C++
ТИЛИГА КИРИШ

услубий қўлланма

Ифодалар
Функциялар
Синфлар
Объектлар
Курсатгичлар
Хаволалар

Ушбу услубий қўлланмадан B5480100 – «Амалий математика ва информатика», B5521900 – «Информацион ва ахборот технологиялари» бакалавриат йўналишлари учун «Алгоритмлар назарияси ва дастурлаш технологияси» фанини ўқитиш бўйича фойдаланиш тавсия этилади.

Услубий қўлланмада C++ тили ва объектга йўналтирилган дастурлаш технологияси ҳақида батафсил маълумотлар берилган.

Тузувчи:

Тақризчилар:

Кириш

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. Биз ушбу мавзуда қуйидаги саволларга жавоб оламиз:

- Нима учун C++ тили дастурий маҳсулотларни ишлаб чиқиш соҳасида стандарт бўлиб қолди?
- C++ тилида дастур ишлаб чиқишни қандай ўзига хос томонлари бор?
- C++ тилида дастур қандай ёзилади ва компиляция қилинади?

C++ тили тарихи

Биринчи электрон ҳисоблаш машиналари пайдо бўлиши билан дастурлаш тиллари эволюцияси бошланади. Дастлабки компьютерлар иккинчи жаҳон уруши вақтида артиллерия снарядларининг ҳаракат траекториясини ҳисоб-китоб қилиш мақсадида қурилган эди. Олдин дастурчилар энг содда машина тилини ўзида ифодаловчи компьютер командалари билан ишлаганлар. Бу командалар нол ва бирлардан ташкил топган узун қаторлардан иборат бўлар эди. Кейинчалик, инсонлар учун тушунарли бўлган машина командаларини ўзида сақловчи (масалан, ADD ва MOV командалари) ассемблер тили яратилди. Шу вақтларда BASIC ва COBOL сингари юқори сатҳли тиллар ҳам пайдо бўлдики, бу тиллар туфайли сўз ва гапларнинг мантиқий конструкциясидан фойдаланиб дастурлаш имконияти яратилди. Бу командаларни машина тилига интерпретаторлар ва компиляторлар кўчирар эди. Интерпретатор дастурни ўқиш жараёнида унинг командаларини кетма - кет машина тилига ўтказади. Компилятор эса яхлит программа кодини бирор бир оралиқ форма - объект файлига ўтказади. Бу босқич компиляция босқичи дейилади. Бундан сўнг компилятор объектли файлни бажарилувчи файлга айлантирадиган компановка дастурини чақиради.

Интерпретаторлар билан ишлаш осонроқ, чунки дастур командалари қандай кетма - кетликда ёзилган бўлса шу тарзда бажарилади. Бу эса дастур бажарилишини назорат қилишни осонлаштиради. Компилятор эса компиляция ва компановка каби қўшимча босқичлардан иборат бўлганлиги учун улардан ҳосил бўладиган бажарилувчи файлни таҳлил қилиш ва ўзгартириш имконияти мавжуд эмас. Фақатгина компиляция қилинган файл тезроқ бажарилади, чунки бундаги командалар компиляция жараёнида машина тилига ўтказилган бўлади.

C++ каби компиляция қилувчи дастурлаш тилларини яна бир афзаллиги ҳосил бўлган дастур компьютерда компиляторсиз ҳам бажарилаверади. Интерпретация қилувчи тилларда эса тайёр дастурни ишлатиш учун албатта мос интерпретатор дастури талаб қилинади.

Айрим тилларда (масалан, VISUAL BASIC) интерпретатор ролини динамик библиотекалар бажаради. Java тилининг интерпретатори эса виртуал машинадир (Virtual Machine,

ёки VM). Виртуал машиналар сифатида одатда броузер (Internet Explorer ёки Netscape) лар қўлланилади.

Кўп йиллар давомида дастурларнинг асосий имконияти унинг қисқалиги ва тез бажарилиши билан белгиланиб келинар эди. Дастурни кичикроқ қилишга интилиш компьютер хотирасини жуда қимматлиги билан боғлиқ бўлса, унинг тез бажарилишига қизиқиш процессор вақтининг қимматбаҳолигига боғлиқ эди. Лекин компьютерларнинг нархи тушиши билан дастур имкониятини баҳолаш мезони ўзгарди. Ҳозирги кунда дастурчининг иш вақти бизнесда ишлатиладиган кўпгина компьютерларнинг нархидан юқори. Ҳозирда профессионал тарзда ёзилган ва осон эксплуатация қилинадиган дастурларга талаб ошиб бормокда. Эксплуатациянинг оддийлиги, конкрет масалани ечиш билан боғлиқ бўлган талабни озроқ ўзгаришига, дастурни ортиқча чиқимларсиз осон мослаштириш билан изоҳланади.

Дастурлар

Дастур сўзи ҳам командаларнинг алоҳида блокини (берилган кодини) аниқловчи сўз, ҳам яхлит ҳолдаги бажарилувчи дастурий маҳсулотни белгиловчи сўз сифатида ишлатилади.

Бу икки хиллилик ўқувчини чалғитиши мумкин. Шунинг учун унга аниқлик киритамиз. Демак дастурни ё дастурчи томонидан ёзиладиган командалар тўплами, ёки амаллар бажарадиган компьютер маҳсулоти сифатида тушинамиз.

Дастурчилар олдида турган масалалар

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.

Фойдаланувчиларнинг ушбу янги авлодини дастурлар билан ишлашларини осонлаштирилиши билан бу дастурларнинг ўзини мураккаблиги даражаси ошади. Замонавий дастурлар - фойдаланувчи билан дўстона муносабатни юқори даражада ташкил қиладиган кўп сондаги ойналар, меню, мулоқот ойналари ва визуал графикавий муҳитлардан таркиб топган интерфейсга эга бўлиши лозим.

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

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.

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.

1. Ҳар бир ходим ҳақидаги ёзувни ўқиймиз.
2. Иш хақи тўғрисидаги маълумотни оламиз.
3. Иш ҳақи қийматини йиғиндига қўшамиз.
4. 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 мувофаққиятли 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 ташкил 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 қайтам 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.

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 методлар билан бирлаштириш имконияти мавжуд.

Объектга мўлжалланган дастурлашнинг асосий мақсади берилганлар ва улар устида амал бажарувчи процедураларни ягона объект деб қарашдан иборатдир.

C++ тили ва объектларга мўлжалланган дастурлаш.

C++ тили объектга мўлжалланган дастурлаш принципларини қўллаб қувватлайди. Бу принциплар қуйидагилардир:

- Инкапсуляция
- Меросхўрлик
- Полиморфизм

Инкапсуляция.

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.

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.

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.

C++ тилида инкапсуляция принципи синф деб аталувчи ностандарт типларни(фойдаланувчи типларини) ҳосил қилиш орқали ҳимоя қилинади.

Синфлар қандай тузилишга эга эканлиги билан кейинроқ танишиб чиқамиз.

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.

Меросхўрлик

`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.

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.

Полиморфизм.

Акселаторни 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.

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.

ANSI стандарти

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 ташкилот) стандарти деб ҳам номланади.

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 боғлиқдир. 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.

САВОЛЛАР

1. Интерпретатор ва компилятор орасидаги фарқ нимадан иборат?
2. Дастурнинг берилган коди қандай компиляция қилинади?
3. Структуравий дастурлаш ва объектга мўлжалланган дастурлашнинг фарқи нимадан иборат?
4. Объектга мўлжалланган дастурлаш принципларини тушунтириб беринг.

ТАЯНЧ ИБОРАЛАР

Дастур, структуравий дастурлаш, объект, объектга мўлжалланган дастурлаш, инкапсуляция, меросхўрлик, полиморфизм.

C++ тилидаги дастурларнинг таркибий қисмлари.

C++ тилида тузилган дастур объектлар, функциялар, ўзгарувчилар ва бошқа элементлардан ташкил топади. Ушбу мавзунинг асосий қисми уларнинг ҳар бирини тўлиқ тавсифлашга бағишланган. Лекин бу элементларни уйғунлашган ҳолда қараш учун бирор бир тугалланган ишчи дастурни қараб чиқиш керак. Ушбу мавзудан қўйидагиларни билиб олиш мумкин.

- C++ тилида дастурлар қандай қисмлардан тузилган.
- Бу қисмлар бир – бири билан қандай алоқа қилади.
- Функция нима ва унинг вазифаси нимадан иборат.

C++ тилида оддий дастур.

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.

2.1. – листинг. `SALOM.CPP` дастури мисолида C++ тилида тузилган дастур қисмларини намойиш қилиш.

```
1: // Salom.CPP dasturi
2: #include <iostream.h >
3:
4: int main( )
5: {
6:  cout << “Salom!\n”;
7:  return 0;
8: }
```

НАТИЖА:

```
Salom!
```

ТАҲЛИЛ

1 – сатрда `iostream.h` файли жорий файлга бириктиляпти. Дастурда биринчи фунта (#) белгиси жойлашган. У препоцессорга сигнал узатади. 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.h` файлини сизнинг компиляторингиз учун мулжалланган `.h` кенгайтмали файлларни ўзида сақловчи папкадан излаши кераклигини кўрсатади. `iostream.h` (`input – output stream` – киритиш–чиқариш оқими) файлида экранга маълумотларни чиқариш жараёнини таъминлайдиган `cout` объекти аниқланган. Биринчи қатор бажарилгандан сўнг `iostream.h` 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. `main()` функцияси фигурали қавсида 3–сатрдан 6–сатргача жойлаштирилган. Фигурали қавсларни ичида жойлашган барча сатрлар функция танаси деб айтилади.

Бизнинг оддий дастуримизнинг барча функционаллиги 4–сатрда келтирилган. `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. `main()` функцияси 6 – сатр билан тугалланади.

`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 бирта оператор деб қарайди. Қуйидаги листингни таҳлил қиламиз.

2.2. – листинг. `cout` объектини қўлланилиши.

```
1: //2.2.-листинг. cout объектини қўлланилиши
2: # include  <iostream.h >
```

```
3: int main()
4: {
5: cout << “Bu son 5 ga teng:” << 5<< “\n”;
6: cout <<“endl operatori ekranda yangi”;
7: cout << “ satrga o`tish amalini bajaradi”;
8: cout <<endl;
9: cout << “Bu katta son:\t“<< 70000 <<
10:endl;
11:cout << “Bu 5 va 8 sonlarining yig`indisi:
12:<<\t“<< 8+5 << endl;
13:cout << “Bu kasr son:\t \t“<< (float) 5\8
14:<< endl;
15:cout << “Bu esa juda katta son: \t”;
16:cout << (double) 7000*7000 << endl;
17:return 0;
18:};
```

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
```

Изох

Айрим компиляторларда cout объектидан кейин математик операцияларни бажариш учун фигурали қавсларни ишлатиш талаб қилинади. У ҳолда 2.2. – листингнинг 11 – сатрида қуйидагича алмаштириш бажариш лозим.

```
11: cout << «Here is the sum
of 8 and 5<< (8+5) << endl;
```

Изох

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.

2.3. – листинг. `Salom.cpp` дастури мисолида изоҳларни намойиш қилиш.

```
1: # include < iostream.h >
2:
3:  main()
4:  {
5:   cout << "Salom!\ n";
6: /* бу изох токи изохнинг
7: охирини кўрсатувчи белги, яъни  юлдузча
8: ва слэш белгиси учрамагунча давом этади */
9: cout << "Bu kommentariy tugadi\ n";
10:// бу изох сатрни охирида тугайди.
11:// Иккита  слэшдан сўнг хеч кандай текст
12:// булмаслиги мумкин.
13:return 0;
14:}
```

НАТИЖА

```
Salom
Bu kommentariy tugadi
```

Функциялар

Биз олдинроқ `main()` функцияси билан танишиб чиққан эдик. Бу функция одатдаги бўлмаган, ягона турдаги функциядир. Функциялар дастурнинг ишлаш даврида чақирилиши керак. `main()` функцияси эса дастур томонидан эмас, балки операцион система томонидан чақирилади.

Дастур берилган матни буйича сатрларни жойлашишига қараб тартиб билан токи бирор бир функция чақирилгунча бажарилади. Кейин эса бошқарув биринчи учраган функцияга берилади. Функция бажарилгандан сўнг бошқарув яна дастурнинг функция чақирилган жойдан кейинги сатрига берилади. (Чақирилган функциядан кейинги сатрга берилади.)

Функцияни ишлаш жараёнига мос ўхшашлик мавжуд. Масалан, сиз расм чизиб турган вақтингизда қаламингиз синиб қолди. Сиз расм чизишни тўхтатасиз ва қаламни йўна бошлайсиз. Кейин эса, расм чизишни қаламингиз синиб қолган жойдан бошлаб давом эттирасиз. 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 листингда намойиш этилган.

2.4. – листинг . Функцияни чақирилишига мисол.

```
1: # include  < iostream.h >
2: //NamoyishFunktsiyasi  функцияси экранга
3: // ахборий маълумот чикаради.
4: void NamoyishFunktsiyasi()
5: {
6: cout<< "\nNamoyishFunktsiyasi chaqirildi\ n;
7: }
8: //main() функцияси олдин ахборот чикаради ва
9: // NamoyishFunktsiyasi функциясини чакиради
10:// Кейин яна намойиш функциясини чакиради
11:int main()
12:{
13: cout << "\ n Bu main() funktsiyasi \ n";
14:NamoyishFunktsiyasi();
15:cout << "main() funktsiyasiga qaytildi\n";
16:return 0;
17:   }
```

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.

Изох

```
   5 – мавзуда функциялар хакида
батафсилрок маълумотлар
берилади. Бу мавзуда берилган
```

машгулотлар умумий бўлиб, факатгина кейинги мавзуларни ўзлаштириш учун етарлидир.

2.5. – листингда иккита бутун сонли параметрни қабул қилувчи ва бутун сонли қиймат қайтарувчи функция намойиш қилинган.

2.5. – листинг. Оддий функцияни ишлатишга мисол

```
1: # include < iostream.h >
2: int Qushish(int x,int y)
3: {
4:  cout <<x<<" va "<<y<< "sonlarini Qushish()"
5:  << "funktsiyasi orqali qushdik\n";
6:  return (x + y) ;
7: }
8:
9:  int main()
10:{
11: cout <<"Biz main() funktsiyasidamiz!\n" ;
12: int a, b, c ;
13: cout << "Ikkita son kiriting:" ;
14: cin >> a ;
15: cin >> b ;
16:cout <<"\nQushish()funktsiyasi chaqirildi";
17:<<endl;
18: c = Qushish(a,b)
19: cout <<"\n main() funktsiyasiga qaytildi."
20:<<endl;
21: cout << "c ning qiymati" << c<< "ga teng"
22: << endl;
23: cout << " \n Ishni tugatish … \ n \ n " ;
24: return 0;
25:}
```

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 2 – сатрда аниқланган. У иккита бутун сонли параметрни қабул қилади ва бутун сонли қиймат қайтаради. Дастурнинг ўзи 9 – сатрдан бошланади, ва у экранга биринчи хабарни чиқаради. Кейин фойдаланувчига иккита сонни киритиш ҳақида хабар берилади. (13 – 15 – сатрлар). Фойдаланувчи пробел орқали ажратган ҳолда иккита сон киритади. Кейин эса `Enter` тугмасини босади. 17 – сатрда `main()` функцияси `Qushish()`функциясига фойдаланувчи томонидан киритилган иккита сонни параметр сифатида узатади.

Дастур бошқаруви 2 – сатрдан бошланувчи `Qushish()` функциясига ўтади. `a` ва `b` параметрлар экранга чиқарилади ва қўшилади. Функция 6 – сатрда натижасини қайтаради ва ўз ишини якунлайди.

САВОЛЛАР

1. Компилятор ва препроцессорнинг фарқи нимадан иборат?
2. `#include` директиваси қандай вазифани бажаради.
3. `main()` функциясининг ўзига хос хусусияти нимадан иборат?
4. Қандай изоҳ турларини биласиз ? Улар нима билан фарқ қилади?
5. Изоҳлар бир неча қаторда ёзилиши мумкинми?

ТАЯНЧ ИБОРАЛАР

Изоҳ, функция, функция параметри, функция аргументи, функция танаси

Ўзгарувчилар ва ўзгармаслар

Дастур ўзи ишлатадиган маълумотларни сақлаш имкониятига эга бўлиши лозим. Бунинг учун ўзгарувчилар ва ўзгармаслардан фойдаланилади. Ушбу мавзуда қуйидагиларни билиб оламиз.

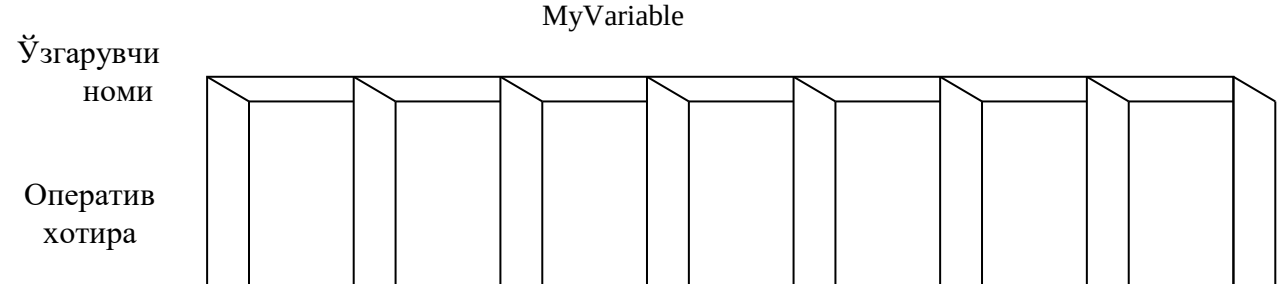
- Ўзгарувчи ва ўзгармасларни қандай аниқлаш керак.
- Ўзгарувчиларга қандай қиймат бериш керак ва уларни дастурда қандай ишлатиш лозим.
- Ўзгарувчи қиймати қандай экранга чиқарилади.

Ўзгарувчи нима?

C++ тилида ўзгарувчилар маълумотни сақлаш учун қўлланилади. Ўзгарувчининг дастурда фойдаланиш мумкин бўлган қандайдир қийматларни сақлайдиган комьютер 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. Бу ғоянинг схематик ифодаси 3.1. – расмда тасвирланган.



3.1. – расм. Ўзгарувчиларни хотирада сақланиши.

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

Хотирани заҳиралаш.

C++ тилида ўзгарувчини аниқлаш учун компьютерга унинг типи (масалан, `int`, `char` ёки бошқа) ҳақида маълумот берилади. Бу ахборот асосида компиляторга ўзгарувчи учун қанча жой ажратиш лозим ва бу ўзгарувчида қанақа турдаги қиймат сақланиши мумкинлиги ҳақида маълумот аниқ бўлади.

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.

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. 3.1. – листингда келтирилган дастур сизнинг компютерингиздаги типларнинг ўлчовини аниқлаб беради.

3.1. – листинг. Таянч типлар учун компютер хотирасидан ажратиладиган байтларни аниқлаш.

```
1: # include  <iostream. h>
2: int main()
3: {
4: cout << “int tipining o`lchami: \t”
5: <<sizeof(int)<<” bayt.”;
6: cout<< “short int  tipining o`lchami:\t”
7: << sizeof(short)<< “bayt.”;
8: cout<<“long int tipining o`lchami:\t”
9: << sizeof(long)<<“bayt.”;
10:cout<< char tipining o`lchami:\t”
11:<< sizeof(long)<<”\t bayt.”;
12:return 0;
13:};
```

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;
```

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.

Ўзгарувчиларнинг таянч типлари.

C++ тилида бошқа берилганлар типлари ҳам қаралган. Улар бутун сонли, ҳақиқий ва белгили бўлиши мумкин. Ҳақиқий ўзгарувчилар каср кўринишда ифодаланувчи қийматларни ҳам ўзида сақлайди. Белгили ўзгарувчилар бир байт жой эгаллайди ва 266 та белги ҳамда `ASCII` белгиларни сақлаш учун ишлатилади.

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.

Калит сўзлар

C++ тилида айрим сўзлар олдиндан заҳираланади. Булар калитли сўзлар деб айтилади. Бундай сўзларни ўзгарувчиларни номлашда ишлатиш мумкин эмас. Уларга if, while, for ва main каби сўзлар киради. Компиляторнинг техник документациясида барча заҳираланган сўзларнинг руйхати туради.

Ўзгарувчига қиймат бериш

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 листингда ўзгарувчиларни аниқлашга оид мисолни қараймиз.

3.2. – листинг. Ўзгарувчиларнинг қўлланиши.

```
1: # include < iostream. h >
2: int main()
3: int Buyi=5, Eni=10,Yuzasi;
4: cout << “Bo’yi:” << Buyi <<  “ \n”;
5: cout << “Eni:” << Eni << endl;
```

```
6: Yuzasi= Buyi*Eni;
7: cout << “Yuzasi:” << Yuza << endl;
8: return 0;
}
```

НАТИЖА:

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

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 булса, шу жойда ишлатилади.

3.3. – листинг . **typedef** оператори орқали типларнинг аниқланиши

```
1: #include <iostream.h>
2: typedef  insigned short int ushort
3: int main()
4: { ushort Buyi = 5;
5: ushort Eni = 10;
6: ushort Yuzasi = Buyi* Eni;
7: cout << “Yuzasi:” << Yuzasi << end;
8: }
```

НАТИЖА:

```
Area: 50
```

Белгилар.

Белгили ўзгарувчилар 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.

Махсус белгилар.

C++ компилятори текстларни форматловчи бир нечта махсус белгилардан ташкил топган. (Улардан энг кўп тарқалгани 3.2.- жадвалда келтирилган). Бу белгиларни дастурда ишлатишда «тескари слеш»дан фойдаланамиз. Тескари слешдан кейин бошқарувчи белги ёзилади. Масалан, табуляция белгиини 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.

3.2. жадвал.

Белгилар	Қиймати
\n	Янги сатрга ўтиш
\t	Табуляция
\b	Битта позицияга ўтиш
\”	Иккиталик қавсча
\’	Битталик қавсча
\?	Сўроқ белгиси
\\	Тескари слеш

Ўзгармаслар

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 белгили ўзгармаслар аниқланган.

Литерал ўзгармаслар

Литералли ўзгармаслар тўғридан-тўғри дастурга киритилади. Масалан:

```
int myAge =39;
```

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

Белгили 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.

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. Масалан, RANG номли санокли тип деб эълон қилайлик ва унинг учун 5 та QIZIL, KUK, YASHIL, OQ, QORA қийматларини аниқлайлик.

Санокли типларни ҳосил қилиш учун enum калитли сўзи ва ундан кейин тип номи ҳамда фигурали қавс ичида вергуллар билан ажратилган ўзгармас қийматлари рўйхати ишлатилади. Масалан,

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

Бунда ифода иккита ишни бажаради:

1.RANG номли янги санокли тип ҳосил қилади;

2.Қуйидаги белгили ўзгармасларни аниқлайди.

0 қиймат билан QIZIL;

1 қиймат билан KUK;

2 қиймат билан YASHIL ва ҳоказо;

Ҳар бир санокли 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 санокли типни аниқласак QIZIL ўзгармаси 100 га, KUK – 200 га, Yashil – 300 га, OQ –301 га, *QORA* – 500 га тенг бўлади.

3.7. – листинг . Санокли ўзгармасни қўлланиши

```
1: # include <iostream.h >
2: int main( )
3: {
4: enum Kunlar{Dushanba, Seshanba, Chorshanba,
5: Payshanba, Juma, Shanba, Yakshanba}
6: int tanlash;
7: cout << “Kun nomerini kiriting (0-6):” ;
8: cin << tanlash;
9: if (tanlash=Yakshanba || tanlash = Shanba)
10:cout <<“\nBugun siz uchun dam olish kuni!”
```

```
11:<<endl;
12:else
13:cout << "\n Bugun siz uchun ish kuni.\n";
14:return 0;
15:};
```

НАТИЖА:

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

САВОЛЛАР

1. Нима учун литералли ўзгармасга нисбатан белгили ўзгармасни ишлатиш яхшироқ?
2. Бутун сонли ва ҳақиқий типларни қандай фарқи бор?
3. unsigned short int ва long int типларининг ўзаро фарқи нимада?
4. const калитли сўзини #define директиваси ўрнига қўллашни афзаллиги нимада?
5. Дастур ишига “яхши” ва “ёмон” номланган ўзгарувчи қандай таъсир қилади?

ТАЯНЧ ИБОРАЛАР

Ўзгарувчи, ўзгарувчининг аниқланиши, хотирани резеврланиши,
Ишорали ва ишорасиз типлар, калитли сўзлар белгилар, махсус
белгилар, ўзгармаслар, литерал ўзгармаслар, белгили
ўзгармаслар, #define директиваси, const калитли сўзи, саноқли
ўзгармаслар.

Ифодалар ва операторлар.

Дастур бирор бир аниқланган кетма - кетликда бажарилувчи командалар тўпламидан иборат. Сиз ушбу мавзудан қуйидагилар билан танишасиз:

- Оператор нима
- Блок нима
- Ифода нима

- Дастурни берилган мантиқий шартларини бажарилиши натижаси асосида тармоқланишини қандай ташкил этиш лозим.
- C++ дастурчиси нуқтаи-назарида “Рост” ва “ Ёлғон” нима?

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.

Блоклар ва комплекс ифодалар.

Баъзан дастур тушунарли бўлиши учун ўзаро мантиқий боғланган ифодаларни блок деб аталувчи комплексларга

бирлаштириш қулайдир. Блок очилувчи фигурали қавс ({) билан бошланади ва ёпилувчи фигурали қавс (}) билан тугайди. Блок очилганда ва ёпилганда нуқтали вергул қўйилмайди.

```
{
  temp= a;
  a = b;
  b = temp;
}
```

Бу блок худди бир ифодадек бажарилади, у a ва в ўзгарувчилар қийматларини алмаштиради.

Амаллар (Операциялар).

Бажарилиши натижасида бирор бир қиймат қайтарадиган барча ифодалар C++ тилида амаллар дейилади. Амаллар албатта бирор бир қиймат қайтаради. Масалан, 3+2 амалси 5 қийматни қайтаради.

Операторлар.

Оператор - бу қандайдир амални бажариш туғрисида компиляторга узатиладиган литералдир. Операторлар операндларга таъсир қилади. C++ да операндлар деб алоҳида литераллар ва бутун ифодалар тушунилади.

C++ тилида икки кўринишдаги операторлар бор:

- ўзлаштириш операторлари
- математик операторлар

Ўзлаштириш оператори.

Ўзлаштириш оператори (=) 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 операторидан чапда жойлашган операнд адресли операнд ёки l-киймат (chap-чап сўзидан олинган) дейилади. Ўзлаштириш операторидан ўнгда жойлашган операнд операцион операнд ёки r-киймат дейилади.

