Programming Language Concepts and Logics
<MD>

A set of instructions to carry out various functions is called a computer program. The language used in the communication of instructions to a computer is known as programming language. Programming languages are used to create software applications, operating systems, and other types of programs that can be run on a computer or other device.

Programming languages have a set of rules and conventions that define how the instructions should be written and formatted. These rules and conventions are known as the syntax of the programming language.

Programming languages also have a set of built-in data types and operations that can be used to manipulate data and perform various tasks. These data types and operations are known as the semantics of the programming language.

There are many different programming languages available, each with its own syntax and semantics. Some common programming languages include C, C++, Java, Python, and JavaScript. Each programming language has its own set of features and capabilities, and is suited for different types of tasks and applications.

Overall, programming languages are an essential tool for creating software applications and other programs that can be executed on a computer or other device.

Types of Programming Languages:

There are 5 types of programming languages on the basis of their generation:

a. **Machine Language / First Generation Language (1GL)**
b. **Assembly Language/ Second Generation Language (2GL)**
c. **Procedural Language/ Third Generation Language(3GL)**
d. **Problem-oriented Language / Fourth Generation Language (4GL)**
e. **Natural Language/ Fifth Generation Language (5GL)**

A. **Machine Language:**

They are the first generation language and can also be called low-level programming languages because they were used to program the computer system at a very low level of abstraction i.e at machine level. They are used to write instructions that can be directly executed by a computer's CPU. All the other programming languages must be translated to machine language before they can be executed by CPU. Machine language consists of two values 0 and 1 where 0 represents disabled state and 1 represents enable state.

Advantages:

a. With the absence of translators, they are directly executed by the computer.
b. The programs written in this language are executed speedily and efficiently.
c. Memory utilization is efficient.

Disadvantages:

a. It is difficult to develop the program in machine language
b. Other programmers have a hard time understanding the programs developed in these languages.
c. It is difficult to find the error and solve it
d. The programs are not portable as they are hardware dependent.

B. **Assembly Language:**

They are the second generation language which is a level above machine language. They use the concept of mnemonics code for writing programs. For example, ADD, MUL, MOV etc. were used to assemble the machine level codes and perform activities.

Assembly Language —-------> Assembler —----> Machine Language Instructions

Fig. The function of assembler

A software called assembler is used to convert the assembly language programs into machine-understandable format.

Advantages:

a. It is easy to develop, understand and modify the programs developed in these languages as compared to machine level language.
b. These languages are less prone to error.

c. The debugging process is easier as compared to machine language

Disadvantages:

a. SInce it is still a machine dependent language, there is less portability.
b. One program written on one computer may not work on another computer.
c. Like machine language, the result is still less productive.
d. Complexity of the program is high
e. It is slow and less efficient than machine language.

C. **Procedural Oriented Language:**
They are the third generation languages that are more easy to understand than ML and assembly languages. In POP, a program is written as a series of procedures or functions, which are blocks of code that perform a specific task. These procedures are then executed in a specific order to solve a problem or accomplish a task. Interpreters or Compilers are used to translate these programs. Some examples of 3GLs are C, C++, JAVA, BASIC, etc.

High Level Language → Compiler/Interpreter → Machine language

Advantages:

a. Easy to learn, develop and understand these programs.
b. They are highly portable since they are machine independent.
c. Program statement resembles more like English and hence easier to work and less time is required to program.
d. Easy to debug.
e. Less prone to errors.

Disadvantages:

a. Slow execution of programs due to time taken by compilation
b. They use computer resources less efficiently.
c. They are difficult to relate to real world objects, hence less effective than object oriented programming
d. Memory required is more than that of 1GL or 2GL

D. **Problem-Oriented Language:**

They are the fourth generation language also called non-procedural language and are useful tools to solve specific problems and tasks. They require less training than 3GLs. Programmers and end-users use 4GLs to develop light-weight software applications. Some examples are : SQL, ACCESS, etc.

Advantages:

a. They are easier to learn and use than 3GLs.
b. They require less time, cost and effort to develop a program.
c. They require less instructions to perform a specific task.
d. They have a user-friendly interface.

Disadvantages:

a. They are executed at a slower speed by CPU.
b. They require more space in memory, disk storage of the computer system.
c. They are not suitable for all types of programming tasks.

E. **Natural Language:**

They are the 5th generation language and are still in the development phase. Natural language is a type of programming language that is designed to be easy for humans to read and write. It is based on natural, human language, rather than a formal, artificial programming syntax. They are specifically designed to use in AI applications.

Advantages:

a. Very simplified and easy way to communicate with the computer system.
b. There is no syntax at all.
c. It does not require any training to learn these languages.
d. Human questions are answered easily.

Disadvantages:

a. It may require a huge memory.
b. It may not recognize context.
c. Due to the ambiguous nature of human language, instructions are not properly executed.

Compiler, Interpreter and Assembler

1. **Assembler:** Assemblers are the system software that converts programs written in assembly language (source code) into machine language (object code). It also assembles the machine language program in the main memory of the computer, and makes it ready for execution.

2. **Compiler**: It is a translator program that translates the entire high-level language (source program) into its equivalent machine language program (object program) in a single operation.
3. **Interpreter**: It is a software that converts programs written in high-level language to machine language code.It converts one instruction at a time.

Differences between Compiler and Interpreter:

- **Execution**: A compiler translates the source code into machine code, which is then executed by the computer. An interpreter executes the source code directly.
- **Speed**: A compiled program typically runs faster than an interpreted program, because the machine code is optimized for the specific hardware and software environment in which it will be run.
- **Portability**: A compiled program can be run on any machine that has the necessary hardware and software, because the machine code is platform-independent. An interpreted program, on the other hand, can only be run on a machine that has an interpreter for the specific programming language.
- **Debugging**: It is usually easier to debug a program that has been compiled, because the machine code is easier for humans to read and understand than the source code. It can be more difficult to debug an interpreted program, because the source code is executed directly.
- **Errors**: Errors are not known until the compilation finishes at the end whereas in interpreter, syntax errors are known at each line while writing the program.
- **Uses**: Compilers are generally used by programming languages such as C, C++, Java, etc. Interpreters are used by Python, Ruby, etc.

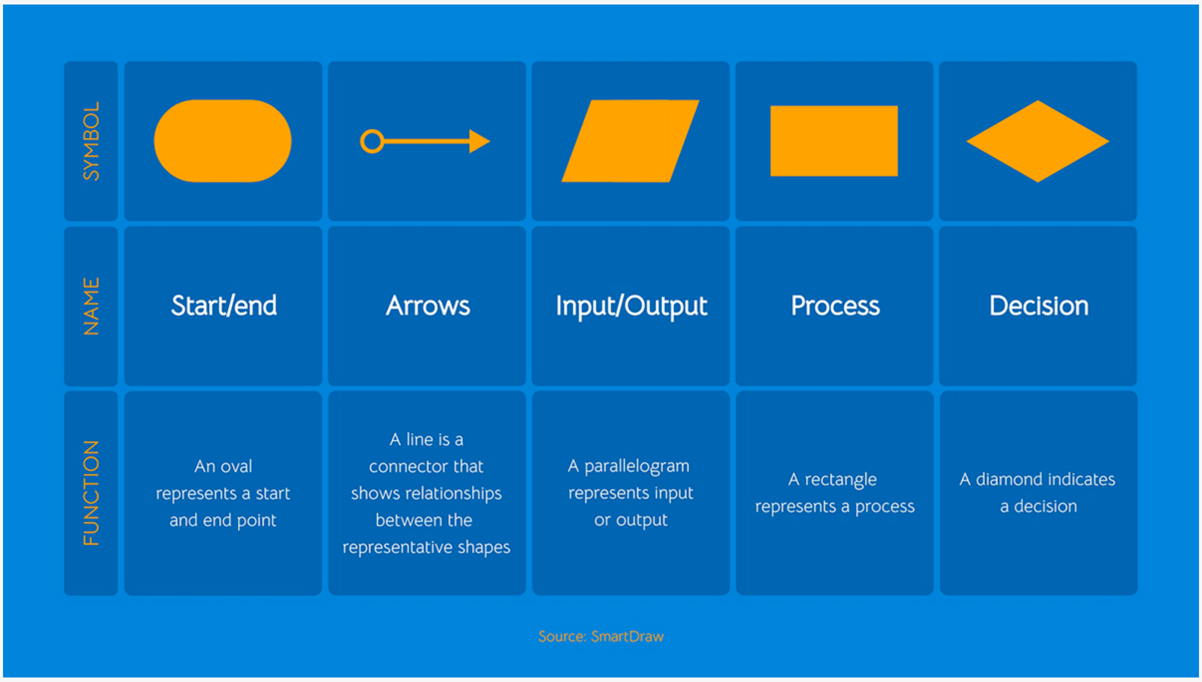
Overall, there are many different types of errors that can occur when programming in C, and it is important to be aware of them and to test and debug your code to ensure that it is free of errors.

