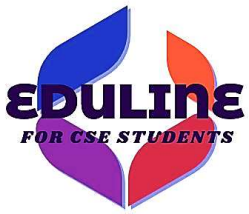
MODULE 2

PROGRAM BASICS

CO - Students will be able to describe the basic concepts of C program with the help of simple examples.



Prepared By Mr. EBIN PM, AP, IESCE 1

BASIC STRUCTURE OF C PROGRAM

- The set of instructions that are provided to computer is known as a program and the development of program is known as programming.
- Programming is a problem-solving activity.
- **C** was evolved from ALGOL, BCPL (Basic Combined Programming Language) and B by **Dennis Ritchie**.
- **C** is structured, high level, machine independent language
- **C** is a very powerful and widely used language. It is used in many scientific programming situations.
- It forms (or is the basis for) the core of the modern languages Java and C++. It allows you access to the bare bones of your computer.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 2

❖STRUCTURE OF A C PROGRAM

DOCUMENTATION SECTION	comments
LINKAGE SECTION	#include files
DEFINITION SECTION	#define
GLOBAL DECLARATION SECTION	variable and functions
MAIN FUNCTION SECTION **Local Declaration Part** **Executable Code Part**	main ( ) function
SUB PROGRAM SECTION **Function 1 ( )** **Function 2 ( )** **……** **Function N ( )**	function definition Section

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 3

❖ Points to remember

- Every statement in **C** should end with a semicolon.
- In **C** everything is written in lower case. However upper case letters used for **symbolic names** representing constants.
- **#include** is a pre-processor directive.
- **Stdio.h** is a header file, for standard input output functions. It activate keyboard and monitor.
- Single line **comment** is represented using **//**
- Multiple line **comment** is represented using **/*……*/**
- Comment lines are not executable statements and therefore anything between/*and*/ is ignored by the compiler.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 4

- Execution begins from **main ( )**
- main has no arguments(or Parameters)
- Every program must have exactly one main function. **C** permits different forms of main statements. They are

  main ( )

  int main ( )

  void main ( )

  main (void)

  void main (void)

  int main (void)

C PROGRAM TOKENS

➢The smallest individual unit in a program is known as **tokens**. **C** has the following tokens

1. **Keywords:** these are the words whose meaning has already been explained to the **C** compiler. The keywords cannot be used as variable names. The keywords are also known as "**Reserved words**". There are only **32** keywords in **C.** Examples are char, double, void, if
2. **Identifiers:** These are fundamental building block of a program. Identifiers refer to the names of variables; functions and arrays. These are user-defined names and consist of a sequence of letters and digits with the letter as a first character.

3. **Constants (literals):** These are data items that never change their value during a program run. These are fixed values. The types of constants are

a) **Integer constant:** These are whole numbers without any fractional part. It must have at least one digit and must not contain any decimal point. It can be either positive or negative. Examples are 426,+200,-760

b) **Character constant:** A character constant is one character enclosed in single quotes. Examples are 'A', '5' ,'=' . A character constant have corresponding **ASCII** (American Standard Code for Information Interchange) values. For example **ASCII** value of 'A' is 65 and **ASCII** value of 'a' is 97.

c) **String constant:** Multiple character constants are treated as string constant. A string constant is a sequence of characters surrounded by double quotes. Examples are "abcd", "seena". Each string constant is by default (automatically) added with a special character **'\0'** which makes the end of a string. Thus the size of a string is

Number of characters + null character ('\0')

- For example "abc" size is 4. Thus "abc" will be automatically represented as "abc\0" in the memory.'\0' is an **end-of-string marker**

d) **Floating constant:** Floating constants are also called real constants. These numbers have fractional part. These may be written in one of the two forms called

➢**Fractional form:** A floating constant in fractional form must have at least one digit before a decimal point and at least one digit after the decimal point.

- It may also have either positive or negative. Examples are 17.8,-13.867

➢**Exponent form:** A real constant in exponent form consist of two parts. (a) Mantissa (b) Exponent.

