( int i=0; i<N; i++ )
cout<< "\t"<<x[i];
};
vector& operator -(vector& v)
{
for ( int i=0; i<v.N; i++ )
v.x[i]=-v.x[i];
return v;
};
int main ( )
{
double A[]={1.0,2.0,3.0,4.0};
vector V(4,A);
V.display();
V=-V;
V.display();
return 0;
}
```

Натижа:

```
Vector koordinatalari : 1 2 3 4
Vector koordinatalari : -1 -2 -3 -4
```

Қуйидаги компаненталарни кайта юклаш мумкин эмас.

`.` - структураланган объект компанентасини тўғридан тўғри танлаш.

`.*` -компонентага кўрсаткич оркали мурожаат килиш;

`? :` -шартли операция:

`::` -кўриниш доирасини аниқлаш;

`sizeof` -хотира хажмини аниқлаш ;

`#` -препроцессор директиваси;

`##` -процессорли амал;

Қайта юклаш механизми яна қуйидаги хусусиятларга эга:

- Стандарт амалларни қўшимча юкланганда приоритетларини ўзгартириш мумкин эмас.
- Қўшимча юкланган амаллар учун ифодалар синтаксисини ўзгартириш мумкин эмас. Унар = ёки бинар ++ амалларни киритиш мумкин эмас.
- Амаллар учун янги символлар киритиш мумкин эмас, масалан кўпайтириш учун `**` белгиси.
- Хар қандай бинар амал икки усул билан аникланади, ёки бир параметрли компанента функция сифатида ёки глобал ёки дўстона глобал икки параметрли функция.

Биринчи холда `x*y` ифода x. operator*(y) мурожатни иккинчи холда эса operator*(x,y) мурожатни билдиради.

- Бинар `'='`, `'[ ]'`, `'->'` амаллар семантикасига кўра `operator=`, `operator[ ]`, `operator->` глобал функция бўлолмайди. Балким ностатик компанента функцияси бўлиши лозим.
- Хар қандай унар амал синф объектлари учун икки усулда аниқланади ёки параметрсиз компонента функция ёки бир параметрли (балким дўстона) глобал функция.
- Префикс амал учун `++z` ифода, компонента функция `z.operator++( )` ёки глобал функция `operator++(z)` чақирилишини билдиради.
- C++ тилида баъзи амалларни бошқа амалларнинг комбинацияси сифатида аникланади. Мисол учун `long m=0` бутун сон `++m` учун `m+=1` ни, бу амал булса m=m+1 ни билдиради. Бундай автоматик алмаштиришлар қўшимча юкланган амаллар учун бажарилмайди. Мисол учун умумий холда `operator*=()` таърифни `operator*( )` таъриф ва `operator=( )` таърифдан келтириб чикариб бўлмайди.
- Агар ифодада фойдаланувчи киритган синф объекти қатнашмаса унинг маъносини ўзгартириб бўлмайди. Мисол учун фақат кўрсаткичларга таъсир қилувчи амалларни киритиш мумкин эмас.
- Агар операция функциянинг биринчи параметри стандарт тип бўлиши керак бўлса, бундай операция-функция компанента-функция булолмайди. Мисол учун `AA` - бирор синф объекти бўлсин ва унинг учун '+'

амали қўлланган бўлсин. AA+2 ифода учун ёки AA.operator(2) ёки operator+(AA,2) ифода чақирилиши мумкин. 2++AA ифода учун operator+(AA,2) чақирилиши мумкин лекин z. operator+(AA) хатога олиб келади.

С ++ тилининг замонавий версияларида префикс ++ ва -- операцияларни қўшимча юклаш бошқа операцияларни юклашдан фарқ қилмайди. Постфикс шаклдаги ++ва -- амалларини кайта юклаганда яна бир int типидаги параметр киритилиши керак.Агар қўшимча юклаш учун глобал функция ишлатилса унинг биринчи параметри шиф типига, иккkинчи параметри int типига эга бўлиши керак.

Дастурда постфикс ифода ишлатилганда бутун параметр хам қийматга эга бўлади.

Қуйидаги дастурда префикс ++ва --хамда постфикс ++ва -- операцияларини қўшимча юклаш кўрсатилган.

```
#include <iostream>
using namespace std;
class Pair
{
int N;
double x;
friend Pair& operator ++(Pair&);
friend Pair& operator ++(Pair&, int);
public:
Pair (int n, double xn)
{
N=n; x=xn;
}
void display ()
{
cout<<"\n Koordinatalar: N="<<N<<"\tx="<<x;
}
Pair& operator --()
{
N/=10; x/=10;
return Pair(N,x);
};
Pair& operator --(int k)
{
N/=2;
x/=2.0;
return Pair (N,x);
}
};
```

```
Pair& operator ++(Pair& P)
{
P.N*=10; P.x*=10;
return P;
}
Pair& operator ++(Pair& P, int k)
{
P.N=P.N*2+k;
P.x=P.x*2+k;
return P;
}
int main ()
{
Pair Z(10,20.0);
Z.display();
++Z;
Z.display();
--Z;
 Z.display();
Z++;
Z.display();
Z--;
Z.display();
return 0;
}
```

Дастур бажарилиши натижалари:

```
Координаталар:     N=10 X=20
Координатадар:     N=100 X=200
Координаталар:     N= 10 X=20
Координаталар:     N=20 X=40
Координаталар:     N=10 X=20
```

Бу мисолда префикс ++ қийматни 10 марта оширишни постфикс ++ булса 2 марта оширишни билдиради. Префикс -- қийматни 10 марта камайтириш, постфикс -- бўлса қийматни 20 марта камайтиришни билдиради.

Типларни ўзгартириш оператори. Юқорида конструктор ёрдамида стандарт типдаги яъни хақиқий ўзгрувчини синф типидаги яъни комплекс типидаги ўзгарувчига келтиришни кўрдик. Тескарисини амалга ошириш учун типни ўзгартириш операторидан фойдаланиш мумкин. Қуйидаги мисолда каср сон синфи киритилиб, каср сонни хақиқий сонга келтириш оператори киритилган:

```
#include <iostream>
using namespace std;
```

```
class Fraction
{
int s,m;
public:
Fraction(int s1,int m1=1){s=s1;m=m1;};
void show(){cout<<s<<'/'<<m<<endl;};
operator double()
{
return (double)s/m;
}
};
int main ( )
{
 Fraction D(1,2);
 D.show();
 double c=D;
 cout<<c<<endl;
 D=1;
 D.show();
return 0;
}
```

Натижа:
1/2
0.5
1/1

13.2. Шаблонлар

Функция шаблони. Шаблонлар (параметрланган турлар) боғланган функциялар ёки синфлар оиласини тузиш имконини беради. Шаблонни аниқлашнинг умумий синтаксиси қуйидаги кўринишга эга:

temlate <шаблонли турлар рўйхати>{эълон қилиш};

Функциялар шаблонлари ва синфлар шаблонлари фарқланади.

Функция шаблони қўшимча юкланаётган функцияларни аниқлаш намунасини беради. Солиштиришга йўл қўядиган ҳар қандай турдаги иккита аргументдан каттароғини қайтарадиган `max (x, y)` функцияси шаблонини кўриб чиқамиз:

template <class T>T max(Tx, Ty){**return**(x>y)? x:y;}:

бунда `<class T>`шаблонининг аргументи томонидан тақдим этилган маълумотлар тури ҳар қандай бўлиши мумкин. Ундан дастурда

фойдаланишда компилятор max функцияси кодини бу функцияга узатилаётган параметрларнинг фактик турига мувофиқ генерация қилади.

Мисол:

```
#include <iostream>
using namespace std;
template <class T>
T maximum(T x, T y)
{
if (x>y) return x;else return y;
};
int main()
{
 int i=3;
 float a=3.0,b=7.5;
 i=maximum(i,0);//аргументлар тури int
 cout<<i<<endl;
 float m=maximum(a, b);// аргументлар тури float
 cout<<m<<endl;
return 0;
}
```

Натижа:

```
3
7.5
```

Фактик турлар компиляция пайтида маълум бўлишлари керак. Шаблонларсиз max функциясини кўп марталаб қўшимча юклашга тўғри келар эди, яъни, гарчи барча функция версияларининг кодлари бир хил бўлса ҳам, ҳар бир қўлланаётган тур учун алоҳида ортиқча юклаш керак бўлар эди. C++стандарти бу мақсад учун **#define max(x, y) ((x>y)? x:y)** макросидан фойдаланишни қатъиян тавсия этмайди, чунки C++тилига оддий C тили устидан шундай афзалликларни берадиган турларни текшириш механизми блокировка қилинган бўлиши мумкин. Шу нарса аёнки, max(x, y) функциясининг вазифаси - бир-бирига мос турларни қиёслаш. Афсуски, макросни қўллаш бир-бирига мос келмайдиган турларнинг (масалан, int ва struct) қиёсланишига йўл қўяди.

```
//иккита ўзгарувчининг ўрнини ўзгартирадиган функция шаблони:
template<class T>//T - параметрланаётган тур номи
void change(T&x, T&y)
{
T z=x;
x=y;
```

```
 y=z;
}
```

Бу функция қуйидагича чақирилиши мумкин:

```
long k=10, i=5;
change(k, i);
```

Компилятор:
`void change(long& x, long& y){long z=x; x=y; y=z;}` таърифини ифодалаб беради.

Функциялар шаблонлари параметрларининг асосий хусусиятлари:

1. Параметрлар номлари шаблоннинг бутун таърифи бўйлаб уникал бўлмоғи лозим.
2. Шаблон параметрларининг рўйхати бўш бўла олмайди.
3. Шаблон параметрлари рўйхатида ҳар бири `class` сўзидан бошланадиган бир нечта параметр бўлиши мумкин.

Функция шаблонини қўшимча юклаш. Функция шаблонини қўшимча юклаш мумкин. Масалан икки бутун сон қўшилганда албатта хақиқий сон хосил бўлиши керак бўлсин:

```
template<class T> T add (T a, T bЬ)
{
return a + b;
}
float add (int a, int b)
{
return (float) a+b;
}
```

Энди add функциясини икки бутун сон учун чақирилса иккинчи функцияга мурожжат қилинади, қолган холларда функция шаблонига мурожаат қилинади.

Юқоридаги функция шаблонларида битта умумий типдан фойдаланилган. Кўп холларда функция шаблонида бир нечта тип кўрсатиш талаб этилади. Қуйидаги мисолда `show_array` функция шаблонидан фойдаланилган бўлиб, массив элементларини чиқариш учун мўлжалланган. T тип массив типини аниқлаш, T1 тип count параметри типини кўрсатиш учун ишлатилади. Дастурда `int` ва `float` типидаги массивлар чиқарилади.

```
#include <iostream>
using namespace std;
template<class T,class T1>
void show_array(T array[],T1 count)
{
```

```
    T1 index;
    for (index =0; index < count; index++) cout <<
array[index]<< "  ";
    cout << endl;
    }
    int main()
    {
       int pages[] = { 100, 200, 300, 400, 500 };
       float prices[] = { 10.05, 20.10, 30.15 };
       show_array(pages, 5);
       show_array(prices, 3);
    return 0;
    }
```

Натижа:

```
100, 200, 300, 400, 500
10.05, 20.10, 30.15
```

Синфлар шаблони. Синфлар шаблони синфлар оиласини аниқлаш намунасини беради. Қуйида бир ўлчамли маълумотлар массиви синфларининг генератори бўлган Массив шаблонига мисол келтирилган:

```
template <class T>class array
```

Ушбу синфнинг турлаштирилган элементлари устида, элементларнинг конкрет туридан қатъи назар, бир хил базавий операциялар (киритилсин, ўчирилсин, индекслансин ва ҳ.к.) бажарилади. Агар турга параметрдек муомала қилинса, бу ҳолда компилятор берилган тур элементларига эга бўлган векторлар синфларини генерация қилади.

Навбатдаги дастур иккита синфни яратишда `array` синф шаблонидан фойдаланади. Бу синфларнинг бири `int` турининг қийматлари билан, иккинчиси `float` турининг қийматлари билан иш кўради.

