1.4. Принятие решений. Логические выражения.

Friday, March 20, 2020 23:19

Мы уже успели немного познакомиться с логическими выражениями, когда знакомились с циклами, – они используются в описании условия повторения цикла. В этом уроке мы рассмотрим логические выражения более глубоко и ознакомимся операторами ветвления:

- `if`
- `if-else`
- `else-if`
- `switch-case`
- тернарный оператор

Оператор `if`

В языке С для принятия решений используется оператор `if`. Синтаксис выглядит следующим образом:

```
if  ( logic_expression )
          program statement
```

Например, можно выводить сообщения при выполнении какаого-то условия. Строки

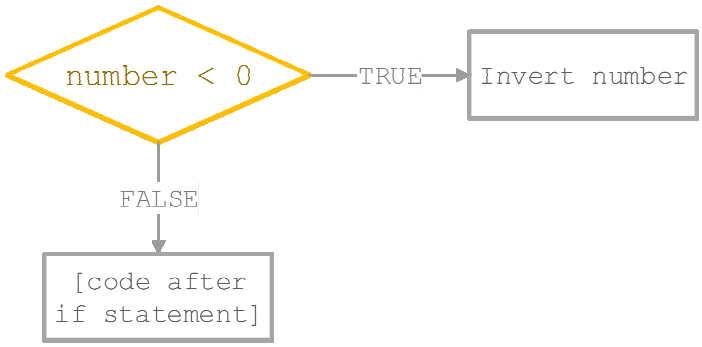
```
if ( count > COUNT_LIMIT )
          printf ("Count limit exceeded\n");
```

выведут сообщение `Count limit exceeded` только когда значение переменной `count` будет больше значения переменной `COUNT_LIMIT`.

Напишем программу, выводящую число по модулю.

```
// Program to calculate the absolute value of an integer

#include <stdio.h>

int main (void)
{
          int  number;

          printf ("Enter your number: ");
          scanf ("%i", &number);

          if (number < 0)
                number = -number;

          printf ("The absolute value is %i\n", number);

          return 0;
}
```

Мы сравниваем число с нулем, и если оно меньше нуля, то инвертируем его знак унарным минусом. Также составим логическую диаграмму, чтобы лучше понимать происходящее и учиться работать с такими диаграммами:



Напишем еще одну программу; которая подсчитает средний балл и покажет количество непроходных баллов.

```c
/* Program to calculate the average of a set of grades
 and count the number of failing test grades        */

#include <stdio.h>

int main (void)
{
          int    numberOfGrades, passingGrade, i, grade;
          int    gradeTotal = 0;
          int    failureCount = 0;
          float  average;

          printf ("How many grades will be entered? ");
          scanf ("%i", &numberOfGrades);
          printf ("Enter the minimum passing grade: ");
          scanf ("%i", &passingGrade);
          printf ("\n");

          for (i = 1; i <= numberOfGrades; i++) {
                    printf ("Enter grade #%i: ", i);
                    scanf ("%i", &grade);
                    gradeTotal += grade;

                    if (grade < passingGrade)
                              failureCount++;
          }

          average = (float) gradeTotal / numberOfGrades;

          printf ("\nAverage grade: %.2f\n", average);
          printf ("Failure count: %i\n", failureCount);

          return 0;
}
```

Здесь при вычислении среднего значения пришлось снова вспомнить об отсечении чисел после запятой при делении целочисленных переменных. Поэтому для получения правильного результата нужно дописать (`float`) перед `gradeTotal`. Параметр `%.2f` выводит число в формате `00.00`, потому что при подсчете оценок нам обычно не нужно больше чисел после запятой.
Будет довольно интересный результат, если ввести 0 при запросе количества оценок. Что при этом произойдет, будет зависеть от того, как система обработает деление на ноль. Windows 8.1 x64 возвращает NaN (Not a Number).

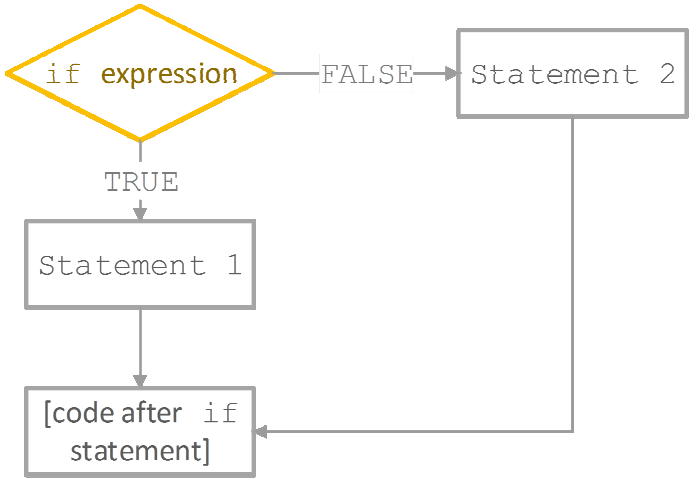
Конструкция `if-else`

Бывает, что нужно выполнить какое-то действие не только когда условие выполняется, но и когда оно не выполняется. Да, можно дважды использовать `if`, но в таком коде будет сложнее разобраться. Гораздо