- For example, 9.8 can be written as 0.98×10^{1}=0.98E01 where mantissa part is 0.98 and exponent part is 1. E01 represent 10^{1}. The exponent must be an integer. The examples are 152E05,1.52E07

4. Punctuators :

[ ] Bracket (Arrays)
( ) Parentheses (Function call & Function parameters)
{ } Braces (Opening and closing of executable statements)
, Comma (Separator)
; Semicolon (Terminator)
: Colon (Labelled statement)
***** Asterisk (Pointer declaration & Multiplication operator)
# Pound sign OR called Hash (Preprocessor directive)
= Equal to (Assignment operator)

5. Operators: There are two divisions of operators. (a) Unary operators (b) Binary operators. **Unary operators** have one operator to operate up on. They are

- **&** Address operator
- **+** Unary plus
- **-** Unary minus
- **~** Bitwise complement
- **++** Increment
- **- -** decrement
- **!** Logical negation

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 11

➢ **Binary operators** require two operands to operate up on. They are

ARITHMETIC OPERATORS

- **+** Addition
- **-** Subtraction
- ***** Multiplication
- **/** Division
- **%** remainder (modulo division)

SHIFT OPERATORS

- **<<** Left shift
- **>>** Right shift

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 12

BITWISE OPERATORS

- `&` Bitwise AND
- `|` Bitwise OR
- `^` Bitwise XOR

LOGICAL OPERATORS

- `&&` Logical AND
- `||` Logical OR
- `!` Logical NOT

ASSIGNMENT OPERATORS

- `=` Assignment
- `*=` Assign product
- `/=` Assign quotient
- `+=` Assign sum

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 13

RELATIONAL OPERATORS

- `<` Less than
- `>` Greater than
- `<=` Less than or equal to
- `>=` Greater than or equal to
- `==` Equal to
- `!=` Not equal to

COMPONENT SELECTION

- `.` Direct component selector
- `→` Indirect component selector

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 14

❖ CONDITIONAL OPERATOR(TERNARY OPERATOR)

? True form : false form

Consider the example

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

In this example x will be assigned the value of b. This can be achieved using the **if….else** statement as follows

```
if (a>b)
   x=a;
else
   x=b;
```


❖ INCREMENT AND DECREMENT OPERATORS

Increment ++

Decrement - -

a= a+1 is same as **++a** or **a++**

a= a-1 is same as **- - a** or **a- -**

- The operator ++ adds 1 to its operand and - - subtracts one. The prefix version comes before the operand (as in ++a or - -a) and the postfix version comes after the operand (as in a++ or a- -).
- The prefix increment or decrement operator follows **Change- Then – Use rule.**
- Postfix increment or decrement operator follows **Use- Then- Change rule.**

❖ PRECEDENCE OF OPERATORS (HIERARCHY OF OPERATIONS)

- While executing an arithmetic statement, which has two or more operators, we may have some problems as to how exactly does it get executed.
- For example, does the expression 2 * x - 3 * y correspond to (2x)-(3y) or to 2(x-3y)? Similarly, does A / B * C correspond to A / (B * C) or to (A / B) * C? To answer these questions satisfactorily one has to understand the 'hierarchy' of operations.
- The priority or precedence in which the operations in an arithmetic statement are performed is called the hierarchy of operations.
- The hierarchy of commonly used operators is shown below.

Priority	Operators	Description
1st	* / %	multiplication, division, modular division
2nd	+ -	addition, subtraction
3rd	=	assignment

- If there are more than one set of parentheses, the operations within the innermost parentheses would be performed first, followed by the operations within the second innermost pair and so on.
- We must always remember to use pairs of parentheses.

Eg: Determine the hierarchy of operations and evaluate the following expression?

