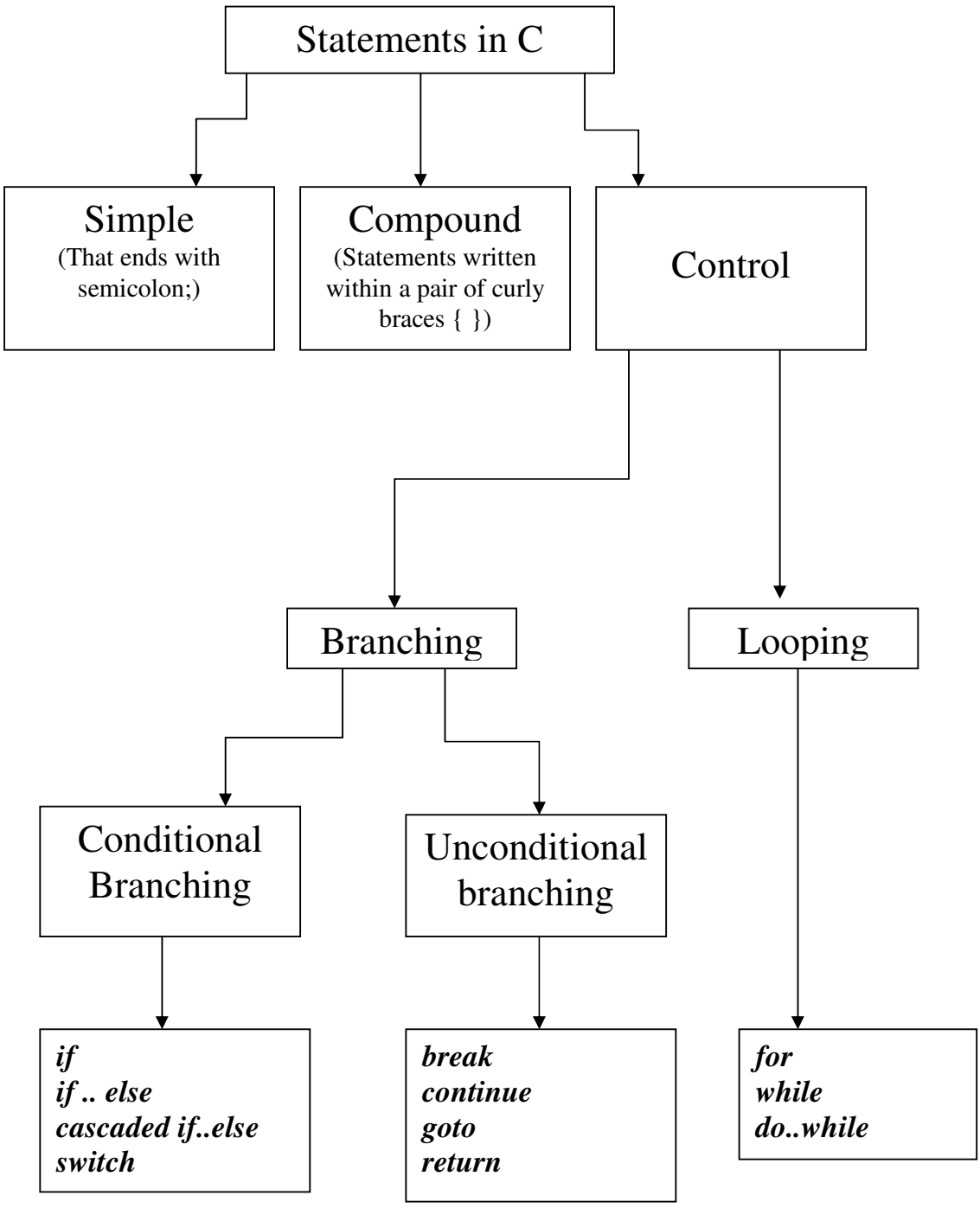
MODULE II

2. BRANCHING AND LOOPING:

2.1 Two way selection (if, if-else, nested if-else, cascaded if-else)
2.2 switch statement
2.3 Ternary operator?
2.4 Loops (for, do-while, while) in C
2.5 break , continue and goto
2.6 Programming examples and exercises.

