Rainbow
PAGE:
DATE: / /

OOP UNIT-III

(1) What is runtime polymorphism ? How it is achieved in cpp. Explain with example.

→ **Runtime polymorphism :-**

- Runtime polymorphism, also known as dynamic polymorphism or late binding is a feature of OOP in which the function call is resolved at run time.
- Runtime polymorphism can be achieved by function overriding or method overriding.
- It is achieved through virtual function & pointer or reference also

Example :

```
#include<iostream>
using namespace std;
class Base
{
    public:
    void draw()
    {
        cout<<"draw";
    }
};
class Derived : public Base
{
    public:
    void draw()
    {
        cout<<"Derived";
    }
};
```

```
int main()
{
    Base b;
    Derived d;
    b.draw();
    d.draw();
}
```

Output:

Draw
Derived

(2) Explain polymorphism and type of polymorphism.

→ Polymorphism:

- It is a ability to take ~~more~~ more than one form. In polymorphism ~~means~~, poly- means many and morphism means forms.
- The primary goal of polymorphism is an ability of object of different types to respond to methods and data values by using the same name.
- There are two main types of polymorphism: compile-time polymorphism and runtime polymorphism.

Rainbow
PAGE:
DATE: / /

(i) compile time (static) Polymorphism:-
- Also known as method overloading or function overloading.
- In this function calls are resolved at compile time.
- Involves multiple functions or methods with the same name but different parameters types or a different no. of parameters.
- The compiler determines which function or method to call based on the number and types of arguments in function call.

example:

```
#include<iostream>
using namespace std;
class mathop
{
    public:
        int add(int a, int b)
        {
            return a+b;
        }

        double add(double a, double b)
        {
            return a+b;
        }
};
```

```
int main()
{
    Mathop m;
    cout<< m.add(5,10);
    cout<< m.add(3.5,2.7);
    return 0;
}
```

Output:

15
6.2

(ii) Runtime (Dynamic polymorphism):

- Also known as method overriding or late binding.
- In this function calls are resolved at run time.
- Involves a base class and one or more derived classes, with same function and name and parameters declared as virtual in base class.

Example: Question 1

③ What is polymorphism? How it is related to function overloading.

→ Polymorphism:

refer q2.

Relation to function overloading:-

(i) function overloading is a specific form of compile-time polymorphism.

(ii) It allows multiple functions with same name to coexist in the same scope, differing in the types or no. of their parameters.

(iii) This provides a way to write more versatile and reliable code, allowing functions to adapt to different input scenarios.

(iv) While function overloading is a form of polymorphism, it specially addresses compile-time decisions.

④ What is operator overloading and write down a cpp program to implement operator overloading of complex class.

→ Operator overloading:-

- It is a compile-time polymorphism, in which operator is overloaded to provide special meaning to user defined data type.
- Operator overloading is used to overload pre defined operator in C++.

Rainbow
PAGE:
DATE: / /

Program:

```
#include<iostream>
using namespace std;
class complex
{
    int real;
    int imag;

    public:
    complex()
    {
        real=0;
        imag=0;
    }

    friend istream &operator >> (istream
            &input, complex &c)
    {
        cout<<" Enter Real Part : ";
        input>> c.real;
        cout<<" Enter Imaginary part : ";
        input >> c.imag;
    }

    friend ostream &operator << ( ostream
            &output, complex &c)
    {
        output << c.real << " : " <<
            c.imag << " i ";
    }
```

```
Complex operator + (complex c)
{
    Complex temp;
    temp.real = real + c.real;
    temp.imag = imag + c.imag;
    return temp;
}

Complex operator * (complex c)
{
    complex temp;
    temp.real = (real * c.real) - (imag *
                                   c.imag)
    temp.imag = (real * c.imag) +
                (imag * c.real)
    return temp;
}
};
int main()
{
    Complex c1, c2, c3, c4;

    cout << " Default Value: ";
    cout << c1;

    cout << " Enter first Number ";
    cin >> c1;

    cout << " Enter second number ";
    cin >> c2;
```

```
C3 = C1 + C2;
cout << " Addition : ";
cout << C3;

C4 = C1 * C2;
cout << " multiplication : ";
cout << C4;
}
```

(5) What are rules for overloading operator?

→ Rules for overloading operator :-

(i) only existing operator can be overloaded
(ii) The basic meaning of operator cannot be changed.
(iii) Overloaded operator must follow the syntax of original operator.
(iv) Overloaded operator must have atleast one operand that is of user defined types.
(v) Binary arithmetic operators (+, -, * and /) must return a value.
(vi) When binary operators overloaded through a member function, the left hand operator must be an object of relevant class
(vii) Binary operators overloaded through a member function must take one explicit argument.
(viii) Binary operators overloaded through friend function takes two arguments.

⑥ Write a program to overload unary operator?

→

```
#Include<iostream>
using namespace std;
class MyNumber
{
    private:
        int number;
    public:
        MyNumber(int num)
        {
            number=num;
        }

        MyNumber operator++()
        {
            number++;
            return *this;
        }

        MyNumber operator--()
        {
            number--;
            return *this;
        }

        bool operator!()
        {
            return (number==0);
        }
```

```
    void display()
    {
        cout<<" Number :"<<number;
    }
};
int main()
{
    MyNumber num(5);
    ++num;
     num.display();

    --num;
    num.display();

    if(!num)
    {
        cout<<" Number is zero";
    }
    else
    {
        cout<<" Number is not zero";
    }
    return 0;
}
```

(7) Explain what is typecasting. Explain implicit and explicit type of conversion with example.

→ Typecasting:-

- It is a process of converting one data type into another. It allows the programmer to explicitly change the data type of a variable to suit the requirements of a particular operation or situation.

- There are two main types of type casting in C++ : implicit and explicit.

(i) Implicit typecasting :- It is also known as type of coercion or automatic type conversion, occurs automatically by the compiler when a value of one data type is used in a context where another data type is expected.