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8 Stepwise evaluation of this expression is shown below:

i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8	operation: *
i = 1 + 4 / 4 + 8 - 2 + 5 / 8	operation: /
i = 1 + 1 + 8 - 2 + 5 / 8	operation: /
i = 1 + 1 + 8 - 2 + 0	operation: /
i = 2 + 8 - 2 + 0	operation: +
i = 10 - 2 + 0	operation: +
i = 8 + 0	operation : -
i = 8	operation: +

- Note that 6 / 4 give 1 and not 1.5. This so happens because 6 and 4 both are integers and therefore would evaluate to only an integer constant. Similarly 5 / 8 evaluates to zero, since 5 and 8 are integer constants and hence must return an integer value

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 19

Algebraic Expression	C Expression
a x b – c xd	a * b – c * d
(m + n) (a + b)	(m + n) * (a + b)
$3x^2 + 2x + 5$	3 * x * x + 2 * x + 5
$\frac{a + b + c}{d + e}$	(a + b + c) / (d + e)

Let us consider the following program which add two numbers and print the result. The following program uses the tokens which we discussed above and you can see the structure of the normal C program also

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 20

```
/* A program to add two numbers and print the result*/
    #include <stdio.h>          /*header files*/
    #include <conio.h>
    void main ( )           /* main function*/
    {
       int x, y, sum;         /* variable declaration*/
       printf ("enter two integer numbers");      /*printing statement*/
       scanf ("%d%d",&x,&y);          /* scanning the input*/
       sum =x+y;                          /* calculation*/
       printf ("The sum is%d",sum);      /* printing output*/
       getch ( );
    }
```


DIFFERENCE BETWEEN = AND = =

- The expression **a= = 5** is test or check whether a is equal to 5
- The expression **a=5** is assign 5 to the variable a

```
    #include <stdio.h>
    #include <conio.h>
     void main ( )
     {
       int p=8;   /*variable initialization*/
       if (p = = 8)    /* check whether p is equal to 8*/
       printf ("Both are same");
       getch ( );
     }
```

Output

Both are same

SHORT HANDS (COMPOUND ASSIGNMENT)

a=a+10; can be written as a+=10;

Syntax

Variable Operator = Expression;

DATA TYPES

➢Data type is a classification identifying one of various types of data. There are three classes of data types.

- **Fundamental (primary) Data types**
- **Derived Data types**
- **User defined data types**

❖Fundamental (primary) data types

1. **int :** for integers (2 byte memory space allocates in memory)
2. **char :** for characters(1 byte memory space allocates in memory)
3. **float :** for floating point numbers(4 byte memory space allocates in memory)
4. **double :** for double precision floating point numbers(8 byte memory space allocates in memory)
5. **void:** for empty set of values and non-returning functions. The void type has no value.

❖Derived data types

➢Derived data types are constructed from fundamental data types. They are

1. **Arrays:** An array is a collection of variables of the same type that are referenced by a common name.
2. **Functions:** A function is a named part of a program that can be invoked from other part of the program.
3. **Pointer:** A pointer is a variable that holds the memory address. This address is usually the location of another variable in memory.
4. **Constant:** The keyword **const** can be added to the declaration of an object to make that object a constant rather than a variable. Thus, the variable of the named constant cannot be altered during the program run.

Syntax

const type name = value;

Eg: const int a=10;

❖User defined data types

1. **typedef:** It allows users to define an identifier that would represent an existing data type.

Syntax

typedef type identifier;

Here, **type** refers to an existing data type and **identifier** refers to the new name given to the data type.

Eg: typedef int units;

typedef float marks;

Here, **units** symbolize **int** and **marks** symbolizes **float**. They can be later used to declare variables as follows

units batch1, batch2;

marks name1,name2;

- **batch1** and **batch2** are declared as **int** variables and **name1** and **name2** are declared as **floating point** variables.
- Advantage of typedef is that we can create meaningful data type names for increasing the readability of the program.

2. **Structure:** A structure is a collection of variables of different data types referenced under one name.
3. **Enumeration:** Enumerated data type (also called enumeration or enum) provides a way for attaching names to numbers there by increasing comprehensibility of the code. It has the following form.

enum identifier {value1,value2,…….,value n};

- The **identifier** is a user defined enumerated data type which can be used to declare variables that can have one of the values enclosed with in the braces (known as enumeration constant).

