CONCEPT OF PROGRAMMING USING C

Unit 3

TOPIC :CONTROL STRUCTURES

By

Navneet kumar solanki

Lecturer-IT

CONTENTS:-

INTRODUCTION:-

A **control structure** is like a block of programming that analyses variables and chooses a direction in which to go based on given parameters. The term flow **control** details the direction the program takes (which way program **control** "flows"). Hence it is the basic decision-making process in computing; It is a prediction.

The three basic types of control structures are sequential, selection and iteration. They can be combined in any way to solve a specified problem.

Sequential is the default control structure, statements are executed line by line in the order in which they appear. The selection structure is used to test a condition. A sequence of statements is executed depending on whether or not the condition it true or false. This means the program chooses between two or more alternative paths. Condition refers to any expression or value that returns a Boolean value, meaning true or false.

The three main types of selection statements are "if," "if/else" and "switch" statements. The most basic and common is the "if" statement. The "if" and "if/else" statements can be nested. Switch statements are ideally used when there are multiple cases to choose from.

The iteration or repetition structure repeatedly executes a series of statements as long as the condition is true. The condition may be predefined or open-ended. "While," "do/while" and "for" loop are the three types of iteration statements. A loop can either be event controlled or counter controlled. An event-controlled loop executes a sequence of statements till and event occurs while a counter-controlled loop executes the statements a predetermined number of times.

DECISION MAKING WITH IF STATEMENT

What is a Conditional Statement?

In a 'C' program are executed sequentially. This happens when there is no condition around the statements. If you put some condition for a block of statements the flow of execution might change based on the result evaluated by the condition. This process is referred to as decision making in 'C.' The decision-making statements are also called as control statements.

In 'C' programming conditional statements are possible with the help of the following two constructs:

1. If statement

2. If-else statement

It is also called as branching as a program decides which statement to execute based on the result of the evaluated condition.

If statement

It is one of the powerful conditional statement. If statement is responsible for modifying the flow of execution of a program. If statement is always used with a condition. The condition is evaluated first before executing any statement inside the body of If. The syntax for if statement is as follows:

```
if (condition)
     instruction;
```

The condition evaluates to either true or false. True is always a non-zero value, and false is a value that contains zero. Instructions can be a single instruction or a code block enclosed by curly braces { }.

Following program illustrates the use of if construct in 'C' programming:

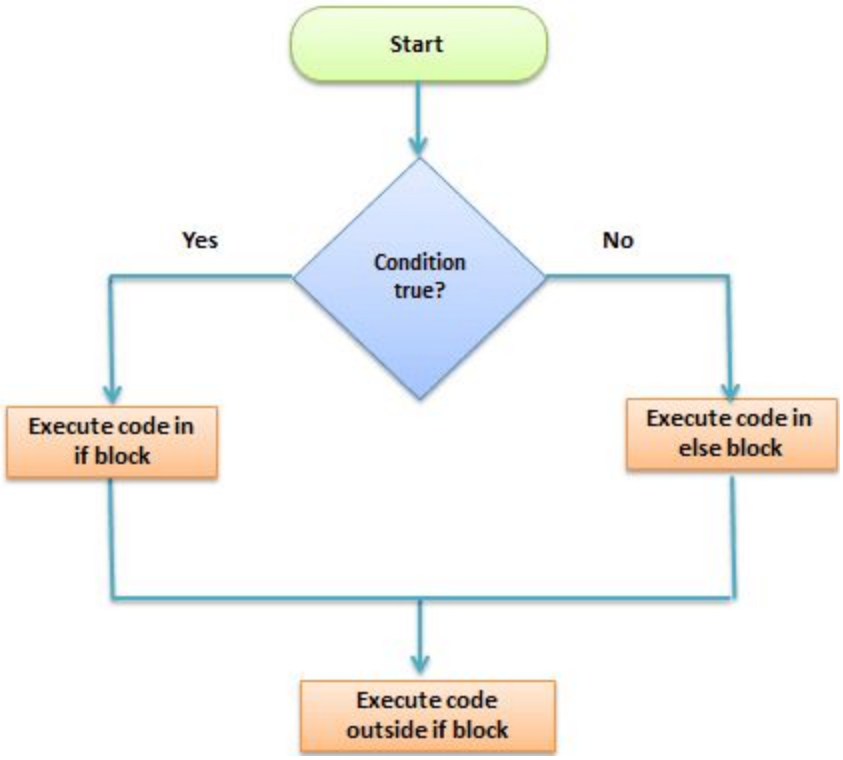
```
#include<stdio.h>
int main()
{
     int num1=1;
     int num2=2;
     if(num1<num2)        //test-condition
```

```
	{
		printf("num1 is smaller than num2");
	}
	return 0;
}
```

