Лабораторная работа

3. Условный оператор и циклы

Варианты заданий

Выполнить задания 1-3 по предложенным вариантам:

	№ Задания		
	1	2	3
Вариант 1	1.I	2.I	3.I
Вариант 2	1.II	2.II	3.II
Вариант 3	1.III	2.III	3.III
Вариант 4	1.IV	2.IV	3.IV
Вариант 5	1.V	2.V	3.V
Вариант 6	1.III	2.II	3.I
Вариант 7	1.I	2.III	3.IV
Вариант 8	1.IV	2.I	3.V
Вариант 9	1.II	2.V	3.III
Вариант 10	1.V	2.IV	3.II

Задание 1.
Пользователь вводит значения *x, a, b, c* (*a, b, c* - действительные числа). Вычислить значения функции F:

I. F={
1/(a*x)-b, при (x+5)<0 и c=0;
(x-a)/c, при (x+5)>0 и c!=0;
10*x/(c-4) - в остальных случаях.
}

II. F={
$a*x^2+b*x+c$, при a<0 и c!=0;
-a/(x-c), при a>0 и b=0;
a*(x+c) - в остальных случаях.

}

III. F={

-a*x-c, при c<0 и x!=0;

(x-a)/-c, при c>0 и b=0;

b*x/(c-a) - в остальных случаях.

}

IV. F={

a-x/(10+b), при x<0 и b!=0;

(x-a)/(x-c), при x>0 и b=0;

3*x+2/c - в остальных случаях

}

V. F={

-3*a/x-c, при a<0 и x!=0;

(x-b)/-c, при c<0 и a=0;

c*(b-a)+x - в остальных случаях

}

Задание 2.

I. Пользователь вводит натуральное число n. Вычислить сумму первых 2n членов ряда $\sum_{k=1}^{2n} \frac{(-1)^{(k+1)}}{k}$, с учетом, что при суммировании исключается каждый 3-й член.

II. Пользователь вводит натуральное число n. Вычислить сумму первых n членов ряда $\sum_{k=0}^{n} \frac{(-1)^k}{2k+1}$, с учетом, что при суммировании исключается каждый 4-й член.

III. Пользователь вводит натуральное число n. Вычислить сумму первых n членов ряда $\sum_{k=1}^{n} \frac{1}{k^2}$, с учетом, что при суммировании исключается каждый 5-й член.

IV. Пользователь вводит натуральное число n. Вычислить сумму первых n членов ряда $\sum_{k=1}^{n} \frac{1}{k(k+1)}$, с учетом, что при суммировании исключается каждый 4-й член.

V. Пользователь вводит натуральное число n. Вычислить произведение первых n членов ряда $\prod_{k=1}^{n} \frac{k+1}{k}$, с учетом, что при умножении исключается каждый 3-й член.

Задание 3.

Вычислить результаты выражений, не используя формулы сумм арифметической и геометрической прогрессии.

I. По данному натуральному *n* вычислите сумму: 1+1/2!+1/3!+…+1/n!;

II. По данному действительному числу *a* и натуральному *n* вычислите сумму $1+a+a^2+...+a^n$.

III. По данному натуральному n вычислите сумму 1+(1+2)+(1+2+3)+...+(1+2+...+n).

IV. По данному натуральному n вычислите сумму 1+1/(1+2)+1/(1+2+3)+...1/(1+…+n).

V. По данному натуральному n вычислите сумму 1+2!+3!+...+n!.

Методические указания

Оглавление

3.1. Оператор if

Условный оператор if используется для разветвления процесса вычислений на два направления

Формат оператора:

```
if ( логическое_выражение ) оператор_1; [ else оператор_2; ]
```

В квадратных скобках указана необязательная часть, т.к. ветвь else может отсутствовать. Сначала вычисляется логическое выражение. Если оно имеет значение true, выполняется первый оператор, иначе — второй. После этого управление передается на оператор, следующий за условным.

Если в какой-либо ветви требуется выполнить несколько операторов, их необходимо заключить в блок. Блок может содержать любые операторы, в том числе описания и другие условные операторы, но не может состоять из одних описаний.

Примеры условных операторов:

```
if ( a < 0 ) b = 1;                                                    // 1
if ( a < b && ( a > d || a == 0 ) ) b++; else { b *= a; a = 0; }  // 2
if ( a < b ) if ( a < c ) m = a; else m = c;
else         if ( b < c ) m = b; else m = c;                           // 3
if ( b > a ) max = b; else max = a;                                    // 4
```

Если требуется проверить несколько условий, их объединяют знаками логических условных операций. Например, выражение в примере 2 будет истинно в том случае, если выполнится одновременно условие a < b и одно из условий в скобках. Оператор в примере 3 вычисляет наименьшее значение из трех переменных. Обратите внимание, что компилятор относит часть else к ближайшему ключевому слову if.