удобнее использовать конструкцию `if-else`. Синтаксис выглядит следующим образом:

```
if   ( expression )
           program statement 1
else
           program statement 2
```

То есть если выражение возвращает `TRUE`, будет выполнено `statement 1`, а если `FALSE`, то `statement 2`. Логическая диаграмма будет выглядеть так:



Напишем программу, которая будет проверять, четность числа.

```
// Program to determine if a number is even or odd

#include <stdio.h>

int main (void)
{
           int number, remainder;

           printf ("Enter your number: ");
           scanf ("%i", &number);

           remainder = number % 2;

           if (remainder == 0)
                 printf ("The number is even.\n");
           else
                 printf ("The number is odd.\n");

           return 0;
}
```

Не забывать о том, что = это присваивание, а == логический оператор сравнения!

Сложные логические выражения (Compound Relational Tests)

В предыдущих примерах мы всегда пользовались простыми логическими выражениями из одного неравенства. Давайте рассмотрим более сложные выражения с операторами булевой алгебры. Например, выражение

```
if   ( grade > 70   &&   grade < 79 )
```

использует логический оператор `AND` и данное выражение сработает только когда переменная `grade` больше 70 **И** меньше 79. А выражение

```
if   ( index < 0   ||   index > 99 )
```

использует логический оператор `OR` и данное выражение сработает когда переменная `index` меньше 0 **ИЛИ** больше чем 99.

Можно составлять и более сложные выражения, но не рекомендуется их делать слишком длинными,

чтобы код было удобнее читать.

Операторы `&&` и `||` выполняют "сокращенное" вычисление результата, то есть сначала оценивают левый операнд, а потом правый, при этом игнорируя правый, если по результатам левого уже можно сделать вывод об истинности выражения. Например, если i = 0, то в выражении

```
(i != 0) && (j / i > 0)
```

будет сразу возвращен 0 без вычисления правого операнда, потому что i != 0 даст 0. Каким бы ни был правый операнд, результат в любом случае будет 0, поэтому вычислять его нет смысла. И нас это здесь спасает от ошибки деления на 0.

Но будьте осторожны с выражениями вроде

```
i > 0 && ++j > 0
```

потому что инкремент будет пропущен, если левый операнд будет равен 0. Если важно чтобы инкремент выполнялся каждый раз, лучше его поместить слева. А еще лучше вообще выполнять его отдельно.

Напишем программу, которая будет определять, високосный введенный год или нет. Год должен делиться на 4, но если он делится и на 100, то нужно дополнительно проверить, делится ли он на 400. Составим логическое уравнение:

```
if (div_4 == 0 && div_100 != 0 || div_400 ==0)
```

То есть, если год делится на 4 и делится на 100, операция `AND` выдаст `FALSE`, т.к. если год делится на 100, то неравенство `div_100 != 0` выдаст `FALSE`. Но, если год делится на 400, операция `OR` выдаст `TRUE` и условие выполнится.
Напишем программу целиком:

```c
// Program to determine if a year is a leap year

#include <stdio.h>

int main (void)
{
          int year, div_4, div_100, div_400;

          printf ("Enter the year: ");
          scanf ("%i", &year);

          div_4   = year % 4;
          div_100 = year % 100;
          div_400 = year % 400;

          if ((div_4 == 0 && div_100 != 0) || div_400 == 0)
                printf ("This is a leap year.\n");
          else
                printf ("This is not a leap year.\n");

          return 0;
}
```

Кстати, компилятору не понравилось, что `div_4 == 0 && div_100 != 0` не было взято в скобки и он мне посоветовал их поставить. Не знаю почему, но на всякий случай скобки поставил. В интернете пишут, типа компилятор может не понять, в каком порядке производить логические операции, `(a && b) || c` или `a && (b || c)`. Очень странно, но ладно.

Логическая диаграмма:

