# Control Flow

## Statements and Blocks

**Statement is an expression that is followed by a semicolon (;).**

```
sum = 0;
x++;
printf (…);
```

**Empty statement (;) does not make anything.**

```
sum = 0;;
```

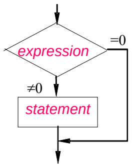
**Compound statement or block is a group of declarations and statements in braces { and } that are syntactically equivalent to a single statement.**

```
{
        declarations of local variables;
        statement
        …
}
```

## Selection Statements

if **statement express decision.**

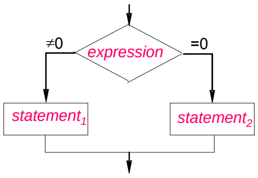
```
if (expression)
   statement
```



**The *expression* is evaluated; if it is true (≠0), *statement* is executed; if it is false (0), the next statement is executed.**

if-else **statement express decision.**

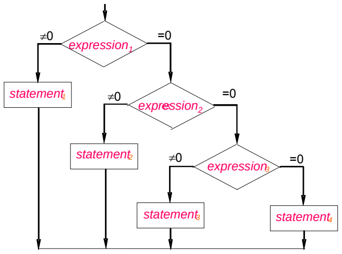
if (*expression*)
   $statement_1$
else
   $statement_2$



**The *expression* is evaluated; if it is true (≠0), *$statement_1$* is executed; if it is false (0), *$statement_2$* is executed instead.**

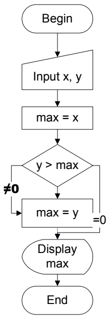
if-else-if **statement express a multi-way decision.**

if ($expression_1$)
   $statement_1$
else if ($expression_2$)
   $statement_2$
…
else
   $statement_n$

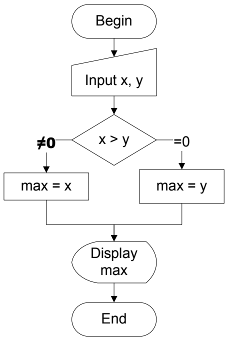


**The *expressions* are evaluated in order; if any *$expression_i$* is true, the *$statement_i$* associated with it is executed, and this terminates the whole chain; the last else part handles the default case where none of the other conditions are satisfied.**

**Exercise: Input two integer numbers and find the maximum number.**



```
/* Maximum number #1 */
#include <stdio.h>
int main()
{
  int x, y, max;
  printf ("Enter x = ");
  scanf ("%d", &x);
  printf ("Enter y = ");
  scanf ("%d", &y);
  max = x;
  if (y > max)
    max = y;
  printf ("Maximum number = %d\n", max);
  return 0;
}
```



```
/* Maximum number #2 */
#include <stdio.h>
int main()
{
  int x, y, max;
  printf ("Enter x = ");
  scanf ("%d", &x);
  printf ("Enter y = ");
  scanf ("%d", &y);
  if (x > y)
    max = x;
  else
    max = y;
  printf ("Maximum number = %d\n", max);
  return 0;
}
```

?: **Ternary operator**

*expression*$_1$ ? *expression*$_2$ : *expression*$_3$

**replaces** if-else **statement**

```
if (expression1)
  expression2
else
  expression3
```

**The** *expression1* **is evaluated; if it is true,** *expression2* **is evaluated and becomes the value of the entire** ?: **expression; if** *expression1* **is false, then** *expression3* **is evaluated and becomes the value of the expression.**

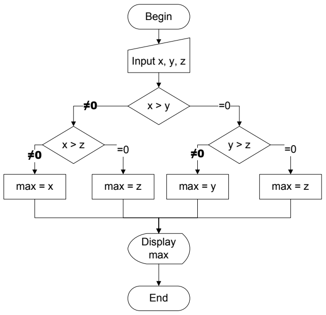
```
max = (x > y) ? x : y;        /* maximum of two numbers */
```

**Example: The program plays the game** "Guess the magic number"**. It prints the message** "Winner!!!" **when the player guess the magic number. If a wrong guess the program provides the player with feedback:** "Too high!!!" **or** "Too low!!!"**.**

