

Module 1

Computer:

Computer is a device that transforms data into meaningful information. The functions of this system are: It accepts data or instruction by way of input, It stores data, It can process data as required by the user, It gives results in the form of output, and It controls all operations inside a computer.

Components:

Refer from class notes.

Need of Computer Programming:

1. Since computer is a machine, Programming helps to understand computer.
2. A program is a set of logically related instruction that is arranged in a sequence that directs the computer in solving problem. The process of writing a program is called programming.
3. A set of rules that provides a way of communicating the computer about operations to be performed is called computer programming language.

Structured Programming:

1. Structured means highly organized or arranged in a definite pattern, Programming tells computer to perform operations and Approach means a methodology.
2. Structured programming enforces a logical structure on the program to make it more efficient and easier to understand and modify.
3. Structured programming facilitates program understanding and modification and has a top-down design approach, where a bigger problem is broken down into smaller tasks.

Algorithm:

1. An algorithm is an effective procedure consisting of sequence of ordered instructions for solving a problem in a finite number of steps.
2. An algorithm is a finite set of statements, each of which has a clear meaning and can be executed in a finite amount of time and with a finite amount of effort.
3. Algorithms are not computer programs, but they are descriptions containing natural language and familiar structures from programming language.

Properties:

1. Multiplicity: The algorithm can be represented in multiple ways, i.e., in English language or by the graphical representation called as flowchart or the pseudo code.
2. Finiteness: An algorithm terminates after a finite numbers of steps.
3. Definiteness: Each of the statement in the algorithm must be clear and precise. This means that the action specified by the step cannot be interpreted in multiple ways & can be performed without any confusion.
4. Range of Input and Output: The range of the input is to be clearly mentioned in the algorithm. An algorithm accepts zero or more inputs and produces at least one output.
5. Effectiveness: The instructions can be performed by using the given inputs in a finite amount of time at a fast speed.

Basic Constructs:

Sequence: A number of instructions is processed one after the other.

Eg: input (name)
 input (age)
 output (message)

Selection: The next instruction to be executed depends on a condition.

Eg: input (age)
 if (age < 21)
 then
 print(young message)
 else
 output(old message)
 end if

Iteration: When the number of instructions are repeated.








Eg: for 20 times
 do
 draw a line
 end do

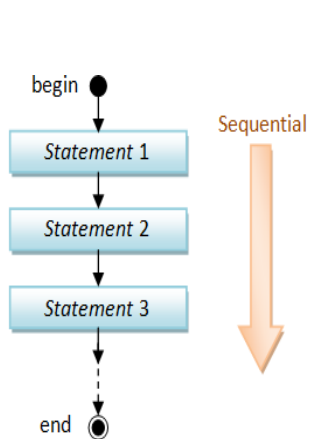
Flowchart and its Needs:

1. Flowchart is a graphical or diagrammatic representation of an algorithm.
2. A flowchart is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting these with arrows.
3. This diagrammatic representation can give a step-by-step solution to a given problem.
4. Flowchart is very helpful in writing program and explaining program to others.

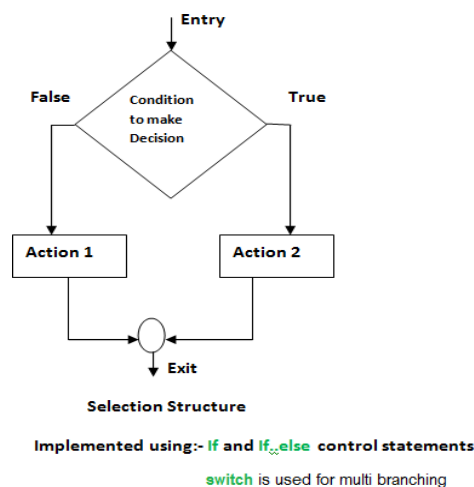
Conventions:

Different symbols are used for different states in flowchart.