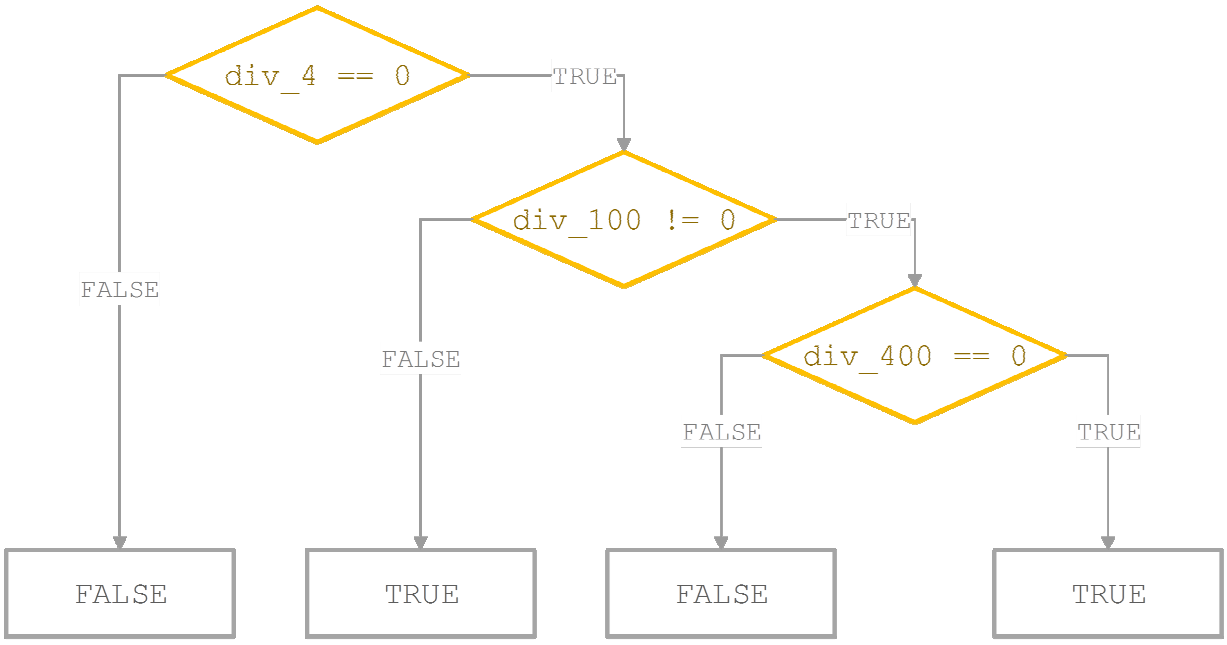


Таблица истинности:

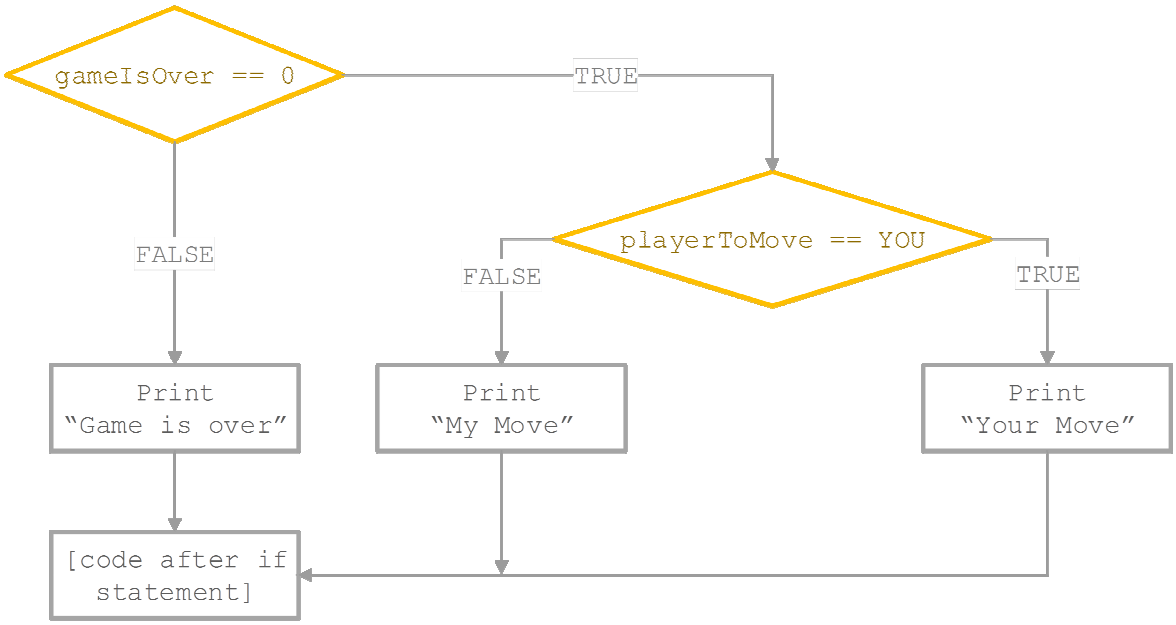
div_4 == 0	div_100 != 0	div_400 == 0	(a && b) || c
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Вложенные конструкции `if`

Нам ничто не мешает каскадировать операторы `if` и `else`. Например, код

```
1 if ( gameIsOver == 0 )
2           if ( playerToMove == YOU )
3                     printf ("Your Move\n");
4           else
5                     printf ("My Move\n");
6 else
7           printf ("The game is over\n");
```

будет проверять переменную `playerToMove` только если переменная `gameIsOver` равна нулю. И после проверки будут выведена строка в зависимости от значения `playerToMove`.
Диаграмма будет выглядеть так:



...надоело составлять диаграммы, хватит.

