# Fundamentals of C/Basic Elements of C

# Character Set

A character set in C contains any alphabet, digit or special symbol used to represent information.

| | |
|---|---|
| Alphabets | A, B,C,…….,Z<br>a, b, c,…….,z |
| Digits or Number | 0,1,2,3,4,5,6,7,8,9 |
| Special Symbols | +, -, <, >, #, !, %, *, ( , ),;,…..etc. |

# C Tokens:

The basic and smallest element used to develop the program and recognized by the C compiler is called as C tokens.

It is the individual unit in a program.

C Token includes:

- Identifier
- Keyword
- Constant and Variable
- Operator

# Identifier:

Names are given to various program elements such as variable, functions, labels and other user-defined items are known as an identifier.

- It can be combination of number, alphabet and special symbol (underscore).
- The first character of an identifier must be an alphabet or an underscore.
- In an identifier uppercase and lowercase alphabets are treated as differently. (case sensitive)
- Identifiers are strings of alphanumeric characters that don’t begin with a numbers.

Example: sum, Sum, a_b, _temp, field4, year2022, roll_no etc.

# Keyword

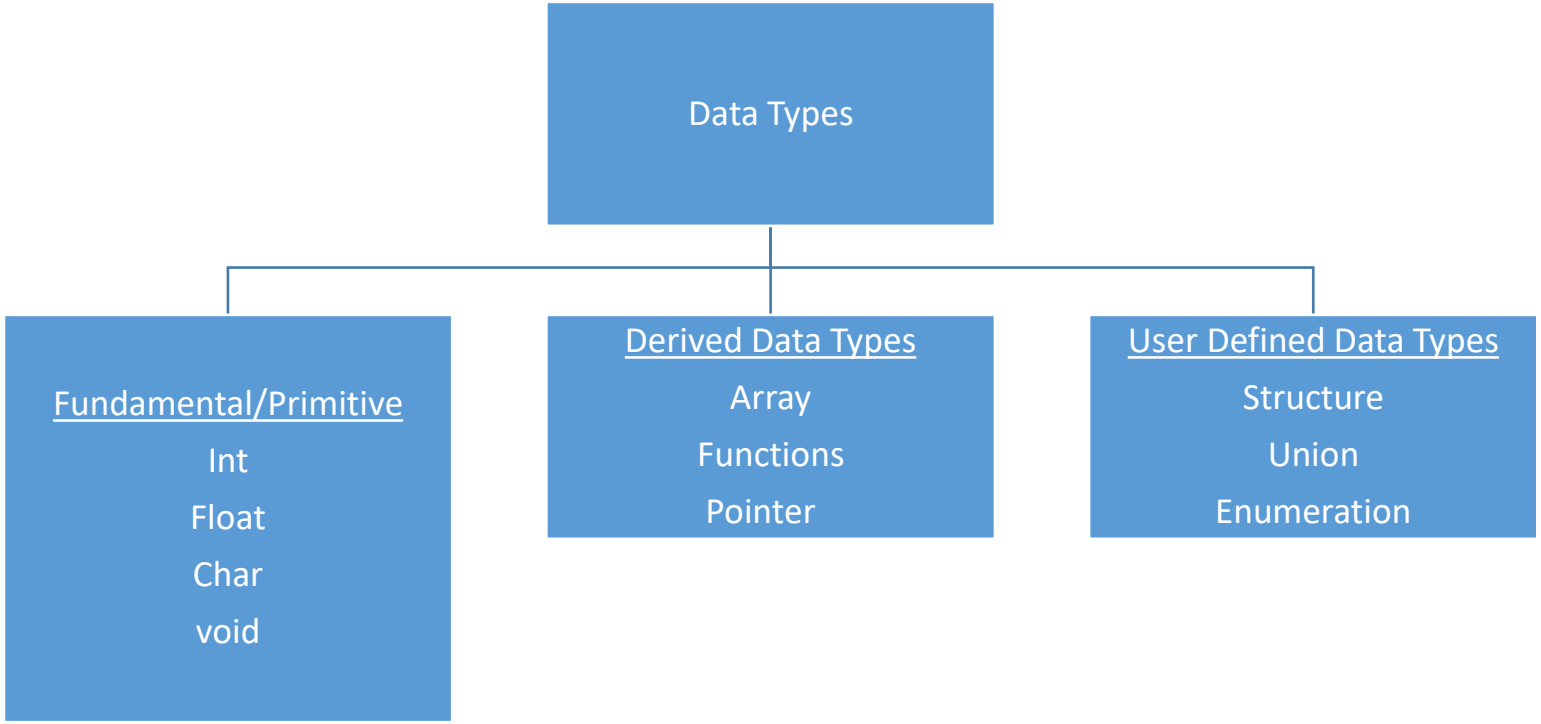
-The keyword is a reserved word that has standard and predefined meanings in C.