Output:

```
num1 is smaller than num2
```

The If-Else statement



The if-else is statement is an extended version of If. The general form of if-else is as follows:

```
if (test-expression)
{
```

```
    True block of statements
}
Else
{
    False block of statements
}
Statements;
```

In this type of a construct, if the value of test-expression is true, then the true block of statements will be executed. If the value of test-expression if false, then the false block of statements will be executed. In any case, after the execution, the control will be automatically transferred to the statements appearing outside the block of If.

Following programs illustrate the use of the if-else construct:

We will initialize a variable with some value and write a program to determine if the value is less than ten or greater than ten.

Let's start.

```
#include<stdio.h>
int main()
{
     int num=19;
     if(num<10)
     {
          printf("The value is less than 10");
     }
     else
     {
          printf("The value is greater than 10");
     }
     return 0;
}
```

Output:

```
The value is greater than 10
```

```c
#include<stdio.h>
int main()
{
    int num=19;
    if(num<10)
    {
        printf("The value is less than 10");
    }
    else
    {
        printf("The value is greater than 10");
    }
    return 0;
}
```

1. We have initialized a variable with value 19. We have to find out whether the number is bigger or smaller than 10 using a 'C' program. To do this, we have used the if-else construct.
2. Here we have provided a condition num<10 because we have to compare our value with 10.
3. As you can see the first block is always a true block which means, if the value of test-expression is true then the first block which is If, will be executed.
4. The second block is an else block. This block contains the statements which will be executed if the value of the test-expression becomes false. In our program, the value of num is greater than ten hence the test-condition becomes false and else block is executed. Thus, our output will be from an else block which is "The value is greater than 10". After the if-else, the program will terminate with a successful result.

Nested If-else Statements

When a series of decision is required, nested if-else is used. Nesting means using one if-else construct within another one.

Let's write a program to illustrate the use of nested if-else.

```
#include<stdio.h>
int main()
{
      int num=1;
      if(num<10)
      {
            if(num==1)
            {
                  printf("The value is:%d\n",num);
            }
            else
            {
                  printf("The value is greater than 1");
            }
      }
      else
      {
            printf("The value is greater than 10");
      }
      return 0;
}
```

Output:

```
The value is:1
```

The above program checks if a number is less or greater than 10 and prints the result using nested if-else construct.

```c
#include<stdio.h>
int main()
{
    int num=1;
    if(num<10){
        if(num==1){
            printf("The value is:%d\n",num);
        }
        else{
            printf("The value is greater than 1");
        }
    }
    else{
        printf("The value is greater than 10");
    }
    return 0;
}
```

1. Firstly, we have declared a variable num with value as 1. Then we have used if-else construct.
2. In the outer if-else, the condition provided checks if a number is less than 10. If the condition is true then and only then it will execute the inner loop. In this case, the condition is true hence the inner block is processed.
3. In the inner block, we again have a condition that checks if our variable contains the value 1 or not. When a condition is true, then it will process the If block otherwise it will process an else block. In this case, the condition is true hence the If a block is executed and the value is printed on the output screen.
4. The above program will print the value of a variable and exit with success.

Nested Else-if statements(LADDER IF ELSE)

Nested else-if is used when multipath decisions are required.