Algorithms and Flowcharts:

Algorithms: Algorithms are a formula or set of steps for solving a particular problem. An algorithm is a set of well-defined instructions for performing a specific task or solving a problem. Algorithms can be written in any programming language, and they can be implemented as standalone programs or as part of larger software systems. They have the following advantages:

a. Efficiency: Designed to solve problems in a more efficient way.
b. Accuracy: They are less prone to errors on a subjective basis.
c. It can be repeated multiple times for different programs.
d. Easy sharing to other people.

Flowcharts: A flowchart is a graphical representation of a process or system that shows the steps or stages as boxes of various kinds, and their order by connecting these with arrows. Flowcharts are often used to document, design, analyze, and communicate complex processes or systems.
Flowcharts are useful because they can help to visualize and understand complex processes, and they can be used to communicate ideas and information to others in a clear and organized way. They can also help to identify problems or inefficiencies in a process, and to develop solutions or improvements.



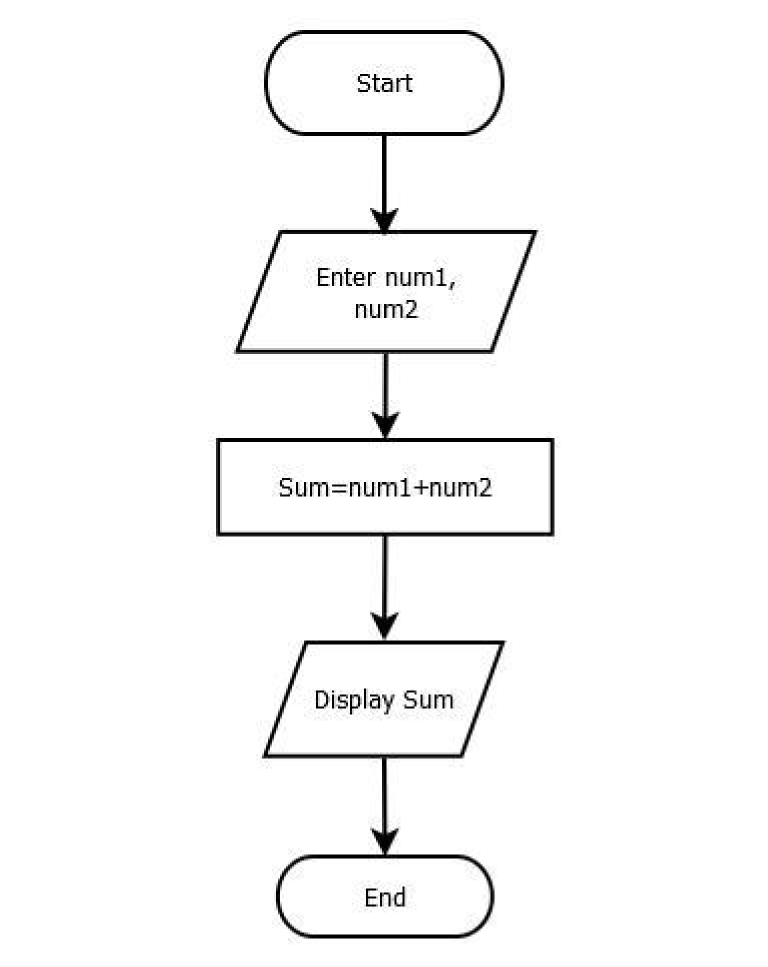
SYMBOL	⬭	○→	▱	▭	◆
NAME	Start/end	Arrows	Input/Output	Process	Decision
FUNCTION	An oval represents a start and end point	A line is a connector that shows relationships between the representative shapes	A parallelogram represents input or output	A rectangle represents a process	A diamond indicates a decision

Source: SmartDraw

Algorithm to add two numbers:

```
Step 1: Start
Step 2: Input two numbers and store it in variable a and b
Step 3: Add a and b and store it in variable c
Step 4: Print c
Step 5: Stop
```

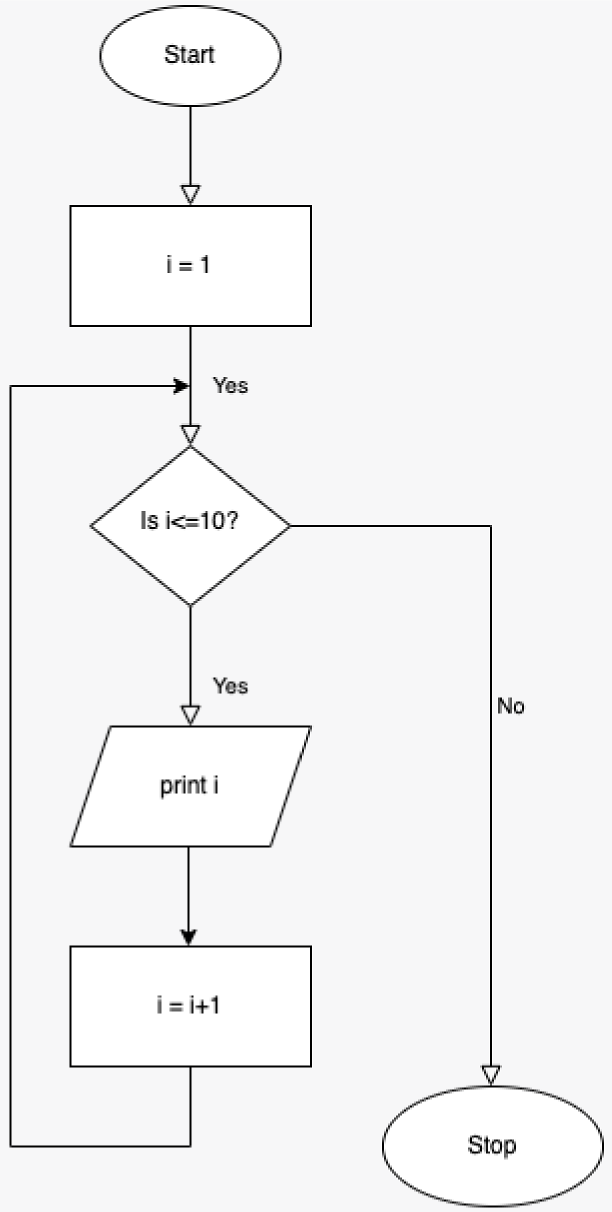
Flowchart:



Algorithm and flowchart to find sum of 10 natural numbers:

Algorithm:

```
1. Start
2. let i = 1
3. While i is less than or equal to 10, do the following:
      a. Print i
      b. i = i+1
4. Stop
```