Eg: enum week_day {Monday, Tuesday...Sunday}; // an enumerated data type week_day has been defined

week_day day1, day2; **//** variables created of type week_day.

day1=Wednesday; **//**correct,day1 can have any of the above given 7 values.

day2=7; **//** incorrect, no other value can be assigned

- **Enumerated means all the values are listed.** The enum specifier automatically enumerates a list of words by assigning them values 0, 1, 2, 3, and so on.
- The compiler automatically assigns integer digits beginning with 0 to all the enumeration constants. That is the enumeration constant Monday is assigned 0, Tuesday is assigned 1 and so on.
- We can give values explicitly (changing default original values) by the following way

enum day {Monday=1, Tuesday, Wednesday=6, Thursday, Friday….};

Now, Monday=1, Tuesday=2, Wednesday=6, Thursday=7, Friday=8 etc.

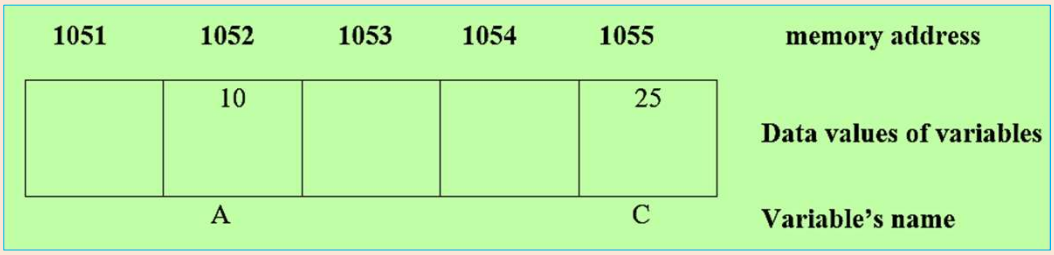
4. Union:

- Structure and union are same but different in memory allocation.
- A union can be declared using the keyword is **union**.
- A union is a memory location that is shared by two or more different variables, generally of different types at different times.
- Defining a union is similar to defining a structure.

VARIABLES

- **A variable is a data name that may be used to store a data value.** A variable may take different values at different times during execution.
- Each variable has a specific storage location in memory where its value is stored. The variables are called symbolic variables because these are named.
- There are two values associated with a symbolic variable.
- **Data value:** stored at some location in memory. This is sometimes referred to as a variable's **r- value.**
- **Location value:** This is the address in memory at which its data value is stored. This is sometimes referred to as variable's **l-value.**

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 31



- **r-value** of A=10 and **l-value** of A=1052
- **r-value** of c=25 and **l-value** of c=1055
- Whenever we use the assignment operator, the expression to the left of an assignment operator must be an l-value. That is, it must provide an accessible memory address where the data can be written to.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 32

❖Variable Declaration

➢A declaration associates a group of variables with a specific data type. Declaration of variables must be done before they are used in the program.

Syntax

Data_ type variable_ name;

Eg: int a;

float a, b, c;

char name [10]; // Character array declaration

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 33

❖Variable Initialization

- The process of giving initial values to variables is called initialization.

Eg: int x=10;

float n=22.889;

char answer='y';

❖Expression

- An expression represents a single data item, such as number or a character. Expression can also represent logical conditions

Eg: x+y

y=z

x=y+z

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 34

- An expression statement consists of an expression followed by a semicolon. For example, following two expression statements cause the value of the expression on the right of the equal sign to be assigned to the variable on the left.

```
x=5;
x=y+z;
```

- A compound statement consists of several individual statements enclosed within a pair of braces ({and}).

ESCAPE SEQUENCES (BACK SLASH) CHARACTERS

➢In character constants, their exist nongraphic characters. Non graphics characters cannot be typed directly from keyboard. These are represented using escape sequences. An escape sequence is represented by a back slash (\).