**The program generates the magic number using the random number generator** rand()**, which returns an arbitrary number between 0 and the maximum integer value.**

```
/* Game: guess the magic number */
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int number,                  /* magic number           */
       guess;                   /* user's guess           */
   number = rand();             /* generate magic number */
   printf ("Guess the magic number: ");
   scanf ("%d", &guess);
   if (guess == number)
      printf ("Winner!!!\n");
   else if (guess > number)
            printf ("Too high!!!\n");
   else
      printf ("Too low!!!\n");
   return 0;
}
```

**Exercise: Input three real numbers and find the maximum number.**



```
/* Maximum number #3 */
#include <stdio.h>
int main()
{
  float x, y, z, max;
  printf ("Enter x, y, z = ");
  scanf ("%f%f%f", &x, &y, &z);
  if (x > y)
    if (x > z) max = x;
    else max = z;
  else
    if (y > z) max = y;
    else max = z;
  printf ("Maximum number = %f\n", max);
  return 0;
}
```

**Exercise: Write a program that converts a number of points between 0 and 100 into a mark using a table:**

| Number of points | Mark |
|---|---|
| < 50 | 2 |
| 50 ÷ 59 | 3 |
| 60 ÷ 69 | 4 |
| 70 ÷ 79 | 5 |
| ≥ 80 | 6 |

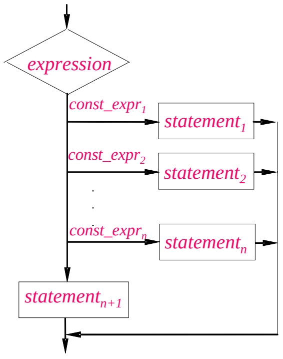
```
if points ≥ 0 and points < 50 then
   mark = 2
else if points < 60 then
   mark = 3
else if points < 70 then
   mark = 4
else if points < 80 then
   mark = 5
else if points ≤ 100 then
   mark = 6
else
   invalid points
```

```
/* Convert points into mark #1*/
#include <stdio.h>
int main()
{
   int points, mark = 0;
   printf ("Enter points = ");
   scanf ("%d", &points);
   if (x >= 0 && x < 50) mark = 2;
   else if (x < 60) mark = 3;
   else if (x < 70) mark = 4;
   else if (x <80) mark = 5;
   else if (x <= 100) mark = 6;
   else printf ("Invalid points!\n");
   if (mark) printf ("Mark = %d\n", mark);
   else printf ("Points are out of range!!!");
   return 0;
}
```

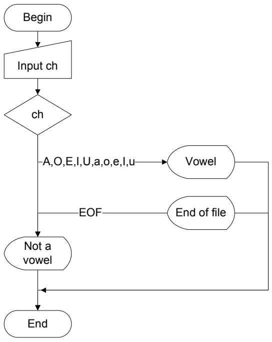
**switch statement is a multi-way decision that tests whether an *expression* matches one of a number of *constant* integer values, and branches accordingly.**

switch (*expression*)
{
  case *constant_expression*$_1$: *statement*$_1$
    [[break;]]
  …
  case *constant_expression*$_n$: *statement*$_n$
    [[break;]]
  [[default: *statement*$_{n+1}$
    [[break;]]]]
}

**If a case matches the *constant_expression*$_i$ value, execution starts with *statement*$_i$ until the break statement. The case labeled default is executed if none of the cases match.**



**Example: The program tests if the input character is a vowel or end of file. (CTRL-Z enters end of file.)**



```
/* Test if the input character is a vowel or end of file. */
#include <stdio.h>
int main ()
{
  int ch;
  printf ("Enter character: ");
  ch = getchar();
  switch(ch)
  {
    case 'A':
    case 'E':
    case 'I':
    case 'O':
    case 'U':
```

```
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':    printf ("Vowel\n");
                 break;
    case EOF: printf ("End of file\n");
                 break;
    default:     printf ("The character is not a vowel!\n");
                 break;
  } /* End of switch */
  return 0;
}
```