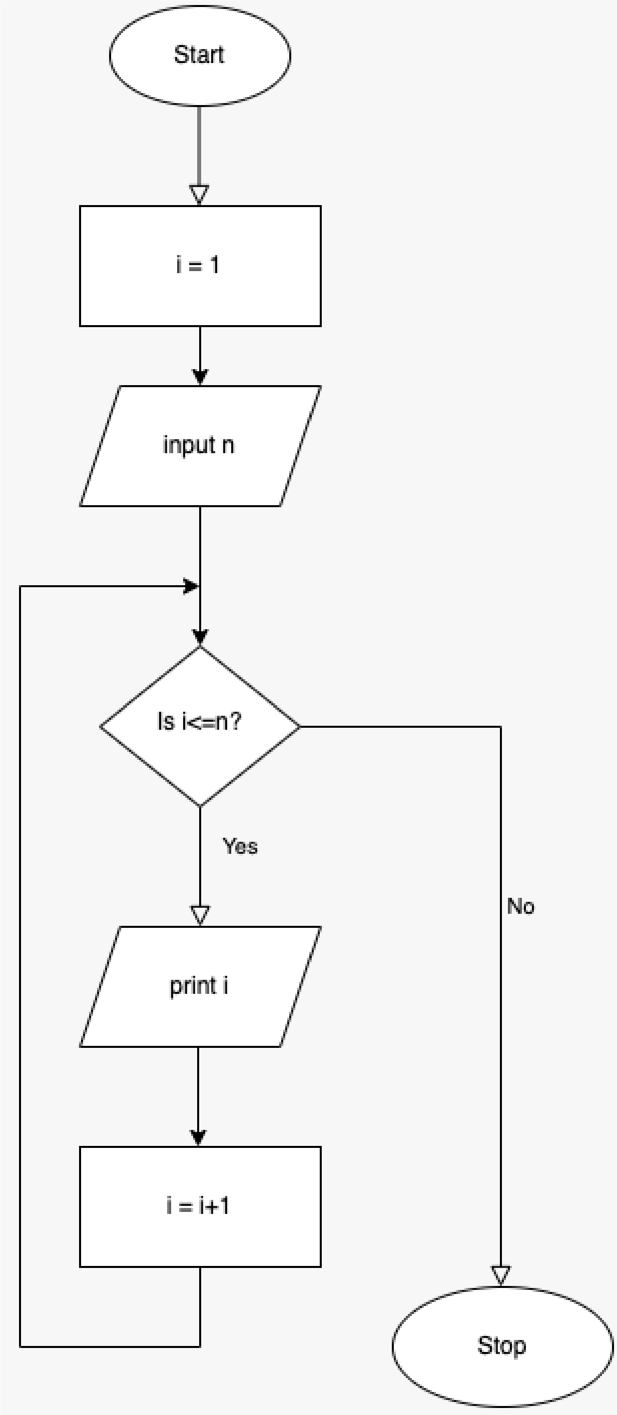


Algorithm and flowchart to find sum of n natural numbers:

Algorithm:

```
1. Start
2. let i = 1
3. Input n
4. While i is less than or equal to n, do the following:
    a. Print i
    b. i = i+1
5. Stop
```

FlowChart:



There are also some more algorithms and flowcharts discussed in class.

- Error (BUG):

There may be several errors inside a program which stops the program from being executed. Thus, an error in a program is known as a bug. This type of error in a program may arise by not following proper rules given by the programming language due to poor problem analysis or maybe due to hardware failure. Error in the program may be five ambiguous (not accepted/false) results. The process of removing errors from a computer program is known as debugging. There are three types of error:

1. **Syntax error:** the error which may arise due to not following proper rules or format given by the programming language. Every programming language has their own vocabulary, punctuation and structure. Programmers need to follow every detail while programming. This type of error can be detected by language processors such as compilers and interpreters. For e.g. In C, every statement should end with a semicolon at the end of the statement then the errors encountered are syntax errors.
2. **Logical error:** Logic refers to an idea or concept used to solve a specific problem. If the problem is poorly understood then it may hamper while solving the problem. The error that is encountered due to poorly developed logic is called logical error. This type of error is not detected by the language processor. For example: In order to calculate simple interest, we use the formula I =(P*T*R)/100, if we write above mentioned formula, as I=P*T*R, then the error encountered is a logical error.
3. **Runtime error:** When a program is running or executing, it is said to be runtime. Hence, the error that is encountered/found while the program is being executed is known as runtime error. It is also not detected by language translators. For example: if there is insufficient memory space, the peripheral device will not turn on.

INTRODUCTION TO C

-> C is a general-purpose, high-level programming language that was originally developed in the early 1970s by Dennis Ritchie at Bell Labs.

-> It is a procedural programming language, which means that the program is composed of procedures or functions that are executed step by step.

-> C programs are easy to learn, they are structured, produces efficient programs, handle low-level activities and can be compiled on various platforms

Uses:

- Portability: C code can be easily ported to different operating systems and hardware platforms, making it a popular choice for embedded systems and other applications that need to run on multiple platforms.

- Speed: C code is typically faster than code written in other high-level languages, as it is closer to machine code. This makes it a good choice for performance-critical applications.
- Low-level access: C provides low-level access to the underlying hardware, which makes it well-suited for system programming and other applications that need to directly interact with the hardware.
- Standardization: C has been standardized by the International Organization for Standardization (ISO) and American National Standards Institute (ANSI), which ensures compatibility and portability of the code written in C.

Advantages:

- Highly portable
- Lots of libraries
- Easy for beginners
- It is structured programming language
- It supports no. of operators
- Building block of many currently known languages

Disadvantages:

a. It doesn’t support OOP.
b. No runtime compilation
c. No concept of namespace
d. No constructors or destructors
e. No script type checking.

Type Specifiers

It is also called format specifier.
Compiler can understand that what type of data is in a variable during taking input using the scanf() function and printing using printf() function.

Format Specifier	Type
%c	Character
%d	Integer

%s	String
%f	Float

Identifiers:

In C programming language, an identifier is a name given to a variable, function, or any other user-defined item. Identifiers are used to uniquely identify these items and to reference them in the program.

An identifier in C must follow certain rules:

1. It must start with a letter or underscore (_), and can be followed by any combination of letters, digits, or underscores.
2. It must not contain any spaces or punctuation marks.
3. It must be a unique name within its scope.

Some examples of valid identifiers in C are:

- myVar
- _my_variable
- sumOfNumbers
- myFunction

Keywords:

Keywords in C programming language are reserved words that have a predefined meaning and cannot be used as identifiers (variable names, function names, etc.). These keywords are used to define the syntax and structure of the C language and are recognized by the compiler.

Here is the list of keywords:

auto break case char const continue default do

double else enum extern float for goto if

int long register return short signed sizeof static

struct switch typedef union unsigned void volatile while

It's important to avoid using these keywords as variable names, function names, or any other identifiers, as this can lead to syntax errors or unexpected behavior in the program.

Structure of C programming

a. **Preprocessor:** In C programming, a preprocessor is a program that runs before the main compilation process and performs text manipulation on the source code. Its main purpose is to provide a way to perform repetitive or conditional tasks that cannot be achieved with regular C syntax.

The preprocessor scans the source code for special directives that begin with a hash symbol (#), such as #include, #define, and #ifdef. These directives instruct the preprocessor to perform various actions, such as including header files, defining constants, or conditional compilation.

Eg: #include <stdio.h> , #define PI 3.14, #include <conio.h>

#include <stdio.h>

It will include stdio.h file into our source program which has the information for all input, output functions such as scanf( ),printf( ) etc

#include <conio.h>
It is a header file which is included in program to use clrscr(), getch() etc.

#include <math.h>
It is used for mathematical function such as pow(),sqrt() etc.

