# ПРАКТИЧЕСКАЯ РАБОТА № 4. ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ РАЗВЕТВЛЯЮЩЕЙСЯ СТРУКТУРЫ

# НА ЯЗЫКЕ C#

**Цель работы**: овладение практическими навыками разработки и программирования вычислительного процесса разветвляющейся структуры на языке С#, получить опыт работы с условными операторами языка С#.

**Теоретическая часть**

На практике решение большинства задач не удается описать с помощью программ линейной структуры. При этом после проверки некоторого условия выполняется та или иная последовательность операторов, однако происходит нарушение естественного порядка выполнения операторов.

Операторы ветвления позволяют изменить порядок выполнения операторов в программе. К операторам ветвления относятся условный оператор *if* и оператор выбора *switch*.

Оператор *if* обеспечивает передачу управления на одну из двух ветвей вычислений, а оператор *switch* — на одну из произвольного числа ветвей.

Условный оператор if может иметь одну из форм: сокращенную или полную.

Форма сокращенного оператора if: **if (B) S;**

где В - логическое или арифметическое выражение, истинность которого проверяется; S - оператор: простой или составной.

При выполнении сокращенной формы оператора if сначала вычисляется выражение B, затем проводится анализ его результата: если B истинно, то выполняется оператор S; если B ложно, то оператор S пропускается. Таким образом, с помощью сокращенной формы оператора if можно либо выполнить оператор S, либо пропустить его.

Форма полного оператора if:

**if (B) S1; else S2;**

где B - логическое или арифметическое выражение, истинность которого проверяется; S1, S2- оператор: простой или составной.

При выполнении полной формы оператора if сначала вычисляется выражение B, затем анализируется его результат: если B истинно, то выполняется оператор S1, а оператор S2 пропускается; если B ложно, то выполняется оператор S2, а S1 - пропускается. Таким образом, с помощью полной формы оператора *if* можно выбрать одно из двух альтернативных действий процесса обработки данных.

Рассмотрим несколько примеров записи условного оператора if:

if (a > 0) x=y; // Сокращенная форма c простым оператором

if (++i) {x=y; y=2\*z;} // Сокращенная форма c составным оператором

if (a > 0 || b<0) x=y; else x=z; // Полная форма с простым оператором

if (i+j-1) { x= 0; y= 1;} else {x=1; y:=0;} // Полная форма с составными операторами

Пример использования условного оператора.

static void Main()

{

Console.Write("x= ");

float x = float.Parse(Console.ReadLine());

Console.Write("y=");

float y = float.Parse(Console.ReadLine());

if (x < y ) Console.WriteLine("min= "+x);

else Console.WriteLine("min= "+y);

}

Результат работы программы:

x y min

0 0 0

1 -1 -1

-2 2 -2

Операторы S1 и S2 могут также являться операторами *if*. Такие операторы называют вложенными. При этом ключевое слово *else* связывается с ближайшим предыдущим словом *if*, которое еще не связано ни с одним *else*.

Оператор *switch* (переключатель) предназначен для разветвления процесса вычислений на несколько направлений. Формат оператора:

**switch ( <выражение> )**

**{**

**case <константное\_выражение\_1>: [<оператор 1>]; <оператор перехода>;**

**case <константное\_выражение\_2>: [<оператор 2>]; <оператор перехода>;**

**...**

**case <константное\_выражение\_n>: [<оператор n>]; <оператор перехода>;**

**[default: <оператор>; ]**

**}**

Выражение, записанное в квадратных скобках, является необязательным элементом в операторе *switch*. Если оно отсутствует, то может отсутствовать и оператор перехода.

Выражение, стоящее за ключевым словом *switch*, должно иметь арифметический, символьный, строковый тип или тип указатель. Все константные выражения должны иметь разные значения, но их тип должен совпадать с типом выражения, стоящим после *switch* или приводиться к нему. Ключевое слово *case* и расположенное после него константное выражение называют также меткой *case*.

Выполнение оператора начинается с вычисления выражения, расположенного за ключевым словом *switch*. Полученный результат сравнивается с меткой *case*. Если результат выражения соответствует метке *case*, то выполняется оператор, стоящий после этой метки, за которым обязательно должен следовать оператор перехода: *break*, *goto* и т.д. При использовании оператора *break* происходит выход из *switch* и управление передается оператору, следующему за *switch*. Если же используется оператор *goto*, то управление передается оператору, помеченному меткой, стоящей после *goto*.

Если совпадения не произошло, выполняются операторы, расположенные после слова *default* (при его отсутствии управление передается следующему за *switch* оператору).

Если необходимо, чтобы для разных меток выполнялось одно и тоже действие, то метки перечисляются через двоеточие.

