SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

PROGRAMMING FOR PROBLEM SOLVING

I YEAR - I SEM

UNIT 2 – C Programming Basics

TOPIC 8 – Decision Making and Branching

INTRODUCTION

- C program is a set of statements which are **normally executed sequentially** in the order in which they appear.
- This happens when no options or **no repetitions** of certain calculations are necessary.
- In practice, we have a number of situations where we may have to **change the order of execution of statements** based on certain conditions, or **repeat** a group of statements until certain specified conditions are met.
- This involves a kind of **decision making** to see whether a particular condition has occurred or not.
- Then direct the computer to execute certain statements accordingly.

DECISION-MAKING STATEMENTS

- C language possesses such decision-making capabilities by supporting the following statements:

    1. **if** statement
    2. **switch** statement
    3. **Conditional operator** statement
    4. **goto** statement

- These statements are popularly known as **decision-making statements**.

- Since these statements **‘control’** the flow of execution, they are also known as **control statements**.

DECISION MAKING WITH IF STATEMENT

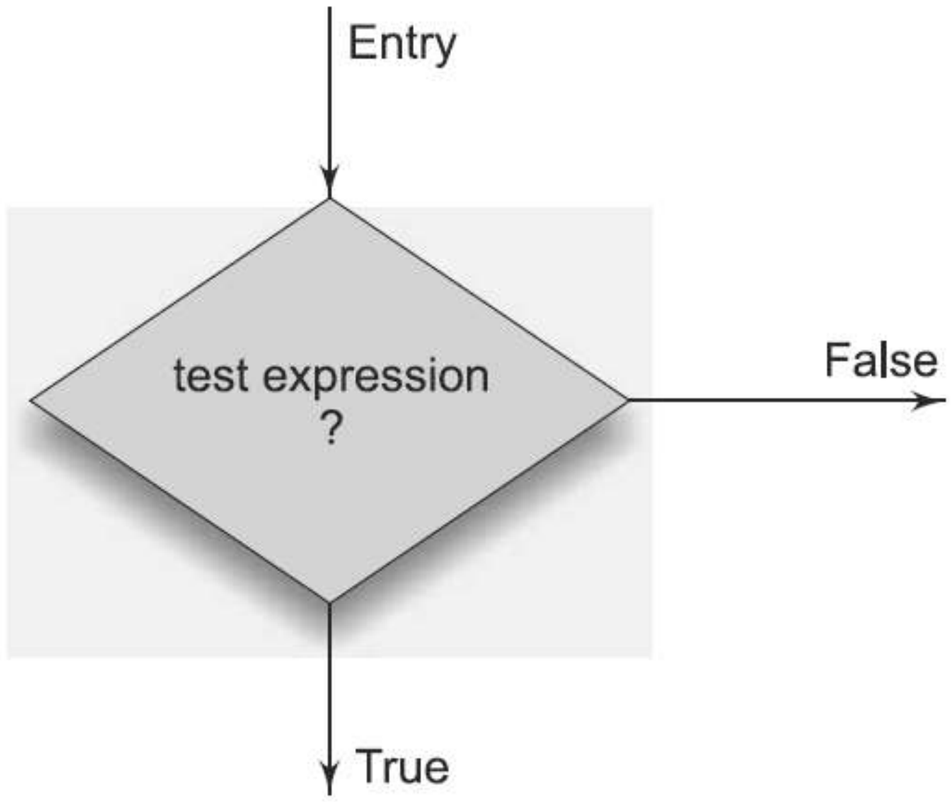
- The **if** statement is a powerful decision-making statement and is used to control the flow of execution of statements.
- It is basically a **two-way** decision statement and is used in conjunction with an expression.
- It takes the following form

  if (test expression)
- It allows the computer to evaluate the **expression first.**
- Then, depending on whether the value of the expression (relation or condition) is '**true**' (or non-zero) or '**false**' (zero), it transfers the control to a particular statement.
- This point of program has **two paths** to follow, one for the **true** condition and the other for the **false** condition as shown in Figure

DECISION MAKING WITH IF STATEMENT

- Some examples of decision making, using if statements are:

1. if (bank balance is zero)
   borrow money
2. if (room is dark)
   put on lights
3. if (code is 1)
   person is male
4. if (age is more than 55)
   person is retired



