**Decision Construct**

**Introduction**

In a **monolithic program**, the instructions are executed sequentially in the order in which they appear in the program. In real life applications, there are number of situations where one has to change the order of execution of statements based on the conditions.

Decision making statements help the programmer to transfer the control from one part to other parts of the program. These statements determine the flow of program.

In decision control statements (C if else and nested if), group of statements are executed when condition is true.  If condition is false, then else part statements are executed.

We have following types of control statements:

* If statement
* If-else statement
* If-else-if ladder statement
* Switch case statement

Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

## if statement

C uses the keyword if to execute a set of command lines or one command line when logical condition is true. This statement evaluates the test expression inside parenthesis.

If test expression is evaluated to true (nonzero), statements inside the body of if is executed. If test expression is evaluated to false (0), statements inside the body of if are skipped.

The statements following the if statement are normally enclosed within curly braces.

The curly braces indicate the scope of the if statement.

**Syntax:**

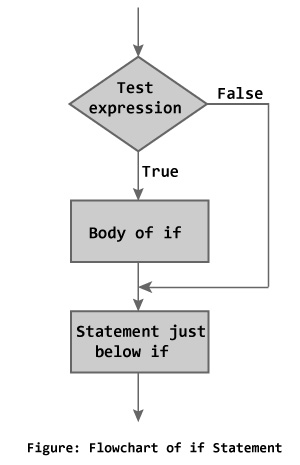
if (testExpression)

{

// statements

}

### Flowchart of if statement



### Example:

#include <stdio.h>

void main()

{

int number;

printf("Enter an integer: ");

scanf("%d", &number);

// Test expression is true if number is less than 0

if (number < 0)

{

printf("You entered negative number”);

}

printf("The if statement is easy");

}

## In the above program is the value entered by user is less than 0 the output would be: You entered negative number The if statement is easy.

## If value entered is greater than 0 then output would be: The if statement is easy.

## if...else statement

In the above if statement, the if block executes only when the condition is true otherwise the next statement in the program is executed. In some cases there is a need to tell the compiler that what should be done if the condition is false.

The if…else statement takes care of true and false condition. It has two blocks. One block is for if when condition is true and other block is for else when condition is false. Else statement cannot be used without if.

The if...else statement executes some code if the test expression is true (nonzero) and some other code if the test expression is false (0).

### Syntax :

if (testExpression) {

// codes inside the body of if

}

else {

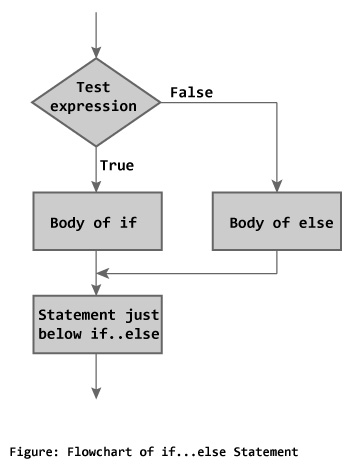
// codes inside the body of else

}

If test expression is true, code inside the body of if statement is executed; and code inside the body of else statement is skipped.

If test expression is false, code inside the body of else statement is executed; and code inside the body of if statement is skipped.

### Flowchart of if...else statement



### Example:

#include <stdio.h>

int main()

{

int number;

printf("Enter an integer: ");

scanf("%d",&number);

if( number<0 )

printf("Number is negative”);

else

printf("Number is positive”);

}

**Output:**

When user enters 7, the test expression ( number%2 == 0 ) is evaluated to false. Hence, the statement inside the body of else statement printf("%d is an odd integer"); is executed and the statement inside the body of if is skipped.

## Nested if...else statement (if...elseif....else Statement)

The if...else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.

**Nested if…else statement/if-else-if Ladder**

The nested if...else statement allows you to check for multiple test expressions and execute different codes for more than two conditions. Here, if the first condition is true the compiler executes the block followed by first if condition, otherwise it skips that block and checks for next logical condition followed by else-if, if the condition is true the block of statements followed by that if condition is executed. The process is continued until a true condition is occurred or an else block is occurred. If all if conditions become false, it executes the else block.

### Syntax:

if (testExpression1)

{

// statements to be executed if testExpression1 is true

}

else if(testExpression2)

{

// statements to be executed if testExpression1 is false and testExpression2 is true

}

else if (testExpression 3)

{

// statements to be executed if testExpression1 and testExpression2 is false and testExpression3 is true

}

.

