Семинар 2. Управляющие операторы языка

Особенности обработки ветвящихся процесов.

1. Управляющие операторы С\С++

***Управляющими*** называют операторы, способные изменять линейность процесса вычислений. К таким операторам относятся оператор условной передачи управления if, оператор выбора switch, операторы циклов while, do, for и операторы безусловной передачи управления goto, break, continue, exit.

## *1.1 Блок операторов и пустой оператор*

При объявлении управляющих операторов широко используют конструкции «блок операторов» и «пустой оператор».

***Блоком операторов*** называют последовательность операторов, заключенную в фигурные скобки:

**{<Оператор>;… <Оператор>;**}

Например:

{

f=a+b;

a+=10;

}

Особенность блока операторов заключается в том, что при анализе синтаксиса он рассматривается как единый оператор. Это позволяет использовать его там, где по правилам записи конструкций возможен только один оператор, а по алгоритму программы должны выполняться несколько действий.

***Пустой оператор*** включает только символ «;», перед которым нет никакого выражения или не завершенного разделителем оператора. Пустой оператор не предусматривает выполнения никаких действий. Он используется там, где синтаксис языка требует присутствия оператора, а по алгоритму программы никакие действия не должны выполняться.

## *1.2 Операторы ветвления*

Для программирования ветвлений, т. е. ситуаций, когда возникает необходимость при выполнении условия реализовывать одни действия, а при нарушении – другие, используют оператор условной передачи управления.

В качестве примера разветвляющегося вычислительного процесса рассмотрим алгоритм нахождения корней квадратного уравнения.

***Пример 1.*** Определение действительных корней квадратного уравнения:

*ax2 + bx + c = 0*.

В зависимости от значения дискриминанта *D* = *b*2 ­- 4*ac*, уравнение имеет либо два действительных корня, либо один, либо вовсе не имеет действительных корней. Поэтому необходимо предварительно вычислить дискриминант *D* и проверить выполнение условий *D****<***0 и *D*=0. Схема алгоритма вычисления корней квадратного уравнения приведена на рисунке 1.

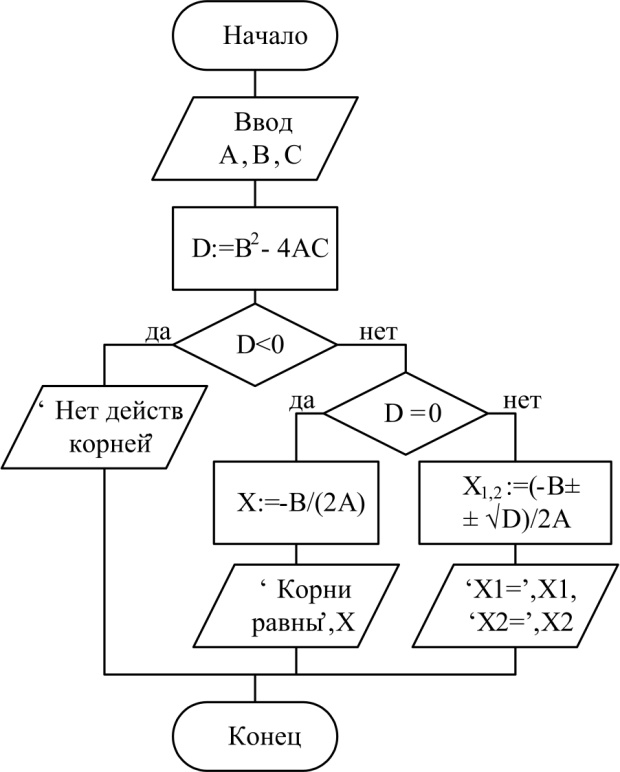


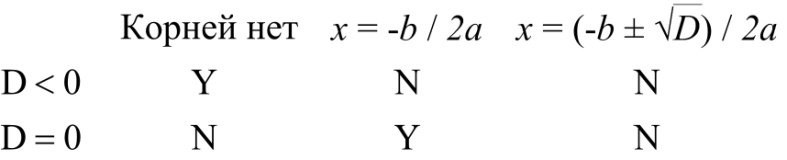
Рисунок 1 – Схема алгоритма нахождения действительных корней квадратного уравнения

Если условий много, то процесс составления алгоритма, содержащего минимальное количество проверок, может вызвать некоторые трудности. В этом случае удобно использовать так называемые *таблицы решений*.

