AMRUTVAHINI COLLEGE OF ENGINEERING, SANGAMNER

Department of

COMPUTER ENGINEERING

STUDENT MANUAL

OOP AND COMPUTER GRAPHICS LAB

COURSE	210247 (2019)
COURSE NAME	OOP AND COMPUTER GRAPHICS LAB
CLASS	SE COMPUTER –SEM I
EXAMINATION SCHEME	TW:25 PRACTICAL:25
CREDITS	02

Course Objectives

1	**Applying:** To understand and apply basic constructs of object oriented programming to solve problems.
2	**Applying:** To understand and implement inheritance, polymorphism, files and STL concepts to implement programs.
3	**Creating:** To understand and implement line drawing, circle drawing, clipping and polygon filling algorithms using object oriented concepts.
4	**Creating:** To understand and implement transformation algorithms, lighting and shading algorithms using object oriented programming.
5	**Creating:** To create fractals and animations using object oriented programming.
6	**Creating:** To implement graphics algorithm in OpenGL and object oriented programming

Course Outcomes (CO)

After successfully completing the course students will be able to,

CO. No.	Description	Bloom's Taxonomy Level
C210247.1	**Interpret** and **apply** object oriented program constructs like inheritance, polymorphism, and exception handling to solve complex problems.	4
C210247.2	**Interpret** and **apply** the concepts of file and templates for storing and retrieving the data from secondary storages.	4
C210247.3	**Use** the concepts STL and containers to implement flexible and memory efficient software.	3
C210247.4	**Analyze** and **apply** computer graphics algorithms for line-circle drawing, scan conversion and polygon filling with the help of object oriented programming concepts.	4
C210247.5	**Interpret** and **apply** the concept of windowing, clipping, curves, fractals, animation.	4
C210247.6	**Apply** logic to implement graphics algorithms, animation and gaming programs using OpenGL.	4

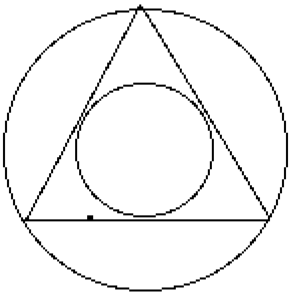
Amrutvahini College of Engineering, Sangamner
Department of Computer Engineering
OOP and Computer Graphics laboratory

INDEX

Class: S.E. **Div: _____** **Batch:________**

Sr. No.	Title of Experiments	Page no.	Date of		Remark	Signature
			Performance	Submission		
Part I : Object Oriented Programming						
Group A						
1	Implement a class Complex which represents the Complex Number data type. Implement the following 1. Constructor (including a default constructor which creates the complex number 0+0i). 2. Overload operator+ to add two complex numbers. 3. Overload operator* to multiply two complex numbers. 4. Overload operators << and >> to print and read Complex Numbers.					
2	Develop a program in C++ to create a database of student's information system containing the following information: Name, Roll number, Class, Division, Date of Birth, Blood group, Contact address, Telephone number, Driving license no. and other. Construct the database with suitable member functions. Make use of constructor, default constructor, copy constructor, destructor, static member functions, friend class, this pointer, inline code and dynamic memory allocation operators-new and delete as well as exception handling.					
3	Imagine a publishing company which does marketing for book and audio cassette versions. Create a class publication that stores the title (a string) and price (type float)					

	of publications. From this class derive two classes: book which adds a page count (type int) and tape which adds a playing time in minutes (type float). Write a program that instantiates the book and tape class, allows user to enter data and displays the data members. If an exception is caught, replace all the data member values with zero values.					
Group B						
4	Write a C++ program that creates an output file, writes information to it, closes the file, open it again as an input file and read the information from the file.					
5	Write a function template for selection sort that inputs, sorts and outputs an integer array and a float array.					
Group C						
6	Write C++ program using STL for sorting and searching user defined records such as personal records (Name, DOB, Telephone number etc) using vector container.					
7	Write a program in C++ to use map associative container. The keys will be the names of states and the values will be the populations of the states. When the program runs, the user is prompted to type the name of a state. The program then looks in the map, using the state name as an index and returns the population of the state.					
Part II : Computer Graphics						
Group A						
8	Write C++ program to draw a concave polygon and fill it with desired color using scan fill algorithm. Apply the concept of inheritance.					
9	Write C++ program to implement Cohen Southerland line clipping					