**Exercise: Write a program that converts a number of points between 0 and 100 into a mark using a table:**

| **Number of points** | points / 10 | **Mark** |
|---|---|---|
| < 50 | 0<br>1<br>2<br>3<br>4 | 2 |
| 50 ÷ 59 | 5 | 3 |
| 60 ÷ 69 | 6 | 4 |
| 70 ÷ 79 | 7 | 5 |
| ≥ 80 | 8<br>9<br>10 | 6 |

```c
/* Convert points into mark #2*/
#include <stdio.h>
int main()
{
  int points, mark = 0;
  printf ("Enter points = ");
  scanf ("%d", &points);
  switch (points / 10)
  {
    case 0: if (points >= 0) mark = 2;
             break;
    case 1:
    case 2:
    case 3:
    case 4: mark = 2;
             break;
    case 5: mark = 3;
             break;
```

```c
    case 6: mark = 4;
             break;
    case 7: mark = 5;
             break;
    case 8:
    case 9: mark = 6;
             break;
    case 10: if (! (points % 10))
                  mark = 6;
              break;
    default: printf ("Invalid points!\n");
             break;
  }
  if (mark) printf ("Mark = %d\n", mark);
  else printf ("Points are out of range!!!");
  return 0;
}
```

**Exercise: Write a program that prints the solutions to the quadratic equation $ax^2 + bx + c = 0$. Read in** *a*, *b*, *c* **and use the quadratic formula:**

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

**If the discriminant is negative, display the real part and the imaginary part of the complex solution.**

$$ax^2 + bx + c = 0$$

$$a = 0, b = 0 \quad it's\ not\ equation$$

$$a = 0, b \neq 0 \quad linear\ equation\ x = -\frac{c}{b}$$

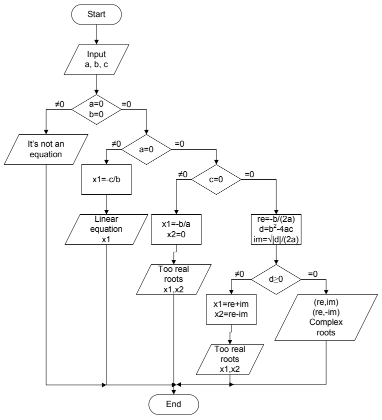
$$a \neq 0, b \neq 0, c = 0 \quad 2\ roots\ x_1 = -\frac{b}{a}, x_2 = 0$$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$d = b^2 - 4ac$$

$$d \geq 0\ 2\ real\ roots$$

$$d < 0\ 2\ complex\ roots$$



```c
/* Quadratic equation */
#include <stdio.h>
#include <math.h>
int main()
{  float a, b, c, re, d, im, x1, x2;
   printf ("Enter a, b, c = ");
   scanf ("%f%f%f", &a, &b, &c);
   if ((a == 0) && (b == 0))
      printf ("It is not an equation\n");
   else if (a == 0)
   {  x1 = -c /b;
      printf ("Linear equation x = %.3f\n", x1)
   }
   else if (c ==0)
   {  x1 = -b / a;
      x2 = 0;
      printf ("Too real roots x1 = %.3f, x2 = %f.3f\n", x1, x2);
   }
```

```
  else
  {
    re = -b / (2 * a);                         /* real part of the root   */
    d = b * b – 2 * a * c;                     /* discriminant            */
    im = sqrt (fabs (d)) / (2 * a);            /* imaginary part          */
    if (d >= 0.0)                              /* positive discriminant */
    {   x1 = re + im;
        x2 = re – im;
        printf ("Too real roots x1 = %.3f, x2 = %f.3f\n", x1, x2);
    }
    else                                       /* negative discriminant */
    {
       printf ("Complex roots x1 = (%.3f, %.3f), x2 = (%.3f, %.3f)/n",
              re, im, re, -im);
    }
  }
  return 0;
}
```

**, Comma operator**

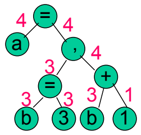
*expression$_1$, expression$_2$, …, expression$_n$*

**The** *expressions* **are evaluated in the given order and the value of the last** *expression$_n$* **becomes the value of the expression.**

```
int a, b;
a = (b=3, b+1);
```

**Result:**

a = 4


=
4
4
a
,
4
3
=
+
3
3
3
1
b
3
b
1