Statements in C



2.1 Two way selection (if, if-else, nested if-else, cascaded if-else)

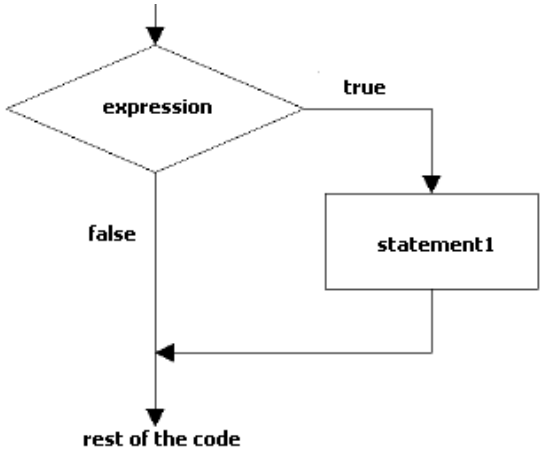
THE if STATEMENT

This is basically a "one-way" decision statement.
This is used when we have only one alternative.

The syntax is shown below:

```
if(expression)
{
        statement1;
}
```

Firstly, the expression is evaluated to *true* or *false*.
If the expression is evaluated to *true*, then statement1 is executed.
If the expression is evaluated to *false*, then statement1 is skipped.
The flow diagram is shown below:



Example: Program to illustrate the use of if statement.

```
#include<stdio.h>
void main()
{
     int n;
     printf("Enter any non zero integer: \n") ;
     scanf("%d", &n)
     if(n>0)
          printf("Number is positive number ");
     if(n<0)
          printf("Number is negative number ");
}

Output:
Enter any non zero integer:
7
Number is positive number
```

THE if else STATEMENT

This is basically a "two-way" decision statement.

This is used when we must choose between two alternatives.

The syntax is shown below:

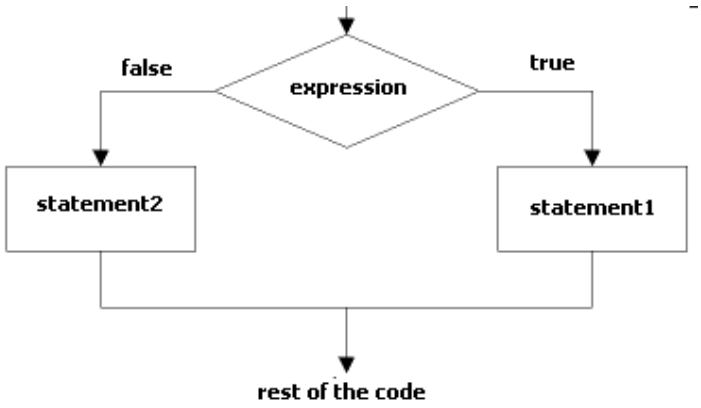
```
if(expression)
{
        statement1;
}
else
{
        statement2;
}
```

Firstly, the expression is evaluated to true or false.

If the expression is evaluated to *true*, then statement1 is executed.

If the expression is evaluated to *false*, then statement2 is executed.

The flow diagram is shown below:



Example: Program to illustrate the use of if else statement.

```
#include<stdio.h>
void main()
{
     int n;
     printf("Enter any non-zero integer: \n") ;
     scanf("%d", &n)
     if(n>0)
          printf("Number is positive number");
     else
          printf("Number is negative number");
}

Output:
Enter any non-zero integer:
7
Number is positive number
```

THE nested if STATEMENT

An if-else statement within another if-else statement is called nested if statement.

This is used when an action has to be performed based on many decisions. Hence, it is called as multi-way decision statement.

The syntax is shown below:

```
if(expr1)
{
        if(expr2)
                statement1
        else
                statement2
}
else
{
        if(expr3)
                statement3
        else
                statement4
}
```

• Here, firstly expr1 is evaluated to true or false.

If the expr1 is evaluated to *true*, then expr2 is evaluated to true or false.

If the expr2 is evaluated to *true*, then statement1 is executed.

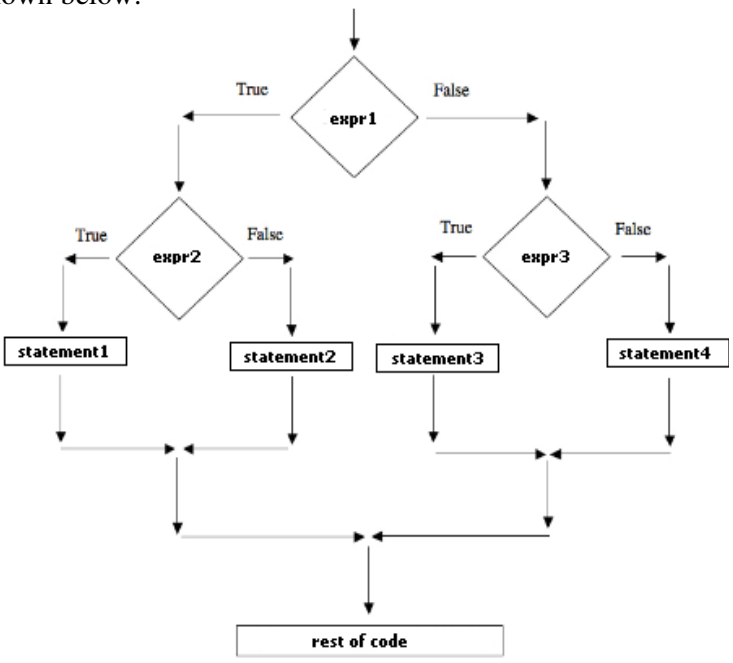
If the expr2 is evaluated to *false*, then statement2 is executed.