.

else

{

// statements to be executed if all test expressions are false

}

### Example:

// Program to relate two integers using =, > or <

#include <stdio.h>

void main()

{

int number1, number2;

printf("Enter two integers: ");

scanf("%d %d", &number1, &number2);

//checks if two integers are equal.

if(number1 == number2)

{

printf("Result: %d = %d",number1,number2);

}

//checks if number1 is greater than number2.

else if (number1 > number2)

{

printf("Result: %d > %d", number1, number2);

}

// if both test expression is false

else

{

printf("Result: %d < %d",number1, number2);

}

}

**Output:**

Enter two integers: 12

23

Result: 12 < 23

You can also use switch statement to make decision between multiple possibilites.

The nested if...else statement allows you to execute a block code among many alternatives. If you are checking on the value of a single variable in nested if...else statement, it is better to use switch statement.

**Switch Statement**

The switch statement is a multi-way branch statement. The switch statement requires only one argument of any data type, which is checked with the number of case options. If the value matches with the case constant, this particular case statement is executed. If not, default is executed.

Every case statement is terminated with ‘:’. The break statement is used to exit from current case structures.

The switch statement is often faster than nested if...else (not always). Also, the syntax of switch statement is cleaner and easy to understand.

Switch statement is mostly used for writing menu driven programs.

## Syntax:

## switch (n)

​{

case constant1:

// code to be executed if n is equal to constant1;

break;

case constant2:

// code to be executed if n is equal to constant2;

break;

.

.

.

default:

// code to be executed if n doesn't match any constant

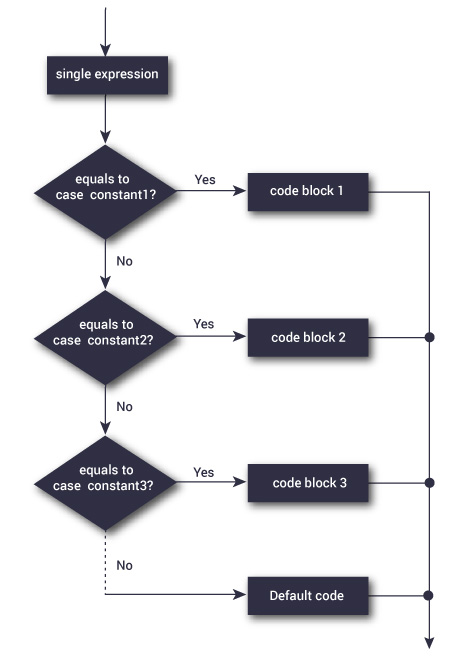
}

When a case constant is found that matches the switch expression, control of the program passes to the block of code associated with that case.

In the above pseudo code, suppose the value of n is equal to constant2. The compiler will execute the block of code associate with the case statement until the end of switch block, or until the break statement is encountered.

The break statement is used to prevent the code running into the next case.

### Switch Statement Flowchart



### Example:

// Program to create a simple calculator

// Performs addition, subtraction, multiplication or division depending the input from user

# include <stdio.h>

void main() {

char operator;

double firstNumber,secondNumber;

printf("Enter an operator (+, -, \*,): ");

scanf("%c", &operator);

printf("Enter two operands: ");

scanf("%lf %lf",&firstNumber, &secondNumber);

switch(operator)

{

case '+':

printf("%.1lf + %.1lf = %.1lf",firstNumber, secondNumber, firstNumber+secondNumber);

break;

case '-':

printf("%.1lf - %.1lf = %.1lf",firstNumber, secondNumber, firstNumber-secondNumber);

break;

case '\*':

printf("%.1lf \* %.1lf = %.1lf",firstNumber, secondNumber, firstNumber\*secondNumber);

break;

case '/':

printf("%.1lf / %.1lf = %.1lf",firstNumber, secondNumber, firstNumber/firstNumber);

break;

// operator is doesn't match any case constant (+, -, \*, /)

default:

printf("Error! operator is not correct");

}

}

**Output:**

Enter an operator (+, -, \*,): -

Enter two operands: 32.5

12.4

32.5 - 12.4 = 20.1

The - operator entered by the user is stored in operator variable. And, the two operands, 32.5 and 12.4 are stored in variable firstNumber and secondNumber respectively.

Then, control of the program jumps to

printf("%.1lf / %.1lf = %.1lf",firstNumber, secondNumber, firstNumber/firstNumber);

Finally, the break; statement ends the switch statement.