Ўзгармаслар фақатгина r-киймат бўлиши мумкин ва ҳеч қачон адресли операнд бўла олмайди, чунки дастурнинг

бажарилиши жараёнида ўзгармас қийматини ўзгартириб бўлмайди.

```
35 = x // нотугри!
```

l–киймат эса r–киймат бўлиши мумкин.

Математик операторлар.

C++ тилида 5 та асосий математик операторлар кўлланилади: қўшиш (+), айириш (-), кўпайтириш (*), бутун сонга бўлиш (\) ва модул бўйича бўлиш (%) (қолдиқни олиш).

Ишорасиз бутун сонларни айиришда, агарда натижа манфий сон бўлса ғайриоддий натижа беради. Буни 4.2. листингдан кўришимиз мумкин.

4.1. – листинг. Айириш натижасида бутун сонни тўлиб қолишига мисол

```
1: #include < iostream.h >
2: int main()
3: {
4: unsiegned int ayirma
5: unsisgned int kattaSon = 100;
6: unsigned int kichikSon = 50;
7: ayirma = kattaSon – kichikSon;
8: cout << “Ayirma“:<< ayirma<< “ ga teng\n”;
9: ayirma = kichikSon - kattaSon ;
10:cout << “Ayirma“:<< ayirma<< “ ga teng\n”;
11:<<endl;
12:return 0;
13:}
```

НАТИЖА:

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

Бутун сонга бўлиш ва қолдиқни олиш операторлари.

Бутун сонга бўлиш одатдаги бўлишдан фарқ қилади. Бутун сонга бўлишдан ҳосил бўлган бўлинманинг фақатгина бутун қисми олинади. 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 кийматини 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. Перефиксли 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. Перефексли оператор қиймат ўзлаштирилгунча, постфиксли оператор эса қиймат ўзлаштирилгандан кейин бажарилади. Буни қуйидаги листингдан кўришимиз мумкин:

4.2. – листинг. Префиксли ва постфиксли операторларни қўлланиши.

```
1: # include < iostream. h >
2: int main()
3: {
4: int myAge = 39;
5: int yourAge = 39;
6: cout << “Men” << MyAge <<”yoshdaman \n”;
```

```
7: cout << “Siz” << yourAge <<“yoshdasiz \n”;
8: myAge++ ; // постфиксли инкремент
9: ++yourAge; // префексли инкремент
10:cout << “Bir yil o`tdi …\ n”;
11:cout << “Men” << MyAge <<”yoshdaman \n”;
12:cout << “Siz” << yourAge <<“yoshdasiz \n”;
13:cout << “Yana bir yil o`tdi \n”;
14:cout <<” Men”<< myAge++ <<“yoshdaman \n”;
15:cout <<”Siz”<<++yourAge<<“yoshdasiz \n”;
16:cout << “Ma`lumotlarni qaytadan”
17:cout << “chiqaraylik \n”;
18:cout <<” Men”<< myAge<<“yoshdaman \n”;
19:cout <<”Siz”<<yourAge<<“yoshdasiz \n”;
20:return 0;
21:}
```

НАТИЖА:

```
Men 39 yoshdaman
Siz 39 yoshdasiz
Bir yil o`tdi …
Men 40 yoshdaman
Siz 40 yoshdasiz
Yana bir yil o`tdi …
Men 40 yoshdaman
Siz 41 yoshdasiz
Ma`lumotlarni qaytadan chiqaraylik
Men 41 yoshdaman
Siz 41 yoshdasiz
```

Операторлар приоритети.

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 операторлар ҳам бу тартибга амал қилмайди. Масалан, ўзлаштириш оператори ўнгдан чапга қараб бажарилади.

Ички қавслар.

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.

Муносабат операторлари.

Бундай операторлар иккита қийматни тенг ёки тенг эмаслигини аниқлаш учун ишлатилади. Таққослаш ифодаси

доимо `true` (рост) ёки `false` (ёлғон) қиймат қайтаради. Муносабат операторларининг кўлланилишига оид мисол 4.1. жадвалда келтирилган.

4.1.жадвал. Муносабат операторлари.

Номи	Оператор	Мисол	Қайтарадиган қиймат
Тенглик	==	100==50 50==50	false true
Тенг эмас	!=	100!=50 50!=50	true false
Катта	>	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

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)
{
   1 - ифода
   2 - ифода
   3 - 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.

4.3. – листинг. Муносабат операторининг қўлланилиши орқали тармоқланишга мисол

```
1: // 4.3. – листингда муносабат оператори
2: // билан биргаликда if инструкциясининг
3: // қўлланилиши намойиш этилган
4: # include <iosrteam.h>
5: int main( )
6: {
7:  int BuxoroGol, PaxtakorGol;
8:  cout<<"Buxoro komandasi kiritgan to`plar"
9:  << "sonini yozing:";
10: cin >> BuxoroGol;
11:
12: cout<<"Paxtakor komandasi kiritgan"
13: << "to`plar sonini yozing:";
14: cin >> PaxtakorGol;
15:
16:cout << "\n";
17:
18: if ( BuxoroGol>PaxtakorGol)
19:cout << "Yashasin Buxoro!\n"
20:
```

```
21:if (BuxoroGol < PaxtakorGol)
22:{
23: cout << “Yashasin PaxtakorGol \n”
24: cout << “Bugun Toshkentda bayram!\n”;
25:}
26:
27:if (BuxoroGol==PaxtakorGol)
28:{
29:  cout << “Durrangmi? Yo-oq? Bo`lishi”<<
30:  “ mumkin emas \n”;
31:  cout <<“Paxtakorning kiritgan to`plari”
32:  << “haqida ma`lumotni qaytadan yozing\n”
33:  cin >> PaxtakorGol;
34:
35:  if (BuxoroGol>PaxtakorGol)
36:{
37:cout<<“Buxoro yutishini oldindan bilgan”
38:<<“ edim! Shuning uchun qayta so`radim\n”;
39:cout<< “Yashasin Buxoro!”;
40:
41:if (BuxoroGol<PaxtakorGol)
42:{
43:cout<<“Paxtakor yutishini oldindan bilgan”
44:<<“ edim! Shuning uchun qayta so`radim\n”;
45:cout<< “Yashasin Paxtakor!”;
46:cout << “Bugun Toshkentda bayram!\n”;
47:
48:  if (BuxoroGol==PaxtakorGol)
cout<<“Qoyil! Haqiqatan ham during ekan\n”;
49:}
50:
   cout<<“\n Ma`lumotingiz uchun rahmat\n”;
51: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;
```

4.4. – листинг. `else` калитли сўзининг ишлатилиши.

```
1: // 4.4. Листинг.  If ва else калит сўзларини
2: // ишлатилишига оид мисол
3:
4: # include <iostream.h>
5: int main()
6: {
7: int BirinchiSon, IkkinchiSon;
8: cout << "Katta sonni kiriting:
9: cin >> BirinchiSon;
10:cout<<"\n Kichik sonni kiriting:";
11:cin >> IkkinchiSon;
```

```
12:if (BirinchiSon > IkkinchiSon)
13:cout << "\n Rahmat! \n";
14:else
15:cout << "\n Ikkinchisi katta son-ku!";
16:return 0;
17:}
```

НАТИЖА:

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

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 ташкил 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 4.5 – листингда келтирилган.

4.5. – листинг. if операторн ички бўлган мураккаб конструкция

```
1: // 4.5. – Листинг. If операторн ички
2: // бўлган мураккаб конструкцияга мисол
3: # include < iostream.h>
4: {
5: // Иккита сон киритамиз.Уларни BirinchiSon
6: //ва IkkinchiSon ўзгарувчиларига beramiz
7: //Агарда KattaSon киймати KichikSon
8: // кийматидан катта бўлса катта сон
9: //кичигига колдиксиз бўлинишини текшира-
10:// миз. Агарда у колдиксиз бўлинса улар
11://тенг ёки тенг эмаслигини текширамиз.
12:int BirinchiSon, IkkinchiSon;
13: cout<<"Ikkita son kiriting.\n Birinchisi: ";
14: cin >> BirinchiSon;
15: cout << "\n Ikkinchisi:
16: cin >> IkkinchiSon;;
17: cout << "\n\n;
18:
19: if (BirinchiSon>=IkkinchiSon;)
20:   if ( (BirinchiSon%IkkinchiSon)==0)
21:    {
22:      if (BirinchiSon==IkkinchiSon;)
23:       cout<< "Ular bir – biriga teng!\n";
24:else
25:cout<< "Birinchi son ikkinchisiga"<<
26:"karrali!\n";
27:}
28:else
29:cout<<"Ikkinchi son katta!\n"
30:return 0;
31:}
```

НАТИЖА

```
Ikkita son kiriting
Birinchisi:    9
Ikkinchisi:    3
```

Birinchi son ikkinchisiga karrali!

Мантиқий операторлар

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 ташкил 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 4.2 – жадвалда келтирилган

4.2 – жадвал. Мантиқий операторлар

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

Мантиқий кўпайтириш оператори

Мантиқий кўпайтириш оператори 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,

```
if(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 оператори `&&` белги орқали белгиланади.

Мантиқий қўшиш оператори

Мантиқий қўшиш оператори ҳам иккита ифода орқали ҳисобланади. Агарда улардан бирортаси рост бўлса мантиқий қўшиш оператори `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,

```
if(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 оператори

Мантиқий инкор оператори текширилаётган ифода ёлғон бўлса `true` қиймат қайтаради. Агарда текширилаётган ифода рост бўлса инкор оператори `false` қиймат қайтаради. Масалан,

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

ифодасининг қиймати, агарда `x` ўзгарувчиси `5` га тенг бўлмаса `true` қиймат қайтаради. Бу ифодани бошқача ҳам ёзиш мумкин:

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

САВОЛЛАР

1. Ифода нима?
2. x=5+7 ёзуви ифода бўла оладими? Унинг қиймати нечага тенг?
3. 201/4 ифоданинг қиймати нечага тенг?
4. 201%4 ифода қиймати нечага тенг?

5. Агарда MyAge, a ва в ўзгарувчиларининг типлари int бўлса уларнинг қийматлари қуйидаги ифодалар бажарилгандан сўнг нечага тенг бўлади.
```
MyAge = 39;
a=MyAge++
b=++MyAge
```
6. If(x = 3) ва if(x = =3) ифодаларнинг фарқи нимадан иборат?

ТАЯНЧ ИБОРАЛАР

Ифода, операция, операнд, оператор, инкремент, декремент, префикс, постфикс, операторлар приоритети, if оператори, мантиқий операторлар

Функциялар.

Объектларга мўлжалланган дастурлашда асосий эътибор функцияга эмас, балки объектга қаратилган бўлсада дастурларда функция марказий компонентлигича қолди. Биз ушбу мавзуда қуйидагилар билан танишамиз:

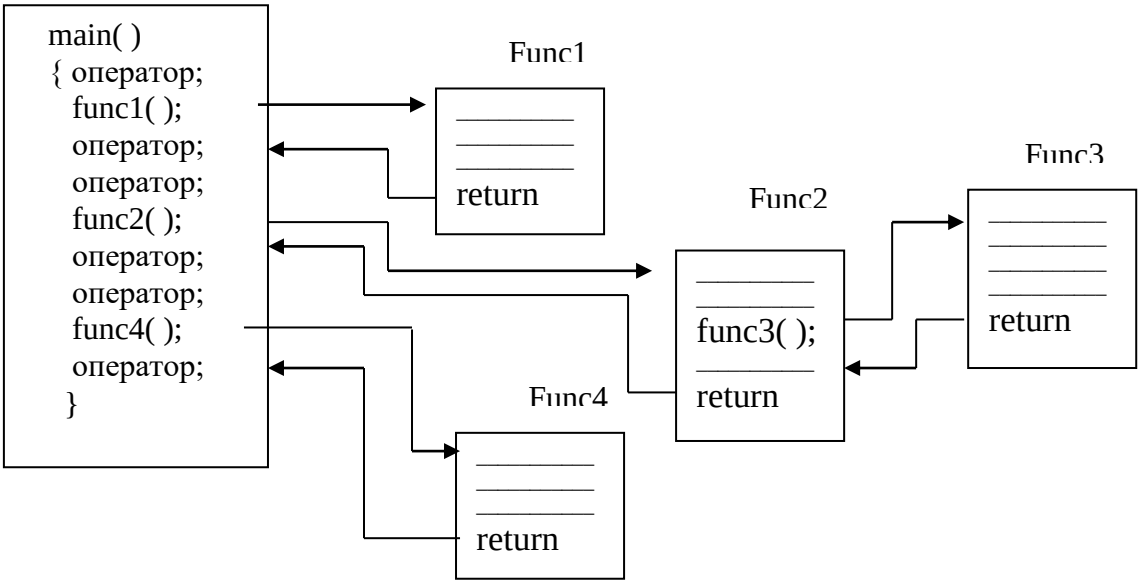
- Функция нима ва у қандай қисмлардан иборат?

- Функция қандай эълон қилинади ва аниқланади?
- Функцияга қандай қилиб параметрлар узатилади?
- Функция қандай қиймат қайтаради?

Функция нима?

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. Дастур бажарилишининг бундай схемаси 5.1.- расмда кўрсатилган.



5.1.-расм. Функциянинг чақирилиши тартиби

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 киймат 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.

Функцияни эълон қилиш ва аниқлаш.

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.

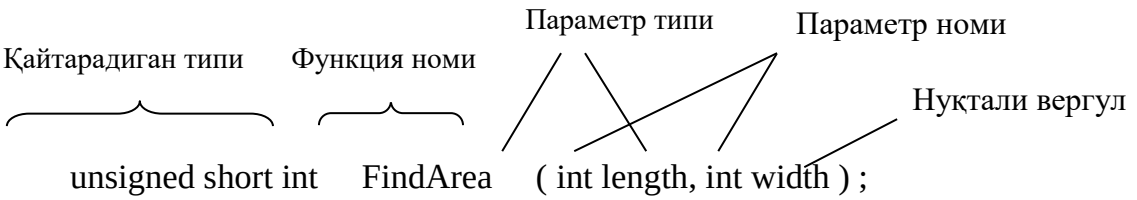
Учинчидан, функциянинг прототиплари дастурни текшириш жараёнида жуда яхши ишлатилади. Агарда функция прототипи аниқланган бўлса унга мувофиқ функция аниқланган параметрини қабул қилади ёки аниқланган бирор бир қиймат қайтаради. Дастурда эълон қилинган прототипга мувофиқ бўлмаган функцияни ишлатишга уринсак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

Функция прототиплари.

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. Функция прототипининг таркибий қисмлари 5.2. - расмда кўрсатилган.



5.2.–расм. Функция прототипининг таркибий кисмлари

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 туғридир:

```
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.

5.1.–листинг. Функцияни эълон қилиниши, аниқланиши ва ишлатилиши.

```
1: // 5.1. – Листинг. Функция прототипини
2:   // кўлланлиши
3: # include <iostream.h >
4: //функция прототипи
5: int Yuza(int uzunlik, int kenglik);
6: int main()
```

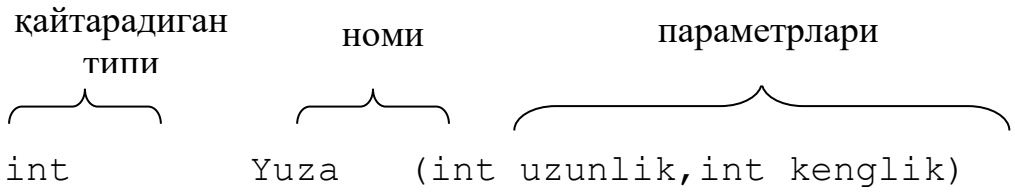
```
7: {
8:  int YerUzunligi,
9: int YerKengligi,
10:int YerMaydoni;
11:cout<< "\n Yerning uzunligi necha metr?\n";
12:cin >> YerUzunligi;
13:cout<< "\n Yerning kengligi necha metr?";
14:cin >> YerKengligi;
15:YerMaydoni=Yuza(YerUzunligi,YerKengligi);
16:cout << "\n Yer maydoni yuzasi ";
17:cout >>YerMaydoni;
18:cout << "kvadrat metr\ n \ n";
19:return 0;
20:}
21:int Yuza(int YerUzunligi,int YerKengligi)
22:{
23:return YerUzunligi* YerKengligi
24:}
```

НАТИЖА

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

Функциянинг аниқланиши.

Функциянинг аниқланиши икки қисмдан – унинг сарлавҳаси ва танасидан иборатдир. Функциянинг сарлавҳаси унинг прототипига ўхшаш аниқланади, фақатгина бу ҳолда параметрлар номланган бўлиши шарт ва сарлавҳа охирида нуқтали вергул қўйилмайди. Функция танаси фигурали қавсга олинган ифодалар тўпламидан иборат. Функциянинг сарлавҳаси ва танаси 5.3 - расмда кўрсатилган.




```
{ - очилувчи фигурали кавс.
```

```
//  funktsiya tanasi

  return (uzunlik*kenglik);

} - ёпилувчи фигурали кавс.
```

5.3 – расм. Функциянинг сарлавхаси ва танаси

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.

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. 5.2. – листингда функция параметрлари ва функция ичида аниқланган локал ўзгарувчиларни қўллашга оид мисол келтирилган.

5.2. – листинг. Функция локал ўзгарувчилари ва параметрларининг қўлланилиши.

```
1: # include <iostream.h>
2: float Almashtirish(float);
3: int main()
4: {
5:   float TempFer;
6:   float TempCel;
7:    cout << "Feringait bo`yicha temperaturani
8:    << "kiriting:";
9:    cin >> TempFer;
```

```
10:TempCel = Almashtirish(TempFer);
11:cout << "\n Bu temperatura selziy shkalasi
12:<< "bo`yicha: ";
13:cout << TempCel << endl;
14:return 0 ;
15:}
16:float Almashtirish(float TempFer)
17:{
18:float TempCel;
19:tempCel=((TempFer-32)*5)/9;
20:return TempCel;
21:};
```

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. Бу ғоя 5.3.- листингда намойиш қилинган.

5.3.–листинг. Глобал ва локал ўзгарувчиларнинг қўлланиши.

```
1: # include <iostream.h>
```

```
2: void MeningFunktsiyam() ; // прототип
3: int x = 5, y = 7; // глобал ўзгарувчилар
4: int main()
5: {
6:  cout << "main()dagi x ning qiymati:"
7:  <<x<<"\n";
8:  cout<<"main()dagi y ning qiymati y:"
9:  <<y<<"\n";
10:
11:MeningFunktsiyam();
12: cout << "MeningFunktsiyam()funktsiyasi"<<
13: "ishini tugatdi!\ n \ n";
14: cout<<"main()dagi x ning qiymati:"
15: <<x<<"\n";
16: cout<<"main()dagi y ning qiymati:"
17: <<y<<"\n";
18: return 0;
19: }
20:  void MeningFunktsiyam();
21:   { int y = 10;
22:     cout<<"MeningFunktsiyam()dagi"<<
23:     << "x:" << x <<"\ n";
24:     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.

5.4. – листинг. Локал ўзгарувчини кўриниш соҳаси.

```
1: # include <iostream.h>
2: void MeningFunktsiyam();
3: int main()
4: {
5:  int  x=5;
     6: cout<<"\n\n main()dagi x ning qiymati:"
7:  <<x;
8:
9:  MeningFunktsiyam();
10:
11:cout<<"\n main()dagi x ning qiymati:"<< x;
12: return 0;
13:}
14:void MeningFunktsiyam();
15:{
16:int x = 8;
17:cout <<"\n\n MeningFunktsiyam()dagi"
18:<<"local x ning qiymati: "<< x << endl;
19:{
20:cout << "\ n\ MeningFunktsiyam() "
21:<<"funktsiyasi blokidagi x ning qiymati"
22:<< " x:"<<x;
23:int x = 9;
24:cout<<"\n Blok ichida aniqlangan"
25:<< x ning qiymati:"<<x;
```

```
26:}
27:cout<<"\n MeningFunktsiyam()dagi"
28:<< "blockdan tashqarisida x ning qiymati:"
29:<<x<<endl;
30:}
```

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
```

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. Бу фикр 5.5 - листингда намойиш қилинган.

5.5.–листинг. Функцияга параметрларни қиймат сифатида узатиш.

```
1: // Функцияга параметрларни киймат сифатида
2: // узатиш
3: #include <iostream.h>
4: void Almashtirish(int x, int y);
5: int main()
6: { int x = 5,  y =10;
7: cout << "Main(). Almashtirishdan oldin,x:"
8: <<x <<"  y:"  << y <<"\n";
9: Almashtirish(x,y);
10:cout<<"Main(). Almashtirishdan keyin, x:"
11:<<x<< "y:" <<y<< "\n";
12:return 0;
13:}
14:void Almashtirish(int x, int y)
15:{
16:int temp;
17:cout<<"Almashtirish().Almashtirishdan "<<
18:"oldin, x: "<<x<<" y: "<<y<< "\n";
19:temp = x ;
```

```
20:x = y;
21:y = temp;
22:cout<<"Almashtirish().Almashtirishdan "<<
23:"keyin, x: "<<x<<" y: "<<y<< "\n";
24:}
```

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. Бу ғоя 5.6 – листингда намойиш қилинган.

5.6. – листинг. Бир нечта `return` операторини қўлланилиши

```
1:  // 5.6. – листинг.
2: # include < iostream.h>
3: int IkkigaKupaytirish(int KupaytSon);
4:
5: int main()
6: {
7: int natija=0;
8: int input;
9: cout << “Ikkiga ko`paytiriladigan sonni”
10:<< “kiriting(0 dan 10000 gacha):”;
11:cin >> input;
12:
13:cout << “\n IkkigaKupaytirish() funktsiyasi”
14:<< “chaqirilishidan oldin\n”;
cout<<“Kiritilgan qiymat:” <<input
<<“Ikkilangani:”<<natija<< “\n”;
15:
16:result = IkkigaKupaytirish(input);
17:cout<<“\nIkkigaKupaytirish() funktsiyasidan”
18:<<“qaytgandan so`ng...\n”;

19:cout<<“Kiritilgan qiymat:” <<input
20:<<“Ikkilangani:”<<natija<< “\n”;

21:return 0;
22:}
23:int IkkigaKupaytirish(int original)
24:{
25:if (original <= 10000)
26:return original*2;
27:else
28:return –1;
29:cout<< “ Siz bu satrga o`ta olmaysiz!\n”;
30:}
```

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
```

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.

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. Бир хил номли функцияларнинг аниқланиши 5.8.–листингда кўрсатилган.

5.8. – листинг. Функцияларнинг полиморфизми

```
// 5.8. – Листинг. Функция полиморфизмига
// мисол.
1: #include <iostream.h>
```

```
2: int Double(int);
3: long Double(long);
4: float Double(float);
5: double Double(double);
6: int main()
7: {
8: int myInt = 6500;
9: long myLong = 65000;
10:float myFloat=6.5;
11:double myDouble = 6.5 e20;
12:int doubledInt;
13:long doubledLong;
14:float doubledFloat;
15:double doubledDouble;
16:cout<<”myInt: ”<< myInt<< “\n”;
17:cout<<”myLong: ”<< myLong<< “\n”;
18:cout<<”myFloat: ”<< myFloat<< “\n”;
19:cout<<”myDouble: ”<< myDouble<< “\n”;
20:doubledInt=Doubled(myInt);
21:doubledLong=Doubled(myLong);
22:doubledFloat=Doubled(myFloat);
23:doubledDouble=Doubled(myDouble);
24:cout<<”doubledInt:”<<doubledInt<<“\n”;
25:cout<<”doubledLong:”<<doubledLong<<“\n”;
26:cout<<”doubledFloat:”<<doubledFloat<<“\n”;
27:cout<<”doubledDouble:”<<doubledDouble<<
28:“\n”;
29:return 0;
30:}
31:int Double(int original)
32:{
33:cout<< “Double(int) funktsiyasi\n”;
34:return 2*original;
35:}
36:long Double(long  original)
37:{
38:cout<< “ Double(long) funktsiyasi \n”;
39:return 2*original;
```

```
40:}
41:float Double(float original)
42:{
43:cout<< “Double(float) funktsiyasi \n”;
44:return 2*original;
45:}
46:double Double(double  original)
47:{
48:cout<< “Double(double) funktsiyasi\n”;
49:return 2*original;
50:}
```

НАТИЖА

```
MyInt:      6500
MyLong:     65000
MyFloat:        6.5
MyDouble:   6.5 e+20
Double(int)    funktsiyasi
Double(long)   funktsiyasi
Double(float)  funktsiyasi
Double(double) funktsiyasi
DoubledInt: 13000
DoubledLong: 130000
DoubledFloat: 13
DoubledDouble: 13e+21
```

Функциялар ҳақида қўшимча маълумот. 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.

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. Fib() функциясининг ишлатилиши 5.10. – листингда кўрсатилган.

