INDEX

NAME : B. Dheeraj Chandan STD.: SEC : ROLL No.

Sl.No.	Date	Title	Page No.	Teacher's Sign/ Remarks
		Basics of C++		
		CPP (C++) PROGRAMMING LANGUAGE		
		(BASICS & CONCEPTS)		
		NOTES BY - DHEERAJ		

3/10/21

1. Introduction to C++

History of C++ :

- C++ is a general purpose Programming Language.
- It was created by Bjarne Stroustrup as an extension of C language. It was developed in 1979 & Standardized in 1998 by International Organisation of Standardization.
- It supports Structured Programming, OOPs (Object Oriented Programmings) & Functional Programmings.

Features of C++ :

- It is an Object Oriented Programming (OOP) Language.
- It is faster than any other language (General purpose Programming language).
- It supports the pointers. (Useful in memory manipulation)
- Its latetest changes made it more dynamic with a lot more library support.

Applications of C++ :

- Operating Systems (OS)
- Embedded Systems
- Data bases
- Graphics / Game Design
- Servers

Advantages of C++ :

- object Oriented Programming language
- Easier to use than the other low level language like binary coding.
- It can support unions & structures.
- C++ does'nt use/support class methods.
- C++ is useful for Low Level programming languag
- C++ is relatively clear & mature standard.

Disadvantages of C++ :

- It is used for platform specific applications only.
- It doesn't support Garbage collection.
- It is not considered secure.
- It is complex to debug at certain applications
- Not easy to start & manage project/dependencies.

C++ Compilers :

- TURBO C++ (discontinued)
- CLANG
- GCC | GNU
- Visual C++
- MINGW

Q) C++ program to print "hello World".

```
#include<stdio
#include<iostream>
int main()
{
    std::cout << "hello World";
    return 0;
}
```

(or)

```
#include <iostream>
int main()
{
    std::cout << "hello World";
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello World" << endl;
    return 0;
}
```

(or) `cout << "Hello World"`

Preprocessors : Preprocessors are programs that process our source code before compilation.

Header File or Standard File — contains definition of predefined functions.

To Run a program : st1 - Goto Launchpad
st2 - Open playground
st3 - Write code & run it

Q) C++ program to take an input number or string & display it.

→
```
#include<iostream>
int main()
{
    int x;
    std::cout<<"Take number as an input :";
    std::cin>> x;
    std::cout<<x;
    std::cout<<"The number is :";
    std::cout<< x;
    return 0;
}
```

Q) C++ program to print your name.

→
```
#include<iostream>
int main()
{
    char s[20];
    std::cout<<"Take name as input : ";
    std::cin>>s;
    std::cout<<s;
    std::cout<<std::endl<<"Name is :";
    std::cout<<s;
    return 0;
}
```

endl → To print in next line

// → single line comment

/* */ → multi line comment

Data Types → int,

For execution, two types of Languages :- Compiled & interpreted

```
#include<iostream>
int main()
{
    int a=12;
    float b=3.5;
    std::cout<< age<<std::endl<< height; // instead of (std::endl) we can write "\n".
    return 0;
}
```

Data Types in C++ :

- User defined → Structure, Union, Enum, Class
- Built-in → int, char, float, double, boolean, void, Wide Character
- Derived → Array, Function, Pointer, reference

In Built-in type, Integral type are : int, char
Floating type are : float, double

Compiled & Interpreted language :

- Compiled Language → (i) It is a language whose implementations are typically compilers and not interpreters.
  (ii) At least two steps to get from code to execution.
  (iii) Compiled programs run faster than interpreted program.
  (iv) Complication errors prevents code from compiling.
  (v) It delivers good & better performance.
  (vi) Compiled language — C, C++, C#, COBOL etc—.

- Interpreted Language → (i) It is a language whose implementations execute instructions directly and freely, without previously compiling a program into machine language instruction.
  (ii) Only one step to get from source code to execution.
  (iii) Interpreted languages (programs) can be modified while the program is running.
  (iv) All the debugging occours at run-time.
  (v) It delivers relatively slower performance
  (vi) Interpreted language — Javascript, Python, Perl, etc—.

	Compiled		Interpreted
1.	Translated at once	→	Translated line by line
2.	Often considered fast	→	Often considered slow
3.	Debugging occurs before execution.	→	Debugging occurs during execution
4.	ex:- C++, C, C#	→	ex:- javascript, perl, python

4/12/21

2. Datatypes & Variables

Syntax rule for C++ :

- Must include Necessary Header files
- Don't forget main function
- Statements ends with semi-colon (;)
- Block of statements ends with '}' & starts with '{'

Variables : It is a name given to a memory location. It is the basic unit of storage in a program and value can be changed during execution of a program.

- Container/Storage block for storing values
- Basic syntax (data_type var_name = Value);
- ex:- int age = 20;
- C++ is a static typed.

Data Types : It defines the type of data a variable can hold. Data types in C++ are categorised in 3 groups i.e. : Built-in , User-Defined , derived .

User-defined → Structure , Union , Enum

primitive/Built in → int, char, float, double, boolean, void, Wide , Character

Derived → Array, function, Pointer

Type Modifiers are which modifies the type of data stored.

Keywords : keywords has special meaning to the C++ compiler and are always written or typed in short cases. These are words that the language uses for a special purpose such as void, int, public etc ... .

Signed data variables, stored both +ve & -ve values
Unsigned data variables, stores only +ve values.
Const data variables, Stored value can't be changed.

Variables Name: A variable is a name given to a memory location. The value stored in a value can be changed during program execution.

A variable name can only have letters (both uppercase & lowercase letters), digits & undersore. The first letter of a variable name should be either a letter or an undersore. There is no rule on how long a variable name can be. (identifier)

- Only alphabets, numbers
- Can't start with number or Uppercase
- Can't be a keyword.

```
#include <iostream>
int main()
{
    std::cout << "Variables & datatypes" << std::endl;
    int a = 12;
    unsigned int b1 = -12;
    signed int c1 = -3;
    unsigned int b2 = 12;
    signed int c2 = 3;
    const int d = 20;
    std::cout << "a: " << a << std::endl; // 12
    std::cout << "b1: " << b1 << std::endl; // 429.... 284
    std::cout << "c1: " << c1 << std::endl; // -3
    std::cout << "b2: " << b2 << std::endl; // 12
    std::cout << "c2: " << c2 << std::endl; // 3
    std::cout << "d: " << d << std::endl; // 20
    return 0;
}
```

Escape Sequences : These are special characters used in control string to modify the format of an output.

\n → next line
\t → tab, 5 spaces
\" → escap " , printing "
\\ → printing \

String → It is a collection of characters.

```
#include<iostream>
int main()
{
    char s[]="Dheeraj";
    std::cout<<s;
    return 0;
}
```

↳ To execute a string statement, we need "std::string"

```
#include<iostream>
int main()
{
    char s[]="Dheeraj";
    std::string str=" Chandan";
    std::cout<<s<<str<<std::endl;
    return 0;
}
```

8/1/22

3. Operators

Operators in C++ : Operators are symbols that perform specific ~~tasks~~ function.
It is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ is rich in built-in operators & provide following types of Operators :-

- Arithmetic operator
- Assignment Operator
- Relational Operator
- Logical Operator

→ Assignment operator is the operator used to assign a new value to a variable, This operator is idicated by '='.

ex:- int a=2;

→ Arthemetic operator is a type of operator which is used to perform common mathematical operations.

operator	name	uses	ex:
+	Addition	To add numbers or values	x+y
−	Substraction	To substract values	x−y
*	Multiplication	To find product of values	x*y
/	Division	To divide values or to find quotient	x/y
%	Modulus	It return remindor	x%y
++, −−	Increment, Decrement	It increses, decreases a valye by 1	++x, x++ −−y, y−−

→ Logical operators are the type of operators which are used to determine the logic between variables or values.

Operator	name	uses	ex
&&	AND	Returns true if both conditions are true.	(x<5) && (x<10)
||	OR	Returns False if both conditions are false.	(x<5) || (x<4)
!	NOT	Return true if condition is false & vice versa	! ((x<5) && (x<10))

→ Relational operators are the operators which is used to compare two values. It returns value of a comparision is either true or false.
Relational operators are also known as comparision operators.

Operator	Name	ex
==	Equal to	x == y
!=	not equal to	x != y
>	greater than	x > y
<	less than	x < y
>=	greater than equal to	x >= y
<=	less than equal to	x <= y

↳ In C++ we also use shorthand operator.
Shorthand operator are the operator which combines one of arthematic or bitwise operators with the assignment operators.

Operator	name	ex:-	expression
+=	Addition assignment	X+=4	X=X+4
-=	Substraction assignment	X-=4	X=X-4
*=	Multiplication assignment	X*=4	X=X*4
/=	Division assignment	X/=4	X=X/4
%=	Reminde assignment (or) Modulus assignment	X%=4	X=X%4

Truth Table for Logical Operators

AND →

X	Y	X && Y
0	0	0
0	1	0
1	0	0
1	1	1

OR →

X	Y	X || Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT →

X	Y	!X	!Y
0	1	1	0
1	0	0	1

→ Teonary Operator is a operator that evaluates the test condition & executes a block ~~that~~ of code based on the result of condition.