**Summary:**

|  |  |  |
| --- | --- | --- |
| **Decision control statements** | **Syntax** | **Description** |
| **if** | if (condition)  { Statements; } | In these type of statements, if condition is true, then respective block of code is executed. |
| **if…else** | if (condition)  { Statement1; Statement2; }  else  { Statement3; Statement4; } | In these type of statements, group of statements are executed when condition is true.  If condition is false, then else part statements are executed. |
| **nested if** | if (condition1){ Statement1; } else\_if(condition2)  { Statement2; }  else Statement 3; | If condition 1 is false, then condition 2 is checked and statements are executed if it is true. If condition 2 also gets failure, then else part is executed. |

**Loops**

You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

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 −



C programming language provides the following types of loops to handle looping requirements. In other words, loops are used in programming to repeat a specific block until some end condition is met. There are three loops in C programming:

1. for loop
2. while loop
3. do...while loop

**Overview:**

|  |  |
| --- | --- |
| **S.N.** | **Loop Type & Description** |
| 1 | [while loop](http://www.tutorialspoint.com/cprogramming/c_while_loop.htm)  Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body. |
| 2 | [for loop](http://www.tutorialspoint.com/cprogramming/c_for_loop.htm)  Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| 3 | [do...while loop](http://www.tutorialspoint.com/cprogramming/c_do_while_loop.htm)  It is more like a while statement, except that it tests the condition at the end of the loop body. |
| 4 | [nested loops](http://www.tutorialspoint.com/cprogramming/c_nested_loops.htm)  You can use one or more loops inside any other while, for, or do..while loop. |

## for Loop

 The “for loop” loops from one number to another number and increases by a specified value each time.

**Syntax:**

for (initializationStatement; testExpression; updateStatement)

{

// codes

}

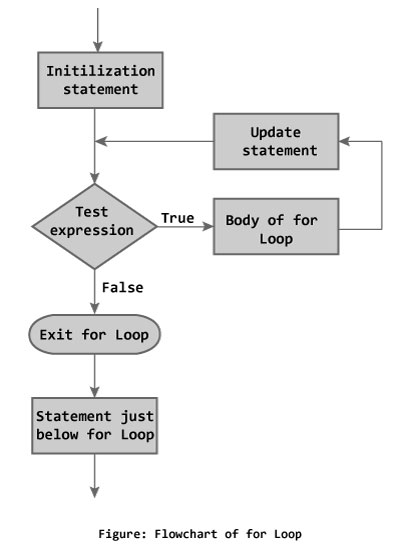
### How for loop works?

The initialization statement is executed only once.

Then, the test expression is evaluated. If the test expression is false (0), for loop is terminated. But if the test expression is true (nonzero), codes inside the body of for loop is executed and the update expression is updated. This process repeats until the test expression is false.

The for loop is commonly used when the number of iterations is known.

### for loop Flowchart



### Example:

// Program to calculate the sum of first n natural numbers

// Positive integers 1,2,3...n are known as natural numbers

#include <stdio.h>

void main()

{

int n, count, sum = 0;

printf("Enter a positive integer: ");

scanf("%d", &n);

// for loop terminates when n is less than count

for(count = 1; count <= n; ++count)

{

sum += count;

}

printf("Sum = %d", sum);

}

**Output:**

Enter a positive integer: 10

Sum = 55

The value entered by the user is stored in variable n. Suppose the user entered 10.

The count is initialized to 1 and the test expression is evaluated. Since, the test expression count <= n (1 less than or equal to 10) is true, the body of for loop is executed and the value of sum will be equal to 1.

Then, the update statement ++count is executed and count will be equal to 2. Again, the test expression is evaluated. The test expression is evaluated to true and the body of for loop is executed and the sum will be equal to 3. And, this process goes on.

Eventually, the count is increased to 11. When the count is 11, the test expression is evaluated to 0 (false) and the loop terminates.

## while loop

A while loop in C programming repeatedly executes a target statement as long as a given condition is true. The syntax of a while loop is as follows.

**Syntax:**

initialization

while (testExpression)

{

//codes

Increment/decrement

}

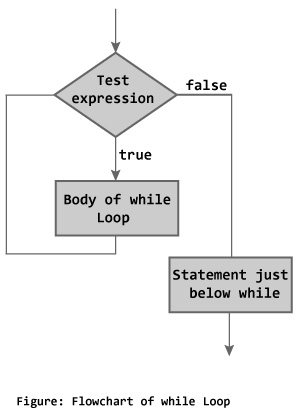
### How while loop works?

The while loop evaluates the test expression.

If the test expression is true (nonzero), codes inside the body of while loop is evaluated. Then, again the test expression is evaluated. The process goes on until the test expression is false.

When the test expression is false, the while loop is terminated.