Two-way branching

DIFFERENT FORMS OF ‘IF’ STATEMENT

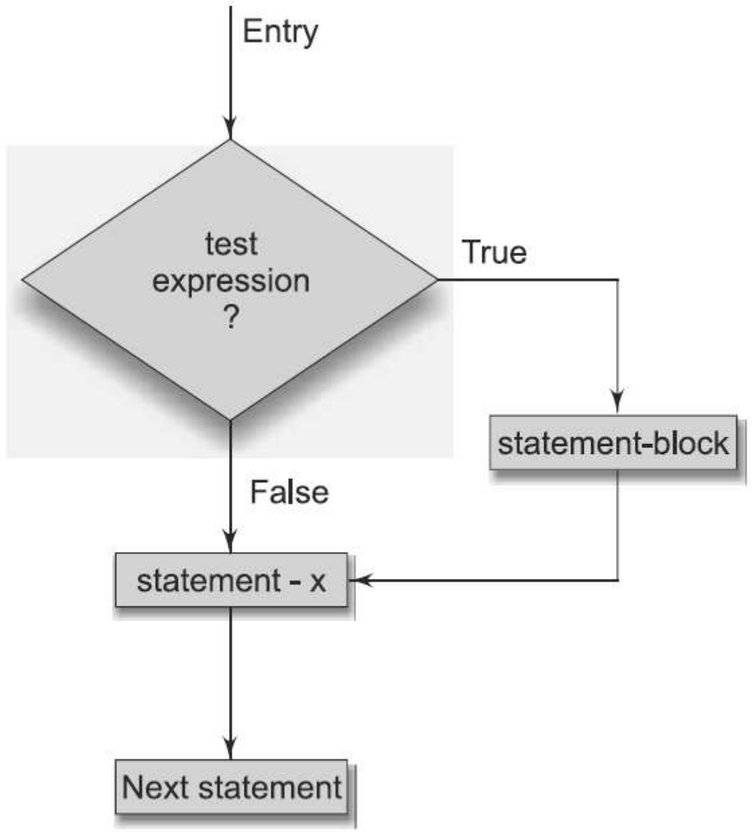
- The if statement may be implemented in different forms depending on the complexity of conditions to be tested.

- The different forms are:
  1. Simple **if** statement
  2. **if.....else** statement
  3. Nested **if....else** statement
  4. else if ladder.

SIMPLE IF STATEMENT

The general form of a simple if statement is:

```
if (test expression)
{
        statement-block;
}
statement-x;
```



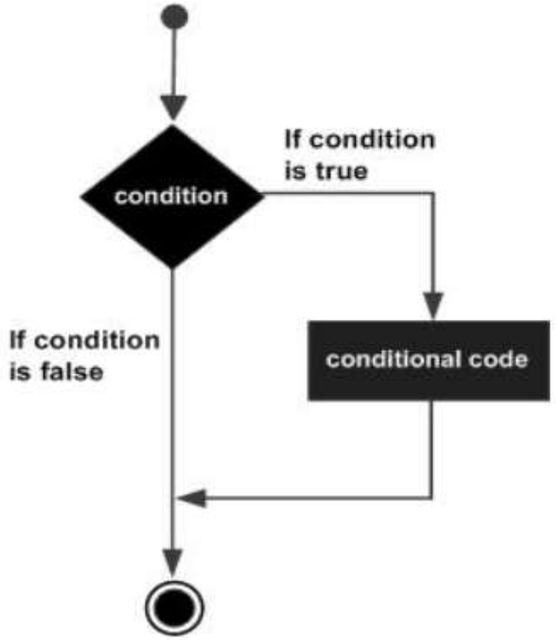
*Flow chart of simple **if** control*

SIMPLE IF STATEMENT

- The ‘statement-block’ may be a **single statement** or a group of statements.
- If the test expression is **true**, the statement-block will be executed
- Otherwise (**False**) the statement-block will be skipped and the execution will jump to the statement-x.
- Remember, when the condition is **true** both the statement-block and the statement-x are executed in sequence.

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* check the boolean condition using if statement */

   if( a < 20 ) {
      /* if condition is true then print the following */
      printf("a is less than 20\n" );
   }

   printf("value of a is : %d\n", a);