```
#include <iostream>
#include <fstream>
using namespace std;
const int MAX=1000;
template<class T,class T1>
class Array
{
public:
   Array(int max_size);
   int add_value(T);
   T sum();
   T1 average_value();
   void show_array();
```

```
    private:
    T data[MAX];
    int max_size;
    int index;
};
template<class T,class T1>
Array<T,T1>::Array(int size)
{
    if (size>=MAX)
    {
        cerr << "Xotira etarli emas - dastur
tugayapti" << endl;
        exit(1);
    }
    Array::max_size = size;
    Array::index = 0;
};

template<class T,class T1>
int Array<T,T1>::add_value(T value)
{
    if (index == max_size)
    return(-1); // Massiv to'la
    else
    {
        data[index] = value;
        index++;
        return(0); // Omadli
    }
}
template<class T,class T1>
T Array<T,T1>::sum()
{
    T sum = 0;
    for (int i = 0; i < index; i++) sum += data[i];
    return(sum);
}
template<class T, class T1>
T1 Array<T,T1>::average_value()
{
    T1 sum =0;
    for (int i = 0; i < index; i++) sum += data[i];
    return (sum / index);
}
template<class T,class T1>
```

```
void Array<T,T1>::show_array()
{
   for (int i = 0; i < index; i++) cout << data[i]
<< ' ';
   cout << endl;
}
int main()
{
// 100 ta elementdan iborat massiv
   Array<int,float> numbers(100);
   // 200 ta elementdan iborat massiv
   Array<float,float> values(200);
   int i;
   for (i = 0; i < 5; i++) numbers.add_value(i);
   numbers.show_array();
   cout << "Sonlar summasi teng " << numbers.sum ()
<< endl;
   cout << "O'rtacha qiymat teng " <<
numbers.average_value() << endl;
   for (i = 0; i < 5; i++) values.add_value((float)i
* 100);
   values.show_array();
   cout << "Sonlar summasi teng " << values.sum() <<
endl;
   cout << "O'rtacha qiymat teng " <<
values.average_value() << endl;
return 0;
}
```

Натижа:

```
0 1 2 3 4
Sonlar summasi teng 10
O'rtacha qiymat teng 2
0 100 200 300 400
Sonlar summasi teng 1000
O'rtacha qiymat teng 200
```

Навбатдаги дастурда `array` синф шаблони глобал функция аргументи бўлиб хизмат қилади.

```
#include <iostream>
using namespace std;
const int MAX=1000;
template<class T>
class array
{
```

```
public:
    array(int max_size);
    int add_value(T);
    int size(){return index;};
    T& operator[](int i)
    {
    return data[i];
    }
private:
    T data[MAX];
    int max_size;
    int index;
};

template<class T>
array<T>::array(int size)
{
    if (size>=MAX)
    {
        cerr << "Xotira etarli emas - dastur
tugayapti" << endl;
        exit(1);
    }
    array::max_size = size;
    array::index = 0;
};
template<class T>
int array<T>::add_value(T value)
{
    if (index == max_size)
    return(-1); // Massiv to'la
    else
    {
        data[index] = value;
        index++;
        return(0); // Omadli
    }
}
template<class T>
T sum(array<T> data)
{
 int index=data.size();
    T sum = 0;
    for (int i = 0; i < index; i++) sum += data[i];
    return(sum);
```

```
}
template<class T, class T1>
T average_value(array<T1> data)
{
 int index=data.size();
   T1 sum =0;
   for (int i = 0; i < index; i++) sum += data[i];
   return (sum / index);
}
template<class T>
void show_array(array<T> data)
{
 int index=data.size();
   for (int i = 0; i < index; i++) cout << data[i]
<< ' ';
   cout << endl;
}
int main()
{
// 100 ta elementdan iborat massiv
   array<int> numbers(100);
   // 200 ta elementdan iborat massiv
   array<float> values(200);
   int i;
   for (i = 0; i < 5; i++) numbers.add_value(i);
   show_array(numbers);
   cout << "Sonlar summasi teng " << sum (numbers)
<< endl;
   cout << "O'rtacha qiymat teng " <<
average_value<float,int>(numbers) << endl;
   for (i = 0; i < 5; i++) values.add_value((float)i
* 100);
   show_array(values);
   cout << "Sonlar summasi teng " << sum(values) <<
endl;
   cout << "O'rtacha qiymat teng " <<
average_value<float,float>(values) << endl;
return 0;
}
```

Натижа:

```
0 1 2 3 4
Sonlar summasi teng 10
O'rtacha qiymat teng 2
0 100 200 300 400
Sonlar summasi teng 1000
```

```
O'rtacha qiymat teng 200
```

Синфлар шаблони хоссалари. Синфлар шаблони қуйидаги қўшимча хоссаларга эга:

Шаблон учун typedef оператори ёрдамида янги тип киритиш мумкин:

```
typedef Array<int,int> IntArray;
```

Бу янги тип ёрдамида ўзгарувчилар таърифи берилиши мумкин:

```
IntArray numbers(100);
```

Шаблон параметри кўзда тутилган қийматга эга бўлиши мумкин

```
template<class T, class T1=int>
```

Бу холда қуйидагича таъриф киритиш мумкин:

```
Array<int> numbers(100);
```

Шаблон параметри стандарт типдаги ўзгарувчи бўлиши мумкин

```
template<class T, class int ki=100,T1=float>
```

Бу ўзгарувчи конструкторда массив ўлчамини бериш учун ишлатилиши мумкин.

Бу холда қуйидагича таъриф киритиш мумкин:

```
Array<int,100> numbers;
Array<float> values;
```

Бу уччала холат қуйидаги мисолда берилган

```
#include <iostream>
using namespace std;
template<class T, class T1=int, int c=0>
class Trio
{
public:
 Trio(T a1,T1 b1){a=a1;b=b1;};
 void show(){cout<<"a="<<a<<" b="<<b<<"
c="<<c<<endl;}
private:
   T a;
   T1 b;
   };

int main()
{
   Trio<int,float,2> a(1,2.5);
   a.show();
   Trio<int> b(1,2);
   b.show();
   typedef Trio<int> IntTrio;
```

```
    IntTrio c(1,1);
    c.show();
return 0;
}
Натижа:
a=1 b=2.5 c=2
a=1 b=2 c=0
a=1 b=1 c=0
```

САВОЛЛАР

1. Деструкторларни қўшимча юклаш мумкинми?
2. Статик усулларни қўшимча юклаш мумкинми?
3. Постфикс ва префикс функцияларни қўшимча юклаш хусусиятларини кўрсатинг.
4. Қандай амалларни фақат синф аъзоси сифатида қўшимча юклаш мумкин?
5. Шаблонлардан нима мақсадда фойдаланилади?
6. Функция шаблони асосий хоссаларини кўрсатинг.
7. Параметрлаштирилган синфлар хоссаларини кўрсатинг.
8. Шаблон параметрлари рўйхати бўш бўлиши мумкинми?
9. Параметрлаштирилган функция қандай чақирилади?
10. Синф шаблони ташқарисида компонента функциялар қандай аниқланади?

МАСАЛАЛАР

1. Массив ўсувчи эканлигини текширувчи функция шаблонини яратинг.
2. Массив камаювчи эканлигини текширувчи функция шаблонини яратинг.
3. Массив хамма элементланри ўзаро тенглигини текширувчи функция шаблонини яратинг.
4. Стек синфи шаблонини яратинг.
5. Навбат синфи шаблонини яратинг.

14 боб. Оқимли синфлар

14.1. Оқимли синфлар ва объектлар

Олдиндан белгиланган объектлар ва оқимлар. C++да киритиш-чиқариш оқимларининг синфлари мавжуд бўлиб, улар киритиш-чиқариш стандарт кутубхонаси (`stdio.h`) нинг объектга мўлжалланган эквиваленти (`stream.h`) дир. Улар қуйидагича:

Ios	базавий оқимли синф
streambuf	оқимларнинг буферланиши
istream	киритиш оқимлари
ostream	чиқариш оқимлари
iostream	иккига йўналтирилган оқимлар
iostream_withassign	ноаниқ ўзлаштириш операцияли оқим
istrstream	сатрли киритиш оқимлари
ostrstream	старли чиқариш оқимлари
strstream	иккига йўналтирилган сатрли оқимлар
ifstream	файлли киритиш оқимлари
ofstream	файлли чиқариш оқимлари
fstream	иккига йўналтирилган файлли оқимлар

Стандарт оқимлар (`istream, ostream, iostream`) терминал билан ишлаш учун хизмат қилади.

Сатрли оқимлар (`istrstream, ostrstream, strstream`) хотирада жойлаштирилган сатрли буферлардан киритиш-чиқариш учун хизмат қилади.

Файлли оқимлар (`ifstream, ofstream, fstream`) файллар билан ишлаш учун хизмат қилади.

Қуйидаги объект-оқимлар дастурда `main` функциясини чақириш олдидан аввалдан аниқланган ва очилган бўлади:

```
extern istream cin;  //Клавиатурадан киритиш стандарт оқими
extern ostream cout; //Экранга чиқариш стандарт оқими
extern ostream cerr; //Хатолар ҳақидаги хабарларни чиқариш стандарт оқими (экран)
extern ostream cerr:  //Хатолар ҳақидаги хабарларни чиқаришнинг буферлаштирилган стандарт оқими.
```

Бу объектлар жойлашган файлда киритиш-чиқариш операциялар таърифлари хам берилгандир. Биринчи операция >> «ўнгга» суриш амали, орқали белгиланиб оқимдан ўқиш деб аталади.

Иккинчи операция << «чапга» суриш амали оркали белгиланиб оқимга чиқариш ёки ёзиш амали деб аталади. Оқимга ўқиш ва ёзиш амаллари << ва >> амалларини қўшимча юклаш орқали хосил қилинган бўлиб, чап операция `iostream` ва `ostream` синфи объекти бўлса ўнг ўзгарувчи ёки ифодадир. Бу ўзгарувчи, константа ёки ифода типи `char, unsigiod short, signed short, signed int, unsigned int, signed long, unsigned long, flout, double, long double, char, void` бўлиши мумкин. Шуни айтиш керакки `char` типидан ташқари хамма кўрсаткичлар `void` типига келтирилади.

Оқим билан алмашиш функциялари. Оқимга қўшиш << ва оқимдан ўқиш >> амаллардан ташқари , кириш –чиқиш библиотекаларида қўшишга фойдали функциялар мавжуддир. Маълумотларни чиқаришда `ostream` синфи ишлатилади . Бу синфда қуйидаги чиқариш функциялари мавжуддир .

```
ostream & ostream  ::put (char cc );
ostream & ostream  ::write (const signed char *array ,int u);
ostream  &ostream::write  (const  nn  signed  char *array,int u);
```

`put ()` функция чиқувчи оқимга битта символ жойлайди : `cout .put ('Z');`

`write()` – функцияси икки параметрга эга

`array` –хотира қисмига кўрсатгич ва `n`- символлар сони.

Мисол:

```
char ss []= "merci";
cout.write (ss,sizeof (ss)-1);
```

Натижа

```
merci
```

Форматсиз ўқиш функциялари `istream` оқимига тегишлидир .

Булар 6 та қўшимча юкланган `get ()` функцияси

Булардан иккисининг прототитини қуйидагича :

```
istream & get (signed char * array ,int  max _len , char ='\n');
istream & get (unsigned char *array , int max _len ,char='\n');
```

Бу икки функция кирувчи оқимдан кетма- кет байтларини ажратиб символли массивга ёзади. Иккинчи параметр байтлар максимал соннини кўрсатади.

Фойдаланувчи томонидан киритилган типлар учун киритиш ва чикариш. Алмашиш операциялари << ва >> ихтиерий типларга қўлланилиши учун янги операция функциялар киритилиши лозимдир. Операцияларни қўллашга юклаш функция-операцияси кўриниши қуйидагичадир :

```
ofstream & operator << (ofstream & out, янги-тип - номи)
{  -----------
   out << ---------;
   return  out ;
}
```

Мисол учун:

```
#include <iostream>
using namespace std;
struct  point
{
 float   x;
 float   y;
 float   z;
};
ostream &operator <<(ostream &t, point d)
{
return t<<"\n x="<<d.x<<" y="<<d.y <<" z="<<d.z;
//return t;
}
int   main()
{
point  F={10.0, 20.0, 30.0};
cout<<"\n nuqta koordinatalari:"<<F;
return 0;
}
```

Натижа:

Нуқта координатлари :

x=10.0 y=20.0 z=30.0

Киритиш амали >>қайта юклаш учун қуйидаги кўринишдаги функция операцияни аниқлаш лозим :

```
istream &operator >>(istream &in, янги тип & ном )
{...
in >> ...;
return in ;
}
```

Мисол учун:

```
#include <iostream>
using namespace std;
struct point
{
float x;
float y;
float z;
};
istream &operator >>(istream& in, point &d)
{
cout<<"\n nuqta koordinatalarini kiriting:";
cout<<"\nx="; in>> d.x;
cout<<"y=";   in >>d.y;
cout<<"z=";   in >>d.z;
return in;
};
int main()
{
point P;
cin>>P;
return 0;
}
```

бу дастур бажарилиши натижаси қуйидагича бўлиши мумкин:
нуқта координаталарини киритинг :
x=100
y=200
z=300

14.2. Форматлаш

Форматлаш байроқчалари. Киритиш << ва чиқариш >> операцияларни `cout, cin, cerr, clog` стандарт потокларга тўғридан тўғри қўллаш қайта узатиш қийматларни ташқи тавсифлаш айтиб ўтилмаган форматлардан фойдаланишга олиб келади.

Форматлаш учун `ios` синфнинг қуйидаги `protected` компоненталари ишлатилади:

`int x_width` – чиқариш майдоннинг минимал эни.

`int x_precision` – киритишда хақиқий сонларнинг тавсифлаш аниқлиги (каср қисимнинг рақамлар сони);

`int x_fill` – чиқаришда тўлдирувчи символ, бўшлик белгиси – кўзда тутилган холда.

Ушбу майдонларни қийматларини олиш (ўрнатиш) учун қуйидаги функция компоненталар ишлатилади:

```
int width();
int width(int);
int precision();
int precision(int);
char fill();
char fill(char);
```

Агар бир марта *cout.fill,* ёрдамида тўлдирувчи-белги танланса *cout.fill* қайта чақирилиб ўзгармагунча хақиқий бўлиб қолади.

Манипуляторлар. Манипуляторлар - оқим ишини модификациялашини имкон этувчи махсус функциялар. Манипуляторларнинг хусусияти шундаки, уларни >> ёки << операцияларнинг ўнг операнд сифатида фойдаланиш мумкин. Чап операнд сифатида эса хар доимгидак оқим (оқимга илова) ишлатилади, ва худда шу оқимга манипулятор таъсир этади. Манипуляторлар икки турга булинади : параметрсиз ва параметрли манипуляторлар.

`endl` фақат чикаришда ишлатилиб, янги сатр символини чикаради

`dec` Ўнлик санок тизимида чикаради (кўзда тутилган бўйича)

`hex` Ўнлик олтилик санок тизимида чикаради

`oct` Саккизлик санок тизимида чикаради

`ws` фақат киритишда ишлатилиб, бўшлик символларини ўтказади

`ends` фақат чикаришда ишлатилиб, оқимга катор охири ноль белгисини кушади

`flush` фақат чикаришда ишлатилиб, оқим буферини тозалайди

// қуйидаги манипуляторлар учун `#include <iomanip>` талаб этилади

`setfill( ch)` ch символ билан бўш жойни тўлдириш

`setprecision( n )` n га тенг бўлган сузувчи нуқтали сонни чиқариш аниқлилигини ўрнатиш

`setw( w )` w га тенг бўлган киритиш ёки чиқариш майдоннинг энини ўрнатиш

`setbase( b )` b асосига эга бўлган бутун сонларнинг чиқариш; b қийматлари 0,8,10 ёки 16 бўлиши мумкин

`resetiosflags(long L)` L параметрининг битли қиймати асосида оқимлар байрокларини тозалайди

`setiosflags(long L)` L параметрининг битли қиймати асосида оқимлар холатлари байрокларини урнатади

7 Мисол:

Функция-компонента cout.fill ва манипулятор setw ()

```
#include <iostream>
using namespace std;
int main()
{
cout << "Ahborot jadvali " << endl;
```

```
    cout.fill ('.');
    cout << "Kompaniya sohasi " << setw(20) << 10 <<
endl;
    cout << "Kompaniya daromadi va zarari " << setw(12)
<< 11 << endl;
    cout << "Kompaniya rahbariyati " << setw(14) << 13
<< endl;
    return 0;
    }
```

14.3. Файллар билан ишлаш

Файллар билан ишлаш синфлари. C++да файллар билан ишлаш `fstream` кутубхонасидаги бирон-бир синфлар ёрдамида амалга оширилади.

`fstream` кутубхонаси файлларни ўқиб олиш учун жавоб берадиган `ifstream` синфига, ҳамда файлга ахоботнинг ёзиб олинишига жавоб берадиган `ofstream` синфига эга.

Бирон-бир файлни ёзиш ёки ўқиш учун очиш учун, `ofstream` турдаги ўзгарувчини яратиб, инициаллашда файл номидан фойдаланиш лозим:

```
ofstream file_object("FILENAME.EXT");
```

Агар файл мавжуд бўлмаса ,янгидан яратилади ва оқимга уланади. Агар файл мавжуд бўлса у ўчирилади, ва бўш файл янгидан яратилади .

Агар файл ҳам дастурнинг бажарилаётган файли жойлаштирилган папкада бўлса, у ҳолда файлнинг номи тўлиқ кўрсатилмаслиги мумкин (фақат файл номи, унга бориш йўлисиз). Бундан ташқари файл номини тўғридан-тўғри кўрсатиш ўрнига, унинг номидан иборат белгилар массивларини кўрсатиш мумкин.

```
char s[20]= "C:\text.txt";
ofstream file_object (s);
```

Қуйидаги дастурда файлга уч қатор маълумот ёзилади:

```
#include <fstream.h>
int main()
{
  ofstream book_file("BOOKINFO.DAT");
  book_file <<"C++ tilida dasturlashni o'rganamiz"
<< endl;
  book_file << "Jamsa Press" << endl;
  book_file << "22.95" << endl;
return 0;
}
```

Бирон-бир файлни ёзиш ёки ўқиш учун очиш учун, ifstream турдаги ўзгарувчини яратиш керак.

```
ifstream input_file("filename.EXT");
```

Бундай файл мавжуд бўлмаса оқим яратилмайди. Қуйидаги дастурда файлдан уч қатор маълумот ўқилади:

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   ifstream input_file("BOOKINFO.DAT") ;
    char one[64], two[64], three[64];
    input_file >> one;
    input_file >> two;
    input_file >> three;
    cout << one << endl;
    cout << two << endl;
    cout << three << endl;
return 0;
  }
```

Сатр ҳақида гап кетганда, чиқариш сатр охири белгиси, яъни \n пайдо бўлишидан олдин амалга оширилади. Белгисиз турга эга бўлган барча ўзгарувчилар олдин белгиларга ўзгартириб олинади.

Ахборотни файлдан ўқиб олиш учун >> операторига эквивалент бўлган get функцияси қўлланади. Бу функция ҳар қандай ўзгарувчиларнинг стандарт турлари ёки белгилар массивлари билан ишлай олади. Шунингдек get га ҳар жиҳатдан эквивалент бўлган getline функцияси мавжуд: фарқи фақат шундаки, getline функцияси сатр охиридани охирги белгини қайтармайди.

Бутун сатрни файлдан ўқиб олиш учун getline усулидан фойдаланиш қулайдир:

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream input_file("BOOKINFO.DAT");
    char one[64], two[64], three[64];
    input_file.getline(one, sizeof(one));
    input_file.getline(two, sizeof(two));
```

```
    input_file.getline(three, sizeof(three));
    cout << one << endl;
    cout << two << endl;
    cout << three << endl;
return 0;
  }
```

Файл охирини аниқлаш. Файл ичидагисини, файл охири учрамагунча, ўкиш дастурдаги оддий файл операцияси ҳисобланади. Файл охирини аниқлаш учун, дастурлар оқим объектининг `eof` функциясидан фойдаланишлари мумкин. Агар файл охири ҳали учрамаган бўлса, бу функция 0 қийматини қайтариб беради, агар файл охири учраса, 1 қиматини қайтаради. `While` циклидан фойдаланиб, дастурлар, файл охирини топмагунларича, қуйида кўрсатилганидек, унинг ичидагиларини узлуксиз ўқишлари мумкин:

```
while (! Input_file.eof())
{
  //Operatorlar
}
```

Ушбу ҳолда дастур, eof функцияси ёлғон (0) ни қайтаргунча, сиклни бажаришда давом этади. Навбатдаги дастур `BOOKINFO.DAT` файли ичидагисини, файл охирига етмагунча, ўкиш учун `eof` функциясидан фойдаланади.

```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
ifstream input_file("BOOKINFO.DAT");
char line[64];
while (! input_file.eof())
   {
    input_file.getline(line, sizeof(line));
    cout << line << endl;
}
return 0;
}
```

Худди шундай, кейинги дастур - файл ичидагисини битта сўз бўйича бир мартада, файл охири учрамагунча, ўқийди:

```
#include <iostream.h>
```

```
#include <fstream.h>
int main ()
{
ifstream input_file("BOOKINFO.DAT");
char word[64];
while (! input_file.eof())
   {
   input_file >> word;
   cout << word << endl;
}
return 0;
}
```

Ва, ниҳоят, кейинги дастур - файл ичидагисини битта белги бўйича бир мартада `get` функциясидан фойдаланиб, файл охири учрамагунча, ўқийди:

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
ifstream input_file("BOOKINFO.DAT");
char letter;
while (! input_file.eof())
{
   letter = input_file.get();
   cout << letter;
}
return 0;
}
```

Файл операцияларини бажаришда хатоларни текшириш. Ҳозиргача тақдим этилган дастурларда кўзланганидек, файл операцияларини бажаришда хатолар содир бўлмайди. Афсуски, бунга ҳамма вақт ҳам эришиб бўлмайди. Масалан, агар сиз киритиш учун файл очаётган бўлсангиз, дастурлар ушбу файл мавжудлигини текшириб кўриши керак. Худди шундай, агар дастур маълумотларни файлга ёзаётган бўлса, операция муваффакятли ўтганига ишонч ҳосил қилиш керак (масалан, дискда бўш жойнинг йўқлиги маълумотларнинг ёзиб олинишига тўсқинлик қилади). Хатоларни кузатиб боришда дастурларга ёрдам бериш учун, файл объектининг `fail` функциясидан фойдаланиш мумкин. Агар файл операцияси жараёнида хатолар бўлмаган бўлса, функция ёлғон (0) ни қайтаради. Бироқ, агар хато учраса, `fail` функцияси ҳақиқатни қайтаради. Масалан, агар дастур файл очадиган бўлса, у, хатога йўл қўйилганини аниқлаш учун, `fail` функциясидан фойдаланиши керак. Бу қуйида шундай кўрсатилган:

```
ifstream input_file("FILENAME.DAT");
if (input_file.fail())
{
   cerr << "Ochilish xatosi FILENAME.EXT" << endl;
   exit(1);
}
```

Шундай қилиб, дастурлар ўқиш ва ёзиш операциялари муваффақиятли кечганига ишонч ҳосил қилишлари керак. Қуйидаги дастурда турли хато вазиятларни текшириш учун `fail` функциясидан фойдаланади:

```
   #include <iostream>
   #include <fstream>
   using namespace std;
int main()
{
   char line[256] ;
   ifstream input_file("BOOKINFO.DAT") ;
   if (input_file.fail()) cerr << "Ochilish xatosi
BOOKINFO.DAT" << endl;
   else
   {
      while ((! input_file.eof()) && (!
input_file.fail()))
      {
          input_file.getline(line, sizeof(line)) ;
         if (! input_file.fail()) cout << line << endl;
      }
   }
   return 0;
}
```

Файлнинг керак бўлмай қолганда беркитилиши. Дастурни тугаллаш учун операция тизими ўзи очган файлларни беркитади. Бироқ, одатга кўра, агар дастурга файл керак бўлмай қолса, уни беркитиши керак. Файлни беркитиш учун дастур, қуйида кўрсатилганидек, дастур `close` функциясидан фойдаланиши керак:

```
input_file.close ();
```

Файлни ёпаётганингизда, дастур ушбу файлга ёзиб олган барча маълумотлар дискка ташланади ва ушбу файл учун каталогдаги ёзув янгиланади.

Ўқиш ва ёзиш операцияларининг бажарилиши. Ҳозирга қадар гап бораётган дастурлар белгили сатрлар устида операциялар бажарар эди.

Дастурларингиз мураккаблашган сари, эҳтимол, сизга массивлар ва тузилмаларни ўқиш ва ёзиш керак бўлиб қолар. Бунинг учун дастурлар `read` ва `write` функцияларидан фойдаланишлари мумкин. `read` ва `write` функцияларидан фойдаланишда маълумотлар ўқиладиган ёки ёзиб олинадиган маълумотлар буферини, шунингдек буфернинг байтларда ўлчанадиган узунлигини кўрсатиш лозим. Бу қуйида кўрсатилганидек амалга оширилади:

```
input_file.read(buffer, sizeof(buffer));
output_file write(buffer, sizeof(buffer));
```

Масалан, қуйидаги дастурда тузилма ичидагисини `EMPLOYEE.DAT` файлига чиқариш учун, `write` функциясидан фойдаланади:

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
struct employee
   {
   char name[64];
   int age;
   float salary;
} worker = { "Djon Doy", 33, 25000.0 };
   ofstream emp_file("EMPLOYEE.DAT");
emp_file.write((char *) &worker, sizeof(employee));
return 0;
}
```

Одатда `write` функцияси белгилар сатрига кўрсаткич олади. (`char*`) белгилари турларга келтириш оператори бўлиб, бу оператор сиз кўрсаткични бошқа турга узатаётганингиз ҳақида компиляторга ахборот беради. Худди шундай тарзда қуйидаги дастурда `read` усулидан хизматчи ҳақидаги ахборотни файлдан ўқиб олиш учун фойдаланади:

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
struct employee
   {
   char name[64] ;
   int age;
```

```
      float salary;
   } worker = { "Djon Doy", 33, 25000.0 };
     ifstream emp_file("EMPLOYEE.DAT");
   emp_file.read((char *) &worker, sizeof(employee));
   cout << worker.name << endl;
   cout << worker.age << endl;
   cout << worker.salary << endl;
   return 0;
   }
```

Файллар билан маълумот алмашиш. Файллар билан ишловчи оқимлар қуйидаги синфлар объектлари сифатида яратилади:

`ofstream` –маълумотларни файлга ёзиш учун ;

`ifstream` –файлдан маълумотларни ўқиш учун;

`fstream` – маълумотларни ўқиш ва ёзиш учун.