Проблема "висящего `else`"

Допустим, у нас есть код:

```
1 if (y != 0)
2     if (x != 0)
3         result = x / y;
4 else
5     printf("Error: y is equal to 0\n" );
```

К какому `if` относится `else`? Глядя на отступы, можно подумать, что `else` принадлежит самому первому `if`, но это не так, компилятору отступы вообще безразличны так-то. В языке C `else` относится к самому ближнему `if` без пары `else`, то есть самому нижнему в этом примере. То есть, правильно будет сделать отступ вот так:

```
1 if (y != 0)
2     if (x != 0)
3         result = x / y;
4     else
5         printf ( "Error : y is equal to 0\n");
```

Если нужно, чтобы `else` относился к самому первому `if`, нужно использовать скобки, чтобы второй `if` воспринимался компилятором как отдельное, самостоятельное выражение:

```
1 if (y != 0) {
2     if (x != 0)
3         result = x / y;
4 } else
5     printf("Error : y is equal to 0\n");
```

Этот пример также показывает, что лучше не лениться лишний раз поставить скобки, потому что иногда это может помочь избежать ошибок.

Конструкция `else if`

Эту конструкцию удобно использовать, когда нужно принять небинарное решение (четное / нечетное число, високосный / не високосный год), а множественное решение. Например, рассмотрим программу, которая на вход получает число и должна вывести, больше ли оно или меньше нуля, или равно ему (*по-нормальному это называется функцией определения знака числа*). Конструкия `if-else` здесь не будет работать. Да, можно использовать три отдельных оператора `if`, но такая конструкция далеко не всегда работает, особенно когда проверяемые условия не взаимоисключаемые.

Лучше всего использовать конструкцию `else if`, – дописав оператор if после `else`:

```
if ( expression 1 )
          program statement 1
else
          if ( expression 2 )
                    program statement 2
          else
                    program statement 3
```

которая расширяет количество возможных исходов с двух до трех. Можно еще добавлять операторы `if` после операторов `else`, если нужно. Такая конструкция так часто используется, что ее форматируют иначе, чтоб сразу можно было увидеть, что здесь принимаются множественные решения:

```
if ( expression 1 )
          program statement 1
else if ( expression 2 )
          program statement 2
else
          program statement 3
```

Напишем программу для определения знака числа, о которой мы говорили в начале главы, используя конструкцию `else if`.

```
// Program to implement the sign function

#include <stdio.h>

int main (void)
{
          int number;

          printf ("Enter your number: ");
          scanf ("%i", &number);

          if (number < 0)
                printf ("It's a negative number.\n");
          else if (number == 0)
                printf ("It's a zero.\n");
          else
                printf ("It's a positive number.\n");

          return 0;
}
```

Напишем что-нибудь покруче. Например, программу, определяющую тип символа: буква алфавита, цифра или специальный символ.

```
// Program to categorize a single character

#include <stdio.h>

int main (void)
{
          char c;

          printf ("Enter your character: ");
          scanf ("%c", &c);

          if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
                printf ("It's an alphabetic character.\n");
          else if (c >= '0' && c <= '9')
                printf ("It's a digit.\n");
          else
                printf ("It's a special characher.\n");
```

```
18
19          return 0;
20 }
```

Такую программу можно использовать, например, для реализации ввода математических выражений с клавиатуры: можно распознать, где цифры, где математические операторы и где переменные.
Но лучше помнить, что такое распознавание символов нацелено на работу только с кодировкой ASCII. На практике, для надежной работы в любой кодировке, лучше использовать операторы `islower()` и `isupper()` из библиотеки `ctype.h`.
Также обратите внимание, что при сравнении цифр мы сравниваем не целочисленные 0..9, а символы '0'..'9'. Это большая разница с точки зрения их цифрового представления в программе.

Раз уж мы упомянули реализацию ввода математических выражений, напишем-ка программу, которая будет рассчитывать простейшие выражения из двух чисел и одного оператора.

```
1  // Program to evaluate simple expressions
2
3  #include <stdio.h>
4
5  int main (void)
6  {
7           float value1, value2;
8           char operator;
9
10          printf ("Enter your expression.\n");
11          scanf ("%f %c %f", &value1, &operator, &value2);
12
13          if (operator == '+')
14                printf ("%.2f\n", value1 + value2);
15          else if (operator == '-')
16                printf ("%.2f\n", value1 - value2);
17          else if (operator == '*')
18                printf ("%.2f\n", value1 * value2);
19          else if (operator == '/')
20                if (value2 == 0)
21                      printf ("Division by zero.\n");
22                else
23                      printf ("%g\n", value1 / value2);
24          else
25                printf ("Invalid expression.\n");
26
27          return 0;
28 }
```

Операторы `%f %c %f` в функции `scanf()` написаны через пробел. Это допускает ввод одного пробела между числами и оператором. Именно что допускает, а не делает обязательным. Ввод `2+2` будет работать так же правильно, как и `2 + 2`. Я здесь еще немного забежал наперед и добавил защиту от ввода неправильного оператора и от деления на ноль. Это хорошая практика, когда программист предусматривает возможные ошибочные состояния программы и реализует их обработку.