   return 0;
}
```

```
a is less than 20;
value of a is : 10
```



THE IF.....ELSE STATEMENT

- The if...else statement is an extension of the simple if statement.
- The general form is

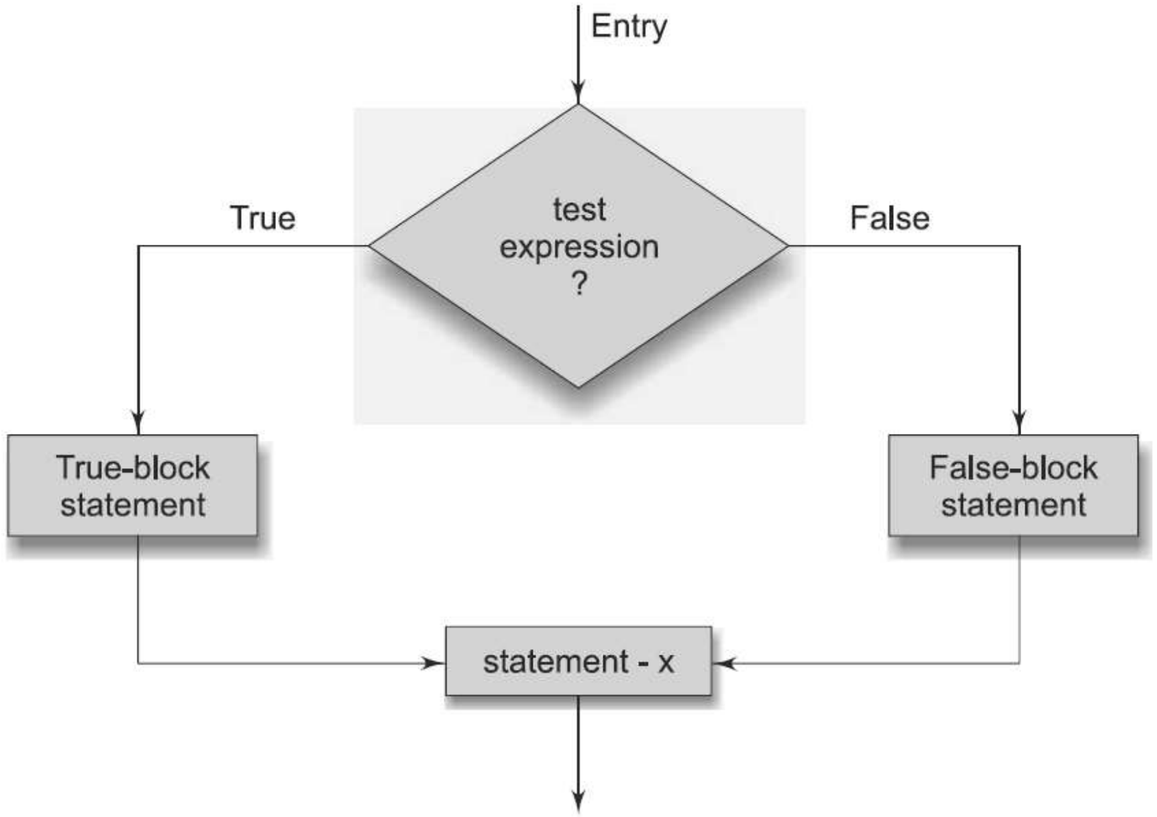
```
If (test expression)
{
        True-block statement(s)
}
else
{
False-block statement(s)
}
statement-x;
```

- If the test expression is **true**, then the true-block statement(s), immediately following the if statements are executed.
- If the test expression is **False**, the false-block statement(s) are executed.
- In either case, either true-block or false-block will be executed, not both.
- In both the cases, the control is transferred subsequently to the statement-x.

THE IF.....ELSE STATEMENT