- The goal is to make the operation valid without requiring explicit casting from the programmer

- example:

```
#include <iostream>
int main()
{
    int val = 5;
    int dval = 2.5;
    double res = val + dval;
    cout << " Implicit Result : " << res;
    return 0;
}
```

(ii) Explicit conversion :-

- It envolves the manual conversion of variable from one data type to another by the programmer.
- Unlike implicit type conversion, which is done automatically by the compiler explicit type conversion requires a specific directive to inform the compiler about the intended conversion.
- example:

```
#include<iostream>
using namespace std;
int main()
{
    double dval = 2.5;
    int val = (int) dval + 1;
    cout<<" sum :"<< val;
    return 0;
}
```

sum = 3.5

(8) Explain use of explicit and mutable keyboard with example.

→ (1) Explicit Keyword:

- The explicit keyword is used to prevent implicit type conversions by the compiler.
- It is applied to constructors with single parameter to avoid unintended automatic conversions.

eg

```
#include<iostream>
using namespace std;
 class Test
 {
        int val;
        public
         explicit Test(int x) : val(x)
         {
              cout << "val : "<<val;
          }
 };
int main()
{
       Test obj = 100;
        return 0;
}
```

output:

Error : intializing : cannot convert from int to Test.

(ii) Mutable Keyword:

It is used to indicate that a particular data member of a class can be modified even in const member function.

eg:

```
#include<iostream>
using namespace std;
```

```
class Test
{
    int a;
    mutable int b;
    public:
      Test()
      {
          a=0;
          b=0;
      }
      void change() const
      {
          a=a+10;   //error
          b=b+20;   //no error
      }
};
int main()
{
    const Test obj;
    obj.change();
    return 0;
}
```

(9) Explain function overloading and function overriding in detail.

→ (i) Function overloading :-

- The concept by which we can define different function in a class with the same name but with different parameters is known as function overloading.
- It takes place during compile time, therefore it is known as compile time polymorphism.
- It happens without inheritance.

eg.

```
#include<iostream>
using namespace std;
class Cal
{
    public:
        int add(int a, int b)
        {
            return a+b;
        }
        double add(double a, double b)
        {
            return a+b;
        }
};
int main()
{
    Cal c;
    Cout<<" Sum of integer :"<<c.add(5,7);
    Cout<<" Sum of double :"<<c.add(3.5,2.7);
    return 0;
}
```

(ii) Function overriding :-

- Function overriding is the concept that allows two classes to have a function with same name. function overriding is accomplished by using inheritance and virtual functions.
- As we know every derived class inherits all the functions of its base class, in this case all the member functions of a derived class override the member functions of base class.

eg.

```
#include<iostream>
using namespace std;
class ~~Animal~~ base
{
    ~~the~~ public:
        virtual void display()
        {
            cout<<" Display of base class";
        }
};
class Derive :public base
{
    public:
        void display()
        {
            cout<<" Derived class Display";
        }
};
int main()
{
    Derive d;
    d.display();
    return 0;
}
```

(10) What is function overloading. write a pro: defination of 3 overloaded function with can add two integer, Float and double Numbers?

→ Function overloading:

refer prev.

Program:

```
#Include<iostream>
using namespace std;
class ~~add~~ cal
{
    public:
    int add(int a ,int b)
    {
        return a+b;
    }
    Float add( Float a, Float b)
    {
        return a+b;
    }
    Double add (double a ,double b)
    {
        return a+b;
    }
};
int main()
{
    cal obj;
    cout<<" Integer Addition: "<< obj.add (5,7);
    cout<<" Float Addition: "<<obj.add (3.5, 2.8);
    Cout<<" Double Addition: "<<obj.add( 3.4,4.2);
    return 0;
}
```

(11) What is virtual function ? Explain how to achieved run time polymorphism.

→ ① Virtual function:

- It is a function declared within a base class that is meant to be overridden or redefined by derived class.
- It allows for late binding or dynamic dispatch, enabling the selection of the appropriate function implementation at runtime based on the actual type of the object.

② - When a virtual function is redefined by a derived class, the keyword virtual is not needed. The virtual function written in base class acts as interface and the function defined in derived class acts as different forms of the same function. This property of virtual function brings the runtime polymorphism.

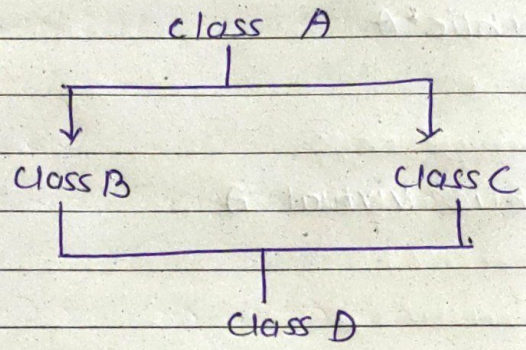
OR

Runtime polymorphism in c++ is achieved using virtual functions and the concept of dynamic binding. When a base class has virtual functions, and these functions are overriden in derived classes, the actual function call is determined at runtime based on the type of object being referred to, rather than compile time.

(12) Explain virtual base class and virtual function with example.

→ Virtual Base class :-

- The virtual base class is concept used in multiple inheritance to prevent ambiguity between multiple instances.
- For example, suppose we created a class "A" and two classes "B" and "C" are being derived from class "A".



- In the above fig. class B & class C are base class of class D so acc. to fig. class 'A' will be inherited twice in class 'D'.
- So when class 'D' will try to access the data member or member function of class A it will cause ambiguity for compiler and the compiler will throw error.
- To solve this we will make class A as virtual base class. To make virtual base class we used virtual keyword.

```
#include<iostream>
using namespace std;
class A
{
  public:
      Show ()
      {
          cout<<" Hello From A";
      }
};
                    virtual
class B : public A
{
};
class C : public virtual A
{
};
class D : public B , public C
{
};
int main()
{
      D obj;
      cout<<" a : "<<obj.a;
      return 0;
}
```

Virtual functions:

- A member functions in the base class which is declared using virtual keyword is called virtual functions.
- They can also be ~~derived it~~ redefined in derived class.

eg.

```
#include<iostream>
using namespace std;
class base
{
    public:
      virtual void print()
      {
          cout<<" print base class";
      }
    void show()
      {
          cout<<" Show base class";
      }
};
class derived : public base
{
    public:
      void print()
      {
          cout<<" print derived class";
      }
      void show()
      {
          cout<<" Show derived class";
      }
};
```

```
int main()
{
    base * bptr;
    derived d;
    bptr = &d;

    bptr -> print();
    bptr -> show();
    return 0;
}
```

Output:

print derived class
show base class.