Оператор `switch`

Мы уже пользовались конструкцией `else if`, при помощи которой мы по очереди сравнивали переменную с несколькими значениями. Специально для этого был создан оператор `switch`.
Синтаксис выглядит следующим образом:

```
1 switch ( expression )
2 {
3           case value1:
4                                 program statement
5                                       ...
6                                 break;
7           case value2:
```

```
                                program statement
                                        ...
                                break;
                        ...
            case valueN:
                                program statement
                                        ...
                                break;
            default:
                                program statement
                                        ...
                                break;
}
```

Здесь оператор `default` срабатывает только если ни один из операторов `case` не сработал. По сути как последний `else` в конструкции `else if`.
Давайте переделаем нашу программу для расчета выражений, используя оператор `switch`.

```
// Program to evaluate simple expressions

#include <stdio.h>

int main (void)
{
            float value1, value2;
            char operator;

            printf ("Enter your expression.\n");
            scanf ("%f %c %f", &value1, &operator, &value2);

            switch (operator)
            {
            case '+':
                  printf ("%.2f\n", value1 + value2);
                  break;
            case '-':
                  printf ("%.2f\n", value1 - value2);
                  break;
            case '*':
            case 'x':
            printf ("%.2f\n", value1 * value2);
            case '/':
                  if (value2 == 0)
                        printf ("Division by zero.\n");
                  else
                        printf ("%g\n", value1 / value2);
                  break;
            default:
                  printf ("Invalid expression.\n");
                  break;
            }

            return 0;
}
```

Обратите внимание, что теперь умножение выполняется не только при помощи оператора `*`, но и при помощи оператора `x`. То есть можно задавать одну и ту же функцию для разных значений переменной, если просто их писать друг за другом в строчку или в столбик.

Наличие `default` совсем не обязательно, если ни одно условие не выполнится, программа просто продолжит работу, ничего не сделав.

Условия также можно писать в строчку, если они должны приводить к одному и тому же действию. Код

```
switch (grade) {
    case 4: case 3: case 2: case 1:
```

```
    3         printf ( "Passing" );
    4         break ;
    5     case 0:  printf ( "Failing" );
    6         break ;
    7     default: printf ( "Illegal grade");
    8         break ;
    9 }
```

эквивалентен коду

```
    1 switch (grade) {
    2     case 4:
    3     case 3:
    4     case 2:
    5     case 1:  printf ( "Passing" );
    6         break ;
    7     case 0:  printf ( "Failing" );
    8         break;
    9     default: printf ( "Illegal grade");
   10         break;
   11 }
```

К сожалению, в C нельзя указывать диапазон возможных значений и каждое значение нужно указывать вручную.

Роль ключевого слова `break`

Как мы уже знаем, слово `break` заставляет программу выйти из цикла или конструкции `switch`. Но почему здесь мы его используем, а в эквивалентных логических выражениях нет?

Дело в том, что `switch` просто "прыгает" (*"computed jump"*) по истинным значениям и продолжит выполнение команд до самого конца, если его не остановить. Рассмотрим код:

```
    1 switch (grade) {
    2     case 4: printf ("Excellent") ;
    3     case 3: printf (" Good ");
    4     case 2: printf ("Average") ;
    5     case 1: printf (" Poor ");
    6     case 0: printf ("Failing" );
    7     default : printf ("Illegal grade");
    8 }
```

Если `grade` = 3, то программа выведет:

```
GoodAveragePoorFailingIllegal grade
```

Не забывайте ставить `break`. Хоть его и пропускают иногда специально, все-таки чаще это ошибка.
Но раз уж упомянули умышленный пропуск `break`, приведем пример такого случая:

```
    1 switch (grade) {
    2     case 4: case 3: case 2: case 1:
    3         num_passing++;
    4         /* FALL THROUGH */
    5     case 0: total_grades++;
    6         break ;
    7 }
```

Комментарий здесь довольно важен, потому что если код будет читать кто-то другой, он может подумать, что здесь `break` был пропущен по ошибке и исправит эту "ошибку", от чего код перестанет работать. Кстати, последний `break` здесь не нужен, но лучше его писать, чтобы о нем не забыть, если будет нужно добавить дополнительные `case`-ы.