	algorithm.					
10	Write C++ program to draw the following pattern. Use DDA line and Bresenham‘s circle drawing algorithm. Apply the concept of encapsulation. 					
Group B						
11	Write C++ program to draw 2-D object and perform following basic transformations, a) Scaling b) Translation c) Rotation. Apply the concept of operator overloading.					
12	Write C++ program to generate snowflake using concept of fractals.					
Group C						
13	Write OpenGL program to draw Sun Rise and Sunset					
14	Write a C++ program to implement bouncing ball using sine wave form. Apply the conceptof polymorphism.					
Mini-Projects/ Case Study						
15	Design and implement game / animation clip / Graphics Editor using open source graphics library. Make use of maximum features of Object Oriented Programming.					

CERTIFICATE

This is to certify that Mr./Ms. ______________________________________
Roll No_________ Examination Seat no- ______________ has performed above mentioned practical’s in the college.

____
Faculty member Incharge
Date: / /2022

Prof. Paikrao R.L.
Head of the Department

Experiment No. 1 (OOP)

Aim: Implement a class Complex which represents the Complex Number data type. Implement the following operations:

1. Constructor (including a default constructor which creates the complex number0+0i).
2. Overload operator+ to add two complex numbers.
3. Overload operator* to multiply two complex numbers.
4. Overload << and >> to print and read Complex Numbers.

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool like G++

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

1) To understand concept of operator overloading.

2) To demonstrate overloading of binary operator, insertion and extraction operator.

Outcomes:

1) Students will be able to demonstrate use of constructor.
2) Students will be able to demonstrate binary operator overloading
3) Students will be able to demonstrate overloading of insertion and extraction operator using friend function.

Theory:

You can redefine or overload most of the built-in operators available in C++. Thus a programmer can use operators with user-defined types as well. Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

1. Unary Operator:

The unary operators operate on a single operand and following are the examples of Unary operators: The increment (++) and decrement (--) operators. Examples are the unary minus (-) operator and the logical not (!) operator.

The unary operators operate on the object for which they were called and normally, this operator appears on the left side of the object, as in !obj, -obj, and ++obj but sometime they can be used as postfix as well like obj++ or obj--. Following example explain how minus (-) operator can be overloaded for prefix as well as postfix usage.

```
#include <iostream>
using namespace std;
class Distance
{
private:
      int feet;                 // 0 to infinite
      int inches;               // 0 to 12 public:
// required constructors
Distance ( )  {
        feet = 0;
        inches = 0;
             }
Distance(int f, int i)  {
        feet = f;
        inches = i;
         }
void displayDistance()    // method to display distance

{
        cout<< "F: " << feet << " I:" << inches <<endl;
}
Distance operator- ()       // overloaded unary minus (-) operator

{
        feet = -feet; inches = -inches;
        return Distance(feet, inches);    }        };
int main()
{
        Distance D1(11, 10), D2(-5, 11);
        -D1;
          D1.displayDistance();    // displayD1
          -D2;
           D2.displayDistance();   // displayD1
           return 0;
}
```

2. Binary Operator:

Overloading with a single parameter is called binary operator overloading. Similar to unary operators, binary operators can also be overloaded. Binary operators require two operands, and they are overloaded by using member functions and friend functions.

Example:

```
using namespace std; class temp
{

complex operator + (complex c2)
{
complex c3; c3.x = x + c2.x; c3.y = y + c2.y; return c3;
}     };
```

ALGORITHM:

1. Start.
2. Create class complex with data members x and y and member functions accept(), display().
3. Initialize 3 objects c1, c2, c3 and k in main().
4. Define default constructor to initialize variables to 0+0i.
5. Define operator overloaded functions to add, subtract, multiply and divide two complex numbers.
6. Call the operator overloaded functions.
7. Use c3=c1+c2; to invoke the overloaded +operator.
8. Use c3=c1-c2; to invoke the overloaded -operator.
9. Use c3=c1/c2; to invoke the overloaded /operator.
10. Use c3=c1*c2; to invoke the overloaded * operator.
11. After performing the required operations calldisplay().
12. Stop.

Input :

Enter 2 complex and any 1 operator.

Output :

Desired arithmetic operation on complex no (i.e.+,-,*, /).

Conclusion: Thus implemented operator overloading for complex numbers.

Questions:

1. Write class student with suitable data and create object for class.

2. Write different constructors for student class.
3. What is Operator Overloading? Explain with example
4. Write Rules of Operator overloading?
5. What is Friend Function? Explain Syntax.