(13) What is pure virtual function? Illustrate use of pure virtual function?

→ Pure virtual function:

- It is a virtual function that is declared in a base class but has no implementation.
- It is marked with "=0" at the end of its declaration.
- The purpose of pure virtual function is to make a class abstract, meaning it cannot be instantiated on its own, and any derived class must provide an implementation for pure virtual function.

Eg: • Refer Question 14.

```
class b
{   virtual void draw() = 0;
};
Class c: derived public b
{
    void draw() { }
};
```

Rainbow
PAGE:
DATE: / /

(14) Explain Abstract class concept with example.

→ Abstract class:

- It is a class that cannot be instantiated on its own because it contains one or more pure virtual functions. It serves as a blueprint for other classes and provides an interface that must be implemented by its derived classes.

example:

```
#include<iostream>
using namespace std;
class shape
{
    public:
        virtual void draw() const=0;

        void displayArea() const
        {
            cout<<" Area";
        }
};
class circle: public shape
{
    public:
        void draw() const override
        {
            cout<<" Drawing circle";
        }
};
```

```
class Rectangle : public Shape
{
    public:
        void draw() const override
        {
            cout << " Drawing rectangle";
        }
};

int main()
{
    Circle c;
    Rectangle r;

    c.draw();
    r.draw();

    c.displayArea;
    return 0;
}
```

IV Unit

Rainbow
PAGE:
DATE: / /

① What is stream? Explain type of stream available in C++.

→

- In C++ input and output is performed on sequence of bytes or more commonly known as stream.

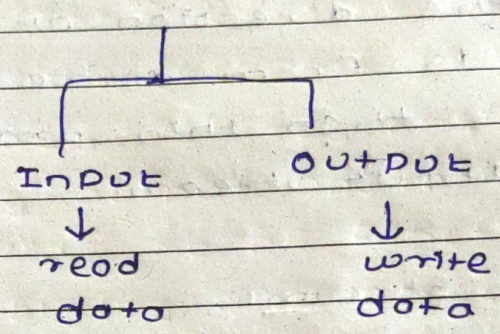
① Input stream:-
IF the direction of flow of bytes is from device to main memory then this process is called Input Stream.

② output Stream:-
- IF the flow of bytes is from main memory to device then this process is called output stream.

• Types of Stream available in C++:-

① Text Stream:-
- Text Stream is sequence of character.
- The Stream hold the same character that we see when Stream data is displayed
- only few character like newline, tab is stored according to system.

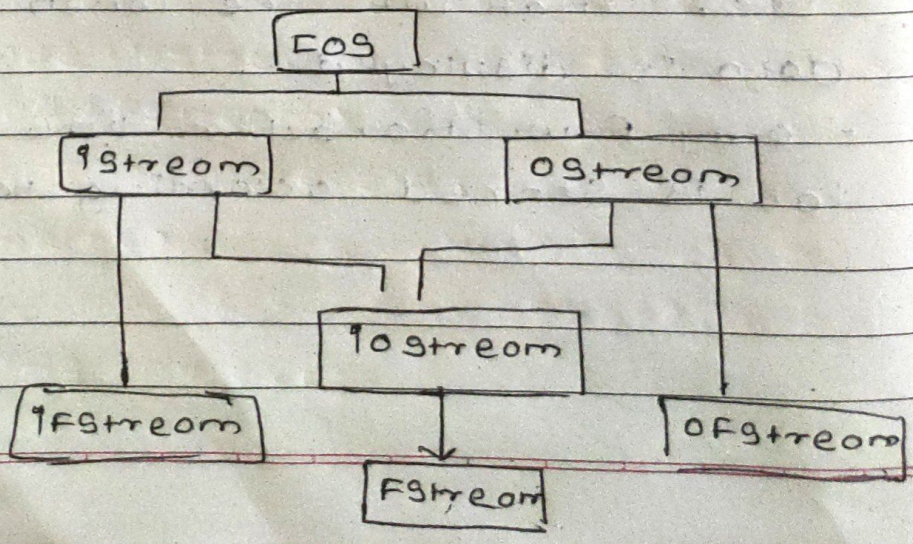
② Binary Stream:-
- Binary stream is collection of bytes.
- Binary stream has one to one relation between bytes in stream and device file.
- No of bytes in file and no of bytes in device file is same.



② What are stream classes? and their use provide heirchy of Stream classes.

→

• Stream classes:-
- The classes which facilitates input and output operation on files and other I/O devices is called Stream class.



- Facilities provided by these stream classes:-

① The ios class:-
- The ios class is responsible for providing all input and output facilities to all other stream class.

② The istream class:-
- This class is responsible for handling input stream.
- It provide no of function for handling char, string and object such as get, read, ignore etc.

③ The ostream class:-
- This class is responsible for handling output stream
- It provide no of function for handling char, string and object such as write, put etc.

④ The Fstream class:-
- This class is responsible for handling file related input and output operation.

Rainbow
PAGE:
DATE: / /

⑤ ifstream :-

- This class is responsible for handling input operation on file.
- It provide no of function for handing char, string and object such as openc), getc), readc), getlinec).

⑥ ofstream :-

- This class is responsible for handling output operation on file.
- It provide no of function for handling char, string and object such as openc), putc), writec).

③ What are fstream, ifstream and ofstream? Illustrate with example.

→

① ifstream :-

Explaination refer @2.