The general syntax of how else-if ladders are constructed in 'C' programming is as follows:

```
if (test - expression 1) {
    statement1;
} else if (test - expression 2) {
    Statement2;
} else if (test - expression 3) {
    Statement3;
} else if (test - expression n) {
    Statement n;
} else {
    default;
}
Statement x;
```

This type of structure is known as the else-if ladder. This chain generally looks like a ladder hence it is also called as an else-if ladder. The test-expressions are evaluated from top to bottom. Whenever a true test-expression if found, statement associated with it is executed. When all the n test-expressions becomes false, then the default else statement is executed.

Let us see the actual working with the help of a program.

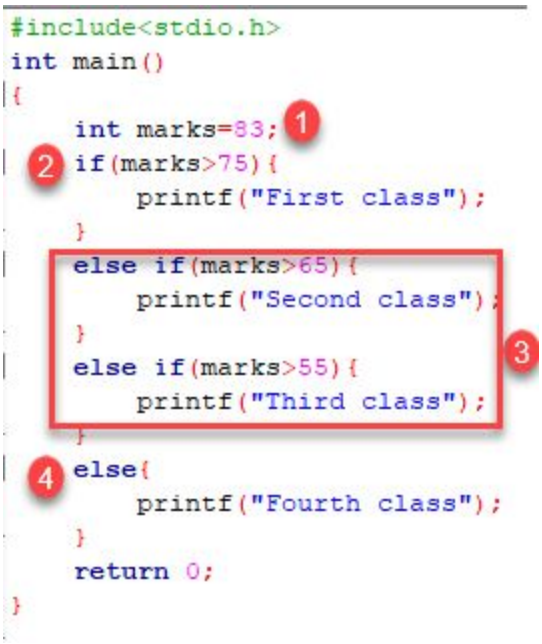
```
#include<stdio.h>
int main()
{
     int marks=83;
     if(marks>75){
          printf("First class");
     }
     else if(marks>65){
          printf("Second class");
     }
     else if(marks>55){
          printf("Third class");
     }
     else{
```

```
            printf("Fourth class");
        }
        return 0;
}
```

Output:

```
First class
```

The above program prints the grade as per the marks scored in a test. We have used the else-if ladder construct in the above program.



1. We have initialized a variable with marks. In the else-if ladder structure, we have provided various conditions.
2. The value from the variable marks will be compared with the first condition since it is true the statement associated with it will be printed on the output screen.
3. If the first test condition turns out false, then it is compared with the second condition.
4. This process will go on until the all expression is evaluated otherwise control will go out of the else-if ladder, and default statement will be printed.

LOOPS

What are Loops?

In looping, a program executes the sequence of statements many times until the stated condition becomes false. A loop consists of two parts, a body of a loop and a control statement. The control statement is a combination of some conditions that direct the body of the loop to execute until the specified condition becomes false.

Types of Loops

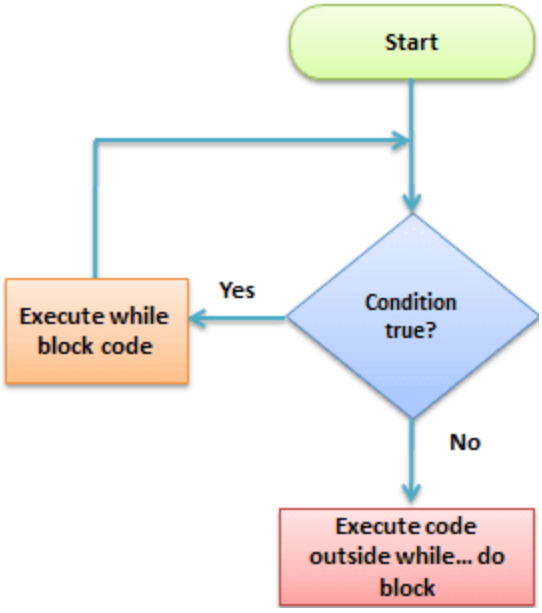
Depending upon the position of a control statement in a program, a loop is classified into two types:

1. Entry controlled loop

2. Exit controlled loop

In an **entry controlled loop,** a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.