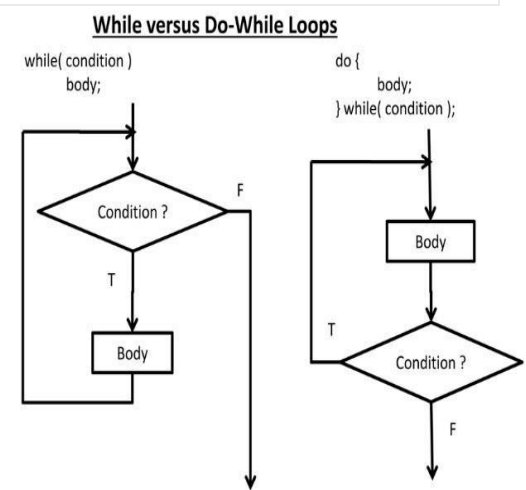
| Symbol | Purpose | Description |
|---|----------------------|--|
|  | Flow line | Used to indicate the flow of logic by connecting symbols. |
|  | Terminal(Stop/Start) | Used to represent start and end of flowchart. |
|  | Input/output | Used for input and output operation. |
|  | Processing | Used for arithmetic operations and data- manipulations. |
|  | Decision | Used to represent the operation in which there are two alternatives, true and false. |
|  | On-page Connector | Used to join different flow line |
|  | Off-page Connector | Used to connect flowchart portion on different page. |



SEQUENCE



CONDITION



LOOPING

C-Programming:

1. C is a general-purpose high level language that was originally developed by Dennis Ritchie for the UNIX operating system in 1972.

2. The name was given because it overcomes the limitations of its preceding language, B, BPCL, etc.
Features: Refer to class notes

Character Set of C:

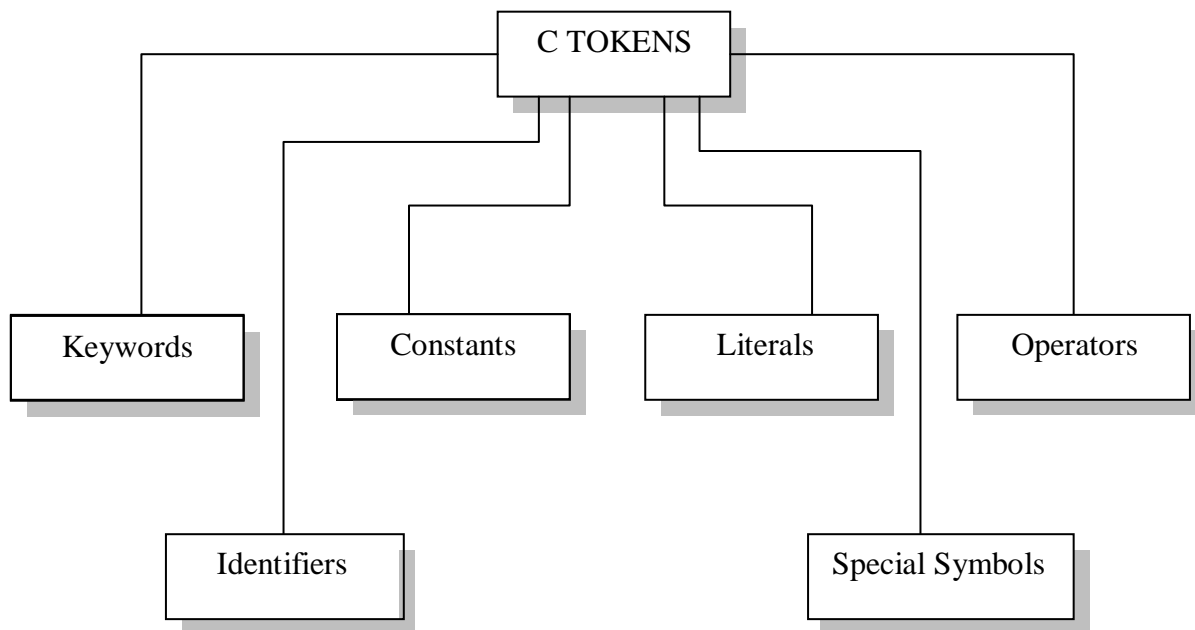
The character set is the fundamental raw material of any language used to represent information.