**Таблицы решений.** Таблица решений составляется следующим образом. По вертикали выписывают все условия, от которых зависят дальнейшие вычисления, а по горизонтали – все варианты вычислений. На пересечении каждого столбца и строки указывают:

* букву Y, если для данного варианта условие должно выполняться,
* букву N, если условие обязательно должно не выполняться,
* прочерк, если исход сравнения не важен.

Например, для алгоритма вычисления корней квадратного уравнения *ax2 + bx + c = 0* можно составить следующую таблицу:



Схему алгоритма строят по таблице. Сначала проверяют выполнение первого условия. Из таблицы следует, что первое условие должно выполняться только для первого варианта решения, поэтому после его проверки ветвь «да» соответствует случаю «нет корней». В ветви «нет» необходимо проверить условие D=0 и в зависимости от его выполнения указать оставшиеся два случая. В итоге получаем алгоритм, схема которого приведена выше.

Иногда составленная таблица решений приобретает сложный вид. Рассмотрим, например, таблицу:



где P1, P2, P3, P4 – варианты решений.

Если сразу приступить к построению алгоритма, то будет получена схема довольно громоздкого алгоритма, приведенная на рисунке 2.



Рисунок 2 – Фрагмент схемы неоптимального алгоритма

Такой алгоритм нельзя считать приемлемым. Однако его можно существенно упростить, если в таблице поменять местами проверяемые условия. Кроме того, для удобства построения алгоритма целесообразно также поменять местами столбцы таблицы:



Алгоритм, построенный по такой оптимизированной таблице, окажется значительно проще, и его построение потребует меньших усилий (см. рисунок 3).



Рисунок 3 – Фрагмент схемы оптимизированного алгоритма

Для реализации таких алгоритмов в процедурных языках С\С++ используются операторы условной передачи управления и ветвления.

**1.2.1 Оператором условной передачи** управления называют конструкцию, позволяющую выбрать одно из возможных продолжений вычислительного процесса в зависимости от результата условия. Он реализует алгоритмическую конструкцию Ветвление (см. рисунок 4).



Рисунок 4– Конструкция Ветвление

На языке С/С++ оператор ветвления имеет следующий вид:

**if (<Выражение>) <Оператор;> [ else <Оператор;>]**

Квадратные скобки показывают, что описание ветви else – необязательно.

*Выражение* – любое выражение, записанное по правилам Си или С++.

Если результат выражения не равен нулю, то выполняется оператор, следующий за выражением.

Если результат выражения равен нулю, то выполняется оператор альтернативной ветви.

При отсутствии описания альтернативной ветви управление передается следующему за if оператору.

В качестве **оператора** может записываться любой оператор С\С++, в том числе другой оператор условной передачи управления, пустой оператор и блок операторов.

По синтаксису, альтернативная ветвь может отсутствовать, а в качестве оператора может быть другой оператор if. В этом случае работают правила вложенности.

Примеры:

а)if (!b)puts("с - не определено"); **//** если b=0, то – ошибка,

else {c=a/b; printf("c=%d\n", c);} **//**  иначе выводится значение с.

б) **if ((c=a+b)!=5) c+=b;**

else **c=a;**

в)if ((ch=getchar())==′q′) **//**  если в ch введено q,

puts ("Программа завершена."); **//** то ...

else puts ("Продолжаем работу..."); **//** иначе ...