5.10. – листинг. Фибоначчи қатори аъзосининг қийматини топиш учун рекурсияни қўлланилишига мисол.

```
1:  #include <iostream.h>
2:
3: int fib(int n);
4:
5: int main( )
6: {
7: int n, javob;
8: cout << “Izlanayotgan nomerni kiriting:”;
9: cin >> n;
10:
11:cout << “\n\n”;
12:
13:javob=fib(n);
14:
```

```
15:cout<< “Fibonachchi qatorining”<< n
16: <<“nomeri qiymati ” <<javob<<“ ga teng \n”;
17:return 0;
18:}
19:int fib(int n)
20:{
21:cout << “fib(“<< n << “) jarayoni…”;
22:  if (n <3)
23:  {
24:   cout<< “1 qiymatni qaytarayapti!”\n;
25:   return (1);
26:}
27:else
28:{
29: cout<< “fib(” << n-2 << “) va fib(” <<n-1
30: cout<< “) funktsiyalari chaqirayapti. \n”;
31:return(fib(n-2)+fib(n-1));
32: }
33:}
```

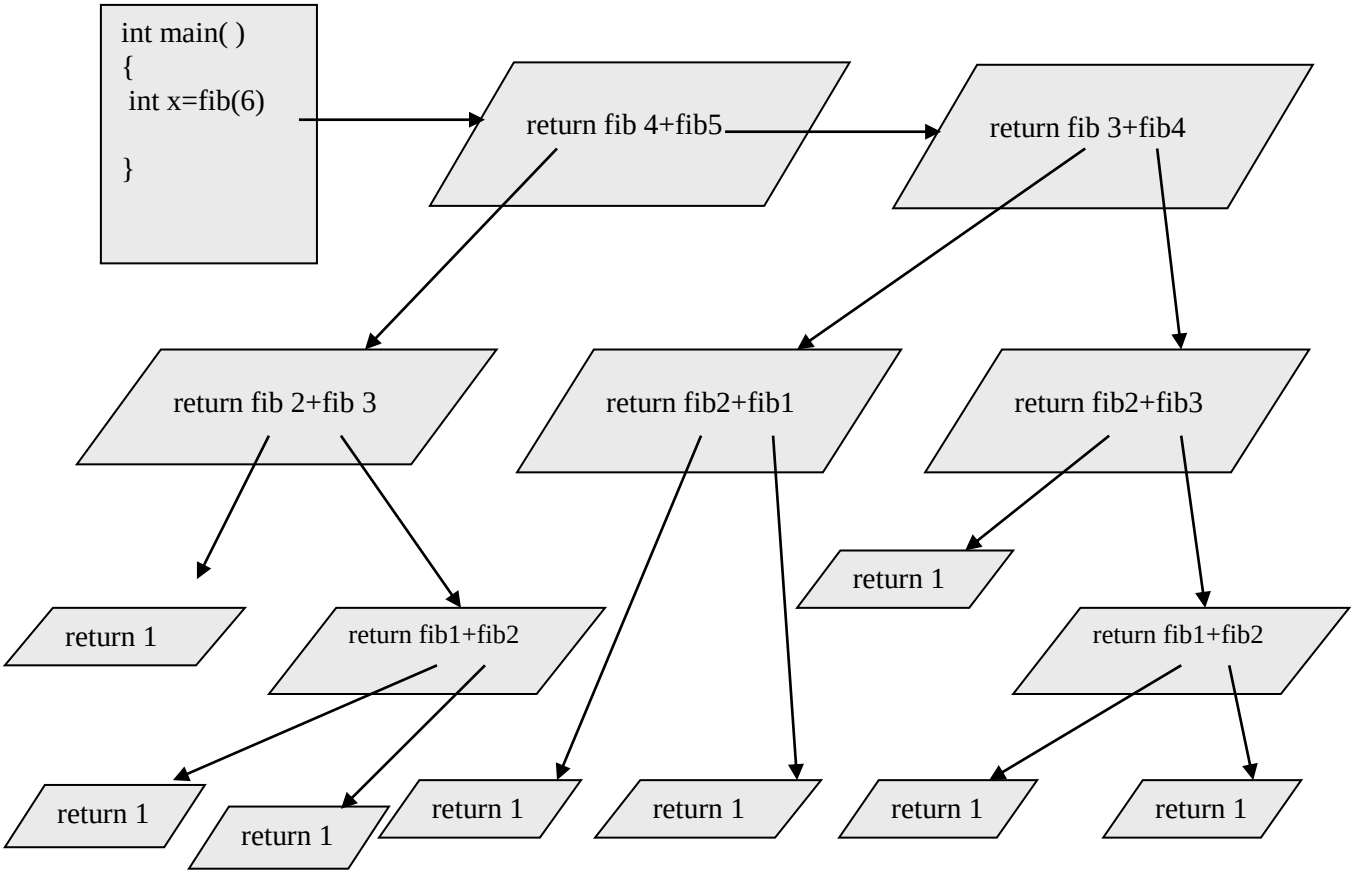
НАТИЖА:

```
Izlanayotgan nomerni kiriting:: 6
fib(6) … fib(4) va fib(5) funktsiyalarini chaqirayapti.
fib(4) …  fib(2) va  fib(3) funktsiyalarini chaqirayapti.
fib(2) … 1 qiymatni qaytarayapti!
fib(3) … fib(2) va fib(1) funktsiyalarini chaqirayapti.
fib(1) … 1 qiymatni qaytarayapti!
fib(2) … 1 qiymatni qaytarayapti
fib(5) … fib(3) va fib(4) funktsiyalarini chaqirayapti.
fib(3) … fib(2) va fib(1) funktsiyalarini chaqirayapti.
fib(1) … 1 qiymatni qaytarayapti!
fib(2) … 1 qiymatni qaytarayapti
fib(4) … fib(2) va fib(3) funktsiyalarini chaqirayapti.
fib(2) … 1 qiymatni qaytarayapti
fib(3) … fib(2) va fib(1) funktsiyalarini chaqirayapti.
fib(1) … 1 qiymatni qaytarayapti!
fib(2) … 1 qiymatni qaytarayapti
Fibonachchi qatorining 6 nomeri qiymati 8 ga teng
```

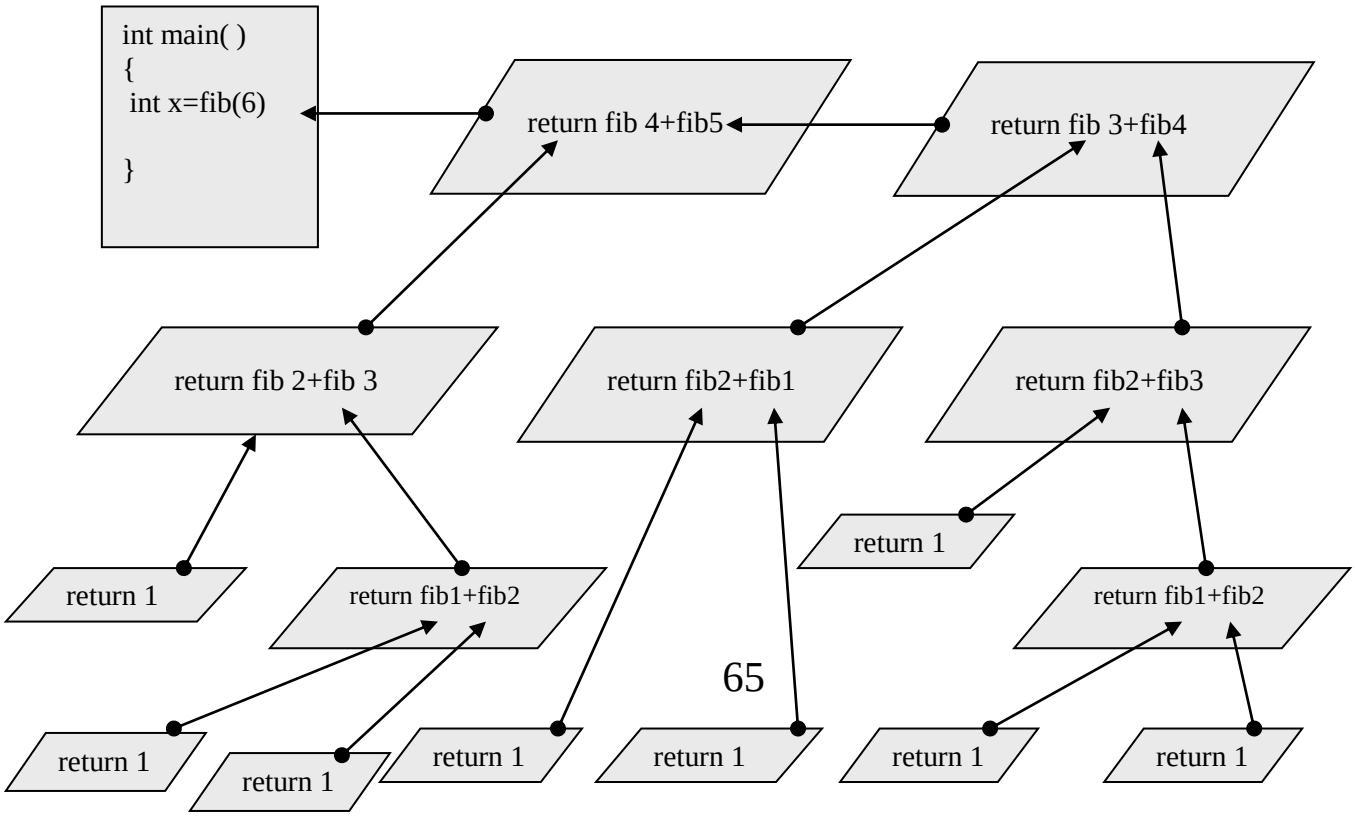
Изох

Айрим компиляторлар cout объекти қатнашган ифодаларда операторларни қуллашда баъзи қийинчиликлар пайдо бўлади. Агарда компилятор 28 сатрдаги ифода ҳақида огоҳлантириш берса айириш операциясини қавс ичига олинг, бу қатор қуйидаги кўринишга эга бўлсин.

```
28: cout << “Call fib(“ <<(n-2)<<”) and fib(“ <<(n-2)<<
“).\n”;
```



5.4. – расм. Рекурсив функцияниng ишлаши.



5.5. – расм. Рекурсив функцияларнинг қиймат қайтариши.

Функциянинг ишлаш тамойили ҳақида

Функция чақирилганда дастур бошқаруви чақирилган функцияга узатилади, яъни функция танасидаги операторларнинг бажарилиши бошланади. Функция операторлари бажарилгач, у бирор қийматни натижа сифатида қайтаради (агарда у аниқланмаган бўлса функция `void` типидаги қийматни қайтаради) ва бошқарув функция чақирилган жойдан давом эттирилади.

Бу масала қандай амалга оширилади? Дастурга унинг бошқаруви қайси блокка ўтиши лозимлиги қандай маълум бўлади? Аргумент сифатида аниқланган ўзгарувчи қаерда сақланади? Функция танасида аниқланган ўзгарувчилар қаерда сақланади? Қайтариладиган қиймат қандай орқага узатилади? Дастурга у қайси жойдан иш бошлаши кераклилиги қаердан маълум?

Кўпгина адабиётларда бу саволларга жавоб берилмайди. Лекин, функцияни ишлаш принципини билмасак бизга компиляторнинг ишлаш принципи жуда мавҳум бўлиб кўринади. Шунинг учун ушбу мавзуда компьютернинг хотираси билан боғлиқ саволлар кўриб чиқилади.

Абстракция даражаси

Компьютерлар, албатта, оддий электрон машиналардир. Улар ойналар ва менюлар, программалар ва командаларни билишмайди. Улар ҳатто 0 ва 1 ларни ҳам тушунишмайди. Ҳақиқатда эса бўладиган барча жараёнлар интеграл микросхеманинг турли жойларида электр токига боғлиқдир. Ва ҳаттоки бу ҳам абстракциядир. Электрнинг ўзи ҳам элементар частоталар ҳаракатини умумлаштирувчи концепциядир.

Айрим дастурчилар оператив хотира (ОЗУ) да сақланадиган қийматлар даражасига тушишгача детализация қилишади. Лекин машинани бошқариш ёки тўпни тепиш учун физика фанидаги элементар частоталарни ўрганиш зарур бўлмаганидек, дастурлашда ҳам бу даражадаги детализация талаб этилмайди.

Демак, компьютер электроникасини тушунмасдан туриб ҳам дастурлаштириш мумкин экан.

Лекин сиз компьютер хотираси қандай ташкил этилганлигини тушунишингиз керак. Сизнинг ўзгарувчиларингиз тузилгандан кейин қаерда жойлашиши ва уларга функциялар орқали қиймат қандай берилишини ҳақида аниқ тасаввурга эга бўлмасангиз, дастурлаш сиз учун сирли бўлиб қолаверади.

Хотиранинг тақсимланиши

Сиз ўзингизни дастурингиз билан ишлай бошлашингиз билан операцион система (Dos ёки Microsoft Windows) компиляторнинг талабига мувофиқ хотира соҳасидан жой ажратади. C++ дастурчиси сифатида сиз глобал номлар фазоси, эркин тақсимланувчи хотира, регистр, сегментли хотира ва стек тушунчаларини билишингиз лозим.

Глобал номлар фазосида глобал ўзгарувчилар сақланади. Глобал номлар фазоси ва эркин тақсимланувчи хотира ҳақида кейинги мавзуларда батафсилроқ танишиб чиқамиз. Ҳозир эса регистр, дастур сегменти ва стек ҳақида тўхталиб ўтамиз.

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

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

Стек – бу сизнинг дастурингизда 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.

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 параметерлари 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. Рекурсия нима?
10. Глобал ўзгарувчиларни қачон ишлатиш лозим?
11. Бир хил номли хар хил функциялар қандай аниқланади?
12. Полиморфизм нима?

ТАЯНЧ ИБОРАЛАР

Глобал ўзгарувчи, локал ўзгарувчи, функция аргументи, функция қайтарадиган қиймат, функцияни эълон қилиш, функцияни аниқлаш, ўзгарувчини кўриниш соҳаси, рекурсия, функция перегрузкаси, полиморфизм

Таянч синфлар.

Таянч синфлар C++ тили ички имкониятларини кенгайтириб дастурчи олдида амалиётнинг мураккаб масалаларини ечишда анча қулайлик яратади. Ушбу мавзуда сиз қуйидагиларни билиб оласиз.

- Синфлар ва объектлар ўзида нимани ифодалайди ?
- Янги синфни ва бу синф объектини қандай ҳосил қилиш керак?
- Функция аъзолар ва ўзгарувчи аъзолар нима ?
- Конструктор нима ва уни қандай ишлатиш керак ?

Янги тип тузиш.

Олдинги дарсларда бутун, ҳақиқий ва белгили типлар билан танишган эдик. Улардан маълумки, тип орқали ўзгарувчи хусусиятлари характерланади. Масалан, агар `Height` ва `Width`

ўзгарувчиларни ишорасиз қисқа бутун (unsigned short int) типида эълон қилсак, уларнинг ҳар бири 0 - 65535 диапазондаги сонларни қабул қилиши мумкин ва бунда улар 2 байтдан жой эгаллайди. Агарда сиз бу ўзгарувчиларга шу ораликдан ташқари бирор сон бермоқчи бўлсангиз хатолик ҳақида ахборот оласиз.

Демак, Height ва Width ўзгарувчиларини ишорасиз бутун сон деб эълон қилишдан сиз бу ўзгарувчиларни қўшиш ёки уларнинг бирининг қийматини иккинчисига ўзлаштириш имконига эга бўласиз.

Демак ўзгарувчи типи:

- унинг хотирадаги ўлчовини
- у сақлаши мумкин бўлган маълумот типини
- унинг ёрдамида бажариш мумкин бўлган операцияларни аниқлайди.

Берилганларни типи категориялари сифатида автомобил, уй, одам, геометрик фигураларни мисол қилиб келтириш мумкин. C++ тилида дастурчи ўзига керакли ихтиёрий типни ҳосил қилиши мумкин. Бу тип эса ички таянч типларни хоссалари ва функционал имкониятларини ўзида ифодалайди.

Нима учун янги тип тузиш керак.

Одатда дастурлар ходимлар хақидаги маълумотларни қайта ишлаш ёки иситиш системаси ишини имитация қилиш каби амалдаги масалаларни ечиш учун ёзилади. Бу масалани фақатгина бутун ёки белгили қийматлар ёрдами билан ҳам ечиш мумкин. Агарда сиз турли объектлар учун улкан типлар ҳосил қилсангиз бу масалаларни ечиш етарлича содда кўринади. Бошқа сўз билан айтганда иситиш системаси ишини имитация қилишда, агарда иссиқлик ўлчагичлар, термостатлар ва бойлерларни ифодаловчи ўзгарувчилар тузилса уни жорий қилиш осонроқ бўлади. Бу ўзгарувчилар реалликка қанчалик яқин бўлса, унинг дастурини тузиш шунчалик осон бўлади.

Синфлар ва синф аъзолари.

Янги тип синфни эълон қилиш билан тузилади. Синф - бу бир – бири билан функционал орқали боғланган ўзгарувчилар ва методлар тўпламидир. Синфларга амалиётдан кўпгина мисоллар келтириш мумкин. Масалан, автомабилни ғилдирак, эшик, ўриндиқ, ойна ва бошқа қисмлардан ташкил топган коллекция

ёки ҳайдаш тезлигини ошириш, тўхтатиш, буриш имкониятларига эга бўлган объект деб тасаввур қилиш мумкин. Автомобил ўзида турли эҳтиёт қисмларни ва уларни функцияларини инкапсуляция қилади. Автомобил каби синфда ҳам инкапсуляция қатор имкониятларни беради. Барча маълумотлар битта объектда йиғилган ва уларга осонгина мурожаат қилиш, уларни ўзгартириш ва кўчириш мумкин. Сизнинг синфингиз билан ишловчи дастурий қисмлар, яъни мижозлар сизнинг объектингиздан, унинг қандай ишлашидан ташвишланмасдан, бемалол фойдаланишлари мумкин.

Синф ўзгарувчиларнинг ихтиёрий комбинациясидан, шунингдек бошқа синфлар типларидан иборат бўлиши мумкин. Синфдаги ўзгарувчилар ўзгарувчи – аъзолар ёки хоссалар дейилади. `Car` синфи ўриндиқ, радиоприёмник, шина ва бошқа ўзгарувчи - аъзолардан иборат. Ўзгарувчи – аъзолар фақатгина ўзларининг синфларида ётадилар. Ғилдирак ва мотор автомобилнинг қандай таркибий қисми бўлса, ўзгарувчи – аъзолар ҳам синфнинг шундай таркибий қисмидир.

Синфдаги функциялар одатда ўзгарувчи аъзолар устида бирор бир амал бажарадилар. Улар функция – аъзолар ёки синф методлари деб айтилади. `Mashina` синфи методлари қаторига `Haydash()` ва `Tuxtatish()` методлари киради. `Mushuk` синфи ҳайвонни ёши ва оғирлигини ифодаловчи ўзгарувчи – аъзоларга эга бўлиши мумкин. Шунингдек, бу синфнинг функционал қисми `Uxlash()`, `Miyovlash()`, `SichqonTutish()` методларидан иборат бўлади.

Функция – аъзолар ҳам ўзгарувчи аъзолар сингари синфда ётади. Улар ўзгарувчи аъзолар устида амаллар бажаради ва синфни функционал имкониятларини аниқлайди.

Синфни эълон қилиш.

Синфни эълон қилиш учун `class` калитли сўзи, ундан сўнг очилувчи фигурали қавс, сўнг хоссалар ва методлари рўйхати ишлатилади. Синфни эълон қилиш ёпилувчи фигурали қавс ва нуқтали вергул орқали якунланади. Масалан, `Mushuk` синфини қуйидагича эълон қилиш мумкин.

```
Class Mushuk
  {
   unsigned int itsYosh ;
```

```
    unsigned int itsOgirlik ;
    void Miyovlash()
  }
```

Mushuk синфини эълон қилишда хотира заҳираланмайди. Эълон қилиш, компиляторга Mushuk синфини мавжудлигини, ҳамда унда қандай қийматлар сақлаши мумкинлиги (itsYosh, itsOgirlik*)* ва у қандай амалларни бажариши мумкинлиги (Miyovlash() методи) ҳақида хабар беради. Бундан ташқари, бу эълон қилиш орқали компиляторга Mushuk синфининг ўлчами, яъни ҳар бир Mushuk синфи объекти учун компилятор қанча жой ажратиши лозимлиги ҳақида ҳам маълумот беради. Масалан, жорий мисолда бутун қиймат учун тўрт байт талаб қилинса, Mushuk синфи объекти ўлчови саккиз байт бўлади. (itsYosh ўзгарувчиси учун тўрт байт, itsOgirlik ўзгарувчиси учун тўрт байт). Miyovlash() методи хотирадан жой ажратишни талаб қилмайди.

Объектни эълон қилиш

Янги турдаги объект худди оддий бутун сонли ўзгарувчидек аниқланади. Ҳақиқатан ҳам ихтиёрий бутун сонли ўзгарувчи қуйидагича аниқланади:

```
unsigned int MyVariable
// ишорасиз бутун сонни аниклаймиз
```

Cat синфидаги объект эса қуйидагича аниқланади:

```
Mushuk Frisky  //  Mushuk объектини аниклаймиз.
```

Бу дастурий кодларда unsigned int типидаги MyVariable номли ўзгарувчи ва Mushuk синфининг Frisky номли объекти аниқланди.

Кўпгина ҳолларда синф ва объект тушунчаларини ишлатишда чалкашликка йўл қўйилади. Шунинг учун, объект синфнинг бирор бир экземпляри (нусхаси) эканлигини яна бир бор таъкидлаш жоиз.

Синф аъзоларига мурожаат қилиш имкони.

Mushuk синфининг реал объектини аниқлаганимиздан сўнг бу объектнинг аъзоларига мурожаат қилиш зарурияти туғилиши

мумкин. Бунинг учун бевосита мурожаат (.) оператори қўлланилади. Масалан, Frisky объектининг Weight ўзгарувчи - аъзосига 50 сонини ўзлаштирмоқчи бўлсак қуйидаги жумлани ёзишимиз лозим.

```
Fresky.Weight=50;
```

Meow() методини чақириш учун эса

```
Frisky.Meow();
```

жумласини ёзиш лозим.

Қиймат синфга эмас объектга ўзлаштирилади

C++ тилида берилганлар типига қиймат ўзлаштирилмайди. Қиймат фақатгина ўзгарувчиларга берилади. Масалан, қуйидаги ёзув нотўғридир:

```
Int=s  // нотугри
```

Компилятор int типига қиймат ўзлаштирилиши хатолик эканлиги ҳақида хабар беради. Худди шу нуқтаи – назардан қуйидаги ёзув ҳам ноўриндир:

```
Cat.itsYosh= 5 // нотўгри
```

Агарда Mushuk объект бўлмасдан синф бўлса, юқоридаги ифодани ҳам компилятор хато деб ҳисоблайди. Ўзлаштириш амалини бажаришда хатоликка йўл қўймаслик учун олдин Mushuk синфига тегишли Frisky объектини ҳосил қилиш ва унинг ItsYosh майдонига 5 қийматини бериш лозим.

```
Mushuk Frisky;
Frisky.itsYosh=5;
```

Синф аъзоларига мурожаат қилиш имконини чегаралаш.

Синфни эълон қилишда бир нечта калит сўзлардан фойдаланилади. Улардан энг муҳимлари public (очиқ) ва private (ёпиқ) калит сўзлари бўлиб, улар орқали объектнинг аъзоларига мурожаат қилиш имконияти чегараланади.

Синфнинг барча методлари ва хоссалари бошланғич ҳолда ёпиқ деб эълон қилинади. Ёпиқ аъзоларга фақатгина шу синфнинг методлари орқалигина мурожаат қилиш мумкин. Объектнинг очиқ аъзоларига эса дастурдаги барча функциялар мурожаат қилишлари мумкин. Синф аъзоларига мурожаат қилиш имконини белгилаш жуда муҳим хусусият бўлиб, бу

масалани ечишда унча катта тажрибага эга бўлмаган дастурларчилар кўпинча қийинчиликларга дуч келадилар. Бу ҳолатни батафсилроқ тушунтириш учун мавзуни бошида келтирилган масаламизга қайтамиз.

```
Class Mushuk
 {
    unsigned int itsYosh;
    unsigned int itsOgirlik;
    void Miyovlash();
 }
```

Бу тарзда синфни эълон қилишда `itsYosh` ва `itsOgirlik` майдонлари ҳам, `Miyovlash()` методи ҳам ёпиқ аъзо сифатида аниқланади. Дастурда юқоридаги тартибда `Mushuk` синфи эълон қилинган бўлса ва бу синф экземпляри бўлган объектнинг `itsYosh` аъзосига `main()` функцияси танасидан туриб мурожаат қилсак компилятор хатолик рўй берганлиги ҳақида хабар беради.

```
Mushuk Baroq;
Baroq.itsYosh = 5 // Хатолик!
// Ёпик аъзога мурожаат килиш мумкин эмас.
```

`Mushuk` синфи аъзоларига дастурнинг бошқа объектлари томонидан мурожаат қилиш имконини ҳосил қилмоқчи бўлсак, уни `public` калитли сўзи орқали амалга оширамиз.

```
Class Mushuk
   {
     public:
      unsigned int itsYosh;
      unsigned int itsOgirlik;
      void Meow( );
   }
```

Энди `public` калитли сўзи орқали синфнинг барча аъзолари (`itsYosh, itsOgirlik, Miyovlash()`) очиқ аъзо бўлди.

6.1 – листингда `Mushuk` синфи очиқ ўзгарувчи аъзолари билан эълон қилинган.

6.1. – листинг. Оддий синфнинг очиқ аъзосига мурожаат.

```
# include < iostream.h >;
class Mushuk;
{
  public:
   int itsYosh;
```

```
    int itsOgirlik;
  }
  int GetYosh(); // Inlizchada  Get- olmoq
 void SetYosh (int Age); //Set – o`zgartirmoq
 void Miyovlash();
   private:
 int itsYosh;

 int main()
 {
 Mushuk Frisky;
 Frisky.itsYosh= 5;
 // ўзгарувчи – аъзога киймат ўзлаштирилди.
 cout << "Frisky " <<Frisky.itsAge;
 cout <<" yoshdagi mushuk.\n";
 return 0;
}
```

НАТИЖА:

```
Frisky 5 yoshdagi mushuk
```

Синф методларини аниқланиши

Синф методини аниқлаш учун биринчи синф номи, кейин иккита икки нуқта, функция номи ва параметрлари ёзилади. 6.3. – листингда `Mushuk` синфининг функция аъзосини аниқланиши кўрсатилган.

6.2. – листинг. Оддий синфнинг методини аниқланиши.

```
 #include <iostream.h>

class Mushuk
{
public:
 int GetYosh();
 void SetYosh(int yosh);
 void Miyovlash();
private:
 int itsYosh;
}

// GetYosh функцияси itsYosh ёпик ўзгарувчи
// аъзонинг кийматини кайтаради
```

```
int Mushuk::GetYosh();
{
return itsYosh;
}

// SetYosh функцияси itsYosh ўзгарувчи –
// аъзога киймат ўзлаштиради.
void Mushuk:: SetYosh(int yosh)
{
// itsYosh ўзгарувчи – аъзосига yosh
// киймати параметри ёрдамида берилади.
itsYosh=yosh;
}

void Mushuk::Miyovlash()
{
 cout << "Miyov,miyov,miyov.\n";
}

int main( )
{
 Mushuk Frisky;
 Frisky.SetYosh(5);
 Frisky.Miyovlash();
cout << "Frisky" <<Frisky.GetYosh();
cout << " yoshdagi mushuk.\n";
Frisky.Miyovlash();
return 0;
}
```

НАТИЖА:

```
Miyov, miyov, miyov
Frisky 5 yoshdagi mushuk
Miyov, miyov, miyov
```

Конструкторлар ва деструкторлар

Бутун сонли ўзгарувчини аниқлашнинг икки хил йўли бор. Биринчиси, олдин ўзгарувчини аниқлаш, кейин эса унга бирор бир қиймат ўзлаштиришдир. Масалан,

```
int Ogirlik;  // ўзгарувчини аниклаш
........    // бу ерда бошка ифодалар бор
Ogirlik=7; // ўзгарувчига киймат ўзлаштирамиз.
```

Иккинчиси, ўзгарувчи аниқланиши билан бирга унга дарҳол қиймат ўзлаштирилади, масалан:

```
int Ogirlik=7;  //ўзгарувчини эълон киламиз
                //ва унга киймат ўзлаштирамиз.
```

Қиймат бериш амали ўзгарувчи аниқланиши билан унга бошланғич қиймат ўзлаштирилишини англатади. Кейинчалик, бу ўзлаштирилган қийматни ўзгартиришингиз ҳам мумкин.