Бу синфлардан фойдаланиш учун дастур матнига ёрдамчи сарловҳали файл `fstream.h` қўшилиши лозим.Шундан сўнг файлли оқимларни қуйидагича аниқлаш мумкин :

```
ofstream out_file;
ifstream in_file ;
fstream io_file;
```

Файлли оқимли яратишдан сўнг конкрет файлга `open` компонента функцияси ёрдамида уланиш мумкин. Бу функция қуйидаги кўринишга эга:

```
void open (const char *filename , int mode=кўзда тутилган қиймат
int protection =қўзда тутилган қиймат )
```

Биринчи параметр `filename` мавжуд ёки яратилаётган файл номи, иккинчи параметр `mode` –файл билан ишлаш режимлари кўрсатувчи белгилар дизъюнкцияси, учинчи параметр `protection` (химоя) –кам ишлатилади. Тўғрироғи дастурчи учун кўзда тутилган қиймати етарлидир .

Файл билан ишлаш режимлари белгилари қуйидагича аниқланади:

```
enum   ios:: open _mode {
in = 0x01 //фақат ўқиш учун очиш:
out =0x02//фақат ёзиш учун очиш:
ate=0x04 //очилганда файл охирини излаш:
app=0x08 //маълумотларни файл охирига қўшиш:
trunc=0x10 //мавжуд файл ўрнига янгисини яратиш:
nocreate =0x20//янги файл очилмасин (файл мавжуд бўлмаса open функцияси хато хақида маълумот беради )
noreplace=0x40//мавжуд файл очилмасин
binary=0x80//иккилик (матнли эмас) алмашинув учун очилсин.
```

`open` функциясини чақириш қуйидагича амалга оширилади

Оқим_ номи open(файл номи ,режим, химоя)

Мисоллар:

```
outFile. open("C:\\user\\result.dat");
inFile. open("Data.txt");
ioFile. Open("Chance.dat", ios::out);
```

Оқим ofstream синфига тегишли бўлса, иккинчи параметр ios:out қийматга эга бўлади.

Очиш open() функциясининг муфаққиятли бажарилганлигини текшириш учун ортиқча юкланган ! амалидан фойдаланилади. Агар хато мавжуд бўлса натижа 0 дан фарқли бўлади. Мисол учун :

```
if(!int file)
{
cerr <<"файлни очишда хато:\n";
exit(1);
}
```

fstream синфга тегишли оқимлар учун иккинчи параметр аниқ кўрсатилиши шарт.

Мисол келтирамиз :

```
#include <iostream>
#include <fstream>
using namespace std;
const int lenname =13;
const int lenstring =60;
int main()
{
char source[lenname];
cout<<"\n fayl nomini kiriting:";
cin>>source;
ifstream infile;
infile.open(source);
if (!infile)
{
cerr<<"\n hato" <<source;
exit(1);
}
char string [lenstring];
char next;
cout<<"\n fayl matni :\n \n";
cin.get();
while (1)
{
infile >> string;
next= infile.peek();
if (next==EOF) break;
cout << string <<" ";
```

```
if (next=='\n')
{
cout<<'\n';
static  int i=4;
if ( !(++i % 20))
{
cout<<"\n ENTER bosing  \n " << endl;
cin.get();
}
}
}
return 0;
}
```

Дастур ишлаши натижаси экранга матнли файлни сахифалаб чиқаришдан иборат.Сахифа 20 қатордан иборат.

14.4. Бинар файллар билан ишлаш.

Оқим кўрсаткичлари. Оқим ўқиш ёки ёзиш позициясини аниқлаш учун ихтиёрий оқим синфида get ёки put кўрсаткичларидан фойдаланилади. Булардан:

• ifstream оқим get кўрсаткичга эга.
• ofstream оқим put кўрсаткичга эга.
• fstream оқим иккала кўрсаткичга эга

Кўрсаткичларни бошқариш учун қуйидаги усуллардан фойдаланилади:

• istream: :tellg(). Файл бошидан get кўрсаткич кўрсатаётган позициягача байтлар сонини қайтаради;
• ostream: :tellp().Файл бошидан put кўрсаткич кўрсатаётган позициягача байтлар сонини қайтаради
• ist ream:: seekg (). оқимда get кўрсаткич холатини ўрнатади.
• ostream: :seekp(). оқимда put кўрсаткич холатини ўрнатади.
• seekg () и seekp(). Кўрсаткич силжиши йўналишини ва катталигини ўрнатади

Силжиш йўналишлари:
ios:beg Силжиш оқим бошидан хисобланади
ios:cur Силжиш оқим жорий позициядан хисобланади
ios:end Силжиш оқим охиридан хисобланади

Қуйидаги мисолда усуллар файл хажмини аниқлаш учун ишлатилади:

```
#include <iostream>
#include <fstream>
using namespace std;
int main(  )
{
int n1, n2;
```

```
ifstream drag("drag.txt", ios::in|ios::binary);
n1 = drag. tellg();
drag.seekg (0, ios::end);
n2 = drag.tellg();
cout<<"Fayl hajmi: " << n2 - n1 << endl;
return 0;
}
```

Натижа:

```
Fayl hajmi: 58
```

Бинар файллар. Бинар файллардан ўқиш ва бинар файлга ёзиш учун `istream::read()` ва `ostream: :write()` усулларидан фойдаланилади.

Қуйидаги дастурда шу усуллардан фойдаланиб `"drag.txt"` файлидан, `"drag2.txt"` файлига нусха олинади.

```
#include <iostream>
#include <fstream>
using namespace std;
int rnain( void )
{
char buffer;
int index = 0;
//файллар номлари
const char filename1[] = "drag.txt";
const char filename2[] = "drag2.txt";
//файлларни очиш
fstream file1(filename1, ios::in);
fstream file2(filename2, ios::out);
//кўрсаткич файл бошига
file1.seekg(0, ios::beg);
file2.seekp(0, ios::beg);
//биринчи символни ўқиш
file1.read(&buffer, 1);
//қолган символларни ёзиш
while(file1.good() && file2.good())
{
file2.write(&buffer, 1);
index++;
file1.seekp(index);
file2.seekg(index);
file1.read(&buffer, 1);
};
//файлларни ёпиш
file1.close();
file2.close();
```

```
return 0;
}
```

Хосил қилинган файл "drag2.txt" мазмуни бўйича "drag.txt" файлидан фарқ қилмайди.

САВОЛЛАР

1. Оқимли синфлар хусусиятларини кўрсатингю
2. Олдиндан яратилган объект оқимларни кўрсатинг.
3. Фойдаланувчи томонидан киритилган типлар учун киритиш ва чикариш.
4. Файллар билан ишлашда қайси библиотекадан фойдаланилади?
5. Файл охирини аниқлаш учун қайси функциядан фойдаланилади?
6. Хатоликни аниқлаш учун қайси функциядан фойдаланилади?
7. Файллар билан ишлаш read ва write функциялари вазифасини кўрсатинг.
8. Файлни очиш режимларини кўрсатинг.
9. Манипуляторлар вазифасини кўрсатинг.
10. Қайси манипуляторлар учун #include <iomanip> улаш зарур?

МАСАЛАЛАР

1. F1 файлдан F2 файлга жуфт сонлардан нусха олинг.
2. F1 файлдан F2 файлга жуфт сатрлардан нусха олинг. F1 ва F2 (байтларда) файллар хажмини хисобланг.
3. F1 файлдан F2 файлга 1 рақамдан бошланувчи сонлардан нусха олинг.
4. F1 файлдан F2 файлга «A» харфдан бошланувчи сўзлардан нусха олинг. F2 файлда сўзлар сонини хисобланг.
5. ФИЛЬМ структураси яратилсин. Структура киритиш ва чиқариш функцияси яратилсин. Дастурда структура типидаги массив киритилиб файлга ёзилсин. Файлга ёзилган маълумотлар массивга ўқилсин. Массив чиқарилсин.

15 боб. Ғайри оддий холатларни дастурлаш

15.1. Ғайри оддий холатлар

Ғайри оддий холатлар дастурда хатони ёки кутилмаган ходисани ифодалайди. Дастур бажарилишида кўзда тутилмаган вазиятга дуч келганда, бошқарувни ушбу муаммони хал қилишга қодир бўлган дастурнинг бошқа қисмига бериши мумкин хамда ё дастурни бажаришни давом эттириш ёки ишни тугаллаш керак.

Ғайри оддий холатларни дастурлаш учун C++ тилида қуйидаги учта хизматчи сўз ишлатилади:

`try` (назорат қилиш)

`catch` (илиб олиш)

`throw` (хосил қилиш, генерация қилиш)

`try` – хизматчи сўзи дастур матни ихтиёрий қисмида назорат қилувчи блок ташқил қилишга имкон беради;

Генераторлар ичида таърифлар эълонлар ва оддий генераторлардан ташқил топади. Ғайри оддий ходисалар хосил қилувчи қуйидаги оператор ҳам ишлатилади:

`throw` ифода.

Бундай оператор бажарилганда махсус четланиш деб аталувчи статик объект хосил қилинади. Бу обект тили ифода тили орқали аниқланади.

Четланишли қайта ишловчилар қуйидаги кўринишга эга бўлади.

```
catch (четланиш тип номи )
{дастурлар}
```

Фигурали қавс ичидаги операторлар четланишларни қайта ишлаш блоки деб аталади. Четланишликларни қайта ишловчи ташқи томондан ва маъно жихатдан қиймат қайтармайдиган бита параметрли функцияга яқиндир. Агар қайта ишловчилар бир нечта бўлса, улар четланиш тиллари фарқ қилишлари лозимдир

C++ ўзи истисно холатларни юзага келтирмайди. Уларни C++ нинг `throw` операторидан фойдаланган дастурлар юзага келтиради. Истисно юзага келганда, `throw` операторидаги ном бериш ифодаси муваққат объектни номлайди (инициаллаштиради), Бунда муваққат объектнинг тури ифода аргументи (далили) нинг турига мос келади. Ушбу объектнинг бошқа нусхалари, масалан, истисно объектидан нусха кўчириш конструктори ёрдамида генерация қилиниши мумкин.

Масалан файл очилишида дастур хато келиб чиқиш шартларини текшириш ва `throw file_open_error()` истисно холатни юзага келтириш мумкин

Қуйидаги try оператори file_copy функцияни чақиришда истисно холатни аниқлаш имконини беради:

```
try
{file_copy("SOURCE.TXT", "TARGET.TXT");
};
```

Қандай истисно холат рўй берганини аниқлаш учун try операторлардан сўнг дастур битта ёки бир нечта catch операторларни жойлаштириш лозим:

```
catch (file_open_error)
{
cerr << "бошлангич ёки максадли файлни очиш
хатолиги"<< endl;
exit(1);
}
```

Бу холда хато типига қарамасдан код хабардор қилади ва дастурни тугатади. Агарда функциянинг чақируви хатосиз бажарилган ва истисно хатолар аниқланмаган бўлса C++ catch операторини шунчаки этиборга олмайди.

Қайта ишловчилар тартиби мухимдир.

```
try
{
   // ...
}
catch (ibuf) { // киритиш буфери тўлишини кайта
ишлаш
}
catch (io) { // киритиш – чикариш хатолигини кайта
ишлаш
}
catch (stdlib) { // библиотекадаги истисно холатни
кайта ишлаш
}
catch (...) {     // колган хамма истисноларни кайта
ишлаш
}
```

Қуйидаги дастур назорат қилувчи блокни ўз ичига олади:

```
#include <iostream>
using namespace std;
int main ( )
{

try
{
```

```
int x = 10;
if (x == 10)
throw x;
}
catch(int x)
{
cerr <<"O yo'q! x teng " << x << "!!!!";
}
catch(float f)
{
cerr << "Nima bo'ldi?";
}
return 0;
}
```

Натижа:

```
O yo'q! x teng 10!!!!
```

Бу дастурда четлашишни генерация қилиш ва қайта ишлаш битта функцияда амалга ошади.