Esc. Seq.	Purpose	Esc. Seq.	Purpose
\n	New line	\t	Horizontal Tab
\b	Backspace	\r	Carriage return
\f	Form feed	\a	Alert
\'	Single quote	\"	Double quote
\\	Backslash	\?	Question mark

Eg: C program to illustrate \n escape sequence

```
#include <stdio.h>
void main ()
   {
       //we are using \n, which
       // is a new line character.
       printf("Hello\n");
       printf("C programming");
       getch();
   }
```

Output

Hello
C programming

TYPE CASTING (TYPE CONVERSION)

➢The process of converting one predefined type in to another is called type casting. It has two forms.

1. **Implicit Type casting:** It performed by the compiler. It is used in mixed mode operations.

- **RULES**
- An arithmetic operation between an integer and integer always yields an integer result.
- An operation between a real (float) and real operation is performed. Hence the result is real.

- An operation between an integer and real always yields a real result.
- In this operation the integer is first promoted to a real and then the operation is performed. Hence the result is real.

2. Explicit Type casting: It is user defined

Syntax

(type) expression;

Eg: b= (float) (x+y/2);


```
#include <stdio.h>
#include<conio.h>
void main ( )
{
  int x, y,
  float sum;
  printf (“enter two integer numbers”);
  scanf (“%d%d”,&x,&y);
  sum =(float) x+y;
  printf (“The sum is %f”,sum);
  getch ( );
}
```

- Consider the following assignment statements.

```
int k =4.8;
float m = 50;
```

- Here in the first assignment statement though the expression's value is a **float** (4.8) it cannot be stored in **k** since it is an **int**. In such a case the **float** is demoted to an **int** and then its value is stored.
- Hence what gets stored in **k** is 4. Exactly opposite happens in the next statement.
- Here, 50 is promoted to 50.000000 and then stored in **m**, since **m** being a **float** variable cannot hold anything except a **float** value.

- Consider the following program fragment.

```
float x, y, z ;
int p ;
p = x * y * z / 67 + 89 / 8 - 2 * 4.8;
```

- Here, in the assignment statement some operands are **int**s whereas others are **float**s.
- As we know, during evaluation of the expression the **int**s would be promoted to **float**s and the result of the expression would be a **float**.
- But when this **float** value is assigned to **p,** it is again demoted to an **int** and then stored in **p**.

PREPROCESSOR DIRECTIVES

- The pre-processor is a program that processes the source code before it passes through the compiler. It operates under the control of directives.
- Before the source code passes through the compiler, it is examined by the pre-processor for any pre-processor directives.
- If there are any, appropriate actions are taken and then the source program is handed over to the compiler. The three categories of directives are

1. **Macro substitution directives (#define)**
2. **File inclusion directives (#include)**
3. **Compiler control directives (#if, #else, #endif)**

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 43

- **#define**: defines a macro substitution.
- **#undef:** undefines a macro
- **#include:** Specifies the files to be included.
- **#if:** Test a compile-time condition

Syntax

#include<file name>

Eg: #include<stdio.h>
#include<conio.h>
#include<math.h>

#include<file name> : The file is searched only in the standard directory

#include "file name" : The file searching is made first in the current directory and then in the standard directories.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 44

MACRO SUBSTITUTION

➢We can define a symbolic constant using **#define**. It has the following form

Syntax

#define symbolic_ name value_ of_ constant

Eg: #define NUMVAL 100

➢ #define statement must not end with semi colon. Symbolic names are written in **CAPITALS.**

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 45

Eg: #include<stdio.h>

```
#define VALUE 25
void main( )
  {
   int i ;
   for ( i = 1 ; i <= VALUE ; i++ )
   printf ( "\n%d", i ) ;
  }
```

➢In this program, instead of writing 25 in the for loop we are writing it in the form of VALUE, which has already been defined before main ( ) through the statement, #define VALUE 25

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 46

➢A **#define** directive could be used even to replace a condition, as shown below.

```
# include<stdio.h>
#define AND &&
#define NEW ( a > 25 AND a < 50 )
void  main( )
 {
    int a = 30 ;
    if ( NEW )
       printf ( "Am your friend" ) ;
    else
       printf ( "Am not your friend" ) ;
}
```


VARIOUS INPUT/OUTPUT FUNCTIONS

1. getchar ( )

- getchar () function reads a single character from standard input. It takes no parameters and its returned value is the input character. It has the following form

variable name=getchar( );

Eg:
```
char c;
printf("Enter a character");
c=getchar ( );
```

- The second line causes a single character to be entered from the standard input device and then assigned to c. The variable name has been declared as 'char' type.