Синфнинг ўзгарувчи–аъзосига қандай қиймат ўзлаштирилди? Бунинг учун синфда конструктор деб аталувчи махсус функция – аъзо ишлатилади. Зарурий вақтда конструктор бир нечта параметрни қабул қилади. Лекин ҳеч қандай типдаги қиймат қайтармайди. Конструктор – бу синф номи билан устма – уст тушадиган синф методидир.

Синфда конструкторни эълон қилиниши билан деструкторлар ҳам аниқланиши лозим. Агарда конструктор синф объектини тузиш ва унинг ўзгарувчи – аъзоларига қиймат бериш вазифасини бажарса, деструктор мавжуд объектнинг хотирадан ўчиради. Деструкторлар синф номи олдига тильда (~) белгисини қўйиш орқали аниқланади. Деструкторлар ҳеч қандай аргумент қабул қилмайди ва ҳеч қандай қиймат қайтармайди. `Mushuk` синфининг деструктори қуйидагича кўринишда аниқланади:

```
~Mushuk()
```

Бошланғич берилган конструктор ва деструкторлар

Агарда сиз конструктор ёки деструкторни аниқламасангиз, сиз учун бу ишни компиляторнинг ўзи бажаради. Стандарт конструктор ва деструкторлар бирорта аргумент қабул қилмайди ва ҳеч қандай амал бажармайди.

6.3. – листинг. Конструктор ва деструкторларни қўлланилиши

```
# include <iostream.h>

class Mushuk
{
```

```
 public:
  Mushuk(int BoshlYosh);
~Cat( );
int GetYosh();
void SetYosh(int yosh);
void Miyovlash();
private:
int itsYosh;
};

// Mushuk синфи конструктори
Mushuk::Mushuk(int BoshlYosh)
{
 itsYosh= BoshlYosh;
}

Mushuk::~Mushuk( )
{

}
// GetYosh функцияси itsYosh ўзгарувчи
// аъзосининг кийматини кайтаради
int Mushuk:: GetYosh()
 {
   return itsYosh;
 }
//SetYosh функцияси itsYosh ўзгарувчи –
// аъзосига киймат ўзлаштиради.
void Mushuk::SetYosh(int yosh)
{
itsYosh= yosh;
}
// Meow функцияси экранга Miyov ёзувини
// чикариш учун ишлатилади.
void Mushuk:: Meow()
{
cout << “Miyov.\n”;
}
```

```
int main()
{
Mushuk Frisky(5);
Frisky.Meow();
cout << "Frisky"<<Frisky.GetYosh();
cout <<"yoshdagi mushuk.\n";
Frisky.Meow();
Frisky.SetYosh(7);
cout<< "Hozir Frisky "<<Frisky.GetYosh();
cout << "yoshda.\n";
return 0;
}
```

НАТИЖА

```
Miyov.
Frisky 5 yoshda.
Miyov.
Hozir Frisky 7 yoshda.
```

`const` спецификатори орқали синф функция аъзоларини эълон қилиш.

C++ тилида синф методларини шундай эълон қилиш мумкинки, улар орқали синфнинг ўзгарувчи – аъзоларининг қийматини ўзгартириш мумкин бўлмайди. Бундай аъзо – функцияларни эълон қилиш учун `const` калит сўзи ишлатилади. Масалан, ҳеч қандай аргумент қабул қилмай, `void` типидаги қиймат қайтарадиган `SomeFunction()` номли синфнинг функция аъзосини шу тарзда эълон қиламиз:

```
void SomeFunction() const;
```

Синф аъзо – ўзгарувчиларига мурожаат қилувчи кўпгина функциялар худди шундай `const` калит сўзи орқали ифодаланиши лозим. Масалан, `Mushuk` синфида аъзо – ўзгарувчиларга мурожаат қилувчи иккита функция бор.

```
void SetYosh( int yosh );
int GetYosh ( );
```

`SetYosh()` функциясини эълон қилишда `const` спецификаторини ишлатиш мумкин эмас, чунки бу функция

объект аъзо–ўзгарувчиси қийматини ўзгартиради. Лекин GetYosh() функциясини эълон қилишда албатта, const калитли сўзини ишлатишимиз лозим, чунки бу функция объектнинг ўзгарувчи аъзосининг қийматини ўзгартирмайди. Демак, Mushuk синфининг аъзо функцияларини қуйидагича эълон қилиш керак экан:

```
void  SetYosh (int yosh);
int GetYosh() const;
```

Синфларни бошқа синфлардан ташкил топиши

Мураккаб синфларни ҳосил қилишда олдин уни ташкил 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 ташкил топган. Чизиқ эса икки нуқта орқали аниқланади. Ҳар бир нуқта x ва y координаталар ёрдамида аниқланади. 6.8. – листингда Turtburchak синфи кўрсатилган. Тўғри тўртбурчак тўртта нуқтани бирлаштирувчи тўртта чизиқ ёрдамида аниқланади. Бунда ҳар бир нуқта графикда координаталарга эга бўлади. Шунинг учун олдин ҳар бир нуқтанинг x, y координаталарини сақлаш учун Nuqta синфи эълон қилинган. 6.9.–листингда иккала синфнинг эълон қилиниши кўрсатилган.

6.8. – листинг. Нуқта ва тўғритўртбурчакнинг эълон қилиниши

```
#include <iostream.h>
class Nuqta
{
public:
void SetX(int x) {itsX = x; }
void SetY(int y) {itsY = y; }
int  GetX() const {return itsX}
int  GetY() const {return itsY }
```

```
private:
int itsX;
int itsY;
}

class Turtburchak
{
public:
Turtburchak(int  Yuqori,int  Chap,int  Quyi,int
Ung) ;
~Turtburchak() { }
int GetYuqori() const {return itsYuqori; }
int GetChap() const {return itsChap; }
int GetQuyi() const {return itsQuyi;}
int GetUng() const  {return itsUng;}
Nuqta        GetYuqoriChap()        const{return
itsYuqoriChap;}
Nuqta GetQuyiChap() const {return itsQuyiChap;}
Nuqta GetYuqoriUng() const {return itsYuqoriUng;
}
Nuqta GetQuyiUng ( ) const { return itsQuyiUng;}

void SetYuqoriChap(Nuqta Urni)
{
itsYuqoriChap= Urni;
}

void  SetQuyiChap (Nuqta Urni)
{
itsQuyiChap= Urni;
}

void  SetYuqoriUng(Nuqta Urni)
{
itsYuqoriUng= Urni;
}

  void SetQuyiUng(Nuqta Urni)
 {
  itsQuyiUng= Urni;
```

```
    }

void SetYuqori(int yuqori){ itsYuqori = yuqori;}
void SetChap (int chap) {itsChap = chap;}
void SetQuyi ( int quyi) { itsQuyi = quyi;}
  void SetUng ( int ung) { itsUng = ung;}
  int GetMaydon ( ) const;
  private:
  Nuqta itsYuqoriChap;
  Nuqta itsYuqoriUng;
  Nuqta itsQuyiChap;
  Nuqta itsQuyiUng;
  int itsYuqori;
  int itsChap;
  int itsQuyi;
  int itsUng;
  }
```

6.9. – листинг. Rect. cpp файли мазмуни

```
  # include "rect.hpp"
  Turtburchak::Turtburchak(int yuqori,
  int chap, int quyi, int ung)
  {
   itsYuqori = yuqori;
   itsChap = chap ;
   itsQuyi = quyi;
   itsUng = ung;

  itsYuqoriChap.SetX(chap);
  itsYuqoriChap.SetY(yuqori);

  itsYuqoriUng.SetX (ung);
  itsYuqoriUng.SetY (yuqori );

  itsQuyiUng.SetX (ung);
  itsQuyiUng.SetY (quyi );

  itsQuyiChap.SetX ( chap );
  itsQuyiChap.SetY ( Quyi );
  }
```

```
// Тўгритўртбурчакни юзасини хисобловчи
// функцияни аниклаймиз
int Turtburchak :: GetMaydon ( ) const
{
int Eni = itsUng – itsChap;
int Buyi = itsYuqori – itsQuyi;
return (Eni * Buyi);
}
int main()
{
Turtburchak=YTurtburchak (100, 20, 50, 80)
int Maydon=YTurtburchak.GetMaydon();
cout << “Maydoni:” << Maydon << “\n”;
cout << “Yuqori chap x ning koordinatasi:”;
cout<< YTurtburchak.GetYuqoriChap().GetX();
return 0;
}
```

НАТИЖА:

```
Maydoni :   3000
Yuqori chap x ninig koordinatasi: 20
```

САВОЛЛАР

- Очиқ (public) ва ёпиқ (private) ўзгарувчи – аъзолар орасида қандай фарқ бор?
- Синфнинг функция аъзолари қачон ёпиқ бўлиши лозим?
- Синфнинг функция аъзолари қачон очиқ бўлиши лозим?
- Агарда Mushuk синфининг иккита объектини эълон қилсак, уларнинг ItsYosh ўзгарувчи аъзолари қиймати турлича бўлиши мумкинми?
- Синф объектини ҳосил қилишда қандай функция чақирилади?
- Синф объекти учун ажратилган хотира майдонини тозалашда қандай функция чақирилади.

ТАЯНЧ ИБОРАЛАР

синф, объект, синф аъзоси, аъзо ўзгарувчи, аъзо функция, синфни эълон қилиш, объектни эълон қилиш, очиқ ва ёпиқ аъзо ўзгарувчилар, конструктор, деструктор, методларни аниқлашда const калит сўзи

Цикллар

Ҳар қандай дастурнинг структураси тармоқланиш ва цикллар тўпламининг комбинациясидан иборат бўлади. Олдинги мавзуларда (4-маърузада) дастурнинг тармоқланиши `if` оператори орқали ташкил этилишини кўриб чиққан эдик. Ушбу маърузада биз қуйидагиларни билиб оламиз:

3. Цикллар нима ва улар қандай ишлатилади ?
4. Цикларни ташкил этишнинг қандай усуллари бор ?
5. Кўп тармоқланувчи ҳолда `if/else` конструкцияси ўрнига бошқа конструкцияни ишлатилиши

Цикларни ташкил этиш.

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.

goto оператори тарихи.

Дастурлашни илк даврларида кичикроқ ҳажмдаги ва етарлича содда дастурлар ишлатилар эди. Бундай дастурларда цикллар `нишон`лардан, 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. `goto` операторига мисол 7.1.- листингда келтирилган.

7.1. – листинг. `goto` оператори ёрдамида цикл ташкил етиш .

```
include  <iostream.h>
 int main()
{
int counter=0; // счётчикни инициализация килиш
loop:
counter ++ ;   //  циклни бошланиши
cout <<"counter: " <<counter<< "\n";
```

```
  if(counter <s) //кийматни 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 булишига 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''.

while оператори орқали цикларни ташкил этиш.

while операторини ёрдамида цикларни ташкил этишда операциялар кетма-кетлиги циклнинг давом этиш шарти «тўғри» бўлсагина унинг навбатдаги операциялари амалга оширилади. 7.1. – листингдаги дастурда counter ўзгарувчиси қиймати токи 5 га тенг бўлгунга қадар ошиб борар эди. 7.2. – листингда худди шу алгоритм while операторини ёрдамида амалга оширилган.

7.2. – листинг. while операторини ёрдамида циклни ташкил этиш

```
include <iostream.h>
int main()
{
int counter=0; //Бирламчи кийматни 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 операторини 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. 7.3. - листингда while оператори конструкциясида мураккаброқ шартларни қўйилишига мисол келтирилган .

7.3. – листинг. while конструкциясидаги мураккаб шартлар.

```
include <iostream.h>
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 нукта 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

Листингда келтирилган 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.

9 – 12 – сатрларда қийматлар киритилади. 15 – сатрда эса циклни давом эттиришнинг қуйидаги учта шарти текширилади:

6. `kichik` ўзгарувчиси қиймати `katta` ўзгарувчиси қийматидан ошмаслиги.
7. `katta` ўзгарувчиси қиймати манфий ва нолга тенг эмаслиги
8. `kichik` ўзгарувчиси қиймати `MaxKichik` қийматидан ошиб кетмаслиги

17 – сатрда эса `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 ва бошқарув 24 – сатрга ўтади.

`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 мисол 7.4. – листингда 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.

7.4. – листинг. break ва continue операторларининг қўлланилиши .

```
include <iostream.h>
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) //мақсадли нуқтага
// тенглигини текшириш
{
cout << "Maqsadga erishildi !;
break;
}
```

```
katta -= 2;
}
cout<< "\n Kichik son:" << kichik <<
<<" 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 ташкил этиш учун шарт сифатида `true` мантиқий ўзгармас қўлланилади. Бу 7.5. - листингда кўрсатилган.

7.5. – листинг. `while` операторини қўллашга оид яна бир мисол.

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

НАТИЖА:

```
Counter:  11
```

do...while конструкцияси ёрдамида цикл ташкил этиш .

Айрим ҳолларда while оператори ёрдамида цикларни ташкил 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. Бу ҳолга 7.6. – листингда мисол келтирилган.

7.6.–листинг. while цикли танасидаги амаллар бажарилмай қолиши

```
 # include <iostream.h >
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 конструкциясида цикл шарти унинг танасидаги операциялар бир марта бажарилгандан сўнг текширилади. Бу цикл операторларини ҳеч бўлмаганда бир марта бажарилишини кафолатлайди.

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

7.7.–листинг. do...while конструкциясининг қўлланилиши

```
# include  <iostream.h>
int main()
{
int counter;
cout<<"How many hellos ?" ;
cin >>counter;
do
{
cout << "hello \h" ;
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
```

for оператори.

while оператори ёрдамида цикларни ташкил 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.

7.8. – листинг. `while` операторининг ишлатилишига яна бир мисол.

```
# include < iostream. h>
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 циклидан фойдаланишга мисол 7.9. – листингда келтирилган.

7.9. – листинг. for циклининг қўлланилишига мисол.

```
#include< iostream. h>
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 операторини 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.

6. Цикл счетчигига бошланғич қиймат ўзлаштирилади.
7. Циклни давом этиш шартидаги ифода қиймати ҳисобланади.
8. 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. 7.10. – листингда иккита счётчикли `for` операторининг қўлланилишини кўриб чиқамиз.

7.10.–листинг. `for` циклида бир нечта счётчикни қўлланилиши

```
# include< iostream. h>
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.

7.11. – листинг. for циклининг нол параметрлари.

```
 # include < iostream. h>
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 7.8. – листингда келтирилган `while` циклини бажарилишига ўхшаш тарзда амалга оширилади. 4-сатрда `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)
```

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 7.12 – листингда кўрсатилган.

7.12. – листинг. `for` операторларини параметрларсиз қўлланилиши.

```
# include< iostream. h>
int main()
{
int counter=0;
int max;
cout<< “How many hellos?”;
cin>> max;
for( ; ; )
{
if (counter <max)
{
cout << “Hello! \h”;
counter++;
}
else
break;
```

```
}
return 0;
}
```

НАТИЖА:

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

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. Танаси бўш бўлган for циклига мисол 7.13. – листингда келтирилган.

7.13. – листинг. For циклининг танаси бўш бўлган ҳолда қўлланилиши.

```
# include< iostream. h>
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 ташкил этилган цикл ички цикл деб айтилади. Бу ҳолда ички цикл ташқи циклни ҳар бир итерациясида тўлиқ бажарилади. 7.14. – листингда матрица

элементларини ички цикл орқали тўлдирилиши намойиш қилинган.

7.14. – листинг. Ички цикллар.

```
# include< iostream. h>
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<colums; j++)
cout << the char;
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.h>
int main()
 {
```

```
    for ( int i = 0; i<5; i++ )
        {
         cout << " i: " << i <<  endl ;
        }
    i=7; // i кўриниш соҳаси 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. h>
int main ()
{
 int i;
 for ( int i = 0; i<5; i++ )
   {
    cout << " i: " << i <<  endl ;
   }
 i=7;  //Энди  i  барча  компиляторлар  томонидан
 //хатоликсиз қабул қилиниши мумкин.
return 0;
}
```

switch оператори

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. Натижада дастурни тармоқланиши нисбатан самаралироқ бўлади. switch операторининг синтаксиси қуйидагичадир.

```
switch(ифода)
{
case Биринчи киймат:    оператор;
                        break;
```

```
case Иккинчи киймат:    оператор;
                        break;
 .....
case N нчи киймат:      оператор;
                        break;
default:                оператор;
}
```

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.

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 ташкил 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.

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 қўлланилишига мисол 7.16.-листингда келтирилган.

7.16. – листинг. `switch` операторининг қўлланилиши

```
include <iostream.h>
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 9 – сатрда ёзилган `kichik son` хабари чиқарилади. Ва ундан кейин ёзилган `break` оператори `switch` конструкциясини бажарилишини якунлайди. Агарда 5 сони киритилса бошқарув 11 – сатрга берилади ва унга мувофиқ хабар чиқарилади. Ундан кейинги токи `break` сўзигача барча сатрлар кетма – кет бажарилади.

Агарда дастурга 5 дан катта сон киритилса 17 – сатрдаги `default` бажарилади, яъни экранга `Big` хабари чиқарилади.

`switch` оператори ёрдамида меню командаларини бажариш.

`for(;;)` оператори орқали чексиз цикларни ташкил 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 циклни ташкил этишга мисол 7.17. – листингда келтирилган.

7.17. – листинг. Чексиз циклга мисол.

```
include< iostream. h>
int menu ( )
void DoTaskOne ( );
void DoTaskMany (int);
int main()
{
 bool exit=false;
 for (; ;)
 {
 int choice=menu ( );
 switch (choice)
 {
 case (1): DoTaskOne ( );
           break;
 case (2): DoTaskMany (2);
           break;
 case (3): DoTaskMany (3);
           break;
case (4):  continue;
           break;
case (5):
          exit=true;
          break;
         default:
```

```
cout << “Please select again! \n”;
          break;
}    // switch блоки тугади.
if (exit)
break;
}
return 0;

int menu()
 {
  int choice;
   cout << **** Menu **** \n\n”;
   cout << “(1) Choice one/ \n”;
   cout << “(2 Choice two \n”;
   cout << “(3) Choice three \n”;
   cout << “(4) Redisplay menu \ n”;
   cout << “(5) Quit   \n\n”;
   cin >> choice;
   return choice;
 }
 void DoTaskOne()
{
cout << “Task One! \n”;
}
void DoTaskMany(int which)
{
  if (which == 2)
    cout << “Task Two! \n”;
else
cout << “Task Three! \n”;
}
```

НАТИЖА:

```
* * * Menu * * * * *
(1) Choice one.
(2) Choice two.
(3) Choice three.
(4) Redisplay menu.
(5) Quit.
:  1
Task One ;
* * * Menu * * * * *
```

```
(1) Choice one.
(2) Choice two.
(3) Choice three.
(4) Redisplay menu.
(5) Quit.
:  3
* * * Menu * * * * *
(1) Choice one.
(2) Choice two.
(3) Choice three.
(4) Redisplay menu.
(5) Quit.
:  5
```

САВОЛЛАР

3. `for` циклида бир нечта счётчик ишлатиш мумкинми?
4. Нима учун `goto` оператори кўп ишлатилмайди?
5. Танасида бирор амал ёзилмаган `for` оператори ёрдамида цикл ташкил этиш мумкинми?
6. `for` цикли ичида `while` циклини ташкил этиш мумкинми?
7. Ҳеч қачон тугалланмайдиган цикл ташкил этиш мумкинми?
8. `while` ва `do .. while` операторларини қандай фарқлари бор?

ТАЯНЧ ИБОРАЛАР

Цикл, итератив жараён, `goto` калитли сўзи, while оператори, `break` ва `continue` операторлари, `do...while` конструкцияси, `for` оператори, ички цикллар, счётчикнинг кўриниш соҳаси, `switch` оператори

Кўрсаткичлар.

C++ дастурлаш тилида хотирага бевосита кўрсаткичлар орқали мурожаат қилиш тилнинг энг катта имкониятидир. Ушбу бўлимда биз қуйидаги саволларга жавоб топамиз.

34: Кўрсаткич нима?

35: Кўрсаткичлар қандай эълон қилинади ва қўлланилади ?

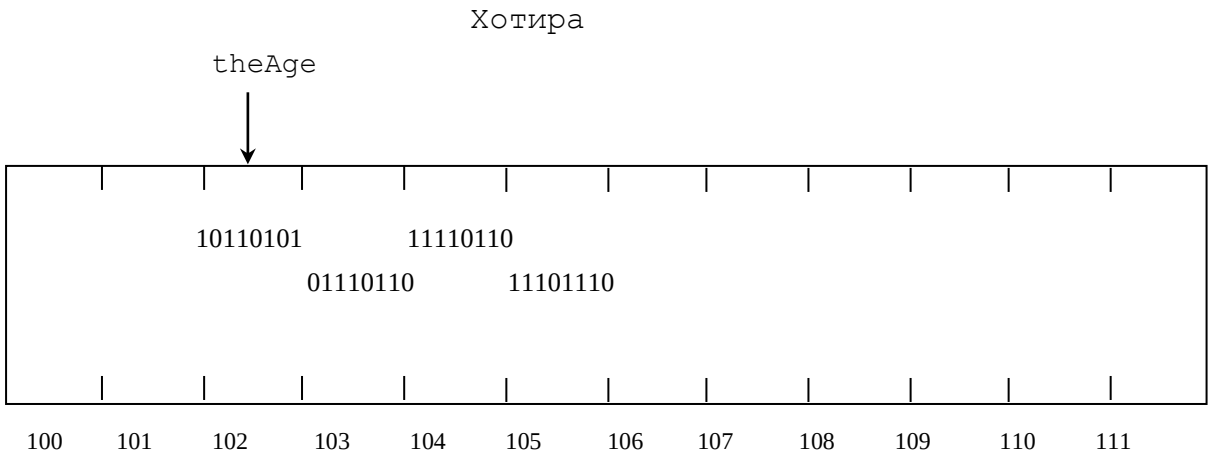
36: Хотира билан қандай ишлаш мумкин?

Ушбу машғулотда биз кўрсатгичлар билан ишлашнинг асосий принциплари билан танишамиз.

Кўрсаткич нима ?

Кўрсаткич – бу компютер хотираси ячейкасининг адреси ёзилган ўзгарувчидир.

Кўрсаткичлар қандай ишлашини билиш учун машина хотираси ташкил этилишининг таянч принципларини билишимиз лозим. Машина хотираси номерланган ячейкалар кетма-кетлигидан иборатдир. Ҳар бир ўзгарувчининг қиймати унинг адреси деб аталувчи алоҳида хотира ячейкасида сақланади. 8.1.- расмда `theAge` номидаги, хотирадан тўрт байт жой эгаллайдиган бутун қийматли ўзгарувчининг хотирада ифодаланиши кўрсатилган.




```
Ҳар бир ячейка = 1 байт.
       unsigned long int theAge = 4байт = 32 бит.
       theAge ўзгарувчи номи биринчи байтни кўрсатади.
       theAge ўзгарувчининг адреси = 102.
```

8.1. – расм. `TheAge` *ўзгарувчининг хотирада сақланиши*

Турли компютерларда хотирани адреслаш турлича қоида асосида ташкил этилади. Кўп ҳолларда дастурчилар учун бирор бир ўзгарувчини аниқ адресини билиш зарур эмас. Зарурат туғилганда бундай ахборотни адрес оператори `(&)` ёрдамида олиш мумкин. Бу операторнинг қўлланилишига мисол 8.1. – листингда келтирилган.

8.1. – листинг. Адрес оператори .

```
#include < iostheam.h >
int main()
{
unsigned short shortVar = 5;
unsigned long longVar = 65535;
long sVar = - 65535;
```

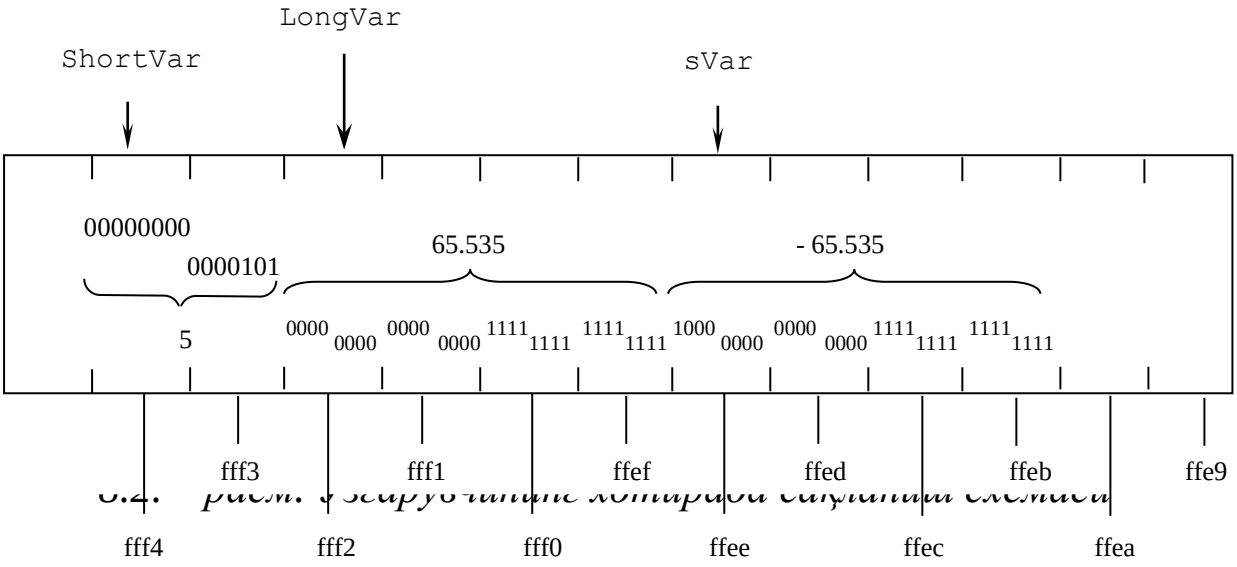
```
cout << “shortVar :\t” << shortVar ;
cout << “Address of shortVar:\t” ;
cout<<&shortVar<< “\n”;
cout << “longVar :\ t” << longVar ;
cout << “ Address of LongVar: \ t” ;
cout<< &longVar<< “\n”;
cout << “ sVar :\t” << sVar;
cout << “Address of sVar:\t”;
cout << &sVar <<“\n” ;
return  0;
}
```

НАТИЖА:

```
shortVar : 5   Address of  shortVar :  0x8fc9:fff4
longVar:65535  Address of  longVar:    0x8fc9:fff2
sVar:   -65535 Address of sVar :       0x8fc9:ffee.
```

ТАҲЛИЛ

Дастурнинг бошида учта ўзгарувчи эълон қилинади ва уларга қиймат ўзлаштирилади: 4– сатрда `unsigned short` типида, 5- сатрда `unsigned long` типида ва 6 – сатрда `long` типида. Кейин эса 8 – 18 – сатрларда бу ўзгарувчиларнинг қийматлари ва адрес оператори (&) ёрдамида олинган адреслари ҳақида маълумотлар чиқарилади. Дастур ишга туширилиши билан 80386 процессорли компьютерда `shortVar` ўзгарувчиси қиймати 5 га, унинг адреси эса `0x8fc9:fff4` тенглиги аниқланади. Ўзгарувчини жойлашиш адреси компютер орқали танланади ва у дастурнинг навбатдаги ишга туширилишида ўзгариши мумкин. Шунинг учун сизнинг натижаларингиз олдингиларидан фарқ қилиши мумкин. Лекин биринчи ва иккинчи ўзгарувчиларнинг адреслари фарқи доимо ўзгармай қолади. `short` типи икки байт жой эгаллашидан бу фарқ 2 байтни ташкил этади. Иккинчи ва учинчи адреслар орасидаги фарқ 4 байтни ташкил этади. Ҳақиқатан ҳам `long` типидаги ўзгарувчи тўрт байт жой эгаллайди. Бу ўзгарувчиларни хотирага жойлаштириш тартиби 8.2. – расмда келтирилган.