Syntax: variable = expression 1 ? expression 2 : expression 3

↳
```
#include <iostream>
using namespace std;
int main ()
{
    int a=10, b=5, c;
    bool res;
    cout << a+b << endl ;        // 15
    cout << ++a << endl ;        // 11
    cout << b-- << endl;         // 5
    res = a==b;
    cout << res << endl ;        // 0
    cout <<(a>b) << endl ;       // 1
    c=!((a>b) && (a<b));
    cout <<c<<endl ;             // 1
    return 0;
}
```

9/1/22

4. Conditional Statements

Case Structure

↳ In C++, there are 4 conditional statements :-

(i) → if statement
(ii) → if-else statement
(iii) → if-else ... if statement (if else ladder)
(iv) → switch statement

(i) It is used to execute a statement if the condition is true.

Syntax :-

```
if (condition)
{
    statements;
}
```

To specify a block of C++ code to be executed if a cond^n is true.

(ii) It is used to execute truth statement if condition is true. If condition is false then it will execute false statement.

Syntax:-

```
if (condition)
{
    Truth statement;
}else{
    False statement;
}
```

(iii) If else ... if statement is also called as if else ladder. It is used to execute statement when the condition that met is true. The program that executes the contents of the if portion of the statement. If the condition is not met, then the contents of the else statement will execute.

(iv) It is used to select on of many code of blocks to be executed.

= expression evaluated one

Syntax :-

```
switch (choice)
{
    Case 1 ;
        statement 1 ;
    break ;
    case 2
        statement 2;
        break ;
    :
    :
default:
    statement n;
    break ;
}
```

Syntax of if else ladder (if else if) :-

```
if (condition)
    statement ;
else if (Condition)
    statement ;
    :
    :
else if (condition)
    statement ;
else
    statement ;
```

Nested if else statement →

(i) It becomes complicated for multiple sections.
(ii) It uses independent expression for each case.
(iii) Test cond^n can be given in special range of values.

Switch statements →

(i) It is easy to understand for multiple selections
(ii) It uses single expression for add cases.
(iii) Only single expression is given in switch statement which returns single value.

↳
```
#include <iostream>
using namespace std;
int main()
{
    int a=10, b=5, age = 20;
    if (a >=18)                    // if statement
    {
        cout << "Elgible to vote" << endl;
    }
    if (a>b)                       // if-else statement
    {
        cout << a << ": is a greater." << endl;
    } else{
        cout << b << ": is a greater." << endl;
    }
    if (a>0)                       // ifelse if (if else ladder) statement
        cout << "Positive\n";
    else if(a>0)
        cout << "Negative\n";
    else
        cout << "zero\n";
    return 0;
}
```

```
#include <iostream>
Using namespace std;
int main()
{
    int n;
    cin >> n;
    switch (n)
    {
        case 1:
        cout << "Monday" << endl;
        break;

        case 2:
        cout << "Tuesday" << endl;
        break;

        case 3:
        cout << "Wednesday" << endl;
        break;

        case 4:
        cout << "Thursday" << endl;
        break;

        case 5:
        cout << "Friday" << endl;
        break;

        case 6:
        cout << "Saturday" << endl;
        break;

        case 7:
        cout << "Sunday" << endl;
        break;

        default:
        cout << "Invalid Choise" << endl;
        break;
}return 0;   }
```

Nested if statements :- It is a statement when an if statement or any conditional statement is placed inside an other conditional statement.

Syntax:-

```
if(condition)
{
    if(condition)
        statement;
}else{
    if(condition)
        Statements;
}
```

↳

```
#include<iostream>
int main()
{
    int a=10, b=5, c=15;
    if(a>b)
    {
        if(a>c)
        {
            std::cout<<a<<std::endl;
        }else{
            std::cout<<c<<std::endl;
        }
    }else{
        if(b>c)
        {
            std::cout<<b<<std::endl;
        }else{
            std::cout<<c<<std::endl;
        }
    }
    return 0;
}
```

18/1/22

5. C++ Loops - 1

Loops in C++ :

↳ It is used for executing a block of statements repeat-edly until a particular condition is satisfied.

↳ It is used to repeat a block of code.

In C++ there are 3 types of loops :

- for loop
- while loop
- do while loop

3 tools required for a loop :

- initialisation
- condition (or) check
- updation (or) change

↳ for loops and while loops are entry controlled loop.
do-while loops are exit controlled (dloop).

Syntax :

- for loop →

```
for (initialisation ; condition ; updation)
{
// statements ;
}
```

- while loop →

```
initialisation ;
while (condition)
{
//Statements ;
updation ;
}
```

```
initialisation ;
while (condition)
{
//statements;
updation ;
}
```

- Do while loop → initialisation;

```
do
{
    //statements;
    updation;
}
while (condition);
```

↳ For Loop is used when you know exactly how many times you want to loop through a block of code.

while loops can execute a block of code as long as a specified condition is reached

do-while loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as condition is true.

↳ Finite Loop :
- It iterates for a finite number of iterations
- It executes a statements for finites times.

Infinite Loop :
- It continues iterating for infinite number of times.
- It executes a statements for infinite number of times.

ex :

finite loop

```
int i;
for(i=0; i<5; i++)
{
    cout<<i<<endl;
}
```

infinite Loop

```
int i;
for(i=0; ; i++)
{
    cout<<i<<endl;
}
```

↳
```
#include <iostream>
using namespace std;
int main()
{
    for(int i=0; i<3; i++)
    {
        cout << "for loop : " << i << " iteration" << endl;
    }
    cout << "\n";
    int j=0;
    while (j<3)
    {
        cout << "While loop : " << j << " iteration" << endl;
        j++;
    }
    cout << "\n";
    int k=0;
    do
    {
        cout << "do-while loop : " << k << " iteration" << endl;
        k=k+1;
    }
    while (k<3);
    return 0;
}
```

↳ For infinite loops :-

```
for(int i=0 ; ; i++)
for(int i=0 ; 1 ; )
for( ; 1 ; )
for( ; ; )
```

```
int i=0;
while(i<3)
{
    //statements;
}
```

```
int i=0;
while(1)
{
    //statements;
}
```

Same as do-while

Note: In infinite loop always cond^n must be true.

24/1/22

6. C++ Loops - 2

Nested Loops :- It is a type of Loop which is placed inside other loop.

↳ It means a loop is placed inside another loop.

↳ Nested loops are useful when for each case pass through the outer loop, you need to repeat some action on the outer loop.

↳ We need & use nested loops in : (max.. cases) in
- patterns (Build patterns)
- Arrays (2-D array)

Array :→ It stores a fixed-size sequential collection of elements of the same type.

→ It is used to collection of data

→ It stores homogenous or same type of data elements.

1D declaration - `int a[3];`

Syntax: `data_type var_name[size];`

ex:- `int a[3] = {1,2,3};`

2 Types of array → 1. 1D Array
2. 2D Array

2D declaration -

Syntax : `data_type var_name[r_size][c_size];`

ex:- `int a[3][3] = {{1,2,3},{4,5,6},{7,8,9}};`

```
→ #include <iostream>
using namespace std;
int main()
{
    for(int i=0; i<3; i++)
    {
        for(int j=0; j<3; j++)
        {
            cout << "* ";
        }
        cout << endl;
    }
    cout << endl;
    for(int i=0; i<3; i++)
    {
        for(int j=0; j<=i; j++)
        {
            cout << "* ";
        }
        cout << endl;
    }
    cout << endl;
    for(int i=0; i<=3; i++)
    {
        for(int j=1; j<=3; j++)
        {
            cout << j << " ";
        }
        cout << endl;
    }
    cout << endl;
    for(int i=0; i<3; i++)
    {
        for(int j=3; j>i; j--) {
            cout << " "; }
        for(int k= i+1; k>0; k--){
            cout << "* "; }
        cout << endl;
    } return 0;
}
```

→

```
#include<iostream>
Using namespace std;
int main()
{
    int a[3]={1,2,3};
    for(int i=0; i<3; i++)
    {
        cout<<a[i]<<" ";
    }
    cout<<endl;
    int b[3][3]={{1,2,3},{4,5,6},{7,8,9}};
    for(int i=0; i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            cout<<b[i][j]<<" ";
        }
        cout<<endl;
    }
    cout<<endl;
    int c[2];
    c[0] = 1;
    c[1] = 2;
    c[2] = 3;
    cout<< c[0]+c[1]+c[2]<<endl;
    cout<<endl;
    int d[2][2];
    d[0][0] = 1;
    d[0][1] = 2;
    d[1][0] = 3;
    d[1][1] = 4;
    cout<<((d[0][0] + d[0][1])-(d[1][0] + d[1][1]))<<endl;
    return 0;
}
```

19/2/22

```
#include <iostream>
using namespace std;
int main()
{
    string name = "Dheeraj";
    for(int i=0; i<name.length(); i++)
    {
        cout << name[i] << endl;
    }
    cout << "\n";
    int numbers[] = {1, 2};
    for(int i=0; i<2; i++)
    {
        cout << numbers[i] << endl;
    }
    cout << "\n";
    int num[2][2] = {{1,2},{3,4}};
    for(int i=0; i<2; i++)
    {
        for(int j=0; j<2; j++)
        {
            cout << num[i][j] << " ";
        }
        cout << "\n";
    }
    cout << "\n";
    string names[2][2] = {{"0","1"},{"Dheeraj","Chandan"}};
    for(int i=0; i<2; i++)
    {
        for(int j=0; j<2; j++)
        {
            cout << names[i][j] << " ";
        }
        cout << "\n";
    }
    cout << "\n";
    return 0;
}
```

Nested for loop →

Syntax :

```
for(initialisation1; condition1; updation1)
{
    for(initialisation2; condition2; updation2)
    {
        statements;
    }
    // statements if required;
}
```

ex:-

```
int a[2][2] = {{1,2},{3,4}};
for(int i=0;i<2;i++)
{
    for(int j=0;j<2;j++)
    {
        std::cout<<a[i][j]<<" ";
    }
    std::cout<<"\n";
}
```

Note:- We most of the time we use nested for loop.
For loop is most popularly used than while & do while loops.