- Its meaning has already been defined by the C compiler.
- We cannot use a keyword for any other purpose other than a predefined task in a C program.
- The keyword can’t be used as variable name.
- C Contains 32 keywords.

| | | | |
|---|---|---|---|
| auto | double | int | struct |
| break | else | long | switch |
| case | enum | register | typedef |
| char | extern | return | union |
| continue | for | signed | void |
| do | if | static | while |
| default | goto | sizeof | Volatile |
| const | float | short | Unsigned |

# Data Types in C

- The data type is an instruction that is used to declare the category or type of variable being used in the program.
- It determines the way a computer organizes data in memory.
- It determines how much space it occupies in storage.



# Integer(int):

- int represents integer numbers (numbers without fraction e.g. 45).
- It requires 2 Bytes memory space.
- It can be either **signed** or **unsigned**.
- Derived data types from int are short int and long int.

# Float:

- Float represents real numbers (numbers with fraction e.g. 3.14).
- It requires 4 bytes memory space.
- It can also be signed or unsigned.
- Derived data types from float are double float, long float.

# Character(char):

- Char represents single alphabet or symbol (e.g. ‘A’, ‘m’, ‘@’, ‘5’) or multiple alphabets or symbols for the string.
- It requires 1 Byte memory space.
- It can also be either signed or unsigned.

# Void:

- Void data type has no values.
- It is used as the return type for functions that do not return a value.

# Constant:

- A constant is a quantity that doesn't change during program execution.
- Constants refer to fixed values that may not be altered by the program.

Types of Constants:

1. Integer Constants
2. Floating Constants
3. Character Constants
4. String Constants

# 1. Integer Constants:

An integer constant is an integer-valued number.

It consists of a sequence of digits.

Example: 4, 66, 87, -5, -56 etc.

Types of integer constants:

1. Decimal Integer Constants: Example: 1223, 456, 879, 45, 5599 etc.
2. Octal Integer Constants: Example: 0764, 05, 0345, 0777 etc.
3. Hexadecimal Integer Constants: Example: 0x456, 0xFA8, 0xDAF, 0x456, 0xBA8 etc.

# 2. Floating Point Constants:

A floating point constant is a number with fraction value or decimal value. Example: 7.5, 45.88, -45.67, 99.45, -4.87 etc.

# 3. Character Constants:

A character constant is a single character enclosed in single quotation marks.

When the C compiler encounters such a character constant, it translates it into the corresponding ASCII code and stores that number in the program. For example: the ASCII(American Standard Code for Information Interchange) code for ‘A’ is 65.

Example of valid character constants: ‘A’, ‘m’, ‘$’, ‘8’ etc.

# String Constants:

A string constant consists of any number of consecutive characters enclosed in double quotation marks.

Some string constants are “Shrija”, “Kathmandu789”, “Rs. 5000”, “I Love Nepal” etc.

The compiler automatically places a null character (“\0”) at the end of every string constants.

The size of a string is the number of characters plus one for this terminator. For example: “abc” size is 4.

Two additional types of Constants:

1. Literal Constants
2. Symbolic Constants

## Literal Constant:

A literal constant is a value that is typed directly into the source code whenever it is needed.

It is the basic and commonly used constant compared to a symbolic constant.