3.4-расм. Ўзгарувчиларнинг хотирада сақланиш схемаси

Кўрсаткичнинг адресларни сақлаш воситаси сифатида қўлланилиши.

Дастурнинг ҳар бир ўзгарувчиси ўзининг адресига эгадир. Бу адресни сақлаш учун эса ўзгарувчига кўрсаткич эълон қилиш керак. Адреснинг ўзининг қийматини билиш эса унчалик шарт эмас.

Масалан, `howOld` ўзгарувчиси `int` типига эга. Бу ўзгарувчини адресини сақловчи `pAge` кўрсаткичини эълон қилиш учун қуйидаги дастур фрагментини ёзиш лозим.

```
int *pAge = 0;
```

Бу сатрда `pAge` ўзгарувчиси `int` типига кўрсаткич сифатида эълон қилинаяпти.

`pAge` ўзгарувчиси бошқа ўзгарувчилардан умуман фарқ қилмайди. Бутун типли ўзгарувчини эълон қилишда биз унда бутун сон сақланишини кўрсатамиз. Қачонки ўзгарувчи бирор бир типга кўрсаткич деб эълон қилинса, бу кўрсаткич ўша типдаги ўзгарувчининг адресини ўзида сақлайди. Демак, кўрсаткичлар ҳам ўзгарувчиларнинг алоҳида тури экан.

Бу мисолда `pAge` ўзгарувчисига нол қиймат ўзлаштирилаяпти. Қиймати нолга тенг бўлган кўрсаткичлар бўш кўрсаткичлар дейилади. Кўрсаткич эълон қилинганда унга албатта бирор бир қиймат ўзлаштирилиши лозим. Агарда бу олдиндан аниқ бўлмаса унга 0 қиймат бериш зарур. Қиймат ўзлаштирилмаган кўрсаткичлар кейинчалик кўплаб нохуш

ҳодисаларга сабаб бўлиши мумкин. Биз улар билан кейинроқ батафсил танишамиз.

pAge кўрсаткичи эълон қилинганда унга 0 қиймат берилиб, кейинчалик эса унга бошқа қиймат, масалан howOld ўзгарувчиси адресини ўзлаштириш мумкин. Қуйида буни амалга оширилиши кўрсатилган.

```
unsigned short int howOld=50; //ўзгарувчини эълон
//қиламиз.
unsigned  short int *pAge=0; //кўрсаткични эълон
//қиламиз.
pAge= &howOld; //pAge кўрсаткичга howOld ўзгарувчи
//адресини ўзлаштирамиз.
```

Биринчи сатрда unsigned short int типидаги ўзгарувчи эълон қилинган ва унга 50 қиймат ўзлаштирилган. Иккинчи сатрда эса unsigned short int типидаги pAge типига кўрсаткич эълон қилинган ва унга 0 қиймат ўзлаштирилган. Тип номи ва ўзгарувчи орасидаги юлдузча (*) белгиси ундан кейин келган ўзгарувчини кўрсаткич эканлигини англатади.

Охирги сатрда pAge кўрсаткичига howOld ўзгарувчиси адреси ўзлаштирилаяпти. Бунда адрес оператори (&) howOld ўзгарувчисини адресини олаяпти ва у ўзлаштириш амали орқали pAge кўрсаткичига ўзлаштирилаяпти.

Бу ҳолатда pAge ўзгарувчисининг қиймати howOld ўзгарувчининг адресига тенгдир. Қараб чиқилган дастур кодидаги охирги икки сатрни битта сатрга бирлаштириш мумкин.

```
unsigned short int = 50;
unsigned short int * pAge = & howOld;
```

Кўрсаткич номлари.

Кўрсаткичлар оддий ўзгарувчи эканлигидан улар учун ҳам ўзгарувчиларнинг номлашда ўринли бўлган барча қоидалар ўринлидир. Кўрсаткичларни бошқа ўзгарувчилардан фарқлаш учун кўп дастурчилар уларни номидан олдин “p” (инглизчадан Pointer) белгисини қўядилар. Масалан, pAge ёки pNumber.

Билвосита мурожаат оператори.

Билвосита мурожаат оператори кўрсаткичда адреси сақланаётган қийматни кўрсаткич номидан фойдаланиб олиш имконини беради.

Кўрсаткичлардан фарқли равишда оддий ўзгарувчиларга мурожаат қилишда уларнинг тўғридан – тўғри қийматига мурожаат қилинади. Масалан, `unsigned short int` типидаги янги ўзгарувчини эълон қилдик, кейин эса унга бошқа ўзгарувчининг қийматини ўзлаштирдик. Буни қуйидагича ёзиш мумкин.

```
unsigned short int yourAge;
yourAge = &howOld;
```

Билвосита мурожаат оператори орқали эса кўрсаткичда адреси сақланган ўзгарувчининг қиймати олинади. `YourAge` янги ўзгарувчига адреси `pAge` кўрсаткичида сақланаётган `howOld` ўзгарувчисининг қийматини бериш учун қуйидаги дастур фрагменти ёзилади.

```
unsigned short int yourAge;
yourAge = * pAge;
```

Билвосита мурожаат оператори `pAge` ўзгарувчисининг «адресида сақланаётган қийматни» олиш учун ишлатилади. Демак, юқоридаги дастур фрагменти «`pAge` *кўрсаткичида адреси сақланаётган қийматни олиб, уни* `yourAge` *ўзгарувчисига ўзлаштирилсин»* каби ўқилади.

Кўрсаткичлар, адреслар ва ўзгарувчилар.

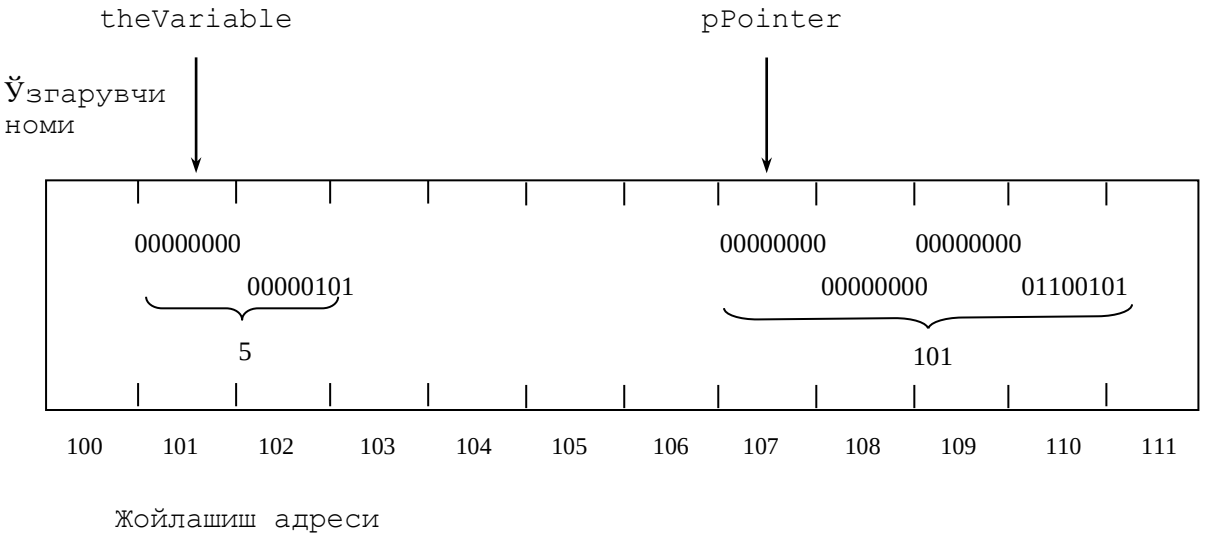
C++ дастурлаш тили имкониятларини ўрганишда, биринчи навбатда кўрсаткичлар, кўрсаткичда сақланаётган адреслар, кўрсаткичда сақланаётган адресда ёзилган қийматлар ўртасидаги фарқларни билишингиз керак бўлади.

Яна бир дастур фрагментини қараймиз.

```
int theVariable = 5;
int * pPointer = &theVariable;
```

Биринчи сатрда бутун типли бўлган `theVariable` ўзгарувчиси эълон қилинди ва унга 5 қиймат ўзлаштирилди. Кейинги сатрда `int` типига кўрсаткич эълон қилинди ва унга `theVariable` ўзгарувчисининг адреси ўзлаштирилди.

pPointer кўрсаткичи theVariable ўзгарувчиси адресини ўзида сақлайди. pPointer да ёзилган адресда сақланувчи қиймат 5 га тенг. 8.3. – расмда бу ўзгарувчилар структураси схематик тарзда кўрсатилган.



8.3. – расм. Хотирани тақсимланиш схемаси

Кўрсаткичлар ёрдамида маълумотларга мурожаат қилиш.

Кўрсаткичга бирор–бир ўзгарувчининг адреси ўзлаштирилса, бу кўрсаткичдан шу ўзгарувчининг қиймати билан ишлашда ҳам фойдаланиш мумкин бўлади. 8.2. - листингда локал ўзгарувчи қийматига унинг кўрсаткичи орқали мурожаат қилишга мисол келтирилган.

8.2. – листинг. Маълумотларга кўрсаткич орқали мурожаат қилиш.

```
// 8.2. листинг. Кўрсаткичларнинг қўлланилиши

#include <iostream.h>
typedef unsigned short int ushort;

int main()
{
ushort myAge;
ushort *pAge = 0;
myAge = 5;
cout<< “MyAge: ” << myAge   << “ \n”;
pAge = &myAge;
```

```
cout << "*pAge : " << *pAge  << "\n\n";
cout << "*pAge = 7 \n" ;
*pAge = 7 ;
cout << " *pAge : " <<*pAge << "\n" ;
cout << " myAge:" << myAge << " \n\n";
cout << " *pAge = 9 \n" ;
myAge = 9 ;
cout << " myAge:" << MyAge << " \n";
cout << " *pAge : " <<*pAge << "\n" ;
return 0;
}
```

НАТИЖА:

```
myAge:  5
*pAge:  5

*pAge = 7
*pAge  : 7
 myAge : 7

myAge = 9
myAge: 9
*pAge : 9
```

ТАҲЛИЛ

Дастурда `unsigned short` типидаги `myAge` номли ўзгарувчи ва шу типга кўрсаткич бўлган `pAge` кўрсаткич эълон қилинди. 10–сатрда `myAge` кўрсаткичига 5 қиймат ўзлаштирилди, 11–сатрда эса бу қиймат экранга чиқарилди.

Кейин 12–сатрда `pAge` кўрсаткичига `myAge` ўзгарувчиси қиймати ўзлаштирилди. Билвосита мурожаат оператори орқали `pAge` кўрсаткичида сақланаётган адресда ёзилган қиймат экранга чиқарилди. Кўриб турганингиздек, олинган натижа `myAge` ўзгарувчиси қиймати билан мос келади. 15 – сатрда `pAge` да адреси ёзилган ўзгарувчига 7 қиймат ўзлаштирилди. Бундай операциядан кейин `myAge` ўзгарувчисининг қиймати 7 га тенг бўлади. Бунга ушбу қийматларни экранга чиқариш орқали ишонч ҳосил қиламиз. (16- , 17- сатрлар).

19–сатрда myAge ўзгарувчисининг қиймати яна ўзгарди. Энди бу ўзгарувчига 9 сони ўзлаштирилди. Кейин эса 20 – ва 21 – сатрларда бу қийматларга бевосита (ўзгарувчи орқали) ва билвосита (кўрсаткичнинг билвосита оператори орқали) мурожаат қилдик.

Кўрсаткичда сақланувчи адресдан фойдаланиш.

Кўрсаткичлар билан ишлашда кўп ҳолларда уларда ёзилган адреснинг қийматидан фойдаланилмайди. Олдинги қисмларда кўрсаткичларга ўзгарувчи адреси ўзлаштирилса кўрсаткич қиймати айнан шу адресга тенг бўлиши айтиб ўтилган эди. Бу тасдиқни 8.3. листинг орқали текшириб ўтамиз.

8.3. – листинг. Кўрсаткичда ёзилган қиймат.

```
#include <iostream.h>
int main()
{
int myAge = 5, yourAge = 10;
int *pAge = &myAge;
cout<<“myAge:\t”<<myAge<<”\t
yourAge:\t”<<yourAge <<”\n”;
cout<<“&myAge:\t”<<&myAge<<”\t&yourAge:\t”
<<&yourAge<<”\n”;
cout<<“pAge:\t” <<pAge<< “\n”;
cout <<“*pAge:\t”<<*pAge<< “\n”;
pAge = &yourAge;
cout<<“myAge:\t”<<myAge<<”\tyourAge<<”\t”<<your
Age <<”\n”;
cout<<“&myAge:\t”<<&myAge<<“\tyourAge:”<<“\t”<<&
yourAge;
cout<<“pAge: \t” << pAge <<“\n”;
cout<<“*pAge:\t”<<* pAge<< “\n”;
cout<<“& pAge: \t” << &pAge<< “\n”;
return 0;
}
```

НАТИЖА

```
myAge:    5          your Age:         10
&myAge:    0x 355c   &your Age :       0x 355 E
pAge:      0x 355c
```

```
*pAge:    5
my Age:   5       your Age:        10
&myAge:   0x355C  &your Age :      0x 355E
page :    0x355E
*page :   10
&page:    0x355A
```

(Сизнинг натижаларингиз юқоридагилардан фарқ қилиши мумкин.)

Кўрсаткичлар нима учун керак

Олдинги қисмларда биз бошқа ўзгарувчилар адресини кўрсаткичларга ўзлаштириш процедуралари билан батафсил танишган эдик. Лекин амалиётда кўрсаткичларни бундай қўлланилиши жуда кам учрайди. Нима учун ўзгарувчида сақланаётган қийматдан фойдаланиш учун кўрсаткичларни ишлатишимиз керак деган ҳақли савол туғилади. Юқорида келтирилган мисоллар кўрсаткичнинг ишлаш механизмини намойиш учун келтирилди. Аслида кўрсаткичлар асосан қуйидаги ҳолларда қўлланилади:

51: Маълумотларни хотиранинг объектлар ўртасида динамик тақсимланувчи соҳасида жойлаштириш ва улардан фойдаланиш учун;
52: Синфнинг ўзгарувчи ва функцияларига мурожаат қилиш учун;
53: Функцияларга параметрларларни мурожаат орқали узатиш учун.

Стекли ва объектлар ўртасида динамик тақсимланувчи хотира.

Олдинги мавзуларда хотирани 5 та соҳага ажратган эдик:

54: Глобал ўзгарувчилар соҳаси.
55: Бўш ёки объектлар ўртасида динамик тақсимланувчи хотира.
56: Регистрли хотира.
57: Дастур сегменти.
58: Стекли хотира.

Локал ўзгарувчилар ва функция параметрлари стекли хотирада жойлаштирилади. Дастур кодлари эса дастур сегментида жойлашади. Глобал ўзгарувчилар хотиранинг глобал

ўзгарувчилар соҳасида жойлашади. Регистрли хотира дастурнинг ички хизматчи маълумотларини сақлаш учун хизмат қилади. Хотиранинг қолган қисми эса турли объектлар ўртасида динамик тарзда тақсимланувчи хотира соҳаси, яъни бўш хотирадан иборатдир.

Локал ўзгарувчиларнинг ўзига хос хусусияти шундан иборатки, дастур бошқаруви улар тегишли функциядан чиқиши билан улар локал ўзгарувчилар учун ажратилган хотира соҳаси бўшатилади, яъни бу ўзгарувчилар ўчирилади.

Глобал ўзгарувчилар эса дастур ишга туширилгандан бошлаб токи ўз ишини якунлагунча хотирадан ўчирилмайди, дастурнинг ихтиёрий жойидан уларнинг қийматидан фойдаланишимиз мумкин бўлади. Лекин бу дастур матни тушунарлилигини анча мураккаблаштиради. Локал ва глобал ўзгарувчилар ўзига хос камчиликларга эга. Локал ўзгарувчилардан фақатгина у эълон қилинган функция ишлаб турганда фойдаланиш мумкин бўлса, глобал ўзгарувчилар дастур ишини бошлагандан токи охиригача глобал ўзгарувчилар соҳасидан жой олиб туради. Динамик хотирадан фойдаланиш бу муаммоларни бутунлай ҳал қилади.

Динамик хотирани ахборотлар ёзилган ячейкаларнинг номерланган тўплами сифатида қараш мумкин. Стек ўзгарувчиларидан фарқли равишда бу хотира ячейкаларини номлаш мумкин эмас. Уларга мурожаат керакли ячейка адресини ўзида сақлаган кўрсаткичлар орқали амалга оширилади.

Маълумки, стек ўзгарувчилари функция ишини тугатиши билан тозаланади. Натижада, барча локал ўзгарувчилар қиймати билан биргаликда хотирадан ўчирилади. Стекдан фарқли равишда динамик хотира дастур ишини тугатгунга қадар бўшатилмайди, бу хотирани бўшатиш билан дастурчиларнинг ўзлари шуғулланишлари лозим.

Динамик хотиранинг яна бир муҳим имконияти шундан иборатки, унинг учун ажратилган хотира соҳаси бўшатилмагунча бу жойдан фойдаланиш мумкин эмас. Яъни, бирор бир қийматни динамик хотирага ёзсак, токи уни у ердан ўчирмагунимизча бу ўзгарувчи учун ажратилган жой бўшатилмайди. Шунинг учун, динамик хотира соҳаси функция билан ишлаш жараёнида ажратилган бўлса, функция ишини тугатгандан кейин ҳам биз ундан бемалол фойдаланишимиз

мумкин бўлади. Хотирани динамик тарзда белгилашни глобал ўзгарувчиларни қўллашдан яна бир устунлиги ундаги маълумотларга фақатгина унинг кўрсаткичига мурожаат қилиш имконига эга бўлган функциялар орқалигина мурожаат қилиш мумкин.

Кириш имконининг бундай тарзда ташкил этилиши жорий маълумотларни ўзгартирилишини қатъий назорат қилиш имкониятини беради.

Бундай услубда ишлаш учун биринчи навбатда динамик хотира соҳаси ячейкаларига кўрсаткич тузиш лозим. Бу қандай амалга оширилишини навбатдаги қисмларда кўриб чиқамиз.

`new` оператори.

Хотиранинг объектлар ўртасидан динамик тақсимланувчи соҳасидан жой ажратиш учун `new` оператори ишлатилади. `new` операторидан кейин хотирага жойлаштириладиган объект типини кўрсатиш лозим. Бу объектни сақлаш учун талаб этиладиган хотира соҳаси ўлчовини аниқлаш учун керак бўлади. Масалан, `new unsigned short int` деб ёзиш орқали биз динамик тақсимланувчи хотирадан икки байт жой ажратамиз. Худди шунингдек, `new long` сатри орқали тўрт байт жой объектлар ўтрасида динамик тақсимланувчи соҳадан ажратилади.

`new` оператори натижа сифатида белгиланган хотира ячейкасининг адресини қайтаради. Бу адрес кўрсаткичга ўзлаштирилиши лозим. Масалан, `unsigned short` типидаги ўзгарувчи учун динамик соҳадан жой ажратиш учун қуйидаги дастур коди ёзилади:

```
unsigned short int *pPointer;
pPointer = new unsigned short int;
```

Ёки худди шу амални битта сатрда ҳам ёзиш мумкин.

```
unsigned short int * pPoiner = new unsigned short int;
```

Иккала ҳолатда ҳам `pPointer` кўрсаткичи `unsigned short int` типидаги қийматни сақловчи динамик соҳа хотирасининг ячейкасини кўрсатиб туради. Энда `pPointer` кўрсаткичини шу типдаги ихтиёрий ўзгарувчига кўрсаткич

сифатида қўллаш мумкин. Ажратилган хотира соҳасига бирор бир қиймат жойлаштириш учун қуйидагича ёзув ёзилади:

```
* pPointer = 72 ;
```

Бу сатр қуйидаги маънони англатади: «pPointer кўрсаткичида адреси сақланаётган хотирага 72 сонини ёзинг». Динамик хотира соҳаси албатта чегараланган бўлади. У тўлиб қолганда new оператори орқали хотирадан жой ажратишга уринсак хатолик юз беради. Бу ҳақда кейинроқ тўхталиб ўтамиз.

delete оператори.

Агарда ўзгарувчи учун ажратилган хотира керак бўлмаса уни бўшатиш зарур. Бу ўзидан кейин кўрсаткич номи ёзиладиган delete оператори ёрдамида амалга оширилади. delete оператори кўрсаткич орқали аниқланган хотира соҳасини бўшатади. Шуни эсда сақлаш лозимки, динамик хотира соҳасидаги адресни ўзида сақловчи кўрсаткич локал ўзгарувчи бўлиши мумкин. Шунинг учун бу кўрсаткич эълон қилинган функциядан чиқишимиз билан кўрсаткич ҳам хотирадан ўчирилади. Лекин new оператори орқали бу кўрсаткичга динамик хотирадан ажратилган жой бўшатилмайди. Натижада хотиранинг бу қисми киришга имконсиз бўлиб қолади. Дастурчилар бу ҳолатни хотиранинг сирқиб кетиши, ёки йўқолиши (утечка памяти) деб тавсифлайдилар. Бу тавсиф ҳақиқатга бутунлай мос келади, чунки дастур ишини якунлагунча хотирани бу қисмидан фойдаланиб бўлмайди.

Хотирани ажратилган қисмини бўшатиш учун delete калитли сўзидан фойдаланилади. Масалан:

```
delete pPointer;
```

Бунда кўрсаткич ўчирилмайди, балки унда сақланаётган адресдаги хотира соҳаси бўшатилади. Белгиланган хотирани бўшатилиши кўрсаткичга таъсир қилмайди, унга бошқа адресни ўзлаштириш ҳам мумкин. 8.4. – листингда динамик ўзгарувчи учун қандай хотира ажратилиши, уни ишлатиш ва ажратилган хотирани бўшатишга оид мисол келтирилган.

8.4. – листинг. Динамик хотирани ажратиш, ундан фойдаланиш ва уни бўшатиш.

```
# include < iostream.h>
```

```
int main()
{
int   local variable = 5;
int * pLocal = & local variable;
pHeap = 7;
cout  <<  “local  variable:”  <<  local  variable
<<”\n”;
cout << “ *pLocal: ”<< *pLocal  << “\n”;
cout << “ *pHeap: ” << *pHeap  << “\n”;
delete pHeap;
pHeap = new int;
*pHeap = 9;
cout <<“*pHeap:”<<*pHeap << “\n”;
delete pHeap;
return 0;
}
```

НАТИЖА:

```
local variable: 5
*pLocal:  5
*pHeap:  7
*pHeap:  9
```

Хотиранинг сирқиб кетиши нима ?

Кўрсаткичлар билан эътиборсиз ишлаш натижасида хотиранинг сирқиб кетишига йўл қўйиш мумкин. Бу кўрсаткич мурожаат қилиб турган хотира бўшатилмасдан, шу кўрсаткичга янги қиймат ўзлаштирилган вақтда рўй беради. Бундай ҳолатга қуйида мисол келтирилган:

```
1.unsigned short int*pPointer=new unsigned short int;
2. *pPointer = 72;
3.  pPointer = new unsigned short int;
4. *pPointer = 84;
```

1 – сатрда кўрсаткич эълон қилиняпти ва `unsigned short int` типидаги ўзгарувчини сақлаш учун хотира ажратилаяпти. Навбатдаги сатрда ажратилган соҳага 72 қиймат ёзилди. 3 – сатрда эса кўрсаткичга хотира соҳасининг бошқа адреси ўзлаштирилди ва у адресдаги соҳага 84 қиймат ўзлаштирилди. Бу операциялардан кейин 72 қийматни сақлаб турган хотира соҳасига мурожаат қилиш имкони йўқолди. Чунки, бу соҳанинг кўрсаткичига янги қиймат берилди. Натижада, резервланган хотирани дастур ишини тугатгунча

умуман ишлатиб бўлмайди. Бундай ҳолларда қуйидагича ёзув тўғри бўлар эди.

```
unsigned short int*  pPointer= new unsigned short int;
*pPointer  = 72;
delete pPointer;
pPointer = new unsigned short int;
*pPointer = 84;
```

Бу ҳолда ўзгарувчи учун ажратилган хотира тўғри бўшатилади.

Хотиранинг объектлар ўртасида динамик тақсимланувчи соҳасига обьектларни жойлаштириш.

Худди бутун типли ўзгарувчини динамик хотирага жойлаштириш сингари ихтиёрий обьектни ҳам объектлар ўртасида динамик тақсимланувчи соҳалда жойлаштиришимиз мумкин. Масалан, агар сиз `Cat` синфи обьектини ҳосил қилиб, бу обьектни ўзгартириш учун унинг адресини ўзида сақловчи кўрсаткич тузишингиз мумкин. Бу ҳолат ўзгарувчини стекка жойлаштиришга ўхшаб қолади. Бу операцияни синтаксиси худди бутун сонли ўзгарувчи учун бўлганидек қуйидагичадир:

```
Cat *  pCat=  new  Cat.
```

Бу ҳолатда `new` оператори синфни бошланғич конструкторини, яъни параметрсиз конструкторини чақиради. Объектни тузишда у стекда ёки динамик хотира соҳасида жойлаштирилишидан қатъий назар доимо у тегишли синф конструктори чақирилади.