Experiment No. 2 (OOP)

Aim: Develop an object oriented program in C++ to create a database of student information system containing the following information: Name, Roll number, Class, division, Date of Birth, Blood group, Contact address, telephone number, driving license no. etc Construct the database with suitable member functions for initializing and destroying the data viz constructor, default constructor, Copy constructor, destructor, static member functions, class, this pointer, inline code and dynamic memory allocation operators-new and delete.

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool like G++

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

1) To understand use of different types of constructor and destructor.
2) To understand use of static members and inline keyword.
3) To understand this pointer, dynamic memory allocation operators like new and delete.

Outcomes:

1) Students will be able to demonstrate different types of constructor and destructor.
2) Students will be able to demonstrate use of static members and inline keyword.
3) Students will be able to demonstrate this pointer, dynamic memory allocation operators like new and delete.

Theory:

A special method of the class that will be automatically invoked when an instance of the class is created is called as constructor. Following are the most useful features of constructor.

1) Constructor is used for Initializing the values to the data members of the Class.
2) Constructor is that whose name is same as name of class.
3) Constructor gets automatically called when an object of class is created.
4) Constructors never have a Return Type even void.

5) Constructor is of Default, Parameterized and Copy Constructors.

The various types of Constructor are as follows:-

Constructors can be classified into 3 types

1 Default Constructor

2 Parameterized Constructor

3 Copy Constructor

Default Constructor:- Default Constructor is also called as Empty Constructor which has no arguments and It is Automatically called when we creates the object of class but Remember name of Constructor is same as name of class and Constructor never declared with the help of Return Type.

Parameterized Constructor: - This is another type constructor which has some Arguments and same name as class name but it uses some Arguments So For this We have to create object of Class by passing some Arguments at the time of creating object with the name of class. When we pass some Arguments to the Constructor then this will automatically pass the Arguments to the Constructor and the values will retrieve by the Respective Data Members of the Class.

Copy Constructor: - This is also another type of Constructor. In this Constructor we pass the object of class into the Another Object of Same Class. As name Suggests you Copy, means Copy the values of one Object into the another Object of Class .This is used for Copying the values of class object into an another object of class So we call them as Copy Constructor and For Copying the values We have to pass the name of object whose values we wants to Copying and When we are using or passing an Object to a Constructor then we must have to use the & Ampersand or Address Operator. Destructor:

As we know that Constructor is that which is used for Assigning Some Values to data Members and For Assigning Some Values this May also used Some Memory so that to free up the Memory which is Allocated by Constructor, destructor is used which gets Automatically Called at the End of Program and we doesn’t have to Explicitly Call a Destructor and Destructor Cant be Parameterized or a Copy This can be only one Means Default Destructor which Have no Arguments. For Declaring a Destructor we have to use ~tiled Symbol in front of Destructor.

Static members

A class can contain static members, either data or functions. A static member variable has following properties:

- It is initialized to zero when the first object of its class is created. No other initialization is permitted.
- Only one copy of that member is created for the entire class and is shared by all the objects of that class.
- It is the visible only within the class but its lifetime is the entire program.

Static data members of a class are also known as "class variables", because there is only one unique value for all the objects of that same class. Their content is not different from one object static members have the same properties as global variables but they enjoy class scope. For that reason, and to avoid them to be declared several times, we can only include the prototype (its declaration) in the class declaration but not its definition (its initialization). In order to initialize a static data-member we must include a formal definition outside the class, in the global scope of this class to another. Because it is a unique variable value for all the objects of the same class, it can be referred to as a member of any object of that class or even directly by the class name (of course this is only valid for static members. A static member function has following properties

- A static function can have access to only other static members (fun or var) declared in the same class
- A static function can be called using the class name instead of its object name *Class_name :: fun_name;*
- Static member functions are considered to have class scope.
- In contrast to non static member functions, these functions have no implicit **this** argument; therefore, they can use only static data members, enumerators, or nested types directly.
- Static member functions can be accessed without using an object of the corresponding class type.

The following restrictions apply to such static functions:

1 They cannot access non static class member data using the member-selection operators(**.**or **–>**).

2 They cannot be declared as **virtual**.

3 They cannot have the same name as a non static function that has the same

argument types.