Example: area=3.14*r*r;  // 3.14 is the constant value of pie.

## Symbolic Constant:

A symbolic constant is a name or symbol to represent some constant value.

Example: area=pie*r*r; // pie is the symbolic constant.

# Variable:

- Variable is a symbolic name which is used to store different types of data in the computer's memory and may change during the program execution.
- When the user gives the value to the variable, that value is stored at the same location occupied by the variable.
- Variable may be of integer, character, string or floating type.
- Every variable must be declare before using it.

# Variable Declaration and Initialization:

All the variable needs to be declared before they are used.

For a  variable declaration, we have to state its type followed by variable name.

Syntax: data type <variable name>;

Example: int a;

float a, b, c;

int sum;

char ch;

# Initialization of Variable:

It is also possible to initialize variable with initial value using assignment operator (=)

For example:

```
int x=10;
float sum=10.589;
int a=1, b=2, c=9;
char ch='A';
```

# Delimiters:

A delimiter is one or more characters that separate text strings.

Common delimiters are commas (,), semicolon (;), quotes ( ", ' ), braces ({}), pipes (|), or slashes ( / \ ).

# Expression:

An expression is a combination of variables, constants and operators written according to the syntax of C language.

Every expression needs to be evaluated and returns some value.

Expression statements are those that consists of any valid C expression.

For example: sum= x + y;

# Simple and Complex Expression:

The simple C expression consists of a single item: a simple variable, literal constant or symbolic constant.

For example: 2+5; m*n; x=9 etc.

Complex expression consists of simpler expressions connected by various operators.

For example: (4*pi*r*r*r)/3+(p*t*r)/100+sqrt(144)%4

# Statements:

Statement is a command or instructions given to the computer to perform a specific task.

## Simple and Compound Statements:

Simple statements are those statements which are finally ended with semicolon ready to perform some action.

For example: printf(“Hello World”);

printf(“Value of a=%d”, a);

A compound statements is the group of valid C statements placed within curly braces { }.

Compound statements is a sequence of single statements used to perform a certain task.

For example:

```
{
        int a, b, s;
        s=a+b;
        printf(“Sum=%d”,s);
}
```

# Escape Sequence:

The back-slash symbol "\" is considered as an "escape character".

All escape sequence is represented by back slash(\) followed by a character.

Every escape sequence is used for a specific purpose.

For example:

| Sequence | Meaning |
|---|---|
| \b | Backspace |
| \f | Form Feed |
| \n | Newline |
| \t | Horizontal Tab |
| \v | Vertical Tab |
| \\ | Backslash |
| \' | Single Quote |
| \" | Double Quote |
| \? | Question Mark |

# Unit 2
# Elements of C

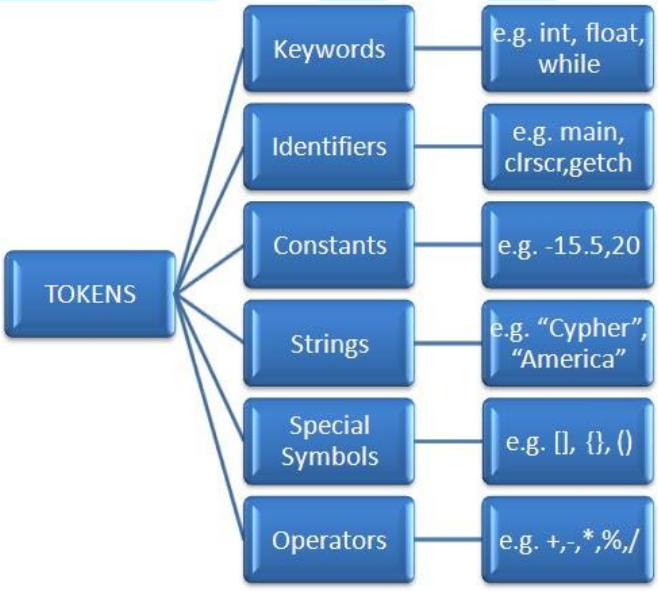
## Character Set

- Set of characters that are used to form words, numbers and expression in C is called C character set.
- Characters in C are grouped into the following four categories:

1. ***Letters or Alphabets:***
   - Uppercase alphabets → A…….Z
   - Lowercase alphabets → a…….z
2. ***Digits:***
   All decimal digits → 0, 1, 2,……..,9
3. ***Special characters:***
   , → comma ; → semicolon “→ quotation mark & → ampersand etc.
4. ***White spaces:***
   Blank spaces, horizontal tab, vertical tab etc.