*Пример*:

switch (oper)

{

case '+': res = a + b; break;

case '-': res = a - b; break;

case '\*': res = a \* b; break;

case '/': case ':': if (b != 0) // перечисление меток

{

res = (float)a / b; break;

}

else goto default;

default: ok = false; break;

}

*Операторы безусловного перехода*

В С# есть несколько операторов, изменяющих естественный порядок выполнения команд: оператор безусловного перехода *goto*, оператор выхода *break*, оператор перехода к следующей итерации цикла *continue*, оператор возврата из метода *return* и оператор генерации исключения *throw*.

##### Оператор безусловного перехода goto

Оператор безусловного перехода goto имеет формат:

**goto <метка>;**

В теле той же функции должна присутствовать ровно одна конструкция вида:

**<метка>: <оператор>;**

Оператор goto передает управление на помеченный меткой оператор. Рассмотрим пример использования оператора *goto*:

static void Main()

{

float x;

metka: Console.WriteLine("x="); //оператор, отмеченный меткой

x = float.Parse(Console.ReadLine());

if (x!=0) Console.WriteLine("y({0})={1}", x, 1 / x );

else

{

Console.WriteLine("функция не определена");

goto metka; // передача управление метке

}

}

Следует учитывать, что использование оператора *goto* затрудняет чтение больших по объему программ, поэтому использовать метки нужно только в крайних случаях, например, в операторе *switch*.

##### Оператор выхода break

Оператор *break* используется внутри операторов ветвления и цикла для обеспечения перехода в точку программы, находящуюся непосредственно за оператором, внутри которого находится *break*.

Операции отношения (<, >,==, <, >, !=) являются бинарными, то есть имеют два операнда, и формируют результат типа bool, равный true или false.

**Вопросы для самоконтроля**

1. Какие условные операторы существуют в языке С#?
2. Для чего используется оператор if?
3. Как в операторе if проверить сразу несколько условий?
4. Как осуществляется выход из оператора switch?
5. Какого типа могут быть константные выражения, используемые в операторе switch?
6. Когда в операторе switch выполняется ветвь default?
7. Какие служебные слова используются при записи условного оператора?
8. Приведите пример составного оператора.
9. Как называются программы, в основе которых лежит структура следование?
10. Какой тип может иметь выражение, стоящее за ключевым словом *switch*?
11. Назначение ветви default оператора *switch*?
12. Назовите операторы перехода языка С#.
13. Каким образом будет выглядеть запись оператора *switch*, если необходимо, чтобы для разных меток выполнялось одно и тоже действие?

**Практическая часть**

**Задание №1.** Написать программу, которая вычисляет значение функции *у*:



















































**Задание №2.** Написать программу, которая определяет:

1. максимальное значение для двух различных вещественных чисел.
2. является ли заданное целое число четным.
3. является ли заданное целое число нечетным.
4. если целое число М делится на целое число N, то на экран выводится частное от деления, в противном случае выводится сообщение "M на N нацело не делится".
5. оканчивается ли данное целое число цифрой 7.
6. имеет ли уравнение *ax*2+*bx*+*c*=0 решение, где *a*, *b*, *c* – данные вещественные числа.
7. какая из цифр четырехзначного числа больше: первая или вторая.
8. одинаковы ли цифры данного трехзначного числа
9. является ли сумма цифр двухзначного числа четной.
10. является ли сумма цифр трехзначного числа нечетной.
11. кратна ли трем сумма цифр двухзначного числа.
12. кратна ли числу А сумма цифр двухзначного числа.
13. какая из цифр трехзначного числа больше: первая или последняя.
14. какая из цифр трехзначного числа больше: первая или вторая.
15. какая из цифр трехзначного числа больше: вторая или последняя.
16. все ли цифры трехзначного числа одинаковые.
17. существует ли треугольник с длинами сторон *a*, *b*, *c*.
18. является ли треугольник с длинами сторон *a*, *b*, *c* прямоугольным.
19. является ли треугольник с длинами сторон *a*, *b*, *c* равнобедренным.
20. является ли треугольник с длинами сторон *a*, *b*, *c* равносторонним.
21. сумму второй и последней цифр четырехзначного числа *А*.
22. произведение первой и последней цифр четырехзначного числа *А*.
23. сумму первой, третьей и последней цифр пятизначного числа *А*.
24. произведение второй и четвертой цифр пятизначного числа *А*.
25. кратна ли числу *А* сумма цифр семизначного числа.

**Задание №3.** Дана точка на плоскости с координатами (*х*, *у*). Составить программу, которая выдает одно из сообщений "Да", "Нет", "На границе" в зависимости от того, лежит ли точка внутри заштрихованной области, вне заштрихованной области или на ее границе.

Области задаются графически следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| 1.  04_01 | 2.  04_02 | 3.  04_03 | 4.  04_04 |
| 5.  04_05 | 6.  04_06 | 7. | 8. |
| 9.  04_09 | 10. | 11.  04_11 | 12. |
| 13.  04_13 | 14.  04_14 | 15.  04_15 | 16.  04_16 |
| 17.  04_17 | 18.  04_18 | 19. | 20.  04_20 |
| 21. | 22. | 23. | 24. |
| 25. |  |  |  |