2. putchar( )

- It displays a single character on the screen. This function takes one argument, which is the character to be sent. It also returns this character as its result. The general form is

  putchar (variable-name);

Eg: char ans='y'

putchar (ans);

3. getche ( )

- The 'e' in getche () function means it echoes (displays) the character.

4. gets( ) : receives a string from the keyboard.

5. puts ( ) : Outputs a string to the screen

Eg: char vehicle [40];

Printf("Enter your vehicle name");

gets (vehicle);

puts (vehicle);

- These lines use the gets and puts to transfer the line of text into and out of the computer.
- When this program is executed, it will give the same result as that with scanf and printf function for input and output of given variable or array.

6. **clrscr ( )** : It is a clear screen function.

7. **printf ( )** : For outputting the result we use printf( ) function.

Syntax

printf ("formatted string", variable);

Eg: printf ("%d", a);

printf ("WELCOME MY FRIEND");

➢ **Formatted strings are**

1. **%d** integers
2. **%f** float
3. **%c** character
4. **%o** octal number
5. **%s** string
6. **%e** exponential notation
7. **%u** unsigned integer
8. **%x** hexadecimal integer
9. **% i** signed decimal integer
10. **%p** display a pointer
11. **%%** prints a percent sign (%)

8. scanf ( ) : It can read data from keyboard. scanf ( ) means "**scan formatted**" .

Syntax

scanf ("formatted string", addressed variable);

Eg: scanf ("%d", &a);

scanf ("%d%f",&num1,&num2);

CONTROL FLOW STATEMENTS

- **C** program is a set of statements which are normally executed sequentially in the order.
- We can change the order of execution of statements.
- For this purpose, we use branching instructions (**Decision-making statements**).
- Since these statements control the flow of execution, they also known as **Branching instructions**.

1. SIMPLE IF STATEMENT

Syntax

```
if (test expression)
    do this;
```

OR

```
if (test expression)
  {
  do this;
  and this;
  }
  statement-x;
```

- 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 **statement-x.**
- If multiple statements are executed, then they must be placed with in a pair of braces.

Eg: /* check a citizen is eligible for voting */

```
#include<stdio.h>
#include<conio.h>
void  main()
 {
   int age;
   printf ("Enter the age : ");
   scanf ("%d",&age);
   if(age >= 18)
   printf("Eligible for voting…");
   getch();
 }
```

OUTPUT

Enter the age:

21

Eligible for voting…

2. IF…..ELSE STATEMENT

- Here, either true block or false block will be executed, not both. In both cases control is transferred to the statement-x.

Syntax

```
if (test expression)
    {
      True  statements;
    }
else
   {
     False statements;
   }
Statement-x;
```


Eg:

```
/* print a number is even or odd */
#include<stdio.h>
#include<conio.h>
void  main ( )
 {
   int number;
   printf (“Enter a number : “);
   scanf (“%d”, &number);
   if ((number %2) = = 0)
        printf (“%d is even number.”, number);
   else
        printf (“%d is odd number.”,number);
   getch( );
 }
```

OUTPUT

```
Enter a number:
12
12 is even number.
```

3. NESTED IF……ELSE

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

Syntax

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

Example

```
if (y>=25)
  {   if (balance>5000)
        printf ("It is good");
      else
        printf ("No response");
   }
 else
    {    printf ("Not bad");
    }
 balance=balance+5000;
```


4. IF ELSE IF LADDER

Syntax

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

Example

```
 /* program to print the grade of student */
#include<stdio.h>
#include <conio.h>
 void main()
{
   int marks;
   printf("Enter marks ? ");
   scanf("%d", &marks);
   if(marks >= 75)
     printf("Distinction");
else if(marks >= 60)
        printf("First class");
     else if(marks >= 50)
             printf("Second class");
          else if(marks >= 35)
                  printf("Third class");
               else
                    printf("Failed");
        getch();
      }
```

5. SWITCH STATEMENT

- The switch statement provides a way of choosing between a set of alternatives, based on the value of an expression. It allows us to make a decision from the number of choices.