| No. | Characters | List | | | |
|-----|--------------------|---|------------------|--------|-----------------------|
| 1 | Alphabets | Uppercase A.....Z Lowercase a...z | | | |
| 2 | Digits | All decimal digits 0.....9 | | | |
| 3 | White spaces | Whitespace is the term used in C to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as int, ends and the next element begins. Blank space, Horizontal tab, New line | | | |
| 4 | Special characters | Symbol | Name | Symbol | Name |
| | | , | Comma | & | Ampersand |
| | | . | Period | ^ | Caret |
| | | ; | Semicolon | * | Asterisk |
| | | : | Colon | - | Minus sign |
| | | ? | Question mark | + | Plus sign |
| | | ' | Apostrophe | < | Opening angle bracket |
| | | " | Quotation mark | > | Closing angle bracket |
| | | ! | Exclamation mark | (| Left parenthesis |
| | | | Vertical bar |) | Right parenthesis |
| | | / | Slash | [| Left bracket |
| | | \ | Backslash |] | Right bracket |
| | | % | Percent sign | { | Left brace |
| | | _ | Under score | } | Right brace |
| | | \$ | Dollar sign | # | Number sign |

C Tokens:

1. The smallest individual units of a program are called as C tokens.

2. C has six types of tokens.



1. Keywords:

Keywords are predefined; reserved words used in programming that have a special meaning. Keywords are part of the syntax and they cannot be used as an identifier.

For example: `int money;`

Here, `int` is a keyword that indicates '`money`' is a variable of type integer.

As C is a case sensitive language, all keywords must be written in lowercase.

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

Identifiers: Identifier refers to name given to entities such as variables, functions, structures, etc. Identifier must be unique. They are created to give unique name to a entity to identify it during the execution of the program. Rules for writing an identifier: A valid identifier can have both uppercase and lowercase letters, digits and underscores but cannot use a keyword. The first letter of an identifier cannot be a digit and it must not contain whitespace.

For example: `int money;` where `money` is an identifier.

One type of Identifier is Variables: C variable is a named location in a memory where a program can manipulate the data. This location is used to hold the value of the variable. The value of the C variable may get change in the program. Memory space is not allocated for a variable while declaration. It happens only on variable initialization which means assigning a value to the variable.

a. Variable declaration: Syntax: `data_type variable_name;` Example: `int x, y, z;`

b. Variable initialization: Syntax: `data_type variable_name = value;` Example: `int x=50, y=30;`

2. Constants:

A constant is an identifier whose value cannot be altered in a program. For example: `1`, `2.5`, `"C programming is easy"`, etc. For example, `const double PI =3.14`. Here, `PI` is a constant.

- b. String Constants: A string constant is a sequence of characters enclosed in double quotes. The characters may be letters, numbers, special characters and blank space. For example: "Hello!" "2017" "WELL DONE"
- c. Backslash Character Constants/Escape Sequence: C supports some special backslash character constants that are used in output functions.

| Constant | Meaning |
|----------|----------------------|
| '\a' | Audible alert (bell) |
| '\b' | Back space |
| '\n' | New line |
| '\t' | Horizontal tab |
| '\'' | Single quote |
| '\"' | Double quote |
| '\?' | Question mark |
| '\\' | Backslash |
| '\0' | Null |

3. Literals: Refer to class notes

4. Special Symbols: Refer from 'Character Set'

5. Operators:

An operator is a symbol that tells the Computer to perform certain mathematical or logical manipulations. An expression is a sequence of operands and operators that reduces to single value after evaluation.

a. Arithmetic Operators:

C provides all the basic arithmetic operators, they are +, -, *, /, %

A. Integer Arithmetic: When the operands in an expression are integers then the expression is an integer expression and the operation is called integer arithmetic.