#include <string.h>
It is a string header file which contains the function definition of string processing functions such as strlen(),strcat() etc.

b. **Functions:** Functions are an essential structure in C programming that provide a structured way to organize code and promote code reuse. We can create our own functions such as :

```
#include <stdio.h>
void add(int a, int b) {
  int sum = a + b;
  printf("The sum of %d and %d is %d.", a, b, sum);
}
int main() {
    add(5,3);
add(6,5);
add(5,6);
    return 0;
}
```

/* Here main is also the function which is a mandatory function to use in C. */

c. **Variables:** In C programming language, a variable is a named storage location in memory that can hold a value of a particular data type, such as integers, characters, or floating-point numbers. Variables allow us to store and manipulate data in our program. For eg: int a = 5;
Here a is a variable that holds the integer value 5.

d. **Statements/Expressions:** In C programming language, statements and expressions are the building blocks of a program.
A statement is a complete unit of execution that performs a specific action, such as declaring a variable, calling a function, or looping through a block of code. **Every statement in C must end with a semicolon (;) character.**
Example: int num = 5; —> Statement
          printf("%d",num); —> Statement
All statements are expressions except initializations. For eg, int x;
This statement is not an expression because there is no value assigned to a variable.

e. **Comments:** In C programming language, comments are used to add notes and explanations to the code. Comments are ignored by the compiler and do not affect the program's execution. They are simply there to help human readers understand the code.
In C, there are two types of comments:

   a. Single-line comments: Single-line comments start with two forward slashes (//) and continue to the end of the line. Anything written after the // is considered a comment and is ignored by the compiler.

      Example:

```
// This is a single-line comment
int x = 42; // This line declares a variable and assigns it the value 42
```

   b. Multi-line comments: Multi-line comments start with /* and end with */. Anything written between the /* and */ is considered a comment and is ignored by the compiler. For example:

```
/* This is a
  multi-line comment
  that spans
  multiple lines */
int y = 13; /* This line declares a variable and assigns it the value 13 */
```

Data Types In C

In C, data types refer to the different kinds of values that can be stored and manipulated in a program. Data types specify the size and type of data that a variable can hold, as well as the operations that can be performed on that data. Some common types of data types are:

1. Integer data type: In C programming language, an integer data type is used to store whole numbers (positive, negative or zero) without a decimal point. It is a primitive data type and has a fixed range of values depending on the number of bits used to represent the integer. We will talk about long later which is a bigger form of int.

Example: int a = 10;

2. **Float Data Type:** In C programming language, the float data type is used to represent single-precision floating-point numbers, which are real numbers with a fractional part. The float data type in C has a size of 4 bytes (32 bits). We will talk about double later which stores more precision value than float.

Example: float pie = 3.14;

3. Character data type: In C programming language, the character data type (char) is used to represent a single character, such as a letter, number, or symbol. In C, characters are represented using ASCII encoding, which assigns a unique numerical value to each character. Example: char myLetter = 'A';

4. String Data Type: In C programming language, a string is a sequence of characters that is terminated by a null character \0. In other words, a string is an array of characters with a special character '\0' added at the end to mark the end of the string. The string data type is used to store and manipulate textual data in C. To declare and initialize a string variable in C, we can use the following syntax: char myString[] = "Hello, world!";

Data type	keyword	size	range
Character	**char**	**1 Byte**	**-128 to 127**
Integer	**int**	**2 Bytes**	**-32768 to 32767**
Float	**float**	**4 Bytes**	**1.2E-38 to 3.4E+38**
Double	**double**	**8 Bytes**	**2.3E-308 to 1.7E+308**

Operands and Operators and Operations:

Let us consider as;
S=A+B
Where S, A, B are operands
+, = are operators
S=A+B denotes sum of two operands A, B which is Operation

Types of operators:

It is a special symbol which helps to make operation successful i.e. it helps to make some specific operation.
Commonly used operators are:

- **Arithmetic operators: +, -, /, ***
  - These operators are used to perform arithmetic operations on numeric values, such as addition, subtraction, multiplication, division, and modulus. Here are some examples:

    ```
    int x = 10, y = 5;

    int sum = x + y; // Addition operator

    int diff = x - y; // Subtraction operator

    int prod = x * y; // Multiplication operator

    int quotient = x / y; // Division operator

    int remainder = x % y; // Modulus operator
    ```

- **Relative/Comparison operators: <, >, >=, <=, =**
  - The relational operators in C are used to compare two values and return a boolean result indicating whether the comparison is true or false. Here's an example of how you can use the relational operator in an `if` statement:

    ```
    #include <stdio.h>
    int main() {
      int a = 5, b = 10;
      if (a < b) {
        printf("a is less than b\n");
      }
      if (a > b) {
        printf("a is greater than b\n");
      }
      if (a == b) {
        printf("a is equal to b\n");
      }
      if (a != b) {
        printf("a is not equal to b\n");
      }
      return 0;
    ```

```
}
```

- **Logical operators: &&, ||, !**
  - In C, logical operators are used to perform boolean logic operations on two or more boolean values. The three logical operators in C are && (logical AND), || (logical OR), and ! (logical NOT). Here is an example of how you can use logical operators in an if statement:

```
#include <stdio.h>
int main() {
  int a = 5, b = 10, c = 15;
  if (a < b && b < c) {
    printf("a is less than b and b is less than c\n");
  }
  if (a > b || b > c) {
    printf("a is greater than b or b is greater than c\n");
  }
  if (!(a == b)) {
    printf("a is not equal to b\n");
  }
    return 0;
}
```

- **Assignment operator: =**
  - In C, the assignment operator = is used to assign a value to a variable. Here's an example:

```
#include <stdio.h>
int main() {
  int x = 10; // declare and initialize variable x to 10
  printf("The value of x is %d\n", x); // output: The value of x is 10
  return 0;
}
```

- **Conditional Operator: ?:**
  - The conditional operator in C is a ternary operator that takes three operands. It is used to evaluate a boolean expression and return one value if the expression is true, and a different value if it is false. The syntax for the conditional operator is:

    condition ? value1 : value2

    Here is an example of how to use the conditional operator in C:

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

```
int main() {
  int a = 5, b = 10;
  int max;
  max = (a > b) ? a : b;
  printf("The maximum value is %d\n", max);
  return 0;
}
```

The expression (a > b) ? a : b evaluates to a if the condition a > b is true, and b otherwise. If a is greater than b, the value of a is assigned to max. Otherwise, the value of b is assigned to max.

- **Comma Operator: ,**
    - The comma operator in C is a binary operator that is used to separate expressions.

    Here is an example of how to use the comma operator in C:

    ```
    #include <stdio.h>

    int main() {
      int x, y, z;
      x = 1, y = 2, z = 3;
      printf("x = %d, y = %d, z = %d\n", x, y, z);
      return 0;
    }
    ```

    In this example, we have three integer variables x, y, and z. In the first expression, we use the comma operator to assign the values 1, 2, and 3 to x, y, and z, respectively. The expressions are evaluated from left to right, but only the value of the last expression is returned.