Syntax

```
switch (expression)
{
      case value-1:
            block1
            break;
      case value-2:
            block2
            break;
        ……..
        ……..
       default:
            default block
             break;
}
statement x;
```


❖working of switch statement

- First, the expression is evaluated.
- It checks for matching one by one case statements.
- When a match is found, program executes the statement.
- If no match is found, default statement is executed.
- **break** statement causes an exit from the switch statement.
- It is possible to nest the switch statement.
- The case can be arranged in any order.
- We can use char values in case and switch.

Example

```
/* program to simulate a simple calculator */
  #include<stdio.h>
  #include<conio.h>
    void main()
     {
         float a,b;
        char opr;
        printf ("Enter number1 operator number2")
        scanf ("%f %c %f",&a,&opr,&b);
        switch (opr)
          {
              case '+':
                 printf ("Sum : %f",(a + b));
                 break;
              case '-':
                 printf ("Difference : %f",(a - b));
                 break;
             case '*':
                 printf ("Product : %f",(a * b));
                 break;
             case '/':
                 printf ("Quotient : %f",(a / b));
                 break;
              default:
                 printf ("Invalid Operation!");
          }
         getch();
     }
```


➢You are also allowed to use **char** values in **case** and **switch** as shown in the following program:

```
void main( )
{
    char c = 'x' ;
    switch ( c )
     {
         case 'v' :
              printf ( "I am in case v \n" ) ;
              break ;
         case 'a' :
              printf ( "I am in case a \n" ) ;
              break ;
         case 'x' :
              printf ( "I am in case x \n" ) ;
              break ;
         default :
              printf ( "I am in default \n" ) ;
     }
}
```

6. THE *GOTO* KEYWORD

```
#include<stdio.h>
void main()
{
  int age;
 g:        //label name
   printf("you are Eligible\n");
 s:       //label name
   printf("you are not Eligible");
   printf("Enter you age:");
  scanf("%d", &age);
 if(age>=18)
     goto g;     //goto label g
 else
     goto s;    //goto label s
getch();
}
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 65

7. THE CONDITIONAL OPERATOR

- The conditional operators **?** and **:** are sometimes called ternary operators since they take three arguments.. Their general form is,

  expression 1 ? expression 2 : expression 3

- What this expression says is: "if **expression 1** is true (that is, if its value is non-zero), then the value returned will be **expression 2**, otherwise the value returned will be **expression 3**". Consider an example:

```
int  x, y ;
scanf ( "%d", &x ) ;
y = (x > 8? 1: 2);
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 66

- This statement will store 1 in **y** if **x** is greater than 8, otherwise it will store 2 in y. The equivalent **if** statement will be,

```
if ( x > 8 )
   y = 1 ;
else
   y = 2 ;
```

- The conditional operators can be nested as shown below.

```
int  big, a, b, c ;
big =  a > b ? ( a > c ? a: c ) : ( b > c ? b: c )  ;
```


DECISION MAKING & LOOPING

- This involves repeating some portion of the program either a specified number of times or until a particular condition is being satisfied.
- The three types of loops available in C are **for**, **while**, and **do-while.** A looping process, in general, would include the following four steps.
- Setting and initialization of a condition variable.
- Execution of the statements in the loop.
- Test for a specified value of the condition variable for execution of the loop.
- Incrementing or updating the condition variable.