Example :-

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    string myext;
```

```
ifstream myReadFile("filename.txt");
while (getline (myReadFile, mytext))
{
        cout<<mytext;
}

myReadFile.close();
```

② ofstream :-

Explaination refer Q2.

Example :-

```
#include<iostream>
#include<fstream>
using namespace std;

int main()
{
    ofstream myFile("filename.txt");
    myFile<<"Files written";
    myFile.close();
}
```

④ What is stream? Write program to illustrate the stream error concept.

→

Refer Q1 For- What is stream?

* Program to illustrate stream error concept:-

```
#include<iostream>
using namespace std;

int main()
{
    cerr<<"welcome to stream!";
    cout<<"welcome to stream!"<<cout;
    return 0;
}
```

⑤ What is File pointer? Write a note a File opening and File closing?

→

* File pointer:

- A File pointer stores the current position of read or write within a File.
- Each File has two pointer associated with it when it is opened.
- one pointer is used For reading data in File is called get pointer and other is used to write data in File called put pointer.

* File opening:-

- Opening a File establishes a connection between program and

File, enabling read or write operations.

* Syntax :-

File = open ("Filename.txt", "r")

* File closing :-

- closing a file ensure that all data is properly written to file.

Syntax :-

File.close()

⑥ Explain File opening mode in detail?

→

- When we open File we mention the Filename and mode in which the file is to be used.
- Where, File name is string and mode is int.

* Different file mode :-

① ios::in :-

Open for input operation

② ios::out :- opens for output operation

③ ios::binary :- open in binary mode

④ ios::ate :- set initial position at end of file.

⑤ ios::app :- All output operation are performed at end of file appending content to current content of file.

⑥ ios::trunc :- If file is open for output operation and it already existed, its previous content is deleted and replace by new one.

⑦ Write a program using open(), eof() and getline() function to open and read content line by line?

→

```
#include<iostream>
#include<fstream>
int main()
{
    char filename[20]="demo2.txt";
    char line[100];
    ifstream obj;
    obj.open(filename);
    cout<<"In file data";
    while(!obj.eof())
    {
        obj.getline(line,100);
        cout<<line<<endl;
    }
obj.close();
return 0;
}
```

Rainbow
PAGE:
DATE: / /

(9) Explain error handling during file operation?

→

When error occurs in file handling, flags are set in state according to general category of error.

* Flag and their error catogories :-

(1) ios::goodbit :-
- This state flag indicates that there is no error with stream.
In case this status variable has value 0.

(2) ios::badbit :-
- This state flag indicates that stream is corrupted and no read/write operation can be performed.

(3) ios::failbit :-
- This state flag indicates that input/output operation failed.

(4) ios::eofbit :-
This state flag indicates input operation reached end of input sequence.

Rainbow

PAGE:
DATE: / /

10) Write a program to create a File, read and write record into it every record contain employee name, id and salary store and retrieve atleast 3 data.

→

```
#include<iostream.h>
#include<fstream.h>
int main

class Employee
{
    int ID, salary;
    char name[40];
    public:
        void input()
        {
            fflush(stdin);
            cout<<"Enter Name :";
            gets(name);
            cout<<" Enter ID :";
            cin>>ID;
            cout<<" Enter Salary :";
            cin>>salary;
        }

        void output()
        {
            cout<<"Name:"<<name<<endl;
            cout<<" ID:"<<ID<<endl;
            cout<<"Salary:"<<salary<<endl;
        }
};
```

```
int main()
{
    char filename[20] = "emp.txt";
    int i;

    ofstream fout;

    fout.open(filename);

    Employee emp;

    cout << "Enter 3 employee data=";

    for(int i=0; i<3; i++)
    {

        emp.input();
        ~~fout.open~~
        ~~fout.write~~
        fout.write((char*)&emp, sizeof
                                (emp));
    }
    fout.close();

    cout << "Data saved";

    ifstream fin;
    fin.open(filename);

    cout << "Employee data from file";
```

```
For (int i=0; i<3; i++)
{
    fin.read (char * &emp, sizeof(emp));
    emp.output();
    cout << endl;
}
fin.close();
return 0;
}
```

(11) Explain command line argument in c++ Write a program for same.

→

- In c++, command line argument are parameter provided to c++ program when it is executed from command line or terminal.
- This argument allow you to pass information to your program at run time.
- The main function i.e int main() contain two parameter (argc (argument count) and argv (string of argument)).

* Syntax:-

```
int main (int argc, char *argv[])
```

Rainbow

PAGE:
DATE: / /

*program:-

```
#include<iostream>
#include<fstream>
using namespace std;

int main(int argc, char *argv[])
{
    int no, marks;
    char name[40];

    if(argc>=2)
    {
        ofstream fout;
        fout.open(argv[1]);
        cout<<"Enter name:";
        cin.getline(name,40);
        cout<<"Enter Roll no:";
        cin>>no;
        cout<<"Enter marks=";
        cin>>marks;

        fout<<"Roll no"<<no;
        fout<<"Name"<<name;
        fout<<"marks"<<marks;
        fout.close();
        cout<<"Data written to file";
    }
    else
    {
        coo+<<"enter File name
                at cmd";
    }
    return 0;
}
```

Polymorphism
↳ Runtime : Dymamic or late binding
↳ Compile-time : Static or early binding

UNIT-VI

1] Explain class template and function template with suitable example.

⇒ Class template :

- A class template is way to create a generic class that can work with different data types without having to rewrite the code for each type.
- It allows you to define a blueprint for a class where certain parts of the class (eg. data types, functions) are parameterized by one or more template parameters.

Syntax :

```
Template <classtype> class classname
{
        // body of class.
};
```

program :

```
#include<iostream>
using namespace std;
template<class T>
class compare
{
    T a,b;
```

Program:

```
#include<iostream>
template<typename T>
class myconst
{
    private:
        T element;

    public:
        myconst(T arg):element(arg)
        { }

        T getvalue() const
        {
            return element;
        }
};
int main()
{
    Mycont<int> intcont(52);
    cout<<" integer container value: "<<
                        intcont.getvalue();

    Mycont<double> doublecont(3.14);
    cout<<" Doubl container value: "<<
                        doublecont.getvalue();

    return 0;
}
```

(II) Function template:

- It is a way to create a generic function that can work with different data types without having to rewrite a code for each type.
- It allows you to define a template for a function where one or more parameters or the return type are specified as template parameters.

Syntax:

```
Template <
name_ofdatatype function_name(namedatatype id1,
                              namedatatype id2)
```

For example:

```
#include <iostream>
template <class T>
T minimum(Ta, Tb)
{
    if (a<b)
        return a;
    else
        return b;
}
int main()
{
    cout<<" min(10,20) : "<< minimum(10,20);
    cout<<" min(10.3,67.2) :"<<minimum(10.3,67.2);
    return 0;
}
```

2] What is user defined exception? Explain with example.

⇒ User defined Exception:

- It is an exception to create a custom exception class by the programmer to handle specific error conditions that may arise during the execution of a program
- User defined exception or custom exception is creating your own exception 'class' and throws that exception using 'throw' keyword. This can be done by extending the class Exception.

Cpp Program to demonstrate User-defined exception:

```
#include<iostream>
#include<exception>
using namespace std;
class Divide_By_zero Exception : public exception
{
    public:
     const char *what() const throw()
     {
         return "Divide By zero Exception";
     }
};
int main()
{
    try
    {
        int a,b;
```

```
        cout<<" Enter two numbers :";
        cin>>a>>b;

        IF (b==0)
        {
            Divide_By_ZeroException d;
            throw d;
        }
        else
        {
            cout<<" a/b : "<<a/b;
        }
    }
    catch(Exception &e)
    {
        cout<< e.what();
    }
}
```

Output:

```
Enter two numbers : 10   2
 a/b: 5
Enter two Numbers: 1   0
 Divide By zero Exception.
```

3] What is namespace? Demostrate namspace with example.

⇒ - Namespace are used to group the entities like class, variables, objects, function under a name.
- The Namespace helps us to divide global scope into sub-scopes, where each sub-scope has its own name.

Rules of Namespaces:

(i) The namespace must be defined using keyword namespace
(ii) The namespace definition must appear at global scope, or it can be present inside another namespace (ie. nested)
(iii) Definition of namespace must not be terminated with semicolon.
(iv) Namespace declaration doesnot ~~require~~ have ~~accep~~ access specifiers (public or privat

Syntax:

```
namespace namespace_name
{
    //body
}
```

Example:

```
#include<iostream>
using namespace std;
namespace ns1
{
    int a=5;
}
namespace ns2
{
    char a[] = "Hello";
}
int main()
{
    cout<<" ns1::a;
    cout<<" ns2::a;
    return 0;
}
```

Unit-5

Rainbow
PAGE:
DATE: / /

(13) Write function template for finding minimum value stored in array?

→

```
#include<iostream.h>
#include<conio.h>
template<class T>

T minimum(T a[], int size)
{
    T min = a[0];

    For(int i=0; i<size; i++)
    {
        IF(a[i] < min)
        {
            min = a[i];
        }

        return min;
    }
}

int main()
{
    int a[10], size, i, min1;
    Float b[10], min2;
    clrscr();
    cout<<"Enter the size:\n";
    cin>>size;
    cout<<"Enter the integer:";
```

```
For(i=0;i<size;i++)
{
    cin>>a[i];
}
cout<<"Enter Floating array=";

For(i=0;i<size;i++)
{
    cin>>b[i];
}

min1 = minimum(a,size);
min2 = minimum(b,size);

cout<<"Minimom integer element=";
cout<<min1;

cout<<"minimum Floot element=";
cout<<min2;

getch();
return 0;
}
```

Rainbow
PAGE:
DATE: / /

(11) What is exception? How is an exception handled in c++?

→

*Exception:-

- In c++, the unexpected event that occur during the execution of program is called exception.
- It distrupt the normal flow of program and can be caused by various reason such as divigion by zero or acessing an invalid memory address.

*To overcome ~~exception~~ these prombiem, we need to handle exception:-

- Handling exception can be done commonly in following three woyg:-

① Progroms take a remedial action and proceed with rest of code
② Just show error condition and ~~Pro~~ repeat the port that generated error.
③ Jugh show proper error messoge guiding user for prombiem and terminate progrom.

- For handling exception c++ provide with three keyword: try, catch, throw; try-catch are used in pair.

Rainbow

PAGE:
DATE: / /

① Try block :-

- The try block allows you to define a block of code that may generate exception.
- If the try block has no error, all code in try block work normally and catch block is skipped and program proceed rest of code after catch block.

*Syntax :-

```
try
{
    // code that may cause an exception
}
```

② Catch block :-

- The catch block allow you to define a block of code to be executed, if error occur in try block.
- We can have multiple catch in catch block and execute multiple statement.

* Syntax :-

```
catch (Type variable)
{
    // code to handle exception
}
```

(3) Throw:-

- The throw keyword throws an exception when problem is detected.
- It is used to print customized error message.

*Syntax:-

```
try
{
    throw exception;
}
```

(10) Difference between overload function and function template with suitable example?

→

Function overloading	Function template
(1) This feature comes under polymorphism	(1) This function comes under generic programming.
(2) mutiple function are defined with same name for different operation.	(2) only one function is defined with generic data type to perform particular task.

Rainbow

PAGE:
DATE: / /

③ Function Parameter are specified as c++ type	③ Function parameter are specified as mixture of template type and c++ type.
④ Overload function have different no of Parameter	④ Since one function is defined, no of parameter will remain same.
⑤ Each overloaded function can have its own different logic and code	⑤ Since there is one template function, same logic and code work For different cals.
⑥ For different data type different Function is required	⑥ For different data type one function is required

(9) What is generic programming how it is implemented in CPP?

→

*Generic programming:-

- It is type of programming in which we are designing the algorithm, where any type of data is specified at run time
- We are writting a generalized code which will work on different data type which is decided by complier.

- Template is best example of Generic Progromming in C++.
- In traditional Progromming language we need to use different algorithm or function for different data type.
- For example, progrom for performing the arithmetic operation on integer number will not be applicable for performing arithematic operation on Float number.

*Progrom to handle addition of two number using template.

Rainbow

PAGE:
DATE: / /

```
#include<iostream>
using namespace std;

int main()
{
    cout<<"\n Addition of two number=";
    int a=10, b=20;
    int c;
    c=a+b;
```

```
#include<iostream>
using namespace std;

template<class temptype>
TemptypeAdd(temptype a, temptype b)
{
    Temptype c;
    c=a+b;
    return c;
}
int main()
{
    int p=4, q=3;
    Float x=3.4, y=1.2, z;

    r = Add<int>(p,q);
    cout<<"Sum="<<r;

    z = Add<float>(x,y);
    cout<<"Sum="<<z;
    return 0;
}
```

Page No:
Date: / /

⑦ write a short Note on typename and expart keyword in C++;

→

typename : In a template declaration typename can be used as a alternative to class to declare type template parameters and template parameters.

- typename is keyword in C++ used while writing templates.
- use :
  - It is used for specifying that dependenet Name in the template defination/declaration a type.
  - It is used as alternative to class keyword while writing / creating template.

* Example :-

```
template <typename T>
class template1
{ public:
        typename T::Nestedtype
                getValue();
};
```

- typename keyword is used in template declarations to indicate that a dependant name is a type. It is often used when working with template involving nested types or dependent names.

export :-

- The export keyword can be used to export the template definitions to other file the keyword export is preceded the template keyword in the template defination.
- this allows the declaration to be in one file and the definition to be in another.