Кейинги мисолда четланиш функция танасидан ташқарида қайта ишланади.

```
#include <iostream>
using namespace std;
void ff(int x)
{
if (x == 10)
throw x;
};
int main ( )
{

try
{
ff(10);
}
catch(int x)
{
cerr <<"O yo'q! x teng " << x << "!!!!";
}
catch(float f)
{
cerr << "Nima bo'ldi?";
}
return 0;
```

```
}
```

Натижа:

```
O yo'q! x teng 10!!!!
```

15.2. Ғайри оддий холатлар синф сифатида

Юқорида келтирилган камчиликдан халос бўлиш учун чекланишни махсус синов объекти сифатида хосил қилиш мумкиндир.

Четланишларни синф сифатида дастурлашни Евклид алгоритм мисолида кўриб чиқамиз. Бу алгоритм икки бутун манфий бўлмаган сонларнинг ЭКУБини топишга мўлжалланган дир. Алгоритм хар бир қадамида қуйидаги амаллар бажарилади.

Агар x>=y бўлса жавоб топилган
Агар x<y бўлса y=y-x
Агар x>y бўлса x=x-y

Қуйидаги мисолда DATA синфи киритилгандир.

```
#include <iostream>
#include <string>

using namespace std;
struct DATA
{
int n,m;
string s;
DATA (int x, int y, string c)
{
n=x; m=y; s=c;
}
};
int GCM_ONE (int x, int y)
{
if (x==0||y==0) throw DATA ( x, y,"\n ZERO!");
if(x<0) throw DATA (x, y, "\n Negative parametr1");
if(y<0) throw DATA (x, y, "\n Negative parametr2");
while (x!=y)
{
if(x>y) x=x-y;
else y=y-x;
}
return x;
```

```
}
int main ( )
{
try
{
cout<<"\n GCM(66,44)="<<GCM_ONE(66,44);
cout<<"\n GCM_ONE(0,7)="<<GCM_ONE(0,7);
cout<<"\n GCM_ONE(-12,8)="<<GCM_ONE(-12,8);
}
catch (DATA d)
{
cerr<<d.s<<" x="<<d.n<<" y="<<d.m;
}
return 0;
}
```

Натижа

```
GCM_ONE(66,44)=22
ZERO! x=0 y=7
```

DATA объекти бу мисолда функция танасида синф конструктори бажарилганда яратилади.

Бу объект чекланиш бўлмаганида чақириш нуқтасида бўлар эди.

Шундай қилиб чегаранишлар синф объекти бўлиши лозимдир. Бу синфларни глобал таърифлаш шарт эмас. Асосий талаб танланиш нуқтасида маълум бщлишир.

Иккита сузувчи сонларнинг бўлинишидан ҳосил бўлган бўлинманинг интерактив ҳисоблагичидан иборат дастурни кўриб чиқамиз. Бу дастурда истиснони тақдим этаётган синфлар иерархияси яратилган.

```
#include <iostream>
using namespace std;

int divide(int a, int b);
class MathError {};
class DivideByZero : public MathError {};
class Overflow : public MathError {};
int divide(int a, int b)
{
if (b == 0)
{
throw DivideByZero();
return false;
}
return a/b;
```

```
};

int main ()
{
try
{
if (!divide(5,0))
throw MathError();
}
catch(DivideByZero)
{
cerr << "Nolga bo'lish qayd etildi!";
}
catch (MathError)
{
cerr << "Qandaydir Matematik hato.";
}
return 0;
}
```

Натижа:

```
Nolga bo'lish qayd etildi!
```

15.3. Ғайри оддий холатлар ва синфлар

Синф яратганда шу синфга хос ғайри оддий холатларни кўрсатиш мумкиндир. Бунинг учун ғайри оддий холатни синфнинг умумий (`public`) элементи сифатида қўшиш лозимдир. Мисол учун қуйидаги `calc` синфи таърифи икки ғайри оддий холатни аниқлайди:

```
#include <iostream>

using namespace std;

class MathError {};
class DivideByZero : public MathError {};
class Overflow : public MathError {};
class calc
{
public:
int divide(int a, int b)
{
if (b == 0)
{
throw DivideByZero();
```

```
return false;
}
return a/b;
};
};
int main ()
{
calc ta;
try
{
if (!ta.divide(5,0))
throw MathError();
}
catch(DivideByZero)
{
cerr << "Nolga bo'lish qayd etildi!";
}
catch (MathError)
{
cerr << "Qandaydir Matematik hato.";
}
return 0;
}
```

Истисно холатнинг маълумотлар элементларидан фойдаланиш

Қуйида кўриб ўтилган мисолда дастур, catch операторидан фойдаланиб, қандай истисно холат рўй берганини ва уларга тегишли холда жавоб беришини имконини беради. Истисно холатга тегишли шундай маълумотни сақлаш учун дастур истисно холат синфига маълумотлар элементларини қўшиш. Агар кейинчалик дастур истисно холатни юзага келтирса, у ушбу маълумотни, қуйида кўрсатилгандек, истисно холатига ишлов берувчи функцияга ўзгарувчи сифатида узатади.

Ғайри оддий холатга ишлов беришда бу параметрлар ғайри оддий холат синфига тегишли ўзгарувчиларга конструктордан фойдаланиб ўзлаштирилади.

Қуйида ким охирги ўйини дастури келтирилган. Ўйин мазхмуни шуки М буюмлардан К дан ортмаган буюмлар танланади. Ким охирги буюмни олса ютади.

```
using namespace std;

class Win
{
int n;
public:
Win(int k) {n=k;};
```

```
int get_number(){return n;}
};

class Error
{
int n;
int k;
public:
Error(int k1,int k2) {n=k1;k=k2;};
int get_number(){return n;}
int get_value(){return k;}
};

class Play
{
int max;
int step;
int number;

public:
Play(int max1,int step1)
{
max=max1;step=step1;number=0;
}
void Step (int k)
{
if((k<1) ||(k>step)) throw(Error(number,k));
if(k>=max) throw(Win(number));
max-=k;
number=1-number;
}
};

int main()
{int k;
int m;
Play pa(10,5);
try
{
while(1)
{
cin>>m;
pa.Step (m);
}
}
```

```
    catch(Win a)
    {
    cout<<"Ura! "<<a.get_number()<<" o'yinchi yutdi";
    cin>>k;
    }
    catch(Error a)
    {
    cout<<a.get_number()<<" o'yinchi "<<a.get_value()<<"
hato yurdi";
    cin>>k;
    }
    return 0;
    }
```

САВОЛЛАР

1. Истисноларни нима учун генерация ва қайта ишлаш керак?
2. Истисно генерацияси синтаксисини келтиринг.
3. Истиснони қайта ишлаш синтаксисини келтиринг.
4. Истиснолар билан қандай операторлар боғлиқ?
5. Истиснони генерация қилувчи функция синтаксисини келтиринг.

МАСАЛАЛАР

1. Истиснолар кўзда тутилган чизиқли тенгамани ечиш функциясини тузинг.
2. Истиснолар кўзда тутилган квадрат тенгамани ечиш функциясини тузинг.
3. Массив синфи шаблони яратилсин. Индекс чегарадан чиққанда истисно генерация қилинсин.
4. Стек синфи шаблонини яратинг. Бўш жой ўчирилганда ёки тўла стекка элемент қўшишга уринилганда истисно генерация қилинсин.
5. Навбат синфи шаблонини яратинг. Бўш жой ўчирилганда ёки тўла навбатга элемент қўшишга уринилганда истисно генерация қилинсин.

16 боб. Кўрсаткичлар ва синфлар

16.1. Объектларга кўрсаткичлар

Структурага кўрсаткичлар. Структурага кўрсаткичлар оддий кўрсаткичлар каби тасвирланади:

```
Complex *cc,*ss; goods *p_goods;
```

Структурага кўрсаткич таърифланганда инициализация қилиниши мумкин. Мисол учун экрандаги рангли нуқтани тасвирловчи қуйидаги структурали тип ва структуралар массиви киритилади. Структурага кўрсаткич қийматлари инициализация ва қиймат бериш орқали аниқланади:

```
Struct point
{int color;
  int  x,y;
} a,b;
 point *pa=&a,pb; pb=&b;
```

Кўрсаткич оркали структура элементларига икки усулда мурожаат қилиш мумкин. Биринчи усул адрес бўйича қиймат олиш амалига асосланган бўлиб қуйидаги шаклда қўлланилади:

(* структурага кўрсаткич).элемент номи;

Иккинчи усул махсус стрелка (->) амалига асосланган булиб куйтдаги куринишга эга:

структурага кўрсаткич->элемент номи

Структура элементларига қуйидаги мурожаатлар ўзаро тенгдир:

```
(*pa).color==a.color==pa->color
```

Структура элементлари қийматларини кўрсаткичлар ёрдамида қуйидагича узгартириш мумкин:

```
(*pa).color=red;
pa->x=125;

pa->y=300;
```

Дастурда нуқтавий жисмни тасвирловчи `particle` структурали типга тегишли `m_point` структураси аниқланган бўлсин. Шу структурага `pinta` кўрсаткичини киритамиз:

```
struct particle * pinta=&m_point;
```

Бу холда `m_point` структура элементларини қуйидагича ўзгартириш мумкин:

```
Pinta->mass=18.4;
for (i=0;i<3;i++)
Pinta->coord[i]=0.1*i;
```

Структураларга кўрсаткичлар устида амаллар. Структураларга кўрсаткичлар устида амаллар оддий кўрсаткичлар устида амаллардан фарқ қилмайди. Агар кўрсаткичга структуралар массивининг бирор элементи адреси қиймат сифатида берилса, массив бўйича узлуксиз силжиш мумкин бўлади.

Мисол тариқасида комплекс сонлар массиви суммасини хисоблаш масаласини кўриб чиқамиз:

```
#include <iostream>
using namespace std;
struct complex
 {
 float x;
 float y;
 };
int main()
{
complex array[]={1.0,2.0,3.0,-4.0,-5.0,-6.0,-7.0,
-8.0};
struct complex summa={0.0,0.0};
struct complex *point=&array[0];
int k,i;
k=sizeof(array)/sizeof(array[0]);
for(i=0;i<k;i++)
{
summa.x+=point->x;
summa.y+=point->y;
point++;
}
cout<<"\n Summa: real="<< summa.x <<
" imag="<<summa.y;
return 0;
}
```

Дастур бажарилиши натижаси:

```
Summa: real=-8.000000, imag=-16.000000
```

Объектларга кўрсаткичлар. Структуралар каби объектлар аниқлангандан сўнг шу объектларга кўрсаткичлар белгилаш мумкин. Масалан:

complex A(5.2,2.7);

complex* PA=&A;

Объектнинг умумий элементларига мурожаат учун -> операцияни ёки исм алмаштириш ва нуқта операциясидан фойдаланиш мумкин

(*PA).real() ёки **PA->real;**

```
#include <iostream>
#include <string>


using namespace std;
class complex
 {
 float x;
 float y;
 public:
 complex(float x1=0,float y1=0):x(x1),y(y1){};
 void show()
{
cout<<"x="<<x<<" y="<<y<<endl;
}
 };
int main()
{
complex a(1.0,2.0);
complex *pa=&a;
pa->show();
complex b;
complex *pb=&b;
(*pb).show();
return 0;
}
```

Дастур бажарилиши натижаси:

```
x=1.0 y=2.0
y=0.0 y=0.0
```

This кўрсаткичи. Агарда конкрет объектга ишлов бериш учун синф аъзоси – функция чақирилса, унда шу функцияга объектга белгиланган кўрсаткич автоматик ва кўрсатилмаган холда узатилади. Бу кўрсатқич **this** исмига эга ва **x* this** каби хар бир функция учун кўрсатилмаган холда белгиланади.