If the expr1 is evaluated to *false*, then expr3 is evaluated to true or false.

If the expr3 is evaluated to *true*, then statement3 is executed.

If the expr3 is evaluated to *false*, then statement4 is executed.

• The flow diagram is shown below:



Example: Program to select and print the largest of the 3 numbers using nested "if-else" statements.

```
#include<stdio.h>
void main()
{
     int a,b,c;
     printf("Enter Three Values: \n");
     scanf("%d %d %d ", &a, &b, &c);
     printf("Largest Value is: ") ;
     if(a>b)
     {
          if(a>c)
               printf(" %d ", a);
          else
               printf(" %d ", c);
     }
     else
     {
          if(b>c)
               printf(" %d", b);
          else
               printf(" %d", c);
     }
}
Output:
Enter Three Values:
7 8 6
Largest Value is: 8
```

THE CASCADED if-else STATEMENT (THE else if LADDER STATEMENT)

This is basically a "multi-way" decision statement.

This is used when we must choose among many alternatives.

The syntax is shown below:

```
if(expression1)
{
     statement1;
}
else if(expression2)
{
     statement2;
}
else if(expression3)
{
     statement3
}
```

```
else
{
        default statement
}
```

The expressions are evaluated in order (i.e. top to bottom).
If an expression is evaluated to true, then
→ Statement associated with the expression is executed &
→ Control comes out of the entire else if ladder
For ex, if exprression1 is evaluated to *true*, then statement1 is executed.
If all the expressions are evaluated to false, the last statement4 (default case) is executed.

2.2 *switch* Statement

This is a multi-branch statement similar to the if - else ladder (with limitations) but clearer and easier to code.

Syntax :

```
switch ( expression )
{
     case constant1 :   statement1 ;
                break ;

     case constant2 :   statement2 ;
                break ;

     ...

     default :   statement ;
}
```

The value of expression is tested for equality against the values of each of the constants specified in the **case** statements in the order written until a match is found. The statements associated with that case statement are then executed until a break statement or the end of the switch statement is encountered.

When a break statement is encountered execution jumps to the statement immediately following the switch statement.

The default section is optional -- if it is not included the default is that nothing happens and execution simply falls through the end of the switch statement.

The switch statement however is limited by the following

- Can only test for equality with **integer constants** in case statements.
- No two case statement constants may be the same.
- Character constants are automatically converted to integer.

For Example :- Program to simulate a basic calculator.

```
#include <stdio.h>
void main()
{
  double num1, num2, result ;
  char op ;

  printf ( " Enter number operator number\n" ) ;
  scanf ("%f %c %f", &num1, &op, &num2 ) ;
  switch ( op )
  {
     case '+' :  result = num1 + num2 ;
                        break ;
     case '-' :   result = num1 - num2 ;
                        break ;
     case '*' :  result = num1 * num2 ;
                        break ;
     case '/' :  if ( num2 != 0.0 ) {
                          result = num1 / num2 ;
                          break ;
                          }
     // else we allow to fall through for error message

     default :  printf ("ERROR -- Invalid operation or division
                           by 0.0" ) ;
  }
    printf( "%f %c %f = %f\n", num1, op, num2, result) ;
}
```

Note : The break statement need not be included at the end of the case statement body if it is logically correct for execution to fall through to the next case statement (as in the case of division by 0.0) or to the end of the switch statement (as in the case of default :).

2.3 Ternary operator- ? :

This is a special shorthand operator in C and replaces the following segment

```
if ( condition )
     expr_1 ;
else
     expr_2 ;
```

with the more elegant:

Syntax: ***condition ? expr_1 : expr_2 ;***

The ?: operator is a ternary operator in that it requires three arguments. One of the advantages of the ?: operator is that it reduces simple conditions to one simple line of code which can be thrown unobtrusively into a larger section of code.