In an **exit controlled loop**, a condition is checked after executing the body of a loop. It is also called as a post-checking loop.



Sample Loop

The control conditions must be well defined and specified otherwise the loop will execute an infinite number of times. The loop that does not stop executing and processes the statements number of times is called as an **infinite loop**. An infinite loop is also called as an "**Endless loop**." Following are some characteristics of an infinite loop:

1. No termination condition is specified.

2. The specified conditions never meet.

The specified condition determines whether to execute the loop body or not.

'C' programming language provides us with three types of loop constructs:

1. The while loop

2. The do-while loop

3. The for loOP

WHILE Loop

A while loop is the most straightforward looping structure. The basic format of while loop is as follows:

```
while (condition) {

                statements;

}
```

It is an entry-controlled loop. In while loop, a condition is evaluated before processing a body of the loop. If a condition is true then and only then the body of a loop is executed. After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed until the condition becomes false. Once the condition becomes false, the control goes out of the loop.

After exiting the loop, the control goes to the statements which are immediately after the loop. The body of a loop can contain more than one statement. If it contains only one statement, then the curly braces are not compulsory. It is a good practice though to use the curly braces even we have a single statement in the body.

In while loop, if the condition is not true, then the body of a loop will not be executed, not even once. It is different in do while loop which we will see shortly.

Following program illustrates a while loop:

```
#include<stdio.h>

#include<conio.h>

int main()

{

     int num=1;    //initializing the variable

     while(num<=10)    //while loop with condition

     {

          printf("%d\n",num);

          num++;         //incrementing operation

     }

     return 0;

}
```

Output:

```
1

2

3

4
```

```
5

6

7

8

9

10
```

The above program illustrates the use of while loop. In the above program, we have printed series of numbers from 1 to 10 using a while loop.

1. We have initialized a variable called num with value 1. We are going to print from 1 to 10 hence the variable is initialized with value 1. If you want to print from 0, then assign the value 0 during initialization.
2. In a while loop, we have provided a condition (num<=10), which means the loop will execute the body until the value of num becomes 10. After that, the loop will be terminated, and control will fall outside the loop.
3. In the body of a loop, we have a print function to print our number and an increment operation to increment the value per execution of a loop.

An initial value of num is 1, after the execution, it will become 2, and during the next execution, it will become 3. This process will continue until the value becomes 10 and then it will print the series on console and terminate the loop.

\n is used for formatting purposes which means the value will be printed on a new line.

Do-While loop

A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.

The basic format of while loop is as follows:

```
do {

  statements

} while (expression);
```

As we saw in a while loop, the body is executed if and only if the condition is true. In some cases, we have to execute a body of the loop at least once even if the condition is false. This type of operation can be achieved by using a do-while loop.

In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.

Similar to the while loop, once the control goes out of the loop the statements which are immediately after the loop is executed.

The critical difference between the while and do-while loop is that in while loop the while is written at the beginning. In do-while loop, the while condition is written at the end and terminates with a semi-colon (;)

The following program illustrates the working of a do-while loop:

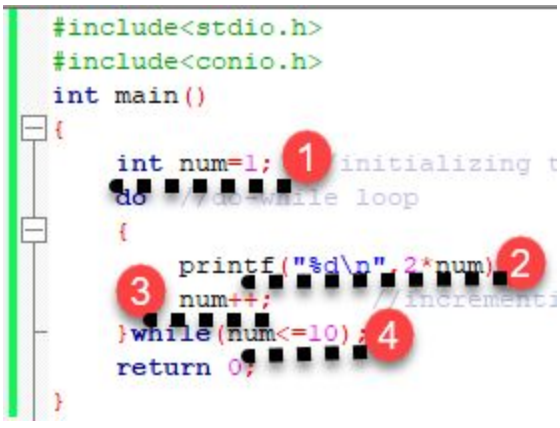
We are going to print a table of number 2 using do while loop.

```
#include<stdio.h>

#include<conio.h>

int main()

{

     int num=1;    //initializing the variable

     do   //do-while loop

     {

          printf("%d\n",2*num);

          num++;         //incrementing operation

     }while(num<=10);

     return 0;

}
```

Output:

```
2

4

6

8

10

12

14

16

18

20
```

In the above example, we have printed multiplication table of 2 using a do-while loop. Let's see how the program was able to print the series.