```c
// 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;
}
```

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



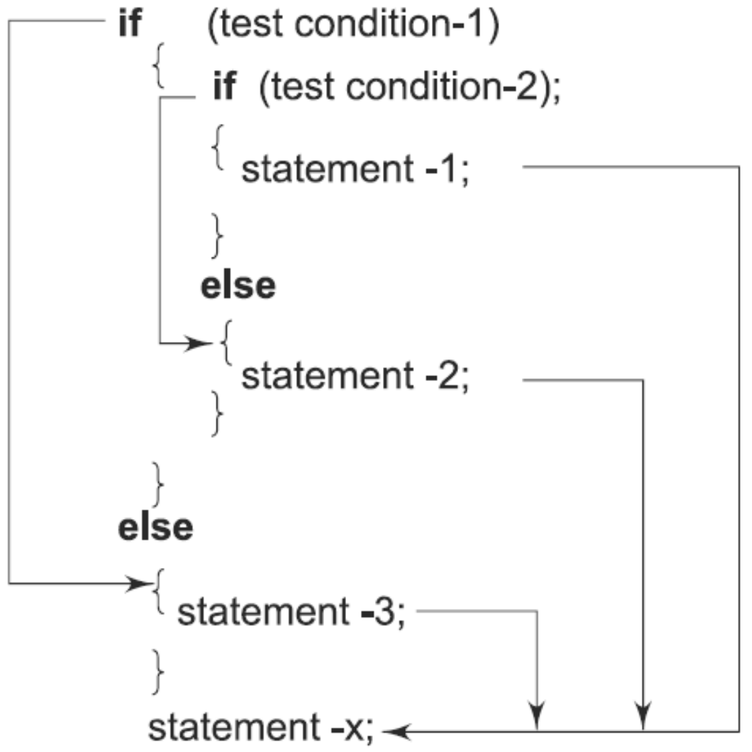
*Flow chart of **if.....else** control*

NESTING OF IF....ELSE STATEMENTS

- When a **series of decisions** are involved, we may have to use **more than one if...else** statement in **nested** form

- The logic of execution in Fig.

- If the **condition-1** is false, the statement-3 will be executed;
- Otherwise it continues to perform the second test.
- If the **condition-2** is true, the statement-1 will be evaluated
- Otherwise the **statement-2** will be evaluated.
- Then the control is transferred to the **statement-x**.

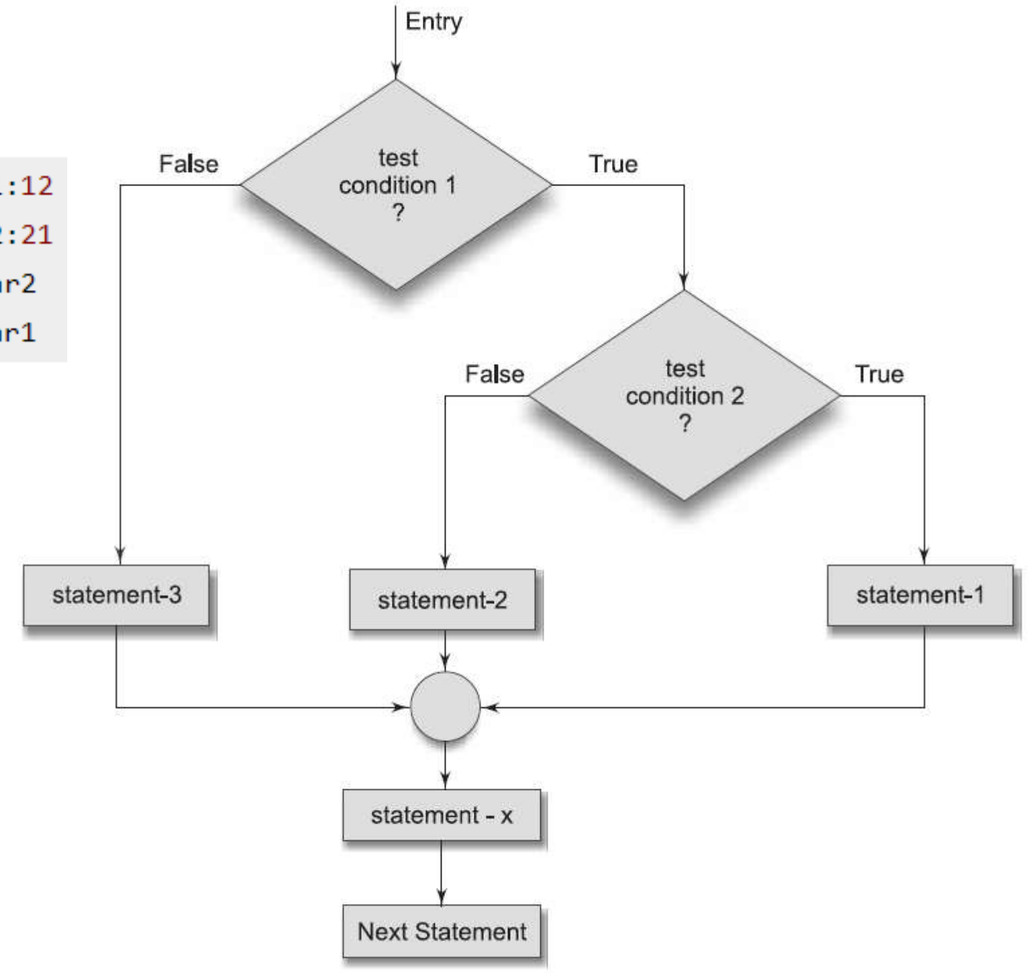


NESTING OF IF....ELSE STATEMENTS

```c
#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:

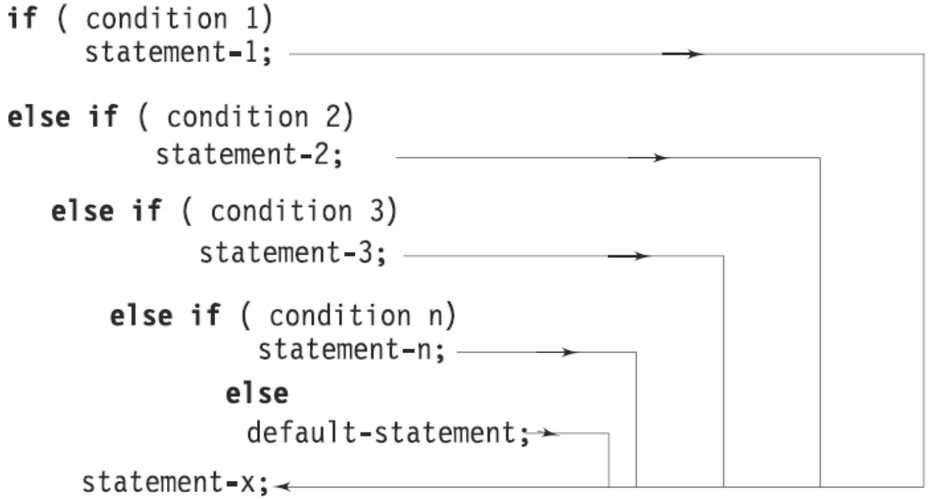
```
Input the value of var1:12
Input the value of var2:21
var1 is not equal to var2
var2 is greater than var1
```