For Example :- to get the maximum of two integers, x and y, storing the larger in max.

```
max =  x >= y  ? x  :  y  ;
```

The alternative to this could be as follows

```
if ( x > = y  )
     max = x ;
else
     max = y ;
```

giving the same result but the former is a little bit more succinct.

2.4 Loops (*for, do-while, while*) in C

for statement

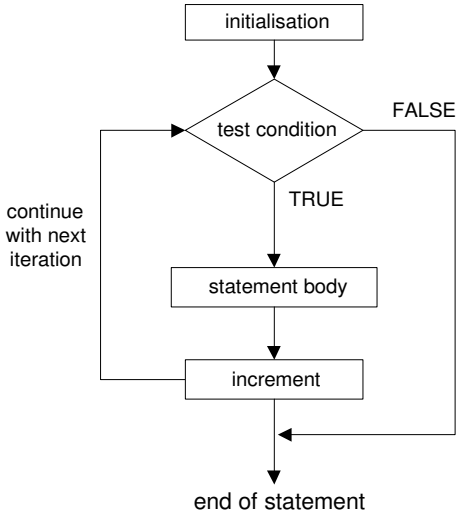
The ***for*** statement is most often used in situations where the programmer knows in advance how many times a particular set of statements are to be repeated. The for statement is sometimes termed a counted loop.

Syntax :

```
for ( [initialisation] ; [condition] ; [increment] )
{
     [statement body] ;
}
```

initialisation :- this is usually an assignment to set a loop counter variable for example.
condition :- determines when loop will terminate.
increment :- defines how the loop control variable will change each time the loop is executed.
statement body :- can be a single statement, no statement or a block of statements.

The *for* statement executes as follows :-



Note : The square braces above are to denote optional sections in the syntax but are not part of the syntax. The semi-colons must be present in the syntax.

For Example : Program to print out all numbers from 1 to 100.

```
#include <stdio.h>

void main()
{
  int x ;

  for ( x = 1;  x <= 100; x++ )
     printf( "%d\n", x ) ;
}
```

Curly braces are used in C to denote code blocks whether in a function as in main() or as the body of a loop.

For Example :- To print out all numbers from 1 to 100 and calculate their sum.

```
#include <stdio.h>

void main()
{
  int x, sum = 0 ;

  for ( x = 1; x <= 100; x++ )
  {
     printf( "%d\n", x ) ;
     sum += x ;
  }
  printf( "\n\nSum is %d\n", sum ) ;
}
```

Multiple Initialisations

C has a special operator called the **comma operator** which allows separate expressions to be tied together into one statement.

For example it may be tidier to initialise two variables in a for loop as follows :-

```
for ( x = 0, sum = 0; x <= 100; x++ )
     {
       printf( "%d\n", x) ;
       sum += x ;
     }
```

Any of the four sections associated with a for loop may be omitted but the semi-colons must be present always.

For Example :-

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

An infinite loop may be created as follows

```
for ( ;  ;  )
     statement body ;
```

or indeed by having a faulty terminating condition.

Sometimes a for statement may not even have a body to execute as in the following example where we just want to create a time delay.

```
for ( t = 0; t < big_num ; t++ )  ;
```

or we could rewrite the example given above as follows

```
for ( x = 1; x <= 100; printf( "%d\n", x++ ) ) ;
```

The initialisation, condition and increment sections of the for statement can contain any valid C expressions.

```
for ( x = 12 * 4 ; x < 34 / 2 * 47 ; x += 10 )
    printf( "%d ", x ) ;
```

It is possible to build a nested structure of for loops, for example the following creates a large time delay using just integer variables.

```
unsigned int x, y ;

for ( x = 0; x < 65535; x++ )
     for ( y = 0; y < 65535; y++ ) ;
```

For Example :
Program to produce the following table of values

```
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
```

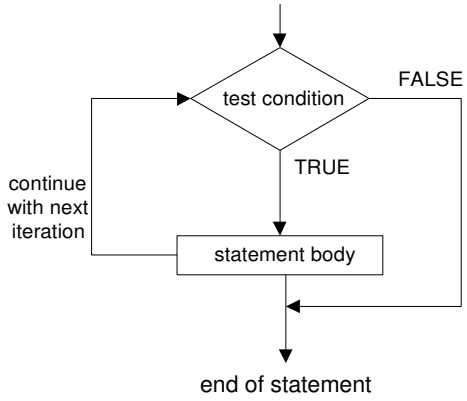
```
#include <stdio.h>

void main()
{
   int j, k ;

   for ( j = 1; j <= 5; j++ )
   {
     for ( k = j ; k < j + 5; k++ )
      {
        printf( "%d  ", k ) ;
     }
     printf( "\n" ) ;
   }
}
```

while statement

The ***while*** statement is typically used in situations where it is not known in advance how many iterations are required.