27/2/2022

```
#include<iostream>
using namespace std;
int main()
{
    int a[]={1,2,3,4,5};
    for(int i=0; i<5; i++)
    {
        if(a[i]%2==0)
        {
            cout<<"Even number is at index :"<<i<<endl;
        }
    }
    cout<<endl;
    char s[]={'a','b','a','c','a'};
    for(int i=0; i<5; i++)
    {
        if(s[i]=='a')
        {
            cout<<"'a' charater is at index:"<<i<<endl;
        }
    }
    cout<<endl;
    string str="B-Dheeraj Chandan";
    for(int i=0; i<str.length(); i++)
    {
        if(str[i]=='a')
        {
            cout<<"'a' character is at index :"<<i<<endl;
        }
    }
    return 0;
}
```

27/2/2022 7. Functions

Function :→

- A function is a code module that performs a single task. (specific task)
- It is a set of statements that take inputs to do some specific task.
- It gives/has return type.

Needs of functions :-

- We need functions to improve the readability of code.
- We need functions to improve reusability of the code.
- If can be used in any other program rather than writing the same code from scratch.
- Easy to understand code & debugging of code would be easier.

Functions are of 2 parts (types) :- i. Standard function (- 3rd party)
ii. User defined function

(ii) User defined function → It is a function which allows the programmer to define their own functions.

ex:-
```
#include <iostream>
int add(int a, int b)          // User defined functions, we can define f with any name.
{
    return (a+b);
}
int main()
{
    int sum;
    sum = add(10,20);
    std::cout << sum;
}
```

Basic declaration :
```
return_Type function_Name()
{
    // block of code;
}
```

↳ The User defined function are of 4 types/categories :-

1. Function with no arguments & no return type
2. Function with no arguments & with return type
3. Function with with arguments & no return type
4. Function with with arguments & with return type

1. ↳ Function has no arguments.
   ↳ It does not receive any data from calling function
   ↳ when it does not return value, the calling function does not receive any data from the called function.
2. ↳ Function that may not take any argument but ↳ returns a value to the calling function
3. ↳ Function that has arguments.
   ↳ It receives data from calling function.
   ↳ It returns no value.
4. ↳ Function having arguments
   ↳ It receives data from calling function
   ↳ Returns value to calling function

Called function → A function is a block of code that performs some operation. A block of code which only runs when it is called.

Calling function → When a program calls a function, program control is transferred to the called function.

Function arguments → Function argument in C++ are passed by default as "Pass by value". The object being passed into the function is copied & then that copy is destroyed when function returns. An argument in a function is a value that must be provided to obtain the function's result.

↳ Arguments in a function must declare a variable that accepts the value of the arguments. These Variable are called formal Parameters. Parameters that are values which determined by the function that accepts values when the function is declared.

↳ The Actual parameters are the variable that are transfered to the function when it is requested. The arguments that are passed in function call are called as actual argument.

↳ While calling a function there are 2 ways that argument can be passed to a function:-

(i) Call by Value → This method copies the actual value of an argument into the formal parameter of the function. Changes made to the parameter inside function have no effect on argument.

(ii) Call by reference → This method copies the reference of an argument into the formal parameter. The reference is used to access the actual argument used in the call. Changes made to the parameter affect the argument.

(iii) Call by Pointer → This method copies address of the argument into the formal parameter. This address is used to access the actual argument used in a call. Changes made to the parameter will affect the argument.

↳ // With argument, With return Type

```
#include <iostream>
int a(int x, int y)
{
    int sum= x+y;
    return sum;
    // return 0;
}
int main()
{
    int add =a(10,20);
    std:: cout << add;
}
```

↳ // With arguments, no return type

```
#include <iostream>
Void a(int x, int y)
{
    int sum=x+y;
    std:: cout << sum;
}
int main()
{
    a(10,20);
    return 0;
}
```

↳ // No argument, with return type.

```
#include <iostream>
int a()
{
    int sum = 10+20;
    return sum;
    return 0;
}
int main()
{
    int add = a();
    std :: cout << add;
}
```

↳ // No argument, no return type

```
#include <iostream>
void a()
{
    int sum = 10+20;
    std :: cout << sum;
}
int main()
{
    a();
    return 0;
}
```

(i) Standard functions → C++ Standard library functions which provides a rich collection of functions for performing common mathematical calculations, string manipulations, character manipulations, i/o, error checking & other useful operations. This makes programmer's job easier because functions provide many of capabilities prog-rammers need.

Note :— Header file names ending with ".h" are 'old style' header files that have been superseded by the C++ Standard Library header files.

Most popular C++ Standard Library header files used:

- <iostream> : Contains function prototypes for C++ standard inputs & outputs functions. (Replaces headerfile - <iostream.h>)
- <cmath> : Contains function prototypes for math library functions. (Replaces header file - <math.h>)
- <cstdlib> : Contains function prototypes for conversion of numbers to text, text to numbers, memory allocation, random numbers & various other utility functions. (Replaces header file - <stdlib.h>)
- <ctime> : Contains function prototypes & types for manipulating the time & date. (Replaces header file - <time.h>)

→ <cctype> : contains function prototypes for functions that tests characters for certain properties (such as whether character is a digit or a punchuation) & can be used to convert lowercase & Uppercase letters.
(Replaces header file – <ctype.h>)

→ <cstring> : Contains function prototypes for C-style string processing functions.
(Replaces header file – <string.h>)

→ <string> : Contains the definition of class string from the C++ standard library.

Mathematical Functions → Functions in header file <cmath>

→ sin(x) : Sine of angle X.
→ Cos(x) : Cosine of angle X.
→ Tan(x) : Tangent of angle X
→ asin(x) : Sin^{-1} of angle X
→ acos(x) : cos^{-1} of angle X
→ exp(x) : Exponential of angle X (e^x)
→ log(x) : Logarithm of X
→ log 10(x) : Logarithm of X to base 10
→ sqrt(x) : Square Root of X
→ pow(x, y) : X to the power of Y
→ abs(x) : Absolute value of integer number X.
→ fabs(x) : Absolute value of real nomber X.

Character Functions ⟶ All the character functions require <cctype> header file.

- isalpha(c) : It checks if character is an alphabet or not. Returns True if character is alphabet else returns False.
- isdigit(c): It returns if 'c' is a digit else it returns false.
- isalnum(c): It returns True if 'c' is digit or alphabetic character (upper or lower) otherwise returns False.
- islower(c): It returns True if 'c' is lowercase else it returns False.
- isupper(c) : It returns True if 'c' is uppercase else it returns False.
- toupper(c) : It converts 'c' to uppercase letter
- tolower(c) : It converts 'c' to lowercase letter.

```
#include <iostream>
#include <cmath>
#include <cctype>
using namespace std;
int main()
{
    int a=100;
    cout<<"Input number : "<<a<<endl<<endl;
    cout<<" Sine "<<sin(a)<<endl
    cout<<" Cosine "<<cos(a)<<endl
    cout<<"Tangent "<<tan(a)<<endl
    cout<<" Sine Inverse "<<asin(a)<<endl
    cout<<" Cosine Inverse "<<acos(a)<<endl
    cout<<" Exponential "<<exp(a)<<endl
    cout<<" Logarithm "<<log(a)<<endl
    cout<<" Logarithm base 10"<<log10(a)<<endl
    cout<<" Square Root "<<sqrt(a)<<endl
    cout<<" To the power 2"<<pow(a,2)<<endl
    cout<<" Absolute of Integer"<<abs(a)<<endl
    cout<<" Absolute of real "<<fabs(a)<<endl<<endl;
    char c1='a', c2='5';
    cout<<"Input character c1 : "<<c1<<endl<<endl;
    cout<<" c1 is Alpha "<<isalpha(c1)<<endl;
    cout<<" c1 is Digit "<<isdigit(c1)<<endl;
    cout<<" c1 is alnum "<<isalnum(c1)<<endl;
    cout<<" c1 is Lower "<<islower(c1)<<endl;
    cout<<" c1 is Upper "<<isupper(c1)<<endl;
    cout<<" c1 to upper "<<toupper(c1)<<endl;
    cout<<" c1 to lower "<<tolower(c1)<<endl<<endl;
```

```
cout<<"Input character c2 : "<<c2<<endl<<endl;
cout<<"  c2 is Alpha   "<<isalpha(c2)<<endl;
cout<<"  c2 is Digit   "<<isdigit(c2)<<endl;
cout<<"  c2 is alnum   "<<isalnum(c2)<<endl;
cout<<"  c2 is Lower   "<<islower(c2)<<endl;
cout<<"  c2 is Upper   "<<isupper(c2)<<endl;
cout<<"  c2 is upper   "<<toupper(c2)<<endl;
cout<<"  c2 is lower   "<<tolower(c2)<<endl<<endl;
}
```

21/5/22

8. Function overloading and Recursion

→ C++ allows you to specify more than one definition for a function name or an operator which is called as function overloading and Operator overloading.

→ Function overloading is defined as the process of having two or more function with the same name but having different parameters (of same name with different type)

→ C++ has ability to provide the operators with a special meaning for a data type, this ability is known as Operator Overloading.

→ The process of selecting the most appropriate overloaded function or operator is called overload resolution.

→ If we call this function using the object of the derived class, the function of derived class is executed. This is known as function overriding. If derived class defines same function as defined in its base class then it is known as function overriding.

↦

```
#include <iostream>
using namespace std;
void task1()
{
    cout<<"This is task1"<<endl;
}
void task2()
{
    cout<<"This is task2"<<endl;
    task1();
}
int main()
{
    task2();
    return 0;
}
```

```
#include<iostream>
Using namespace std;
int add()
{
    cout << "20+30 = " << 20+30 << endl;
    return 0;
}
int add(int a)
{
    cout << "User Input is : " << a << endl;
    return 0;
}
int add(double a)
{
    cout << "User Input is : " << a << endl;
    return 0;
}
int add (int a, double b)
{
    double sum= a+b;
    cout << "sum of numbers is : " << sum << endl;
    return 0;
}
Void add(string name)
{
    cout << "My name is : " << name << endl;
}
Void add(string name, int age)
{
    cout << "Name : " << name << ", \t Age" << age << endl;
}
int main()
{
    add();
    add(10);
    add(15.0);
    add(10, 15.0);
    add("Dheeraj");
    add("Dheeraj", 22);
    return 0;
}
```

Recursive :-

→ When function is called within the same function it is known as Recursion.

→ The function which calls the same function is known as Recursive function.

→ A function that calls itself, & doesn't perform any task after function call is known as tail recursion.

```
// Factorial of a number.
#include<iostream>
using namespace std;
int fact(int n)
{
    if(n==0)
    {
        return 1;
    }else{
        return n*fact(n-1);
    }
    //return 0;
}
int main()
{
    int num;
    cin >> num;
    cout << fact(num);
    //return 0;
}
```

```
#include<iostream>
Using namespace std;
int array(int arr[3])
{
    return arr[1];
    // return 0;
}
int main()
{
    int a[3]={10,20,30};
    cout<<array(a);
    //return 0;
}
```

```
// Pass array & array element to different functions.
#include<iostream>
using namespace std;
int array1(int n)
{
    return n;
}
Void array2(int arr[3])
{
    for(int i=0;i<3;i++)
    {
        cout<<arr[i]<<endl;
    }
}
int main()
{
    int a[3]={10,20,30};
    cout<<array1(a[1])<<endl<<endl;
    array2(a);
    return 0;
}
```

18/6/22

9. Arrays

Array :-

↳ It is a collection of similar data items stored at contiguous memory locations and elements that can be accessed randomly using indices of an Array.

→ They can be stored as collection of primitive data types such as int, float, double, char, etc....

→ It stores a fixed sized sequential collection of elements of the same type.

→ It is a continous collection of homogeneous data.

→ Array Index : The location of an element in an array has an index, which identifies the element. Array index starts from 0.

Array element : Items stored in an array is called an element. The elements can be accessed via its index.

Array length : The length of an array is defined based on number of elements an array can store.

Syntax :-

```
data_type var_name[] = {element1, element2, ...., element n};
```

→ Declaration of Array size :

```
int a[8];
int n=5;
int a[n];
```

→ Initializing of Array :

```
int a[] = {1,2,3}
int a[3] = {1,2,3}
```

4

1 →
```
#include <iostream>
using namespace std;
int main()
{
    int a1[] = {1,2,3};
    int a2[3] = {1,2,3};
    int a3[5] = {10,20,30};
    cout<< a1[0]<<','<<a1[1]<<','<<a1[3]<<endl;
    cout<< a2[0]<<','<<a2[2]<<','<<a2[4]<< endl;
    cout<< a3[0]<<','<<a3[2]<<','<<a3[6]<< endl;
    string s[3] = {"Hi","Hello","Bye"};
    cout<<s[0]<<','<<s[1]<<','<<s[2]<<endl;
    return 0;
}
```

26/1/23

→
```
#include<iostream>
using namespace std;
int main()
{
    int a[] = {1,2,3};
    int b[3] = {10,20,30};
    int ArrayLenOf_b = sizeof(b)/sizeof(int); //sizeof(b[0])
    cout<<"Length of array a - "<<sizeof(a)/sizeof(int)<<"\n";
    cout<<"Length of array b - "<<ArrayLenOf_b<<"\n";
    for(int i=0; i<3; i++)
    {
        cout<<a[i]<<":"<<b[i]<<"\n";
    }
    return 0;
}
```

→
```
#include<iostream>
#include<array>
using namespace std;
int main()
{
    std::string s[]={"Balivada","Dheeraj","Chandan"};
    char c[3]={'a','b','c'};
    int ArrayLenOf_s = sizeof(s)/sizeof(string); //sizeof(s[0]), sizeof(std::string)
    cout<<"Length of array c - "<<sizeof(c)/sizeof(char)<<"\n";
    cout<<"Length of array s - "<<ArrayLenOf_s<<"\n";
    for(int i=0;i<ArrayLenOf_s;i++)
    {
        cout<<c[i]<<":"<<s[i]<<"\n";
    }
    int a[]={1,2,3};
    int b[5]={10,20,30,40,50};
    int ArrLen_a = sizeof(a)/sizeof(a[0]);
    int TotArrLen_ab = ArrLen_a +(sizeof(b)/sizeof(b[0]));
    int ab[TotArrLen_ab];
    for(int i=0;i<ArrLen_a;i++)
    {
        ab[i] = a[i];
    }
    for(int i=ArrLen_a;i<TotArrLen_ab;i++)
    {
        ab[i]= b[i-ArrLen_a];
    }
    for(int i=0;i<TotArrLen_ab;i++)
    {
        cout<<ab[i]<<" ";
    }
    cout<<"\n";
    std::array<int,3> m={100,200,300};
    cout<<"Array size of m - "<<m.size()<<"\n";
    cout<<"Array front of m - "<<m.front()<<"\n";
    cout<<"Array back of m - "<<m.back()<<"\n";
    return 0;
}
```

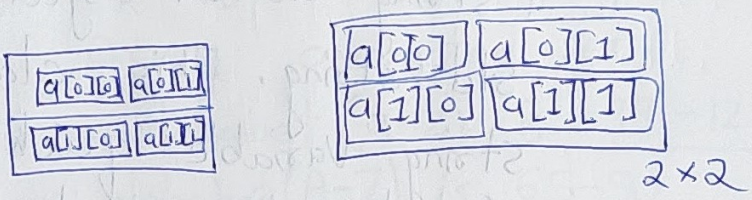
Multi - Dimension Array :-

↳ It is termed as an array of arrays that stores homogeneous data in tabular form.

↳ They are stored in row major order.

Syntax : datatype Variable_name[size1][size2] = {{},{},...{}};

example : int a[2][2] = {{1,2},{3,4}};
int b[2][4] = {{1,2,3,4},{5,6,7,8}};
std::string s[2][2] = {{"ab","bc"},{"de","fg"}};

a[2][2] → 

→
```
#include<iostream>
using namespace std;
int main()
{
    std::string s[2][2]={{"ab","bc"},{"de","fg"}};
    char c[3][2]={{'a','b'},{'c','d'},{'e','f'}};
    int a[2][3]={{1,2,3},{4,5,6}};
    int SizeRow_a = sizeof(a)/sizeof(a[0]); // number of Rows
    int SizeColumn_a = sizeof(a[0])/sizeof(a[0][0]); // number of column
    cout<<"Dimension of Array a - "<<SizeRow_a<<"X"<<
                                    SizeColumn_a<<"\n";
    for(int i=0;i<SizeRow_a;i++)
    {
        for(int j=0;j<SizeColumn_a;j++)
        {
            cout<<a[i][j]<<" ";
        }cout<<"\n";
    }cout<<"\n";
    int ArrayLenOf_s = sizeof(s[0])/sizeof(string);
    cout<<"Length of Array s - "<<ArrayLenOf_s<<"\n";
    for(int i=0; i< sizeof(c)/sizeof(c[0]);i++) // rows
    {
        for(int j=0;j< sizeof(c[0]);j++) // columns
        {
            cout<<c[i][j]<<" ";
        }cout<<"\n"; } return 0;
}
```

11/3/23

10. String & String operators

String :- A string in C++ is an object that represents sequence of characters.

↳ A string in C is represented by an array of characters.

↳ C++ supports C-style string as well.

↳ C++ is based on OOPs concept. It enables to represent a string as an object of the string class as std::string. This class allows you to declare a string variable quickly and store any sequence of characters in it.

↳ String is actually an 1D array of characters which is terminated by a null character '\0'. A null terminated string containg the characters that comprise the following by a null.

Syntax :- `std::string var_name = "<value>";`

`char var_name[] = "<value>";`

example :-

```
#include<iostream>
int main()
{
    char c1='\0'; // null character can be represented as '' or '\0'.
    std::string s1="Dheeraj";
    char s2[]="Chandan";
    std::cout<<c1<<"\n"<<s1<<"\n"<<s2<<
                                    std<<endl;
    return 0;
}
```

↳ String class stores the characters as a collection of bytes in continguous memory locations.

ASCII code

ASCII :- ASCII is also known as "American Standard Code for Information Interchange"

↳ A char variable in C++ is a 1 byte memory location where a single character value can be stored.

↳ One 1 byte can hold values between 0 to 255 which means there are upto 256 different characters in the ASCII character set.

↳ ASCII value of lowercase alphabet (a-z) is 97-122.
ASCII value of Uppercase alphabet (A-Z) is 65-90.

↳ // Program in C++ to print all ASII values.

```
#include<iostream>
using namespace std;
int main()
{
    for(int i=0; i<=255; i++)
    {
        cout<<"The ASCII value of "<<(char)i<<"="<<i<<endl;
    }
    return 0;
}
```

↳ ASCII value of numbers (0-9) is 48-57.

```
#include<iostream>
Using namespace std;
int main()
{
    char ch1 = 'a';
    char ch2 = 'A';
    string name = "Dheeraj";
    char ASCII_range[] = {'A','Z','a','z','0','1'};
    cout<<"ASCII value of "<<ch1<<"="<< int(ch1)<<endl;
    cout<<"ASCII value of "<<ch2<<"="<<int(ch2)<<endl;
    cout<<"Length of String \""<<name<<"\"="<<
                                    name.length()<<endl;
    cout<<"Sizeof char \"ASCII range\" Array ="<<
                                    sizeof(ASCII_range)<<endl;
    for(int i=0; i<name.length(); i++)
    {
        cout<<"ASCII value of "<<name[i]<<"="<<
                                    int(name[i])<<endl;
    }
    return 0;
}
```

```
#include<iostream>
#include<array>
using namespace std;
int main()
{
    int c=0;
    int arr[]={1,2,3,4,5}
    for(auto i:arr)
    {
        c++;
    }
    cout<<"\n The length of given array is :"<<c;
    cout<<"\n The length of array is :"<<end(arr)-begin(arr);
    int a1 = sizeof(arr)/sizeof(arr[0]);
    cout<<"\n The length of the array is :"<<a1;
    array<int,5> arr3={1,2,3,4,5};
    cout<<"\n The length of given Array"<<arr3.size();
    return 0;
}
```

↳ For taking an input we have use 'std :: cin'

Syntax : std :: cin >> Var_name;

ex :
```
std:: string s1;
std :: cin >> s1; // Dheeraj
std :: cout << s1 << std :: endl
```

↳ In string if we take as input for string having space then it will print 1st letter of that string or all characters before 1st space.

```
std:: string s1 = "Dheeraj Chandan";
Std:: cout << s1 << std:: endl; // Dheeraj Chandan

Std :: string s2;
std :: cin >> s2; // Dheeraj Chandan
std :: cout << s2 << std:: endl; // Dheeraj
```

↳
```
#include <iostream>
using namespace std;
int main()
{
    string s1;s5;
    string s2 = "Dheeraj";
    string s3 = "chandan";
    cout << "Concatinated string : " << s2+s3 << endl;
    cout << "Input a string having no space : ";
    cin >> s1;
    cout << s1 << "\n";
    string name = s1 + s2 + s3;
    cout << "Full name is : " << name << endl;
    string s4 = "Dheeraj Chandan";
    cout << "String with space in a variable : " << s4 << endl;
    cout << "Input a string having space : ";
    cin >> s5;
    cout << "\n" << s5 << endl;   return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    string s1, s3;
    string s2 = "Dheeraj from srikakulam";
    cout << "String with space in a variable : \n" << s2
                                                  << endl;
    cout << "Input a string having space :";
    cin >> s1;
    cout << "\n" << s1 << endl;
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    string s3;
    cout << "Enter a string with space as input \n" << endl;
    getline(cin, s3);
    cout << s3 << endl;
}
```

```
#include<iostream>
using namespace std;
int main()
{
    string s1="B.Dheeraj Chandan";
    string s2="Srikakulam";
    string s3 = s1.append(" from ");
    string s4 = s3 + s2;
    cout<<"Total string after contatenation :"<<s4<<"\n"<<endl;
    int StringLen = s4.length();
    cout<<"Total length of a string \""<< s4 <<"\":"<<
                                    StringLen <<"\n"<<endl;
    cout<<"First charater of a string"<<s4<<" : "<<
                                    s4.front()<<endl;
    cout<<"Last character of a string "<<s4<<" : "<<
                                    s4.back()<<endl;

    int StartPos, EndPos;
    cin>>StartPos;
    cin>>EndPos;
    cout<<"substring position index ("<<startPos<<","<<EndPos<<")"<<
                    " : "<<s4.substr(StartPos,EndPos)<<endl;
    for(int i=0;i<StringLen;i++)
    {
        //cout<<"Character at position index "<<i<<" : "<<s4[i]<<endl;
        //As s4 is a string which is a collection of characters &
        //hence we can extract character by indexing.
        cout<<"Character at position index "<<i<<" : "<<s4.at(i)<<endl;
        //character from string_variable can be extracted by using
        //a function .at()
        // syntax - stringname.at(position)
        // example - s4.at(1) where s4="Dheeraj"
    }
return 0; }
```

```
#include<iostream>
Using namespace std;
int main()
{
    string s="B.Dheeraj Chandan";
    cout<<s.substr(0,5)<<endl;
    cout<<s.substr(3,5)<<endl;
    cout<<s.substr(5,5)<<endl;
    String str[3]={"Balivada","Dheeraj","Chandan"};
    cout<<str[1]<<endl;
    cout<<str[1][0]<<endl;
    return 0;
}
```

```
#include<iostream>
Using namespace std;
int main()
{
    string s1="Dheeraj from Srikakulam";
    String s2;
    cout<<"Enter the string:";
    getline(cin,s2);
    cout<<s2<<endl;
    cout<<"String \""<<s2<<"\" in \""<<s1<<"\":";
    if(s1.find(s2)!= string::npos)
    {
        cout<<"Found at position "<<s1.find(s2)<<endl;
    }else{
        cout<<"Not found"<<endl;
    }
    return 0;
}
```

String functions in C++ :

↳ .find() - It returns the index of the first occurance of specified string or character.

.at() - It is designed to access a particular character residing in a string. To extract a character from a string.

.substr() - A method used to manipulate text to store a specific part of a string. To extract a string from within a string.
Extracts character from starting index to next n^{th} + startingindex position.

.length() - String length represents the number of bytes used to encode the given string. To find length of a string. It can be found out using the length(), size(), strlen().

.front() - It is used to reference/extract a first character of a string.

.back() - It is used to reference/extract a last character of a string.

.append() - It is used to concatinate 2 strings. Used to combine/join/add 2 strings. Used to add strings together.

strcpy() - It copies a string into other string.
strcpy(S1,S2) : copies string S2 into S1.

strcat() — It is used to concatinate two strings.
strcat(s1, s2) : concatenates string s2 onto the end of string s1.

strlen() — It returns length of a string.

strcmp() — if strcmp(s1, s2) is considered then it returns, =0 if s1 & s2 are same
<0 if s1 is less than s2 s1<s2
>0 if s1 is greater than s2 s1>s2

strchr() — It returns a pointer to the first occurance of character(ch) in a string(s1). strchr(s1, ch)

strstr() — It returns a pointer to the first occurance of a string in a string. strstr(s1, s2)

push_back() — This function is used to input a character at the end of a string.

pop_back() — This function is used to delete the last character from the string.

capacity() — It returns the capacity allocated to the string, which can be equal to or more than the size of the string. Additional space is allocated so that when the new characters are added to the string so that operations can be done efficiently.

resize() — Function changes the size of the string, the size can be increased or decreased.

begin() — This function returns an iterator to the beginning of the string.

end() — This function returns an iterator to the next to the end of the string.

rbegin() — It returns a reverse iterator pointing at the end of the string.

rend() — It returns a reverse iterator pointing to the previous of beginning of the string.

cbegin() — It returns a constant iterator pointing to beginning of the string, it cannot be used to modify the content it points to.

cend() — It returns a constant iterator pointing to the next of end of the string, it cannot be used to modify the contents it point to.

crbegin() — It returns a constant reverse iterator pointing to the end of the string, it cannot be used to modify the contents it points to.

crend() — It returns a constant reverse iterator pointing to the previous of beginning of string, it cannot be used to modify the contents it points to.

.copy() — It copies the content of a string to other string.

.swap() — It is used to exchange the contents/values of two string objects.

.reverse() — It is used to reverse a string

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
    string s1="B.Dheeraj Chandan";
    int a[]={1,2,3,4,5};
    cout<<s1.find("eer")<<endl;          // .find()
    cout<<s1.find("a")<<endl;
    cout<<s1.find("ae")<<endl;
    cout<<s1.at(3)<<endl;                // .at()
    cout<<s1.at(16)<<endl;
    cout<<s1.substr(0)<<endl;            // .substr()
    cout<<s1.substr(4,5)<<endl;
    cout<<s1.substr(4,2)<<endl;
    cout<<s1.substr(3,3)<<endl;
    cout<<s1.substr(7)<<endl;
    cout<<endl;
    for(int i=0;i<s1.length();i++)
    {
        cout<<"Index :"<<i<<"; Character :"<<s1.at(i)<<endl;
    }
    cout<<s1.length()<<endl;             // .length()
    cout<<s1.size()<<endl;               // .size()
    cout<<"Size of String :"<<sizeof(s1)<<"\nSize of Character :"
        <<sizeof(char)<<endl;
    cout<<sizeof(s1)/sizeof(s1[0])<<endl;
    cout<<"Size of Array:"<<sizeof(a)<<"\nSize of Integer element:"
        <<sizeof(a[0])<<endl;
    cout<<sizeof(a)/sizeof(a[0])<<endl;
    cout<<s1.front()<<endl;              // .front()
    cout<<s1.back()<<endl;               // .back()
    cout<<s1.append(" from sklm.");      // .append()
    string s2="Drj";
    cout<<"Before swap : s1="<<s1<<"; s2="<<s2<<endl;
    s1.swap(s2)
    cout<<"After swap : s1="<<s1<<"; s2="<<s2<<endl;
    return 0;
}
```

16/4/2023

↳

```
#include <iostream>
// #include<string>
#include <string>
#include <algorithm>
using namespace std;
int main()
{
    char s1[] = "B Dheeraj Chandan";
    char s2[] = "";
    string str = "Dheeraj Chandan";
    cout << "Before copying a value of s2= " << s2 << endl;
```

20/05/23

11. OOPs

Enums, Class, objects & Constructors

Enum or Enumeration :-

↳ Enumeration is an user defined data type that can be assigned some limited values. It is a datatype consisting of named values like elements, members etc.. that represents integral constants. It enables for a variable to be a set of predefined constants.

↳ Syntax :

```
enum enumerated_Type_Name
{
    value1,
    Value2,
    :
    Value N
};
```

↳ The enum keyword is used to declare enumerated types after that enumerated type name was written then under curly brackets possible values are defined.

↳ Enumerates can be created in two type :

(i) It can be declared during declaring enumerated types, just add the name of the variable before semicolon.

(ii) Create enumerated type variable as same as normal variables.

```
#include <iostream>
using namespace std;
enum a
{
    a1 = 10,
    a2 = 20,
    a3 = 30
};
enum bc
{
    b,
    c
};
enum de
{
    d,
    e
};
a enum_a = a2;
bc enum_b = b; // bc enum_b;
// enum b = b;
bc enum_c = c; // bc enum_c;
// enum_c = c;
int main()
{
    de enum_d;
    enum_d = d;
    de enum_e;
    enum_e = e;
    cout << enum_a << endl;
    cout << enum_b << endl;
    cout << enum_c << endl;
    cout << enum_d << endl;
    cout << enum_e << endl;
    return 0;
}
```

Structures : → It is an user defined data type in C/C++.

↳ A structure creates a data type that can be used to group items of possibly different types into a single type.

↳ A collection of variables of different data types.

↳ 'struct' keyword is used to create a structure.

Syntax :

```
struct structure_Name
{
    member1;
    member2;
    member3;
    .
    .
    .
    member N;
};
```

↳ Structures in C++ can contain two types of members :

(i) Data Member — These members are normal C++ variable. We can create a structure with variable of different data types in C++.

(ii) Member Functions — These members are normal C++ functions. We can add functions inside a structure declaration.

↳

```
#include <iostream>
Using namespace std;
struct Point_1
{
    int x=10, y;
} P1;
struct Point_2
{
    int x=30, y=20;
};
int main()
{
    struct Point_2 P2;
    // struct Point_1 P1;  // Already defined
    P1.y = 5;
    cout << "sum of x&y in struct Point_1 =" << P1.x+P1.y << endl;
    cout << "Sumof x&y in struct Point_2 =" << P2.x+P2.y << endl;
    return 0;
}
```

↳ ~~using namespace~~

```
#include <iostream>
Using namespace std;
struct student
{
    string name;
    int roll;
    float marks;
    char grade;
};
struct employee
{
    char name[50]="Dheeraj";
    int empid = 12345;
    int salary = 500000;
};
struct variable
{
    int year, month;
};
int main()
{
    // struct student s;
    // struct employee e;
    //struct variable dym={2023,5};
    std::getline(std::cin, s.name);
    cin >> s.roll ;
    cin >> s.marks ;
    cin >> s.grade ;
    cout<< "Student Information :"<< endl;
    cout<< "\tName ->"<< s.name << "\n";
    cout<< "\tRoll ->"<<s.roll << "\n";
    cout<< "\tMarks ->"<< s.marks << "\n";
    cout<< "\tGrade ->"<< s.grade << "\n";
```

```
cout << "Employee Information :" << endl;
cout << "\t Name -> " << e.name << endl;
cout << "\t ID -> " << e.empid << endl;
cout << "\t Salary -> " << e.salary << endl;
cout << "Date Information :" << endl;
cout << "\t Year -> " << dym.year << "\n";
cout << "\t Month -> " << dym.month << endl;
return 0;
}
```

Class & objects :

↳ class in C++ is the building block that lead to object-Oriented Programming. It is an user-defined data type, which holds its own data members functions, which can be accessed and used by creating an instance of that class.

↳ C++ class is a blue print ofor an object.

↳ A class is a user defined data type that has data members and member functions.

↳ Data members are the data variables and member functions are the functions used to manipulate these variables together, these data members and member functions define the properties & behaviour of the object in a class.

↳ An object is an instance of a class.

Declaring objects → When a class is defined, only specification for the object is defined, no memory or storge allocated. To use data & access functions defined in class we need to create objects.

Syntax : class_Name Object_Name;

↳ Accessing data members depends solely on the access control of that data members. This access control given by Access modifiers in C++ .ie. public, private & protected

```
#include <iostream>
using namespace std;
class student
{
    public:
        string name;
        int roll;
        float marks;
        char grade;
};
int main()
{
    student s;
    std::getline(std::cin, s.name);
    cin >> s.roll;
    cin >> s.marks;
    cin >> s.grade;
    cout << "Student Information :" << endl;
    cout << "\tName -> " << s.name << "\n";
    cout << "\tRoll -> " << s.roll << "\n";
    cout << "\tmarks -> " << s.marks << "\n";
    cout << "\tGrade -> " << s.grade << endl;
    return 0;
}
```

```
#include <iostream>
using namespace std;
class variable
{
    public:
        int year;
        int month;
        string Sdate(){
            return "\tYear::Month ->";
        }
        int IYear(){
            return year;
        }
        int IMonth(){
            return month;
        }
};
```

```
int main()
{
    Variable dym;
    cin >> dym.year;
    cin >> dym.month;
    cout << "Date Information :" << endl;
    cout << "\t Year -> " << dym.year << "\n";
    cout << "\t Month -> " << dym.month << "\n";
    cout << dym.Sdate() << dym.IYear() << "::" << dym.IMonth();
    return 0;
}
```

Constructors :- It is a special method that is invoked automatically at the time of object creation.

- It is used initialize the data members of new objects.
- Constructors in C++ has same name as class or structure.
- It is a member function of class, whose name is same as class name.
- It is a special special type of member function used to initialize data members for an object of a class.
- It is invoked at the time of object creation.
- It donot return value, they do not have a return type.

21/05/2023

12. Class Functions, Function Overloading Inheritance

OOPs in C++ :-

↳ It is about creating objects that contains both data & functions.

↳ It is a software design approach that focuses on breaking large programs into smaller, more manageable components called objects.

↳ It allows for easier mantainance & improved program organisation

↳ It enables application to be more flexible & extinsible.

↳ It aims to implement real world entities like inheritance, hiding, polymorphism etc. in programing

↳ These are some basic concepts that act as building blocks of OOPs i.e.

(i) Class
(ii) objects
(iii) Encapsulation
(iv) Abstraction
(v) Polymorphism
(vi) Inheritance
(vii) Dynamin Binding
(viii) Message Passing

(i) Class – Building blocks of C++ that leads to Object Oriented Programing is a class.
- It holds its own data members & member functions which can be accessed used by creating an instance of that class
- It is a user defined data type that has data members & member functions.

(ii) Objects – It is an identificiable entity with some characters & behaviour.
- It is an instance of a class.
- When class is defined no memory is allocated but when object is created then memory is allocated.

21/05/2023

12. Class Functions, Function Overloading Inheritance

OOPs in C++ :-

↳ It is about creating objects that contains both data & functions.
↳ It is a software design approach that focuses on breaking large programs into smaller, more manageable components called objects.
↳ It allows for easier mantainance & improved program organisation
↳ It enables application to be more flexible & extinsible.
↳ It aims to implement real world entities like inheritance, hiding, polymorphism etc. in programing
↳ These are some basic concepts that act as building blocks of OOPs i.e.

(i) Class
(ii) Objects
(iii) Encapsulation
(iv) Abstraction
(v) Polymorphism
(vi) Inheritance
(vii) Dynamin Binding
(viii) Message Passing

(i) Class – Building blocks of C++ that leads to Object Oriented Programing is a class.
- It holds its own data members & member functions which can be accessed used by creating an instance of that class
- It is a user defined data type that has data members & member functions.

(ii) Objects – It is an identificiable entity with some characters & behaviour.
- It is an instance of a class.
- When class is defined no memory is allocated but when object is created then memory is allocated.

(iii) Encapsulation – It is defined as wrapping up of a data & information under a single unit.
- In OOPs, it is defined as binding together the data and the functions that manipulate them.
- Encapsulation also leads to data abstraction or data hiding. Using encapsulation also hides the data.

(iv) Abstraction – It means displaying only essential information & hiding the details.
- It refers to providing only essential information about the data to the outside world, hiding the background details or implementation.
- Using classes : ⊙ Class help us to group data member & member function using available access specifiers.
  ⊙ Class can decide which data member will be visible to the outside world & which is not.
- ~~Using~~ In header files : ⊙ Abstraction in C++ can be header files.

(V) Polymorphism – It allows us to reuse code by creating one function that's usable for multiple users.
- Allows a specific routine to use variable of different types at different times.
- The behaviour depends upon types of data used in the operation. C++ supports operator overloading & Function overloading.
- Operator overloading : The process of making an operator exhibit different behaviours in different instances.
  Function overloading : Function overloading is using a single function name to perform different type of tasks. It is extensively used in implementing inheritance.

(vi) Inheritance — The capability of a class to derive prop & characteristics from another class.
- Sub class : It inherits the properties from another class is called sub class or Derived cla
Super class : Class whose properties are inherited by a sub class is called Base class or Super class.
Reusability : Inheritance supports concept of 'reusabi When we want to create a new clas & there is already a class that includes some of the codes that we want, we can derive our new class from the existing class.

(vii) Dynamic Binding — The code to be executed in response to the function call is decided at run time.

(viii) Message Passing — Objects communicate with one another by sending & recieving information.
- The message for an object is a request for the execution of a procedure & therefore will invokes a function in the recieving object that generates the desired results.

Class :- ↳ A fundamental block of a program that has its own set of methods & variables.
↳ We can create an object or the instances of the Class to access these methods or variables.

Access specifiers :- It decides the accessibility of the clas members like variables or methods in other classes.
↳ It is a defining code element that can be determine wh elements of a program are allowed to access a specifi Variable or othe piece of data.

↳ There are 3 types of access specifiers :-
(i) Public
(ii) Private
(iii) Protected

(i) Public – All the class members declared under these public specifier will be available to everyone.
- Data member & member function declared as public can be accessed by other classes and functions.

(ii) Private – The class members declared as private can be accessed by only by the member functions inside the class.
- They are not allowed to be accessed directly by any object or function outside the class.

(iii) Protected – The protected access specifier is similar to the private access modifier in the sense that it can't be accessed outside of its class unless with the help of friend class.
- It allows access within same class & don't allows within derived class or outside class.

Member & Methods :-
↳ Members are the variables in the class
↳ Methods are the function in the class

Constructor :- It is a special type of a member fuction which is called automatically when an object is created.
↳ It has same name as class but does'nt return anything.
↳ A constructor with no parameters known as default constructor.
↳ A constructor with parameters known as parameterised constructor.

↳ It is a special method that is invoked automatically at the time of object creation.
↳ It is used to initialize the data members of new objects.
↳ It is invoked at the time of object creation.
↳ Constructor do not return value, hence they do not have a return type.

↳ A constructor with no parameters known as default constructor.
↳ A constructor with parameters is known as a parameterized constructor.

Function Overloading :- It provides multiple definitions of the function by changing signature. i.e. changing number of parameters, changing datatype of parameters, return type doesn't play any role.

Function overriding :- It is redefinition of a base class function in its derived class with same signature. i.e return type & parameters.

↳ It is a special method that is invoked automatically at the time of object creation.
↳ It is used to initialize the data members of new objects.
↳ It is invoked at the time of object creation.
↳ Constructor do not return value, hence they do not have a return type.

↳ A constructor with no parameters known as default constructor.
↳ A constructor with parameters is known as a parameterized constructor.

Function Overloading :- It provides multiple definitions of the function by changing signature. i.e. changing number of parameters, changing datatype of parameters, return type does'nt play any role.

Function overriding :- It is redefinition of a base class function in its derived class with same signature. i.e return type & parameters.

```
#include <iostream>
using namespace std;
class A
{
    public:
    int a1 = 10
    string ch1;
    public:
    int a2=20;
    A(string s)
    {
        cout << "Constructor "+s << endl;
    }
};
class B
{
    private:
    int b1=10;
    char ch2[10]= "Dheeraj";
    public:
    int b2;
};
int main()
{
    A obj1("Balivada");
    B obj2;
    getline(cin, obj1.ch1);
    cin >> obj2.b2;
    cout << obj1.ch1 << endl;
    cout << obj1.a1 << endl;
    cout << obj1.a2 << endl;
    //cout << obj2.ch2 << endl;
    //cout << obj2.b1 << endl;
    cout << obj2.b2 << endl;
    return 0;
}
```

```
#include<iostream>
Using namespace std;
class parent
{
    public:
    int p1=10;
    int p2(int n)
    {
        return n;
    }
};
class child: public parent
{
    public:
    int c1=30;
};
int main()
{
    parent p;
    child c;
    cout << p.p1 << endl;
    cout << p.p2(20)<< endl;
    cout << c.c1 << endl;
    cout << c.p1 << endl;
    cout << c.p2(40) << endl;
    //cout << p.c1 << endl;
    return 0;
}
```

```
#include <iostream>
using namespace std;
class A
{
    public:
    string a()
    {
        return "Hi I'm Dheeraj !!";
    }
    int b(int b1, int b2)
    {
        return b1+b2;
    }
    float c(float c1)
    {
        return c1;
    }
};
int main()
{
    A obj1;
    cout<<obj1.a()<<endl;
    cout<<obj1.b(10,20)<<endl;
    float n;
    cin>>n;
    cout<<obj1.c(n)<<endl;
    return 0;
}
```

```
#include<iostream>
using namespace std;
class A
{
    public:
    string a()
    {
        return "Hi, I'm Dheeraj";
    }
    int a(int a1, int b1)
    {
        return a1+b1;
    }
    int a(int a1)
    {
        return a1 * 10;
    }
    float a(float c1)
    {
        return c1;
    }
};
int main()
{
    A obj1;
    cout << obj1.a() << endl;
    cout << obj1.a(10,20) << endl;
    cout << obj1.a(100) << endl;
    float n;
    cin >> n;
    cout << obj1.a(n) << endl;
    return 0;
}
```

27/5/23

13. Pointers in C++

Pointers :- Pointers are symbolic representation of addresses

↳ Assigning the address of a variable to a pointer using the unary operator (&) which returns the address of that variable.

↳ Accessing the value stored in the address using unary operator (*) which returns the value of the variable located at the specified by its operand.

↳ Main purposes of pointers :

(i) To allocate new objects on the heap.
(ii) To pass functions to other functions.
(iii) To iterate over elements in arrays or other data structures.

↳
```
#include<iostream>
using namespace std;
int main()
{
    int i=10;                  // Variable value
    int* ptr1 = &i;            // saving address to a pointer
    int* ptr2 = ptr1;          // Pointing to an address
    int j = *ptr1;             // accessing a variable
    int** ptr3 = &ptr1;        // Saving address of a pointer.
    cout<<i<<",\n";
    cout<<ptr1<<",\n";
    cout<<ptr2<<",\n";
    cout<<j<<",\n";
    cout<<ptr3<<",\n";
    return 0;
}
```

↳
```
#include<iostream>
using namespace std;
int main()
{
    int a[3]={1,20,30};
    int* ptr1=a;
    int b=20;
    int* ptr2=&b;
    cout<<ptr1<<endl;
    cout<<ptr2<<endl;
    cout<<ptr1<<endl;
    cout<<ptr1+1<<endl;
    cout<<ptr1+2<<endl;
    cout<<*ptr1<<endl;
    cout<<*ptr1+1<<endl;
    cout<<*(ptr1+1)<<endl;
    cout<<*ptr1+2<<endl;
    cout<<*(ptr1+2)<<endl;
    cin>>a[0];
    cout<<*ptr1<<endl;
}
```

~~Call by value~~ :-

↳ Functions can be invoked in two ways :-

(i) call by value
(ii) cdl by reference

↳ Parameters passed to the function called actual parameters
↳ Parameters recieved by the function called formal parameters.

The value of actual parameters are copied to the functi formal parameters.

(i) call by value :- Method of parameter passing, the actual parameters are copied to the function's formal parameters.

(ii) call by reference :- Method of parameter passing, the address of actual parameter is passed to the function as the formal parameters.

Memory management :-

↳ It is a process of managing computer memory, assigning the memory space to the programs to improve the overall system performance.

↳ static - Allocated at startup & exists for the entire life of program.

Automatic - Allocated upon block entry exists upon the block

Dynamic - Allocated when we allocate & deleted when we delete or when garbage collector removes it.

↳ #include<iostream>
using nam

```
#include <iostream>
Using namespace std;
void a(int a1, int a2);
int main()
{
    int x=10, y=20;                    // call by value.
    a(x,y);
    cout << x << " , " << y << endl;
    return 0;
}
Void a (int a1, int a2)
{
    cout << a1 * a2 << endl;
}
```

```
#include <iostream>
using namespace std;
Void a (int* a1, int* a2);
int main()
{
    int x=10, y=20;
    a(&x,&y);                          // call by reference
    cout << x << " , " << y << endl;
    return 0;
}
Void a (int *a1, int *a2)
{
    cout << a1 << " , " << a2 << endl;
    cout << *a1 << " , " << *a2 << endl;
}
```

28/05/2023 14. File Handling

File Handling :- ↳ It helps in creating & reading file data programatically.
↳ It is useful in storing data persistently
↳ Uses file streams to do so.
↳ Importance : It offers mechanism through which you can collect the output of a program in a file and then perform multiple operations on it.

↳ For achieving file handling we need to follow steps :-
(i) Naming a file
(ii) Opening a file
(iii) Writing data into the file
(iv) Reading data from the file
(v) Closing a file

↳ There are few file types used in C++ :
ex :- dat, txt, json, yaml, csv
↳ Files are not good for storing data.
↳ Databases like mySQL, SQLite, MongoDB should be used.
↳ Useful for storing configuration locally.

File streams :- The sequence of bytes given as input to the executing program and the sequence of bytes that comes as output from the executing program are called as stream.
↳ Streams are nothing but the flowe of data in a sequence.

↳ File access streams :- The I/O system of C++ contains a set of classes which defines the file handling methods.
↳ These includes ifstream, ofstream and fstream classess.

↳ ifstream - This class provides input operation.
- It contains open() function with default input mode.
- Inherits the functions get(), getline(), read(), seekg() & tellg() functions from the isstream.

ofstream - This class provides output operation.
- It contains open() function with default output mode.
- It inherits the functions put(), write(), seekp() and tellp() functions from the ostream.

fstream - This class provides support for simultaneous input & output operations.
- It inherits all the functions from the istream & ostream classes through iostream.

↳ Ofstream - creates & writes
Ifstream - read from files
Fstream - A combination of both Ofstream & Ifstream

↳ All above three classes are derived from fstreambase and from the corresponding iostream class and they are designed specifically to manage disk files.

↳ C++ provides us with operations in file handling :

(i) Creating a file : open()
(ii) Reading data : read()
(iii) Writing new data : write()
(iv) closing a file : close()

(i) Opening a file - To open a file

↳ syntax :

```
void open(const char* file_name, ios::openMode mode);
```

↳ example:

```
fstream new_file;
new_file.open("newfile.txt", ios::out);
```

(ii) Reading data - To Read the data from file.

↳ example :

```
new_file.open("new_file_write.txt", ios::in);
```

(iii) Writing data - To write the data to file.

↳ example :

```
new_file.open("new_file_write.txt", ios::out);
```

(iv) Closing a file - To close a file.

↳ example :

```
new_file.open("new_file.txt", ios::out);
new_file.close();
//return 0;
```

↳ .isopen() is a function used to check if file is opened or not.

↳ In the syntax,

void open(const char* file_name, ios::openmode mode);

Here, first argument of open() function defines the name and format of the file with the address of the line.

Second argument represents the mode in which file has to opened.

↳ Modes & descriptions of the following :-

in — Opens the file to read (default for ifstream)
out — Opens the file to write (default for ofstream)
binary — Opens the file in binary mode.
app — Opens the file and appends all the outputs at the end.
ate — Opens the file and moves the control to the end of the file.
trunc — Removes the data in the existing file
nocreate — Opens the file only if it already exists
noreplace — Opens the file only if it does not already exist.

example :-

```
fstream new_file;
new_file.open("newfile.txt", ios::out);
```

08/06/2023

Seekg() → It is a function in the iostream library allows to seek an arbitary position in a file

→ It is included in fstream header file.

→ It is used in file handling to sets the position of the next character to be extracted from the input stream from a given file.

↳
```
#include<iostream>
#include<fstream> // it include ifstream & ofstream
using namespace std;
int main()
{
    cout<<"Hi \n How are you"<<endl;
    string name;
    int roll;
    cout<<"Enter the name : ";
    cin>>name;
    cout<<"Enter the Roll :";
    cin>>roll;
    cout<<endl;
    // system("clear");
    cout<<"Name :"<<name<<endl<<"Roll :"<<roll<<endl;
    ofstream myfile1
```

```
#include<iostream>      // it includes ifstream & ofstream
#include<fstream>
using namespace std;
int main()
{
    cout << "Hi \n How are you" << endl;
    string name;
    int roll;
    cout << "Enter the name : ";
    cin >> name;
    cout >> "Enter the roll : ";
    cin >> roll;
    cout << endl;
    // system("clear");
    cout >> "Name : " << name << endl << "Roll : " << roll << endl;
    ofstream myfile1;
    myfile1.open("C:\\ ..... \\ CSimple.txt");
    myfile1 << "Name : " << name << endl << "\n Roll : " << roll << endl;
    myfile1.close();
    string data;
    ifstream myfile2;
    myfile2.open("C:\\ ..... \\ CSimple.txt");
    myfile2 >> data;
    cout << "\n Text file read : " << data << endl;
    if(myfile2.is_open())
    {
        cout << "File is open so text in file : \n";
        myfile2.seekg(0);
        while (getline (myfile2, data))
            cout << data;
        cout << endl;
        cout << "closing the file" << endl;
        Myfile2.close();
    }else{
        cout << "file closed !!" << endl;
    } return 0;
}
```

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
    ofstream myfile1;
    myfile1.open("C:\\___\\FileDataAppend.txt");
    myfile1 << "Name : Dheeraj \n Roll : 21" << endl;
    myfile1.close();

    string data;
    ifstream myfile2;
    myfile1.open("C:\\___\\FileDataAppend.txt", ios::app);
    myfile1 << "Im fine" << endl;
    myfile1.close();
    myfile2.open("C:\\___\\FileDataAppend.txt");
    myfile2 >> data;
    myfile2.seekg(0);
    cout << "Text in file : " << data << endl;
    if(myfile2.is_open())
    {
        cout << "File is open so full text in file : \n";
        myfile2.seekg(0);
        while(getline(myfile2, data))
            cout << data;
        cout << endl;
        cout << "closing the file" << endl;
        myfile2.close();
    }else{
      cout << "File is closed" << endl;
    }
    return 0;
}
```

```
#include<iostream>
#include<fstream>
void ReadData();
using namespace std;
int main()
{
    fstream myfile1("c:\\...\\FuncFileHandling.txt", ios::app);
    string name; int age, marks;
    cout<<"Enter your 1st name : ";
    cin>>name;
    cout<<"Enter your age : ";
    cin>>age;
    cout<<"Enter your marks : ";
    cin>>marks;
    myfile1<<"Name -"<<name<<"\nAge -"<<age<<
        <<"\nMarks -"<<marks<<endl;
    myfile1.close();
    //system("clear");
    ReadData();
    return 0;
}
void ReadData()
{
    ifstream myfile2;
    myfile2.open("c:\\...\\FuncFileHandling.txt");
    string name, data; int age, marks;
    //cout<<"NAME"<<"\t"<<"AGE"<<"\t"<<"MARKS"<<endl;
    /*while((myfile2>>name>>age>>marks))
    {
        cout<<name<<"\t"<<age<<"\t"<<marks<<endl;
    }*/
    while(getline(myfile2, data))
    {
        cout<<data<<endl;
    }
}
```

11/06/2023 15. Exceptions

Exceptions in C++ :-

↳ An exception is a problem that arises during the execution of a program.

↳ It is a response to an exceptional circumstances that arises while a program is running such as dividing by Zero.

↳ It provides a way to transfer control from one part of a program to another.

↳ C++ Exceptional handling is built upon three keywords : (i) try
(ii) catch
(iii) throw

(i) try : A try block identifies a block of code for which particular exceptions will be activated. It allows you to define block of code to be tested for errors while it is being executed.

(ii) catch : A program catches an exception with an exception handler at the place in a program where you want to handle the problem.
It allows you to define a block of code to be executed, if an error occurs in the try block.

(iii) throw : A program throws an exception when a problem shows up. This is done using a throw keyword.
It throws an exception when a problem is detected, which tells us create a custom error.

```
#include <iostream>
using namespace std;
int main()
{
    int age;
    cout << "Enter the age for access : ";
    try
    {
        cin >> age;
        cout << age << endl;
        if(age >= 18)
        {
            cout << "Access Available!! " << endl;
        }else{
            throw 404;
        }
    }
    catch(int ErrCode)
    {
        cout << "Access Denied !! : " << ErrCode << endl;
    }
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    int num1 = 20;
    int num2;
    cout << "Input for a number : ";
    try
    {
        cin >> num2;
        cout << num2 << endl;
        if (num2 != 0 && num1 >= num2)
        {
            cout << "Difference Calculation : " << (num1 - num2) << endl;
        }else{
            throw 1001;
        }
    }
    catch (int ErrCode)
    {
        cout << "Input error : " << ErrCode << endl;
    }
    return 0;
}
```