Объектни динамик тақсимланувчи хотирадан ўчириш.

`delete` оператори ишлатилганда автоматик тарзда ундан кейин ёзилган кўрсаткичда адреси сақланувчи объект тегишли синф деструктори чақирилади. Қоида бўйича синф деструктори объектнинг динамик хотира соҳасида эгаллаган барча хотира соҳасини бўшатади. Объектни динамик хотирага жойлаштириш ва ўчиришга оид мисол 8.5- листингда кўрсатилган.

8.5. – листинг. Динамик хотира соҳасига объектларни жойлаштириш ва уларни ўчиришга оид мисол.

```
// 8.5. – листинг.
```

```
// Динамик тақсимланувчи соҳада объектларни
//жойлаштириш ва ўчириш
# include <iostream.h>

class SimpleCat
{
public:
SimpleCat();
~SimpleCat();
private:
int itsAge;
}

SimpleCat:: SimpleCat( )
{
cout<< "Contructor called .\n ";
itsYosh= 1;
}
SimpleCat:: ~ SimpleCat( )
{
cout <<"Destructor called .\n";
}
int main()
{
cout << "Simple Cat Fricky… \n";
SimpleCat Frisky;
cout << "SimpleCat *pRags = new SimpleCat… \n";
SimpleCat* pRags = new SimpleCat;
cout<< "delete pRags… \n";
delete pRags
cout<< "Exiting, watch Fricky go … \n";
return 0;
}
```

НАТИЖА

```
SimpleCat Frisky…
Constructor called.
Simple Cat*pRags = new Simple Cat…
Constructor called
delete pRags…
Destructor called
```

```
Exiting, wath Frisky go …
Destructor called.
```

ТАҲЛИЛ

6 – 13 – сатрларда оддий `SimpleCat` синфининг тавсифи берилган. Синф конструкторининг тавсифи 9 – сатрда, унинг танасининг тавсифи эса 15 – 19 – сатрларда берилган. Деструктор эса 10 – сатрда, унинг танаси эса 21 – 24 – сатрларда тавсифланган.

29 – сатрда бу синфнинг стекда жойлашувчи экземпляри тузилади. Бунинг учун `SimpleCat` синфининг конструктори ошкормас тарзда чақирилади. Синфининг иккинчи объекти 31 – сатрда тузилади. Бу ҳолда ҳам конструктор чақирилади. `SimpleCat` синфининг деструктори 33 – сатрда `pRags` кўрсаткичи учун `delete` оператори қўлланилиши натижаси сифатида чақирилади. Функциядан чиқиш вақтида `Frisky` ўзгарувчиси кўриниш соҳаси чегарасидан чиқади ва унинг учун ҳам деструктор чақирилади.

Синф экземпляри аъзоларига мурожаат.

Синфнинг экземпляринг (объектининг) локал ўзгарувчи бўлган аъзоларига мурожаат тўғри мурожаат`(.)` оператори ёрдамида амалга оширилади. Динамик тақсимланувчи соҳада ҳосил қилинган синф экземплярларининг аъзоларига эса қуйидаги тарзда мурожаат қилинади: олдин синф экземплярига уни адресини ўзида сақлаган кўрсаткич орқали мурожаат қилинади (билвосита мурожаат оператори орқали), кейин эса тўғри мурожаат оператори орқали унинг аъзоларига мурожаат қилинади. Масалан, `GetAge()` функция аъзосини чақириш учун қуйидагича ёзув ёзиш лозим:

```
(*pRags).GetAge();
```

Бу ерда қавслар билвосита мурожаат оператори (*) `GetAge()` функцияси чақирилишидан олдин бажарилишини англатади.

Бундай конструкция ёзиш учун бироз ноқулайроқдир. Бу муаммо стрелкани эслатувчи, `синф аъзосига билвосита мурожаат оператори(->)` орқали ҳал қилинади. Бу операторни ёзиш учун узлуксиз равишда кетма–кет икки белги, тире ва катта ишорасини териш лозим. C++ да бу белгилар битта оператор сифатида каралади. 8.6. - листингда динамик соҳада

жойлашган синф экземпляринининг аъзолари, унинг майдонлари ва методларига мурожаат қилиш намойиш этилган.

8.6. – листинг. Динамик тақсимланувчи соҳадаги объектнинг аъзоларига мурожаат қилишга оид мисол.

```
// 8.6. – листинг
// Динамик тақсимланувчи соҳага объектларни
//  жойлаштириш ва уларни ўчириш
# include < iostream.h>
class SimpleCat
{
public:
SimpleCat() { itsYosh= 2; }
~SimpleCat() { }
int GetAge() const   { return itsAge; }
void SetAge(int age) {itsYosh= age; }
private:
int itsAge;
}
int main()
{
SimpleCat *Frisky = new SimpleCat;
cout<<"Frisky"<<Frisky->GetAge()<<"years old\n";
Frisky->SetAge(5);
cout<<"Frisky"<<Frisky->GetAge()<<"years old\n";
cout << "years old\n";
delete Frisky:
return 0;
}
```

НАТИЖА:

```
Frisky 2 years old
Frisky 5 years old
```

ТАҲЛИЛ

17 – сатрда динамик тақсимланувчи соҳадан `SimpleCat` синфининг экземпляри учун жой ажратилди. Бунда чақирилган конструктор ҳосил қилинган объектнинг `itsYosh` хоссасига икки қийматини ўзлаштирди. Айнан шу қиймат 18 – сатрда чақирилган `GetAge()` функция аъзосининг бажарилиш натижаси бўлди. Биз бу ерда объектнинг аъзо функциясини

чақириш учун *синф аъзосига билвосита мурожаат оператори* (->) *дан* фойдаландик. 21 – сатрда янги ёш қийматини ўрнатиш учун SetAge() методи чақирилди ва объектнинг ItsYosh хоссасига 5 қиймати ўзлаштирилди. Бу қийматни экранга чиқариш эса GetAge() методини қайта чақирилиши билан амалга оширилди.

Синф аъзоларини динамик тақсимланувчи соҳада жойлаштирилиши.

Синф аъзолари динамик тақсимланувчи соҳада жойлаштирилган объектларга кўрсаткичлар ҳам бўлиши мумкин. Бундай ҳолатларда бу объектлар учун хотирадан жой ёки конструктор, ёки синфнинг бошқа бир методлари орқали ажратилади. Қоида бўйича объект банд этган хотира соҳасини бўшатиш шу объектнинг деструкторини чақириш орқали амалга оширилади. (8.7. – листинг)

8.7. – листинг. Кўрсаткичлар синф аъзолари сифатида.

```
//8.7. – листинг.
//Синф аъзоларига кўрсаткичлар
# include <iostream.h>
class SimpleCat
{
public:
SimpleCat( );
~SimpleCat( );
int GetAge( ) const { return *itsAge;}
void SetAge(int age) { * itsYosh= age }
int GetWeight ( int weight){ return * itsWeight;
}
void SetWeight (int weight){ *itsOgirlik= weight
}
private:
int * itsAge;
int * itsWeight;
}
SimpleCat::SimpleCat( )
{
itsYosh= new int(2);
itsOgirlik= new int(5);
```

```
}
SimpleCat::~SimpleCat( )
{
delete itsAge;
delete itsWeight;
}
int main( )
{
SimpleCat* Frisky = new SimpleCat;
cout<<"Frisky"<<Frisky->GetAge()<<"years old\n";
Frisky -> SetAge(5);
cout<<"Frisky"<<Frisky->GetAge()<<"years old\n";
delete Frisky;
return 0;
}
```

НАТИЖА:

```
Frisky 2 years old
Frisky 5 years old
```

ТАҲЛИЛ

2 та ўзгарувчи – аъзоси `int` типига кўрсаткич бўлган синф эълон қилдик. Синф конструкторида (17 – 21 – сатрлар) бу ўзгарувчиларни сақлаш учун жой ажратилди ва уларга бошланғич қиймат берилди. Ўзгарувчи – аъзо учун ажратилган хотира деструктор орқали бўшатилди (22 – 26 – сатрлар). Хотира бўшатилиши билан деструкторлар томонидан кўрсаткичларга нол қиймат ўзлаштирилиш ҳеч қандай маънога эга эмас, чунки кўрсаткич тегишли бўлган объектнинг ўзи ҳам ўчирилаяпти. Бундай ситуация хотира бўшатилгандан кейин кўрсаткичга ҳеч қандай қиймат ўзлаштирмаса бўладиган бир ҳолдир.

Синф аъзоларига мурожаат қилувчи функция (жорий мисолда `main()` функцияси) бажарилишида, сиз унинг қандай мурожаат қилишини билмаслигингиз ҳам мумкин. Сиз фақатгина синфнинг мос методлари (`GetAge()` ва `SetAge()`) ларни чақирдингиз, хотира билан бошқа операциялар синфнинг ички механизми орқали бажарилди. `Frisky` объектини ўчиришда `SimpleCat` синфининг деструктори чақирилди (40– сатр). Деструкторда синф аъзолари томонидан белгиланган

хотира бўшатилди. Агарда синфнинг бирор бир аъзоси фойдаланувчи томонидан аниқланган бошқа бир синфнинг экземпляри бўлса, у ҳолда бу синфнинг ҳам деструкторини чақириш зарурияти туғилади.

Албатта жорий мисолимиз учун ўзгарувчи – аъзоларни динамик хотирага жойлаштириш унчалик шарт эмас. Лекин реал дастурларда маълумотларни бундай кўринишда сақлаш услуби анча самарали усул ҳисобланади. Бунинг учун эса ечилиши керак бўлган масалани аниқ қўйилиши ғоятда муҳимдир. Ихтиёрий дастур лойиҳалаштиришдан бошланиши сизга маълум. Масалан, ўзгарувчи аъзоси иккинчи синфнинг объекти бўлган биринчи синфни ҳосил қилиш керак бўлсин. Бунда иккинчи синфнинг объекти биринчи синф объекти ҳосил қилингунча мавжуд эди ва у ўчирилгандан кейин ҳам йўқолмаслиги керак. Бундай ҳолларда албатта иккинчи объектга мурожаат кўрсаткич орқали амалга оширилиши лозим.

Масалан, биринчи объект ойна иккинчиси эса хужжат бўлсин. Маълумки, ойна орқали албатта хужжатга мурожаат бўлиши лозим. Бошқа томондан, ҳужжатнинг мавжудлик даври ойна томонидан ҳам бошқарилмайди. Шунинг учун ойна бу объектга кўрсаткич орқали мурожаат қилиши лозим.

Бундай ҳолларда кўрсаткич ўрнига `ҳавола` номли ўзгарувчи ҳам ишлатилади. `ҳавола` ҳақида 9 – мавзуда батафсил танишамиз.

this кўрсаткичи.

Синфнинг ҳар бир методи яширин параметрга – `this` кўрсаткичига эгадир. Бу кўрсаткич жорий объектнинг адресини ўзида сақлайди. Олдинги қисмларда қараб чиқилган `GetAge()` ва `SetAge()` функциялари ҳам ўзларида бу параметрни сақлар эдилар.

8.8. листингда `this` кўрсаткичини ошкор кўринишда қўлланилишига мисол келтирилган.

8.8. – листинг. `this` кўрсаткичи.

```
// 8.8. - листинг
// this кўрсаткичи
# include <iostream.h>
class Turtburchak
```

```
{
public:
Turtburchak( );
~Turtburchak( );
void         SetLength(int         Length){this-
>itsLength=length }
int GetLength() const {return this ->itsLength;
}
void SetWidth(int width) {itsWidth= width;}
int GetWidth( ) const { return itsWidth;}
private:
int itsLength;
int itsWidth;
}
Turtburchak::Turtburchak()
{
itsWidth = 5;
itsLength = 10;
}
Turtburchak:: ~Turtburchak()
{  }
int main()
{
Turtburchak theRect;
cout<<"theRect is"<<theRect.GetLength()<<"meters
long.\n";
cout<<"theRect  is"<<theRect.GetWidth  (  )  <<
"meters wide.\n";
TheRect.SetLength(20);
TheRect.SetWidth(10);
cout<<"theRect is"<<theRect.GetLength()<<"meters
long.\n";
cout<<"theRect is"<<TheRect.GetWidth()<< "meters
wide.\n";
return 0;
```

НАТИЖА:

```
The Rect is 10 meters long
The Rect is  5  meters wide
The Rect is 20 meters long
The Rect is 10 meters wide
```

ТАҲЛИЛ

`SetLength()` ва `GetLength()` функцияларида `Turtburchak` синфининг ўзгарувчисига мурожаат қилишда `this` кўрсаткичи ошкор кўринишда қўлланилади. `SetWidth()` ва `GetWidth()` функцияларида эса бундай мурожаат ошкормас кўринишда амалга оширилади. Синтаксисдаги фарқларга қарамасдан иккала операция ҳам бир хилдир.

Аслида `this` кўрсаткичининг роли бу ерда кўрсатилганига нисбатан муҳимроқдир. `this` ўзгарувчиси кўрсаткичдир ва у жорий объектнинг адресини ўзида сақлайди. Бу ҳолда у етарлича кучли инструмент вазифасини бажариши мумкин.

10– мавзуда операторларнинг перегрузкасига оид муаммолар муҳокама қилинишида `this` кўрсаткичининг қўлланилишининг реал мисоллари келтирилади. Бу ўринда эса `this` – ўзи ишлатилаётган объектнинг адресини сақловчи кўрсаткичдир.

`this` кўрсаткичи учун хотира ажратилиши ёки унинг бўшатилиши дастурий йўл билан амалга оширилмайди.

Кўрсаткичларни осилиб қолиши

Кўрсаткичларни осилиб қолиши дастурчиларнинг кенг тарқалган хатоликларидан биридир. Агарда кўрсаткич мурожаат қилган объектни `delete` оператори орқали ўчирсак ва бу кўрсаткичга 0 қиймати ўзлаштирилмаса, бу ҳолда жорий кўрсатгич осилиб қолади. Кейинчалик дастурда бу кўрсаткичдан фойдаланиш турли муаммоларни келтириб чиқариши мумкин. Кўрсаткичнинг осилиб қолишида туғиладиган муаммо қуйидаги муаммога ўхшашдир. Почта хизмати янги офисга кўчди, сиз эса унинг олдинги телефонига қўнғироқ қилишни давом эттирасиз. Агарда бу номер ўчириб қўйилган бўлса яхши, бу катта муаммони туғдирмайди. Лекин, бу номер қандайдир ҳарбий заводга берилган бўлса, у ҳолда юз берадиган ҳолатни тасаввур қилишингиз мумкин.

Бир сўз билан айтганда, кўрсаткичлар устида `delete` операторини ишлатишда эҳтиёт бўлиш лозим. Бу ҳолда, агарда кўрсаткичга 0 қиймат ўзлаштирилмаса у олдинги қийматини, яъни ўчирилган объект адресини ўзида сақлайди. Дастурни

ишлаши давомида айнан шу адресга бошқа объект аниқланиши мумкин, чунки олдинги объект delete оператори орқали ўчирилганидан сўнг, динамик хотирани бу қисмидан бемалол фойдаланиш мумкин бўлиб қолади. Натижада биз жорий кўрсаткичда биз умуман кутмаган объектга мурожаат ҳосил бўлиб қолади ва у дастурни ишлаш жараёнида нотўғри ҳодисаларни келтириб чиқариши мумкин. 8.9.- листингда осилиб қолган кўрсаткичларни ишлатилишига мисол келтирилган.

8.9. – листинг. Кўрсаткични осилиб қолишига мисол.

```
// 8.9. – листинг.
// Кўрсаткични осилиб қолишига мисол
typedef unsigned short int ushort;
# include <iostream.h>

int main()
{
ushort * pInt = new ushort
*pInt = 10;
cout <<”* pInt:”<<* pInt <<endl;
delete pInt;
long * plong = new long;
*pLong = 90000;
cout << “*pLong: ” <<*pLong <<endl;
*pInt=20 //бўш  кўрсаткичга қиймат ўзлаштириш.
cout << “*pInt:”<<* pInt <<endl;
cout << “*pLong:”<<*pLong <<endl;
delete pLong;
return 0;
}
```

НАТИЖА:

```
* pInt :      10
* pLong:  90000
* pInt:       20
* pLong:  65556.
```

(Сизнинг натижаларингиз юқоридагилардан фарқ қилиши мумкин)

ТАҲЛИЛ

7 – сатрда `pInt` ўзгарувчиси `ushort` типига кўрсаткич сифатида эълон қилинади ва бу типдаги маълумотни сақлаш учун хотира ажратилади. 8 – сатрда бу кўрсаткичдаги адресга 10 қиймати ёзилади. 9 – сатрда эса у экранга чиқарилади. Кейин эса `pInt` учун ажратилган хотира `delete` оператори орқали бўшатилади. Бундан сўнг жорий кўрсаткич осилиб қолади.

11 – сатрда `pLong` номли янги кўрсаткич эълон қилинади ва унга `new` оператори орқали ажратилган хотиранинг адреси ўзлаштирилади. 12 – сатрда кўрсаткичнинг бу адресига 90000 сони ёзилади ва 13 – сатрда бу қиймат экранга чиқарилади.

14 – сатрда `pInt` кўрсатиб турган адресга 20 қиймат ёзилади. Лекин, бу кўрсаткич учун ажратилган жой бўшатилган эди. Шунинг учун бундай операция нокорректдир. Бундай ўзлаштириш нотўғри натижаларни келтириб чиқаради. 15 – сатрда `pInt` нинг янги қиймати экранга чиқарилади. Бу қиймат кутилганидек 20 га тенг экан. 16 – сатрда эса `pLong` кўрсаткичининг қиймати чиқарилди. Лекин унинг қиймати биз кутмаган натижага тенг – 65556. Бу ерда иккита савол туғилади:

59: Биз қандай қилиб `pLong` нинг қийматини ҳаттоки, уни ишлатмасдан туриб ўзгартирдик.

60: 17 сатрда ўзлаштирилган қиймат қаерга жойлаштирилди.

Сиз албатта бу иккала савол ўзаро боғлиқ эканлигини пайқаган бўлсангиз керак. 17 – сатрга ўзлаштирилган қиймат `pInt` кўрсаткичи шу вақтгача кўрсатиб турган адресга ёзилди. Лекин бу адрес 11 – сатрда бўшатилган эди ва компилятор бу соҳани бошқа маълумотларни ёзиш учун ишлатган. `pLong` кўрсаткичи эълон қилинганда (13 – сатрда), олдин `pInt` кўрсаткичи мурожаат қилган хотира қисми у томонидан резервланди. (Айрим компютерларда бу ҳол бўлмаслиги ҳам мумкин). `pInt` кўрсатиб турган адресга 20 қийматнинг ёзилиши, шу адресда сақланаётган `pLong` кўрсаткичи кўрсатиб турган қийматни ўзгартиради. Чунки иккала кўрсаткич ҳам битта адресни кўрсатиб турибди. Шунинг учун олинган натижалар кутилганидек эмас.

Бир оз четга чиқиш бўлсада pLong кўрсаткичи адресидаги қиймат нима учун 65556 га тенг бўлиб қолганлигини батафсил кўриб чиқамиз.

61: pInt кўрсаткичи хотирани биз 10 сонини ёзган қисмини кўрсатиб турар эди.

62: delete оператори орқали биз компиляторга хотиранинг бу қисмида бошқа маълумотларни сақлашга имкон ҳосил қилдик, яъни хотирани бу жойини бўшатдик.

63: *pLong ўзгарувчисига 90000 қиймат ўзлаштирилди. Компьютерда long типи тўрт байт жой эгаллайди ва машина даражасида 90000 сони 5F 90 00 01 кўринишда ифодаланади.

64: Кейин pInt кўрсаткичига 20 қиймат ўзлаштирилди, у эса 16 – лик саноқ системасида 00 14 га эквивалент. Иккала кўрсаткич ҳам хотирани бир жойига мурожаат қилишидан 90000 сонининг олдинги икки байти қайта аниқланади. Натижада 00 14 00 01 сонига эга бўламиз.

65: pLong кўрсаткичидаги қийматни экранга чиқаришда байтлар тартиби 00 01 00 14 га ўзгаради ва бунга эквивалент бўлган сон 65556 ҳосил бўлади.

Кўрсаткичларни эълон қилишда const калитли сўзини ишлатилиши.

Кўрсаткичларни эълон қилишда const калитли сўзи тип спецификаторидан ёки олдин, ёки кейин ёзилади. Масалан, уларнинг қуйидаги вариантларда эълон қилиниши тўғридир:

```
const int *pOne;
int * const pTwo;
const int *  const pThree;
```

Бу мисолда pOne кўрсаткичи int типидаги ўзгармасни адресини ўзида сақлайди. Шунинг учун у кўрсатаётган қийматни ўзгартириш мумкин эмас.

pTwo кўрсаткичи int типи кўрсаткич бўлган ўзгармасдир. Бу ҳолда кўрсаткич адресида ёзилган қийматни ўзгартириш мумкин, лекин адреснинг ўзини ўзгартириб бўлмайди.

Ниҳоят, pThree кўрсаткичи int типидаги ўзгармасни адресини ўзида сақловчи ўзгармас бўлган кўрсаткичдир. У

хотиранинг битта ва фақат битта адресини ўзида сақлайди ва бу адресдаги қиймат ҳам ўзгармасдир.

Биринчи навбатда қандай қиймат ўзгармас деб эълон қилинаётганлигини билишимиз керак. Кўрсаткични ўзими ёки унда сақланаётган қийматми? Агарда const калитли сўзи ўзгарувчининг типидан олдин ёзилган бўлса, демак аниқланган ўзгарувчимиз ўзгармас бўлади. Агарда const калитли сўзи ўзгарувчи типидан кейин қўйилган бўлса, у ҳолда кўрсаткичнинг ўзи ўзгармасдир. Қуйидаги мисолларни қараймиз.

```
const int * p1 // int типли ўзгармасга  кўрсаткич.
int *const p2 // ўзгармас     кўрсаткич,
//яъни  доимо   хотирадаги бир соҳани кўрсатиб туради.
```

Функция аъзоларга кўрсаткичларни эълон қилишда const калитли сўзини қўлланилиши.

Олдинги мавзуларда синф функция – аъзоларини аниқлашда const калитли сўзини қўлланилишини кўриб чиққан эдик. Агарда функция ўзгармас деб эълон қилинса, компилятор у орқали жорий объектнинг маълумотларини ўзгартирилишини чеклар эди.

Агарда объектга кўрсаткич ўзгармас деб эълон қилинган бўлса, биз ундан фақатгина const спецификатори билан аниқланган методларни чақириш учунгина фойдалана оламиз. Бу 8.10. - листингда кўрсатиб ўтилган.

8.10. – листинг. Ўзгармас объектларга кўрсаткичлар.

```
//8.10. – листинг
//Кўрсаткичёрдамида    константа    методларини
//чақирилиши
# include < iostream.h>

class Turtburchak
{
public:
Turtburchak();
~ Turtburchak();
void SetLength(int Length){ itsLength = Length;}
int GetLength() const {returu  itsLength; }
void SetWidth(int width) {itsWidth = width; }
int GetWidth() const { return itsWidth}
```

```
private:
int itsLength;
int itsWidth;
};

Turtburchak :: Turtburchak ( )
{
its Width = 5;
its Length = 10;
}

Turtburchak :: ~Turtburchak ( )
{ }
int main()
{
Turtburchak *pRect = new Turtburchak;
const Turtburchak *pConstRect=new Turtburchak;
Turtburchak* const  pConstPtr = new Turtburchak;
cout<<"pRect width:"<<pRect->GetWidth()
<<"meters \n";
cout<<"pConstRect width:"
<<pConstRect->GetWidth()
cout <<" meters\n";
cout << "pConstPtr width:"
<<pConstPtr->GetWidth()
cout << "meters\n";
Prect-> SetWidth (10);
//PconstRect -> SetWidth(10);
PconstPtr -> SetWidth(10);
cout<< "pRect width:"<<pRect -> GetWidth()
cout<<" meters\n";
cout<<"pConstRect width:"
<<"pConstRect->GetWidth()
cout<< "meters\n";
cout<<"pConstPtr width:"<<pConstPtr->GetWidth()
cout<< "meters \n";
return 0;
}
```

НАТИЖА:

```
pRect width:        5 meters.
pConstRect width:   5 meters
pConstPtr width :   5 meters
pRect width:         10 meters
pConstRect  width: 5 meters
pConstPtr  width:   10 meters.
```

ТАҲЛИЛ

5 – 17 – сатрларда `Turtburchak` синфининг тавсифи берилган. 13 – сатрда ёзилган, `const` спецификаторига эга бўлган `GetWidth()` методига эътиборимизни қаратамиз.

27 – сатрда `Turtburchak` синфи объектига кўрсаткич эълон қилинаяпти. 28 – сатрда худди шу синфнинг ўзгармас объектига кўрсаткич эълон қилинаяпти. Ўзгармас кўрсаткич эса 29 – сатрда эълон қилинаяпти.

`pRect` кўрсаткичи томонидан чақирилган `SetWidth()` методи объектнинг кенглигини ўрнатаяпти. 39 – сатрда `pConstRect` кўрсаткичи орқали синф методи чақирилишига мисол келтирилган. Лекин `pConstRect` кўрсаткичи ўзгармас объектга кўрсаткич бўлиб, у орқали `const` спецификатори бўлмаган методларни чақириб бўлмайди. Шунинг учун бу сатр изоҳга айлантирилган. 40 – сатрда эса `SetWidth()` методи `pConstPtr` кўрсаткичи орқали чақирилаяпти. Бу кўрсаткич ўзгармас ва фақат хотиранинг бир соҳасига мурожаат қилиши мумкин. Лекин, у мурожаат қилаётган объект константа эмас, шунинг учун бу операция тўғридир.

const this кўрсаткичи.

Ўзгармас объект эълон қилингандан кейин, унинг `this` кўрсаткичи ҳам ўзгармас кўрсаткич сифатида ишлатилади. `const this` кўрсаткичи орқали фақатгина `const` спецификатори мавжуд бўлган методларни чақириш мумкин.

Бу масала навбатдаги машғулотларда батафсил ўрганилади.

Кўрсаткичлар ёрдамида ҳисоблаш

Бир кўрсаткични бошқа кўрсаткичдан айириш мумкин. Масалан, агарда иккита кўрсаткич массивнинг турли

элементларига мурожаат қилса, бир кўрсаткичнинг иккинчисидан айирмаси, берилган массив элементларининг орасидаги элементлар сонига тенг бўлади. Бу методика белгили массивлар устида иш бажаришда самарали тарзда қўлланилади. Бунга мисол 8.11. листингда келтирилган.

