## Control Structures

## *Decision/Selection*

The structure is used for decision making within a program. It allows alternative actions to be taken according to conditions that exist at particular stages within a program.

The structure uses a test condition statement, which upon evaluation by the computer gives rise to certain conditions which may evaluate to a Boolean true or false. Based on the outcome of the test condition, the computer may execute one or more statements.

In reality, a logical test using logical and relational operators may require to be used in order to determine which actions to take (subsequent statements to be executed) depending on the outcome of the test. This is **selection**. For example:

**if (score >= 50)**

**printf(“Pass”);**

**else**

**printf(“Fail”);**

In addition, a group of statements in a program may have to be executed repeatedly until some condition is satisfied. This is known as **looping**. For example, the following code prints digits from 1 to 5.

**for(digit = 1; digit < = 5; digit++)**

**printf(“\n %d”, digit)**

## Selection Structure

### The if statement

The *if* statement provides a junction at which the program has to select which path to follow. The general form is :

*if(expression)*

*statement;*

If *expression* is true (i.e. non zero) , the *statement* is executed, otherwise it is skipped. Normally the expression is a relational expression that compares the magnitude of two quantities ( For example x > y or c = = 6)

Examples

(i) if (x<y)

printf(“x is less that y”);

(ii) if (salary >500)

Tax\_amount = salary \* 1.5;

(iii) if(balance<1000 || status =’R’)

print (“Balance = %f”, balance);

The statement in the if structure can be a single statement or a block (compound statement).

If the statement is a block (of statements), it must be marked off by braces.

if(expression)

{

block of statements;

}

**Example**

if(salary>5000)

{

tax\_amt = salary \*1.5;

printf(“Tax charged is %f”, tax\_amt);

}

### if - else statement

The if else statement lets the programmer choose between two statements as opposed to the simple if statement which gives you the choice of executing a statement (possibly compound) or skipping it.

The general form is:

if (expression)

statement;1

else

statement2;

If expression is true, statement1 is executed. If expression is false, the single statement following the else (statement2) is executed. The statements can be simple or compound.

***Note:*** Indentation is not required but it is a standard style of programming.

##### **Example**:

if(x >=0)

{

printf(“let us increment x:\n”);

x++;

}

else

printf(“x < 0 \n”);

### Multiple Choice: else if (Nested IF)

This is a control structure that is used when more than two choices have to be made. It involves having IF structure inside another IF structure either in the true or false part.

The general form is:

*if (expression 1)*

*statement 1;*

*else if (expression 2)*

statement 2;

*else if (expression 3)*

*statement 3;*

*-------------*

else

*statement n;*

(Braces still apply for block statements) In this structure expression 2 is only tested if condition one is false. Explain how execution of the statements occurs.

**Examples**:

1. A program to call students marks, grade and output the marks and grade.

2. A program to determine the job group of an employee given some scales.

3. A program to prompt for three values a, b and c and determine which is the greatest value.

**Example**

if(sale\_amount>=10000)

Disc= sal\_amt\* 0.10; /\*ten percent/

else if (sal\_amt >= 5000 && sal\_amt < 1000 )

printf (“The discount is %f “,sal\_amt\*0.07 ); /\*seven percent \*/

else if (sal\_amt = 3000 && sal\_amt < 5000)

{

Disc = sal\_amt \* 0.05; /\* five percent \*/

printf ( “ The discount is %f “ , Disc ) ;

}

else

printf ( “ The discount is 0”) ;

**Example : Determining grade category**

#include<stdio.h >

#include<string.h >

main()

{

int marks;

char grade [15];

printf (“ Enter the students marks \n”);

scanf( “%d “,&marks ) ;

if ( marks > =75 && marks <=100)

{

strcpy(grade, “Distinction”); /\* Copy the string to the grade \*/

printf(“The grade is %s” , grade);

}

else if( marks > = 60 && marks < 75 )

{

strcpy(grade, “Credit”);

printf(“The grade is % s” , grade );

}

else if(marks>=50 && marks<60)

{

strcpy(grade, “Pass”);

printf(“The grade is % s” , grade );

}

else if (marks>=0 && marks<50)

{

strcpy(grade, “Fail”);

printf (“The grade is % s” , grade) ;

}

else

printf(“The mark is impossible!” );

return 0;

}

### The ‘switch’ and ‘break’ statements

The *switch - break* statements can be used in place of the *if - else* statements when there are several choices to be made. The switch structure is used to test if a variable equals some constant values then executes the equivalent statement.