*Flow chart of nested **if...else** statements*

THE ELSE IF LADDER

- There is another way of putting ifs together when multipath decisions are involved.
- A multipath decision is a chain of **if**s in which the statement associated with each else is an if.
- It takes the following general form

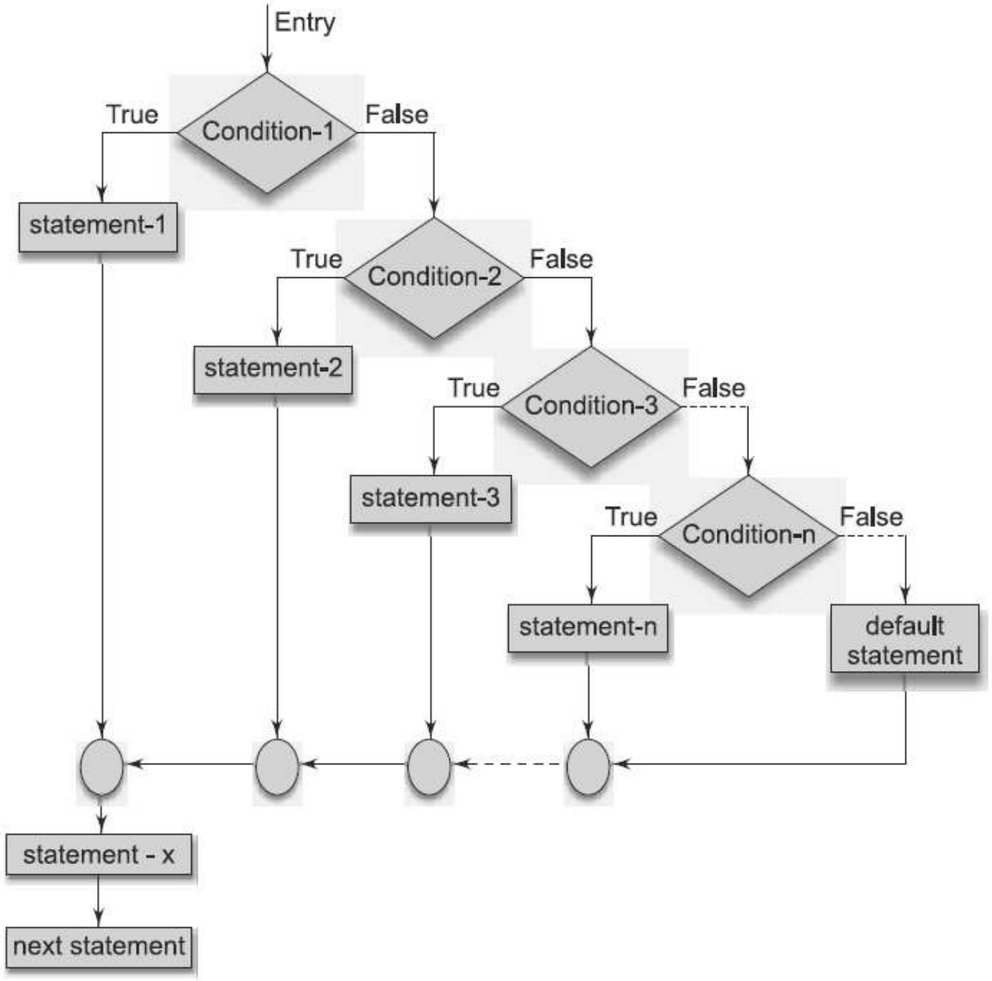


- This construct is known as the **else if** ladder.
- The conditions are evaluated from the top (of the ladder), downwards.
- As soon as a **true** condition is found, the statement associated with it is executed and the control is transferred to the statement-x (skipping the rest of the ladder).
- When all the n conditions become false, then the final else containing the **default-statement** will be executed.

THE ELSE IF LADDER

```c
#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;
}
```

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



Flow chart of **else..if** *ladder*

RULES FOR INDENTATION

- When using control structures, a statement often controls many other statements that follow it.
- In such situations it is a good practice to use **indentation** to show that the indented statements are dependent on the preceding controlling statement.