8.11. – листинг. Белгили массивдан сўзларни белгилаш.

```
# include < iostream.h>
# include < ctype.h>
# include < string.h>
bool GetWord ( char * string, char * word, int&
WordOff Set);
// Асосий дастур
int main( )
{
const int bufferSize = 255;
char  buffer[bufferSize+1];//  Барча  сатрларни
//сақлаш учун ўзгарувчи
char word[bufferSize+1];//Барча сўзларни сақлаш
//учун ўзгарувчи
int wordOffSet=0;  //Биринчи белгидан бошлаймиз
cout <<”Enter a string:”
cin.getLine(buffer, bufferSize);
While(GetWord( buffer, word, wordOffSet))
{
cout<< “Got  this word:”<<word<<endl;
}
return 0;
}
//  Белгилар  сатридан  сўзни  белгилаб  олувчи
функция
bool  GetWord(char  *string,  char  *word,  int
&wordOffSet)
{
if(!string[wordOffSet])//Сатр охирини аниқлайди.
return false;
char* p1, p2;
p1= p2 = string+ wordOffSet // навбатдаги сатрга
кўрсаткич
// Сатр бошидаги пробелларни ўчирамиз.
```

```
for(int i=0;i<(int)strlen(p1)&& !isalnum(p1[0]);
i++)
p1++;
// Сўз мавжудлигини текшириш
if(!isalnum(p1[0]))
return false;
// p1 кўрсаткич навбатдаги сўзнинг бошини //
кўрсатади. худди шундай p2 ҳам
p2 = p1;
// P2 ни сўзнинг охирини кўрсатишига қадар
ўзгартирамиз
while (isalnum(p2[0]))
p2++;
// p2 сатр охирини кўрсатади.
// p1 эса сатр бошини кўрсаткичларнинг айирмаси
//эса сўзнинг узунлигини беради.
int len = int (p2- p1);// Сўзни буферга оламиз.
Strncopy(word, p1, len );
// ва янги сатрга ўтиш белгиини қўямиз
word[len]="\0";
//Навбатдаги сўзини бошланишини излаймиз.
for(inti=int(p2-string); i<(int)strlen(string)&&
!isalnum(p2[0]); i++);
p2++;
wordOffSet = int(p2-string );
return true;
}
```

НАТИЖА:

```
Enter a string: this code first appeared in C++
Report
   Got this word : this
   Got this word : code
   Got this word : first
   Got this word : appeared
   Got this word : in
   Got this word : C++
   Got this word : Report
```

ТАҲЛИЛ

Листингнинг 13 – сатрида фойдаланувчи матн киритади. Киритилган сатр `GetWord()` функцияси орқали қайта ишланади. Бу функцияга `WordOffSet` бутун ўзгарувчиси параметр сифатида узатилади. `WordOffSet` ўзгарувчисига 11 – сатрда 0 қиймат ўзлаштирилади.

`GetWord()` функцияси ҳар сафар чақирилганда дастур бошқаруви 23–сатрга берилади. 23–сатрда эса `string[wordOffSet]` қиймати 0 га тенг ёки тенг эмаслиги текширилади. Шартни бажарилиши сатрни чегарасида эканлигимизни англатади. `GetWord()` функцияси `false` қиймат қайтаради.

24–сатрда белгили типга кўрсаткич бўлган иккита ўзгарувчи эълон қилинади. 26 – сатрда бу ўзгарувчиларга `WordOffSet` ўзгарувчиси қийматининг навбатдаги сўзи бошланишининг адреси ўзлаштирилади. `p1` кўрсаткичига ҳарф ёки рақам бўлган 1 – белгининг адреси ўзлаштирилади. Агарда бундай белги учрамаса, функция `false` қиймат қайтаради.

Шундай қилиб `p1` кўрсаткичи навбатдаги сўзнинг бошига мос келади. 46 – сатрда `p2` кўрсаткичга ҳам худди шу қиймат ўзлаштирилади.

45 – ва 49 – сатрларда берилган сатрдан ҳарф ҳам, рақам ҳам бўлмаган биринчи белгини излаш амалга оширилади. `p2` кўрсаткич шу белгига кўчирилади. Энди `p1` ват `p2` кўрсаткичлар мос равишда сўзнинг бошини ва охирини кўрсатиб турадилар. `p2` кўрсаткичнинг қийматидан `p1` кўрсаткични қийматини айириб уни бутун сонли типга ўгирамиз. Бундай операциянинг бажарилиши натижасида шу сўзнинг узунлигини олишимиз мумкин. Кейин эса сўзнинг бошланиш жойи ва узунлигига асосланган ҳолда уни буфер ўзгарувчисига кўчирамиз.

САВОЛЛАР.

66: Кўрсаткич нима?
67: Кўрсаткичлар билан ишлаш орқали қандай имкониятларга эришамиз?
68: Объектларни хотиранинг динамик соҳасига жойлаштириш қандай қулайлик келтиради?
69: Ўзгарувчи адресини олиш учун қандай оператор қўлланилади?
70: Кўрсаткич кўрсатаётган адресда жойлашган қийматни қандай оператор орқали олиш мумкин?
71: Кўрсаткичда сақланаётган адрес билан шу адресда жойлашган қийматларни фарқи нимада?
72: Қуйидаги эълон қилинганларни бир - биридан фарқи нимадан иборат?

```
const int * ptrOne ва int * const ptrTwo
```

ТАЯНЧ ИБОРАЛАР

Кўрсаткич, ўзгарувчи адреси, адрес оператори, стекли хотира, динамик тақсимланувчи хотира, *new* оператори, *delete* оператори, хотирани сирқиб кетиши, *this* кўрсаткичи, осилган кўрсаткичлар

Ҳаволалар (Ссылкалар)

Олдинги мавзуларда бўш хотирадаги объектларни бошқаришда кўрсаткичлардан фойдаланиш ва бу объектларга мурожаатларни қандай ташкил қилиш мумкинлигини билан танишган эдик. Ушбу мавзуда биз қуйидагиларни ўрганамиз:

73: Ҳавола нима?
74: Ҳаволани кўрсаткичдан қандай фарқи бор?
75: Ҳаволалар қандай ҳосил қилинади ва қўлланилади?
76: Ҳаволалардан фойдаланишда қандай чегаралар мавжуд?
77: Бир функциядан бошқа функцияга объектларни ҳавола сифатида узатилиши қандай амалга оширилади?

Ҳавола нима?

Ҳавола – бу ўзгарувчи псевдонимидир. Ҳавола ҳосил қилиниши билан биз унга бошқа объект номи ёрдамида унинг адресатини ўзлаштирамиз. Шу вақтдан бошлаб ҳавола жорий объектнинг алтернатив номи бўлиб хизмат қилади ва бу ҳавола устида бажарилган барча амаллар ушбу объектга тегишли бўлади.

Ҳаволаларни эълон қилишда адресатдаги объект типи, ундан кейин ҳавола оператори `(&)`, ундан кейин эса ҳавола номи ёзилади. Ҳаволаларни номлашда ўзгарувчиларни номлаш учун мумкин бўлган ихтиёрий номдан фойдаланиш мумкин. Лекин кўп дастурчилар ҳавола номлари олдидан «`r`» перефиксини ёзишни лозим деб ҳисоблашади. Агарда `someInt` номли бутун типли ўзгарувчи берилган бўлса, бу ўзгарувчига ҳаволани тузиш учун қуйидаги ёзувни ёзиш лозим.

```
int & rSomeInt = someInt
```

Бу ёзув қуйидагича ўқилади: `rSomeInt` – бутун сонли қийматга ҳавола бўлиб, у `SomeInt` ўзгарувчиси адресини ўзлаштириб олган. Ҳаволаларни тузиш ва уларни ишлатишга оид мисол 9.1. листингда келтирилган.

9.1. – листинг. Ҳаволаларнинг қўлланилиши.

```
// 9.1. – листинг
// Ҳаволаларни қўлланилишига мисол

# include < iostream.h>
int main()
```

```
{
int   intOne;
int &rSomeRef =  intone;
intOne = 5;
cout << “intOne:”<< intOne << endl;
cout<< “rSomeRef :”<<rSomeRef<<endl;
rSomeRef = 7;
cout<< “intOne:” << intOne <<endl;
cout<< “rSomeRef:” <<rSomeRef << endl;
return 0;
}
```

НАТИЖА:

```
int One:        5
rSomeRef:       5
intOne:         7
rSomeRef:       7
```

ТАҲЛИЛ

6–сатрда `intOne` номли бутун типдаги ўзгарувчи эълон қилинди. 7–сатрда эса `rSomeRef` номли бутун қийматли ҳавола эълон қилинаяпти ва унга `intOne` ўзгарувчисининг адреси ўзлаштирилаяпти. Агарда ҳавола эълон қилинса ва унга қиймат ўзлаштирилмаса дастурни компиляция қилиш вақтида хатолик рўй беради. Кўрсаткичдан фарқли равишда ҳавола эълон қилиниши билан унга қиймат ўзлаштирилиши лозим.

8–сатрда `intOne` ўзгарувчисига 5 қиймат ўзлаштирилди. 9–10 сатрларда `intOne` ўзгарувчиси ва `rSomeOne` ҳаволани қийматлари экранга чиқарилаяпти. Албатта, улар бир хил қийматга эга.

11 – сатрда `rSomeOne` ҳаволага 7 қиймат ўзлаштирилди. `RSomeOne` ҳавола `intOne` ўзгарувчисига псевдоним бўлганлиги сабабли 7 сони `intOne` ўзгарувчисига ҳам ўзлаштирилди. 12– ва 13– сатрлар орқали уларнинг қийматлари экранга чиқарилиб ҳавола ўзгарувчининг псевдоними (синоними) эканлиги яна бир бор тасдиқланган

Ҳаволалар билан ишлашда адрес операторининг(&) қўлланилиши.

Агарда ҳаволанинг адресини олмоқчи бўлсак, у ўзининг адресати адресини қайтаради. Ҳавола ўзига хос табиати айнан шу ҳолатдир. Ҳаволалар ўзларининг адресатларининг псевдонимлари ҳисобланадилар. Яъни улар адресатлари билан бир хил адресга эгадирлар. Ҳаволаларнинг бу хоссаси 9.2 – листингда намойиш қилинган.

9.2 – листинг. Ҳаволанинг адресини олиш.

```
// 9. 2. – листинг.
// Ҳаволаларни ишлатишга мисол.
# include < iostream.h>
int main( )
{
int   intOne;
int &rSomeRef = intOne;
int One = 5;
cout << " intOne:" << intOne << endl;
cout <<"& intOne:"<< & intOne<< endl;
cout<< "& rSomeRef:"<< &rSomeRef << endl;
return 0;
}
```

НАТИЖА:

```
intOne :    5
rSomeRef :  5
& intOne :  0 X 3500
& rSomeRef: 0 X 3500
```

ТАҲЛИЛ

Бу ҳолатда ҳам `rSomeRef` ҳаволага `intOne` ўзгарувчи ўзлаштирилди. Дастурда `rSomeRef` ҳаволаси `intOne` ўзгарувчисининг адреси чиқарилди ва улар бир хил эканлиги аниқланди. C++ тилида ҳаволани адресини олиш операцияси қараб чиқилмаган, чунки бу мантиққа мос келмайди. Ҳаволалар тузилиши билан уларга қиймат ўзлаштирилади ва улар ҳақиқатан ҳам ўз адресатларининг синоними бўлиб қоладилар. Адрес оператори буни тасдиқлаб турибди.

Ҳаволаларга қайта қиймат ўзлаштирилмайди.

Ҳаволалар ўзларининг адресатларининг псевдоними бўлганликлари сабабли уларга қайта қиймат ўзлаштириш мумкин эмас. Агарда бундай ҳол рўй берса, ҳавола мурожаат қилиб турган адресатга ҳам янги қиймат ўзлаштирилади. 9.3-листингда бу ҳолат намойиш қилинган.

9.3. – листинг. Ҳаволага қиймат ўзлаштириш

```
// 9.3. – Листинг.
// Ҳаволаларга қиймат ўзлаштириш

# include <iostream.h>

int main()
{
  int  intOne;
  int  & rSomeRef = intone;

intOne = 5;
cout << " intOne:\ t"  << intOne << endl;
cout << " rSomeRef :\ t"<< rSomeRef << endl;
cout << "& intOne:\ t"  << & intOne << endl;
cout << "& rSomeRef:\t"<< & rSomeRef << endl;

 int   intTwo= 8;
 rSomeRef = intTwo;
// Бу ерда сиз кутмаган амал бажарилди.
 cout << "\n intOne: \ t" << intOne << endl ;
 cout << "intTwo: \ t " << intTwo << endl;
 cout<<"rSomeRef :\t" << rSomeRef << endl;
 cout<<"& intOne : \t" << & intOne << endl;
 cout <<"& intTwo : \t" << &intTwo << endl;
cout <<"& rSomeRef: \t" <<& rSomeRef<< endl;
return 0;
}
```

НАТИЖА:

```
intOne :     5
rSomeRef :   5
&intOne :    0 * 213 e
```

```
&rSomeRef :  0 * 213 e

intOne :     8
intTwo :     8
rSomeRef :   8
&intOne :    0 * 213 e
&intTwo:    0 * 2130
&rSomeRef : 0 * 213 e
```

ТАҲЛИЛ

8 – ва 9 – сатрларда бутун сонли ўзгарувчи ва унга ҳавола эълон қилинди. 11–сатрда эса бутун сонли ўзгарувчига 5 қиймат ўзлаштирилди, 12 – 15 – сатрларда эса ўзгарувчи ва унга ҳаволанинг қиймати, ҳамда уларнинг адреслари экранга чиқарилди.

17 – сатрда `intTwo` номли янги ўзгарувчи тузилди ва унга 8 қиймат ўзлаштирилди. 18 – сатрда `rSomeRef` ҳаволага қайта қиймат ўзлаштиришга, яъни уни `intTwo` ўзгарувчисининг псевдонимига айлантиришга ҳаракат қилинди. Лекин биз кутган ҳол юз бермади. `rSomeRef` ҳавола олдингидек `intOne` ўзгарувчисининг псевдоними бўлиб қолаверди, 18 – сатрда бажарилган амал эса қуйидаги жумлага эвивалент бўлиб қолди холос.

```
intOne = intTwo
```

Бу 19 – ва 21 – сатрлар орқали уларнинг қийматини экранга чиқарилиши натижасида яна бир бор ўз тасдиғини топди: уларнинг қиймати `intTwo` ўзгарувчисининг қиймати билан устма – уст тушди. 23 – 25 – сатрлар орқали экранга чиқарилган маълумот эса `rSomeRef` ҳавола `intTwo` ўзгарувчисига эмас, балки ҳали ҳам `intOne` ўзгарувчисига мурожаат қилаётганлигини тасдиқлайди.

Синф объектларига ҳаволаларни аниқланиши.

Ҳаволалар орқали ихтиёрий объектларга, шунингдек фойдаланувчи томонидан аниқланган объектларга ҳам мурожаат қилиш мумкин. Ҳаволалар синфлар учун эмас, балки объектлар

учун тузилади. Ҳақиқатан ҳам ҳавола қуйидагича эълон қилиниши мумкин эмас. `int & rIntRef = int ; // нотўғри`

`rIntRef` номли ҳаволага бутун қийматли ўзгарувчини ўзлаштириш мумкин. Масалан,

```
int howBig = 200 ;
int & rIntRef = howBig ;
```

Худди шунингдек, `CAT` синфини ҳам ҳаволага ўзлаштириш мумкин эмас.

```
Cat & rCatRef = Cat// нотўғри
```

`rCatRef` ҳаволага `CAT` синфининг бирор конкрет объектидан фойдаланиб қиймат ўзлаштириш мумкин.

```
Cat& frisky;
  Cat & rCatRef = frisky;
```

Объектга ҳавола худди объектнинг ўзини ишлатиш каби қўлланилади. Объект майдонлари ва методларига ҳавола одатдаги синф аъзоларига оддий мурожаат оператори (.) орқали амалга оширилар эди. Объект ҳавола учун ҳам бу ўринлидир. Жорий факт 9.4.- листингда намойиш қилинган.

9.4. – листинг. Синф объектига ҳавола.

```
// 9.4. – Листинг
// Синф объектига ҳавола

# include < iostream.h >

class SimpleCat
{
  public :
  SimpleCat(int age, int weight);
  ~ SimpleCat() {  }
  int GetAge() {return itsAge;}
  int GetWeight(){return itsWeight;}
private:
 int itsYosh;
 int itsOgirlik;
};

SimpleCat::SimpleCat(int age, int weight)
{
```

```
itsYosh= age ;
itsOgirlik= weight;
}
int main( )
{
SimpleCat Frisky(5,3);
SimpleCat& rCat= Frisky;
cout << " Frisky:";
cout << "Frisky.GetAge()<<"years old \n";
cout << " and Frisky`s weight: "
cout <<rCatGet.Weight()<<"kilogram\n";
return 0 ;
}
```

НАТИЖА:

```
Frisky: 5 years old
and Frisky`s weight : 3 kilograms
```

ТАҲЛИЛ

26 – сатрда `SimpleCat` синфининг объекти сифатида `Frisky` ўзгарувчиси эълон қилинди. 27– сатрда эса `SimpleCat` синфининг бирор объектига ҳавола қилувчи `rCat` ҳавола эълон қилинди ва унга `Frisky` объекти ўзлаштирилди. 30– ва 32– сатрларда `SimpleCat` синфи аъзоларига мурожаат қилинди. Бунда биринчи мурожаат `SimpleCat` синфининг объекти орқали, иккинчи мурожаат эса `SimpleCat` синфи объектининг ҳаволаси (`rCat`) орқали амалга оширилди. Эътибор берган бўлсангиз иккала ҳолда ҳам натижалар бир хилдир. Бу ердан яна қуйидагича хулоса келиб чиқади:

Ҳавола – бу реал объектнинг псевдонимидир.

Нол кўрсатгичлар ва ҳаволалар.

Агарда кўрсаткичга қиймат ўзлаштирилмаган ёки у бўшатилган бўлса унга нол қиймат (0) бериш лозим. Бу ҳолат ҳаволаларга (ссылкаларга) тегишли эмас. Умуман, ҳаволалар нол бўлмаслиги лозим ва нол объектларга ҳаволаларни сақловчи дастур нокоррект ҳисобланади. Нокоррект дастурнинг ишлаш жараёнида турли хатолик пайдо бўлиши мумкин.

Функцияга аргументларни ҳавола кўринишда узатилиши.

5 – мавзуда функциялардаги иккита камчилик, уларга аргументларни қиймат сифатида узатилишида аргумент қилиб узатилган объект билан узатилаётган ўзгарувчиси орасида алоқани узилиши (уларни алоҳида – алоҳида объект деб эътироф этилиши) ҳамда функцияларни фақатгина битта қиймат қайтариши билан танишган эдик.

Бу иккала чеклашни функцияга аргументларни ҳавола кўринишда узатиш орқали ҳал қилиш мумкин. C++ тилида функцияга аргументларни ҳавола кўринишида узатишда икки усулдан фойдаланилади. Улар ёки кўрсаткичлар ёрдамида, ёки ҳаволалар ёрдамида узатилади.

Кўрсатгичнинг қўлланилиш синтаксиси ҳаволанинг ишлатилиш синтаксисидан фарқ қилишига қарамай улар ёрдамида бир хил иш бажарилади. Функцияга унинг кўриниш соҳаси чегараси учун объектнинг нусхаси ўрнига унинг ўзи узатилади.

Биз 5 – мавзуда функцияга узатиладиган параметрлар хотира стекида жойлашиши билан танишган эдик. Агарда функцияга ҳавола кўринишида қиймат узатилса стекда объектнинг ўзи эмас балки унинг адреси жойлашади.

Кўпгина компютерларда адрес махсус регистрда сақланади. Ихтиёрий ҳолда компиляторни жорий объектга ҳавола қилиш, зарурат туғилганда уни ўзгартириш унинг нусхаси устида эмас, балки тўғридан – тўғри объектнинг ўзида бажарилади. Қуйидаги листингларда функцияга параметрларни қиймат ва ҳавола сифатида узатишнинг ўзига хос хусусиятлари билан танишамиз.

9.5. – листинг. Функцияга аргументни қиймат сифатида узатилиши

```
//9.5.-листинг. Параметрларни қиймат сифатида
узатиш
# include < iostream.h>
void  swap ( int x, int y);
int main( )
{
int  x=5,y=10;
cout<<"Main. Before swap, x:"<<x<<"y:"<<y
```

```
cout<<"\n";
swap(x,y);
cout<<"Main.After swap, x:"<<x<<"y:"<<y
cout<<"\n";
return 0;
}
void  swap ( int x, int y)
{
int       temp;
cout << " Swap. Before Swap, x :"<<x<< "y:"
cout <<y<< " \ n";
temp = x;
x=y;
y=temp;
cout << "Swap. After Swap, x:" << x
cout << "y:"<<y<< "\ n";
}
```

НАТИЖА:

```
Main.Before Swap,  x: 5   y: 10
Swap.Before Swap,  x: 5   y: 10
Swap.After Swap,   x: 10  y: 5
Main.After Swap,   x: 5   y: 10
```

Бу программада `main()` функциясида иккита ўзгарувчига қиймат ўзлаштирилади, кейин эса улар `swap()` функциясига узатилади. `swap()` функцияси бу ўзгарувчиларнинг қийматларини алмаштиради. Лекин бу ўзгарувчилар `main()` функциясида қайта текширилганда улар ўзгармаганлиги аниқланади. Бу ерда муаммо нимадан иборат?

x ва y ўзгарувчилар `swap()` функциясига қиймат бўйича узатилди. Яъни бу ҳолатда бу ўзгарувчиларнинг локал нусхаси функция ичида ҳосил қилинади. `swap()` функцияси ичида амаллар ана шу ўзгарувчиларнинг нусхалари устида бажарилган. Функция ўз ишини тугатиши билан бу нусхалар учун ажратилган жой тозаланади, яъни ўзгарувчилар хотирадан ўчирилади. `swap()` функциясидан ташқарида эса `x` ва `y` ўзгарувчилари олдинги қийматларини сақлаб турган эди. Натижада `x` ва `y` ўзгарувчиларининг қийматларини `swap()` функцияси ишини тугатгандан кейин чиқарилишида `x` ва `y` ўзгарувчилар дастлабки қийматларини қайтардилар.

Функция ўз ишини тугатгандан кейин ҳам у томонидан ўзгартирилган объектлар ўз қийматларини сақлаб қолиш учун жорий объектларни функция параметрига ҳавола кўринишида узатиш лозим. C++ тилида бу масалани ечишни икки усули мавжуд: `swap()` функциясига параметрларни берилган қийматларга кўрсатгич қилиб узатиш ёки берилган қийматларга ҳаволани параметр сифатида узатиш лозим.

`swap()` функциясига параметр сифатида кўрсатгичларни узатиш.

Функцияга аргумент сифатида кўрсаткични узатишда, биз объектнинг адресини функцияга берамиз ва функция бу орқали берилган адресдаги қийматни ўзгартириши мумкин, яъни функция бевосита ўзига параметр қилиб узатиладиган объект устида амаллар бажаради. `swap()` функцияси берилган ўзгарувчиларни реал қийматларини ўзгартириши учун, уни шундай эълон қилиш лозимки, функция иккита бутун қийматга кўрсатгични узатиши мумкин бўлсин. Функция ичида кўрсаткич адресидаги қийматларни олиш операторидан фойдаланиб `x` ва `y` узгарувчиларнинг реал қийматларини алмаштирамиз. Бу ғоя 9.6. – листингда намойиш қилинган.

9.6.- листинг. Кўрсатгич ёрдамида аргументларни мурожаат кўринишида узатиш.

```
//9.6. – листинг.
Аргументни ҳавола кўринишида узатилиши
# include <iostream.h >
void swap(int * x,int *y );
int main()
{
x= 5, y = 10;
cout<<"Main.Before  Swap,  x:"<<x<<"y:"<<y  <<  "
\n" ;
swap ( &x, &y );
cout<<"Main.After Swap,x:"<<x<< "y:" <<y<< "\n";
return 0 ;
}
void swap ( int * px, int * py )
```

```
{
int temp ;
cout<<"Swap.BeforeSwap,*px:"<<*px<<"*py:"<<*py<<
"\n";
temp = *px ;
px = *py ;
*py = temp;
cout<<"Swap.AfterSwap,*px:"<<*px<<"*py:"<<*py<<"
\n;
};
```

НАТИЖА:

```
Main.Before Swap   x  : 5       y  : 10
Swap.Before Swap  *px : 5      *py : 10
Swap.After Swap   *px : 10     *py : 5
Swap.After Swap    px : 10      y  : 5
```

ТАҲЛИЛ

Бу листингда биз кўзлаган мақсадимизга эришдик. 4–сатрда `swap()` функцияси прототипи қуйидагича ўзгартириш киритилди. Параметр сифатида `int` типидаги ўзгарувчилар ўрнига `int` типидаги қийматга кўрсаткичлар аниқланди. 9 – сатрда `swap()` функцияси чақирилганда унга параметр сифатида `x` ва `y` ўзгарувчиларнинг адреси берилди.

15–сатрда `swap()` функцияси учун локал бўлган `temp` ўзгарувчиси эълон қилинди. Уни кўрсаткич деб эълон қилиш шарт эмас, чунки бу ўзгарувчида фақатгина функцияни ҳаёти давомида `*px` (яъни, `x` ўзгарувчиси қиймати) сақланади. Функция ишини тугатгандан кейин `temp` ўзгарувчиси керак бўлмайди.

17 – сатрда `temp` ўзгарувчисига `px` адресида сақланувчи қиймат ўзлаштирилади. 18 – сатрда `px` адресда сақланувчи қийматга, `py` адресли ячейкадаги қиймат ўзлаштирилади. 19 – сатрда эса `py` адресли ячейкага `temp` ўзгарувчиси қиймати ўзлаштирилади.

Натижада адреслари `swap()` функциясига берилган ўзгарувчилар қийматлари мувофаққиятли алмаштирилди.

swap() функциясига параметр сифатида ҳаволаларни узатиш.

Юқорида келтирилган дастур албатта тўғри ишлайди, лекин ундаги swap() функциясининг синтаксиси бироз мураккаб бўлиб тушуниш учун қийиндир. Биринчидан, swap() функцияси ичида бир неча марта кўрсаткич адресидаги қийматни олиш оператори билан ишлашда хатолик рўй бериши мумкин, бундан ташқари кўрсаткич адресидаги қийматни ўқиш операторида ишлатиш ноқулайдир. Иккинчидан, чақирилаётган функцияга ўзгарувчи адресини бериш зарурияти swap() функцияси бажарилишининг инкапсуляция принципига мувофиқ келмайди.

