Лекция 3

МОДУЛЬДІК ПРОГРАММАЛАУ

Программаның көлемі ұлғайған сайын оның барлық ұсақ-түйек мәселелерін есте сақтау мүмкіндігі азаяды. Кез келген есептің күрделілігін төмендетудің табиғи тәсілі оны шағын бөліктектерге бөлу болып табылады. C++ тілінде кез келген программаны ***функциялар*** арқылы қарапайым және түсінікті бөліктерге бөлуге болады, осыдан кейін программаны функциялардың өзара әрекеттесуі деңгейінде ірілендірілген түрде қарастыру мүмкіндігі туады. Адам фактілердің шектеулі көлемін ғана есте сақтауға қабілетті болғандықтан, осы мәселенің өзі өте маңызды болып саналады. Функцияларды пайдалану программаның абстрактылық дәрежесін арттырудың алғашқы қадамы болғандықтан, ол программа құрылымын қарапайым етуге мүмкіндік береді.

Программаның абстрактылық дәрежесін арттыратын келесі қадам – функцияларды және олармен байланысқан мәліметтерді жеке файлдарға (модульдерге) топтастырып жазу, олардың әрқайсысы компиляциядан жеке-жеке өткізіледі. Компиляция нәтижесінде пайда болған объектілік модульдер жинақтауыш көмегімен атқарылатын программаға біріктіріледі. Программаны модульдерге бөлу қайта компиляциялау уақытын азайтады және программаны түзету процесін жеңілдетеді, себебі ол маңызды емес іс-әрекеттерді модуль интерфейсінен тыс орналастырады да, программаны бөлшектей отырып түзетуге (немесе әртүрлі программалаушылардың көмегімен түзетуіне) мүмкіндік береді.

Модуль мәліметтер мен оларды өңдейтін функциялардан тұрады. Басқа модульдердің осы мәліметтерді өңдейтін өзіндік құралдарының болмағаны дұрыс, олар бұл үшін алғашқы модуль функцияларын қолдануы тиіс. Модульді қолдану үшін оның жүзеге асырылуының барлық ерекшеліктерін білудің қажеті жоқ, тек оның интерфейсін білу жеткілікті. Модульдер бір-бірінен неғұрлым тәуелсіз түрде жұмыс істейтін болса, программаны түзету соғұрлым жеңіл болады. Мұндай тәсіл программаны жөндеу кезінде бір мезгілде есте сақталуы қажет ақпараттың жалпы көлемін азайтады. Программаны барынша тәуелсіз бөліктерге бөлу күрделі мәселе болып табылады, ол программаны жобалау кезеңінде жүзеге асырылуы тиіс.

Жүзеге асыру ерекшеліктерін жасыру ***инкапсуляция*** деп аталады. Инкапсуляция тек құрылымдық емес, объектіге бағытталған программалаудың да негізгі идеясы болып табылады. Инкапсуляция мысалы – программалық код бөлігін функцияға жазып орналастыру және оған қажетті барлық мәліметтерді параметрлер ретінде беру. Мұндай функцияны пайдалану үшін оның тақырыбымен анықталатын (аты, қайтаратын мәнінің типі және параметрлерінің типі) интерфейсін білу жеткілікті. Барлық функциялардың тақырыптары және сырттан қол жеткізуге болатын типтердің, айнымалылардың және тұрақтылардың сипаттамалары ***интерфейстің модулі*** болып табылады. Ауқымды (глобалды) программалық объектілердің сипаттамалары программаның барлық модульдеріне сәйкес келуі тиіс.

ФУНКЦИЯЛАР

Функция – бұл белгілі бір аяқталған әрекетті орындайтын сипаттамалар мен операторлардың атау берілген тізбегі. Функция параметрлерді қабылдап, мән қайтара алады.

C++ тіліндегі кез келген программа функциялардан тұрады, олардың бірі міндетті түрде ***main*** деп аталуы тиіс (программаның орындалуы осы функциядан басталады). Функцияны шақырған кезде ол орындала бастайды. Кез келген функция жариялануы және

анықталуы тиіс. Функция да басқа шамалар сияқты бірнеше рет жариялануы мүмкін, бірақ ол бір-ақ рет анықталуы тиіс. Компилятор функцияны шақырудың дұрыстығын тексеруі үшін программа мәтінінде функцияны жариялау оны шақырудан ерте орналасуы тиіс.