E.g. // static members in classes

```
    class StaticTest { private: static int x; public: static int count)
      {
       return x;
         }
      };
  int StaticTest :: x = 9;
  int main()
     {
        printf("%d\n", StaticTest::count());
      }
```

Friend functions:

In principle, private and protected members of a class cannot be accessed from outside the same class in which they are declared. However, this rule does not affect *friends*. Friends are functions or classes declared as such. If we want to declare an

external function as friend of a class, thus allowing this function to have access to the private and protected members of this class, we do it by declaring a prototype of this external function within the class, and preceding it with the keyword *friend.*

Properties of friend function:

1. It is not in the scope of the class to which it has been declared as friend.
2. Since it is not in the scope of the class, it cannot be called using the object of that class It can be invoked like a normal function w/o the help of any object.
3. It can be declared in private or in the public part of the class.
4. Unlike member functions, it cannot access the member names directly and has to use an object name and dot operator with each member name.

```
// friend functions
#include<iostream>
using namespace std;
class CRectangle {
        int width, height; public:
        void set_values (int, int);
        int area () {
                return (width * height);
         }
      friend CRectangle duplicate (CRectangle);
};
void CRectangle::set_values (int a, int b) {
      width = a; height = b;
}
CRectangle duplicate (CRectangle rectparam)
{
        CRectangle rectres;
        rectres.width = rectparam.width*2;
        rectres.height = rectparam.height*2; return (rectres);
}
int main()
{
       CRectangle      rect,      rectb;
      rect.set_values(2,3);
       rectb = duplicate (rect);
       cout<<rectb.area(); return 0;
}
```

The duplicate function is a friend of CRectangle. From within that function we have been able to access the members width and height of different objects of type CRectangle, which are private members. Notice that neither in the declaration of duplicate() nor in its later use in main() have we considered duplicate a member of class CRectangle.

Friend classes

Just as we have the possibility to define a friend function, we can also define a class as friend of another one, granting that second class access to the protected and private members of the first one.

Pointers:

A pointer is a derived data type that refers to another data variable by storing the variables memory address rather than data. Declaration of pointer variable is in the following form :

Data_type * ptr_var;

Eg int * ptr;

Here ptr is a pointer variable and points to an integer data type.

We can initialize pointer variable asfollows inta, *ptr;   //declaration

ptr =&a   //initialization

Pointers to objects:

Consider the following eg

item X; // where item is class and X is object Similarly we can define a pointer it_ptr of type item as follows Item *it_ptr ;

Object pointers are useful in creating objects at runtime. We can also access public members of the class usingpointers.

Eg   item X;

item *ptr = &X;

the pointer ptr is initialized with address of X.

we can access the member functions and data using pointers as follows

ptr-> getdata();

ptr->show();

this pointer:

C++ uses a unique keyword called **this** to represent an object that invokes a member function. **this** is a pointer that points to the object for which *this* function was called. This unique pointer is automatically passed to a member function when it is called.

Important notes on this pointer:

- **this** pointer stores the address of the class instance, to enable pointer access of the members to the member functions of the class.
- **this** pointer is not counted for calculating the size of the object
- **this** pointers are not accessible for static member functions.
- **this** pointers are not modifiable.

Algorithm:

1. Start
2. Read personnel information such as Name, Roll number, Class, division, Date of Birth, Blood group, Contact address, telephone number, driving license no.etc
3. Print all information from database.

4. Stop

Input:

Personnel information such as Name, Roll number, Class, division, Date of Birth, Blood group, Contact address, telephone number, driving licence no. etc

.

Output:

Display personnel information from database. The result in following format:

Name DOB ……. Driving License No

1

2

.

.

N

Conclusion: Thus, students are able to implement database using constructors and friend functions.

QUESTIONS:-

1) What is Friend Function? Syntax.

2) What is Static Data Member and its Properties?

3) What is Dynamic Memory Allocation? Explain new and delete.

Experiment No. 3 (OOP)

Aim: Imagine a publishing company which does marketing for book and audio cassette versions. Create a class publication that stores the title (a string) and price (type float) of a publication. From this class derive two classes: book, which adds a page count (type int), and tape, which adds a playing time in minutes (type float).Write a program that instantiates the book and tape classes, allows user to enter data and displays the data members. If an exception is caught, replace all the data member values with zero values.

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool like G++

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

1) To learn and understand code reusability and demonstrate it using Inheritance concepts.
2) To learn, understand and demonstrate exception handling in object oriented environment.