The structure of a switch is as follows:

**switch** (integer expression)

{

**case** constant 1:

statement; optional

**case** constant 2:

statement; optional

…………

**default**: (optional)

statement; (optional)

}

***Note:***

1. The switch labels (case labels) must be type int (including char) constants or constant expression.
2. You cannot use a variable for an expression for a label expression.
3. The expressions in the parenthesis should be one with an integer value. (again including type char)

Examples:

1. A program using switch statement that prompts for entry of a number and states the number entered.

**Example: Demonstrating the ‘switch’ structure**

#include<stdio.h>

main()

{

int choice;

printf(“Enter a number of your choice ”);

scanf(“ %d”, &choice);

if (choice >=1 && choice <=9) /\* Range of choice values \*/

switch (choice)

{ /\* Begin of switch\* /

case 1: printf(“\n You typed 1”); break;

case 2: /\* label 2\* /

printf(“\n You typed 2”);

break;

case 3: /\* label 3\* /

printf(“\n You typed 3”);

break;

case 4: /\* label 4\* /

printf( “ \n You typed 4”);

break;

default:

printf(“There is no match in your choice”);

} /\* End of switch\*/

else

printf(“Your choice is out of range”);

return (0);

} /\* End of main\*/

**Explanation**

The expression in the parenthesis following the switch is evaluated. In the example above, it has whatever value we entered as our choice.

Then the program scans a list of labels (case 1, case 2,…. case 4) until it finds one that matches the one that is in parenthesis following the switch statement.

If there is no match, the program moves to the line labeled default, otherwise the program proceeds to the statements following the switch.

The break statement causes the program to break out of the switch and skip to the next statement after the switch. Without the break statement, every statement from the matched label to the end of the switch will be processed.

For example if we remove all the break statements from the program and then run the program using the number 3 we will have the following exchange.

**Enter a number of your choice 3**

**You typed 3**

**You typed 4**

**There is no match in your choice**

**Example: Demonstrating the ‘switch’ structure**

#include <stdio.h>

main()

{

char ch;

printf(“Give me a letter of the alphabet \n”);

printf(“An animal beginning with letter”);

printf (“is displayed \n “);

scanf(“%c”, &ch);

if (ch>=’a’ && ch<=’z’) /\*lowercase letters only \*/

switch (ch)

{ /\*begin of switch\*/

case `a`:

printf(“Alligator , Australian aquatic animal \n”):

break;

case ‘b’:

printf(“Barbirusa, a wild pig of Malaysia \n”);

break;

case ‘c’:

printf(“Coati, baboon like animal \n”);

break;

case ‘d’:

printf(“Desman, aquatic mole-like creature \n”);

break;

default:

printf(“ That is a stumper! \n”)

}

else

printf(“I only recognize lowercase letters.\n”);

return 0;

} /\* End of main \*/

### The ‘continue’ statement

Like the break statement the continue statement is a jump that interrupts the flow of a program. It is used in loops to cause the rest of an iteration to be skipped and the next iteration to be started.

If a break is used in a loop it quits the entire loop.

* **Looping/Iteration/repetition**

Let's consider a situation when you want to print **Hello, World!** five times. Here is a simple C program to do the same −

[Live Demo](http://tpcg.io/g6QM4l)

#include <stdio.h>

int main() {

printf( "Hello, World!\n");

printf( "Hello, World!\n");

printf( "Hello, World!\n");

printf( "Hello, World!\n");

printf( "Hello, World!\n");

}

When the above program is executed, it produces the following result −

Hello, World!

Hello, World!

Hello, World!

Hello, World!

Hello, World!

It was simple, but again, let's consider another situation when you want to write **Hello, World!** a thousand times. We can certainly not write printf() statements a thousand times. Almost all the programming languages provide a concept called **loop**, which helps in executing one or more statements up to a desired number of times. All high-level programming languages provide various forms of loops, which can be used to execute one or more statements repeatedly.

Let's write the above C program with the help of a **while loop** and later, we will discuss how this loop works

[Live Demo](http://tpcg.io/9mkWqV)

#include <stdio.h>

int main() {

int i = 0;

while ( i < 5 ) {

printf( "Hello, World!\n");

i = i + 1;

}

}

When the above program is executed, it produces the following result −

Hello, World!

Hello, World!

Hello, World!

Hello, World!

Hello, World!