Логический тип переменных (Boolean Variables)

Рассмотрим задачу составления списка простых чисел, т.е. таких, которые делятся только на 1 и сами на

себя. Существует множество решений этой задачи, но мы рассмотрим самое просто решение в лоб: для каждого числа `p` проверять делимость на числа в диапазоне от `d = 2` до `p-1`.

```
 1 // Program to generate a table of prime numbers
 2
 3 #include <stdio.h>
 4
 5 int main (void)
 6 {
 7           int p, d;
 8           _Bool isPrime;
 9
10           for (p = 2; p <= 50; p++) {
11                       isPrime = 1;
12
13                       for (d = 2; d < p; d++)
14                               if (p % d == 0)
15                                       isPrime = 0;
16
17                       if (isPrime != 0)
18                                       printf("%i ",p);
19           }
20
21           printf ("\n");
22           return 0;
23 }
```

Здесь сделаны два вложенных цикла, первый цикл выбирает число `p` и задает `isPrime = TRUE`. Второй цикл проверяет, делится ли число `p` на числа, которые меньше него. Если находится число, которое на него делится нацело, то флаг сбрасывается: `isPrime = FALSE`. Если флаг был сброшен, то мы не выводим число `p`, при проверке которого флаг `isPrime` был сброшен.
Давайте теперь вспомним урок 1.2, где мы говорили о том, что значение переменной считается TRUE, если она ненулевая. То есть нам ничто не мешает написать, например

```
1 if (100)
2           printf ("This will always be printed.\n");
```

и оператор `printf()` всегда будет выполняться. То есть можно опустить оператор сравнения и писать просто

```
    if (isPrime)
```

что будет эквивалентно

```
    if (isPrime != 0)
```

Если надо проверить, равна ли переменная нулю (`FALSE`), можно использовать логический оператор `NOT`

```
    if (! isPrime)
```

который инвертирует значение переменной (она станет 0, если была ненулевая и станет 1, если была 0).
Еще один вариант использования оператора `NOT`:

```
    myMove = !myMove;
```

Его также можно применять к выражениям:

```
    ! (x < y)
```

На самом деле это выражение можно было записать как `x >= y`, но я не придумал лучшего примера.

Также из урока 1.2 вспомним, что тип переменной `bool` со значениями `true` и `false` можно подключить из библиотеки `stdbool.h`. Перепишем нашу предыдущую программу с использованием этого типа переменной.

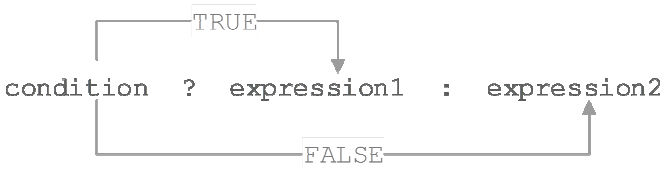
```
// Program to generate a table of prime numbers

#include <stdio.h>
#include <stdbool.h>
```

```c
int main (void)
{
        int p, d;
        bool isPrime;

        for (p = 2; p <= 50; p++) {
                        isPrime = true;

                        for (d = 2; d < p; d++)
                                if (p % d == 0)
                                        isPrime = false;

                        if (isPrime != false)
                                printf ("%i ", p);
        }

        return 0;
}
```

Всё работает. Да, это просто косметические изменения, смысл никак не изменился, просто стало проще читать код.

Тернарный условный оператор (Ternary conditional operator)

Довольно странная штука. Называется тернарным (третичным), потому что имеет три операнда, разделенные знаком вопроса (`?`) и двоеточием (`:`). Синтаксис и логика работы выглядит следующим образом:



То есть, если условие (`condition`) выполняется, то будет выполнено первое выражение (`expression1`), а если не выполняется, то второе выражение (`expression2`). Не знаю, какой садист это придумал, но зачем-то это существует. Аналогичная конструкция через `if-else`:

```c
if (condition)
    expression1;
else
    expression2;
```

Тернарный оператор чаще всего используется, когда нужно присвоить переменной два разных значения в зависимости от некоторого условия. Представим, что у нас есть целочисленные переменные `x` и `s` и нужно присвоить переменной `s` значение `-1`, если `x < 0`, а во всех остальных случаях присвоить ей x^2. Реализация через тернарный оператор будет выглядеть вот так:

```c
s = (x < 0) ? -1 : x * x;
```

Здесь `x < 0` взято в скобки просто для надежности, в простых выражениях не обязательно их использовать.
Приведем еще один пример, где переменной `maxValue` присваивается значение большей из переменных `a` и `b`:

```c
maxValue = (a > b) ? a : b;
```

Как и с конструкциями `else if`, ничто не мешает каскадировать тернарные операторы:

```
sign = ( number < 0 ) ? -1 : (( number == 0 ) ? 0 : 1);
```

Здесь, если `number < 0`, то `sign = -1`, а если `number = 0`, то `sign = 0`, во всех остальных случаях `sign = 1`. Кстати, это реализация определения знака переменной. Одной короткой строкой. Охренеть.

Если что, можно без скобок. Операторы обрабатываются слева направо. То есть выражение

```
e1 ? e2 : e3 ? e4 : e5
```

будет обработано как

```
e1 ? e2 : ( e3 ? e4 : e5 )
```

Не знаю, кому я это пишу, правда, лично мне всегда удобнее лишний раз скобки поставить. Ну, чужой код смогу правильно прочесть по крайней мере.