Syntax :

```
while ( condition )
{
     statement body ;
}
```



For Example : Program to sum all integers from 100 down to 1.

```
#include <stdio.h>
void main()
{
  int sum = 0, i = 100 ;

  while ( i )
     sum += i-- ;// note the use of postfix decrement operator!
  printf( "Sum is %d \n", sum ) ;
}
```

where it should be recalled that any non-zero value is deemed TRUE in the condition section of the statement.

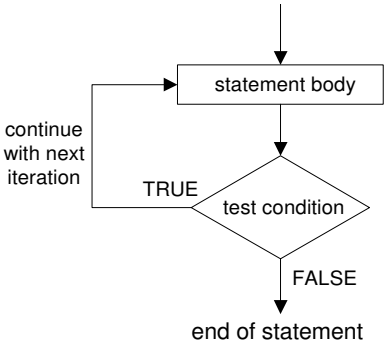
do while

The terminating condition in the for and while loops is always tested before the body of the loop is executed -- so of course the body of the loop may not be executed at all.

In the ***do while*** statement on the other hand the statement body is always executed at least once as the condition is tested at the end of the body of the loop.

Syntax :

```
do
{
     statement body ;
}  while ( condition ) ;
```



For Example : To read in a number from the keyboard until a value in the range 1 to 10 is entered.

```
int i ;

do
{
  scanf( "%d\n", &i ) ;
}   while ( i < 1 && i > 10 ) ;
```

In this case we know at least one number is required to be read so the do-while might be the natural choice over a normal while loop.

2.5 break , continue and goto

break statement

When a ***break*** statement is encountered inside a while, for, do/while or switch statement the statement is immediately terminated and execution resumes at the next statement following the statement.

For Example :-

```
          …
for ( x = 1 ; x <= 10  ; x++  )
{
     if ( x > 4  )
          break ;

     printf( “%d  “ , x ) ;
}
printf( "Next executed\n" );//Output : “1   2   3   4   Next
Executed”
          …
```

continue statement

The ***continue*** statement terminates the current iteration of a while, for or do/while statement and resumes execution back at the beginning of the loop body with the next iteration.
For Example :-

```
          ...
for ( x = 1; x <= 5; x++ )
{
     if ( x == 3 )
          continue ;
     printf( "%d  ", x ) ;
}
printf( "Finished Loop\n" ) ; // Output : "1 2 4 5 Finished Loop"
          ...
```

goto statement

goto statement can be used 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.

The syntax is shown below:

Example: Program to detect the entered number as to whether it is even or odd. Use goto statement.

```
#include<stdio.h>
void main()
{
     int x;
     printf("Enter a Number: \n");
     scanf("%d", &x);
     if(x % 2 = = 0)
          goto even;
     else
          goto odd;
 even:printf("%d is Even Number");
          return;
 odd:printf(" %d is Odd Number");
}
Output:
Enter a Number:
5
5 is Odd Number.
```

2.6 Programming examples and exercises.

1. Write a program to find the roots of a user specified quadratic equation.

Recall the roots of $ax^2 + bx + c = 0$ are

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The user should be informed if the specified quadratic is valid or not and should be informed how many roots it has, if it has equal or approximately equal roots ($b^2 == 4ac$), if the roots are real ($b^2 - 4ac > 0$) or if the roots are imaginary ($b^2 - 4ac < 0$). In the case of imaginary roots the value should be presented in the form (x + i y).

2. Write a program that allows the user to read a user specified number of double precision floating point numbers from the keyboard. Your program should calculate the sum and the average of the numbers input. Try and ensure that any erroneous input is refused by your program, e.g. inadvertently entering a non-numeric character etc.

3. Write a program to print out all the Fibonacci numbers using **short** integer variables until the numbers become too large to be stored in a short integer variable i.e. until overflow occurs.

a. Use a for loop construction.
b. Use a while loop construction.

Which construction is most suitable ?
Note: Fibonacci numbers are (0, 1,1,2,3,5,8,13,...

4. Write a program which simulates the action of a simple calculator. The program should take as input two integer numbers then a character which is one of +,-,*,/,%. The numbers should be then processed according to the operator input and the result printed out. Your program should correctly intercept any possible erroneous situations such as invalid operations, integer overflow, and division by zero.