B. Floating Point Arithmetic: Floating Point Arithmetic involves only real operands of decimal or exponential notation.

Note: % cannot be used for floating point data.

b. Relational Operator:

These are the operators used to compare arithmetic, logical and character expressions. The value of a relational express is either one or zero. It is 1 if one is the specified relation is true and zero if it is false. The relational operators in C are: <, <=, >, >=, ==, !=

c. Logical Operator:

Logical Operators are used when we want to test more than one condition and make decisions. Here the operands can be constants, variables and expressions. Logical operators are &&, ||, !

| OP1 | OP2 | OP1&&OP2 | OP1 OP2 | OP | ! |
|-----|-----|----------|----------|----|---|
| 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | | |
| 0 | 0 | 0 | 0 | | |

Program:

```
#include<stdio.h>
void main()
{
int num1 = 30, num2 = 40;
if(num1>=40 || num2>=40)
    printf("Or If Block Gets Executed");
if(num1>=20 && num2>=20)
    printf("And If Block Gets Executed");
if( !(num1>=40))
    printf("Not If Block Gets Executed");
}
```

Output:

Or If Block Gets Executed And If Block Gets Executed Not If Block Gets Executed

d. Assignment Operator(=)

Used to assign the result of an expression to a variable. In addition C has a set of "short hand" assignment operators of the form: variable operator= expression;

Shorthand assignment operator is used express the syntax shorter way

| Operator symbol | Name of the operator | Example | Equivalent construct |
|-----------------|---------------------------|---------|----------------------|
| += | Addition assignment | x += 4; | x = x + 4; |
| -= | Subtraction assignment | x -= 4; | x = x - 4; |
| *= | Multiplication assignment | x *= 4; | x = x * 4; |
| /= | Division assignment | x /= 4; | x = x / 4; |
| %= | Remainder assignment | x %= 4; | x = x % 4; |

Example:

```
#include<stdio.h>
void main() {
int res;
res =11;
res +=4;
printf("\nResult of Ex1 = %d", res);
res =11;
res -= 4;
printf("\nResult of Ex2 = %d", res);
res =11;
res *=4;
printf("\nResult of Ex3 = %d", res);
res =11;
res /=4;
printf("\nResult of Ex4 = %d", res);
res = 11;
res %= 4;
printf("\nResult of Ex5 = %d", res);
}
```

Output:

Result of Ex1 = 15 Result of Ex2 = 7 Result of Ex3 = 44 Result of Ex4 = 2 Result of Ex5 = 3

e. Increment and Decrement: Refer from class notes

f. Conditional Operator (Ternary Operator):

They also called as ?: operator. Ternary Operators takes on 3 Arguments

Syntax:

expression 1 ? expression 2 : expression 3 where

expression1 is Condition

expression2 is Statement Followed if Condition is True

expression3 is Statement Followed if Condition is False

g. Bitwise operator:

are used to perform operations at binary level i.e. bitwise. these operators are used for testing the bits, or Shifting them right or left

i) << Left Shift:

a=13 00001101

Consider a << 2 which Shifts two bits to left, that is 2 zeros are inserted at the right and two bits at the left are moved out.

Finally the result is 00110100

ii) >> Right Shift:

a=13 00001101

Consider a >> 2 which Shifts two bits to right, that is 2 zeros are inserted at the left and two bits at the right are moved out.

Finally the result is 00000011

iii) ~ Complement: Convert 0's to 1's and 1's to 0's .

iv) & (Bitwise AND):

v) | (Bitwise OR):

vi) ^ (Bitwise Exclusive-OR):

| a = 13 | 0000 1101 | Op1 | Op2 | AND | OR | EX-OR |
|--------|-----------|-----|-----|-----|----|-------|
| b = 6 | 0000 0110 | 0 | 0 | 0 | 0 | 0 |
| a & b | 0000 0100 | 0 | 1 | 0 | 1 | 1 |
| a b | 0000 1111 | 1 | 0 | 0 | 1 | 1 |
| a ^ b | 0000 1011 | 1 | 1 | 1 | 1 | 0 |

h. Special:

i) sizeof operator: is used to find the on. of bytes occupied by a variable / data type in computer memory. eg :

sizeof (float) returns 4

int m, x [50]

sizeof (m) returns 2

sizeof (x) returns 100 (50 x 2)

ii) comma operator: can be used to link the related expressions together.

