Лабораторно упражнение № 2

Оператори и управляващи конструкции в език C#

I. Теоретична част

1. Оператори в език C#

Както всеки друг програмен език, C# разполага с определен набор оператори, които позволяват върху данните да бъдат извършени различни действия. Операторите в езика са специални символи чрез които върху данните, наречени *операнди*, се изпълняват съответните операции. Множеството от оператори в език C# до голяма степен се припокрива с това в език C++. Съответно, операторите могат да бъдат разделени в следните категории:

- Аритметични оператори
- Оператори за сравнение
- Логически оператори
- Побитови оператори
- Оператори за присвояване
- Оператори за работа с типове

Според броя на операндите (данните за операцията), операторите се разделят на:

- Унарни – такива, които изискват един операнд;
- Бинарни – такива, които изискват два операнда.

Изключение прави операторът за условен израз (? :), който изисква три операнда. Този оператор е наречен още тернарен (ternary).

1.1. Аритметични оператори

Бинарни аритметични оператори в C# са

- Събиране (+)
- Изваждане (-)
- Умножение (*)
- Деление (/)

Деление по модул или остатък от целочислено деление (%)

Език C# поддържа следните унарни аритметични оператори:

- Запазване на знак (+)
- Промяна на знак (-)
- Инкрементиране (++)
- Декрементиране (--)

1.2. Оператори за сравнения

Операторите за сравнение в C# не се различават от тези в C/C++. Те извършват сравнение на две стойности като резултатът от изпълнението им е стойност от тип bool – true или false. Опрераторите за сравнение са:

- по-голямо (>)
- по-малко (<)
- по-голямо или равно (>=)
- по-малко или равно (<=)
- равно (==)
- различно (!=)

1.3. Логически оператори

Логическите оператори изискват като операнди изрази от булев тип и връщат като резултат стойност от същия тип. Логически оператори в C# са:

- логическо И (&&)
- логическо ИЛИ (||)
- логическо изключващо ИЛИ (^)
- логическо НЕ (!)

1.4. Побитови оператори

Побитовите оператори в C# са следните:

- преместване наляво (<<)
- преместване надясно (>>)
- побитово И (&)
- побитово ИЛИ (|)
- побитово изключващо ИЛИ (^)
- инвертиране на битове (~)

1.5. Оператори за присвояване

Операторът за присвояване (=) в език C# има същият синтаксис и действие както в програмен език C/C++. Стойността на втория операнд се присвоява на променливата отляво на равенството. Каскадното присвояване също е реализирано в езика като присвояванията са от дясно наляво. Например:

```
int a, b;
a = b = 0;
```

В език C# са допустими и съкратените форми на оператор присвояване:

+=, -=, *=, /=, %=,<<=, >>=, &=, |=, ^=

Тяхното действие е същото както в програмен език C/C++. Например, програмен ред a = a + b; е еквивалентен на a += b;

В общ вид, синтаксисът на съкратената форма на оператор присвояване е следната

операнд1 оператор= операнд2;

което заменя:

операнд1 = операнд1 оператор операнд2;

1.6. Оператори ?: и ??

Оператор за условен израз (?:) се нарича още тернарен оператор, тъй като изисква три операнда:

операнд1 ? операнд2 : операнд 3

Първият операнд е от логически тип и в зависимост от неговата стойност резултатът от изпълнението на условния оператор е операнд2, ако стойността на операнд1 е true; или операнд3, ако стойността на операнд1 е false. Оператор ?? изисква два операнда, прави проверка дали първият операнд има стойност null. Ако стойността не е null, резултатът от изпълнението е първия операнд, в противен случай резултатът е вторият операнд.

2. Управляващи конструкции

Управляващите конструкции в C# не се различават съществено от тези в език C/C++. Към тях спадат условните конструкции if, if-else, switch и итерационните конструкции while, do-while и for. В C# съществува още една итерационна конструкция – foreach. За повечето управляващи конструкции е необходимо да се дефинира управляващ израз, който в език C# задължително е от булев тип, за разлика от C/C++, където е позволено управляващият израз в if, if-else, while, do-while и for да е от аритметичен тип.

2.1. Конструкции if и if-else

Конструкция if има следния общ вид:

```
if (условен израз)
  {
        тяло на if-конструкцията
    }
```

Действието на конструкцията е следното: пресмята се условието; ако резултатът е ***true***, изпълнява се тялото на if-конструкцията. При невярно твърдение т.е. резултат false, се продължава със следващата конструкция след ***if***. Задължително условният израз е от булев тип.

Конструкция if-else има следния общ вид:

```
if (условен израз)
  {
     Тяло на if-блок
    }
else
  {
```

Тяло на else-блок

}

Действието на конструкцията е следното: пресмята се условието; ако стойността на израза е ***true***, изпълнява се *if-блок*. Ако стойността на израза е ***false***, изпълнява се *else-блок*.

Ако операторът в тялото на if или else блоковете е само един, тогава той може да не се поставя в блок { …}.