- **Unary Operator: ++, –**
    - It is used to increment or decrement the value of a variable
    - For eg: i = i+1; can be written as i++;
    - I = i-; can be written as i–;

Simple Statement & Compound Statement

SImple Statement

A simple statement consists of an expression followed by a semicolon.
Eg:

```
printf(“Enater a number”);
 b=5;
```

Compound Statement

A compound statement consists of several individual statements enclosed in a pair of braces{}.
Eg.

```
{
  r=5;
  area=3.14*r*r:
}
```

Write a program in C to add two integers.

```
#include <stdio.h>

int main(){

        int num1, num2, Sum;

        printf(“Enter the first number”);

        scanf(“%d”,&num1);

        printf(“Enter the second number”);

        scanf(“%d”,&num2);

        Sum = num1 + num2;

        printf(“The sum of two numbers is %d”,Sum);

        return 0;

}
```

OUTPUT:

```
Enter the first number: 10
Enter the second number: 10
The sum of two numbers is 20
```

Write a program to find the area of a rectangle.

```
#include <stdio.h>

int main(){

        float l, b, Area;

        printf(“Enter the length”);

        scanf(“%f”,&l);

        printf(“Enter the breadth”);

        scanf(“%f”,&b);

        Area = l*b;

        printf(“The area of rectangle is %.2f”,Area);

        return 0;

}
```

```
OUTPUT:

Enter the length: 10.2

Enter the breadth: 5.2

The area of rectangle is 50.40
```

Write a program to find the simple interest.

```
#include <stdio.h>

int main(){

        float p, t, r, SI;

        printf(“Enter the principal”);

        scanf(“%d”,&p);
        printf(“Enter the rate”);

        scanf(“%d”,&r);

        printf(“Enter the time”);

        scanf(“%d”,&t);

        SI = (p*t*r)/100;;

        printf(“The Simple Interest is %.2f”,SI);

        return 0;

}
```

OUTPUT:

```
Enter the principal: 100

Enter the rate: 5

Enter the time: 1

The Simple Interest is 5.00
```

Find the square of the given number.

```
#include<stdio.h>
#include<conio.h>
void main(){
                int n,sq;
                printf(“Enter a number:”);
                scanf(“%d”,&n);
                sq=n*n;
                printf(“Square of a given number is %d”,sq);
                getch();
        }
```

OUTPUT:

```
Enter a number: 5
Square of a given number is 25
```

WAP to enter full name and print on screen.

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

```
#include<conio.h>
void main(){
        char name[100];
        puts("Enter your full name:");
        gets(name);
        puts(name);
        getch();
        }
OUTPUT:
 Enter your full name : Manish Dangi
 Manish Dangi
```

WAP to enter name,age and salary and print them on screen.

```
#include<stdio.h>
#include<conio.h>
void main(){
                char name[100];
                int age;
                float salary;
                printf("Enter name:");
                gets(name);
                printf("Enter age:");
                scanf("%d",&age);
                printf("Enter salary:");
                scanf("%f",&salary);
                printf("Name=%s",name)
                printf("\n Age= %d \n Salary=%.2f",age,salary);
                getch();
}
OUTPUT:
 Enter name : Hari Om
 Enter age: 25
 Enter salary:50000
 Name=Hari Om
Age=25
Salary=30000.00
```

WAP to input two numbers and print remainder and quotient.

```
#include<stdio.h>
#include<conio.h>
void main(){
                int n1,n2,r,q;
                printf("Enter two number:");
                scanf("%d%d",&n1,&n2);
                r=n1%n2;
                q=n1/n2;
```

```
                    printf(“Remainder is %d”,r);
                    printf(“Quotient is %d”,q);
                    getch();
            }
```

```
OUTPUT:
Enter two number: 7  2
Remainder is 1
Quotient is 3
```

WAP to input seconds and convert it into hours,minutes and seconds.

```
#include<stdio.h>
#include<conio.h>
void main(){
                    int s,h,m,r1,r2;
                    printf(“Enter seconds:”);
                    scanf(“%d”,&s);
                    h=s/3600;
                    r1=s%3600;
                    m=r1/60;
                    r2=r1%60
                    printf(“Hour = %d minutes=%d seconds=%d”,h,m,r2);
                    getch();
            }
```

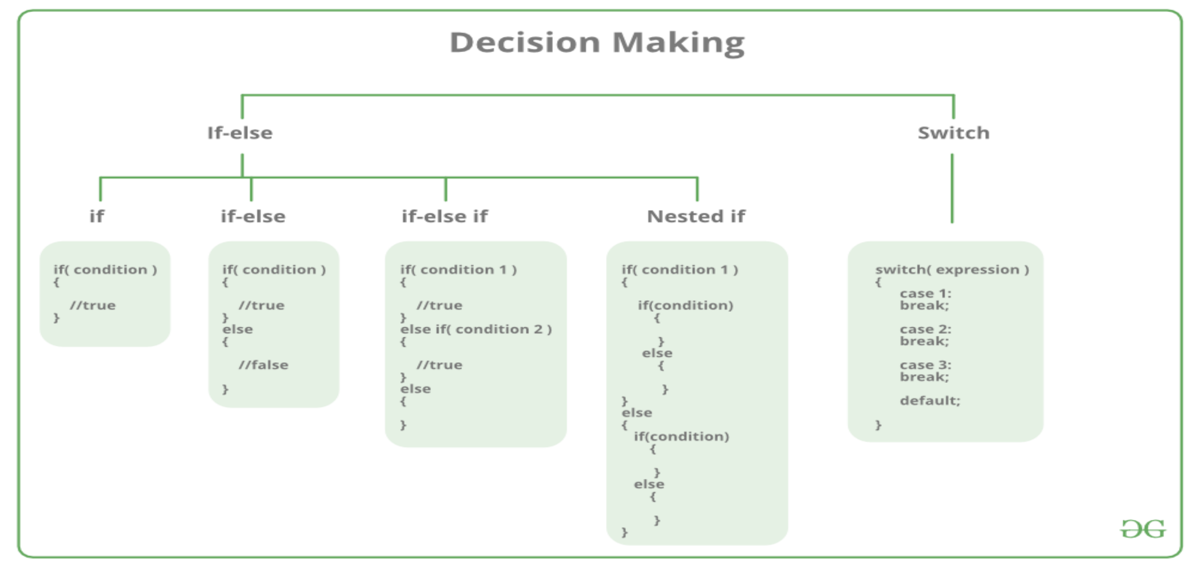
```
OUTPUT:
Enter seconds: 4000
Hour = 1 minutes=6 seconds=40
```

Calculate area of circle.

```
#include<stdio.h>
#include<conio.h>
void main(){
                    float r,a;
                    printf(“Enter  radius:”);
                    scanf(“%f”,&r);
                    a=3.14*r*r;
                    printf(“area=%.2f”,a);
                    getch();
            }
```

```
OUTPUT:
Enter radius: 5.0
Area= 78.50
```

Branching Statement(Selection Control Statement)



```
if( condition )
{
   //true
}
```

```
if( condition )
{
   //true
}
else
{
   //false
}
```

```
if( condition 1 )
{
   //true
}
else if( condition 2 )
{
   //true
}
else
{
}
```

```
if( condition 1 )
{
  if(condition)
    {
    }
   else
    {
    }
}
else
{
 if(condition)
   {
   }
  else
   {
   }
}
```

```
switch( expression )
{
    case 1:
    break;

    case 2:
    break;

    case 3:
    break;

    default;
}
```

1. if statement

- It is the most simple decision making statement.
- It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
if(condition) {
                 // Statements to execute if
                // condition is true
                }
```

Write a program to demonstrate the use of if statement.

```
#include<stdio.h>
#include<conio.h>
void main(){
                              int n;
                              printf("Enter  a number:");
                              scanf("%d",&n);
                              if(n>=1)
                              printf("\n Entered number is positive.");
                              getch();
                  }
```

OUTPUT:

```
Enter  a number: 5
Entered number is positive.
```

2. If else statement

- It is another form of selective control structure which can handle both expected as well as unexpected situations.
- If the condition is true, the statement written in if part will execute otherwise it will execute the statement written in else part.

Syntax:

```
if(condition) {
                 Statement 1;
                }
else{
        Statement 2;
      }
```

Write a program which reads any two no's and display the largest one

```
#include<stdio.h>
#include<conio.h>
void main(){
                              int n1,n2;
                              printf("Enter  two number:");
                              scanf("%d%d",&n1,&n2);
                              if(n1>n2)
                              printf("\n %d is larger",n1);
                              else
                              printf("\n %d is larger",n2);
                              getch();
                 }
```

OUTPUT:

```
Enter two number : 11  12
 12 is larger
```

3. if-else-if :

- When we have two or more conditions to be checked in a series we can use if-else if statement.
- It is also known as the if-else ladder.
- If none of the conditions are true, then the final else statement will be executed.

Syntax:

```
if (condition)
       statement;
else if (condition)
      statement;
          ……
           …..
else statement;
```

Write a program to find the largest number among three numbers.