Полный синтаксис оператора if выглядит следующим образом:

```
if(выражение_1) оператор_1
else if(выражение_2) оператор_2
…
```

```
else if(выражение_K) оператор_K
else оператор_N
```

Логические выражения if заключаются в круглые скобки и имеют значения true или false. Каждый из операторов может быть блоком, в частности, if-оператором. Поэтому возможна и такая конструкция:

```
if(выражение1) if(выражение2) if(выражение3) …
```

Ветви else if, позволяющие организовать выбор из многих возможностей, могут отсутствовать. Может быть опущена и заключительная else-ветвь.

Семантика оператора if проста и понятна. Выражения if проверяются в порядке их написания. Как только получено значение true, проверка прекращается и выполняется оператор (это может быть блок), который следует за выражением, получившим значение true. С завершением этого оператора завершается и оператор if. Ветвь else, если она есть, относится к ближайшему открытому if.

3.2. Цикл for

Оператор цикла for обобщает известную конструкцию цикла типа арифметической прогрессии. Его синтаксис:

```
for(инициализаторы; условие; список_выражений) оператор
```

Рассмотрим простой пример:

```
for (int i = 0; i < 10; i++)
{
    Console.WriteLine(i);
}
```

Цикл изменяет значение счетчика ***i*** от 0 до 9 включительно и выводит эти значения на консоль. В случае если действие в цикле одно – фигурные скобки можно опустить.

Оператор, стоящий после закрывающей скобки, задает тело цикла. В большинстве случаев телом цикла является блок. Сколько раз будет выполняться тело цикла, зависит от трех управляющих элементов, заданных в скобках. **Инициализаторы** задают начальное значение одной или нескольких переменных, часто называемых счетчиками или просто переменными цикла. В большинстве случаев цикл for имеет один счетчик, но часто полезно иметь несколько счетчиков, что и будет продемонстрировано в следующем примере. **Условие** задает условие окончания цикла, соответствующее выражение при вычислении должно получать значение true или false**. Список выражений**, записанный через запятую, показывает, как меняются счетчики цикла на каждом шаге выполнения. Если условие цикла истинно, то выполняется тело цикла, затем изменяются значения счетчиков и снова проверяется условие. Как только условие становится ложным, цикл завершает свою работу.

Инициализация служит для объявления величин, используемых в цикле, и присвоения им начальных значений. В этой части можно записать несколько операторов, разделенных запятой, например:

```
for ( int i = 0, j = 20; ...) {…}
int k, m;
for ( k = 1, m = 0; ...) {…}
```

Обратите внимание, что областью действия переменных, объявленных в части инициализации цикла, является цикл! За пределами цикла эти переменные будут не видны. Инициализация выполняется один раз в начале исполнения цикла.

Выражение типа bool определяет условие выполнения цикла: если его результат равен true, цикл выполняется.

Модификации выполняются после каждой итерации цикла и служат обычно для изменения параметров цикла. В части модификаций можно записать несколько операторов через запятую, например:

```
for ( int i = 0, j = 20; i < 5 && j > 10; i++, j-- ) ...
```

Простой или составной оператор представляет собой тело цикла. Любая из частей оператора for может быть опущена (но точки с запятой надо оставить на своих местах!).

Для примера вычислим сумму чисел от 1 до 100:

```
int s = 0;
for (int i = 1; i <= 100; i++) s += i;
```

В цикле for тело цикла может ни разу не выполняться, если условие цикла ложно после инициализации, а может происходить зацикливание, если условие всегда остается истинным. В нормальной ситуации тело цикла выполняется конечное число раз.

Цикл for может не содержать ни инструкции инициализации, ни инструкции проверки, ни инструкции итерации. Два оператора (;) внутри цикла for означают вечный цикл: for(;;).

3.3. Циклы while

Цикл while (выражение) является универсальным видом цикла, включаемым во все языки программирования. Тело цикла выполняется до тех пор, пока остается истинным выражение while (выражение должно быть логического типа). В языке C# у этого вида цикла две модификации - с проверкой условия в начале и в конце цикла. Первая модификация имеет следующий синтаксис:

```
while(выражение) оператор;
```

Рассмотрим пример, аналогичный циклу for, который выводит на консоль числа от 0 до 9:

```
int i = 0;
while (i < 10)
{
    Console.WriteLine(i);
    i++;
}
```

Модификация ***while*** соответствует стратегии: "сначала проверь, а потом делай". В результате проверки может оказаться, что и делать ничего не нужно. Тело такого цикла может ни разу не выполняться. Конечно же, возможно и зацикливание. В нормальной ситуации каждое выполнение тела цикла - это очередной шаг к завершению цикла.

Выражение вычисляется перед каждой итерацией цикла. Если при первой проверке выражение равно false, цикл не выполнится ни разу.

В качестве примера рассмотрим программу, выводящую для аргумента x, изменяющегося в заданных пределах с заданным шагом, таблицу значений следующей функции:

$$\mathbf{X} = \begin{pmatrix} x, x < 0 \\ tx, o \leqslant x < 10 \\ 2t, x \geqslant 10 \end{pmatrix}$$

Назовем начальное значение аргумента Xn, конечное значение аргумента — Xk, шаг изменения аргумента — dX и параметр t. Все величины вещественные. Программа должна выводить таблицу, состоящую из двух столбцов: значений аргумента и соответствующих им значений функции.

