Лабораториялық жұмыс 1. Қолданушы функцияларын құру.

Лабораториялық жұмыстың мақсаты: Студенттерде қолданушы функцияларын құру дағдысын қалыптастыру.

Лабораториялық жұмысты орындау нәтижесінде студенттер келесі қабілеттерге ие болады:

- Қолданушы функцияларына параметрлер беру;
- Функциялардан мән қайтару.

Тапсырма: Төменде көрсетілген мысалдармен танысып, Абрамянның программалау бойынша тапсырмалар жинағының «Функциялар және процедуралар» бөлімінде берілген тапсырмаларды нұсқаңызға сәйкес орындаңыз.

Күрделі әрі көлемді программалар жазу кезінде, программаны *функцияларға* – шағын бөліктерге бөлу арқылы жұмысты жеңілдетуге және бастапқы код көрнекілігін арттыруға болады. Мысалы, ағымдағы бухгалтерлік шоттармен жұмыс істейтін программа жазу қажет болсын делік. Мұнда кітап операцияларын орындайтын басты бір функция жазып, шоттарды төлеу ісін атқаратын тағы бір функция, сонан соң қабылдап алынатын шоттарды реттейтін функция және баланстық есеп беру жұмысын орындайтын функциялар қарастырылса, программаны оқу, түсіну жеңіл болар еді. Егер осы әрекеттерді орындайтын барлық операторлар **Main** ішінде орналасқан болса, онда программа өте көлемді болып кетеді және оны түсіну де қиын болар еді. Оның үстіне, программа көлемі мен күрделілігі артқан сайын, онда қате жіберу мүмкіндігі де ұлғаяды.

Берілген бір тапсырманы атқаратын атаулы (өзіндік аты бар) операторлар жиыны *қолданушы функциясы* болып табылады. Мысалы, **hello_world** деп аталатын келесі функция **Console.WriteLine()** операторы арқылы экранға мәлімет жазып шығарады:

```
static void hello_world(void)
{
  Console.WriteLine("Hello world!!!");
}
```

Көбінесе функциялар программаға өз есептеулерінің нәтижесін қайтарады. Егер функция ешқандай да мән қайтармайтын болса, онда ол функция атының алдында **void** түйінді сөзі көрсетілуі қажет. Функция атынан кейін жақша ішінде оның параметрлері көрсетіледі. *Параметр – бұл функцияға өңдеу үшін берілетін мәлімет*. Егер функцияның параметрлері болмаса, онда дөңгелек жақшалар ішіне **void** түйінді сөзі жазылады немесе ештеңе көрсетілмейді. Функцияны пайдалануды функцияны шақыру деп атайды. Келесі программада **hello_world** функциясы пайдаланылады.

```
using System;
namespace Example
{
  class Program
  {  static void hello_world()
     {
       Console.WriteLine("Hello_world()!!!");
     }
     static void Main(string[] args)
     {
```

```
            Program.hello_world();
            Console.ReadKey();
        }
    }
}
```

Программаны орындау нәтижесі:

```
Hello world!!!
```

Сонымен, программаны іске қосқан кезде бірінші болып **Main** функциясы орындалады. Жоғарыдағы программада **hello_world** функциясын шақыру **Main** функциясында атқарылады. Егер программаны орындау кезінде функцияны шақыру әрекеті кезіксе, онда басқару бірден функцияға беріледі де, оның операторлары ең алғашқы жолдан бастап орындала бастайды. Функцияның ең соңғы операторы орындалып болған соң, басқару негізгі программадағы функцияны шақыру жолынан кейін тұрған операторға беріледі.

Функцияларды пайдалану программалаушыларға модульдік программалауды пайдалану мүмкіндігін береді, яғни программаны модульдерден құрастыруға болады. Әрбір функция белгілі бір тапсырманы (есепті) орындауы тиіс және функция аты сол тапсырманың мазмұнына сәйкес келуі керек. Сонда ғана бұл функцияны басқа программаларды жазу кезінде де қолдануға болады.

Функцияның мән қайтаруы

Үнемі функция нақты бір жұмысты атқаруды (мысалға, қандай да бір хабарламаға қорытынды), және де шақыру функциясының есептеу нәтижесін қайтаруды талап етеді. Шақыру кодының нәтижесін қайтаруды қамтамасыз ету үшін функцияда **return** операторын пайдалану керек.

Функцияның типі оның (**int, float, char,** т.б.) мағынасының қайтарылу типімен анықталады. Мысалы, егер функция **int** типін қайтаратын болса, онда сол типті функция идентификаторы алдында көрсету қажет:

```
int some_function(int value)
{
   // функция операторлары
}
```

Келесі **icube** функциясы параметр ретінде берілген санның кубын қайтарады. Мысалы, функцияға 5 саны берілсе, онда **icube** функциясы 5·5·5 мәнін, яғни 125 мәнін қайтарады:

```
int icube(int value)
{
  return(value * value * value);
}
```

Сонымен, функция шақырылған соң, оның жұмыс нәтижесін қайтару үшін **return** операторы қолданылады (төмендегіні қараңыз). Программада функцияның қайтарған мәні айнымалыға меншіктеледі немесе басқа бір функцияның параметрі ретінде келесідей түрде көрсетілуі мүмкін:

```
result = icube(5);
```