```
#include<stdio.h>
#include<conio.h>
void main(){
                    int n1,n2,n3;
                    printf(“Enter  three numbers:”);
                    scanf(“%d%d%d”,&n1,&n2,&n3);
                    if(n1>n2&&n1>n3)
                    printf(“\n %d is larger”,n1);
                    else if(n2>n3)
                    printf(“\n %d is larger”,n2);
                    else
                    printf(“\n %d is larger”,n3);
                    getch();
            }
```

OUTPUT:

```
Enter three numbers: 1 2 3

 3 is larger
```

4. nested if else

- Nested if else statement means an entire if else statement written inside if part or else part of another if else statement.
- It is used when a condition is to be checked inside another condition at a time in the same program.

  Syntax:

```
         if (condition1) {
             if (condition2)
                   statement 1;
             else
                   statement 2;
         else
             statement 3;
                }
```

Write a C program that prompts the user to enter an integer and then uses a nested if-else statement to determine if the number is positive or negative, and if positive, whether it is even or odd. The program should output the results accordingly.

```
#include <stdio.h>

int main() {
  int num;

  printf("Enter a number: ");
```

```
    scanf("%d", &num);

    if (num > 0) {
        printf("Positive number.\n");

        if (num % 2 == 0) {
            printf("Even number.\n");
        } else {
            printf("Odd number.\n");
        }
    } else {
        printf("Negative number.\n");
    }

    return 0;
}
```

```
OUTPUT:
Enter a number: 3
Odd number
```

5. Switch case statements

- The switch statement is a multiway branch statement.
- It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
- Switch is a control statement that allows a value to change control of execution.

Syntax:

```
  switch (n) {
  case 1: // code to be executed if n = 1;
  break;
  case 2: // code to be executed if n = 2;
  break;
  default: // code to be executed if n doesn't match any cases
}
```

Example:

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int x = 2;
   switch (x)
   {
       case 1: printf("Choice is 1");
               break;
       case 2: printf("Choice is 2");
                break;
       case 3: printf("Choice is 3");
               break;
       default: printf("Choice other than 1, 2 and 3");
                break;
   }
   getch();
}
```

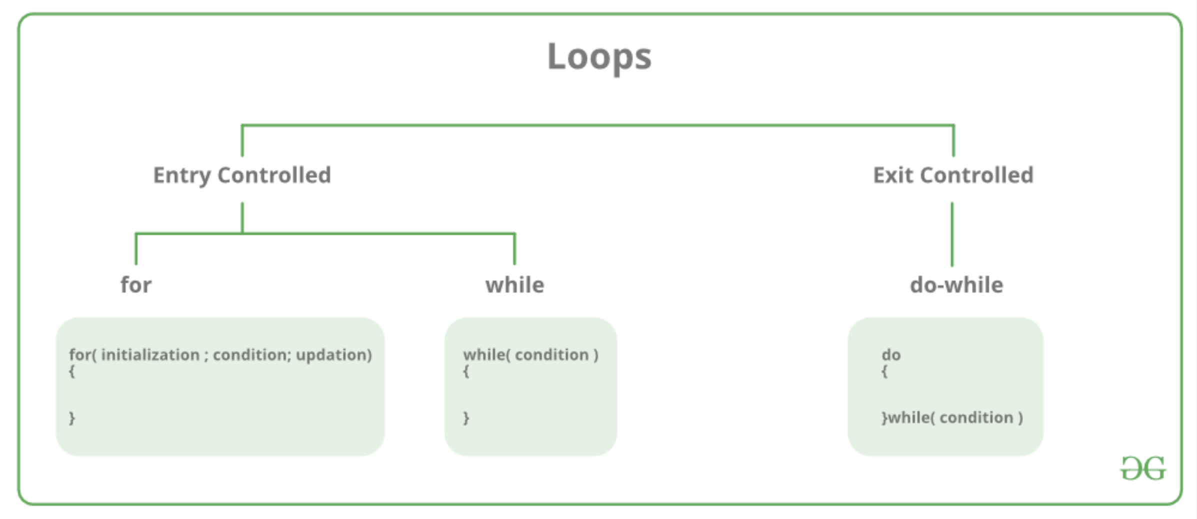
OUTPUT:

```
Choice is 2.
```

Write a program which reads any two integer values from user and calculates sum, difference and product using switch statement

```
#include <stdio.h>

int main() {
    int num1, num2;
    int choice;

    printf("Enter two integer values: ");
    scanf("%d %d", &num1, &num2);

    printf("Choose an operation to perform:\n");
    printf("1. Addition\n");
    printf("2. Subtraction\n");
    printf("3. Multiplication\n");
    printf("Enter your choice (1-3): ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Sum = %d\n", num1 + num2);
            break;
        case 2:
            printf("Difference = %d\n", num1 - num2);
            break;
        case 3:
            printf("Product = %d\n", num1 * num2);
            break;
        default:
            printf("Invalid choice!\n");
    }

    return 0;
}
```

Looping Statement
Looping is the process of executing the same program statement or block of program statements repeatedly for a specified number of times.



while loop:

- It executes the program statement repeatedly until the given condition is true.
- If checks the condition at first; if it is found true then it executes the statements written in the body part, otherwise it jump out of loop.
- It is also called entry control looping statements.

Syntax:

```
initialization;
while(condition){
statements;
increment/decrement;
}
```

Write a program to display “C is the best” 10 times.

```
#include<stdio.h>
#include<conio.h>
void main(){
                    int i=1;
                    while(i<=10)
                    {
                    printf(“\n C is the best”);
                    i++;
                    }
                    getch();
            }
```

OUTPUT:

Write a program to display numbers from 1 to 10.

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

Write a program to calculate and display the sum of the numbers from 1 to 10.

```
#include<stdio.h>
#include<conio.h>
void main(){
                    int i=1,sum=0;
                    while(i<=10)
                    {
                    sum=sum+i;
                    i++;
                    }
                    printf("\n Sum of the number from 1 to 10=%d",sum);
                    getch();
            }
OUTPUT:
```

do while statement:

- It executes the program statement repeatedly until the given condition is true.
- It executes the program statements at once then only it checks the condition.
- If the condition found true then it executes the program statements again ,otherwise it jumps out of the loop.
- It is also called exit control looping statements.

Syntax:

```
initialization;
do{
statements;
increment/decrement;
}
while(condition);
```

Write a program to display the series 1 6   11   16 …….101.

```
#include<stdio.h>
#include<conio.h>
void main(){
                        int i=1;
                        do{
                        printf("\n %d",i);
                        i=i+5;
                        } while(i<=101);
                        getch();
              }
OUTPUT:
```

Write a program to display the series 5 9 13 …...upto 10th term.

```
#include<stdio.h>
#include<conio.h>
void main(){
                        int i=1,n=5;
                        do{
                        printf("\n %d",i);
                        n=n+4;
                        i++;
                        } while(i<=10);
                        getch();
              }
OUTPUT:
```

Write a program to display a multiplication table of 6.

```
#include<stdio.h>
#include<conio.h>
void main(){
                        int i=1,p;
                        do{
                        p=6*i;
                        printf("\n 6*%d=%d",I,p);
                        i++;
                        } while(i<=10);
                        getch();
              }
OUTPUT:
```

for loop:

A for loop is a repetition control structure which is used to execute program

statements for a specific number of times.
It is an entry control looping statement.