## C Tokens

- C tokens are the basic buildings blocks in C language which are constructed together to write a C program.
- Each and every smallest individual unit in a C program is known as C tokens.
- C tokens are of six types.




## Keywords

- Keywords are predefined words for a C programming language.
- All keywords have fixed meaning and these meanings cannot be changed.
- The keywords cannot be used as variable names.

E.g.

| | | | |
|---|---|---|---|
| auto | double | int | struct |
| break | else | long | switch |
| case | enum | register | typedef |
| char | extern | union | continue |
| return | sizeof | for | signed |
| void | do | if | static |
| while | default | goto | volatile |
| const | float | short | unsigned |

## Identifiers

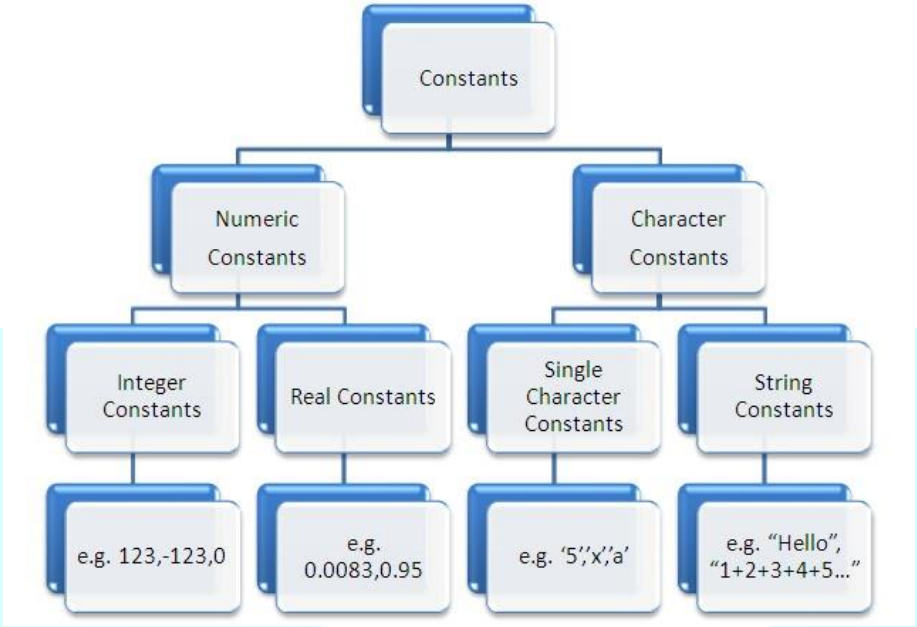
- Every word used in C program to identify the name of variables, functions, arrays, pointers and symbolic constants are known as identifiers.
- These are user defined names consisting of arbitrarily long sequence of letters and digits with either a letter or the underscore ( _ ) as a first character.
- There are certain rules that should be followed while naming C identifiers:
  1. They must begin with a letter or underscore (_).
  2. They must consist of only letters, digits, or underscore. No other special character is allowed.
  3. It should not be a keyword.
  4. It must not contain white space.
  5. It should be up to 31 characters long as only first 31 characters are significant.
  6. Uppercase and lowercase letters are not interchagable.