C++ тилида дастурлашнинг асосий мақсади, функция бажарилишида унинг деталларини фойдаланувчидан яширин ҳолда фойдаланишдир. Кўрсатгич ёрдамида параметрларни узатишда ўзгарувчиларни адресини олиш ўрнига чақириладиган функциянинг танасида ўзгарувчиларнинг адресини олиш ҳам мумкин. Бу масалани бошқачароқ ечими 9.7. – листингда келтирилган.

9.7. – листинг. swap() функциясида ҳаволадан фойдаланиш.

```
// 9.7. – листинг. Аргументларни ҳавола орқали
//мурожаат сифатида узатишга мисол.
# include <iostream.h>
void swap( int &x, int &y);
int main( )
{
int x = 5, y = 10;
cout << “Main. Before swap, x:” << x << “y :”
<<y<< “\n”;
swap(x,y );
cout<< “Main. After swap,    x:” << x << “y :”
<<y<< “\n”;
return 0;
}
void swap( int &rx, int ry)
{
```

```
int temp;
cout<< "Swap.Before swap, rx:"<<rx<< "ry:" <<
ry<< "\n";
temp = rx;
rx = ry;
ry = temp;
cout << "Swap.After swap, rx:" <<rx<<"ry:" << ry
<< "\n";
}
```

НАТИЖА:

```
Main. Before swap, x: 5, y: 10
Swap. Before swap, rx: 5, ry: 10
Swap. After swap, rx: 10, ry: 5
Main. After swap, x: 10, y:5
```

ТАҲЛИЛ

Бу листингда худди кўрсатгичлар билан берилган мисолдагидек 6 – сатрда иккита ўзгарувчи эълон қилинаяпти. Уларнинг қиймати эса 7 – сатрда экранга чиқарилаяпти. 8 – сатрда `swap()` функцияси чақирилаяпти ва унга `x` ва `y` ўзгарувчиларнинг адреслари эмас, балки ўзлари параметр сифатида узатилаяпти.

`swap()` функцияси чақирилгандан кейин дастурнинг бошқаруви 18 – сатрга ўтади. Параметр сифатида узатилган ўзгарувчиларнинг қийматлари 15–сатрда экранга чиқади. Бунда эътибор берган бўлсангиз, уларнинг қийматини экранга чиқариш учун ҳеч қандай махсус оператор талаб этилмайди, чунки биз бу ерда берилган қийматларнинг псевдонимидан фойдаландик.

16– 18 – сатрларда қиймат алмашинуви бажарилди ва улар 19 – сатрда экранга чиқарилди. Дастур бошқаруви `swap()` функцияси чақирилган жойга ўтди ва 9–сатрда, яъни `main()` функциясининг ичида бу қийматлар яна экранга чиқарилди.

Демак, биз функция ичида ҳаволаларни ишлатиш орқали янги имкониятларга эга бўлдик, яъни чақирилаётган функция орқали берилган маълумотни ўзгартирдик. Бунда чақирилаётган функция одатдаги функциялардан ҳеч қандай фарқ қилмайди.

Функциянинг бир неча қиймат қайтариши

Олдинги мавзулардан маълумки, функция фақат битта қиймат қайтаради. Агарда функциядан иккита қиймат олиш керак бўлса нима қилиш керак? Бундай муаммонинг ечимларидан бири функцияга иккита объектни ҳавола кўринишида узатишдир. Функциянинг бажарилиши давомида бу объектларга керакли қийматлар ўзлаштирилади. Ўзига ҳавола кўринишида узатилган объектларни функция томонидан ўзгартирилиши факти бу функциянинг иккита қиймат қайтариши билан тенг кучлидир.

Бундай натижага нафақат ҳаволалар орқали балки, кўрсаткичларни билан ҳам эришиш мумкин. 9.8. – листингда кўрсатилган функция учта қиймат қайтаради: иккитаси кўрсатгич параметрлар орқали, бири эса қайтариладиган қиймат сифатида.

9.8. – листинг. Кўрсатгич ёрдамида қийматларни қайтарилиши

```
// 9.8. – листинг.
// Кўрсатгич ёрдамида функциянинг бир нечта
қиймат қайтариши
#include <iostream.h>

short Factor( int n, int * pSquared, int *
pCubed);
int main( )
{
int number, squared, cubed;
short error;
cout << “Enter a number ( 0 – 20 ):”;
cin >> number;
error = Factor( number, &squared, &cubed);
if ( !error )
{
cout << “number: ” << number<< “\n”;
cout << “square: ” << squared<< “\n”;
cout << “cubed: ” << cubed << “\n”;
}
else
cout<< “Error encountered!!\n”;
return 0;
```

```
}
short Factor(int n, int *pSquared, int *pCubed)
{
short Value = 0;
if (n>20)
Value = 1;
else
{
pSquared = n* n;
pCubed = n * n * n;
Value = 0;
}
return Value;
}
```

НАТИЖА:

```
Enter a number  ( 0 - 20 ) :  3
number : 3
square :  9
cubed :  27
```

ТАҲЛИЛ

7 – сатрда `int` типидаги `number`, `squared` ва `cubed` ўзгарувчилари эълон қилинди. `number` ўзгарувчисига фойдаланувчи томонидан киритилган қиймат ўзлаштирилади. Бу қиймат, ҳамда `squared` ва `cubed` ўзгарувчиларининг адреслари `factor()` функциясига параметр сифатида берилади.

`factor()` функциясида қиймат сифатида узатилган биринчи параметр таҳлил қилинади. Агарда у 20 дан катта бўлса `value` ўзгарувчисига 1 қиймат берилади ва бу хатолик рўй берганлигини англатади.

29– ва 30– сатрларда кўрсаткичлар орқали `main()` функциясидаги ўзгарувчиларга қиймат ўзлаштирилаяпти. 31– сатрда `value` ўзгарувчисига функция ишини мувофаққиятли тугатганлигини билдирувчи қиймат ўзлаштирилаяпти.

Ҳавола ёрдамида қийматни қайтарилиши

9.8. – листингда келтирилган мисол тўғри ишлашига қарамай, уни ўқиш ва ишлатиш учун албатта кўрсатгичлар

ўрнига ҳаволалардан фойдаланиш лозим. 9.9. – листингда худди шу дастур келтирилган.

9.9. – листинг. Ҳаволалар ёрдамида қийматларни қайтарилиши

```
//9.9. – листинг.
//ҳавола ёрдамида функция бир неча қийматни
қайтариши
# include <iostream.h>
typedef unsigned short USHORT;
enum ERR_CODE{ SUCCESS, ERROR };
ERR_CODE Factor (USHORT, USHORT&, USHORT&);
int main( )
{
USHORT number, squared,cubed;
ERR_CODE result;
cout << “Enter a number ( 0 – 20 ): ”;
cin >> number;
result = Factor(number, squared, cubed);
if (result == SUCCESS)
{
cout << “number:” << number<< “\n”;
cout << “square:”<<squared<< “\n”;
cout << “cubed:” << cubed<< “\n”;
}
else
cout << “Error encountered!!\n”;
return 0;
}
ERR_CODE Factor(USHORT n,USHORT & Squared,USHORT
& rCubed)
{
if ( n>20)
return ERROR;
else
{
rSquared = n * n;
rCubed  =  n * n * n;
return SUCCESS;
}
}
```

НАТИЖА:

```
Enter a number  ( 0 - 20 ) :  3
number : 3
square :  9
cubed :  27
```

ТАҲЛИЛ

9.9. – листинг 9.8. – листингдан иккита хусусияти билан фарқ қилади. Биринчиси, `ERR_CODE` саноқли тўплами хатолик ҳақидаги хабарни нисбатан аниқроқ кўрсатади.

Иккинчиси, `Factor()` функциясидаги кўрсаткичлар ўрнига ҳаволалар ишлатилишидир. Ҳаволаларни ишлатилиши кўрсаткичларнинг ишлатилишига нисбатан тушунарлироқдир.

Дастур самарадорлигини ошириш мақсадида ҳаволаларнинг ишлатилиши

Функцияга ҳар сафар объектни қиймат сифатида узатишда бу объектнинг нусхаси тузилади. Функция объектни қиймат сифатида қайтаришда ҳам яна бирта нусха тузилади.

5 – мавзудан маълумки, бу объект нусхалари стекка кўчирилади ва бу жараён учун вақт ҳамда хотира сарф қилинади. Таянч бутун сонли қийматларга ўхшаган кичик объектлар учун бундай сарфларнинг қиймати билинмайди.

Лекин фойдаланувчи томонидан тузиладиган катта объектлар учун сарф қилинадиган ресурслар сезиларли даражада ўсади. Стекка ёзиладиган бундай объектнинг ўлчами унинг барча ўзгарувчи – аъзоларининг йиғиндисини ташкил этади. Бунда ҳар бир ўзгарувчи – аъзо яна бир мураккаб объект бўлиши ҳам мумкин. Шунинг учун бундай катта структураларни стекка кўчиришда катта хотира ва вақт талаб қилади.

Бундан ташқари бошқа сарфлар ҳам мавжуд. Синф объектининг вақтинчалик нусхасини тузиш учун компилятор махсус конструктор – кўчирувчини чақиради. Конструктор – кўчирувчининг ишлаши, қандай қилиб хусусий конструктор - кўчирувчини яратиш каби масалалари билан кейинги машғулотларда танишамиз. Ҳозир эса конструктор – кўчирувчи объектнинг вақтинчалик нусхаси стекка кўчирилишида

чақирилишини билишимиз етарлидир. Объектнинг вақтинчалик нусхасини стекдан ўчириш учун синф деструктори чақирилади. Агарда функция объектни қиймат сифатида қайтарса олдин бу объектнинг вақтинчалик нусхаси тузилади, кейин эса у ўчирилади. Катта объектлар билан ишлаганда бундай конструктор ва деструкторларни чақирилиши дастурнинг ишлаш тезлиги ва компьютер хотирасидан фойдаланишига жуда катта таъсир қилади. Бу ғояни намойиш қилиш учун 9.10. – листингда `SimpleCat` фойдаланувчи объекти тузилган. Мисолда

кўчирувчи–конструктор ва деструкторларни қанчалик тез – тез чақирилишини кўрсатилган.

Демак, 9.10 – листингда `SimpleCat` объекти тузилади, ундан сўнг иккита функция чақирилади. Биринчи функция `Cat` синфи объектини қиймат сифатида қабул қилади ва қиймат сифатида қайтаради. Иккинчиси функция эса объектни ўзини эмас, балки унга кўрсатгични қиймат сифатида қабул қилади ва узатади.

9.10. – листинг. Кўрсатгичлар ёрдамида объектларни ҳавола кўринишда узатиш.

```
//9.10. – листинг.
// Объектга кўрсатгичларни узатилиши

#include<iostream.h>

class SimpleCat
{
public:
 SimpleCat(); //конструктор
 SimpleCat(Simple Cat&);//кўчирувчи- конструктор
 ~SimpleCat(); //деструктор
}

SimpleCat: : SimpleCat()
{
cout << " Simple CatConstructor … \ n" ;
}

SimpleCat::SimpleCat(SimpleCat& )
{
```

```
cout<<"Simple CatCopy Constructor…\ n" ;
}

SimpleCat::~SimpleCat( )
{
cout << "SimpleCat Destructor …\ n" ;
}

SimpleCat FunctionOne (SimpleCat theCat) ;
SimpleCat* FunctionTwo (SimpleCat *theCat) ;

int main()
{
cout <<"Making a Cat…\ n" ;
SimpleCat Frisky ;
cout<< "Calling FunctionOne…\ n" ;
FunctionOne(Frisky) ;
cout<<"Calling FunctionTwo …\ n" ;
FunctionTwo(&Frisky)
return 0;
}

//FinctionOne функциясига қиймат сифатида
узатилаяпти
SimpleCatFunctionOne ( SimpleCat theCat)
{
cout << "FunctionOne Returning …\ n" ;
return theCat;
} ;

//FinctionTwo функциясига ҳавола сифатида
узатилаяпти
SimpleCat* FunctionTwo ( SimpleCat* theCat)
{
cout << " Function Two. Returning …\ n" ;
return tneCat;
}
```

НАТИЖА:

```
Making a Cat…
Simple CatConstructor …
Calling FunctionOne …
```

```
Simple CatCopy Constructor …
FunctionOne returning …
Simple CatCopy Constructor …
Simple CatDestructor …
Simple CatDestructor …
Calling FunctionTwo …
FunctionTwo Returning …
Simple CatDestructor …
```

ТАҲЛИЛ

6 – 12 – сатрларда жуда ҳам соддалаштирилган `SimpleCat` синфи эълон қилинаяпти. Синфнинг барча компонентлари – конструктор, конструктор – кўчирувчи ва деструкторлар таналарида экранга ахборот чиқарувчи фрагмент аниқланган бўлиб, бу орқали улар қачон чақирилганини билиб олиш мумкин бўлади.

34–сатрда `main()` функцияси ўзининг биринчи ахборотини экранга чиқаради. 35–сатрда эса `SimpleCat` синфининг экземпляри тузилади. Бу эса ўз навбатида `SimpleCat` синфининг конструкторини чақиради. Конструктор чақирилганда бу ҳақида экранга ахборот чиқарилади.

36 – сатрда `main()` функцияси `FunctionOne` функцияси чақирилганлиги ҳақида хабар чиқаради. `FunctionOne()` функциясига `SimpleCat` синфи объекти қиймат сифатида узатилади ва бунинг натижасида чақириладиган функция учун локал бўлган `SimpleCat` объекти нусхаси стекда жойлаштирилади. Бу ҳолда кўчирувчи конструктор чақирилади.

Дастурнинг бошқаруви чақирилган функцияни танаси ҳисобланган 46 – сатрга ўтади. Кейин эса бу функция чақирган функцияга бошқарувни беради ва `SimpleCat` синфи объекти яна қиймат сифатида қайтарилади. Бу ҳолда конструктор–кўчирувчини чақириш орқали яна битта объект нусхаси тузилади ва экранга навбатдаги хабар чиқарилади.

`FunctionOne()` функцияси қайтарган қиймат бирор бир объектга ўзлаштирилмайди ва функциянинг қиймат қайтариш механизмининг реализациясида ҳосил қилинган вақтинчалик объект умуман фойдаланилмасдан ўчирилади. Чунки `FunctionOne()` функцияси ишини тугатиши билан унинг локал объектлари хотирадан ўчирилади.

Дастурнинг бошқаруви яна `main()` функциясига қайтгандан сўнг `FunctionTwo()` функцияси чақирилади. Унга параметр ҳавола кўринишида узатилади. Шунинг учун унинг ишлаши давомида объектнинг ҳеч қандай нусхаси ҳосил бўлмайди ва ўз навбатида конструктор – кўчирувчи ҳам, деструктор ҳам чақирилмайди.

Ушбу дастурнинг ишини таҳлил қилсак, `FunctionOne()` функцияси чақирилганда кўчирувчи – конструкторга икки марта, деструкторга икки марта ҳавола қилинади, агарда объект ҳавола кўринишида узатилса бундай мурожаатлар бўлмас экан.

Ўзгармас кўрсатгичларни параметр сифатида узатиш.

`FunctionTwo()` функцияси самарадор бўлиши билан бирга айрим камчиликларга ҳам эга. `FunctionTwo()` функцияси чақирилганда унга узатиладаган `SimpleCat` синфининг объектини ўзгартиришга ҳуқуқи йўқ эди. Бундай усулда узатиш объектни ўзгартириш имконини очиб беради ва объектни қиймат сифатида узатилишида таъминланган ҳимоясини бекор қилади.

Объектларни қиймат сифатида узатиш музейга ҳақиқий асар фотографиясини топширишга ўхшаган амалдир. Агарда бирор бир бекорчи фотосуратни бузиб қўйса унинг оригиналига ҳеч қандай зарар етмайди. Ҳавола бўйича объектларни узатиш эса музейга ўзингизни уй адресингизни қолдириш ва ўзингизни иштирокингизда қимматбаҳо жиҳозга кўришга ўхшайди.

Бу муаммони функцияга `SimpleCat` синфининг объектига ўзгармас кўрсатгични узатиш орқали ҳал қилиш мумкин. Бу ғоя 9.11. - листингда намойиш қилинган.

9.11. – листинг. Ўзгармас кўрсатгичларни параметр сифатида узатиш.

```
// 9.11. – листинг.
// Объектни ўзгармас кўрсаткичларини узатиш

# include < iostream.h >

class SimpleCat
{
 public :
```

```
 SimpleCat();
 SimpleCat(Simple Cat&);
 ~SimpleCat();

 int GetAge() const { return itsYosh};
 void SetAge(int  age) { itsYosh= age ; }

 private :
 int itsYosh;
 };

 SimpleCat:: SimpleCat()
 {
 cout << "Simple CatConstructor …\ n" ;
 itsYosh= 1;
 }

 SimpleCat:: SimpleCat( SimpleCat& )
 {
 cout<<"SimpleCat Copy Constructor … \n" ;
 }

 SimpleCat::~SimpleCat()
{
cout<<"SimpleCat Destructor …\n" ;
}

SimpleCat  *FuctionTwo(const  SimpleCat  *  const
theCat);

int main()
{
cout<< "Making a Cat… \ n";
SimpleCat Frisky;
cout<<"Frisky is " ;
cout<< Frisky.GetAge ( );
cout<< "years old \n";
int age = 5;
Frisky SetAge(age);
cout <<"Frisky is";
cout <<Frisky.GetAge( );
```

```
cout << “years old \n”;
cout << “Calling Function Two …\ n”;
FunctionTwo( &Frisky ):
cout << “Frisky is”;
cout << Frisky.GetAge();
cout <<” years old \ n”
return 0;
}
// FunctionTwoга ўзгармас кўрсатгич узатилаяпти.
const SimpleCat* const FunctionTwo (const
SimpleCat* const theCat)
{
cout << “Function Two. Returning . . .\n”;
cout << “Frisky is now” <<  theCat – >GetAge( );
cout << “years old\n”;
// theCat->SetAge(8); Const!
return theCat
}
```

НАТИЖА:

```
Making a Cat…
Simple CatConstructor…
Frisky is 1 years old
Frisky is 3 years old
Calling FunctionTwo…
FunctonTwo. Returning…
Frisky is now 5 years old
Frisky is 5 years old
Simple CatDestructor.
```

ТАҲЛИЛ

`SimpleCat` синфига иккита функция ва битта ўзгарувчи – аъзо (`itsAge`) қўшилди. Функциялардан бири ўзгармас (`GetAge`), иккинчиси эса ўзгармас функция эмас (`SetAge`).

Дастурда конструктор, конструктор–кўчирувчи ва деструктор худди олдинги листингдагидек аниқланган. Лекин натижалардан бирор марта конструктор – кўчирувчи чақирилмаганлигини кўриб турибмиз. Бунинг сабаби объект ҳавола кўринишида узатиляпти ва бунинг натижасида унинг

нусхаси тузилмаяпти. 41–сатрда бошланғич берилган қийматдаги ёш билан тузилди. Бу қиймат 42 – сатрда экранга чиқарилди.

46–сатрда `SetAge()` методи ёрдамида `itsYosh` ўзгарувчисининг қиймати ўзгартирилди, унинг натижаси эса 47 – сатрда экранга чиқарилди. Бу дастурда `FunctionOne()` функцияси ишлатилмаяпти, лекин `FunctionTwo()` функциясининг озгина ўзгартирилгани чақирилаяпти. Унинг эълон қилиниши 36 – сатрда келтирилган. Бу сафар унинг параметри ҳам, қайтарадиган қиймати ҳам ўзгармас объектга ўзгармас кўрсатгич сифатида аниқланган.

Параметр ҳам, қайтариладиган қиймат ҳам ҳавола сифатида аниқланганлиги учун ҳеч қандай нусха тузилмайди ва кўчирувчи – конструктор ҳам чақирилмайди. Фақатгина `FunctionTwo()` функциясининг кўрсатгичлари энди ўзгармасдир, ва ўз навбатида у орқали ўзгармас бўлмаган методни (`SetAge()`ни) ишлатиб бўлмайди. Агарда 64 – сатрдаги `SetAge()` методига ҳавола изоҳга олинмаса эди, дастурнинг компиляция жараёни амалга ошмас эди.

`main()` функцияси ичида тузилган объект ўзгармас эмас эди ва `Frisky` объекти `SetAge()` методини чақира олар эди. Бу оддий объектнинг адреси `FunctionTwo()` функциясига берилди. `FunctionTwo()` функцияси эълон қилинишида унга узатиладиган параметр ўзгармас объектга ўзгармас кўрсаткич эканлиги айтилган эди. Шунинг учун бу объектнинг функцияси ўзгармас бўлсагина унга ҳавола қилиш мумкин.

Ўзгармас объектларни узатишда ҳаволаларни қўлланилиши.

9.11.-листингда кўрсатилган дастурда объектнинг вақтинчалик нусхасини тузиш заруриятини четлаб утиш синф конструкторлари ва деструкторига ҳаволалар сонини қисқартиришга ва натижада дастурни самарали ишлашини таъминлашига мисол келтирилган. Бу мисолда ўзгармасли объектга ўзгармасли кўрсатгич қўлланилган ва бу орқали фунция томонидан объектни ўзгартирилмаслиги таъминланган. Фақатгина функцияга кўрсатгич узатиш синтаксиси бир оз мураккаброқдир.

Маълумки, объект ҳеч қачон нол бўлмайди. Агарда кўрсатгич ҳаволага алмаштирилса функциянинг ички мазмуни соддароқ бўлар эди. Бу 9.12.листингда ўз тасдиғини топган.

9.12. – листинг. Объектга ҳавола узатиш.

```
// 9.11. – листинг.
// Объектни ҳаволаларини узатиш

# include < iostream.h >

class SimpleCat
{
public :
 SimpleCat( );
 SimpleCat( SimpleCat& );
 ~SimpleCat( );

int GetAge() const { return itsYosh};
void SetAge(int  age) { itsYosh= age ; }

private :
 int itsYosh;
};

SimpleCat:: SimpleCat( )
{
cout << " Simple Cat Constructor …\ n" ;
itsYosh= 1;
}

SimpleCat:: SimpleCat( SimpleCat& )
{
cout << "SimpleCat Copy Constructor … \ n" ;
}

SimpleCat:: ~SimpleCat()
{
cout << " SimpleCat Destructor …\ n" ;
}
```

```
const   SimpleCat& FuctionTwo (const SimpleCat&
const  theCat) ;

int main()
{
cout << " Making a Cat… \ n" ;
SimpleCat Frisky;
cout << "Frisky is" ;
cout << Frisky.GetAge();
cout << " years old \n";
int age = 5;
Frisky SetAge(age);
cout <<"Frisky is";
cout <<Frisky.GetAge ( );
cout << "years old \ n";
cout << "Calling Function Two …\ n";
FunctionTwo ( Frisky ):
cout << "Frisky is";
cout << Frisky.GetAge();
cout << "years old\n";
return 0;
}

// FunctionTwo га ўзгармас ҳавола узатилаяпти.
const SimpleCat& FunctionTwo (const SimpleCat&
theCat)
{
cout << "Function Two. Returning . . .\n";
cout << "Frisky is now" <<  theCat.GetAge( );
cout << "years old\n";
// the Cat.SetAge(8); Const!
return theCat;
}
```

НАТИЖА:

```
Making a Cat…
SimpleCat Constructor…
Frisky is 1 years old
Frisky is 5 years old
Calling FunctionTwo
FunctionTwo. Returninig…
```

```
Frisky is now 5 years old
Frisky is 5 years old
SimpleCat Destructor.
```

ТАҲЛИЛ

Бу дастурни натижаси 9.11. – листингда кўрсатилган натижа билан бир хилдир. Ягона мавжуд ўзгариш – `FunctionTwo()` функцияси энди ўзгармас объектга ҳаволани қабул қилади ва қиймат сифатида қайтаради.

Қачон ҳаволаларни, қачон эса кўрсатгичларни ишлатиш лозим.

Малакали мутахассисликлар ҳаволаларини қўллашни кўрсатгичларни қўллашдан афзал билишади. Ҳаволаларни ишлатиш осон ва улар масалани ечишда ахборотларни қисқартиришга олиб келади.

Лекин ҳаволаларга қиймат бериш мумкин эмас. Агарда сизга олдин бир, кейин эса бошқа объектни кўрсатиш лозим бўлса у ҳолда албатта кўрсатгичлардан фойдаланишга тўғри келади.

Ҳаволалар нол қийматни қабул қилмайди. Шунинг учун қаралаётган объектнинг нол бўлиб қолиш эҳтимоли мавжуд бўлса ҳам ҳаволаларни ишлатиш мумкин эмас. Бу ҳолда ҳам кўрсатгичлар ишлатилиши керак.

Мисол сифатида `new` операторини қараймиз. Агарда `new` оператори янги объект учун хотирадан жой ажратмаса у нол кўрсатгични қиймат сифатида қайтаради. Ҳавола нол бўлмаслигидан, хотиранинг нол эмаслигини текширмасдан, унга бу хотирага ҳаволани ўзлаштиришингиз мумкин эмас.

Қуйидаги мисолда бу ҳолат қандай текширилиши кўрсатилган.

```
int * pInt = new int
if (pInt! = null)
int & rInt =* pInt;
```

Бу мисолда `new` оператори қайтарадиган хотира соҳасини `int` типдаги қийматли `pInt` кўрсатгичи ўзлаштирилади. Бу хотира соҳасининг адреси текширилади ва агарда у `null`

қийматига тенг бўлмаса `pInt` кўрсатгич адресидаги қиймат `rInt` ҳаволага ўзлаштирилади.

САВОЛЛАР:

1. Нима учун ҳаволалар қўлланилади?
2. Ҳаволалар билан ишлаш осон бўлса кўрсатгичлар нима учун керак?
3. Агарда янги объект тузиш учун хотира етишмаса *new* оператори қандай қиймат қайтаради?
4. Ўзгармас ҳавола нима ва у нима мақсадда ишлатилади?
5. Функцияга объектни ҳавола сифатида узатиш билан қиймат сифатида узатишни нима фарқи бор?

ТАЯНЧ ИБОРАЛАР

Ҳавола, объектни ҳавола сифатида узатиш, ўзгармас ҳавола, конструктор – кўчирувчи, объект нусхаси

АДАБИЁТЛАР

1. Жесс Либерти, “Освой самостоятельно C++ за 21 день”, Санкт Петербург 2000, 815 с.

МУНДАРИЖА