## ❖ Decision Making and Branching:-

- C language has some decision-making capabilities by supporting of the following statements:
    1) 1. Simple if statement.
    2) 2. if…..else statement.
    3) 3. Nested if…..else statement.
    4) 4. else if ladder.
    5) 5. Switch statement.
    6) 6. goto statement.

## ❖ Simple IF statement: -

- The if statement is powerful decision making statement
- The if statement is used to control the flow of execution of statement.
- It is two-way decision statements and is used to conjunction with expression.
- General form of if statement is:

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

- The 'block-statement' may be a single statement or a group of statements.
- If the test condition is true, the block statement will be executed.
- Otherwise the block-statement will be skipped and execution will be jumped to the statement-x
- Consider the following example.

```
……..

if(category == sports)
{
        marks = marks + bonus_marks;
}
printf("%f", marks);
……..
```

❖ **The if…..else statement: -**

- The if…..else statement is a extension of simple is statement.
- The general form of if…..else statement is as under.

```
if (test condition)
{
true block statement;
}
else
{
false block statement;
}
statement-x;
```

- If the test condition is true, then the true block statement is executed.
- If the test condition is false, then the false block statement is executed.
- In either case, either true block or false block will be executed, not both.
- In the both cases, the control is transferred subsequently to statement-x.
- Let us consider an example of counting the number of boys and girls in class.
- We use the code 1 for boys and code 2 for girls.

```
……..
if(code == 1)
{
boy = boy + 1;
}
else
{
girl = girl + 1;
}
……..
……
```

❖ **Nesting of if…..else statement: -**

- When a series of decision are involved, we may have to use more then one if….else statement in
- nested from as shown below.

```
if(test condition-1)
{
if(test condition-2)
{
statement-1;
}
else
{
statement-2;
}
}
else
{
statement-3;
}
statement-x;
```

- If the condition-1 is true then test condition-2 is executed.
- If condition-2 is true then statement-1 is executed.
- If condition-2 is false then statement-2 is executed.

- But if the condition-1 is false then control is transfer to the statement-3. In this case statement-1
- and statement-2 is not executed.
- In both case control is transfer to the statement-x.

**For example:**

```
…….
if(s = 'f')
{
if(balance >= 5000)
{
bonus = 0.05 * balance;
}
else
{
bonus = 0.02 * balance;
}
}
else
{
bonus = 0.02 * balance;
}
balance = balance + bonus;
……
```

❖ **The else if ladder: -**

- There is another way of putting ifs together when multi path decisions are involved.
- A multi path decision is a chain of ifs in which the statement associated with each else is an if.
- General form of else if ladder is:

```
    if (condition-1)
statement-1;
else if (condition-2)
statement-2;
else if (condition-3)
statement-3;
else if (condition-n)
statement-n
else
default-statement;
statement-x;
```

- This construct is known as the else if ladder.
- The conditions are evaluated from the top (of 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.
- When all the conditions become false, then the final else containing the default-statement will be
- executed.
- Let us consider the following example.

```
……
if (code == 1)
colour = "red";
else if (code == 2)
colour = "green";
else if (code == 3)
colour = "white";
else
colour = "yellow";
……..
……..
```

❖ **The switch statement: -**

- The switch statement is another decision making and branching statement.
- We have seen that when one of many alternatives is to be selected, we can use an if statement to
- control the selection.
- When the numbers of alternatives are increased at that time we have to use switch statement.
- C has built-in multi way 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 statement associated with that case is executed.
- The general form of the switch statement is shown below:

```
switch (expression)
{
case value-1:
statement-1;
break;
case value-2:
statement-2;
break;
case value-3:
statement-3;
break;
.....
.....
default :
default statement;
break;
}
statement-x;
```

- The switch expression must be either integer or character type.
- Case labels must be integer or character.
- Case labels must be unique. No two labels can have the same values.
- Case labels must end with semicolon.
- 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
- statement.
- The default label is optional. If present, it will be executed when the expression does not find a
- matching case labels.
- There can be at most one default labels.
- The default may be placed anywhere but usually placed at the end.
- It is permitted to nest switch statements.