- Some guidelines that could be followed while using indentation are listed below:
  - Indent statements that are dependent on the previous statements; provide at least three spaces of indentation.
  - Align vertically else clause with their **matching if clause**.
  - Use **braces on separate lines** to identify a **block** of statements.
  - Indent the statements in the block by at least three spaces to the right of the braces.
  - Align the **opening and closing** braces.
  - Use appropriate **comments** to signify the beginning and end of blocks.
  - Indent the nested statements as per the above rules.
  - Code only one clause or statement on each line.

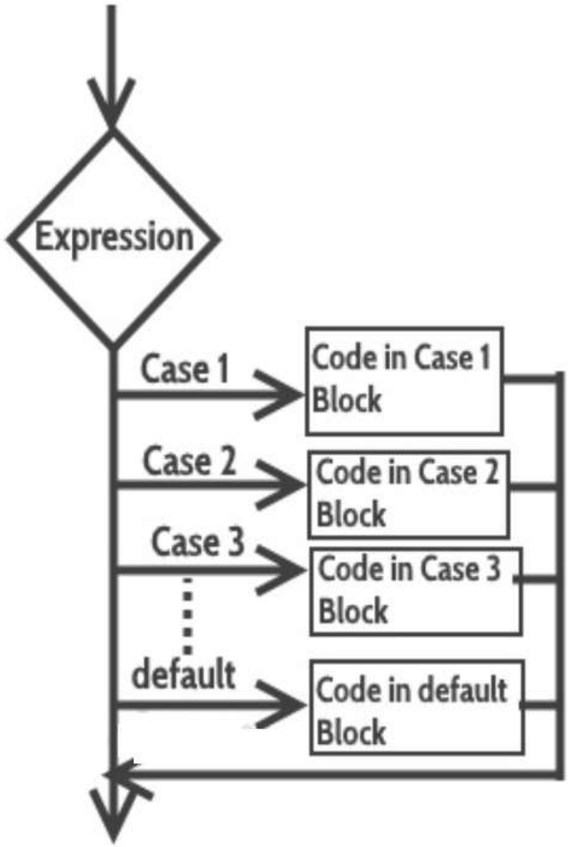
THE SWITCH STATEMENT

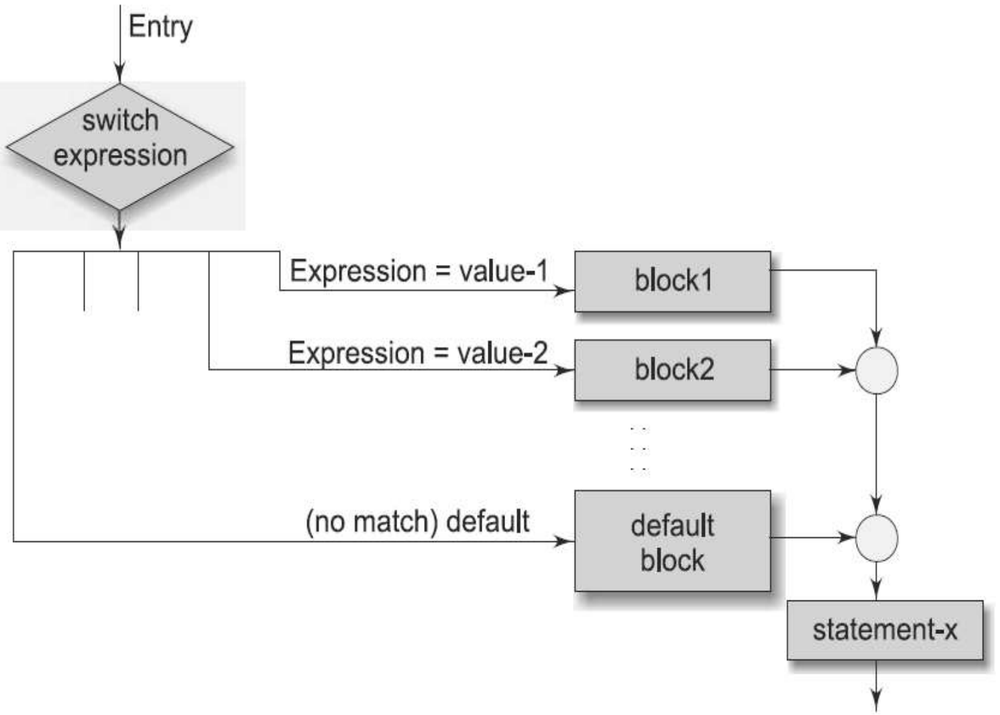
- We have seen that when **one of the many** alternatives is to be selected, we can use an **if statement** to control the selection.
- However, the **complexity** of such a program increases dramatically when the number of alternatives increases.
- The program becomes difficult to read and follow.
- At times, it may confuse even the person who designed it.
- Fortunately, C has a built-in **multiway** decision statement known as a **switch**.
- The **switch statement** tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statements associated with that case is executed.
- The general form of the switch statement is as discussed further.