### Flowchart of while loop



### Example :

// Program to find factorial of a number

// For a positive integer n, factorial = 1\*2\*3...n

#include <stdio.h>

void main()

{

int number;

long long factorial;

printf("Enter an integer: ");

scanf("%d",&number);

factorial = 1;

// loop terminates when number is less than or equal to 0

while (number > 0)

{

factorial \*= number; // factorial = factorial\*number;

--number;

}

printf("Factorial= %ld", factorial);

}

**Output:**

Enter an integer: 5

Factorial = 120

## do...while loop

The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed once, before checking the test expression. Hence, the do...while loop is executed at least once.

### Syntax:

initialization

do

{

// codes

Increment/decrement

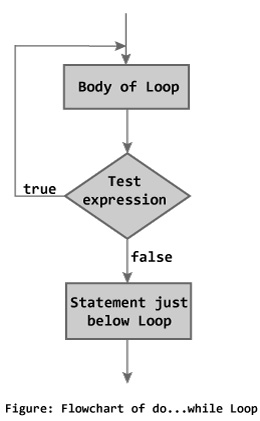
}while (testExpression);

### How do...while loop works?

The code block (loop body) inside the braces is executed once.

Then, the test expression is evaluated. If the test expression is true, the loop body is executed again. This process goes on until the test expression is evaluated to 0 (false).

When the test expression is false (nonzero), the do...while loop is terminated.



### Example:

// Program to add numbers until user enters zero

#include <stdio.h>

void main()

{

float number, sum = 0;

// loop body is executed at least once

do

{

printf("Enter a number: ");

scanf("%f", &number);

sum += number;

}

while(number != 0.0);

printf("Sum = %f",sum);

return 0;

}

**Output:**

Enter a number: 1.5

Enter a number: 2.4

Enter a number: -3.4

Enter a number: 4.2

Enter a number: 0

Sum = 4.70

**The Infinite Loop**

A loop becomes an infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose. Since none of the three expressions that form the 'for' loop are required, you can make an endless loop by leaving the conditional expression empty.

**Example:**

#include <stdio.h>

int main () {

for( ; ; ) {

printf("This loop will run forever.\n");

}

return 0;

}

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but C programmers more commonly use the for(;;) construct to signify an infinite loop.

NOTE − You can terminate an infinite loop by pressing Ctrl + Break keys.

**while Vs do while loop**

|  |  |
| --- | --- |
| **While loop** | **Do while loop** |
| Condition is checked in the starting. | Condition is checked in the end. |
| This loop is also called entry controlled loop | This loop is also called as exit controlled loop. |
| If the condition is false, the loop will not execute at all. | Loop will run at least once, even if the condition is false. |
| Syntax:  initialization  while(condition)  {  …  Statements  …  Increment/Decrement  } | Syntax:  initialization  do  {  …  Statements  …  Increment/Decrement  }while(condition); |

**Loop Control Statements**

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

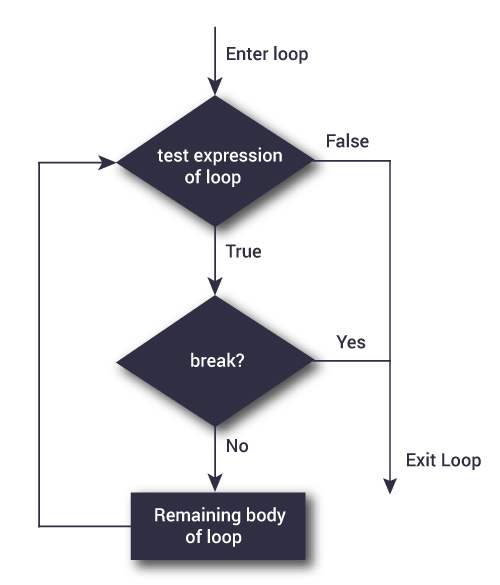
C supports the following control statements.’

|  |  |
| --- | --- |
| **S.N.** | **Control Statement & Description** |
| 1 | **break statement**  Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch. |
| 2 | **continue statement**  Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |
| 3 | [**goto statement**](http://www.tutorialspoint.com/cprogramming/c_goto_statement.htm)  Transfers control to the labeled statement. |

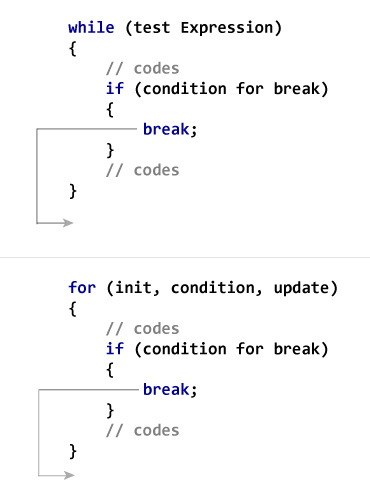
**break statement**

The break statement allows you to exit a loop from any point within its body, bypassing its normal termination expression. When the break statement is encountered inside a loop, the loop is immediately terminated, and program control resumes at the next statement following the loop. The break statement can be used with all three of C's loops. You can have as many statements within a loop as you desire. It is generally best to use the break for special purposes, not as your normal loop exit. break is also used in conjunction with functions and case statements which will be covered in later sections.

### Flowchart of break statement



### How break statement works?



**Example:**

void main()

{

int i;

for(i=1;i<=10;i++)

{

if(i==5)

break;

else

printf(“\n%d”,i);

}

**Output:**

1

2

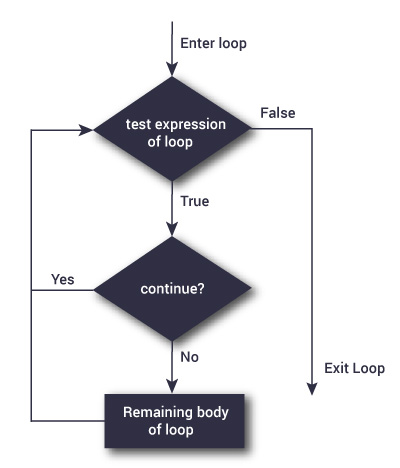
3

4

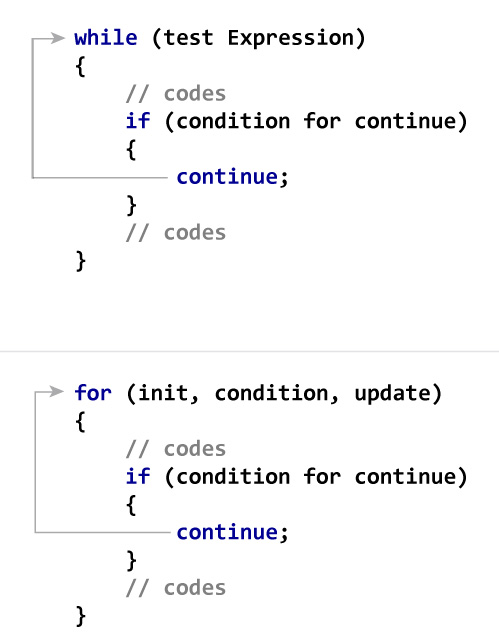
**continue statement**

The continue statement is somewhat the opposite of the break statement. It forces the next iteration of the loop to take place, skipping any code in between itself end of the loop. In while and do-while loops, a *continue* statement will cause control to go directly to the test condition and then continue the looping process. In the case of the for loop, the increment part of the loop continues. One good use of continue is to restart a statement sequence when an error occurs.

### Flowchart of continue Statement



### How continue statement works?



**Example:**

void main()

{

int i;

for(i=1;i<=10;i++)

{

if(i==5)

continue;

else

printf(“\n%d”,i);

}

**Output:**

1

2

3

4

6

7

8

9

10

**Break Vs Continue**

|  |  |
| --- | --- |
| **Break** | **Continue** |
| Break statement is used to transfer the control of the program outside loop or switch case. | Continue statement is used to skip some statement of the loop and move to the next iteration in the loop. |
| Break statement can appear in loops and in switch case. | Continue statement is only used in loops. |
| It terminates the loop. | It do note terminate the loop. |

**goto statement**

A goto statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

NOTE − Use of goto statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.

**Syntax:**

The syntax for a goto statement in C is as follows −

goto label;

..

.

label: statement;

Here **label** can be any plain text except C keyword and it can be set anywhere in the C program above or below to **goto** statement.

goto statements is used to transfer the normal flow of a program to the specified label in the program.

Below is the syntax for goto statement in C.

{

…….

go to label;

…….

…….

LABEL:

statements;

}

**Example:**

#include <stdio.h>

int main()

{

int i;

for(i=0;i<10;i++)

{

if(i==5)

{

printf("\nWe are using goto statement when i = 5");

goto HAI;

}

printf("%d ",i);

}

HAI : printf("\nNow, we are inside label name \"hai\" \n");

}

**Output:**

0 1 2 3 4

We are using goto statement when i = 5

Now, we are inside label name “hai”