E.g. Valid and Invalid identifiers

| Valid | Invalid |
|---|---|
| sum | 7of9 |
| c4_5 | x-name |
| A_NUMBER | name with spaces |
| longnamewithmanychars | 1234a |
| TRUE | int |
| _split_name | AXYZ& |

## Constants

- A C constant refers to the data items that do not change their value during the program execution.
- These fixed values are also called **literals**.
- Several types of C constants that are allowed in C are:




***Integer constants:***

- Integer constants are whole numbers without any fractional part. It must have at least one digit and may contain either + or – sign. A number with no sign is assumed to be positive.
- There are three types of integer constants: ***Decimal integer constant***, ***Octal integer constant*** and ***Hexadecimal integer constant***.
- A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.

E.g. Decimal integer constant: 1, 3, 65, 5436664, -785
Octal integer constant: 037, 0, 0320, 0432
Hexadecimal integer constant: 0x4, 0X563, 0x1A

***Real constants:***

- The numbers having fractional parts are called real or floating point constants. These may be represented in one of the two forms called ***fractional form*** or the ***exponent form*** and may also have either + or – sign preceding it.
- Example of valid real constants in fractional form or decimal notation: 0.05, -0.905, 562.05, 0.015

***Representing a real constant in exponent form***:

- The general format in which a real number may be represented in exponential or scientific form is: ***mantissa e exponent***
- The mantissa must be either an integer or a real number expressed in decimal notation.
- The letter e separating the mantissa and the exponent can also be written in uppercase i.e. E and, the exponent must be an integer.
- Examples of valid real constants in exponent form are: 252E85, 0.15E-10, -3e+8, 4.1e8

***Character constants:***

- A character constant contains one single character enclosed within single quotes.
  E.g. 'a', 'Z', '5'
- It should be noted that character constants have numerical values known as ASCII values, for example, the value of 'A' is 65 which is its ASCII value.

***String constant:***

- String constants are sequence of characters enclosed within double quotes.
- May contain letters, numbers, special characters or blank spaces.
- For e.g., "hello", "abc", "hello91", "2077"

➢ **symbolic constants**

A symbolic constant is a name given to some numeric value, or a character constant or string constant.

***Defining symbolic constants:***

1) Using pre-processor directive **#define**
   Syntax: #define CONSTANT_NAME literal
   e.g. #define PI 3.14159

2) Using keyword literal
   Syntax: Const datatype CONSTANT_NAME = literal
   e.g. Const float PI= 3.14159

## Special Symbols

The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose.

[] () {} , ; : * . = #

***Braces { }:*** These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.

***Parentheses ( ):*** These special symbols are used to indicate function calls and function parameters.

***Brackets [ ]:*** Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.

## Escape Sequence

- An escape sequence is a non-printing characters used in C.
- These non-printing characters can be represented by using escape sequences represented by a backslash(\) followed by one or more characters.
- Each sequence are typically used to specify actions such as carriage return, backspace, line feed or move cursors to next line.

E.g.

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

```
#include <stdio.h>
#include <conio.h>
main()
{
   printf("Hello\tworld!!\n");
   printf("Hello!\n How are you?");
   getch();
   return 0;
}
```

***Output:***
Hello world!!
Hello!
How are you?

## Delimiters

- A delimiter is a unique character or series of characters that indicates the beginning or end of a specific statement, string or function body set.
- Delimiter examples include:
  - Parentheses: ( )
  - Curly brackets: { }
  - Escape sequence or comments: /*
  - Double quotes for defining string literals: " "

## Data Types

- A data type is a **type of data**.
- Data type is a data storage format that can contain a specific type or range of values.
- Data type in C refers to an extensive system used for declaring variable for function of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

ANSI C supports three classes of data types:

- Primary/fundamentals data types
- User-defined data types
- Derived data types

### *Primary Data Types*

Primary data types are categorized into five types:

1. ***Integer type (int):***
   - Integers are whole numbers.
   - It requires 16 bit of storage i.e. 2 bytes.
   - Three classes of integer: **Integer(int)**, **Short integer (short int)** and **Long integer (long int)**

| | Type | Size (bytes) | Range |
|---|---|---|---|
| Integer (int) | Signed int | 2 | $-2^{15}$ to $2^{15}-1$ |
| | Unsigned int | 2 | 0 to $2^{16}-1$ |
| Short integer (short int) | Signed short int | 2 | $-2^{15}$ to $2^{15}-1$ |
| | Unsigned short int | 2 | 0 to $2^{16}-1$ |
| Long integer (long int) | Signed long int | 4 | $-2^{31}$ to $2^{31}-1$ |
| | Unsigned long int | 4 | 0 to $2^{32}-1$ |

