Andhra Pradesh State Skill Development Corporation

Programming in C

Conditional Statements

Conditional Statements

Conditional Statements are used to execute a set of Statements on some conditions. It provides a unit of block in which we can either one statement or more than one statements . If the given condition is True then the set of statements are executed otherwise the body is skipped.

Statements :

A statement is a syntactic unit of an imperative programming language that expresses some action to be carried out.

To control the flow of program execution c programming languages uses control statements.

- C provides two styles of flow control:
  - Branching (or) Selectional statements
  - Looping (or) Iterative statements

Branching is deciding what actions to take and looping is deciding how many times to take a certain action.

Selectional statements are used to make one-time decisions in C Programming, that is, to execute some code/s and ignore some code/s depending upon the test expression.

There are following types of conditional statements in C :

1. If statement
2. If-Else statement
3. If-Else If ladder
4. Nested If-else statement
5. Switch statement

Importance of Conditional Statements

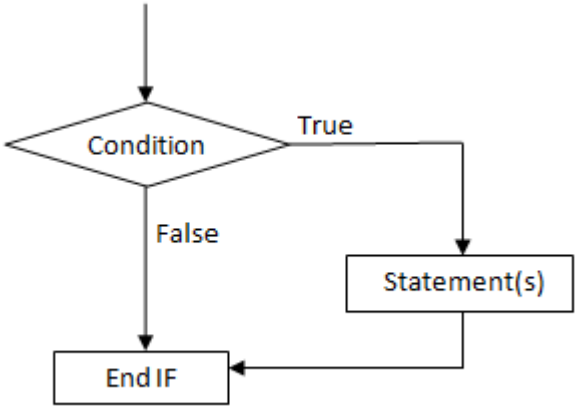
This is the ability to test a variable against a value and act in one way if the condition is met by the variable or another way if not. ... A logic statement or operation can be evaluated to be True or False.

IF Statement Syntax:

```
if (text expression)
{
        //Block of C statements here
        //These statements will only execute if the condition is true
}
```

The if statement checks whether the text expression inside parentheses () is true or not. If the test expression is true, statement/s inside the body of if statement is executed but if test is false, statement/s inside body of if is ignored.

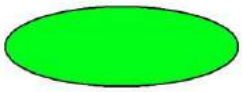
Flowchart of if statement :-



Flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem. It makes use of symbols which are connected among them to indicate the flow of information and processing.
The process of drawing a flowchart for an algorithm is known as "flowcharting".

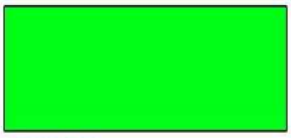
Basic Symbols used in Flowchart Designs

1. **Terminal:** The oval symbol indicates Start, Stop and Halt in a program's logic flow. A pause/halt is generally used in a program logic under some error conditions. Terminal is the first and last symbols in the flowchart.

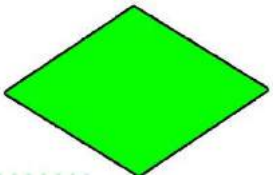


2. **Input/Output:** A parallelogram denotes any function of input/output type. Program instructions that take input from input devices and display output on output devices are indicated with parallelogram in a flowchart.

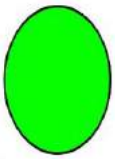
3. **Processing:** A box represents arithmetic instructions. All arithmetic processes such as adding, subtracting, multiplication and division are indicated by action or process symbol.



4. **Decision Diamond symbol represents a decision point.** Decision based operations such as yes/no question or true/false are indicated by diamond in flowchart.



5. **Connectors:** Whenever flowchart becomes complex or it spreads over more than one page, it is useful to use connectors to avoid any confusions. It is represented by a circle.



6. **Flow lines:** Flow lines indicate the exact sequence in which instructions are executed. Arrows represent the direction of flow of control and relationship among different symbols of flowchart.

Examples of if with single statement:

```
#include <stdio.h>
int main()
{
        int x = 20;
        int y = 22;
        if (x<y)
         {
                printf("Variable x is less than y");
         }
return 0;
}
```

Output: Variable x is less than y

Explanation:

The condition (x<y) specified in the "if" returns true for the value of x and y, so the statement inside the body of if is executed.

Example of multiple if statements:

We can use multiple if statements to check more than one conditions.

```
#include <stdio.h>
int main()
{
        int x, y;
        printf("enter the value of x:");
        scanf("%d", &x);
        printf("enter the value of y:");
        scanf("%d", &y);
        if (x>y)
        {
                printf("x is greater than y\n");
         }
         if (x<y)
        {
                printf("x is less than y\n");
```

```
        }
        if (x==y)
        {
                printf("x is equal to y\n");
        }
    printf("End of Program");
    return 0;
    }
```

In the above example the output depends on the user input.

Output:

enter the value of x:20
enter the value of y:20
x is equal to y
End of Program

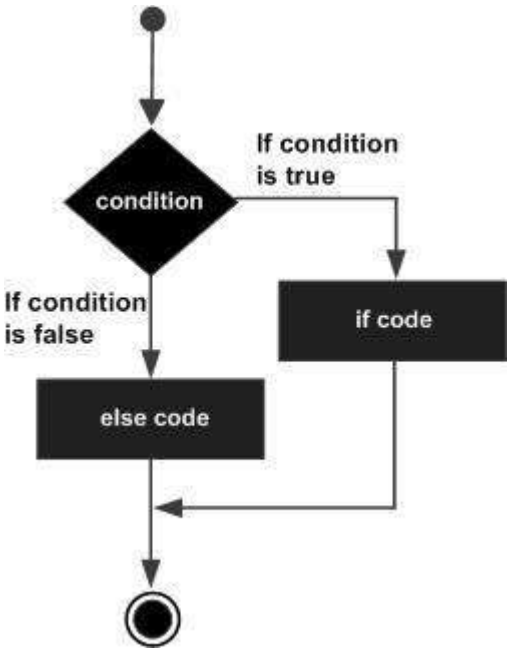
If Else Statement:

An if statement can be followed by an optional statement,which executes when the Boolean expression is false.

```
if (test expression)
{
        // statements to be executed if the test expression is true
}
else
{
        //statements to be executed if the test expression is false
}
```

If the test expression is evaluated to true,statements inside the body of if are executed. statements inside the body of else are skipped from execution.If the test expression is evaluated to false,statements inside the body of else are executed.statements inside the body of if are skipped from execution.

Flowchart of if else statement :-



Example for If else statement :-

// Check whether an integer is odd or even

```
#include<stdio.h>
int main()
{
        int number;
        printf("Enter an integer: ");
        scanf("%d",&number);

        // True if the remainder is 0
         if  (number%2 == 0)
         {
                printf("%d is an even integer.",number);
          }
        else {
                        printf("%d is an odd integer.",number);
                }

        return 0;
 }
```

Output:

Enter an integer: 7
7 is an odd integer**.**

Explanation:

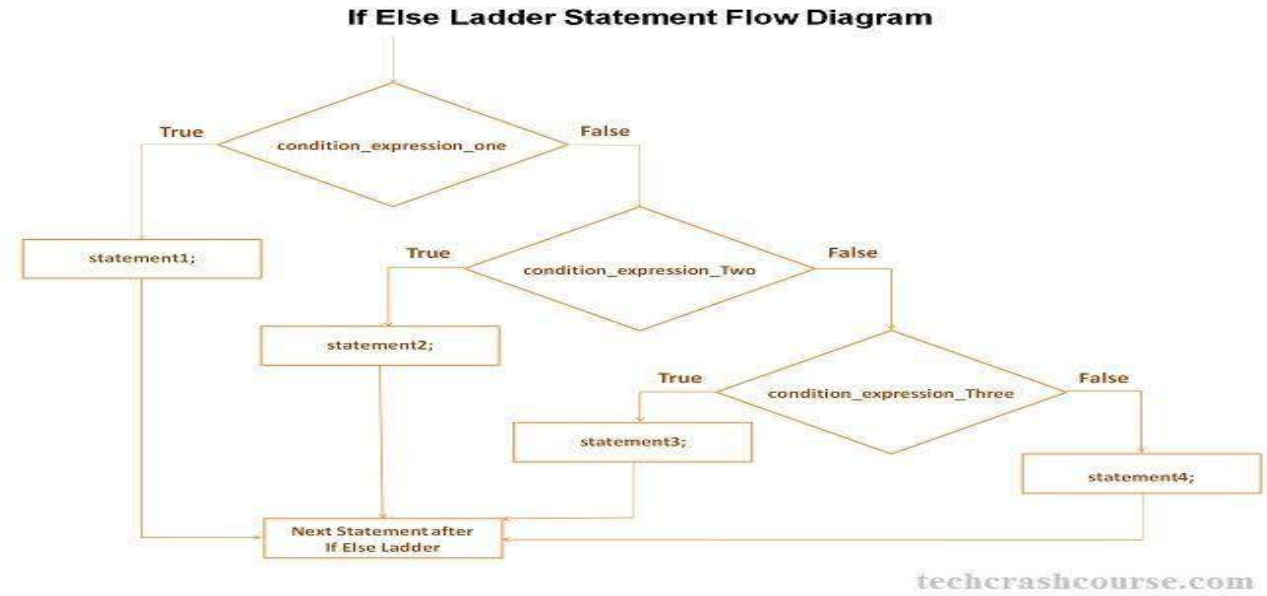
When the user enters 7, the test expression number%2==0 is evaluated to false. Hence, the statement inside the body of else is executed.

If..else Ladder:

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.The if...else ladder allows you to check between multiple test expressions and execute different statements.

Syntax:

```
if (test expression1)
{
        // statement(s)
 }
else if(test expression2)
{
        // statement(s)
}
else if (test expression3)
{
        // statement(s)
}
.
.
else
{
        // statement(s)
}
```



Example for If else Ladder statement :-

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

```
#include <stdio.h>
int main()
{
```

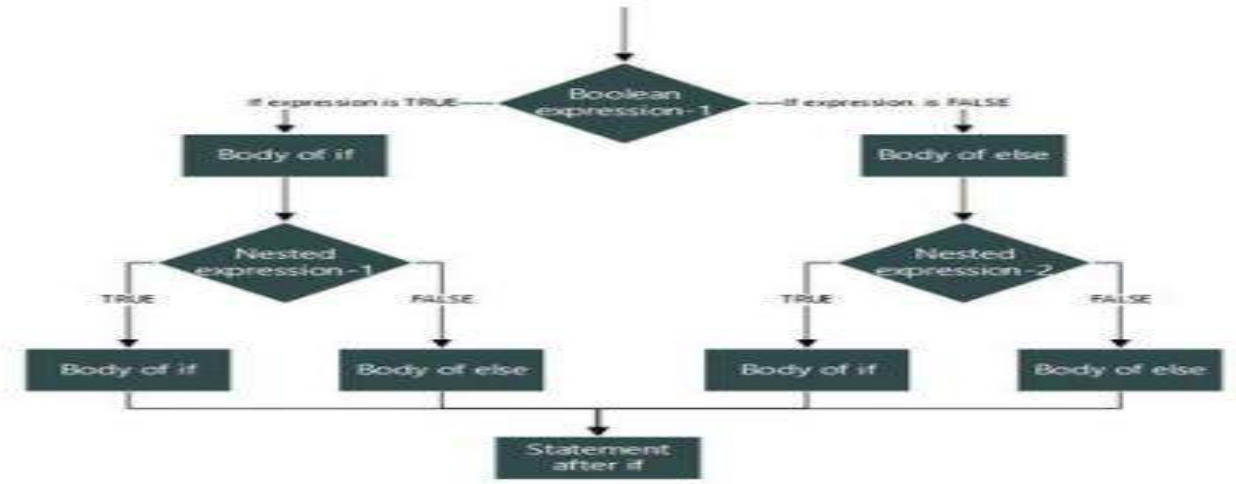
```
    int number1, number2;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    //checks if the 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);
     }

    //checks if both test expressions are false
    else
     {
        printf("Result: %d < %d",number1, number2);
     }
    return 0;
}
```

Output:

```
Enter two integers : 12   23
Result : 12 < 23
```

Nested If..else statement:

A nested if in C is an if statement that is the target of another if statement. Nested if statements means an if statement inside another if statement. Yes, both C and C++ allow us to nested if statements within if statements, i.e, we can place an if statement inside another if statement. When an if else statement is present inside the body of another "if" or "else" then this is called nested if else.



Syntax of Nested if else statement:

```
if(condition)
{
        //Nested if else inside the body of "if"
        if(condition2)
        {
                //Statements inside the body of nested "if"
        }
        else {
                //Statements inside the body of nested "else" }
        }
else {
        //Statements inside the body of "else"
}
```

Example of nested if..else

```
#include <stdio.h>
int main()
{
        int var1, var2;
        printf("Input the value of var1:");
        scanf("%d", &var1);
        printf("Input the value of var2:");
        scanf("%d",&var2);
        if (var1 != var2)
        {
                printf("var1 is not equal to var2\n");
                //Nested if else
                if (var1 > var2)
                {
                        printf("var1 is greater than var2\n"); }
                else {
                        printf("var2 is greater than var1\n");}
        }
        else {
                printf("var1 is equal to var2\n"); }
        return 0;
}
```

Output:

```
Input the value of var1:12
Input the value of var2:21
var1 is not equal to var2
var2 is greater than var1
```

Switch Statement:

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case

- The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
- Switch is a control statement that allows a value to change control of execution.

Rules for switch statement −

● The expression used in a switch statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.

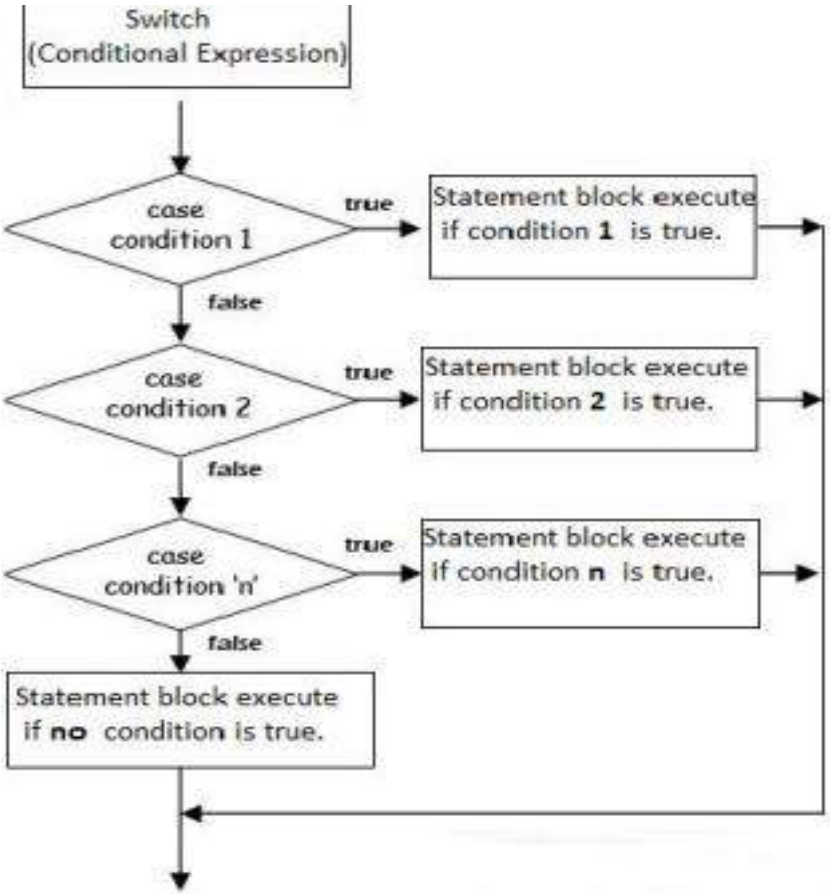
● You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
● The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.

● When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached. ● When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.

● Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.

● A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Flow chart of Switch statement :



Syntax of switch...case :

```
switch (expression)
{
        case constant1:
        // statements
        break;
        case constant2:
       // statements
       break;
        .
        .
        .
        default:
        // default statements
}
```

Example

```
#include <stdio.h>
int main ()
{
        /* local variable definition */ char grade = 'B';
        switch(grade)
         {
                case 'A' :
                printf("Excellent!\n" );
                break;
```

```
                case 'B' :
                case 'C' :
                printf("Well done\n" );
                break;
                case 'D' :
                printf("You passed\n" );
                break;
                case 'F' :
                printf("Better try again\n" );
                break;
                default :
                printf("Invalid grade\n" );
        }
        printf("Your grade is %c\n", grade );
        return 0;
}
```

Output:

```
Well done
Your grade is B
```

Conclusion:

An important aspect of programming is being able to determine if a condition is true or false and then perform specific actions depending upon the outcome of the condition test. This can be especially useful when used with the exit status of the previous command.We call conditional statements as controls statements also.