X синфни эквивалент кўринишда шундай тавсифлаш мумкин:

```
class x {
int m;
public:
int readm() { return this->m; }
};
```

Аъзоларга мурожаат этишда this дан фойдаланиш ортиқча. Асосан this бевосита кўрсаткичлар билан манипуляция қилиш учун аъзо функцияларини яратилишида фойдаланилади.

```
#include <iostream>
using namespace std;
class Pair
{
int N;

double x;
friend Pair& operator ++(Pair&);
friend Pair& operator ++(Pair&, int);
public:
Pair (int n, double xn)
{
N=n; x=xn;
}
void display ()
{
cout<<"\n Koordinatalar: N="<<N<<"\tx="<<x;
}
Pair& operator --()
{
N/=10; x/=10;
return*this;
}
Pair& operator --(int k)
{
N/=2;
x/=2.0;
return*this;
}
};
Pair& operator ++(Pair& P)
{
P.N*=10; P.x*=10;
return P;
}
Pair& operator ++(Pair& P, int k)
{
P.N=P.N*2+k;
P.x=P.x*2+k;
return P;
}
int main ()
{
Pair Z(10,20.0);
Z.display();
```

```
++Z;
Z.display();
--Z;
 Z.display();
Z++;
Z.display();
Z--;
Z.display();
return 0;
}
```

Виртуал усуллар ва кўрсаткичлар. Аждод синф кўрсаткичига авлод синф объекти адресини қиймат сифатида бериш ва бу кўрсаткич орқали авлод синф усулларини чақириш мумкин. Қуйидаги мисолда аждод синф ва авлод синф бир хил усулга эга бўлган хол кўрилган:

Мисол:

```
#include <iostream>
using namespace std;
class base
{
public:
virtual void print(){cout<<"\nbase";}
};

class dir : public base
{
public:
void print(){cout<<"\ndir";}
};

int main()
{
base B;
dir D;
base *bp=&B;
bp->print(); // base
bp=&D;
bp->print(); // dir
return 0;
}
```

Натижа:

```
base
dir
```

Бу мисолда авлод синфи объектига адрес қиймат сифатида берилган аждод синф туридаги кўрсаткич ёки илова орқали авлодда қўшимча юкланган усулни чақиришга эътибор бериш лозим. Агар функция новиртуал бўлса аждод синф усули, виртуал бўлса авлод синф усули чақирилади.

Шундай қилиб аждод синф туридаги кўрсаткич орқали виртуал усул чақириш натижаси шу кўрсаткич қиймати яъни чақирик бажарилаётган объект тури билан аниқланади.

Қайси виртуал функцияни чақириш кўрсаткич турига эмас шу кўрсаткич қаратилган(дастур бажарилиш жараёнида) объект турига боғлиқ.

Масалан юқоридаги мисол қуйидагича таърифланиш мумкин

```
#include <iostream>
using namespace std;
class superbase
{
public:
virtual void print()=0;
};
class base:public superbase
{
public:
void print(){cout<<"\nbase";}
};

class dir : public superbase
{
public:
void print(){cout<<"\ndir";}
};
int main()
{
base B;
dir D;
superbase *bp=&B;
bp->print(); // base
bp=&D;
bp->print(); // dir
return 0;
}
```

16.2. Конструктор ва Деструктор

Деструктор. Синфнинг бирор объекти учун ажратилган хотира объект йўқотилгандан сўнг бўшатилиши лозимдир. Синфларнинг махсус компонентаси **деструкторлар**, бу вазифани автоматик бажариш имконини яратади. Деструкторни стандарт шакли қуйидагича:

~ <синф_номи> ( ) {<деструктор танаси>}

Деструктор параметри ёки кайтарилувчи қийматга эга бўлиши мумкин эмас (хатто void типидаги). Дастур объектни ўчирганда деструктор автоматик чақирилади.

Агарда синфда деструктор очиқ кўрсатилмаган бўлса, унда компилятор кўрсатилган объект эгаллаган хотирани бўшатувчи деструкторни генерациялайди. Бошқа объектлар эгаллаган хотирани бўшатмоқчи бўлсак, деструкторни очиқ аниқлаш лозим. Масалан, string объектдаги ch кўрсатган сахифани

```
#include <iostream>
#include <string.h>

using namespace std;

class Person
{
char * name;
int  age;
public:

Person (char* st,int N):age(N)
{
int k;
k=strlen(st);
name= new char[k+1];
strcpy(name,st);
}

Person ()
{
name= NULL;age=0;
}

int GetAge (void)
{
return age;
}
```

```
char * GetName ()
{
return name;
}

~Person()
{
delete[] name;
}

};
int main()
{
Person worker("Happy Jamsa", 51);
cout << "Ism: " << worker.GetName()<< endl;
cout << "Yosh: " << worker.GetAge()<< endl;
}
```

Натижа:

```
Ism: Happy Jamsa
Yosh: 51
```

Нусха олиш конструктори. Кўрсатилмаган холда `T::T(const T&)` кўринишдаги нусха конструктори яратилади, бу ерда `T` – синф исми. Хар гал синфга тегишли объектларни нусхаси олинаётганда нусха конструктори чақирилинади. Хусусан:

а) объект функцияга қиймати бўйича узатилса;

б) функция қийматларини қайтарувчи вақтинчалик объектларни яратишда;

в) бошқа объектни инициализациялаш учун объектдан фойдаланишда.

Агарда синфда аниқ бир нусхалаш конструктори очиқ кўрсатилган бўлмаса, унда юқорида айтиб отилган уч холатдан биттаси мавжуд бўлса объектни нусхалаш бит бўйича амалга оширилади. Лекин бит бўйича нусхалаш хар доим хам адекват деб хисобланмайди. Худди шу холларда хусусий нусхалаш конструкторини белгиламоқ лозим.

Инициализация масаласини хал қилиш учун нусха олиш конструктори киритиш лозим:

```
//Нусха олиш конструктори
Person(const Person& st)
{
int len=strlen(st.name);
name=new char[len+1];
strcpy(name,st.name);
age=st.age;
```

```
}
```

Қиймат бериш ва инициализация. Қиймат бериш ва инициализация турли амаллардир. Айниқса деструктор аниқланганда бу мухимдир. Бирор X типидаги объектни инциализация қилиш нусха олиш конструктори ёрдамида амалга оширилади. Сатр – бу символлар векторига кўрсаткич.

Массив конструктор томонидан яратилиб, деструктор билан ўчирилганда муаммо туғилиши мумкин:

Person s1(10);
Person s2(20)
s1 = s2;

Бу ерда икки символли массив жойлашади, лекин s1 = s2 қиймат бериш натижасида бири ўчирилиб, иккинчиси нусхаси билан алмаштирилади. Функциядан чиқишда s1 ва s2 учун деструктор чақирилади ва битта сатр икки марта ўчирилади. Бу муаммони хал қилиш учун қиймат бериш амалини қўшимча юклаш лозим:

```
// Қиймат бериш амали қўшимча юклаш
Person& operator=(const Person& a)
{
  if (this !=&a) {    // опасно, когда s=s
     delete name;
  int len=strlen(a.name);
     name = new char[len];
     strcpy(name,a.name);
  age=a.age;
  }
  return *this;
}
```

Натижада синф қуйидаги кўринишга келади

```
#include <iostream>
#include <string.h>
using namespace std;
class Person
{
char * name;
int  age;
public:

Person (char* st,int N):age(N)
{
int k;
k=strlen(st);
```

```
name= new char[k+1];
strcpy(name,st);
}

Person ()
{
name= NULL;age=0;
}

//Нусха олиш конструктори
Person(const Person& st)
{
int len=strlen(st.name);
name=new char[len+1];
strcpy(name,st.name);
age=st.age;
}
// Қиймат бериш амали қўшимча юклаш
Person& operator=(const Person& a)
{
  if (this !=&a) {   // опасно, когда s=s
     delete name;
  int len=strlen(a.name);
     name = new char[len];
     strcpy(name,a.name);
  age=a.age;
  }
  return *this;
}

int GetAge (void)
{
return age;
}
char * GetName ()
{
return name;
}

~Person()
{
delete[] name;
}

};
```

```
int main()
{
Person worker("Happy Jamsa", 51);
cout << "Ism: " << worker.GetName()<< endl;
cout << "Yosh: " << worker.GetAge()<< endl;
return 0;
}
```

Натижа:

```
Ism: Happy Jamsa
Yosh: 51
```

Вектор шаблони. Синф объектлари билан ишлаш учун vector қўшимча юкланган шаблон синфи:

```
#include <iostream>
using namespace std;
template<class T>
class vector
{
T* data;
int len;
public:
vector(int k)
{
len =k;
data = new T[len];
}
//Нусха олиш конструктори
vector(const vector& st);
// Қиймат бериш амали қўшимча юклаш
vector& operator=(const vector& a);
T& operator[](int i)
{
return data[i];
}

T& at(int i)
{
if (i<len) return data[i];
}

int size()
{
return len;
}
```

```
    ~vector() {
    delete []data; }

    };

    //Нусха олиш конструктори
    template<class T > vector <T >:: vector(const
vector& st)
    {
    len=st.len;
    data=new T[len];
    for (int i = 0; i < len; i++) data[i]=st.data[i];
    }
    // Қиймат бериш амали қўшимча юклаш
    template<class T > vector<T>& vector<T>::
operator=(const vector& a)
    {
     if (this !=&a) {    // опасно, когда s=s
         delete[] data;
      len=a.len;
         data = new T[len];
        for (int i = 0; i < len; i++)
data[i]=st.data[i];
    }
     return *this;
    };

    template<class T >
    void input_vector(vector<T>& data)
    {
    int n=data.size();
    for (int i = 0; i < n; i++) {cin>>data[i];}
    }
    template<class T >
    void show_vector(vector<T>& data)
    {
    int n=data.size();
    cout<<endl;
    for (int i = 0; i < n; i++){ cout << data[i] << '
';}
    }

    int main()
    {
    vector<int> B(3);
```

```
input_vector(B);
vector<int> C(B);
show_vector(C);
vector<int> D=C;
show_vector(D);
return 0;

}
```

Дастур бажарилишига мисол:

Киритилган маълумот:
1
2
3

Натижа:
1 2 3
1 2 3

16.3. Динамик информацион структуралар

Бир элементли рўйхат. Кўп мисолларда таркиби, хажми ўзгарувчан булган мураккаб конструкциялардан фойдаланишга тугри келади. Бундай ўзгарувчан маълумотлар динамик информацион структуралар деб аталади.

Энг содда динамик информацион структура элементлари қуйидаги структурали тип орқали таърифланган объектлардан иборат бир боғламли рўйхатдир.

Struct структурали тип номи
{
структура элементлари ;
Struct структурали тип номи*кўрсаткич;
};

Бу рўйхат маълумот сақловчи майдонлар, хамда кейинги элемент адресини сақловчи кўрсаткичдан иборат.

Клавиатура оркали ихтиёрий сондаги структураларни бир боғламли рўйхатга бирлашган холда киритиш, сўнгра рўйхат элементларини киритилган тартибда экранга чикариш масаласини кўрамиз.

Рўйхатнинг хар бир бўғинида маълумот ва кейинги элемент адреси жойлашган.Агар бу кўрсаткич ноль қийматга эга бўлса рўйхат охиригача укиб булинган. Рўйхатни куриб чикишни бошлаш учун биринчи элементининг адресини билиш етарлидир. Рўйхатни яратиш функцияси ва рўйхатнинг элементларини экранга чикариш функциясини куриб чикамиз. Рўйхатни тулдириш функцияси қуйидаги прототипга эга:

```
struct cell* input();
```

Бу функция клавиатура оркали киритилган маълумотлар билан тулдирилган рўйхатга кўрсаткич кайтаради. Функцияга хар гал мурожаат килинганда янги рўйхат яратилади. Агар рўйхатнинг навбатдаги структурасининг `weight` ўзгарувчисига ноль қиймат берилса функция уз ишини тухтатади.

Қуйидаги дастур куйилган вазифани бажаради:

```
#include <iostream>
using namespace std;
struct cell
{
char sign[10];
int weight;
struct cell * pc;
};
cell* input()
{
cell * end=NULL;
cell * beg=NULL;
cell * rex;
do
{
rex=new cell();
cout<<"\n sign=";cin>>rex->sign;
cout<<" weight=";cin>>rex->weight;
if (rex->weight==0)
{
   delete rex;
   break;
}
if (beg==NULL&&end==NULL)
end=beg=rex;
else
{
end->pc=rex;
end=rex;
end->pc=NULL;
}
}
while(1);
return beg;
};

void print(cell* beg)
{
```

```
cell * rex;
rex=beg;
while(rex!=NULL)
{
cout<<"\nsign="<<rex->sign;
cout<<"\nweight="<<rex->weight;
rex=rex->pc;
}
};

int main()
{
cell*beg=input();
cout<<"\nRo'yhatni chiqarish:";
print(beg);
return 0;
}
```

Дастур бажарилишига мисол:

Структура хакидаги маълумотни киритинг:

```
Sign=sigma
Weight=16
Sign=omega
Weight=44
Sign=alfa
Weight=0
```

Рўйхатни чикариш

```
Sign=sigma
Weight=16
Sign=omega
Weight=44
```

Дастурда маълумотларни киритиш цикл оркали бажарилади. Цикл тугатилиши шарти навбатдаги структуранинг `int weight` элементига киритилган ноль қийматдир.