```
// declaration (usually in a heaader file)
template <typename T>
 export
 T add (Ta, Tb);

// Definition (usually in separate
                implementation file)
template <typename T>
T add (Ta, Tb)
{
    return a+b;
};
```

Page No:
Date: / /

* template :-

- Template is a like a data structure where we can add data dynamically

* addition of two no using Template.

```
# include <iostream>
template <typename T>
T add (T num1, T num2)
{
    return num1 + num2;
}
int main()
{
    cout << add <int>(4,5)<<endl
    cout << add <double> (3.7,4.8)<<endl

    return 0;
}
```

Q4. Explain class template using multiple parameters. Write a program in cpp.

→

- class templates are used when we need to perform similar operations on group of data items.

- The functions & data members of the class can either work with template (generic) type or mixed data types.

Syntax:

```
template <class TypeName>
class Name
{
    // Data members
    // Functions
};
```

- template is a keyword to that start template data definition.

- class keyword in < > is just to define new template Type Name.

- In C++, a class template with multiple parameters allows you to create a generic class that can work with different types.

- In class template we can declare two parameters. These parameters will represent the types that class will use.

Syntax:

```
template < class T, class U>
class Pair
{
    // Data members
    // functions
};
```

Benefits:

1. Code Reusability:
   - With class templates, you can write a generic class once & reuse it with different types.

2. Type safety:
   The compiler ensures type safety for each instantiation of the template, preventing incompatible types.

3) Flexibility:

Allows you to create instances of the class with various combinations of types.

Program:

```
#include <iostream>
using namespace std;

template<typename T, typename V>
class Pair
{
  private:
     T first;
     V second;
  public:

  Pair (T f, V s) : first(f) , second(s) {}

  void display()
  {   cout<<" Pair: "<<first<<", "<<second<<endl;
  }
};

int main() {
    pair<int, double> myPair(1, 3.14);
     myPair.display();

     return 0;
  }
```

Page No.
Date

Q5. Explain Exception handling mechanism in cpp ? Write a program in cpp to handle divide by zero exception.

→

Program:

```
#include <iostream>
using namespace std;

int main()
{
    int numerator, denominator;

    cout<<" Enter numerator;
    cin>> numerator,

    cout<<" Enter denominator";
    cin>> denominator;

    if (denominator == 0) {
     throw runtime_error ("Error: Divide by zero
        is not allowed.");
    }
    double result= static_cast<double>(numerator)/
            denominator;

    cout<<" Result: "<< result;
}
    catch (const runtime_error& e){
      cerr <<e.what();
}
    return 0;
}
```

Q6. How multiple catching is implemented in exception handling?

→

- For exception handling we use try, catch & throw keywords.

- 'catch' block is used for handling exception.

- We can define multiple 'catch' blocks to handle different exceptions with a try block.

- This facilitater programmer to handle each type of exception separately.

- If there is no exception thrown in try block then none of catch block works.

- If exception is thrown in try block, then program jumps to catch blocks.

- The catch block whose Type matches with the type of exception thrown, only works, & the other catch blocks are skipped.

```
#include <iostream>
#include <stdexcept>

int main()
{
    try{
    int numerator, denominator;

    cout<<" Enter numerator";
    cin>>numerator;

    cout<<" Enter denominator";
    cin>> denominator;

    if (denominator == 0){
      throw runtime_error (" Error: Division by
        zero is not allowed.");
    }
    int result= numerator / denominator
}
    catch ( const runtime_error& e)
    {  cerr<<" Runtime Error:" <<e.what();
    }
    catch ( const logic_error& e)
    {
        cerr<<" Logic Error:" <<e.what();
    }
    return 0;
}
```

Unit-6

Rainbow
PAGE:
DATE: / /

① Write a Program to implement map in STL.

→

```
#include<iostream>
#include<map>
#include<string>
using namespace std;

int main()
{
    typedef map<string,int>maptype;
    maptype pm;
    pm.insert(pair<string,int>("Delhi",123));
    pm.insert(pair<string,int>("Pujab",1234));
    pm.insert(pair<string,int>("Bengal",12345));

    maptype::iterator itr;
    cout<<"==== Population of State ====";
    cout<<"Size of population"<<pm.size();
    string s_name;
    cout<<" Enter state Name=";
    cin>>s_name;
    itr=pm.find(s_name);
    if(itr!=pm.end())
    {
        cout<<state_name<<"population is"
        <<itr->second;
    }
    else
    {
        cout<<"Key not found";
    }
    pm.clear();
```

Rainbow
PAGE:
DATE: / /

② What is container? List container classes in CPP. Explain any of them using program

→

- container is class template that provides a way to store and acess element.
- These are object which hold multiple other object.
- These classes provide a ready code to manipulate object held in container.
- eg - vector, map, queue.
- Container are mainly three types
  ① Sequence container
  ② Associative container
  ③ Unordered Associative container

*Container class in CPP:-

① Array
② vector
③ list
④ map
⑤ List
⑥ Deque

* Array:-

- Array is fixed size sequence container that hold element ordered in specific linear sequence.
- Array have fixed size and do not manage the allocation of its element through an allocator.

* Program :-

```
#include <iostream>
#include <array>
#include <tuple>
using namespace std;

int main()
{
    array<int,6> ar = {1,2,3,4,5,6};

    cout << "The array element using at()
            Function";

    for(int i=0; i<6; i++)
    {
        cout << "ar.at(i) << " ";
        cout << endl;
    }


    //Printing using get()
    cout << "Array element using get() ";
    cout << get<0>(ar) << " " << get<1>(ar) << " "
    cout << get<2>(ar) << " " << get<3>(ar) << " ";
    cout << get<4>(ar) << " " << get<5>(ar) << " ";
    cout << endl;

    //printing using operator[]
    cout << "the array element using operator[]";
    for(int i=0; i<6; i++)
    {
        cout << ar[i] << " ";
        cout << endl;
    }
```

return 0;
}

③ Explain forword, bidirectional, random acegg iterator with help of suitable example.

→

*Iterator :-

- Iterator are used to point a memory address of STL container.
- Iterator are used to acegg the element stored in container.
- Using iterator one can can cycle through content i.e acegg the object of container in an order.
- Since iterator are like pointer we can increment or decrement iterator.
- For example :-
  iterator can acegg element of array one by one.

* Bidirectional iterator :-

- The Bidirectional iterator is an iterator that support all feature of forword iterator plus it add one more feature i.e decrement operator(--).
- We can move backword by decremen ting an iterator.

* operator used in bidirectional :-

1. Increment (++)
2. Assignment (==)
3. equal operator (=)
4. Not equal (!=)
5. Decrement (--)

* code :-

```
int main()
{
    list<int> v1 = {1,2,3,4,5};
    list<int> ::iterator i;

    for (i=v1.end(); i!=v1.begin(); --i)
    {
        if (i!=v1.end())
        {
            cout << (*i) << " ";
        }
    }

    cout << (*i);
    return 0;
}
```

* Random access iterator :-

- A Random access iterator is an iterator provide random access of an element at arbitary location.

Rainbow

PAGE:
DATE: / /

- It has all Feature of Forword Iterator plus it add one more feature i.e pointer addition and pointer Subtraction to provide random acess to element.

*exomDie :-

```
int main()
{
    vector<int> v1={1,2,3,4,5};

    vector<int>::iterator i;

    For (i=v1.begin(); i!=v1.end(); ++i)
    {
        *i=7;
    }

    For (i=v1.begin(); i!=v1.end();++i)
    {
        cout<<(*i)<<"GG";
    }
return 0;
}
```

UNIT - VI

1) What is stack? How is it important in STL?

→ * STACK:

- stack are type of container adapter, specially designed to operate in LIFO (last in first out), where elements are inserted and extracted only from one end of the container
- The end from which elements are added or removed is called as TOP.
- Stack uses some other standard container like vector or Dequee for its working.
- The function associated with stack are:
  (i) empty()
  (ii) size()
  (iii) top()
  (iv) push(g)
  (v) pop()
  Time complexity of all oper function is O(1).

Program:

```
#include<iostream>
#include<stack>
using namespace std;
void showstack(stack<int> a)
{
        while(!s.empty())
        {
                cout<<"\t"<<s.top();
                s.pop();
        }
        cout<<"\n";
}
```

display function

```
int main()
{
        stack<int> s;
        s.push(10);
        s.push(30);
        s.push(20);
        s.push(5);
        s.push(1);

        cout<<"The stack is :";
         showstack(s);

        cout<<"\n s.size() :"<<s.size();
        cout<<"\n s.top() :"<<s.top();

        cout<<"\n s.pop() :"<<s.pop();
        showstack(s);

        return 0;
}
```

Output :

```
The stack is : 1  5  20  30  10
s.size() : 5
s.top() : 1
s.pop() : 5  20  30  10
```

5] State functions of vector STL ? Write program to explain the same.

→ certain functions associated with the vector are :

(i) begin() :
- Returns an iterator pointing to the first element in vector.

(ii) rbegin() :
- Returns a reverse iterator pointing to the last element in vector. It moves from last to first element.

(iii) cbegin() : Returns a constant iterator pointing to the first element in vector.

(iv) end() :
- Returns an iterator pointing to the theoretical element that follows the last element in the vector.

(v) rend() :
- Returns a reverse iterator pointing to the theoretical element preceding the first element in the vector (considered reverse end).

(vi) cend() :
- Returns a constant iterator pointing to the theoretical element that follows the last element in the vector.

(vii) crbegin(): Returns a constant reverse iterator pointing to the last element in vector.

(viii) crend(): Returns a constant reverse iterator pointing to the theoritical element preceding the first element in vector.

program:

```
#include<iostream>
#include<vector>
Using namespace std;
int main()
{
        vector<int> gl;

        for(int i=1;i<=5;i++)
        {
                gl.push_back(i);
        }

        cout<<"Output of begin and ends:";
        for(auto i=gl.begin();i!=gl.end();++i)
        {
                cout<<*i<<" ";
        }