1. WHILE LOOP

- It is an entry control loop. In a while loop, loop control variable should be initialized before the loop begins. The loop variable should be updated inside the body of the while

Syntax

```
initialize loop counter;
while (test loop counter using a condition)
   {
     do this;
     and this;
     increment loop counter;
   }
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 69

Eg:

```
#include <stdio.h>
void main( )
   {
      int s=1;
      while (s<=10)
        {
           printf ("\n%d",s);
           s++;
        }
      getch()
   }
```

OUTPUT

```
1
2
3
4
5
6
7
8
9
10
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 70

- Instead of incrementing we can even decrement a loop counter

```
void main( )
{
   int k = 4 ;
   while ( k >= 1 )
    {
       printf ( "decrement counter") ;
       k = k- 1 ;
    }
}
```

OUTPUT

Decrement counter
Decrement counter
Decrement counter
Decrement counter

2. FOR LOOP

- It is an entry control loop.

Syntax

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

- Initialization of the control variable is done first. The variable '**i**' is known as **loop control variable.**
- Next check the test condition. If it is true, the body of the loop is executed. After this, the control variable is incremented

OUTPUT

0 1 2 3 4 5 6 7

Eg:
```
for (i=0; i< 8; i++)
 {
    printf ("%d",i);
 }
```

- Testing and incrementation of loop counter is done in the **for** statement itself.
- Instead of **i++**, the statements **i = i + 1** or **i += 1** can also be used.

3. DO –WHILE LOOP

- It is an exit control loop. That is, it evaluates its test expression at the bottom of the loop after executing its loop body statement.
- **do -while** loop execute at least once even when the test expression evaluates to false initially

Syntax

```
do
 {
    this;
    and this;
    and this;
  } while (this condition is true);
```

Eg:

```
void main( )
{
    int p =3;
    do
     {
        printf("Am in Do-While");
     } while ( p < 2 ) ;
}
```

OUTPUT

Am in Do-While

- In this program the **printf ( )** would be executed once, since first the body of the loop is executed and then the condition is tested.
- There are some occasions when we want to execute a loop at least once no matter what.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 75

❖NESTING OF LOOPS

- The way **if** statements can be nested, similarly **while** and **for** can also be nested. To understand how nested loops work; look at the program given below:

```
/* Demonstration of nested loops */

void main ( )
{
    int a, b, sum ;
    for ( a = 1 ; a <= 3 ; a++ )    /* outer loop */
    {
        for ( b = 1 ; b <= 2 ; b++ )   /* inner loop */
        {
            sum = a + b;
            printf ( "a = %d b = %d sum = %d\n", a, b, sum ) ;
        }
    }
}
```

OUTPUT

```
a = 1 b = 1 sum = 2
a = 1 b = 2 sum = 3
a = 2 b = 1 sum = 3
a = 2 b = 2 sum = 4
a = 3 b = 1 sum = 4
a = 3 b = 2 sum = 5
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 76

❖THE *BREAK* STATEMENT

- A **break** statement inside a loop will abort the loop and transfer control to the statement following the loop.
- A **break** statement may appear inside a loop (while, do-while, for) or a switch statement.

Eg: Write a program to determine whether a number is **prime or not**. A prime number is one, which is divisible only by 1 or itself.

Eg:

```
void main( )
{
    int  num, i ;
   printf ( "Enter a number " ) ;
   scanf ( "%d", &num ) ;
   i = 2 ;
   while ( i <= num - 1 )
     {
       if ( num % i == 0 )
       {
           printf ( "Not a prime number" ) ;
           break ;
       }
       i++ ;
    }
  if ( i == num )
    printf ( "Prime number" ) ;
}
```

❖THE *CONTINUE* STATEMENT

- In some programming situations we want to take the control to the beginning of the loop, bypassing the statements inside the loop, which have not yet been executed.
- The keyword **continue** allows us to do this. When **continue** is encountered inside any loop, control automatically passes to the beginning of the loop.
- A **continue** is usually associated with an **if**. As an example, let's consider the following program

Eg:

```
void main( )
    {
        int  i, j ;
        for ( i = 1 ; i <= 2 ; i++ )
            {
               for ( j = 1 ; j <= 2 ; j++ )
                {
                  if ( i == j )
                  continue ;
                  printf ( "\n%d %d\n", i, j ) ;
               }
            }
        }
```

OUTPUT

1 2

2 1