Также не обязательно чтоб тернарный оператор был частью функции присваивания какой-то переменной. Ничто не мешает его использовать, например, в функции `printf ()`

```
printf ("Sign = %i\n", ( number < 0 ) ? -1 : ( number == 0 ) ? 0 : 1);
```

Здесь нам не пришлось использовать переменную `sign` для хранения знака числа, а мы сразу вывели его в строку.

Источники:

1. Stephen Kochan – Programming in C (4th Edition); chapter 5
2. https://sites.google.com/a/jcu.edu/cs128fall2012/daily-lessons/10-26/boolean-logic-example-leap-year
3. https://ci-plus-plus-snachala.ru/?p=85
4. K.N. King – C Programming: A Modern Approach, 2nd Edition; chapter 5

Упражнения

2. Здесь достаточно простейшей конструкции `else if`.

```
/*  Program to test if number is evenly
    divisible by other number        */
#include <stdio.h>

int main (void)
{
        int num1 = 0, num2 = 0;

        printf ("Enter your numbers, divided by space: ");
        scanf ("%i %i", &num1, &num2);

        if (num2 == 0)
                printf ("Can't divide by zero.\n");
        else if (num1 % num2 == 0)
                printf ("%i is evenly divisible by %i.\n", num1, num2);
        else
                printf ("%i isn't evenly divisible by %i.\n", num1, num2);

        return 0;
}
```

3. Непонятно что тут делает это упражнение, как-то слишком просто.

```
//  Program to divide two integers
```

```c
#include <stdio.h>

int main (void)
{
        int num1, num2;

        printf ("Enter your numbers, divided by space: ");
        scanf ("%i %i", &num1, &num2);

        if (num2 == 0)
                printf ("Can't divide by zero.\n");
        else
                printf ("%i / %i = %.3f\n", num1, num2,
                          (float) num1 / num2);

        return 0;
}
```

4. Не так уж и сложно, как показалось на первый взгляд. Но громоздко.

```c
//  "Logging" calculator

#include <stdio.h>
#include <stdbool.h>

int main (void)
{
        bool  run = true;
        float num, acc = 0;
        char oper;

        printf ("Begin Calculations\n");

        do {
                        scanf ("%f %c", &num, &oper);

                        switch (oper) {

                        case 'S': case 's':
                                acc = num;
                        break;

                        case '+':
                                acc += num;
                        break;

                        case '-':
                                acc -= num;
                        break;

                        case '*':
                                acc *= num;
                        break;

                        case '/':
                                if (num == 0)
                                        printf("Division by zero.\n");
                                else
                                        acc /= num;
                        break;

                        case 'E': case 'e':
                                run = false;
                        break;
```

```
45
46                          default:
47                                  printf("Invalid expression.\n");
48                          break;
49                          }
50
51                          printf("= %g\n", acc);
52                  }
53                  while(run);
54
55                  printf("End of Calculations.\n");
56
57                  return 0;
58  }
```

Исходник:

main

5. Вернемся к упраженению 10 урока 1.3. Очевидно, что нужно проверить знак числа, и если оно отрицательное, вывести минус, инвертировать знак числа и переворачивать как раньше.

```
1  //  Reverse number of any sign
2
3  #include <stdio.h>
4
5  int main (void)
6  {
7          int num;
8
9          printf ("Enter your number: ");
10         scanf ("%i", &num);
11
12         if (num < 0) {
13                 num = -num;
14                 printf("-");
15         }
16
17         do {
18                 printf("%i", num % 10);
19                 num /= 10;
20         }
21         while (num != 0);
22
23         return 0;
24 }
```

6. Эту задачу явно нужно решать через массивы, но я "типа" не умею с ними еще работать. Забавно. Потратив чуть больше времени чем было нужно, я решил сделать следующим образом:
- считаем количество нулей в конце числа, ибо основной алгоритм не будет способен их распознать
- отрезаем от числа цифры, пока не останется одна цифра (число < 10)
- с каждой отрезаемой цифрой домножаем переменную множителя на 10
- оставшуюся от числа цифру выводим текстом в консоль, домножаем на множитель и вычитаем ее из изначального числа
- дописываем столько нулей, сколько насчитали в начале

То есть, например, у числа 5600 мы насчитаем два нуля, а после после первой итерации его обработки останется цифра 5 и множитель будет равен 1000.
И тогда в консоль будет выведено `five`, и от числа 5600 будет вычтено 5 × 1000 = 5000.
Таким образом мы выбросим первую цифру исходного числа. Останется 600, которое отправится на