Operator Precedence & Associativity:

The precedence of the operators means the sequence in which the operators will be operated on, in case of multiple operators in a statement. So it is a priority that indicates which operator has to be evaluated first when there is more than one operator.

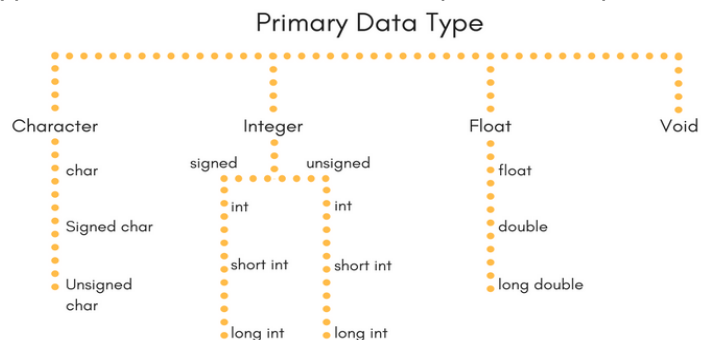
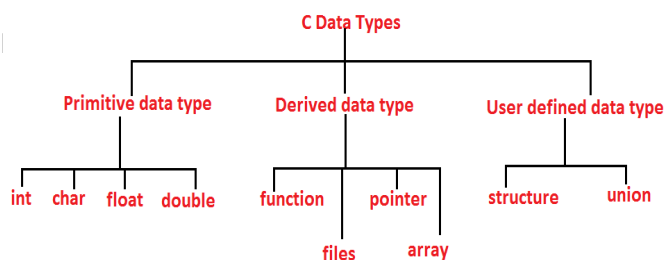
Associativity: When there is more than one operator with same precedence or priority then we consider associativity, which indicated the order in which the expression has to be evaluated. It may be either from Left to Right or Right to Left. The associativity of operators refers to the direction in which the operation will be performed in case of equal precedence operators

| Operator | Description | Associativity | Rank |
|----------|-------------------------|---------------|------|
| () | Function call | Left to right | 1 |
| [] | Array element reference | | |

| | | | |
|---|--|---------------|----|
| ++ -- ! ~ * & sizeof (type) | Increment Decrement Logical negation Ones complement Pointer reference Address Size of an object Type cast(conversion) | Right to left | 2 |
| * / % | Multiplication Division Modulus | Left to right | 3 |
| + - | Addition Subtraction | Left to right | 4 |
| << >> | Left shift Right shift | Left to right | 5 |
| < <= > >= | Less than Less than or equal to Greater than Greater than or equal to | Left to right | 6 |
| == != | Equality Inequality | Left to right | 7 |
| & | Bitwise AND | Left to right | 8 |
| ^ | Bitwise XOR | Left to right | 9 |
| | Bitwise OR | Left to right | 10 |
| && | Logical AND | Left to right | 11 |
| | Logical OR | Left to right | 12 |
| ?: | Conditional expression | Right to left | 13 |
| = *= /= %= | Assignment and shorthand assignment operators | Right to left | 14 |
| , | Comma operator | Left to right | 15 |

Data types:

It refers to the variety of data types available allow the programmer to select the type appropriate to the need of application as well as machine. Data types in C determine how much space it occupies in storage. C Supports Three classes of data types.



Integer Types: Integers are whole numbers with a range of variables supported by a particular machine. C has three classes of integer storage: short int, int, long int. It has a set of qualifiers i.e., signed and unsigned

| Data type | Size (bits) | Range |
|-----------|-------------|-------|
|-----------|-------------|-------|

| | | |
|-------------------------------|----|---------------------------------|
| char or signed char | 8 | -128 to 127 |
| unsigned char | 8 | 0 to 255 |
| int or signed int | 16 | -32,768 to 32,767 |
| unsigned int | 16 | 0 to 65535 |
| short int or signed short int | 8 | -128 to 127 |
| unsigned short int | 8 | 0 to 255 |
| long int or signed long int | 32 | -2,147,483,648 to 2,147,483,647 |
| unsigned long int | 32 | 0 to 4,294,967,295 |
| float | 32 | 3.4e-38 to 3.4e+38 |
| double | 64 | 1.7e-308 to 1.7e+308 |
| long double | 80 | 3.4e-4932 to 1.1e+4932 |

Floating Point Types: Floating Point numbers are stored with 6 digits of precision defined with keyword float. double gives a precision of 14 digits for more accuracy.