- Defined as:

```
int a;
int x=5;
```

***Signed integer vs Unsigned integer:***

| Signed Integer | Unsigned Integer |
|---|---|
| It represents both positive and negative integers | It represents only positive integers |
| The data type qualifier is **signed int or int.** Variables are defined as: signed int a; Int b; | The data type qualifier is **unsigned int or unsigned** Variables are defined as: unsigned int a; unsigned b; |
| By default all int are signed | Unsigned int have to be declared explicitly |
| It reserves 16-bit (2 bytes) in memory | It reserves 16-bit (2 bytes) in memory |
| Range $-2^{15}$ to $+2^{15}$ i.e. -32,768 to 32,767 | Range from 0 to $+2^{16}$ i.e. 0 to 65,535 |
| Its conversion character is **d** | Its conversion character is **u** |

### 2. *Floating point type (float):*

- Floating point type are fractional numbers.
- A variable of float type requires 4 bytes and the range of values that can be stored in it, is 3.4e-38 to 3.4e+38.
- Variable is defined as:

```
float a;
float x=23.7;
```

### 3. *Character type (char):*

- All single character used in programs belong to character type.
- The character data type holds exactly 8 bits (1 byte).
- The unsigned char has values between 0 and 255.
- The signed char has values from -128 to 127.
- Variable is declared as:

```
char var1= 'h';
```

Here, var1 is a variable of type char which is storing a character 'h'.

### 4. *Double precision floating point type (double):*

i. Double precision:
- It reserves 8 bytes in memory.
- It represents fractional number of the range 1.7e-308 to 3.4e+308

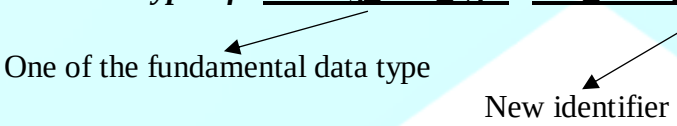
ii. Long double precision:
- It reserves 10 bytes in memory.
- It represents fractional numbers of the range 3.4e-4932 to 1.1e+4932

***5. Void type:***

- The void type has no value.
- This is usually used to specify a type of function when it does not return any value to the calling function.
- E.g. void main()

***User Defined Data Types***

- C supports a feature called type definition which allows users to define an identifier that would represent an existing data type.
- **typedef** statement is used to give new name to an existing data type.
- It allows users to define new data types that are equivalent to an existing data types.
- General form: ***typedef existing_data_type new_name_for_existing_data_type;***




- E.g. **typedef int** salary;
  Here salary symbolizes int data types. They can be later used to declare variables as:
  **salary dept1, dept2;**
  Therefore dept1 and dept2 are indirectly declared as integer datatype.

***Derived Data Types***

- Array, functions, pointers are derived data types they are discussed in unit 6, 7 and 9.

**Conversion specifier**

%d → integer
%f → floating point
%c → character
%s → string

Input/Output
scanf()/printf()

printf("%d",100);

scanf("conversion specifier", &variable_name);

if x is an integer variable.
scanf("%d",&x)

```
#include <stdio.h>
int main()
{
   char ch = 'A';
   printf("%c\n", ch);
   return 0;
}
```

```
#include <stdio.h>
int main()
{
   float a = 12.67;
   printf("%f\n", a);
   return 0;
}
```

## Variables

- A symbolic name which is used to store data item i.e. a numerical quantity or a character constant.
- Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.
- The same variable can store different value at different portion of a program.
- Variable name may consists of letters, digits or underscore characters.
- The rules for naming variables are similar to those of identifiers.