        cout<<"\nOutput of crbegin and crend:";
        for(auto ir=gl.crbegin();ir!=gl.crend();++ir)
        {
                cout<<*ir<<" ";
        }
return 0;
}
```

Date / /

(7) what is STL & List and explain different types of STL container.

STL :- standard Template Library.

- STL is powerful set of c++ templat classes to provide general purpose classes and fuctions with templates that implement many popular and commonly used algorithms and data structures like vector, lists, queues and stacks.

(1) container
(2) Algorithms
(3) Iterators.

(1) containers :

- container are the objects which hold multiple other objects. These object can be same or different types of element.
- These ~~element~~ classes provide ready code (fuctions) to manipulate objects held in the containers
- e.g : Vector , Map , Queue etc.
- container are mainly three types: sequence containers, Associative container, Unordered Associative container.

(2) Algorithm :

- STL provide number of algorithim a algorithm refers to a set of template fuctions that perform various operation on sequence of elements
- Algorithm library contains built in fuctions which perform complex operations on the data structures.
- Algorithms act on container They provide the means by which you will manipulate the content of container;
- Their capabilities include initialization, sorting, searching and transforming the contents of containers.

(3) Iterator :

- Iterator are the object that allows iterating over the elements of a container (like arrays, vectors, lists, etc) In generic way.
- Iterators are objects that act, more or less, like pointers.
- It point element in the container.
- Iterator actually act as a bridge between container and algorithm using iterator one cycle through the contents
- Iterators are like pointers we can increment or decrement iterator. depending on type of Iterator.

* what is purpose of iterator & algorithm.

→ 1] Iterator :-
- Iterator are the object that allow iterating over the elements of a container. (like array, vector, list)
- purpose :-
1] Abstraction of iterating : Iterators provide a uniform way to iterate over elements in a container, abstracting away the underlying details of the data structure
2] Generic Access : They allow algorithms to access elements in a container without knowing the specific details of the container's implementation.
3] support for different containers : since iterator provide a common interface, algorithms can work with various containers (array's, vectors list) as long as they support the required iterator operations.

2] Algorithms :-
- STL provide number of algorithm a refers to a set of templete fuction that perform various operation's on sequence of element.
- Purpose :-
1] Reusable operations : Algorith in the STL encapsulate comon operations such as sorting searching and manipulating of element in a

reusable and generic form.

2] consistent Interface
Algorithms use iterors to define the range of elements the operate on providing a consistent and predictable interface.

3] Efficiency and correctness:
- STL algorithms are implemented with efficiency and correctness in mind they are often highly optimized and using them can lead to more efficient code than implementing the same fuctionality from scarth.

Page No.
Date

Q9. Describe the process of using the STL algorithms for quick / heap sort.

→

Heap sort:

- The heap_sort() is an STL algorithm which sorts a heap within the range specified by start & end sorts the elements in the heap range [start, end] into ascending order.

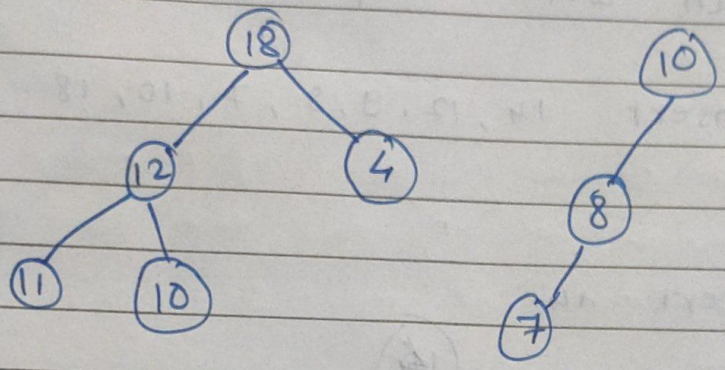
- The second form allows you to specify a comparison function that determines when one element is less than another.

- Heap is a complete binary tree or a almost complete binary tree in which every parent node be either greater or lesser than its child nodes.

max heap:

A max heap is a ~~complete binary tree or a almost~~ in which value of each node is greater than or equal to the value of its children nodes.

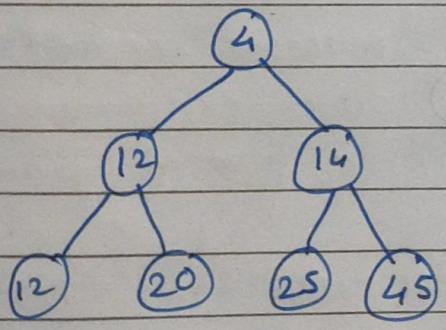
for example:



min heap:

A min heap is a tree in which value of each node is less than or equal to value of its children nodes.

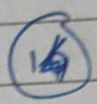
for example:



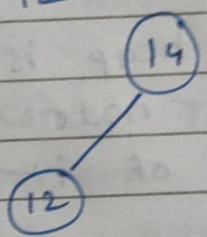
We can construct heap using top down approach with repeated insertion of element.

Insert 14, 12, 9, 8, 7, 10, 18

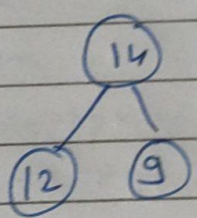
insert 14



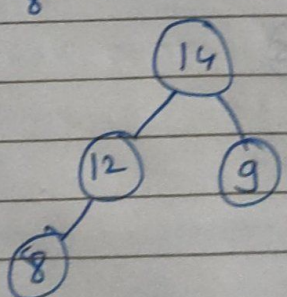
insert 12



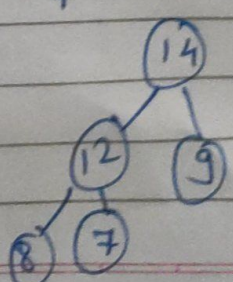
insert 9



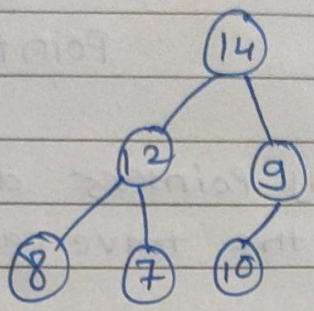
insert 8



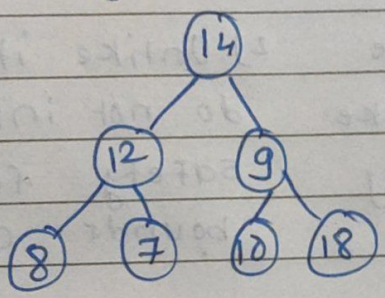
insert 7



insert 10



insert 18



After applying heap sort technique the sorted list for max heap will be

18 14 12 10 9 8 7

After applying heap sort technique the sorted list for min heap will be

7 8 9 10 12 14 18

Page No.
Date

Q10. → Difference between iterator & pointer.

Iterator	Pointer
1) Iterator provide a level of abstraction, allowing you to access elements of a container	1) Pointers do not abstract the traversal logic.
2) Iterators often come with safety features like bounds checking, reducing the risk.	2) Unlike iterators, pointers do not inherently provide safety features like bounds checking.
3) Iterators can be ~~b~~used with a variety of containers, such as arrays, linked lists, maps & more.	3) Pointers are more flexible & can be used for various purposes beyond simple traversal such as dynamic memory allocation & manipulation.
4) We can delete a pointer using delete.	4) Since an iterator refers to objects in a container, unlike pointers, there's no concept of delete of an iterator.
5) Not all iterators allow these operations i.e. decrement, etc.	5) We can perform simple arithmetic on pointers like increment, decrement etc.
6) Synatax: int *Ptr = &num	6) Syntax: class_name :: iterator ; ;