**Правило вложенности.**

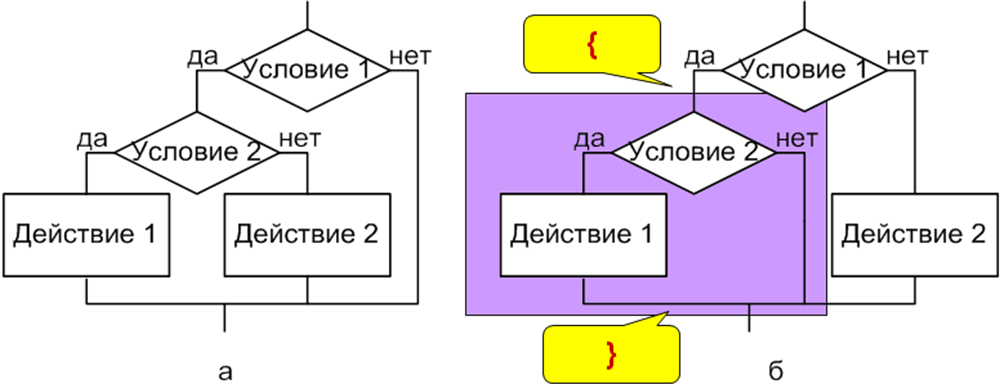
При вложенности операторов ветвления ветвь **else** всегда относится к ближайшему **if.**

При записи двух вложенных операторов if с одной альтернативной ветвью возникает неоднозначность: не понятно, которому из двух операторов if принадлежит альтернативная ветвь (см. рисунок).

if (<Условие 1> )

if <Условие 2> <Действие 1>

else <Действие 2>



Приведенный выше фрагмент реализует алгоритм варианта а.

Для реализации варианта *б* используют блок операторов {…}:

**if** <Условие1>

**{if** <Условие2> <Действие1> }

**else** <Действие 2>

При необходимости проверить последовательно несколько условий возможно каскадирование условных операторов:

**if** условие1

команды1

**else**

**if** условие2 команды2

**else**

**if** условие3 команды3

...

**else**

**if** условиеN+1 командыN+1

**else** команды;

В этом случае условия будут проверяться последовательно, и как только встретится истинное, будет выполнен соответствующий набор команд и исполнение перейдёт к команде, следующей за условным оператором. Если ни одно из условий не окажется истинным, выполняются команды из ветви else.

***Тестирование ветвящихся алгоритмов.*** Тестирование – это важнейший процесс разработки программ. Оно позволяет обнаружить в программе ошибки, чтобы потом их локализовать и исправить. Следует отметить особенности тестирования и отладки ветвящихся алгоритмов. При тестировании таких алгоритмов, тестовые наборы выбираются так, чтобы каждая ветвь алгоритма была выполнена хотя бы один раз. То есть, количество тестов должно быть не меньше, чем ветвей. Но, иногда их может быть и больше. Это объясняется тем, что при составлении алгоритма разработчик мог просто упустить какое-то важное условие. Никакой тест эту ошибку не обнаружит. Но по косвенным признакам дополнительных тестов это можно предвидеть.

Рассмотрим несколько задач на ветвления.

Задача 1. Определить площадь треугольника по заданным трем его сторонам.

Анализ задачи. Для вычисления площади треугольника по трем сторонам a,b,c используется формула Герона.