Структуралар рўйхати динамик ташкил этилади. `Beg` ва `end` кўрсаткичлар ноль қиймати оркали инициализация қилинган.

Рўйхат тузишда синф шаблонидан фойдаланиш.

```
#include <iostream>
```

```
using namespace std;

struct cell
{
char sign[10];
int weight;
};
template<class T>
struct link
{
T info;
link* next;
link(T a):
info(a){}
};
template<class T>
class que
{
link<T>* beg;
link<T>* end;
public:
que()
{
beg=end=NULL;
};

void push(T a)
{
link<T> * rex;
rex=new link<T>(a);
rex->info=a;
if (beg==NULL&&end==NULL)
end=beg=rex;
else
{
end->next=rex;
end=rex;
end->next=NULL;
}
};

void pop()
{
link<T> * rex;
rex=beg;
```

```
if (beg!=end) beg=beg->next;
else end=beg=NULL;
if (rex!=NULL) delete rex;
};
cell top()
{
return beg->info;
};
bool empty()
{
if(beg==NULL && end==NULL) return false;else return
true;
}
~que()
{
link<T>* rex=beg;
link<T>* temp;
while(rex!=NULL)
{
temp=rex;
rex=temp->next;
delete temp;
}
}
};

template<class T>
void inp(int n, que<T>& a)
{
cell rex;
for(int i=0;i<n;i++)
{
cout<<"\nsign=";cin>>rex.sign;
cout<<"\nweight=";cin>>rex.weight;
 a.push(rex);
}
};

template<class cell>
void print(int n, que<cell>& a)
{
cell rex;
for(int i=0;i<n;i++)
{
rex=a.top();
```

```
cout<<"\nsign="<<rex.sign;
cout<<"\nweight="<<rex.weight;
a.pop();
}
};

int main()
{
que<cell> s;
inp(2,s);
print(2,s);
return 0;
}
```

САВОЛЛАР

1. Объектларга кўрсаткичлар таърифи.
2. Кўрсаткич орқали синф компононталарига мурожаат усуллари.
3. Қандай холларда кўрсаткичидан фойдаланилади.
4. Виртуал усуллар ва кўрсаткичлар.
5. Деструктор вазифаси.
6. Деструкторни нима учун қўшимча юклаб бўлмайди?
7. Нусха олиш конструкторини қандай холда ошкор таърифлаш лозим?
8. Қиймат бериш амалини қандай холда ошкор таърифлаш лозим?
9. Динамик информацион структура деб қандай структурага айтилади?
10. Қўшимча link структураси нима учун киритилган?

МАСАЛАЛАР

1. Абитуриент (исми, туғилган йили, йиғилган балл, аттестат ўрта бали) синфини яратинг. Синфга конструктор, деструктор ва нусха олиш конструктори ва қўшимча юкланган қиймат бериш амали киритилсин.

2. Ходим (исми, лавозими, туғилган йили, ойлиги).

структурасини яратинг. Синфга конструктор, деструктор ва нусха олиш конструктори ва қўшимча юкланган қиймат бериш амали киритилсин.

3. Мамлакат (номи, пойтахт, ахоли сони, эгаллаган майдони).

структурасини яратинг. Синфга конструктор, деструктор ва нусха олиш конструктори ва қўшимча юкланган қиймат бериш амали киритилсин.

4. Навбат синфи шаблонига нусха олиш конструктори ва қўшимча юкланган қиймат бериш амали киритилсин.

5. Стек синфи шаблонини яратинг.

17 боб. График синфлар библиотекасини ишлаб чиқиш

17.1. Матнли режимда экран билан ишлаш

Символли киритиш ва чикариш. Қуйидаги функциялар дастурда символларни киритиш ва чикариш учун ишлатилади.

getch(arg) – битта символ киритилишини кутиш. Киритилаётган символ мониторда акс этмайди. Бу функцияни программа охирида аргументсиз ишлатилса, мониторда маълумотларни то клавиша босилгунча ўкиш мумкин бўлади.

putch(arg)- битта символни стандарт окимга чиқариш учун ишлатилади.Символ мониторда акс этмайди.

getchar(arg) – битта символ киритилишини кутиш. Киритилаётган символ мониторда акс этади. Бу функцияни программа охирида аргументсиз ишлатилса, мониторда маълумотларни то клавиша босилгунча укиш мумкин бўлади.

putchar(arg)- битта символни стандарт окимга чикариш учун ишлатилади.Символ мониторда акс этади.

Бу функциялар **stdio.h** модулида жойлашгандир.

Мисол:

```
#include <stdio.h>
int main()
{
int c;
c=getchar();
putchar(c);
getch();
return 0;
}
```

Экран билан ишловчи функциялар. Қуйидаги функциялар матнли режимда экран билан ишлашга мўлжалланган.

void clrscr(void) – экранни тозалаш

void gotoxy(int x, int y) – курсорни кўрсатилган нуқтага кўчириш

void textcolor(int c) – текст рангини ўрнатиш

void textbackground (int c) – текст фони рангини ўрнатиш

Бу функциялар **conio.h** модулида жойлашгандир.

Экран билан ишлашга мисол.

Дўстона функцияга эга бўлган синфга мисол:

```
#include <conio.h>
class charlocus
{
```

```
int x,y;
char cc;
friend void friend_put (charlocus p, char c);
public:
charlocus (int xi, int yi, char ci)
{
x=xi; y=yi; cc=ci;
}
void display (void)
{
gotoxy(x,y); putch(cc);
}
};
void friend_put(charlocus p, char c)
{
p.cc=c;
}
int main ()
{
charlocus D(20,4, 'd');
charlocus S(10,10, 's');
clrscr( );
D.display( ); getch( ); S.display( ); getch( );
friend_put(D, 'x'); D.display( ); getch( );
friend_put(S, '#'); S.display( ); getch( );
return 0;
}
```

Дастурда D ва S объектлари яратилиб улар учун экранда кординаталар ва `(d,s)` символлари аниқланади. Сўнг синф функцияси `charlocus : : display ( )` символларни кўрсатилган позицияга чиқаради. Глобал `friend_put` функцияси символларнинг ўрнини алмаштириб қўяди.

17.2. График режида экран билан ишлаш

График библиотека. Turbo C ва Borland C++ компиляторларида график библиотека билан боғланиш учун graphic.h – сарлавхали файл қўлланилади.

Бу библиотекага кирувчи баъзи график функциялар:

`void initgraph(int* graphdriver, int* graphmode, char* pathtodriver)`- график режимга ўтказиш

`void closegraph(void )`-график режимдан матнли режимга ўтказиш.

`void putpixel(int x, int y, int color)` - Экранда `color` рангли `(x,y)` кординатали нуқтани тасвирлайди.

`void line (int x1, int y1, int x2, int y2)` - Экранда чизиқ чизади чизади.

`void rectangle (int left, int top, int right, int bottom)` - Экранда тўртбурчак чизади.

`void circle (int x, int y; int radius)` - Экранда айлана чизади.

`void ellipse (int x, int y, int stangle, int endangle, int xradius, int yradius)` - Экранда эллипс чизади.

`void outtextxy (int x, int y, char* textstring)` – Техтни берилган позицияда чиқаради.

`void outtext (char* textstring)` – Техтни жорий позицияда чиқаради.

`int getbcolor(void)` - Фон рангини қайтаради

`int getcolor(void)` - Тасвир рангини қайтаради.

`void getimage (int left, int top, int right, int bottom, void* bitmap)` – экран ойнасини хотирада сақлаш;

`putimage (int left, int top, void* bitmap, int op)`- хотирада сақланган тасвирни экранга жойлаш;

График синфга мисол. Мисол тариқасида нуқта тушунчасини аниқловчи `point` синфини яратиб `point.cpp` файлига ёзиб қўямиз:

```
#ifndef POINTH
#define POINTH 1
class point
{
protected:
int x,y;
public:
point ( int xi=0, int yi=0);
int& givex(void);
int& givey (void);
void show(void);
void move(int xn=0, int yn=0);
private:
void hide();
}
#endif
```

Келгусида `point` синфини бошқа синфларга қўшиш мумкин бўлгани учун шартли процессор директиваси `#ifndef POINTH` ишлатилган. Процессорли идентификатор `POINTH #define POINT 1` директиваси орқали киритилган.

Шунинг учун #include "point.h" директиваси бир неча марта ишлатилганда хам POINT синфи таърифи тексти фақат бир марта компиляция килинаетган файлда пайдо бўлади.

POINT синфи компонента функцияларини қуйидагича таърифлаймиз:

```
ifdef POINTCPP
#define POINTCPP 1
#include <graphics.h>
#include "point.h"
point ::point(int xi=0, int yi=0)
{
x=xi; y=yi;
}
int &point::givex(void)
{
return x;
}
int &point::givey(void)
{
return y;
}
void point::show(void)
{
putpixel(x,y, getcolor());
}
void point::hide (void)
{
putpixel(x,y,getbcolor());
}
void point::move(int xn=0, int yn=0)
{
hide( );
x=xn; y=yn;
show( );
}
#endif
```

Киритилган point синфи ва компонента функциялари билан ишловчи дастурни келтирамиз:

```
#include <graphics.h>
#include <conio.h>
#include "point.cpp"
int main()
{
point A(200,50);
```

```
point B;
point D(500,200);
int dr=DETECT, mod;
initgraph(&dr, &mod, "c:\\borlandc\\bgi");
A.show();getch();
B.show();getch();
D.show();getch();
A.move();getch();
B.move(50,60);getch();
closegraph();
return 0;
}
```

Қўшимча юклаш. Мисол тарикасида '+' амалини point синфи билан таърифланувчи дисплейдаги нуқталарга қўллаймиз. Соддалаштириш учун point синфида энг керакли компаненталарни қолдирамиз.

```
#include <graphics.h>
#include <conio.h>
class point
{
protected:
int x,y;
public:
point (int xi=0, int yi=0)
{
x=xi; y=yi;
}
void show (void)
{
putpixel (x,y,get color ( ) );
};
point operator+ (point p);
};
point point : : operator + (point &p)
{
point d;
d.x=this.x+p.x;
d.y=this.y+p.y;
return d;
}

int main()
{
```

```
int dr=DETECT, mod;
point A(200,50);
point B;
point D(50,120);
INITGRAPH (&dr, &mod, "C\\borlandc \\ bgi");
A.show ( );
getch ( );
B.show ( ); getch ( );
D.show ( ); getch ( );
B=A+D
;
B.show ( ); getch ( );
B=A.operator + (B);
B.show ( ); getch ( );
closegraph ( );
return 0;
}
```

Дастур бажарилиши натижасида дисплей экранига кетма-кет қуйидаги нуқталар қўйилади: `A(200,50); B(0,0); D(50,120); B(250,70), B(450,220)`

Локал синф. Локал синфлардан фойдаланиш хусусиятларини тушунтириш учун қуйидаги масалани кўриб чиқамиз. «Квадрат» синфини аниқлаш керак бўлсин. Квадрат томонларини координаталар ўқига параллел деб қараймиз. Хар бир квадрат берилганлиги сифатида марказ координаталари ва томон узунлиги олинади. Квадрат синфи ичида «кесма локал»синфини аниқлаймиз. Хар бир кесмани берилганлари сифатида учларининг координаталарини оламиз.

Учлари мос равишда олинган тўртта кесма квадратни хосил қилади.Шу усулда квадрат экранда тасвирланади.

```
#include<conio.h>
#include "point.cpp"
class square
{
class segment
{
point pn, pk;
public:
segment(point pin=point(0,0), point  pik=point(0,0))
{
pn.givex()=pin.givex();
pn.givey()=pin.givey();
pk.givex()=pik.givex();
```

```
    pk.givey()=pik.givey();
    }
    point &beg(void)
    {
    return pn;
    }
    point &end (void)
    {
    return pk;
    }
    void showSeg()
    {
    line(pn.givex(), pn.givey(), pk.givex(),
pk.givey());
    };
    };
    segment ab,bc,cd,da;
    public:
    square(point ci=point(0,0), int di=0)
    {
    point a,b,c,d;
    a.givex()=ci.givex()-di/2;
    a.givey()=ci.givey()-di/2;
    b.givex()=ci.givex()+di/2;
    b.givey()=ci.givey()-di/2;
    c.givex()=ci.givex()+di/2;
    c.givey()=ci.givey()+di/2;
    d givex()=ci.givex()-di/2;
    d.givey()=ci.givey()+di/2;
    ab.beg()=a; ab.end()=b;
    bc.beg()=b; bc end()=c;
    cd.beg()=c; cd.end()=d;
    da.beg()=d; da end()=a;
    }
    void showsquare(void)
    {
    ab.showSeg();
    bc.showSeg();
    cd.showSeg();
    da.showSeg();
    }
    };

    int main()
    {
```

```
int dr=DETECT,mod;
initgraph(&dr,&mod,"c:\\borlondc\\ bgi");
point p1(80,120);
point p2(250,240);
square A(p1,30);
square B(p2,140);
A.showsquare(); getch();
B.showsquare(); getch();
closegraph();
return 0;
}
```

17.3. Ворисликка асосланиб график синфларни яратиш

График синфларда ворисликкка мисол. Кейинги мисолда " экрандаги нуқта" асосида " экрандаги дарча " синфи яратилади.

