# Ветвление

Шокуров Антон В.
shokurov.anton.v@yandex.ru
http://машинноезрение.рф

16 декабря 2016 г.

Версия: 0.16

**Аннотация**

В рамках прошлой заметки в программе для любых введенных данных всегда выполнялись одни и те же действия. Такой подход не годится для произвольных программ. Иногда требуется варьирование выполняемых действий в зависимости от тех или иных обстоятельств.

Цель: Выполнять в зависимости от некоего условия дополнительных/иных действий.

Предварительный вариант!

## 1 Ветвление

Языковая конструкция *ветвления* необходима для выполнения в зависимости от некого условия дополнительных действий. Например, выполнение выхода из программы при вводе некорректных данных. В частности, при решении квадратного уравнения в случае отрицательного дискриминанта данная языковая конструкция позволяет прекратить дальнейшие вычисления.

### 1.1 Условные операции

Как было показано в прошлой заметке в Си можно вычислить значение арифметического выражения, например:

$$2 + 3 \sim 5.$$

Но, в этих выражениях можно использовать не только арифметические операции, но и стандартные условные операции сравнения (<, <=, и тому подобные).

В последнем случае выражение тоже будет равно некому определенному числу, например:

$$3 * (2 < 3) + 2 * (-5 >= 11) \sim 3 * 1 + 2 * 0 \sim 3.$$

**бинарная операция** Арифметические выражения были отмечены в прошлой заметке. Так вот над числами можно выполнить и условные выражения:

```
1  int a=10, b=a+25;//Объявление и инициализация переменных.
2  //a~10, b~35.
3  //(a<b)~1, так как истинное утверждение.
4  //(a>b)~0, так как ложное утверждение.
5  //Печать значения условного выражения:
6  printf("viraz ~ %d\n", a>b);//Будет напечатано:
7  //viraz ~ 0
8  int c = (a<=b);//присвоение условного выражения.
9  //c~1, так как было истинное утверждение.
10 c=(a<b)+(a+1<=b);
11 //c~2, сумма двух истинных утверждений.
```

Ровно также как арифметические выражения сводятся к числу (например, $2+3$ к 5), условные выражения тоже сводятся к некому числовому значению. Так, ложное утверждение превращается в 0ое значение (см. строчку 4), а истинное – к 1 (см. строчку 3).

Как показано в строчке 8, значение можно присвоить переменной, например, имеющей тип *int* (целый). Вообще говоря, результат условной операции, как и логического действия, имеет тип *bool* (булевский), но пока нам это не очень важно. Важно то, что его можно присвоить переменной, например, имеющей тип *int*.

Имеются следующие стандартные условные операции:

| операция | название | операция | название |
|---|---|---|---|
| < | меньше | <= | меньше равно |
| > | больше | >= | больше равно |
| == | равно | != | не равно |

**Особенности применения** Следует отметить, что дву-символьные операции должны писаться ровно так, как записано в таблице, т.е. без пробелов между составными символами (см. стр. 2) и именно в указанном порядке (см. стр. 3). Можно например так как в стр. 4.

```
1  int d;
2  d = (a< =b);//так нельзя писать, пробел нужно убрать.
```

```
3 d = (a=<b);//ровно как и так, порядок нужно поменять.
4 d = (a <= b);//можно, например, так.
```

Особо обращаю внимание на условные операции (строка 3 таблицы) проверки на равенство и на неравенство значений. Проверке на равенство, да, соответствует комбинация из двух традиционных символов равно (==): не путать с оператором присвоения (=)! Если в условном выражении вместо двойного символа равно применить однократное, то не стоит ожидать, что компилятор об этом предупредит или выдаст сообщение об ошибке. Он может выдать, а может и не выдать предупреждающее сообщение. В любом случае, программа не будет работать корректно, т.е. для определенных значений выдаст ошибочный результат.

Особо отмечу, что отмеченные выше операции проверки на равенство и на неравенство нельзя использовать с вещественными числами. Детали о том как быть в случае вещественных чисел будут даны на одном из следующих далее семинаре.

**Контрольные упражнения** Упражнение. Напиши программу, которая печатает значение условных выражений. Поиграйся с условными операциями. В особенности с == и !=.

Уже данные рассмотренные бинарные условные операции позволяют реализовать вычисление абсолютного значения:

```
1 int a;
2 scanf("%d", &a);
3 int b;
4 //Вычислим абсолютное значение числа a:
5 b = (a>=0)*a + (a<0)*(-a);
```

Упражнение. Убедись в правильности предыдущей программы. Какие есть подвохи?

## 1.2 Логические выражения

Ветвление выполняется в зависимости от истинности логического выражения, которое может состоять из одной единственной условной операции. Но выражение может быть и более сложным. Например, являться отрицанием некого выражения или строиться из несколько более простых посредством связок И и ИЛИ. Далее пойдет речь об этих вещах.

**Числовое значение** С точки зрения языка Си выражение, являющееся числом, можно использовать в логических выражениях вместо самих логических подвыра-

жений, в частности, в конструкциях ветвления. В согласии с последним, 0ю соответствует ложь, а любому ненулевому числу – истина. Фактически, при проверке числа на истинность в языке Си проверяется условная операция на неравенство числа нулю. Таким образом, выражения

$$13, -5, 3 - 5, 2 * (-7)$$

имеют истинное логическое значение, а выражение

$$0$$

имеет ложное логическое значение.

**Унарный оператор** Восклицательный знак (!), поставленный перед выражением, выполняет его логическое отрицание, причем, результат уже будет согласован со значением условных операций, а именно – ложному утверждению соответствует 0 значение (см. строчку 4), а истинному 1 (см. строчку 5).

```
1 int a=10, b=0;//Объявление и инициализация переменных.
2 //a~10, b~0.
3 //a ~ истина, b ~ ложь.
4 //(!a)~0, так как a истинно.
5 //(!b)~1, так как b ложно.
```

Упражнение. Какое значение соответствует выражениям !4 и !0? Напиши программу и проверь свой ответ!

Упражнение. Что будет если воспользоваться двойным восклицательным знаком? Какое значение соответствует выражениям !!4 и !!0? Допиши программу!

**Бинарный оператор** При построение сложных логических утверждений, на подобии $2 <= x < 11$, оно разбивается на логическую комбинацию более простых (рассматриваемое, например, на $2 <= x$ и $x < 11$). Для связывания простых выражений используются операции: И (таб. 1, а), и ИЛИ (таб. 1, б).

Отмеченные операции имеют два операнда, каждый из которых может принимать два значения: истина и ложь. Значение логической операции, в свою очередь, тоже принимает эти же самые два значения. В таблицах (таб. 1) приведено значение этих операций для всевозможных значений операндов.

Исходя из последнего, упомянутое ранее выражение представляется как $(2 <= x)$ и $(x < 11)$, где в языке Си вместо связки "и"будет использована комбинация символов && (можно использовать и ключевое слово `and`).

а)

| И(&&) | | Значение | |
|---|---|---|---|
| | | Истина | Ложь |
| Знач. | Ист. | Ист. | Ложь |
| | Ложь | Ложь | Ложь |

, б)

| ИЛИ(\|\|) | | Значение | |
|---|---|---|---|
| | | Истина | Ложь |
| Знач. | Ист. | Ист. | Ист. |
| | Ложь | Ист. | Ложь |

Таблица 1: Таблицы истинности для И и ИЛИ.

```
1 int a;//Объявление переменной.
2 printf("enter a:");//приглашение на ввод
3 scanf("%d", &a);//считываем значение
4 printf("it is ~ %d\n", (2<=x) && (x<11));//печатаем
5 //истинность (2<=x) && (x<11) => 0 - ложь, 1 - истина.
```

Упражнение. Поиграйся с логическими выражениями. Напиши программу, которая выводит 1 тогда и только тогда, когда вводимое число лежит либо в отрезке $[-5., 9.]$ либо в интервале $(15., 20.)$.

Упражнение. Как записать логическое выражение соответствующее истине для нечетных чисел? А для четных чисел? Как вывести число 1 для четных положительных чисел и 0 в противном случае?

Упражнение. Как сделать так, чтобы в продолжении предыдущих упражнений логическое выражение принимала некие произвольно заданные значения a и b, а не 0 и 1?

## 1.3 Условный переход

Условный переход необходим для указания тех действий, которые будут выполнены при выполнении определенных условий.

**условная (if) конструкция** Условный переход позволяет в зависимости от условия (см. стр. 6) выполнить дополнительную (в общем случае, блок) команду (см. стр. 7):

```
1 //В общем случае:
2 //if( логическое выражение )
3 //      команда/действие;
4 //в частности,
5 //продолжим предыдущую программу.
6 if( (2<=x) && (x<11) )//Отмечу, что здесь нет символа ;
7         printf("yes\n");//Он здесь, у целевой команды.
```

Последняя программа напечатает в консоли "yes"в случае выполнения условия и ничего в противном случае.

Отмечу, что в строчке 6 отсутствует стандартный завершающий символ ; команды. Последние связано с тем, что конструкция `if` является цельной командой, а не составленной из команд. Поэтому символ ; ставится в самом конце, а именно – после целевой команды (см. стр. 7), а не ранее. Последние обстоятельство существенно влияет на логику программы. Если поставить символ ; в конце строки 6, то целевая команда вопреки ожиданиям будет выполнятся всегда!

Замечу, что целевая команда (см. 7) написана с отступом. Такие приняты правила стиля написания программ, в данном случае условной конструкции. Их следует неукоснительно придерживаться, иначе зачета не видать.

Упражнение. Напиши программу вычисления абсолютного значения числа. Программа должна: вывести приветствие, запросить ввод числа, считать число от пользователя, вычислить абсолютное значение, сохранить его в переменную (именно вычислить и сохранить, а не сразу вывести значение на печать!), и напечатать итоговое значение (извлеченной из переменной).

В том случае, если мы хотим, чтобы в противном случае она напечатала "no то в полной аналогии и помня действие символа ! можно дописать к предыдущей программе еще и этот кусок кода:

```
1 //Продолжение предыдущей программы...
2 if( !((2<=x) && (x<11)) )
3         printf("no\n");
```

Выполнено отрицание всего условного выражения: сработает либо то, либо это действие. Таким образом, в любом случае что-то будет напечатано.

**if-else (конструкция иначе)** Но, благо в условной конструкции языка Си существует и чисто языковый способ сделать тоже самое:

```
1 //Продолжим первоначальную программу.
2 if( (2<=x) && (x<11) )
3         printf("yes\n");
4 else
5         printf("no\n");
```

Так, составной частью условной конструкции является термин else (см. стр. 4). Команда ему относящиеся (см. стр. 5) выполняется в случае не удовлетворения основного условия (см. стр. 2).

**Цепочка if-if-...-else**

**Вложенный if-else** В качестве команды `if-else` конструкции может выступать ещё одна `if-else` конструкция.

## 1.4 Блок кода

Локальные переменные. Область видимости. Обычно применяется для построения сложных ветвлений и функций.

Сразу отмечу, что переменные объявленные вне всех блоков называются *глобальными*. Их использование в рамках настоящего курса строго запрещено. Нарушение данного запрете влечет автоматический незачет по данному курсу!

**Блок кода** Данная конструкция необходима для замены единственного действия в некой конструкции на список действий, а также может быть использована для обособления части кода, в частности, переменных. В рассматриваемом фрагменте блок кода (4-16) задается парными фигурными скобками (см. 3 и 17):

```
1  int a=10;//Объявление и инициализация переменной.
2  //a~10.
3  {//блок задается парными фигурными скобками.
4          //a~10.
5          a=2;//можно присвоить переменной a значение.
6          //a~2.
7          int c;//Объявление локальной переменной блока.
8          //a~2, c~мусор.
9          a = 5;//присвоение значения переменной a и
10         c = 3;//локальной переменной  c.
11         //a~5, c~3.
12         int a;//Объявление локальную для данного блока
13         //переменную, перекрывающую уже существующую.
14         //a~мусор, c~3.//!! в локальной a мусор!
15         a = 7;//присвоение локальной переменной a.
16         //a~7 c~3.
17 }
18 //Переменная имеет то значение, которое она имела до
19 //появления локальной переменной с тем же именем:
20 //a~5.
21 //При выходе из блока, переменная c не существует.
22 c=5;//выдаст ошибку!
```

Внутри блока можно создавать свои (локальные) переменные (см. стр. 7), которые не "видны"снаружи блока как до, так и после него (см. стр. 22). Более того, пока

нет *перекрытия видимости*(см. 12), т. е. пока не объявлена локальная переменная с тем же самым именем, можно присвоить значение переменным, (см. 5 и 9) объявленным до начала блока (см. 1). После перекрытия видимости, все операции (присвоение и считывание значения) выполняются именно над текущей локальной переменной (т.е. *видимой*), в тоже время, внешняя переменная по отношению к данному блоку будет оставаться неизменной (скрытой).

При выходе из блока локальные переменные уничтожаются (см. стр. 22). Отмечу, что значение переменной `a` (см. стр. 2) поменялось крайний последний раз, когда ему было присвоено (см. стр. 9) значение до появления локальной переменной с тем же именем (см. стр. 12). Дальнейшие присвоения переменной `a` внутри блока изменяют только локальную переменную, внешняя по отношению к данному блоку так и остается неизменной.

**Вложение блоков** Блоки естественно можно вкладывать друг в друга.

```
1  int a = 2;//Объявление переменной.
2  //a~2.
3  {
4          a = 3;изменили переменную
5          //a~3.
6          int a;//объявили локальную переменную.
7          //a~мусор.
8          a=6;
9          {
10                 //a~6.
11                 a = 5;
12                 //a~5.
13                 int a;
14                 //a~мусор.
15                 a=4;
16                 //a~4.
17         }
18         //a~5.
19 }
20 //a~3.
```

**Итог** Обычно блоки кода используются с определенными языковыми конструкциями языка Си, где нужно заменить одну команду на несколько. Иначе, для отладки, когда нужно выполнить определенный фрагмент кода, который не должен оказывать влияния (перекрытие переменных) на оставшуюся программу.

Задание...

## 1.5 Усложнение действий

Действия условной конструкции могут быть усложнены.

**Больше одного действия** Единичного действия в условной конструкции может не хватить. В условных конструкциях целевые команды можно заменить ранее описанными целыми блоками кода.

```
1  //Продолжим первоначальную программу.
2  int y = 0;
3  if( (2<=x) && (x<11) )
4  {
5          y = x*x;
6          printf("yes\n");
7  }
8  else
9  {
10         printf("no\n");
11 }
12 printf("y ~ %d\n", y);
```

В данном случае в зависимости от условия (см. стр. 3) программа не только печатает фразу (см. стр. 6), но и присваивает некое значение переменной y (см. стр. 5). Так, в случае наступления условия программа присваивает переменной y квадрат значения переменной x. В конце программа в любом случае делает печать итогового значения переменной y. Так, если x равен $-1$, то будет напечатано: "y ~0 а если 5, то – "y ~25".

Отметим, что естественно любой блок команд может состоять из одной единственной команды (см. стр. 10).

**Выход** В качестве одной из команд может выступать и команда return, возвращающая значение текущей функции, в данном случае, являющимся и кодом завершения работы программы.

```
1  //Продолжим первоначальную программу.
2  if( !((2<=x) && (x<11)) )
3  {
4
5          printf("no\n");
```

```
6           return -1;
7   }
8   y = x*x;
9   printf("y ~ %d\n", y);
```

Отмечу, что вложенность блоков не важна, т.е. при выполнении команды `return` выход выполняется в любом случае из текущей функции – той, которой эта серия вложенных блоков принадлежит.

Данный способ стандартен при проверке данных на ввод и корректность. Без конструкции `if-` это бессмысленно, так как другие команды после выполнения команды **return** не выполняются.

## 1.6 Сложный ввод

Рассмотренная ранее функция scanf может одновременно считать несколько чисел и, как функция, как раз возвращает количество успешно считанных аргументов.

```
1  int a, b, c;
2  c = scanf("%d%d", &a, &b);
3  //В случае успешного считывания и a и b => c~2,
4  //Если успешно считано только a => c~1,
5  //Если ни одного => c~0,
6  //В случае ошибки => значение меньше 0, (-1).
```

Отмечу, что считать только значение `b` нельзя, так как переменные считываются последовательно.

Исходя из последнего ввод чисел должен выполняться так:

```
1  int a;//Объявление переменной.
2  printf("enter a:");//приглашение на ввод
3  int ret = scanf("%d", &a);//считываем значение
4  //в переменную ret присваивается количество успешно
5  //считанных чисел. В данном случае -- 1.
6  if( ret != 1 )
7  {
8           printf("Input error!\n");
9           return -1;
10 }
11 else
12 {
13          printf("it is ~ %d\n", (2<=a) && (a<11));
14 }
```

Но, на самом деле в данном случае нет необходимости писать явное ветвление `else`, так как после *return* будет произведен выход из именного блока (функции), а в рассматриваемом нашем случае выход из программы:

```
1  int a;//Объявление переменной.
2  printf("enter a:");//приглашение на ввод
3  int ret = scanf("%d", &a);//считываем значение
4  //в переменную ret присваивается успех.
5  if( ret != 1 )
6  {
7          printf("Input error!\n");
8          return -1;
9  }
10 //явная ветка else не нужна,
11 //как и сами фигурные скобки бывшего блока.
12 printf("it is ~ %d\n", (2<=a) && (a<11));
```

Последний фрагмент кода показывает предпочитаемый способ обработки любых ошибок. В противном случае, код программы будет постоянно съезжать вправо из-за неизбежного вложения блоков. Следует стараться придерживаться отмеченного правила.

Естественно, что мы можем обойтись и без промежуточной переменной ret (см. стр. 3).

```
1  int a;//Объявление переменной.
2  printf("enter a:");//приглашение на ввод
3  if( scanf("%d", &a) != 1 )
4  {
5          printf("Input error!\n");
6          return -1;
7  }
8  if( a < 0 )//Не умеем вычислять квадратный корень
9  {//отрицательных чисел.
10         printf("Negative number entered!\n");
11         return -1;
12 }
13 printf("it is ~ %d\n", (2<=sqrt(a)) && (sqrt(a)<11));
```

## 1.7 Итого

**Выжимка**

**Эталонная программа** Приведем в согласии с данной заметкой фрагмент эталонной программы:

**Контрольные упражнения** Упражнение. Написать полноценную программу, решающую квадратное уравнение. Считаем, что корни действительные, иначе печатаем соответствующее сообщение об ошибке (на подобии см. 8-12).