повторную обработку. Затем выбросим цифру 6. После этого цикл завершится, потому что останутся только нули, которые будут дописаны сразу после завершения цикла ровно столько раз, сколько мы насчитали в начале.
Если же была введена одна цифра, она будет выброшена все равно, т.к. стандартное значение множителя равно 1 и введеное число будет вычтено само из себя, что даст 0, который и является условием завершения программы.

```
1  //  Number to text conversion
2
3  #include <stdio.h>
4
5  int main (void)
6  {
7          int num, buffer, multiplier, zerocount = 0;
8
9          printf ("Enter your number: ");
10         scanf ("%i", &num);
11
12         buffer = num;
13
14         if (buffer != 0)                    // can't divide by zero
15                 while (buffer % 10 == 0) { // count trailing zeros
16                         buffer /= 10;
17                         zerocount ++;
18                 }
19
20         do {
21                 multiplier = 1;
22                 buffer = num;
23                                             // count number of digits
24                 while (buffer > 9) {       // for each iteration
25                         buffer /= 10;
26                         multiplier *= 10;
27                 }
28
29                 num  -= buffer * multiplier;
30
31                 switch (buffer) {
32                 case 0:
33                         printf("zero ");
34                         break;
35                 case 1:
36                         printf("one ");
37                         break;
38                 case 2:
39                         printf("two ");
40                         break;
41                 case 3:
42                         printf("three ");
43                         break;
44                 case 4:
45                         printf("four ");
46                         break;
47                 case 5:
48                         printf("five ");
49                         break;
50                 case 6:
51                         printf("six ");
52                         break;
53                 case 7:
54                         printf("seven ");
55                         break;
56                 case 8:
57                         printf("eight ");
58                         break;
59                 case 9:
```

```
60                          printf("nine ");
61                          break;
62                  }
63          }
64          while(num != 0);
65
66          for ( ; zerocount != 0; zerocount--)
67                  printf("zero ");          // print trailing zeros
68
69          printf("\n");
70
71          return 0;
72  }
```

Исходник:

main

P.S. Анон дал мне задание вывести число по-нормальному, как в синтезаторах речи. Сделал без особого труда. Единственное что, программа поддерживает числа только до 9999. Дальше было лень делать.
Заодно научился работать в дебаггере, потому что долго не понимал, почему цикл `while (buffer > 10)` выходит из цикла при buffer = 10 (подсказка: 10 не больше 10).

Исходник:

main

7. Даже не знаю, как прокомментировать. Просто задача на операторы ветвления. Постарался задействовать максимум выученного.

```
 1  // Quicker prime number list generator
 2
 3  #include <stdio.h>
 4  #include <stdbool.h>
 5
 6  int n, p;
 7  bool isPrime;
 8
 9  int main (void)
10  {
11          for (n = 2; n <= 50; n++) {
12                  if (n != 2 && (n % 2 == 0))
13                          continue;
14
15                  isPrime = false;
16
17                  for (p = 2; p < n; p++) {
18
19                          if (p != 2 && (p % 2 == 0))
20                                  continue;
21                          else if (n % p == 0)
22                                  isPrime = true;
23                  }
24
25                  if (!isPrime)
26                          printf("%i ", n);
27          }
28
29          return 0;
```

```
30 }
```

Еще более оптимальный вариант: зачем проверять делимость на 2, когда можно вообще исключить возможность появления четных чисел.

```
1  // Quicker prime number list generator v1.1
2
3  #include <stdio.h>
4  #include <stdbool.h>
5
6  int n, p;
7  bool isPrime;
8
9  int main (void)
10 {
11         printf("2 ");
12
13         for (n = 3; n <= 50; n += 2) {
14                 isPrime = true;
15
16         for (p = 3; p < n; p += 2)
17                 if (n % p == 0)
18                         isPrime = false;
19
20         if (isPrime)
21                 printf("%i ", n);
22         }
23
24         return 0;
25 }
```

Проекты из K. N. King, chapter 5

0. *Write a single expression whose value is either -1, 0, or +1, depending on whether `i` is less than, equal to or greater than `f`, respectively.*

```
1 #include <stdio.h>
2
3 int main (void)
4 {
5       int i = 3, j = 3, result = 0;
6       result = result - (i < j) + (i > j);
7       printf("%d", result);
8       return 0;
9 }
```

1. Write a program that calculates how many digits a number contains.

```
1  #include <stdio.h>
2
3  int main (void)
4  {
5      int num, result = 0;
6      printf("Enter a number: ");
7      scanf("%3d", &num);
8      result = (num >= 0 && num < 10) ? 1 : (num >= 10 && num < 100) ? 2 :
9                (num >= 100 && num < 1000) ? 3 : 0;
10     printf("The number %d has %d digits", num, result);
11     return 0;
12 }
```

2. Write a program that asks the user for a 24-hour time, then displays the time in 12-hour form.

```c
#include <stdio.h>

int main (void)
{
     int hr24, hr12, min;

     printf("Enter a 24-hour time: ");
     scanf("%2d:%2d", &hr24, &min);

     hr12 = (hr24 == 0) ? 12 : (hr24 < 13) ? hr24 : (hr24 - 12);
     printf("12-hour time: %d:%d %s", hr12, min, (hr24 < 12) ? "AM" : "PM");

     return 0;
}
```