1. First, we have initialized a variable 'num' with value 1. Then we have written a do-while loop.
2. In a loop, we have a print function that will print the series by multiplying the value of num with 2.
3. After each increment, the value of num will increase by 1, and it will be printed on the screen.
4. Initially, the value of num is 1. In a body of a loop, the print function will be executed in this way: 2*num where num=1, then 2*1=2 hence the value two will be printed. This will go on until the value of num becomes 10. After that loop will be terminated and a statement which is immediately after the loop will be executed. In this case return 0.

For loop

A for loop is a more efficient loop structure in 'C' programming. The general structure of for loop is as follows:

```
for (initial value; condition; incrementation or
decrementation )

{

  statements;

}
```

- The initial value of the for loop is performed only once.
- The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
- The incrementation/decrementation increases (or decreases) the counter by a set value.

Following program illustrates the use of a simple for loop:

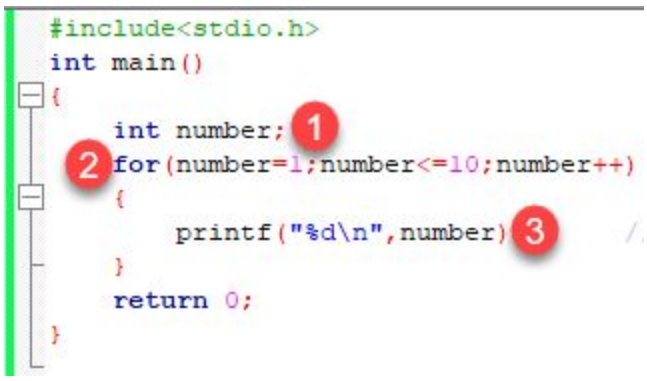
```
#include<stdio.h>

int main()

{

     int number;

     for(number=1;number<=10;number++)   //for loop to
print 1-10 numbers

     {

          printf("%d\n",number);     //to print the number

     }

     return 0;

}
```

Output:

```
1

2

3

4

5
```

```
6

7

8

9

10
```

The above program prints the number series from 1-10 using for loop.



1. We have declared a variable of an int data type to store values.
2. In for loop, in the initialization part, we have assigned value 1 to the variable number. In the condition part, we have specified our condition and then the increment part.
3. In the body of a loop, we have a print function to print the numbers on a new line in the console. We have the value one stored in number, after the first iteration the value will be incremented, and it will become 2. Now the variable number has the value 2. The condition will be rechecked and since the condition is true loop will be executed, and it will print two on the screen. This loop will keep on executing until the

value of the variable becomes 10. After that, the loop will be terminated, and a series of 1-10 will be printed on the screen.

In C, the for loop can have multiple expressions separated by commas in each part.

For example:

```
for (x = 0, y = num; x < y; i++, y--) {

  statements;

}
```

Also, we can skip the initial value expression, condition and/or increment by adding a semicolon.

For example:

```
int i=0;

int max = 10;

for (; i < max; i++) {

  printf("%d\n", i);

}
```

Notice that loops can also be nested where there is an outer loop and an inner loop. For each iteration of the outer loop, the inner loop repeats its entire cycle.

Consider the following example, that uses nested for loops output a multiplication table:

```
#include <stdio.h>

int main() {

int i, j;

int table = 2;

int max = 5;

for (i = 1; i <= table; i++) { // outer loop

  for (j = 0; j <= max; j++) { // inner loop

    printf("%d x %d = %d\n", i, j, i*j);

  }

  printf("\n"); /* blank line between tables */

}}
```

Output:

```
1 x 0 = 0

1 x 1 = 1

1 x 2 = 2
```

```
1 x 3 = 3

1 x 4 = 4

1 x 5 = 5


2 x 0 = 0

2 x 1 = 2

2 x 2 = 4

2 x 3 = 6

2 x 4 = 8

2 x 5 = 10
```

The nesting of for loops can be done up-to any level. The nested loops should be adequately indented to make code readable. In some versions of 'C,' the nesting is limited up to 15 loops, but some provide more.

The nested loops are mostly used in array applications which we will see in further tutorials.

Break Statement

The break statement is used mainly in in the switch statement. It is also useful for immediately stopping a loop.

We consider the following program which introduces a break to exit a while loop:

```
#include <stdio.h>

int main() {

int num = 5;

while (num > 0) {

  if (num == 3)

    break;

  printf("%d\n", num);

  num--;

}}
```

Output:

```
5

4
```

Continue Statement

When you want to skip to the next iteration but remain in the loop, you should use the continue statement.

For example:

```
#include <stdio.h>

int main() {

int nb = 7;

while (nb > 0) {

  nb--;

  if (nb == 5)

    continue;

 printf("%d\n", nb);

}}
```

Output:

```
6

4

3

2

1
```

So, the value 5 is skipped.

Which loop to Select?

Selection of a loop is always a tough task for a programmer, to select a loop do the following steps:

- Analyze the problem and check whether it requires a pre-test or a post-test loop.
- If pre-test is required, use a while or for a loop.
- If post-test is required, use a do-while loop.

Summary

- Looping is one of the key concepts on any programming language.
- It executes a block of statements number of times until the condition becomes false.
- Loops are of 2 types: entry-controlled and exit-controlled.
- 'C' programming provides us 1) while 2) do-while and 3) for loop.
- For and while loop is entry-controlled loops.
- Do-while is an exit-controlled loop.

Switch Case Statement in C Programming with Example

What is a Switch Statement?

A switch statement tests the value of a variable and compares it with multiple cases. Once the case match is found, a block of statements associated with that particular case is executed.

Each case in a block of a switch has a different name/number which is referred to as an identifier. The value provided by the user is compared with all the cases inside the switch block until the match is found.

If a case match is found, then the default statement is executed, and the control goes out of the switch block.

In this tutorial, you will learn-

Syntax

A general syntax of how switch-case is implemented in a 'C' program is as follows:

```
switch( expression )

{

     case value-1:

               Block-1;

               Break;

     case value-2:

               Block-2;
```

```
            Break;
    case value-n:
            Block-n;
            Break;
    default:
            Block-1;
            Break;
}
Statement-x;
```

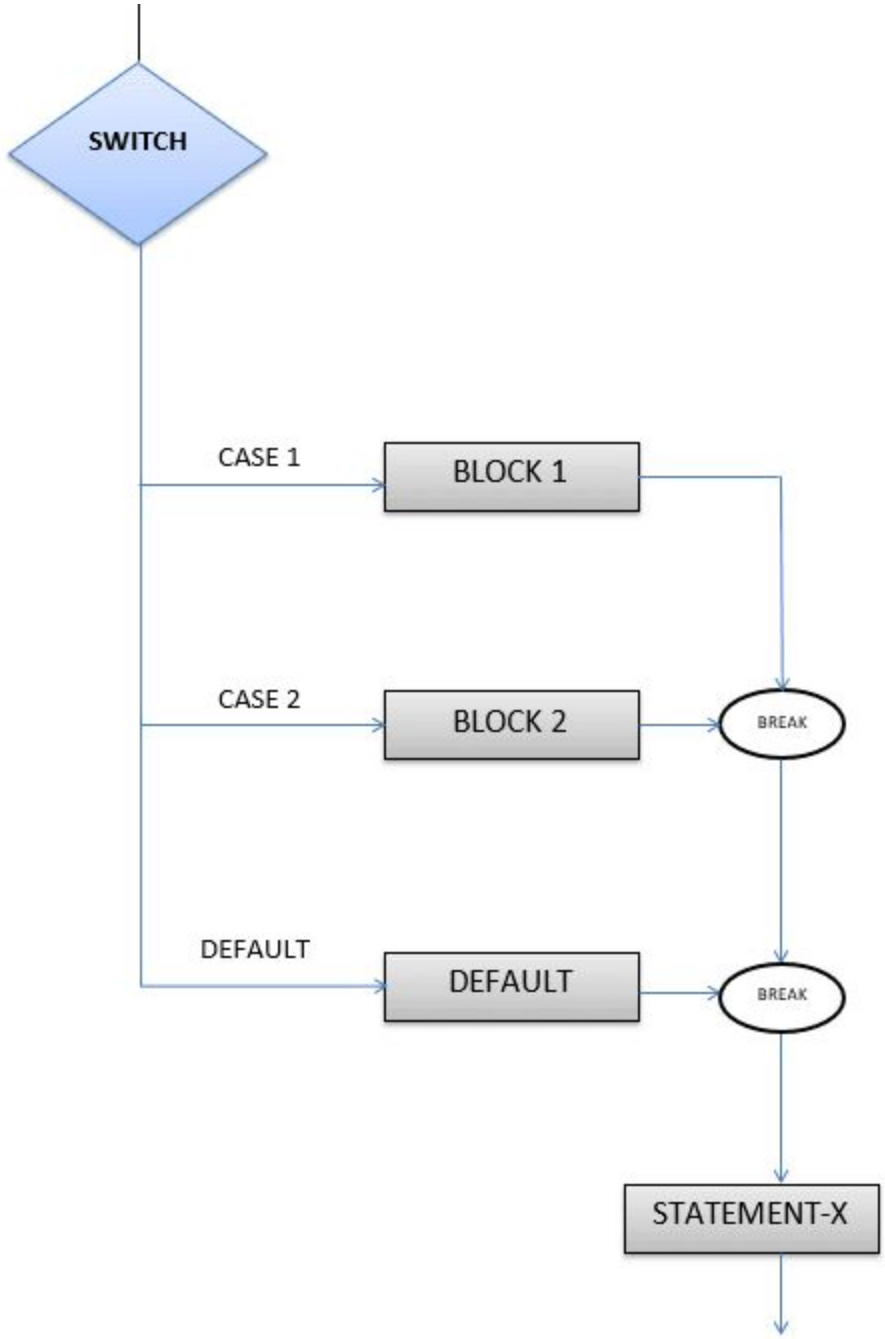
- The expression can be integer expression or a character expression.
- Value-1, 2, n are case labels which are used to identify each case individually. Remember that case labels should not be same as it may create a problem while executing a program. Suppose we have two cases with the same label as '1'. Then while executing the program, the case that appears first will be executed even though you want the program to execute a second case. This creates problems in the program and does not provide the desired output.
- Case labels always end with a colon (:). Each of these cases is associated with a block.
- A block is nothing but multiple statements which are grouped for a particular case.
- Whenever the switch is executed, the value of test-expression is compared with all the cases which we have defined inside the switch. Suppose the test expression contains value 4. This value is compared with all the cases until case whose label four is found in the program. As

soon as a case is found the block of statements associated with that particular case is executed and control goes out of the switch.

- The break keyword in each case indicates the end of a particular case. If we do not put the break in each case then even though the specific case is executed, the switch will continue to execute all the cases until the end is reached. This should not happen; hence we always have to put break keyword in each case. Break will terminate the case once it is executed and the control will fall out of the switch.
- The default case is an optional one. Whenever the value of test-expression is not matched with any of the cases inside the switch, then the default will be executed. Otherwise, it is not necessary to write default in the switch.
- Once the switch is executed the control will go to the statement-x, and the execution of a program will continue.

Flow Chart Diagram of Switch Case

Following diagram illustrates how a case is selected in switch case:



How Switch Works

Example

Following program illustrates the use of switch:

```c
#include <stdio.h>

    int main() {

        int num = 8;

        switch (num) {

            case 7:

                printf("Value is 7");

                break;

            case 8:

                printf("Value is 8");

                break;

            case 9:

                printf("Value is 9");

                break;

            default:

                printf("Out of range");

                break;

        }

        return 0;
```

```
    }
```

Output:

```
Value is 8
```

```c
#include <stdio.h>
    int main() {
        int num = 8;
    switch (num) {
            case 7:
                printf("Value is 7");
                break;
            case 8:
                printf("Value is 8");
                break;
            case 9:
                printf("Value is 9");
                break;
            default:
                printf("Out of range");
                break;
        }
        return 0;
    }
```

1. In the given program we have initialized a variable num with value 8.
2. A switch construct is used to compare the value stored in variable num and execute the block of statements associated with the matched case.
3. In this program, since the value stored in variable num is eight, a switch will execute the case whose case-label is 8. After executing the case, the control will fall out of the switch and program will be terminated with the successful result by printing the value on the output screen.

Try changing the value of variable num and notice the change in the output.

For example, we consider the following program which defaults:

```
#include <stdio.h>

int main() {

int language = 10;

  switch (language) {

  case 1:

    printf("C#\n");

    break;

  case 2:

    printf("C\n");

    break;

  case 3:

    printf("C++\n");

    break;

  default:

    printf("Other programming language\n");}}
```

Output:

```
Other programming language
```

When working with switch case in C, you group multiple cases with unique labels. You need to introduce a break statement in each case to branch at the end of a switch statement.

The optional default case runs when no other matches are made.

We consider the following switch statement:

```
#include <stdio.h>

int main() {

int number=5;

switch (number) {

  case 1:

  case 2:

  case 3:

    printf("One, Two, or Three.\n");

    break;

  case 4:

  case 5:

  case 6:

    printf("Four, Five, or Six.\n");

    break;
```

```
  default:
    printf("Greater than Six.\n");}}
```

Output:

```
Four, Five, or Six.
```

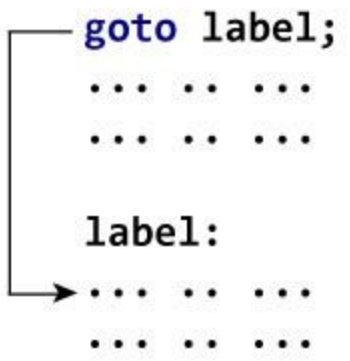
GO TO STATEMENT IN C

The `goto` statement allows us to transfer control of the program to the specified `label`.

Syntax of goto Statement

```
goto label;
... .. ...
... .. ...
label:
statement;
```

The `label` is an identifier. When the `goto` statement is encountered, the control of the program jumps to `label:` and starts executing the code.



Example: goto Statement

```
// Program to calculate the sum and average of positive numbers
// If the user enters a negative number, the sum and average are
displayed.

# include <stdio.h>

int main()
{

    const int maxInput = 5;
    int i;
    double number, average, sum=0.0;

    for(i=1; i<=maxInput; ++i)
    {
        printf("%d. Enter a number: ", i);
        scanf("%lf",&number);

        if(number < 0.0)
            goto jump;

        sum += number;
    }

    jump:
    average=sum/(i-1);
    printf("Sum = %.2f\n", sum);
    printf("Average = %.2f", average);

    return 0;
}
```

Output

```
1. Enter a number: 3
2. Enter a number: 4.3
3. Enter a number: 9.3
4. Enter a number: -2.9
Sum = 16.60
```

Reasons to avoid goto

The use of `goto` statement may lead to code that is buggy and hard to follow. For example,

```
one:
for (i = 0; i < number; ++i)
{
    test += i;
    goto two;
}
two:
if (test > 5) {
  goto three;
}
... .. ...
```

Also, the `goto` statement allows you to do bad stuff such as jump out of the scope.

That being said, `goto` can be useful sometimes. For example: to break from nested loops.

Should you use goto?

If you think the use of `goto` statement simplifies your program, you can use it. That being said, `goto` is rarely useful and you can create any C program without using `goto` altogether.