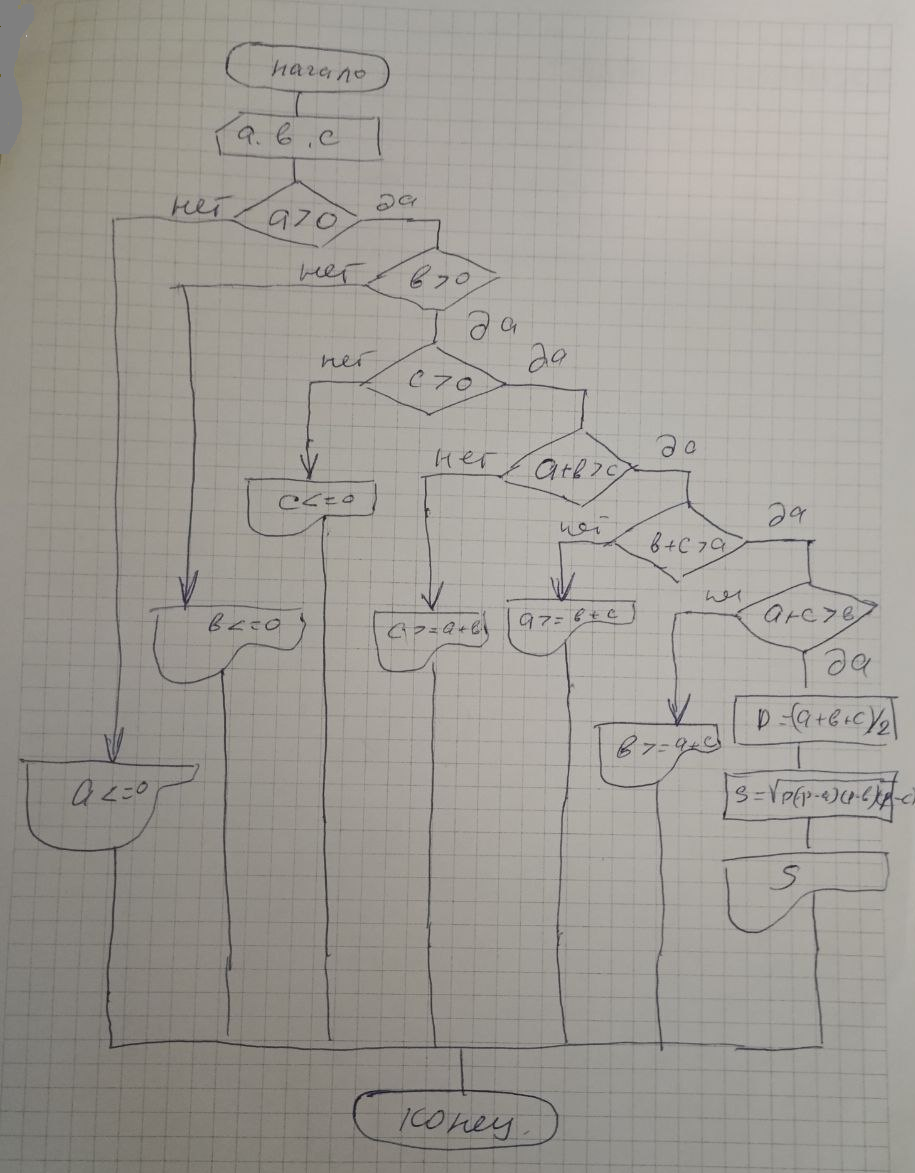
где p=(a+b+c)/2 - полупериметр треугольника.

Однако, не любые три числа являются сторонами треугольника. Условия существования треугольника, заданного тремя длинами сторон:

1. длина стороны треугольника – положительные числа;
2. длина каждой стороны меньше суммы двух других.

Получается 6 условий. Условия можно проверять по одному. А можно объединить их по три. Однако, с целью облегчения отладки и обучения этому процессу на первых примерах лучше условия проверять по одному, с выдачей сообщений при их невыполнении. Как видно из анализа, мы имеем дело с вложенными операторами ветвления

Тогда схема алгоритма решения задачи выглядит так.



Текст программы.