❖ **The ? : operator: -**

- Conditional operator is also known as a ternary operator.
- A conditional operator pair " ? : " is available in C.
- A conditional operator is used to create conditional expression.

Syntax is :

exp1 ? exp2 : exp3;

- In this syntax exp1 is evaluated first. If it is nonzero (true), then the expression exp2 is evaluated and become the value of expression.
- If exp1 is false, exp3 is evaluated and its value becomes the value of the expression.

- Note that only one of the expressions either exp2 or exp3 is evaluated.

For example:

```
int a=10;
int b=15;
int x;
x = (a<b) ? a : b ;
```

In this example, x will be assigning the value of b.

❖ **The go to statement: -**

- The goto requires a label in order to identify the place where the branch is to be made.
- A label is valid variable name, and must be followed by colon.
- The label is placed immediately before the statement where the control is to be transfer.
- The general form of goto statement is as under:

```
Goto label; label:
…….. statement;
…….. ……
…… ……..
Label: ……..
Statement; goto label;
```

**Forward Jump Backward Jump**

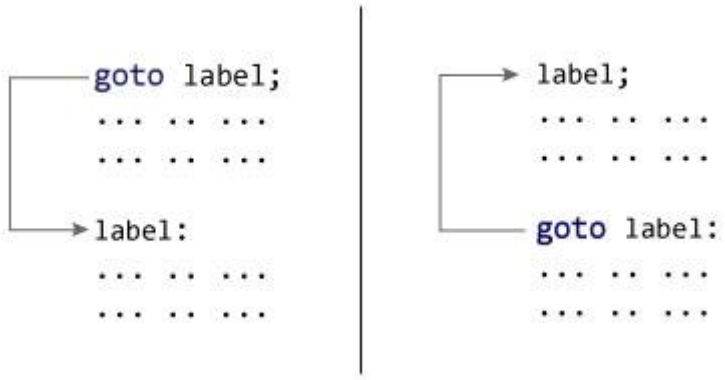
- The label can be anywhere in he program either before or after the goto label.
- If the label: is before the statement goto label; will be known as backward jump.
- If the label: is after the statement goto label; will be known as forward jump.
- In C programming, goto statement is used for altering the normal

sequence of program execution by transferring control to some other part of the program.

**Syntax of goto statement**

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

- In this syntax, label is an identifier. When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code below it.




**Example:**

```
int main()
{
  int age;
  Vote:
          printf("you are eligible for voting");
  NoVote:
           printf("you are not eligible to vote");
  printf("Enter you age:");
  scanf("%d", &age);
  if(age>=18)
```

```
        goto Vote;
    else
        goto NoVote;

    return 0;
}
```

**Explanation:**

- In the above example, Vote and NoVote are labels. When the input is >= 18, the goto statement is transferring the control to label – Vote, otherwise it transfers the control to label-NoVote.

**Reasons to avoid goto statement**

- Though, using goto statement give power to jump to any part of program, using goto statement makes the logic of the program complex and tangled.
- In modern programming, goto statement is considered a harmful construct and a bad programming practice.
- The goto statement can be replaced in most of C program with the use of break and continue statements.

❖ **Decision Making and Looping: -**

- Depending on the position of the control statement in the loop, a control structure may be
- classified either as the entry-control loop or as the exit-control loop.
- In entry control loop, the control conditions are tested before the start of the loop execution.
- If the conditions are not satisfied, then the body of the loop will not be executed.
- In the case of exit control loop, the test is performed at the end of the body of the loop and
- therefore the body is executed unconditionally for the first time.
- The entry control and exit control loops are also known as pre

tested and post tested loops.

- There are three type of loops available in C.