Void types:

A void type has no value this is usually used to specify the return type of function, this function does not return any value to calling function.

Character type:

Characters are usually stored in 8 bits.

Type Casting:

1. Type casting is a way to convert a variable from one data type to another data type.
2. For example, if you want to store a 'long' value into a simple integer then you can type cast 'long' to 'int'.

3. Syntax: (type_name) expression

4. Consider the following example –

```
#include <stdio.h>
void main() {
int sum = 17, count = 5;
double mean;
mean = (double) sum / count;
printf("Value of mean : %f\n", mean );
}
```

When the above code is compiled and executed, it produces the following result –
Value of mean: 3.400000

5. Type conversions can be implicit which is performed by the compiler automatically, or it can be specified explicitly through the use of the cast operator. It is considered good programming practice to use the cast operator whenever type conversions are necessary.

Integer Promotion:

Integer promotion is the process by which values of character type "smaller" than int are converted either to int. Consider an example of adding a character with an integer –

```
#include <stdio.h>
void main() {
int i = 17;
char c = 'c'; /* ascii value is 99 */
int sum;
sum = i + c;
printf("Value of sum: %d\n", sum );
}
```

When the above code is compiled and executed, it produces the following result –
Value of sum: 116

Because the compiler is doing integer promotion and converting the value of 'c' to ASCII before performing the actual addition operation.

Data Input and Output Basic Operations:

C has many input output functions in order to read data from input devices and display the results on the screen.

Classification:

| Formatted | | Unformatted | |
|-----------|-----------|-------------|---------|
| Input | Output | Input | Output |
| scanf() | printf() | getch() | putch() |
| | | getche() | |
| | | getchar() | |
| | | gets() | puts() |

Formatted Functions:

scanf() function is used to read values using keyboard.

Syntax: scanf("format string", list_of_addresses_of_variables);

The format string contains Conversion Specifier. Conversion specifications begin with a % sign.

| Data Type | Conversion Specifier |
|------------------|-----------------------------|
| integer | %d |
| float | %f |
| double | %lf |
| char | %c |

& is called the "address" operator which indicates the memory location of the variable.

printf() function is used to print/display values of variables.

Syntax: printf("text string", list_of_variables);

printf() examines the format string from left to right and prints all the characters. When it finds % (Conversion Specifier) it picks up the first value. This process continues till the end of format string is reached.

Unformatted functions:

getchar(): used to read one character at a time from the keyboard. When this function is executed, the computer will wait for a key to be pressed and assigns the value to the variable when the "enter" key pressed.

Eg.:

```
void main () {  
char ch;  
printf("Enter a char");  
ch = getchar();  
printf("\nch=%c", ch);  
}
```

Output:

```
Enter a char M  
ch =M
```

putchar(): used to display one character at a time on the monitor.

Eg.:

```
char ch = 'M';  
putchar(ch);
```

getch(): used to read a char from a keyboard and without expecting the enter key to be pressed. When this function is executed, computer waits for a key to be pressed from the keyboard. As soon as a key is pressed, the control is transferred to the next line of the program and the value is assigned to the char variable.

getche(): used to read a char from the key board without expecting the enter key to be pressed. The character which is read will be displayed on the monitor.

Eg.:

```
void main () {  
char ch;  
printf("Enter a char");  
ch = getche();  
printf("\nch=%c", ch);  
}
```

Output:

```
Enter a char MM  
ch =M
```

gets(): used to read a string of characters including white spaces. When this function is executed the computer waits for the string to be entered

Eg.:

```
char S[20];  
gets(S);
```

puts(): used to display a string of characters including white spaces.