THE SWITCH STATEMENT

➢ General Form

```
switch (expression)
{
        case value-1:
                block-1
                break;
        case value-2:
                block-2
                break;
.....
.....
default:
        default-block
        break;
}
statement-x;
```





*Selection process of the **switch** statement*

THE SWITCH STATEMENT

- The **expression** is an integer expression or characters.
- **Value-1, value-2** ..... are constants or constant expressions (evaluable to an integral constant) and are known as **case labels**.
- Each of these values should be unique within a switch statement. **block-1, block-2** .... are statement lists and may contain zero or more statements.
- There is no need to put braces around these blocks.
- Note that case labels end with a colon (:).
- When the switch is executed, the value of the expression is successfully compared against the values value-1, value-2,....
- If a case is found **whose value matches** with the value of the expression, then the block of statements that follows the case are executed.
- The **break** statement at the end of each block signals the end of a particular case and causes an **exit** from the switch statement, transferring the control to the statement-x following the switch.
- The **default** is an optional case.
- When present, it will be executed if the value of the expression does not match with any of the case values.
- If not present, no action takes place if all matches fail and the control goes to the statement-x.

- General Form

```
switch (expression)
{
        case value-1:
                block-1
                break;
        case value-2:
                block-2
                break;
.....
.....
default:
        default-block
        break;
}
statement-x;
```

THE SWITCH STATEMENT

➢ General Form

```
switch (expression)
{
        case value-1:
                block-1
                break;
        case value-2:
                block-2
                break;
.....
.....
default:
        default-block
        break;
}
statement-x;
```

```
#include <stdio.h>
int main()
{
    int i=2;
    switch (i)
    {
        case 1:
          printf("Case1 ");
          break;
        case 2:
          printf("Case2 ");
          break;
        case 3:
          printf("Case3 ");
          break;
        case 4:
          printf("Case4 ");
          break;
        default:
          printf(“Case not Found");
    }
}
```

Output:

Case 2

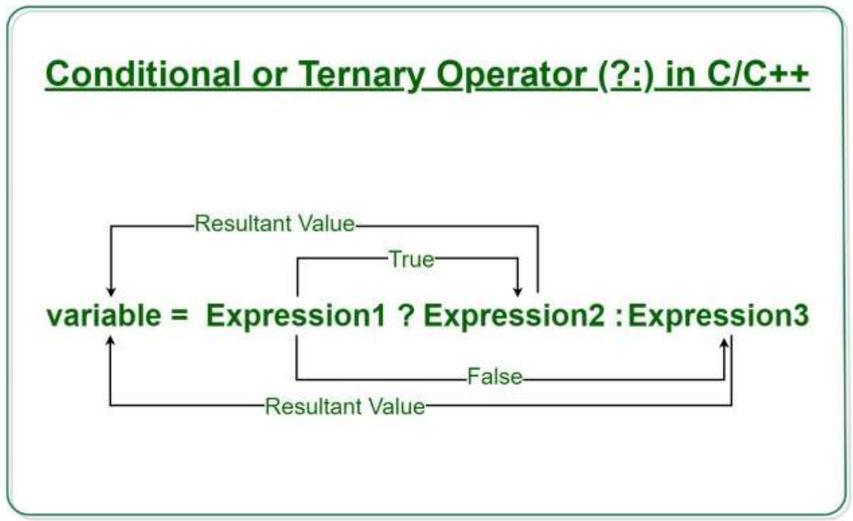
RULES FOR SWITCH STATEMENT

- The switch expression must be an integral type.
- Case labels must be constants or constant expressions.
- Case labels must be unique. No two labels can have the same value.
- Case labels must end with colon.
- The break statement transfers the control out of the switch statement.
- The break statement is optional. That is, two or more case labels may belong to the same statements.
- The default label is optional. If present, it will be executed when the expression does not find a matching case label.
- There can be at most one default label.
- The default may be placed anywhere but usually placed at the end.
- It is permitted to nest switch statements.

THE ? : OPERATOR

- The C language has an unusual operator, useful for making **two-way** decisions.
- This operator is a combination of ? and : , and takes **three operands**.
- This operator is popularly known as the **conditional operator**.
- The general form of use of the conditional operator is as follows:

  conditional expression ? expression1 : expression2
- The conditional expression is evaluated first.
- If the result is non-zero, expression1 is evaluated and is returned as the value of the conditional expression.
- Otherwise, expression2 is evaluated and its value is returned.