**For loop, while loop and do..while loop.**

❖ **The while statement: -**

- The simplest of all the looping structure in C is the while loop.
- The basic format of the while statement is:

```
while (test condition)
{
body of the loop;
}
statement-x;
```

- The while is an entry controlled loop statement.
- The test-condition is evaluated and if the condition is true, then the body of the loop is executed.
- After execution of the body, the test-condition is once again evaluated and if it is true, the body is
- executed once again.
- The process of repeated execution of the body continues until the test-condition finally becomes
- false and the control is transferred out of the loop.
- On exit, the program continues with the statements immediately after the body of the loop.
- Let us consider the following example.

```
sum =0;
n=1;
while(n<=10)
{
sum = sum + n * n ;
```

```
n = n + 1;
}
printf("sum = %d", sum);
……..
```

❖ **The do...while statement: -**

- On some time it is required to execute the body of the loop before the test condition is checked.
- This kind of situation can be handle with help of the do while statement.
- The do while statement is also known as exit control or post tested loop.
- General form of do while statement is :

```
do
{
body of the loop;
}
while (test condition);
statement-x;
```

- On reaching the do statement, the program proceeds to evaluate the body of the loop first.
- At the end of loop, the test condition in the while statement is executed.
- If the condition is true then control is further transfer to body of the loop.
- But if the condition is false control is transfer to statement-x.
- The do while loop create an exit control loop and therefore body of the loop is always executed at least once.
- A simple example of do while loop is :

```
do
{ printf("input a number");
number = number + 1 ;
```

```
} while (number>0 && number < 100);
```

❖ **The for statement: -**

- The for loop is another entry control or pre tested loop.
- The general form of for loop is:

```
for (initialization ; test condition ; increment or decrement)
{
body of the loop;
}
statement-x;
```

- There are three part in a for loop.
- Initialization, test condition and increment or decrement.
- Initialization done first. Initialization portion is executed only one time.
- Then control is transfer to test condition. If the condition becomes true then control is transfer to body of the loop.
- After the execution of body of the loop control is goes to increment or decrement portion.
- After the execution of increment or decrement part control is goes to test condition.
- If condition becomes false then control is transfer to statement-x.
- Consider the following example:

```
for (x=0; x<=10 ; x++)
{
printf("%d", x);
}
printf("\n");
```

**Nesting of for loop: -**

- Nesting of for loops, that is, one for statement within another for statement.
- For example consider the following :

```
for(row = 1; row <= romax; row++)
{
        for (column =1; column <= colmax; ++column)
        {
                y = row * column;
                printf("%d", y);
        }
        printf("\n");
}
```

- The nesting may continue up to any desired level.

❖ **Difference between for loop, while loop, and do while loop**

- While loop checks for the condition first. so it may not even enter into the loop, if the condition is false.
- do while loop, execute the statements in the loop first before checks for the condition. At least one iteration takes places, even if the condition is false.
- for loop is similar to while loop except that initialization statement, usually the counter variable initialization a statement that will be executed after each and every iteration in the loop, usually counter variable increment or decrement.

❖ **The differences between the entry and exit controlled loops are as follows.**

| No. | Topics | Entry controlled loops | Exit controlled loops |
|---|---|---|---|
| 01 | Test condition | Test condition appears at the beginning. | Test condition appears at the end. |
| 02 | Control variable | Control variable is counter variable. | Control variable is counter & sentinel variable. |
| 03 | Execution | Each execution occurs by testing condition. | Each execution except the first one occurs by testing condition. |
| 04 | Examples | = =<br>while(num>0)<br>{<br>printf("Input a number.\n");<br>scanf("%d", &num);<br>}<br>= = = = = == = = | = = = = = =<br>do<br>{<br>printf("Input a number.\n");<br>scanf("%d", &num);<br>}<br>while(num>0);<br>= = = = = = |

**Jumping in loops: -**

- Loops perform the set of operations repeatedly until the control variable fails to satisfy the test condition.
- But sometimes, when executing a loop it becomes desirable to skip a part of loop or to leave the loop as soon as a certain condition is occurs.
- At that time we have to use a break statement. When the break statement is encountered inside
- the loop, the loop is immediately exited and the program continues with the statement immediately following the loop.