***Variable Declaration:***

- Any variable should be defined before using it in a program.
- The variable are declared using following syntax:

```
data_type  variable_name1, variable_name2, ……..;
```

E.g.
```
int n1;
int v1, v2, v3;
char c;
float radius;
```

## Preprocessor Directives

- Collection of special statements that are executed at the beginning of a compilation process.
- Placed in the source program before the main function.

```
#include<stdio.h>      //used for file inclusion
#define PI 3.1416      //defining symbolic constant PI
```

- These statements are called preprocessor directives as they are processed before compilation of any other source code in the program.

## Statement

- Statement is the complete direction to computer to perform specific task.
- In C, a statement is terminated by semicolon(;)

***Types of statement***:

1) Null statement: Does nothing.

```
;
```

2) Compound statement: Block of statement

```
{
    ……;
    ……;
}
```

Compound statement

**Expression**

- Combination of variable, constant, operators etc. on the basis of language grammar.
- Every expression consists of at least on operand and can have one or more operators.
- Operands are values and operators are symbols that represent particular actions.
- E.g. a+b, a+b*c, a*b/3

**Some Q & A (C Program)**

***Q. Write a program to input the marks of a student in different 5 subjects in a class test and compute total marks and percentage score. Assume each subject has full marks 20.***

```
#include<stdio.h>
#include<conio.h>
#define FM 20
void main()
{
   float sub1, sub2, sub3, sub4, sub5, total, percentage;
   printf("Enter the marks of 5 subjects:");
   printf("\nSub1:");
   scanf("%f", &sub1);
   printf("\nSub2:");
   scanf("%f", &sub2);
   printf("\nSub3:");
   scanf("%f", &sub3);
   printf("\nSub4:");
   scanf("%f", &sub4);
   printf("\nSub5:");
   scanf("%f", &sub5);
   total= sub1+sub2+sub3+sub4+sub5;
   percentage=total/(5*FM)*100;
   printf("he total marks obtained=%f",total);
   printf("\nThe percentage score=%f", percentage);
   getch();
}
```

***Q. An employee has some basic salary. He gets 10% performance allowance and 20% expense allowances. The provident fund is deducted as 10% of his basic salary and 1% tax is ducted after provident fund. Find his net payment of a month.***

```
#include<stdio.h>
#include<conio.h>
void main()
{
   float BS, PA, EA, PF, tax, TS, NS;
   printf("Enter the basic salary:");
   scanf("%f",&BS);
   PA=(10.0/100)*BS;
```

BS→Basic salary
PA→Performance allowance
EA→Expense
PF→Provident fund
TS→Total salary
NS→Net salary

```
    printf("The PA is %f\n", PA);
    EA=(20.0/100)*BS;
    printf("The EA is %f\n", EA);
    PF=(10.0/100)*BS;
    printf("The PF is %f\n", PF);
    TS=BS+PA+EA-PF;
    printf("The TS is %f\n", TS);
    tax=(1.0/100)*TS;
    printf("The tax is %f\n", tax);
    NS=TS-tax;
    printf("The NS is %f", NS);
    getch();
}
```

***Q. Write a program to compute the area of circle.***

```
#include<stdio.h>
#include<conio.h>
#define PI 3.1416
int main()
{
   float radius, area;
   printf("\nEnter the radius of Circle : ");
   scanf("%f", &radius);
   area = PI * radius * radius;
   printf("\nArea of Circle : %f", area);
   getch();
   return 0;
}
```

***Q. Write a program to compute the roots of quadratic equation $ax^2 + bx + c = 0$ where the coefficient a, b, c are inputs to the program.***

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    float a, b, c, d, root1, root2;
    printf("Enter the coefficient of x^2:");
    scanf("%f",&a);
    printf("\nEnter the coefficient of x:");
    scanf("%f",&b);
    printf("\nEnter the constant:");
    scanf("%f",&c);
    d=b*b-4*a*c;
    root1=(-b+sqrt(d))/(2*a);
```

```
    root2=(-b-sqrt(d))/(2*a);
    printf("\nroot1=%f",root1);
    printf("\nroot2=%f",root2);
    getch();
}
```