Функцияны жариялау (прототип, тақырып, сигнатура) оның атын, қайтарылатын мәнінің типін және оған берілетін параметрлер тізімін тағайындайды. Функцияны анықтауда, оны жариялаумен қатар, функция ***тұлғасы*** қамтылады, ол жүйелі жақшалар ішіндегі операторлар мен сипаттамалар тізбегінен тұрады:

[класс] типі атауы([параметрлер_тізімі]) [throw

(аластамалар)] { функция тұлғасы }

Фукнкцияны анықтаудың құрама бөліктерін қарастырайық.

✓ Міндетті түрде қажет етілмейтін ***класс*** модификаторының көмегімен, ***extern*** және ***static*** түйінді сөздерін қолдана отырып, функцияның көріну аймағын айқын түрде көрсетуге болады:

■ extern – функцияның программаның барлық модульдерінде ауқымды,

яғни глобалды (келісім бойынша) түрде көрінуі;

■ static – функцияның тек өзі анықталған модуль шеңберінде көрінуі.

✓ Функцияның қайтаратын мәнінің ***типі*** жиым мен функциядан (бірақ жиымға немесе функцияға нұсқауыш бола алады) басқа кез келген тип түрінде болуы мүмкін. Егер функция оны шақырған программаға ешқандай мән қайтармайтын болса, онда ***void*** типі көрсетіледі.

✓ ***Параметрлер тізімі*** функцияға оны шақыру кезінде берілуі тиіс болатын шамаларды анықтайды. Параметрлер тізімінің элементтері бірбірінен үтір арқылы ажыратылады. Функцияға берілетін әрбір параметрдің аты мен типі көрсетіледі (жариялау кезінде оның атын көрсетпеуге де болады).

✓ Функцияның тақырыбында ол тікелей немесе жанама түрде туындатуы мүмкін ***аластамалар*** тізімін беруге болады. Тақырып бөлігі функцияның интерфейсі болғандықтан, мұнда аластамалар тізімін беру арқылы қолданушылар функцияны пайдалану туралы ақпарат алады және де көзделмеген аластама пайда болса, бұл жағдай анықталатыны туралы кепілдік беріледі.

Функцияны ***inline*** модификаторының көмегімен ***құрамдас функция*** ретінде анықтауға болады, бұл модификатор компиляторға функцияны пайдалану орнына оның кодын әрбір шақыру нүктесіне тікелей орналастыру керек екенін көрсетеді. ***inline*** модификаторы функция типінің алдына қойылады. Ол қысқа функцияларды шақыруға кететін қосымша шығындарды азайту үшін (регистрлерді сақтау және қалпына келтіру, басқаруды беру) қолданылады.

inline директивасы ұсынылатын әрекет ретінде беріледі, сондықтан компилятор оны мүмкіндігіне қарай орындайды. ***inline*** функцияларды пайдалану атқарылатын программаның көлемін ұлғайтып жіберуі мүмкін. Функцияны анықтау оны шақырулардан бұрын орындалуы тиіс, әйтпесе компилятор ***inline*** кеңейтілудің орнына қарапайым функцияны шақыру әрекетін атқарады.

Қайтарылатын мәннің типі және параметрлердің типтері біріге отырып, ***функцияның типін*** анықтайды.

Қарапайым жағдайда ***функцияны шақыру үшін*** оның атын, содан кейін жай жақша ішінде үтірлер арқылы бөлініп берілетін аргументтер аттарын көрсету керек. Функцияны шақыру программаның кез келген бөлігінде орындалуы мүмкін, ол тек синтаксис бойынша функцияның қайтаратын типі тұра алатын орында тұрса болғаны. Егер функцияның қайтаратын мәнінің типі ***void*** болмаса, онда ол кез келген өрнектің құрамына енуі мүмкін немесе жалпы меншіктеу операторының оң жақ бөлігінде орналаса алады.