Наслдан ўтувчи компоненталарга қўшимча `spot` синфига қуйидаги компанентаналарни киритамиз: тасвир радиуси (`rad`); экранда кўриниши белгиси (`vir=0` экранда тасвир йук; `vi1==1` экранда тасвир бор); тасвирни битли матнда сақлаш белгиси (`tag=0` тасвир сақланмайди; `tag==1` тасвир сақланади); тасвирни битли матнда сақлаш учун ажратилган хотира қисмига кўрсатгич `pspot`.

```
//Spot.cpp
#ifndef SPOT
#define SPOT  1
#include "point. cpp"
class spot: public point
{
int rad;
int vil;
int tag;
void * pspot;
public:
spot (int xi, int yi, int ri): point (xi, yi)}
{
int size;
vir =0; tag=0; rad=ri;
size=imagesize (xi-ri, yi-ri, xis- ri, yi- ri);
pspot=new char [size];
}
~ spot ()
{
hide();
tag =0;
```

```
delete[] pspot;
}
void show ()
{
if (tag==0)
{
circle (x, y, rad);
floodfill (x, y, getcolor ());
getimage (x-rad, y-rad, y+rad, pspot);
tag=1;
};
else
putimage (x-rad, y-rad, pspot, XOR_PUT);
vis=1;
}
void hide ()
{

if (vis==0) return;
putimage (x-rad,y-rad, pspot, XOR_PUT);
vis=0;
}
void move (int xn, int yn)
{
hide ();
x= xn; y=yn;
show ();
}
viod vary (float dr)
{
float a;
int size;
hide ();
tag=0;
delete pspot;
a=dr*rad;
if (a<=0) rad=0;
else rad= (int) a;
size=imagesize (x-rad; y-rad, x+rad, y+rad);
pspot=new char [size];
show ();
}
int& giver (void);
{
return rad;
```

```
}
};
# endif
```

Spot синфида конструктор, деструктор ~ spot () ва бешта метод кўрсатилган:

show ()-- экранга доирани чизиб, битли тасвирни хотирага олиш;

hide ()-- экрандан доира тасвирини учириш;

move ()--тасвирни экраннинг битта жойига кучириш;

vary ()--экрандаги тасвирни узгартириш (кичкиналаштириш ёки катталаштириш);

giver () --доира радиусига мурожатни таъминлаш;

point синфидан spot синфи наслга нуқта маркази (x,y) координаталарини ва givex, givey методларни олади, point : : show () ва point : : move () методини худди шу номли янги функциялар билан алмаштирилган point :: hide() функцияси номи ўтмайди, чунки point синфида у хусусий (private) статусига эга. spot() конструктори уч параметрга эга -марказ координаталари (xi,yi) ва доира радиуси (ri).

Аввал point синфи конструктори чақирилади бу конструктор xi,yi га мос келувчи хақиқий параметр асосида доира марказини аниклайди. Асосий синф конструктори хар доим хосилавий синф конструкторидан олдин чаакирилади. Сунгра spot() синфи конструктолари бошланади. Бу конструктор vis, tag параметрларининг бошлангич қийматини аниклайди ва ri га мос келувчи хақиқий параметр қиймати асосида доира радиуси red аникланади. Стандарт функция imagesize ёрдамида доира жойлашувчи квадратик оператив хотирада аниқлаш учун зарур булган хотира хажми хисобланади. Керакли хотира new стандарт операция ёрдамида ажратиб size элементидан иборат char массивлар ёзилади. Агар айтилган хотира spot синфида protected статутисига эга булган pspot кўрсаткичига уланади.

Варисликда деструкторлар хоссалари. Синфнинг хар бир объекти яратилганда синф конструктори чақирилиб, объект учун керакли хотира яратиш ва инициализация қилиш вазифаларини бажаради. Объект йукотилганда ёки синф таъсир доирасидан ташқарига чиқилганда тескари вазифалар бажариш керак бўлиб, булар ичида энг кераклиси хотирани озод килишдир. Бу вазифаларни бошкариш учун синфга махсус функция деструктор киритилади. Деструктор қуйидаги шаклга эга бўлган аник номга эга ~ синф-номи.

Деструктор хатто void типидаги параметрларга эга бўлмайди ва хатто void типидаги қиймат қайтармайди. Деструктор статуси алохида эълон қилинмаган бўлса умумийдир.

Содда синфларда деструктор автоматик аникланади, мисол учун `paint` синфида деструктор эълон килинмаган ва компилятор қуйидаги деструкторни автоматик чакиради

```
~ point () {};
```

spot синфида деструктор аниқ кўринишга эга;

```
~ spot () {hide (); tag=0;delete [] pspot;}
```

Бу деструктор вазифалари доира тасвирини `spot::hide()` функцияси оркали ўчириш; `tag` белгисига 0 қийматини бериш; объект битли тасвирни саклаш учун ажратилган хотирани тозалаш.

Деструкторлар наслга ўтмайди, шунинг учун хосилавий синфда деструктор мавжуд бўлмаса асосий синфдаги деструктор чақирилмайди. Балки компилятор томонидан яратилади. Кўрилаётган мисолда қуйидагича:

```
public: ~spot () {~point ();}
```

Асосий синфлар деструкторлар рўйхатда кўрсатилганидек тескари тартибда бошқарилади. Шундай қилиб объектларни ўчириш тартиби яратилиш тартибига тескаридир. Агар объект яратилганда дастурда хотира ажратилган булса деструктор дастурда чақирилиши лозим.

Spot синфи объектлари билан ишловчи дастурни келтирамиз:

```
#include<graphics.h>
#include<conio. h >
#include "spot. cpp"
int main()
{
int dr=DETECT, mod;
initgraph (&dr, &mod);
{
spot A(200,50,20);
spot D(500,200,30);
A.show();
getch ();
D.show ();
getch();
A.move(50,60);
getch ();
}
closegraph();
return 0;
}
```

График абстракт синф ва унинг ворисларига мисол. Қуйидаги мисолда абстракт синфлар умумий тушунчаларни тавсифлаш учун ишлатилади `point` синфи вориси сифатида `figure` абстракт синфи яратилади. Бу синфда конструктор, соф виртуал функция `show ()`, хамда

hide() ва move() методлари аниқланган. Дастурда figure синфи асосида иккита авлод синф circle (айлана) ва ellips (элипс)синфлари аниқланади.

Иккала синфда point синфидан шакллар марказлари координаталари наслга ўтган. Иккала синфда конкрет show() метод аниқланган ва figure () абстракт синфидан move() ва hide() функциялари наслга ўтган.

```
// figure.cpp абстракт синф
# include "point.cpp"
class figure: public point
{
public :
//figure  синф конструктори
figure (point p) :point (p.give x(), p.give y()){}
// соф виртуал функция
virtual void show() = 0;
// фигурани ўчириш функцияси
void hide()
{
int bk,cc;
bk = getbkcolor();
cc = getcolor();
setcolor(bk);
show();
setcolor(cc);
}
// фигурани харакатлантириш функцияси
void move (point p)
{
hide ();
x=p.givex();y=p.givey();
show();
}
};
```

figure абстракт синф асосида конкрет синфлар яратамиз;

```
// ELLIPS CPP
class ellips: public figure
{
int rx, ry;
public:
// конструктор
ellips(point d, int radx, int rady): figure (d)
{
rx = radx; ry = rady;
```

```
}
void show()
{
ellipse (x,y,0, 360, rx, ry);
return;
}
};

// circ.fig  айлана синфи
class circ: public figure
{
int radius;
public:
// конструктор
circ (point e, int rad): figure (e)
{
radius=rad;
}
void show () {
circle(x,y,radius);
}
};
```

қуйидаги дастурда учта синф хаммаси ишлатилган;

```
# include <graphics.h>
# include "figure.cpp"
# include "circ.fig"
# include "ellips.fig"
# include <conio.h>


int main ()
{
point A(100,80), B(300,200);
circ C(A,60);
ellips E(B,200,100);
{
int dir = DETECT, mod;
initgraph (&dir, &mod, "c\\borlandc\\ bgi");
A.show (); getch();
B.show (); getch();
C.show (); getch();
E.show (); getch();
C.move(B); getch();
E.hide(); getch();
```

```
C.hide(); getch();
}
closegraph();
return 0;
}
```

САВОЛЛАР

1. Символли киритиш функциялари.
2. Символли чиқариш фугкциялари
3. Матнли режимда экранни тозалаш функцияси
4. Матнли режимда курсорни ўрнатиш функцияси.
5. График функциялар қайси библиотекада сақланади.
6. График режимга ўтиш командаси.
7. График режимдан чиқиш командаси
8. Экранда чизиқ чизиш функцияси
9. Экранда айлана ва эллипс чизиш функцияси.
10. Экран қисмини қандай қилиб хотирада сақлаш мумкин?

МАСАЛАЛАР

1. Матнли режимда ўз исми шарифингизни форматланган шаклда чиқаринг. Исм шариф синфини яратинг.
2. Матнли режимда чопувчи қатор синфини яратинг.
3. График режимда ўз исми шарифингизни форматланган шаклда чиқаринг. Исм шариф синфини яратинг.
4. График режимда чопувчи қатор синфини яратинг яратинг.
5. Ворислик асосида турли учбурчаклар синфлари оиласини яратинг.

АДАБИЁТ

Асосий адабиёт:

1. Гради Буч. Объектно –ориентированной анализ и проектирование с примерами приложений на С++. Невский диалект, 2001 г., 560 с.,
2. Грехем И. Объектно ориентированные методы. Принципы и практика. Вильямс., 2004, 879 с.,
3. Иванова Г.С. Объектно ориентированное программирование. Учебник., МГТУ им Баумана, 2003, 320 с.
4. Ашарина Н.А. Основы программирования на языках Си, С++. Учебный курс., М.: 2002
5. Шмидский Я.К. Прораммирование на языке С++: Самоучитель. Учебное пособие., Диалектика, 2004 г., 361 с.
6. Страуструп Б. Язык программирования С++. Третье издание, М.: Бином, 1999
7. Пол Айра. Объектно-ориентированное программирование на С++. Второе издание. – М.: Бином, 1999.
8. Аммераль Л. STL для программистов на С++., М: ДМК, 1999.

Қўшимча адабиёт:

1. Крупник А.Б. Изучаем С++. Питер. 2003, 251 с.
2. Мейерс С. Наиболее эффективное использование С++. 35 новых рекомендаций. ДМК-Пресс, 2000, 304 с.
3. Николаенко Д.В. Самоучитель по Visual С++., Спб, 2001.
4. Элджер Дж. С++: библиотека программиста, СПб: Питер, 1999.
5. Либерти Д. Освой самостоятельно С++: 10 минут на урок. Пер с англ. Вильямс, 2004, 374 с.
6. Шилдт Г. Самоучитель С++. Второе издание, СПб.: BHV, 1998.
7. Луис Д. С и С++. Справочник., М: Бином, 1997.
8. Подбельский В.В. Язык С++ – М.: Финансы и статистика, 1996.
9. Фейсон Т. Объектно-ориентированное программирование на С++ 4.5. – Киев: Диалектика, 1996.
10. Шилдт Г. Теория и практика С++., СПб.: BHV, 1996.
11. Керниган Б., Ритчи Д. Язык программирования Си. М. Финансы и статистика. 1985.

МУНДАРИЖА

№	Мавзулар	
	Кириш	3
1.	Тил лексик асослари	4
2.	Дастурларнинг таркибий қисмлари	22
3.	Операторлар	39
4.	Функциялар хоссалари	57
5.	Массивлар ва сатрлар	75
6.	Структуралар	93
7.	Кўрсаткичлар, массивлар, функциялар	103
8.	Препроцессор воситалари	119
9.	Объектга мўлжалланган дастурлаш асослари	135
10.	Таянч синфлар	149
11.	Синфлар орасида муносабатлар	167
12.	Ворислик	179
13.	Синфларда полиморфизм	197
14.	Оқимли синфлар	215
15.	Ғайри оддий холатларни дастурлаш	231
16.	Кўрсаткичлар ва синфлар	240
17.	График синфлар библиотекасини ишлаб чиқиш	258
	Адабиёт	272
	Мундарижа	273