**Skipping a part of loop:-**

- During the loop operation, it may necessary to skip a part of the body of the loop under certain conditions.

- Like the break statement, C supports another similar statement called continue statement.
- The continue statement tells the compiler that "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT STATEMENT."
- We can use the continue statement with for loop, while loop, do while loop, and goto statement.

  For example. :- while(test condition)

  { ……..

  if( condition)

  continue;

  ……..

  }

❖ **break and continue Statement**

- There are two statements built in C programming, break; and continue; to alter the normal flow of a program.
- Loops perform a set of repetitive task until text expression becomes false but it is sometimes desirable to skip some statement/s inside loop or terminate the loop immediately without checking the test expression.
- In such cases, break and continue statements are used. The break; statement is also used in switch statement to exit switch statement.

***break Statement***

- In C programming, break is used in terminating the loop immediately after it is encountered.
- The break statement is used with conditional if statement.

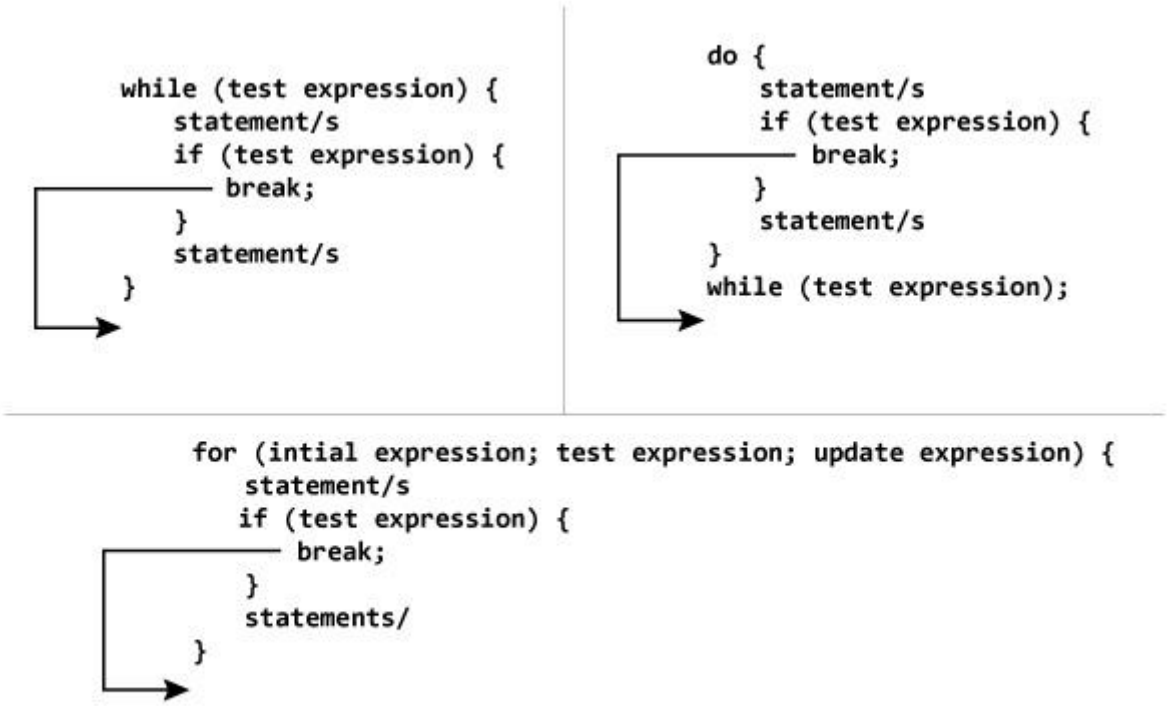
**Syntax of break statement**

```
break;
```

- The break statement can be used in terminating all three loops for,

while and do...while loops.

- The figure below explains the working of break statement in all three type of loops.




NOTE: The break statment may also be used inside body of else statement.

## *continue Statement*

- It is sometimes desirable to skip some statements inside the loop. In such cases, continue statements are used.

### Syntax of continue Statement

continue;

- Just like break, continue is also used with conditional if statement.

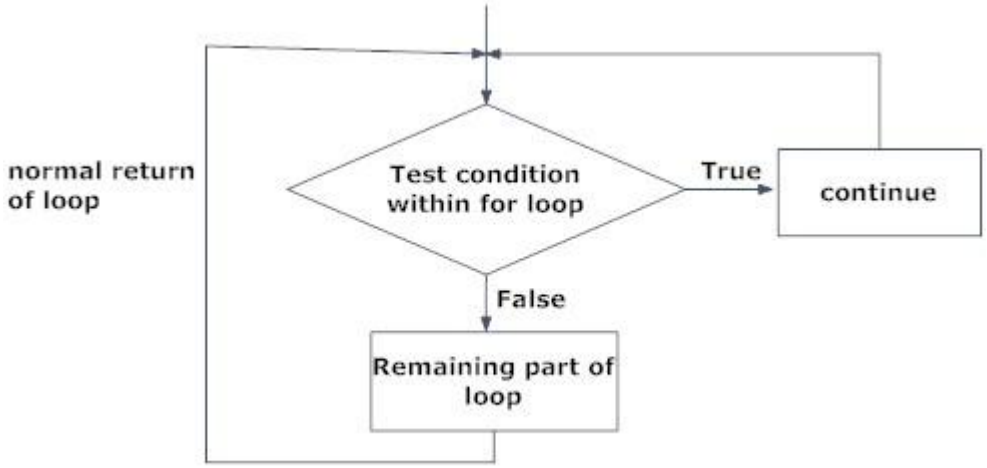



Fig: Flowchart of continue statement

- For better understanding of how continue statements works in C programming. Analyze the figure below which bypasses some code/s inside loops using continue statement.

```
while (test expression) {
    statement/s
    if (test expression) {
        continue;
    }
    statement/s
}
```

```
do {
    statement/s
    if (test expression) {
        continue;
    }
    statement/s
}
while (test expression);
```

```
for (intial expression; test expression; update expression) {
    statement/s
    if (test expression) {
        continue;
    }
    statements/
}
```

NOTE: The continue statment may also be used inside body of else statement.

***Example – Use of break in a while loop***

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

```
    int num =0;
    while(num<=100)
    {
       printf("variable value is: %d", num);
       if (num==2)
       {
          break;
       }
       num++;
    }
    printf("Out of while-loop");
    return 0;
}
```

**Output:**

```
variable value is: 0
variable value is: 1
variable value is: 2
Out of while-loop
```

***Example: Use of continue in a do while loop***

```
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do
   {
      if( a == 15)
      {
```

```
            /* skip the iteration */
            a = a + 1;
            continue;
         }
         printf("value of a: %d\n", a);
         a++;

      }while( a < 20 );

      return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
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
```

**Difference Between Break and Continue Statement in C**

| Break | Continue |
|---|---|
| Break statement is used to transfer control of the program outside loop or switch case statement either conditionally or unconditionally. | Continue statement is used to skip some statement of the loop and moves to the next iteration in the loop. |
| Break statement is used in loop as well as Switch Case Statement. | Continue statement is used only within loop. |
| **Example:**<br>for(i=1;i<=10;i++)<br>{<br>if(i%5==0)<br>break;<br>else<br>printf("%d",&i);<br>}<br>**Output:** | **Example:**<br>for(i=1;i<=10;i++)<br>{<br>if(i%5==0)<br>continue;<br>else<br>printf("%d",&i);<br>}<br>**Output:** |