The above program makes use of a **while loop**, which is being used to execute a set of programming statements enclosed within {....}. Here, the computer first checks whether the given condition, i.e., variable "a" is less than 5 or not and if it finds the condition is true, then the loop body is entered to execute the given statements. Here, we have the following two statements in the loop body −

* First statement is *printf()* function, which prints Hello World!
* Second statement is *i = i + 1*, which is used to increase the value of variable **i**

After executing all the statements given in the loop body, the computer goes back to while( i < 5) and the given condition, (i < 5), is checked again, and the loop is executed again if the condition holds true. This process repeats till the given condition remains true which means variable "a" has a value less than 5.

To conclude, a loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages −



This tutorial has been designed to present programming's basic concepts to non-programmers, so let's discuss the two most important loops available in C programming language. Once you are clear about these two loops, then you can pick-up C programming tutorial or a reference book and check other loops available in C and the way they work.

**The while Loop**

A **while loop** available in C Programming language has the following syntax −

while ( condition ) {

/\*....while loop body ....\*/

}

The above code can be represented in the form of a flow diagram as shown below −



The following important points are to be noted about a while loop −

* A while loop starts with a keyword **while** followed by a **condition** enclosed in ( ).
* Further to the while() statement, you will have the body of the loop enclosed in curly braces **{...}**.
* A while loop body can have one or more lines of source code to be executed repeatedly.
* If the body of a while loop has just one line, then its optional to use curly braces **{...}**.
* A while loop keeps executing its body till a given **condition** holds true. As soon as the condition becomes false, the while loop comes out and continues executing from the immediate next statement after the while loop body.
* A condition is usually a relational statement, which is evaluated to either true or false. A value equal to zero is treated as false and any non-zero value works like true.

**The do...while Loop**

A while loop checks a given condition before it executes any statements given in the body part. C programming provides another form of loop, called **do...while** that allows to execute a loop body before checking a given condition. It has the following syntax −

do {

/\*....do...while loop body ....\*/

}

while ( condition );

The above code can be represented in the form of a flow diagram as shown below −



If you will write the above example using **do...while** loop, then **Hello, World** will produce the same result −

[Live Demo](http://tpcg.io/xoAxs1)

#include <stdio.h>

int main() {

int i = 0;

do {

printf( "Hello, World!\n");

i = i + 1;

}

while ( i < 5 );

}

When the above program is executed, it produces the following result −

Hello, World!

Hello, World!

Hello, World!

Hello, World!

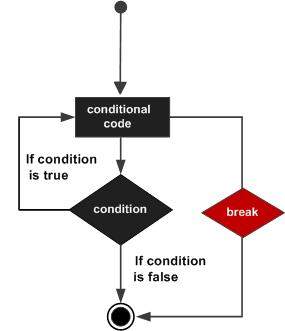
Hello, World!

**The break statement**

When the **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop. The syntax for a **break** statement in C is as follows −

break;

A **break** statement can be represented in the form of a flow diagram as shown below −



Following is a variant of the above program, but it will come out after printing Hello World! only three times −

[Live Demo](http://tpcg.io/UcFo1P)

#include <stdio.h>

int main() {

int i = 0;

do {

printf( "Hello, World!\n");

i = i + 1;

if( i == 3 ) {

break;

}

}

while ( i < 5 );

}

When the above program is executed, it produces the following result −

Hello, World!

Hello, World!

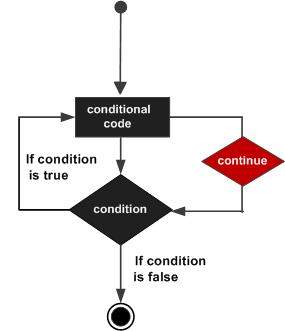
Hello, World!

**The continue statement**

The **continue** statement in C programming language works somewhat like the **break** statement. Instead of forcing termination, **continue** forces the next iteration of the loop to take place, skipping any code in between. The syntax for a **continue** statement in C is as follows −

continue;

A **continue** statement can be represented in the form of a flow diagram as shown below −



Following is a variant of the above program, but it will skip printing when the variable has a value equal to 3 −

[Live Demo](http://tpcg.io/GmJzRF)

#include <stdio.h>

int main() {

int i = 0;

do {

if( i == 3 ) {

i = i + 1;

continue;

}

printf( "Hello, World!\n");

i = i + 1;

}

while ( i < 5 );

}

When the above program is executed, it produces the following result −

Hello, World!

Hello, World!

Hello, World!

Hello, World!