немесе

```
Console.WriteLine("5 cubed is {0}", icube(5));
```

Енді толық программа қарастырайық.

```
using System;
namespace Example
{
  class Program
  {  static int icube (int value)
     {
       return (value * value * value);
     }
     static void Main(string[] args)
     {
        Console.WriteLine("3 cubed is " + Program.icube(3));
        Console.WriteLine("5 cubed is " + Program.icube(5));
        Console.WriteLine("7 cubed is " + Program.icube(7));
        Console.ReadKey();
     }
   }
}
```

Программаны орындау нәтижелері:

```
3 cubed is 27
5 cubed is 125
7 cubed is 343
```

оның параметрі типіне сәйкес болу керек. Егер, мысалы, нақты (жылжымалы нүктелі сан) санның үшінші дәрежесін есептеу керек болса, онда типі басқа **fcube** функциясын құру керек:

```
float fcube(float value)
{
  return(value * value * value);
}
```

C# тілі **return** сөзі кездескенде, бірден функция жұмысын аяқтап, нәтижесін негізгі программаға (оны шақырушы функцияға) қайтарады. Функцияны орындау кезінде **return** операторынан кейін тұрған кез келген операторлар еленбей қалады. Программа өз жұмысын функцияны шақырған жолдан ары қарай жалғастыра береді.

Нақты және формальды (көрсетпелі) параметрлер

Формальды параметрлер деп функция анықталуында көрсетілетін параметрлер атауларын айтамыз. Мысалы, мынадай функцияда:

```
void job_information (int age, float salary, int number_job)
{
   // функциялар операторлары
}
```

age, salary, number_job атаулары берілген функция үшін формальды немесе көрсетпелі параметрлер болып саналады.

Функцияны программадан немесе басқа бір функциядан шақырған кезде, функцияға берілетін мәндер нақты немесе қойылмалы параметрлер болып табылады.

Төмендегі мысалда берілген 30, 42000.00 және 321 мәндері **job_information** функциясын шақыру кезіндегі көрсетпелі параметрлерді алмастырып солардың орнына қойылатын нақты параметрлер болып табылады:

```
job_information (30,42000.00,321);
```

Функцияға тасымалданып қойылатын параметрлер сандық мән немесе айнымалы түрінде болуы мүмкін. Оларға қойылатын жалғыз талап – қойылмалы нақты параметрдің (немесе айнымалының) типі көрсетпелі параметр типімен бірдей болуы тиіс. Келесі код фрагменті қойылмалы параметр орнына айнымалыны пайдалану тәсілін көрсетеді:

```
int workers_age = 30;
float workers_salary = 42000.00;
int job_number = 321; ,
job_information {workers_age, workers_salary, job_number);
```

Функциялар мен айнымалылар атауларының әрекет ету аймағы

Программадағы функциялар мен айнымалылар атауларының өз әрекет ету (немесе көріну) аймағы, яғни оларды пайдалануға болатын, мәндері белгілі болып келетін программаның жұмыс аймағы болады.

Келесі программаны қарастырайық.

```
using System;
namespace Example
{
  class Program
  {
     static void pr(int pr_count)
     {
        int count;
        for (count = 1; count<=pr_count; count++)
         Console.Write('A');
     }
     static void Main()
     {
        int count;
        for (count = 1; count <= 3; count++)
        {
          Console.WriteLine("Your computer should print {0}
time(s)", count);
          pr(count);
        }
        Console.WriteLine();
      }
   }
}
```

Программаның орындалу нәтижелері:

```
Your computershould print 1 time(s)
A
Your computer should print 2 time(s)
AA
Your computer should print 3 time(s)
AAA
```

Мұнда **pr** және **Main** функцияларының әрқайсысында **count** айнымалысы пайдаланылады, бірақ бұл айнымалылардың екеуі екі түрлі болып табылады, өйткені олардың әрқайсысының өз жұмыс аймағы бар. **Pr** функциясындағы **count** айнымалысы тек функцияның орындалуы кезінде ғана белгілі (көріну аймағы функция ішінде ғана) болып саналады. Сол сияқты екінші **count** айнымалысы тек **Main** функциясының орындалуы кезінде ғана жұмыс істей алады. Сондықтан **for** цикліндегі **count** айнымалысының өзгеру нәтижесі **Main** функциясындағы **count** айнымалысына ешқандай әсерін тигізбейді.

Айнымалының жұмыс істеу аймағын қарастыру кезінде локальдық және глобальдық айнымалылар түсінігі жиі пайдаланылады.

Локальдық (жергілікті) айнымалы – бұл белгілі бір функцияның ішкі операторларында ғана белгілі болып табылатын, әрекет ету аймағы шектеулі айнымалы.

Глобальды (ауқымды) айнымалы – бұл программаның барлық аймақтарында толық белгілі болып саналатын айнымалы.

Жоғарыла мысалда көрсетілген **count** айнымалысы тек қарастырылған функция ішінде ғана анықталған локальдық айнымалы болатын.