3.2. Конструкция switch

Чрез конструкция ***switch*** се осъществява избор от няколко възможни варианта. Общият вид на конструкцията е следния:

```
switch (израз)
{ case   константенИзраз1: оператор 11;оператор 12;... break;
  case   константенИзраз2: оператор 21;оператор 22;...; break;
  ...
  case   константенИзраз n: операtр n1;оператор n2;...; break;
  default : оператор 1; оператор 2;...;
}
```

Действието на оператор switch е напълно аналогично на това в C/C++:
пресмята се стойността на управляващия израз и се сравнява с константните. Когато стойността на управляващият съвпадне със стойността на някой от константните, изпълнява се съответният блок оператори. Ако стойността на управляващият израз не съвпада с нито един от константните, изпълнява се блокът оператори след ***default***, ако има такъв, или управлението се предава на следващият оператор след блока ***switch.***

3.3. Итерационни конструкции while, do-while, for и foreach

В език C# итерационните конструкции while, do-while и for имат същите синтаксис и действие както в език C/C++. Единствената разлика е, че условният израз в скобите при while и do-while и във втората секция на for задължително е от булев тип.

Управляваща конструкция ***while*** реализира цикъл с пред условие. Конструкцията има следния общ вид:

```
while (условен израз)
{
    // група оператори, съставляващи тялото на цикъла
}
```

Действието на конструкцията е следното: проверява се условието; ако стойността на израза ***true***, изпълнява се тялото на цикъла и отново се проверява условието за край, т.е. *тялото на цикъла се изпълнява, докато условието е вярно*. Когато стойността на условието стане ***false***, прекратява се изпълнението на цикъла.

Конструкцията ***do-while*** реализира цикъл със след условие (постусловие) и има следния общ вид:

```
do
{
    // група оператори, съставляващи тялото на цикъла
}
while (условие);
```

Действието на ***do-while*** е следното: *тялото на цикъла се изпълнява до тогава, докато условието е вярно* т.е. докато изразът представящ условието има стойност ***true***. Когато стойността на израза стане ***false***, прекратява се изпълнението на цикъла.

Действието на ***while*** и ***do-while*** е идентично. Основната разлика между двете конструкции произтича от това, че първата реализира цикъл с предусловие, а втората - цикъл със следусловие. Съответно, при ***while*** проверката за изход от цикъла е преди тялото, а при ***do-while*** – след тялото на цикъла. Следователно възможно е, тялото на конструкция ***while*** да не бъде изпълнено нито веднъж, докато тялото на оператор ***do-while*** ще бъде изпълнено поне веднъж.

Конструкция ***for*** се използва предимно за реализация на цикли с известен брой повторения. По своето действие тя реализира цикъл с предусловие. Синтактично общият вид на конструкцията е следния:

```
for (секция 1;секция 2;секция 3)
{
 // група оператори, съставляващи тялото на цикъла
}
```

Секция 1 и секция 3 се състоят от един или няколко оператора, отделени със запетая. Секция 2 е условен израз.

Действието на конструкция ***for*** е следното:

- Изпълняват се действията в секция 1. Това е т.нар. инициализираща секция, която се изпълнява еднократно и с която се задават началните условия на цикъла.
- Изпълнението на ***for*** продължава в следната последователност: изчисляване на израза в секция 2; изпълнение на оператора след скобите; изпълнение на секция 3. Тази последователност се изпълнява дотогава, докато стойността на израза в секция 2 има стойност различна от 0 или ***true***.

Конструкция for много често се използва когато се обработват масиви.

В програмен език C# съществува още една итерационна конструкция – foreach. Тази конструкция е предвидена за работа с масиви, когато е необходимо последователно да се обработят всички елементи на масива, от първия до последния.

Общият вид на конструкцията е следният:

```
foreach (променлива in колекция/масив)
```

{

// група оператори, съставляващи тялото на цикъла

}

При всяко изпълнение на тялото на конструкция foreach променливата последователно приема стойността на текущият елемент на колекцията или на масива.

Пример:

```
int[] mA = { 2, 4, 6, 8, 10};
            foreach (int k in mA)
            {
                Console.Write(k + " ");
            }
```

II. Задачи за изпълнение:

1. Да се създаде конзолно приложение на C#, с което се демонстрира действието на аритметичните, логическите и побитовите оператори в езика.

2. Да се създаде конзолно приложение, чрез което се определя вида на триъгълник (равностранен, равнобедрен, разностранен) по зададени три страни.

3. Да се създаде конзолно приложение на C#, с което се намират корените на квадратно уравнение.

4. Да се създаде конзолно приложение на C#, с което да се изведат стойностите от 1 до n (цяло положително число) в прав и обратен ред.

5. Да се създаде конзолно приложение на C#, с което се намира n! (n факториел).

6. Да се създаде конзолно приложение, което намира сумата на произволно въведени положителни числа от клавиатурата. Въвеждането на стойност 0 да прекрати по-нататъшното въвеждане на числа.

7. Да се създаде конзолно приложение, с което се отпечатват следните фигури: квадрат от **n** звездички; правоъгълник от **m X n** звездички; триъгълник от **n** звездички (стойностите на **m** и **n** се въвеждат от клавиатурата). Пример:

```
****            **********          *
****            **********          **
****            **********          ***
****            **********          ****
```

8. Да се създаде конзолно приложение, с което се отпечатват фигура, имитираща коледна елха:

```
   0
  101
 21012
3210123
```

Да се модифицира програмния код като цифрата се замени със символ *.