Екі бүтін шаманың қосындысын қайтаратын функция мысалы:

```
#include <iostream.h>
int sum(int a, int b);            // функцияны жариялау
int main(){
   int a = 2, b = 3, c, d;
   c = sum(a, b);                 // функцияны шақыру
   cin >> d;
   cout << sum(c, d);             // функцияны шақыру
   return 0;
}
int sum(int a, int b){            // функцияны анықтау
   return (a + b);
}
```

Экранға функцияға берілген құрылым өрістерін шығару мысалы:

```
#include <iostream.h>
struct Worker{
   char fio[30];
   int date, code;
   double salary;
};
void print_worker(Worker);        // функцияны жариялау
int main(){
   Worker stuff[100];
   ... /*stuff жиымын қалыптастыру */
   for (int i = 0; i<100; i++)
   print_worker(stuff[i]);        // функцияның шақыру
   return 0;
}
void print_worker(Worker w){      // функцияны анықтау
   cout << w.fio << ' ' << w.date << ' ' << w.code
        << ' ' << w.salary << endl;
}
```

Функцияның ішінде сипатталған барлық шамалар және оның параметрлері ***жергілікті*** (локалды) болып табылады. Олардың әрекет ету аймағы – функция іші (тұлғасы). Функцияны шақыру кезінде, кез келген блокка енген кездегі сияқты, стекте жергілікті автоматты айнымалылар үшін жады бөлінеді. Сонымен қатар, стекте функцияны шақырудың алдыңғы сәтіндегі процессор регистрлерінің мәндері және функциядан кері қайту адресі сақталады, ол осы функциядан шыққаннан кейін оны шақырған программаның (басқа функцияның) орындалуын ары қарай жалғастыру үшін қажет болып табылады.

Функциядан шыққан кезде ол орналасқан стек аймағы босатылады, сондықтан бір функцияның бірнеше шақырылулары арасындағы жергілікті (локалды) айнымалылардың

мәндері сақталмайды. Егер мұндай жағдайдан құтылу қажет болса, жергілікті айнымалыларды жариялау кезінде ***static*** модификаторы қолданылады:

```
#include <iostream.h>
void f(int a){
   int m = 0;
   cout << "n m p\n";
   while (a--){
      static int n = 0;
      int p = 0;
      cout << n++ <<''<< m++<<''<< p++ <<'\n';
   }
}
int main(){ f(3); f(2); return 0;
}
```

Ауқымды айнымалылар

Ауқымды (глобалды) айнымалылар өздерімен аттас жергілікті айнымалылар сипатталмаған барлық функцияларда көрінеді, сондықтан оларды функциялар арасында мәліметтер алмасу үшін қолданған өте жеңіл. Дегенмен, бұл тәсілді пайдаланбаған жөн, себебі мұндайда программаны түзету қиындайды және функцияларды ортақ пайдаланылатын кітапханалар ішіне орналастыруға кедергі келтіреді. Функциялар барынша тәуелсіз болып, олардың интерфейсі толығымен функция прототипі арқылы анықталуына ұмтылу керек.

Қайтарылатын мән

Функциядан оны шақырған функцияға қайту механизмі келесі оператор арқылы жүзеге асырылады**: return [өрнек];**

Функцияның құрамында бірнеше ***return*** операторлары болуы мүмкін (бұл алгоритм қажеттіліктерімен анықталады). Егер функция ***void*** ретінде сипатталса, өрнек көрсетілмейді. Егер функциядан қайту жүйелі жақшаны жабу алдында орындалса, ***void*** типті функция үшін return операторын жазу міндетті емес. ***main*** функциясы үшін де ***return*** операторын жазбауға болады. Бұл кітапта орынды үнемдеу мақсатында main функциясында ***return*** операторы көрсетілмеген, сондықтан мысалдарды компиляциялау кезінде экранға ескерту шығарылады. ***return*** операторынан кейін көрсетілген өрнек функцияның қайтаратын мәні типіне түрлендіріліп, функцияның шақырылу нүктесіне беріледі.