Syntax:

```
for ( initialization;condition;increment/decrement )
 {
statement(s);
 }
```

C program to display numbers from 0 to 10 using a for loop.

```
#include <stdio.h>
#include<conio.h>
 void main () {
 int i;
 for(i=0;i<=10;i++)
 {
 printf("value of i: %d \n", i);
 }
getch();
 }
```

OUTPUT:

program to calculate and display the value of y raised to power x using for loop.

```
#include <stdio.h>
#include<conio.h>
 void main () {
    int i,x,y,z=1;
    printf(\n Enter the values for x and y:”);
    scanf(“%d%d”,&x,&y);
    for(i=1;i<=x;i++)
    {
     z=z*y;
    }
    printf(“\n %d raise to power %d: %d", y,x,z);
    getch();
 }
```

OUTPUT:

Program to calculate and display factorial of 5.

```
#include <stdio.h>
#include<conio.h>
 void main () {
```

```
 int i,fact=1;
 for(i=1;i<=5;i++)
 {
  fact = fact*i;
}
  printf(“Factorial of 5 is %d \n", fact);
getch();
 }
```

OUTPUT:

Nesting of Loop

When a loop is written inside the body of another loop then, it is known as nesting of loop. Any type of loop can be nested in any type such as while, do while, for. For example nesting of for loop can be represented as :

```
#include <stdio.h>
void main()
 {
int i,j;
 for(i=0;i<2;i++)
for(j=0;j<5; j++)
printf(“%d %d”, i, j);
}
```

OUTPUT:

program to display the following.

```
            1
            1         2
            1         2           3
```

```
#include <stdio.h>
#include<conio.h>
 void main () {
      int i,j;
      for(i=1;i<=3;i++)
      {
      for(j=1;j<=i;j++)
     {
        printf(“%d \t", j);
      }
            printf(“\n”);
   }
getch();
 }
```

OUTPUT:

Questions for practice:

```
1 1 1 1 1
2 2 2 2
3 3 3
4 4
5
```

```
5
4 4
3 3 3
2 2 2 2
1 1 1 1 1
```

```
5 4 3 2 1
5 4 3 2
5 4 3
5 4
5
```

Jumping Statements in C/C++

It is used to jump execution of program statements from one place to another place inside a program.

1. break statement:

- It is used to break the normal flow of program statement execution in loop and switch case statements.
- When a break is encountered in a program, the remaining part of the program is skipped and terminated.

Syntax:

break;

```
#include<stdio.h>
#include<conio.h>
void main(){
int i;
   for (i = 1; i<=5; i++) {
      if (i==4)
         break;
      printf(“%d\t”,i);
   }
}                        OUTPUT:
                         1  2  3
```

2. **continue statement:**

- **It is used to continue the normal flow of program statement execution in loop; skipping the iteration in the loop as soon as the certain condition is satisfied.**
- **When continue is encountered in program then that particular iteration is skipped and the loop will continued with next iteration.**

Syntax:

continue;

Example:

```
#include <stdio.h>
#include<conio.h>
void main(){
int i;
   for (i = 1; i<=5; i++) {
      if (i==4)
         continue;
      printf(“%d\t”,i);
   }
}                        OUTPUT:
                         1  2  3  5
```

3. **goto statement:**

When goto statement is encountered in a program then it transfer the control of program statement execution to the specified location by goto statement.

syntax:

label : statement

Where label is an identifier that is used to label the target statement to which the control is transferred.

program to display numbers from 1 to 10 using goto statement.

```
#include <stdio.h>
#include<conio.h>
 void main () {
 int i=1;
 label 1;
 printf(“%d\n”,i)
```

```
   i++;
 if(i<=10)
 goto label 1;
 getch();
 }
```

//Q. program to display even odd.

Some common programs:

Write a program to display the following Fibonacci series:

1 1 2 3 5…to nth term.

```
#include<stdio.h>
#include<conio.h>
 void main(){
  int i, a = 0, b = 1, c=1, n;
  printf("\n Enter the number of terms to be displayed:");
  scanf("%d",&n);
  for (i = 1; i <= n; i++)
  {
    printf("%d\t",c);
    c = a + b;
    a = b;
    b = c;
  }
  getch();
}
```

OUTPUT:

Write a program to calculate and display the sum of the digits present in the given number.

```
#include <stdio.h>
#include<conio.h>
void main()
{
  int n, r, sum = 0;
  printf("Enter an number:\n");
  scanf("%d", &n);
  do
  {
    r = n % 10;
    sum= sum + r;
    n = n / 10;
  }while(n!=0);
  printf("Sum of the digits= %d", sum);
```

```
   getch();
}
```

OUTPUT:

Write a program to check if the given number is palindrome or not.

```
#include<stdio.h>
#include<conio.h>
void main(){
int n, temp, sum=0, r;
printf("Enter a number: ");
scanf("%d", &n);
temp=n;
while(n!=0) {
r=n%10;
sum=sum*10+r;
n=n/10;
}
if(temp==sum)
printf("%d is a palindrome.", n);
else
printf("%d is not a palindrome.", n);
getch();
}
```

OUTPUT:

Write a program to check whether the given number is prime or composite.

```
#include<stdio.h>
#include<conio.>
int main() {
int i, n;
printf("Enter a number n");
scanf("%d", &n);
for (i=2; i < = n; i ++) {
        if(n % i = = 0)
        break;
         }
    if (i = = n)
    printf ("The number is PRIME");
   else
    printf ("The number is COMPOSITE");
 getch();
}
```

OUTPUT:

Write a program to display all prime numbers from 1 to 100.

```
#include <stdio.h>
#include<conio.h>
void main(){
int n, i, j;
for(n=1;n<=100;n++) {
      i=0;
     for(j=1;j<=n;j++) {
         if(n % j==0)
           i++;
       }
   if(i==2)
   printf("\n %d",n);
  }
}
```

OUTPUT:

ARRAY

Array is the collection of similar data types or collection of similar entities stored in contiguous memory locations. Array of characters is a string. Each data item of an array is called an element. And each element is unique and located in a separated memory location. Each element of an array shares a variable but each element has a different index no. known as subscript.

An array can be a single dimensional or multidimensional and the number of subscripts determines its dimension. And the number of subscripts always starts with zero. One dimensional array is known as vector and two dimensional arrays are known as matrix.

ADVANTAGES: array variable can store more than one value at a time where other variables can store one value at a time.

Example: int arr[100];
int mark[100];

DECLARATION OF AN ARRAY :
Its syntax is :
Data type array name [size];

```
int arr[100];
int mark[100];
int a[5]={10,20,30,100,5}
```

Total size in byte for 1D array is:
Total bytes=size of (data type) * size of array.
Example : if an array declared is:
int [20];
Total byte= 2 * 20 =40 byte.

ACCESSING OF ARRAY ELEMENT:

```
 /*Write a program to input values into an array and display them*/
 #include<stdio.h>
 int main() {
int arr[5], i;
//Reading elements
 for(i=0;i<5;i++) {
printf(“enter a value for arr[%d] \n”,i);
scanf(“%d”,&arr[i]);
}
//Displaying elements
 printf(“the array elements are: \n”);
for (i=0;i<5;i++) {
printf(“%d\t”,arr[i]);
}
return 0;
 }
```

OUTPUT:

```
Enter a value for arr[0] = 12
 Enter a value for arr[1] =45
Enter a value for arr[2] =59
Enter a value for arr[3] =98
Enter a value for arr[4] =21
 The array elements are 12 45 59 98 21
```

//WAP to add 10 array elements
//WAP to enter marks of five subjects and calculate average
//WAP to ask for 5 random numbers, store them in array and display only the even numbers

Two dimensional arrays:

Two dimensional array is known as a matrix. The array declaration in both the array i.e.in single dimensional array single subscript is used and in two dimensional array two subscripts are used.
Its syntax is :
Data-type array name[row][column];

Or we can say a 2-d array is a collection of 1-D arrays placed one below the other.
Total no. of elements in 2-D array is calculated as **row*column**

Example:-
int a[2][3];
Total no of elements=row*column is 2*3 =6
It means the matrix consist of 2 rows and 3 columns

For example:-

a[2][3]