**#include <stdio.h> // подключение процедур ввода вывода**

**#include <math.h> // подключение математических функций**

**int main()**

**{float a,b,c,p,s;**

**puts("Input side of treangle:");**

**scanf("%f %f %f",&a,&b,&c);**

**printf("inputed a=%7.2f, b=%7.2f, c=%7.2f.\n",a,b,c);**

**if(a>0)**

**if(b>0)**

**if(c>0)**

**if((a+b)>c)**

**if((a+c)>b)**

**if((c+b)>a)**

**{ p=(a+b+c)/2;**

**s=sqrt(p\*(p-a)\*(p-b)\*(p-c));**

**printf("Square of treandle=%12.5f\n",s);**

**}**

**else**

**printf("Side a<=c+b\n");**

**else**

**printf("Side b<=a+c\n");**

**else**

**printf("Side c<=a+b\n");**

**else**

**printf("Side c<=0\n");**

**else**

**printf("Side b<=0\n");**

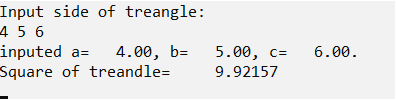
**else**

**printf("Side a<=0\n");**

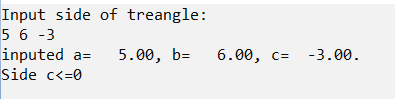
**return 0;**

**}**

**Примеры работы программы**







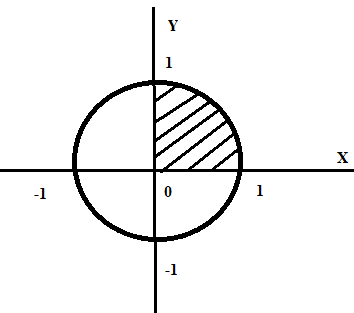
**Тестирование программы**

Ожидаемый результат можно получить, используя любой онлайн калькулятор, или вычислить вручную. Так как проверяется 6 условий, тестов должно быть не менее 7.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | a | b | c | Ожидаемый результат | Полученный результат |
| 1 | 4.0 | 5.0 | 6.0 | S=9.922 | S=9.92157 |
| 2 | 7.0 | 2.0 | 4.0 | Треугольник не существует | Сторона a больше b+c |
| 3 | 5.0 | 6.0 | -3.0 | Треугольник не существует | Сторона с отрицательна |

……….

**Задача 2.** Определить принадлежность точки заданной области.



**Анализ задачи**.

Как видно их рисунка, область поиска ограничена Окружностью единичного радиуса и двумя прямыми x=0 и y=0.

Больше всего точек отсекает окружность. Только точки, лежащие **внутри** окружности и на ее **границе**, могут лежать в заштрихованной области. Используя формулу окружности, определим первое условие

X^2+Y^2<=R^2 или, при R=1 получаем

X^2+Y^2<=1;

Второе условие, это точка должна лежать внутри окружности, но выше примой X=0, а именно X>=0;

Третье условие, это точка должна лежать внутри окружности, но правее примой Y=0, а именно Y>=0;

Алгоритм программы будет выглядеть так:



**Текст программы**

**#include <stdio.h> // подключение процедур ввода вывода**

**#include <math.h> // подключение математических функций**

**int main()**

**{float x,y;**

**puts("Input x and y:");**

**scanf("%f %f",&x,&y);**

**printf("inputed x=%7.2f, y=%7.2f.\n",x,y);**

**if(x\*x+y\*y<=1)**

**if(x>=0)**

**if(y>=0)**

**printf("Point in Square of figure\n");**

**else**

**printf("Point out Square, y<0\n");**

**else**

**printf("Point out Square, x<0\n");**

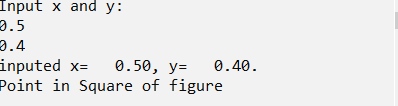
**else**

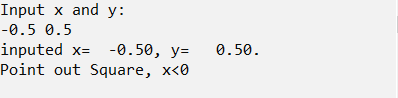
**printf("Point out Square, x^2+y^2>1\n");**

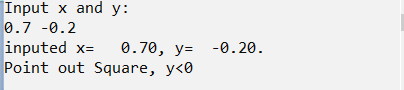
**return 0;**

**}**

**Примеры работы программы**

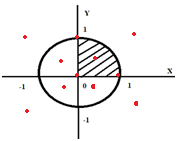






**Тестирование программы**

Хотя условий проверки всего три, количество тестов должно быть в данном случае гораздо больше, но не менее 4. Необходимо проверить, определяется ли правильно точка во всех областях НЕ принадлежности.



На рисунке показаны точки, которые необходимо включить в тестовый набор, а затем занести результаты в таблицу вида:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | X | Y | Ожидаемый результат | Полученный результат |
| 1 | 0.5 | 0.5 | Точка принадлежит обл. | Принадлежит |
| 2 | 0.7 | -0.2 | Точка не принадлежит обл. y<0 | Не принадлежит Y<0 |

………..

И в таблице, и в программе для случая **НЕ** принадлежности точки области, следует указывать причину, по которой это происходит. Это значительно облегчить отладку программы.

**Задача 3.** Вычислить значение функции, заданной следующим образом.

Y= где: z= sin(x), при x<=a;

z= x^3, при a<x<b;

z= 1+x, при x>=b.

x,a,b вводятся с клавиатуры и a<b.

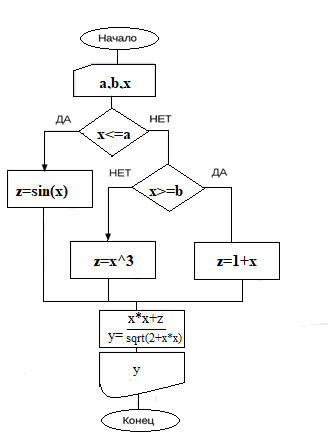
**Анализ задачи.**

Как видно из условия, значение переменной Y зависит от двух переменных x,z. И если х задается вводом с клавиатуры, то значение переменной z зависит от введенного х и значений a,b. Поэтому, сначала нужно проверить х, и в зависимости от его значения вычислить переменную z по указанной формуле, а потом только вычислять переменную Y. Для проверки ввода a,b, удовлетворяющих условию, можно указать условие в подсказке при вводе. Однако, это не исключает не верного ввода и отсюда – ошибки при вычислении. Можно проверить ввод с помощью оператора ветвления, а можно использовать цикл для ввода правильного значения. Но этот вариант будет рассмотрен в разделе циклы. В примере ниже при вводе a и b дается подсказка.

Так как х непрерывно на всем диапазоне от –∞ до +∞, достаточно проверить только два условия, так как в случае невыполнения первых двух условий, третье выполняется автоматически. Проверим более простые условия x<=a и x>=b.

Для этого используем вложенные операторы ветвления.

Схема алгоритма выглядит так:



**Текст программы**

**#include <stdio.h> // подключение процедур ввода вывода**

**#include <math.h> // подключение математических функций**

**int main()**

**{ float a,b,x,z,y;**

**puts("Input a,b, a<b :");**

**scanf("%f %f",&a,&b);**

**puts("Input x :");**

**scanf("%f",&x);**

**printf("inputed x=%7.2f, a=%7.2f, b=%7.2f.\n",x,a,b);**

**if(x<=a)**

**{z=sin(x);**

**printf("z=sin(x)= %10.5f\n",z);**

**}**

**else**

**if(x>=b)**

**{z=1+x;**

**printf("z=1+x= %10.5f\n",z);**

**}**

**else**

**{z=x\*x\*x;**

**printf("z=x\*x\*x= %10.5f\n",z);**

**}**

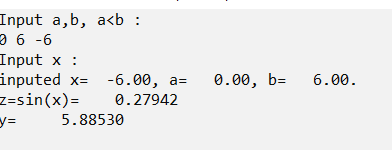
**y=(x\*x+z)/sqrt(2.0+x\*x);**

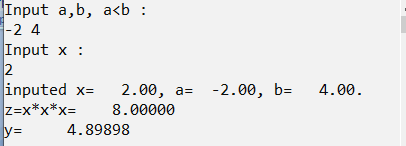
**printf("y=%12.5f\n",y);**

**return 0;**

**}**

**Примеры работы программы**



****

**Тестирование**

Так как веток алгоритма три, количество тестов должно быть не менее трех. Однако, количество тестов будет больше, так как нужно протестировать все три ветки при разных a и b, при одних и тех же х.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | a | b | x | Ожидаемый результат | Полученный результат |
| 1 | 0.0 | 6.0 | -6.0 | z=0.279, y=5.89 | z=0.27942,y=5.88530 |
| 2 | -2.0 | 4.0 | 2.0 | z=8.0, y=4.899 | z=8.0000,y=4.89898 |
| 3 | …. |  |  |  |  |

Задания для выполнения дома:

1. Написать программу определения вида треугольника - равносторонний, равнобедренный, прямоугольный.
2. Написать программу проверки принадлежности точки заштрихованной области