Eg.:

```
char S[20]="Hello World";  
puts(S);
```

Errors. Types of errors:

Errors also known as bugs in the world of programming which occur unwillingly. It prevents the program to compile and run correctly as per the expectation of the programmer.

Basically there are three types of errors in c programming:

1. Runtime Errors: C runtime errors are those errors that occur due to some illegal operation performed in the program. Examples are: Dividing a number by zero, Trying to open a file which is not created, etc.
2. Compile Errors: Compile errors are those errors that occur at the time of compilation of the program. When the rules of the c programming language are not followed, the compiler will show syntax errors. For example, consider the statement
 int a,b
The above statement will produce syntax error because statement is not terminated with semi-colon.
3. Logical Errors: Logical errors are the errors in the output of the program. The presence of logical errors leads to undesired or incorrect output and are caused due to error in the logic applied in the program to produce the desired output.

Inbuilt Functions in C:

1. isalnum():

- a. Defn: checks whether given character is alphanumeric or not. It is defined in ctype.h header file.
- b. Syntax: int isalnum (char x);
- c. Example:

```
#include <stdio.h>
void main(){
char ch='A';
if( isalnum(ch) )
    printf ( "\nEntered character is alphanumeric" ) ;
else
    printf ( "\nEntered character is not alphanumeric" ) ;
}
```

2. abs():

a. Defn: returns the absolute value of an integer. The absolute value of a number is always positive. It is defined in stdlib.h header file.

b. Syntax: int abs (int n);

c. Example:

```
#include<stdio.h>
#include<stdlib.h>
void main(){
int m = abs(200);
int n = abs(-400);
printf("Absolute value of m = %d\n", m);
printf("Absolute value of n = %d \n",n);
}
```

3. floor():

a. Defn: returns the nearest integer value which is less than or equal to the given number. It is defined in math.h header file.

b. Syntax: double floor (double x);

c. Example:

```
#include <stdio.h>
#include <math.h>
void main(){
float i=5.1, j=-6.9, k=-5.4;
printf("floor of  %f is  %f\n", i, floor(i));
printf("floor of  %f is  %f\n", j, floor(j));
printf("floor of  %f is  %f\n", k, floor(k));
}
```

4. round():

a. Defn: returns the nearest integer value of the argument passed to this function. It is defined in math.h header file.

b. Syntax: double round (double a);

c. Example:

```
#include <stdio.h>
#include <math.h>
void main(){
float i=5.4, j=5.6;
printf("round of  %f is  %f\n", i, round(i));
printf("round of  %f is  %f\n", j, round(j));
}
```

5. ceil()

a. Defn: returns nearest integer value which is greater than or equal to the given number. It is defined in math.h header file.

b. Syntax: double ceil (double x);

c. Example:

```
#include <stdio.h>
#include <math.h>
void main(){
float i=5.4, j=-5.6;
```

```
printf("ceil of %f is %f\n", i, ceil(i));  
printf("ceil of %f is %f\n", j, ceil(j));  
}
```

6. sqrt()

a. Defn: used to find the square root of the given number. It is defined in math.h header file.

b. Syntax: double sqrt (double x);

c. Example:

```
#include <stdio.h>  
#include <math.h>
```

```
void main(){  
printf("sqrt of 16 = %f\n", sqrt (16) );  
printf("sqrt of 2 = %f\n", sqrt (2) );  
}
```

7. pow():

a. Defn: used to find the power of the given number. It is defined in math.h header file.

b. Syntax: double pow (double base, double exponent);

c. Example:

```
#include <stdio.h>  
#include <math.h>  
void main(){  
printf ("2 power 4 = %f\n", pow (2.0, 4.0) );  
printf ("5 power 3 = %f\n", pow (5, 3) );  
}
```

8. trunc():

a. Defn: truncates the decimal value from floating point value and returns integer value. It is defined in math.h header file.

b. Syntax: double trunc (double a);

c. Example:

```
#include <stdio.h>  
#include <math.h>  
void main(){  
printf ("truncated value of 16.99 = %f\n", trunc (16.99) );  
printf ("truncated value of -20.1 = %f\n", trunc (-20.1) );  
}
```