Текст программы:

```
        static void Main(string[] args)
        {
            double Xn = -2, Xk = 12, dX = 2, t = 2, y;
            Console.WriteLine( "|   x    |    y   |" ); // заголовок таблицы

            double x = Xn;
            while ( x <= Xk )
            {
                y = t;
                if ( x >= 0 && x < 10 ) y = t * x;
                if ( x >= 10 )          y = 2 * t;
                Console.WriteLine( "| {0,6} | {1,6} |", x, y );
                x += dX;
        }
```

Цикл, проверяющий условие завершения в конце, соответствует стратегии: "сначала делай, а потом проверь". Тело такого цикла выполняется, по меньшей мере, один раз. Вот синтаксис этой модификации:

```
do
      оператор
while(выражение);
```

Такая реализация удобна для ввода пользовательских данных. Когда требуется ввести несколько значений подряд.

```
            do
            {
                s=Console.ReadLine();
                Console.WriteLine("Введено: "+s);
            }
            while(s!="");
```

Подробнее о способах ввода в циклах см. в разделе 3.5.

3.4. Операторы break и continue

В структурном программировании признаются полезными "переходы вперед" (но не назад), позволяющие при выполнении некоторого условия выйти из цикла, из оператора выбора, из блока. Операторы break и continue специально предназначены для этих целей.

При выполнении оператора break в теле цикла завершается выполнение самого внутреннего цикла. В теле цикла чаще всего оператор break помещается в одну из ветвей оператора if, проверяющего условие преждевременного завершения цикла.

```
            for (int i = 0; i < 100; i++)
            {
                if (i==33)
                    break;
                Console.WriteLine(i);
            }
```

Оператор continue используется только в теле цикла. В отличие от оператора break, завершающего внутренний цикл, continue осуществляет переход к следующей итерации этого цикла.

```
            for (int j = 0; j < 100; j++)
            {
                if (j % 2 == 0)
                    continue;
                Console.WriteLine("{0}", j);
            }
```

3.5. Вечные циклы

При написании приложений с использованием циклов вам следует остерегаться зацикливания программы. Зацикливание — это ситуация, при которой условие выполнения цикла всегда истинно и выход из цикла невозможен. Давайте рассмотрим простой пример.

```
        static void Main(string[] args)
        {
            int n1, n2;
            n1 = 0;
            n2 = n1 + 1;
            while (n1 < n2)
            {
                Console.WriteLine("n1 = {0} , n2 = {1} ", n1, n2);
            }
        }
```

Здесь условие (n1 < n2) всегда истинно. Поэтому выход из цикла невозможен. Следовательно, программа войдет в режим вечного цикла. Такие ошибки являются критическими, поэтому следует очень внимательно проверять условия выхода из цикла.

Однако иногда бывает полезно задать в цикле заведомо истинное условие. Типичным примером вечного цикла является следующая запись:

```
while(true)
{...}
```

Возможно, что такая конструкция приведет к зависанию системы, если не задать в теле цикла инструкцию его прерывания. Рассмотрим пример программы:

```
        static void Main(string[] args)
        {
            String Name;
            while (true)
            {
                Console.Write("Введите ваш имя ");
                Name = Console.ReadLine();
                Console.WriteLine("Здравствуйте {0}", Name);
            }
        }
```

Такая программа не имеет выхода. Что бы не ввел пользователь, программа выдаст строку приветствия и запросит ввод имени заново. Однако все изменится, если в программу добавить условие, при выполнении которого цикл прерывается.

```
static void Main(string[] args)
{
    string Name;
    while (true)
    {
        Console.WriteLine("Введите ваш имя ");
        Name = Console.ReadLine();
        if (Name == "")
            break;
        Console.WriteLine("Здравствуйте {0 ) ", Name);
    }
}
```

На этот раз, как только пользователь нажмет клавишу «Enter» без ввода строки данных, сработает инструкция break, и программа выйдет из цикла. Создание вечных циклов оправдывает себя, если существует несколько условий прерывания цикла и их сложно объединить в одно выражение, записываемое в блоке условия. Вечный цикл можно создать не только при помощи оператора while. Любой оператор цикла может быть использован для создания вечных циклов. Вот как выглядит та же программа, но с использованием цикла for:

```
static void Main(string[] args)
{
    string Name;
    for (; ; )
    {
        Console.Write("Введите ваш имя ");
        Name = Console.ReadLine();
        if (Name == "")
            break;
        Console.WriteLine("Здравствуйте {0} ", Name);
    }
}
```

Источники:

1. Биллиг В. Основы программирования на C#. Режим доступа: http://www.intuit.ru/studies/courses/2247/18/info
2. Павловская Т.А. Программирование на C#. Учебный курс. Режим доступа: http://ips.ifmo.ru/courses/csharp/index.html
3. Изучаем C Sharp (C#). Программирование на C Sharp (C#) с нуля. Режим доступа: http://simple-cs.ru/