THE ? : OPERATOR

Conditional Operators Example

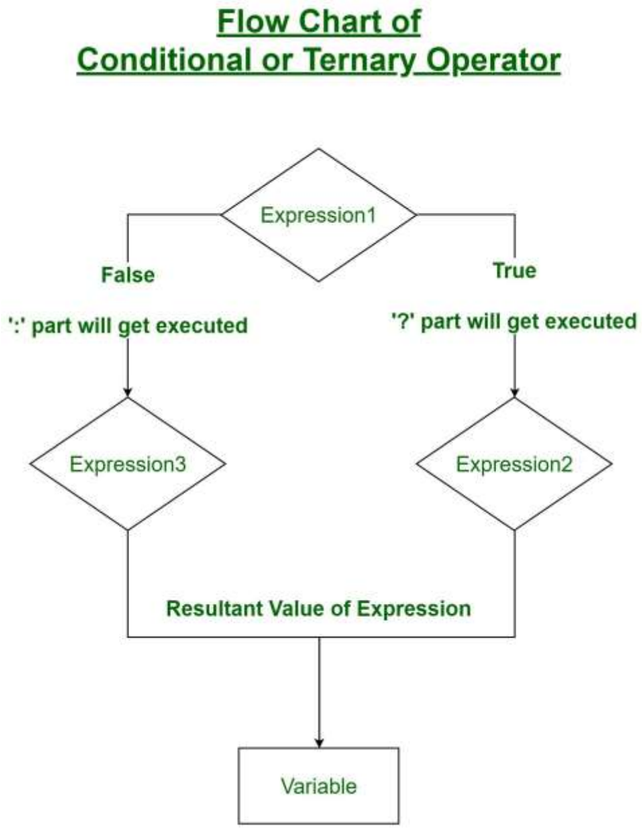
```
#include<stdio.h>
        void main()
        {
                int a, b, x;
                printf("Enter the values of a add b : ");
                scanf("%d %d", &a, &b);
                x=(a>b)?a:b;
                printf("Biggest Value is :%d",x);
        }
```

- **Output 01:**

Enter the values of a and b
2
1
Biggest value is 2

- **Output 02:**

Enter the values of a and b
1
2
Biggest value is 2

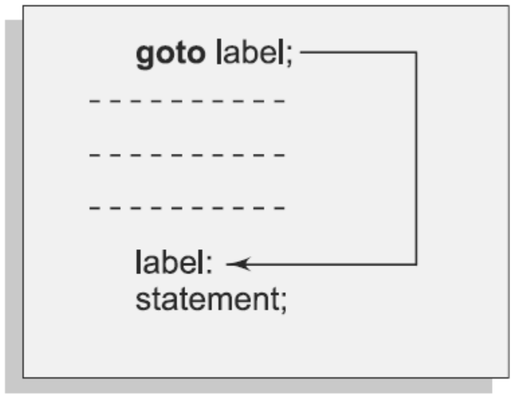


GUIDELINES FOR WRITING MULTIWAY SELECTION STATEMENTS

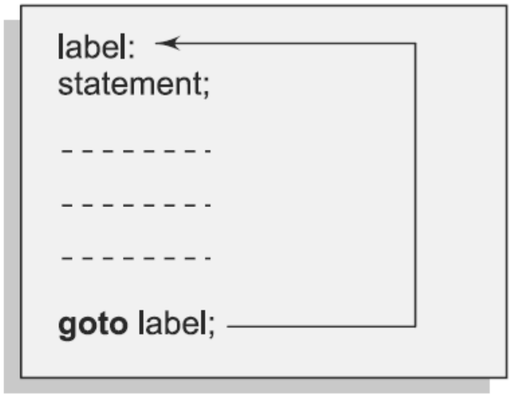
- Avoid compound negative statements.
- Use positive statements wherever possible.
- Keep logical expressions simple.
- Try to code the normal/anticipated condition first.
- Use the most probable condition first.
- This will eliminate unnecessary tests, thus improving the efficiency of the program.
- The choice between the nested if and switch statements is a matter of individual's preference.
- A good rule of thumb is to use the switch when alternative paths are three to ten.
- Use proper indentations (See Rules for Indentation).
- Have the habit of using default clause in switch statements.
- Group the case labels that have similar actions.

THE GOTO STATEMENT

- So far we have discussed ways of controlling the flow of execution based on certain specified conditions.
- Like many other languages, C supports the **goto** statement to branch unconditionally from one point to another in the program.
- The goto requires a **label** in order to identify the place where the branch is to be made.
- A label is any valid variable name, and must be followed by a **colon**.
- The **label** is placed immediately **before the statement** where the control is to be transferred.



Forward jump



Backward jump

THE GOTO STATEMENT

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   LOOP:do {

      if( a == 15) {
         /* skip the iteration */
         a = a + 1;
         goto LOOP;
      }

      printf("value of a: %d\n", a);
      a++;

   }while( a < 20 );

   return 0;
}
```

➢ Output

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```