2	3	4
5	6	7

Here, a[0][0]=2 , a[0][1]=3, a[0][2] = 4, a[1][0] = 5, a[1][1] = 6, a[1][2] = 7

Accessing 2-d array /processing 2-d arrays
For processing 2-d array, we use two nested for loops.
The outer for loop corresponds to the row and the inner for loop corresponds to the column.
For example
int a[4][5];

for reading value:-

```
for(i=0;i<4;i++) {
 for(j=0;j<5;j++) {
 scanf("%d",&a[i][j]);
 }
 }
```

For displaying value:-

```
 for(i=0;i<4;i++) {
for(j=0;j<5;j++) {
printf("%d",a[i][j]);
 }
}
```

2-D array can be initialized in a way similar to that of 1-D array.
for example:- int mat[4][3]={11,12,13,14,15,16,17,18,19,20,21,22};
These values are assigned to the elements row wise, so the values of elements after this initialization are
Mat[0][0]=11, Mat[1][0]=14, Mat[2][0]=17 Mat[3][0]=20
Mat[0][1]=12, Mat[1][1]=15, Mat[2][1]=18 Mat[3][1]=21
Mat[0][2]=13, Mat[1][2]=16, Mat[2][2]=19 Mat[3][2]=22

We can also give the size of the 2-D array by using symbolic constant Such as

```
 #define ROW 2;
 #define COLUMN 3;
 int mat[ROW][COLUMN];
```

Write a program to find the transpose of a matrix by taking row and column as input.

```
#include <stdio.h>

int main() {
    int rows, cols, i, j;

    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    printf("Enter the number of columns: ");
    scanf("%d", &cols);

    int matrix[rows][cols], transpose[cols][rows];

    printf("Enter the elements of the matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            printf(“Enter values for matrix A [%d][%d] :”,i+1,j+1);
            scanf("%d", &matrix[i][j]);
        }
    }

    printf("Original matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }

    // Transpose the matrix
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            transpose[j][i] = matrix[i][j];
        }
    }

    printf("Transposed matrix:\n");
    for (i = 0; i < cols; i++) {
        for (j = 0; j < rows; j++) {
            printf("%d ", transpose[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

OUTPUT:

In this program, the user is prompted to enter the number of rows and columns of the matrix. The matrix is then initialized as a 2-dimensional array of size rows x cols.

The program prompts the user to enter the elements of the matrix, which are stored in the matrix array using nested for loops. The original matrix is then printed using another set of nested for loops.

The program then uses another set of nested for loops to perform the transpose of the matrix. The transposed matrix is stored in a new 2-dimensional array called transpose.

Finally, the program prints the transposed matrix using nested for loops. The transposed matrix is printed as a cols x rows matrix, where the number of rows and columns are swapped from the original matrix.

//WAP in C to matrix addition and subtraction.

WAP in C to do matrix multiplication

```
#include <stdio.h>


int main() {

  int a[2][2], b[2][2], c[2][2];

  int i, j, k;


  // Ask user for matrix A

  printf("Enter values for matrix A:\n");

  for (i = 0; i < 2; i++) {
```

```
  for (j = 0; j < 2; j++) {
    printf("Enter value for A[%d][%d]: ", i, j);
    scanf("%d", &a[i][j]);
  }
}

// Ask user for matrix B
printf("Enter values for matrix B:\n");
for (i = 0; i < 2; i++) {
  for (j = 0; j < 2; j++) {
    printf("Enter value for B[%d][%d]: ", i, j);
    scanf("%d", &b[i][j]);
  }
}

// Matrix multiplication
for (i = 0; i < 2; i++) {
  for (j = 0; j < 2; j++) {
      c[i][j]=0;
    for (k = 0; k < 2; k++) {
      c[i][j] += a[i][k] * b[k][j];
    }
  }
}
```

```
  // Print the resulting matrix
  printf("Result matrix:\n");
  for (i = 0; i < 2; i++) {
    for (j = 0; j < 2; j++) {
      printf("%d ", c[i][j]);
    }
    printf("\n");
  }

  return 0;
}
```

OUTPUT:

String library function

There are several string library functions used to manipulate string and the prototypes for these functions are in header file “string.h”. Several string functions are

strlen()

This function return the length of the string. i.e. the number of characters in the string excluding the terminating NULL character.

It accepts a single argument which is pointer to the first character of

the string. For example-

strlen(“suresh”);

It return the value 6.

In array version to calculate length:-

```
 int str(char str[])
{
int i=0;

while(str[i]!='\o')
{
i++;
}
return i;
}
```

Example:-

```
#include<stdio.h>
 #include<string.h>
void main(){

char str[50]; print(”Enter a string:”);
gets(str);
printf(“Length of the string is %d\n”,strlen(str));
}
```

Output:

```
Enter a string: C in Depth
```

```
 Length of the string is 8
```

strcmp( )

This function is used to compare two strings. If the two strings match, strcmp() returns a value 0 otherwise it returns a non-zero value. It compares the strings character by character and the comparison stops when the end of the string is reached or the corresponding characters in the two strings are not the same.

strcmp(s1,s2)

return a value:

<0 when s1<s2

=0 when s1=s2

>0 when s1>s2

The exact value returned in case of dissimilar strings is not defined. We only know that if s1<s2 then a negative value will be returned and if s1>s2 then a positive value will be returned.

For example

```
/*String comparison */

#include<stdio.h>

#include<string.h>

int main(){
```

```
char str1[10],str2[10]; printf("Enter
two strings:");
 gets(str1);
gets(str2);
if(strcmp(str1,str2)==0){
printf("String are same\n");
}
          else{
               printf("Strings are not same \n");
}
          return 0;
}
```

OUTPUT:

This function is used for copying one string to another string. The function strcpy(str1,str2) copies str2 to str1 including the NULL character. Here str2 is the source string and str1 is the destination string.

The old content of the destination string str1 is lost. The function returns a pointer to destination string str1.

Example:-

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

```
#include<string.h>

int main(){

char str1[10],str2[10];

printf(“Enter a string:”);

scanf(“%s”,str2);

strcpy(str1,str2);

printf(“First string:%s\t\tSecond string:%s\n”,str1,str2);

strcpy(str,”Delhi”);

strcpy(str2,”Bangalore”);

printf(“First string :%s\t\tSecond string:%s”,str1,str2);

}

return 0;

}
```

OUTPUT:

strcat( )

This function is used to append a copy of a string at the end of the other string. If the first string is ”Purva” and second string is “Belmont” then after using this function the string becomes “PusvaBelmont”. The NULL character from str1 is moved and str2 is added at the end of str1. The 2nd string str2 remains unaffected. A pointer to the first string str1 is returned by the function.

Example:-

```
#include<stdio.h>

 #include<string.h>

void main(){

char str1[20],str[20];

printf(“Enter two strings:”);

 gets(str1);
gets(str2);
strcat(str1,str2);
printf(“First string:%s\t second string:%s\n”,str1,str2);
strcat(str1,”-one”);
printf(“Now first string is %s\n”,str1);

}
```

OUTPUT

```
Enter two strings: data Base

First string: database

second string: database `

 Now first string is: database-one
```

WAP to input full name and roll number of 5 students and print them using array

```
#include <stdio.h>

int main() {
    int i;
    char names[5][50];
    int roll_no[5];

    // Input the data for 5 students
    for (i = 0; i < 5; i++) {
        printf("Enter the full name of student %d: ", i + 1);
        scanf("%s", names[i]);

        printf("Enter the roll number of student %d: ", i + 1);
        scanf("%d", &roll_no[i]);
    }

    // Print the data for 5 students
    printf("\nStudent Data:\n");

    for (i = 0; i < 5; i++) {

        printf("Name: %s\n", names[i]);

        printf("Roll No.: %d\n\n", roll_no[i]);

    }



    return 0;

}
```

OUTPUT: