Unit-II

Am.
9:30 Mon-22-Jan
2024
Gsled.

Control Structures

Simple sequential programs, Conditional statements (if, if-else, switch), Loops (for, while, do-while), Break and Continue.

C program is a set of statements which are normally executed sequentially in the order in which they appear.

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

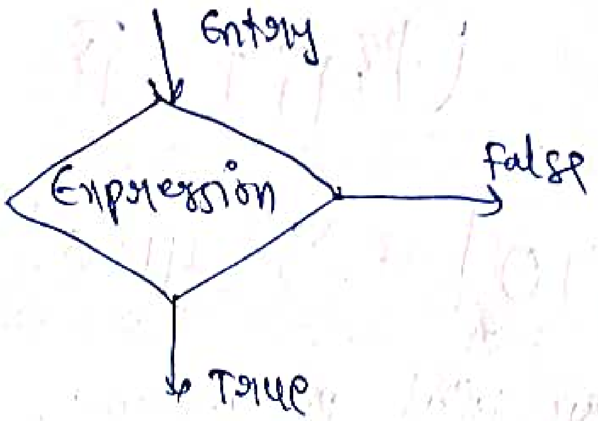
→ if statement.
→ Switch statement
→ Conditional operator statement
→ go to statement.

These statements are popularly known as decision-making statements. These statements control the flow of execution, they are also known as control statements.

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 2-way decision statement and is used in conjunction with an expression.



2-way branching.

It allows the computer to evaluate the expression first and then, depending on whether the value of the expression is true or false, it transfers the control to a particular statement.

eg: if (room is dark)
put on lights.

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

The different forms are:

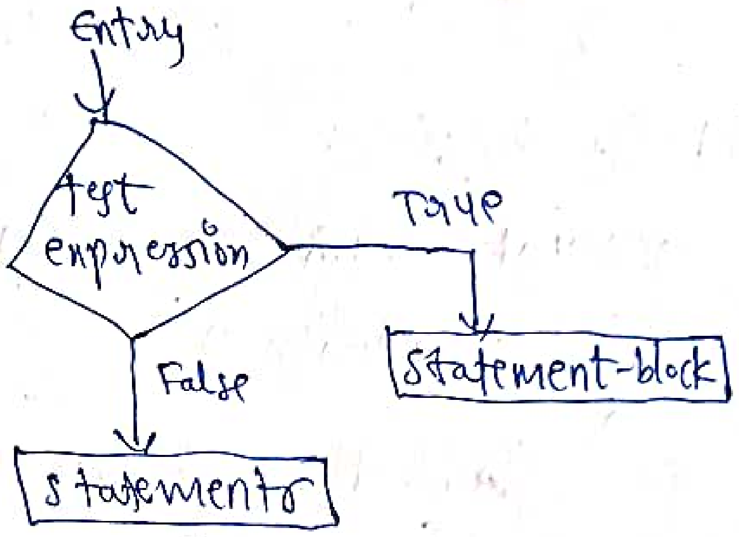
→ simple if
→ if else
→ Nested if-else
→ else if ladder.

Simple if:

The general form of simple if is

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

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 the statement-block will be skipped and the execution will jump to the statements.

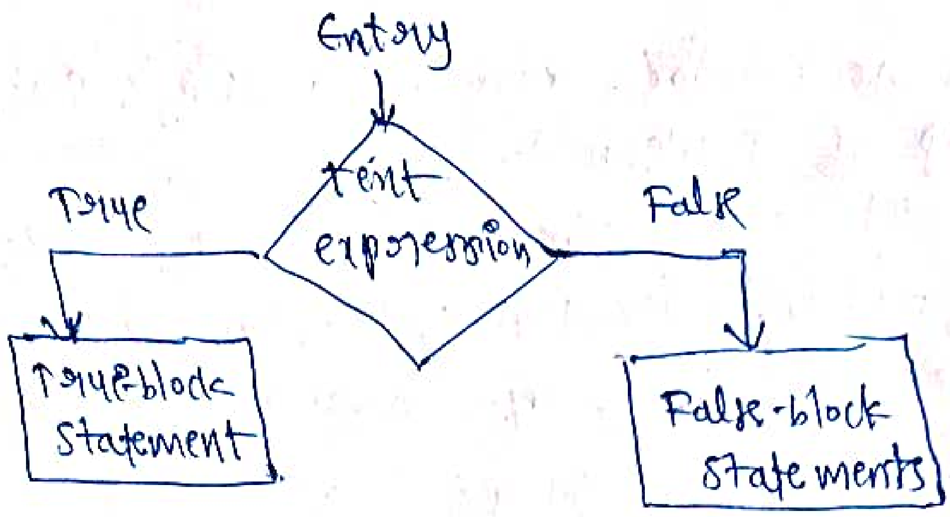


If-else:

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

```
if (test expression)
    {
        True-block statements.
    }
else
    {
        False block statements
    }
```

If the test expression is true, then the true-block statement, otherwise false-block statements are executed.

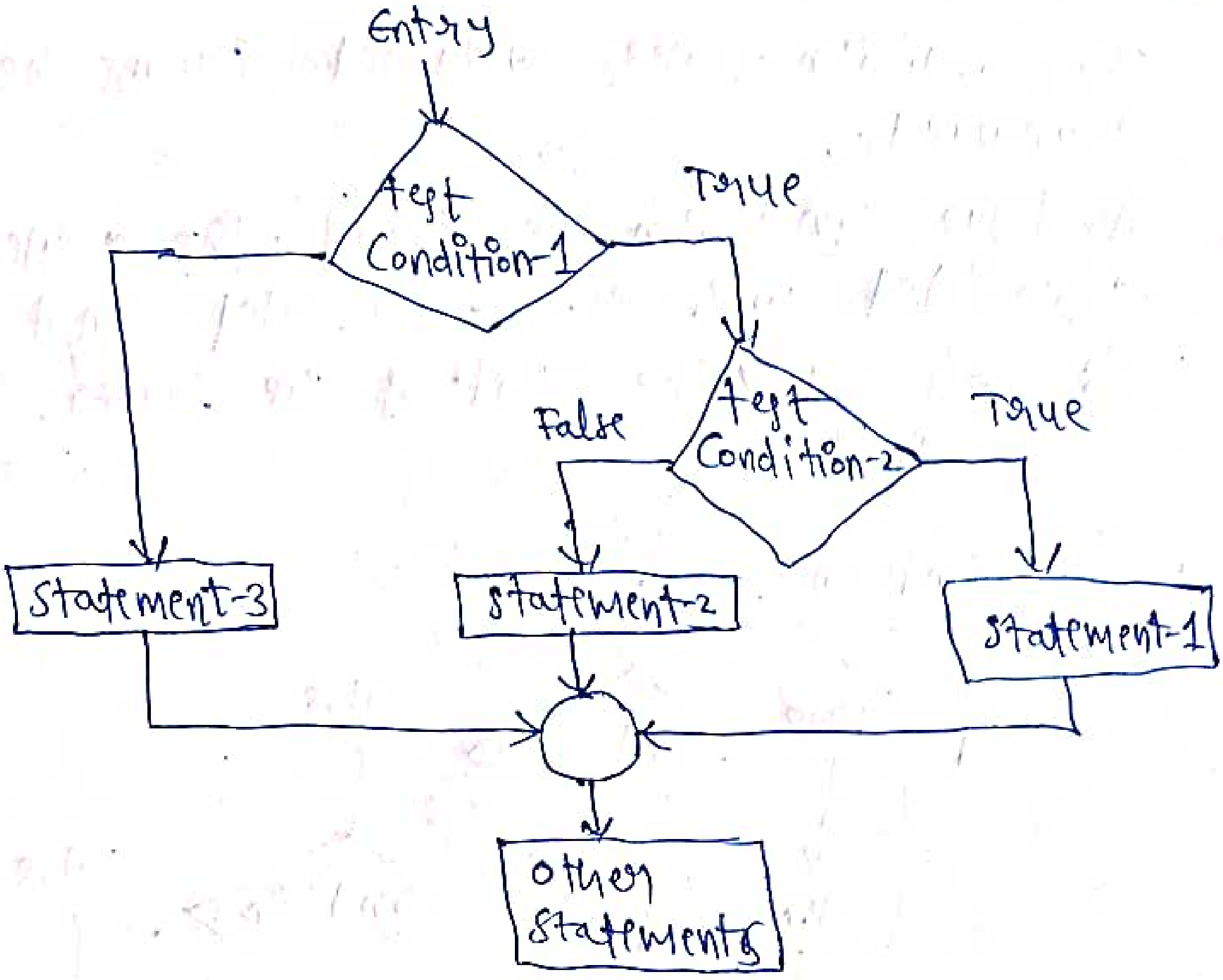


Nested-if-else:

The general form of nested if-else statement is

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

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.



Flowchart of nested if-else.

The else-if ladder:

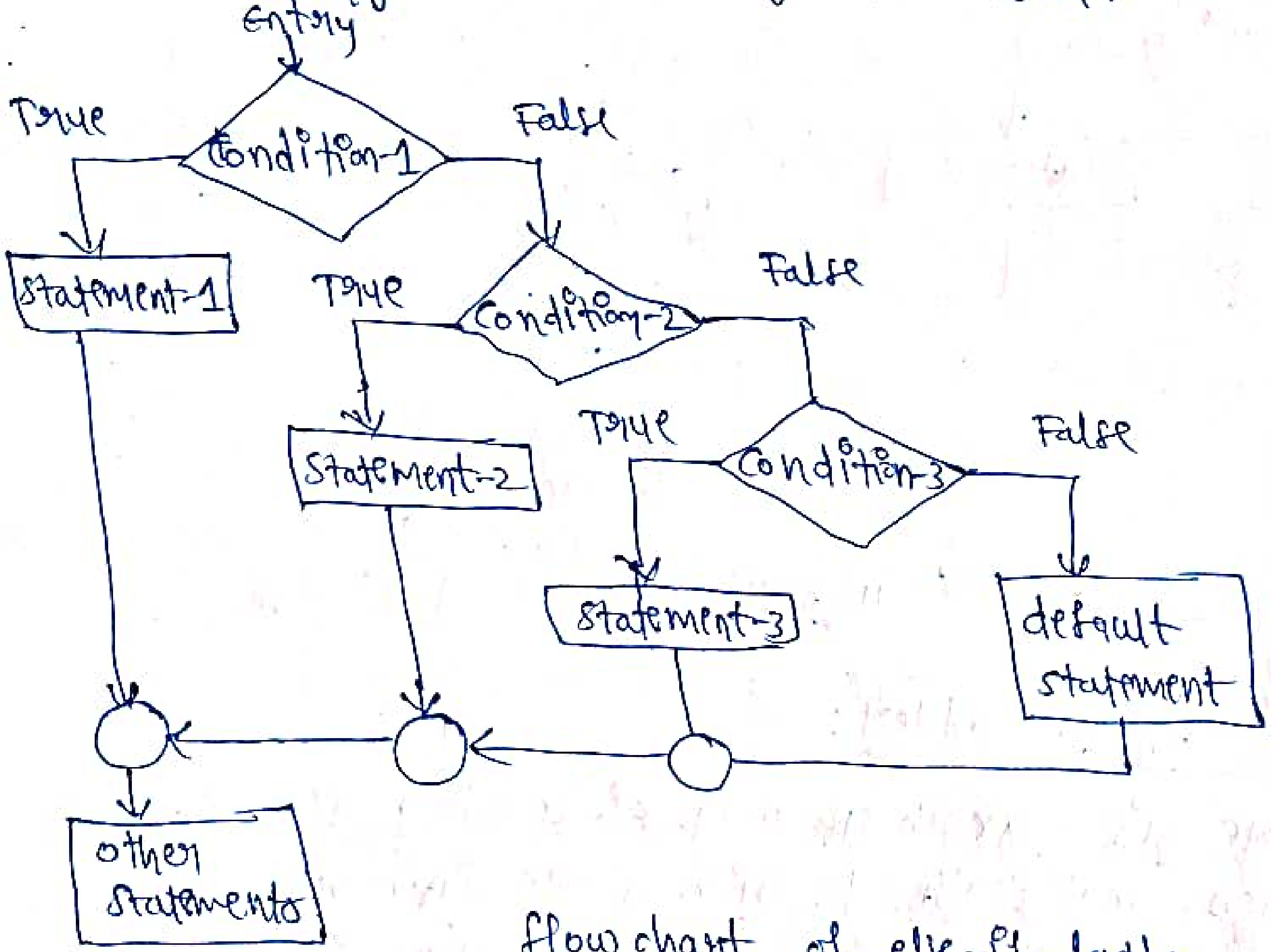
There is another way of putting ifs together when multipath decisions are involved.

A multipath decision is a chain of ifs in which the statements associated with each else is an if.

The 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 {
  default statement-4;
  }
```

The conditions are evaluated from the top downwards.
A true condition is found, the statement associated with it is executed and the control is skipping ~~to~~ the rest of the ladder.



Flow chart of else-if ladder.

Rules of Indentation:

When using control structures a statement often controls many other statements that follow it.
It is a good practice to use indentation to show that the intented statements are dependent on the preceding controlling statements.

Some guidelines should be followed while using indentation:

→ 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.

<u>Switch Statement</u>:

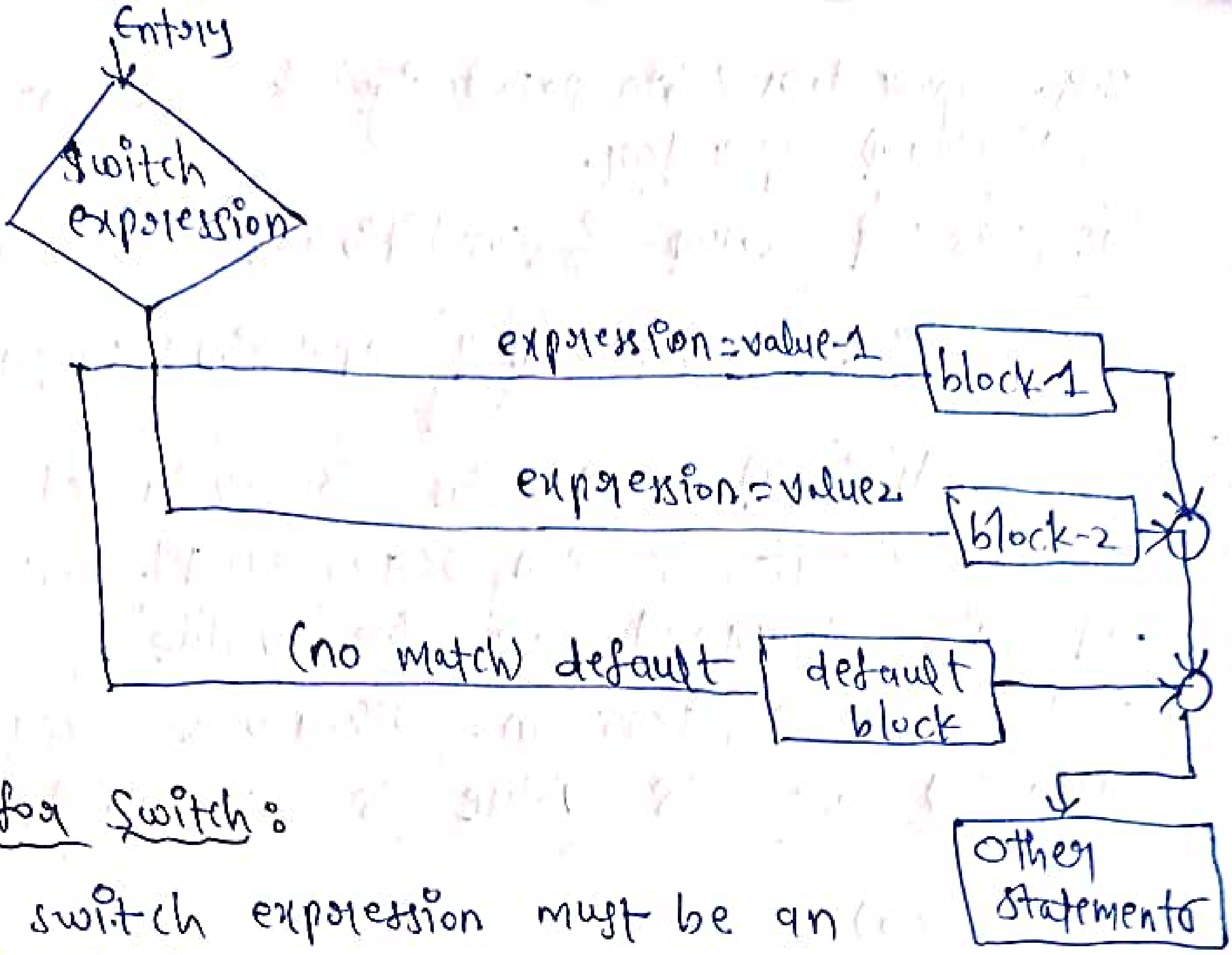
C has a built-in multiway decision statement known as a switch. The switch statement tests the value of a given variable against a list of case values and when a match is found, a block of statements associated with that case is executed.

The general form of the switch statement is

```
switch (expression)
{
    case value1:
            block-1
            break;
    case value-2:
            block-2
            break;

    case value-3:
            block-3
            break;
    -----
    -----
    default:
            default-block
            break;
}
```

Values are known as case labels. Each of these values should be unique within a switch statement. Blocks may contain zero or more statements. Note that case labels end with a colon (:). The break statement at the end of each block signals the end of a particular case and causes an exit from the switch statement. The default is an optional case. It will execute if the value of the expression does not match with any of the case values.



Rules for switch:

→ The switch expression must be an integral type

→ case labels must be constants (or constant expressions.

→ case labels must be unique.

→ Case labels must be end with colon(:).

→ The break statement transfers the control out of the switch statement.

→ The break statement is optional.

→ The default label is optional. There can be at most one default label.

→ It is permitted to nest switch statements.

The ?: Operator:

The C language has an unusual operator, useful for making two-way decisions. The operator is a combination of ? and :, and takes three operands.

This operator is popularly known as the Conditional operator.

The general form of conditional operator:

$$\boxed{\text{condition expression ? expression 1 : expression 2.}}$$

. The conditional expression is evaluated first. If the result is non-zero, expr-1 . is evaluated and is returned as the value of the conditional expression. Otherwise expr-2 is evaluated as its value is returned.

```
if (x<0)
      flag =0;        can be written as
else
       flag =1;            flag = (x<0) ? 0 : 1 ;
```

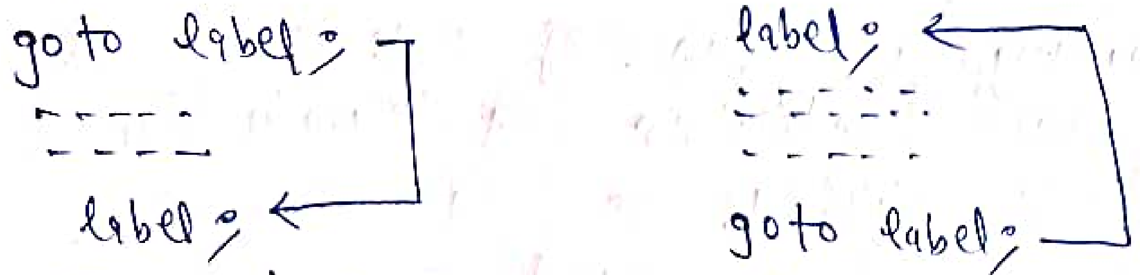
<u>The Go-to statement:</u>

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 general form of goto statement is



Forward jump   Backward jump

Note that a goto breaks the normal sequential execution of the program.

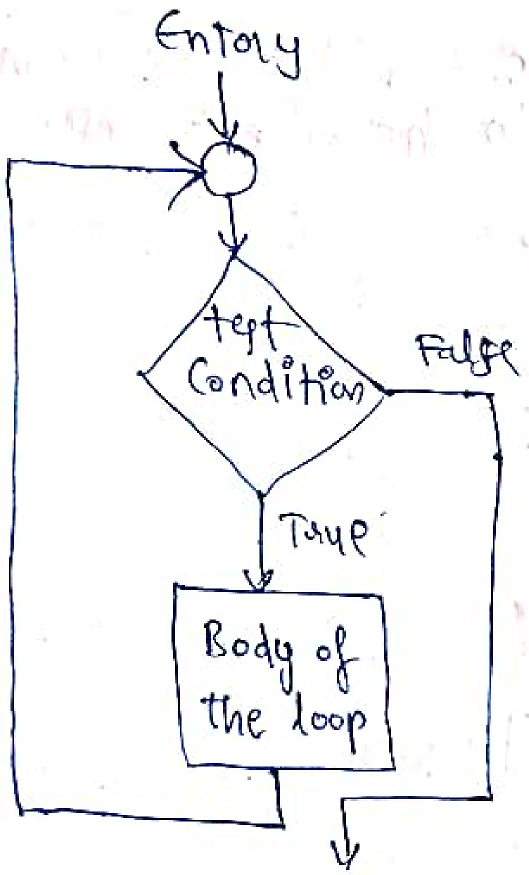
A goto is often used at the end of a program to direct the control go to the input statement, to read further data.

eg:-
```
#include<stdio.h>
void main()
{
    double x,y;
    read:
    scanf("%f", &x);
    if (x<0) goto read;
    y=sqrt(x);
    printf("%f %f\n",x,y);
    goto read;
}
```

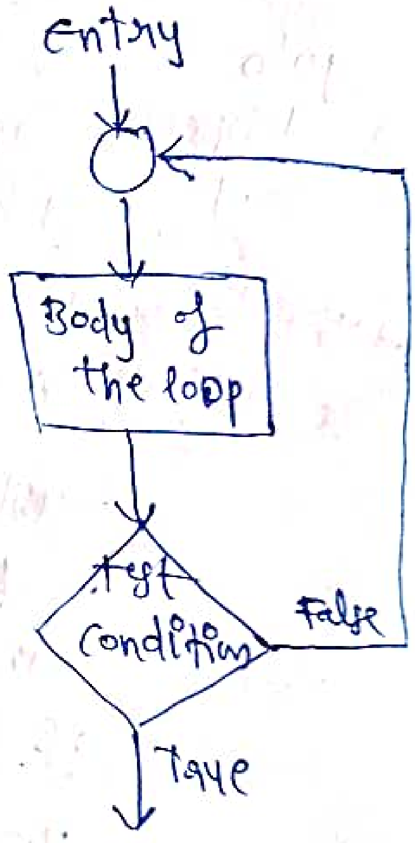
This program puts the computer in a permanent loop known as an infinite loop.

Loops (for, while, do-while) :-

In looping, a sequence of statements are executed until some conditions for termination of the loop are satisfied.
A program loop consists of two segments, one known as the body of the loop and the other known as the control statement.



Entry controlled loop.



Exit controlled loop

The C language provides three constructor for performing loop operations: they are:
→ The while statement
→ The do-while statement
→ The for statement.

Based on the nature of control variable the loops may be classified into following two general categories:

→ Counter-controlled loops

→ Sentinel-controlled loops.

When we know in advance exactly how many times the loop will be executed, we use a counter controlled loop. We use a control variable known as counter.

A counter-controlled loop is sometimes called definite-repetition loop.

In a sentinel-controlled loop, a special value called a sentinel value is used to change the loop control expression from true to false. The control variable is called sentinel variable. A sentinel-controlled loop is often called indefinite repetition loop.'

The while statement:

The simplest of all the looping structures in C is the while statement.

The basic format of the while statement is

```
while ( test condition)
  {
        body of the loop
   }.
```

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.
This process of repeated execution of the body continues until the test-condition finally becomes false and the control is transferred out ~~tof~~ the loop.

<u>The do-while loop:</u>
while loop makes a test of condition before the loop is executed.
The general form of the do-while loop is

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

It ~~might~~ be necessary to execute the body of the loop before the test is performed.
On reaching the do statement, the program proceeds to evaluate the body of the loop first.
At the end of the loop, the test-condition in the while statement is evaluated. This process continuous as long as the condition is true.

When the condition becomes false, the loop will be terminated.
The do-while loop is an exit-controlled loop.

The for statement:

The for loop is another entry-controlled loop that provides a more concise loop control structure.
The general form of the for loop is

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

Initialization of the control variables are known as loop-control variables.
The test-condition is a relational expression. If the condition is true, the body of the loop is executed, otherwise the loop is terminated.
When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop.
The for statement allows for negative increments.
One of the important points about the for loop is that all the three actions, namely initialization, testing, and incrementing, are placed in the for statement itself, thus making them visible to the programmers and users.

eg:- for (n=1 ; n<=10 ; ++n)

```
{
  ______
}
```

```
n=1;
while (n<=10)
  {
   ______
   n=n+1;
  }
```

```
n=1;
do
{ ______
  n=n+1;
}
while (n<=10);
```

If the test-condition is not present, the for statement sets up an 'infinite' loop. Such loops can be broken using break or goto statements in the loop.

· C compiler will not give an error message if we place a semicolon by mistake at the end of a for statement. The semicolon will be considered as a null statement.

Nested of for loops:

Nesting of loops, that is, one for statement within another for statement.

```
for (i=1; i<10; i++)
  {
     for (j=1; j<6 ; j++) ]
     { ______             ] Inner loop   | outer
     }                                   | loop
  }
```

ANSI `C' allows up to 15 levels of nesting.
The outer loop controls the rows, while the inner loops controls the columns.

Selecting a loop:

To choose one of the three ~~loop~~ loop supported by C, we may use the following strategy:

→ Analyse the problem and see whether it required a pre-test or post-test loop.

→ If it requires a post-test loop, then we can use only one loop; do-while.

→ If it requires a pre-test loop, then we have two choices: for and while.

→ Decide whether the loop termination requires counter-based control or sentinel-based control.

→ Use the for loop if the counter-based control is necessary.

→ Use the while loop if the sentinel-based control is required.

C permits a jump from one statement to another within a loop as well as a jump out of a loop.
Exit from a loop can be accomplished by using the break statement or the goto statement.

Break and Continue :- Break :

The break statement is used to terminate loops or to exit from a switch. It can be used within a for, while, do-while, or switch statement.

The break statement is written simply as,

break;

Ex:
```
switch ( choice = toupper(getchar()))
{
    case 'R':
        printf("RED");
        break;
    case 'W':
        printf("WHITE");
        break;
    default:
        printf("ERROR");
        break;
}
```

Notice that each group of statements ends with a break statement, in order to transfer control out of the switch statement.

If a break statement is included in a while, do-while, and for loop, then control will immediately be transferred out of the loop when the break statement is encountered.

eg: while (x<10) {

```
eg: while (x<10) {
        if (x<0) {
            printf("Error");
            break;
        }
    }
```

If the character variable c is assigned an asterisk (*), then the while loop will be terminated.

Continue:

The continue statement is used to bypass the remainder of the current pass through a loop. The loop does not terminate when a continue statement is encountered.

The remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop.

The continue statement can be included within a while, do-while, or a for statement.

It is written simply as,

```
        continue;
```

```
eg: do {
        if (x<0) {
            printf("Error");
            continue;
        }
    } while (x<=100);
```

if program:

```
#include<stdio.h>
int main()
 {
     int number;
     printf(" Enter a number:");
      scanf(" %d", &number);
      if (number >0)
        {
           printf (" The number is positive\n");
         }
     return 0;
 }
```

if-else program:

```
#include<stdio.h>
 int main() {
       int number;
       printf(" Enter a number:");
       scanf(" %d", &number);
       if (number >0) {
           printf(" the number is positive\n");
        } else {
             printf(" the number is negative\n");
         }
     return 0;
 }
```

Switch Program:

```
#include <stdio.h>
int main() {
        char grade;
        printf("Enter your grade (A,B,C,D):");
        scanf("%c", &grade);
        switch(grade) {
                case 'A':
                        printf("Excellent \n");
                        break;
                case 'B':
                        printf("Good \n");
                        break;
                case 'C':
                        printf("Average\n");
                        break;
                case 'D':
                        printf(" Below average \n");
                        break;
                default:
                        printf("Invalid grade \n");
        }
        return 0;
}
```

For loop program:

```
#include<stdio.h>
int main() {
    int i;
    printf("Printing numbers from 1 to 5:\n");
    for(i=1; i<=5; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

While loop program:

```
#include<stdio.h>
int main() {
    int i=1;
    printf("Printing numbers from 1 to 5:\n");
    while (i<=5) {
        printf("%d\n", i);
        i++;
    }
    return 0;
}
```

Do-while program:

```
#include<stdio.h>
int main() {
    int i=1;
    printf("Printing numbers from 1 to 5:\n");
```

```
   do {
      printf("%d\n", i);
       i++;
   } while (i<=5);
   return 0;
}
```

Break Program:

```
#include <stdio.h>
 int main() {
     int i;
     for(i=1; i<=10; i++) {
           if (i==5) {
              printf("Breaking the loop at i=%d\n", i
              break;
           }
           printf("%d\n", i);
     }
    return 0;
}
```

①

Continue Program:

from ① replace it by the following / given code

```
printf(" Skipping iteration at i=%d\n", i);
continue;
```

14:15 PM   Mon-22-Jan
2024
Asked.