Outcomes:

Students will be able to learn and understand inheritance and exception handling.

Theory:

Inheritance

Inheritance is the process by which objects can acquire the properties of objects of other class. In OOP, inheritance provides reusability, like, adding additional features to an existing class without modifying it. This is achieved by deriving a new class from the existing one. The new class will have combined features of both the classes. **Types of Inheritance**

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hierarchical inheritance
5. Hybrid Inheritance

There are three types of class inheritance: public, private and protected The protected access specifier is similar to private. Its only difference occurs in fact with inheritance. When a class inherits from another one, the members of the derived class can access the protected members inherited from the base class, but not its private members.

Difference between public, private and protected

- A member (either data member or member function) declared in a private section of a class can only be accessed by member functions and friends of that class.
- A member (either data member or member function) declared in a protected section of a class can only be accessed by member functions and friends of that class, and by member functions and friends of derived classes.
- A member (either data member or member function) declared in a public section of a class can be accessed by anyone.

What a derived class inherits

- Every data member defined in the parent class (although such members may not always be accessible in the derived class!)
- Every ordinary member function of the parent class (although such members always be accessible in the derived class!)

What a derived class doesn't inherit

- The base class's constructors and destructor
- The base class's assignment operator
- The base class's friends

What a derived class can add

- New data members New member functions
- New constructors and destructor

Facilities:

Linux Operating Systems, GCC, CodeBlocks

```
#include<iostream>
#include<string.h> using namespace std;
class publication //Abstract class
{public:
float price;
string title;
 void getpdata();
void displaypdata();
void setdata();
void displaybdata();
void displaytdata();
void publication::getpdata() //outside the class implementation
{cout<<"enter the title and price of book"<<endl;
cin>>title>>price;
}
Void publication::displaypdata()                    //outside the class implementation
{cout<<"title is="<<title<<endl;
cout<<"price="<<price<<endl;
}
class book: public publication //book inherit properties of publication
{
public:
int page_count;
void setdata() //method overriding
{
cout<<"Enter the page count"<<endl;
cin>>page_count;
try
{
if(page_count<0)
{
throw 1;
}
}
catch(inta)
{
title="0";
price=0.0;
page_count=0;
cout<<"page count should be grater than zero"<< endl;
}
}
Void displaybdata()
{
cout<<"page count is="<<page_count<<endl;
}
  };
Class tape:public publication            //tape inherit properties of publication
{
public:
float playing_time; book b;
void setdata()                              //method overriding
{
cout<<"enter the cassette playing time in min"<<endl;
cin>>playing_time;
try
{
if(playing_time<0)
{
throw 1;
}
}
catch(inta)
{
playing_time=0;
}
}
void displaytdata()
{
cout<<"play time in minute is="<<playing_time<<endl;
}
};
int main()
```

```
{book b;
tape t;
b.getpdata();           //class publication property.
b.setdata();           //class book property.
t.setdata();           //class tape property.
b.displaypdata();//class publication property.
b.displaybdata();//class book property.
 t.displaytdata(); //class tape property.
return 0;
}
```

Algorithm:

1. Start
2. Read information i.book title(string),price(float), pagecount,
3. Print all information from database.
4. Stop

Input:

Base class publication which consist data members name of book(type string) and price (type float)etc. two derived classes Book which contains data members page count (type int)and Tape class consist playing time in minutes (type float).

Output:

1. Display table containing all data Member
2. Insert a new entry

Conclusion: Thus , students are able to apply inheritance and use of exception handling.

QUESTIONS

1. What is Exception Handling with Example?
2. What is Abstract Class? explain in detail.
3. What is multi-level inheritance?

Experiment No. 4 (OOP)

Aim: Write a C++ program that creates an output file, writes information to it, closes the file and open it again as an input file and read the information from the file.

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool like G++

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

: 1) To learn and understand streams and files in object oriented paradigm.

2) To demonstrate file operations like create, open, read, write and close a file.

Outcomes:

1) Students will be able to learn and understand concepts of streams and files in C++.

2) Students will be able to demonstrate various operations like creating a new file, opening an existing file, reading from file, writing in file, closing file.

Theory:

we have been using the iostream standard library, which provides cin and cout methods for reading from standard input and writing to standard output respectively.

DataType	Description
ofstream	This data type represents the output file stream and is used to create files and to write information to files.
ifstream	This data type represents the input file stream and is used to read information from files.
fstream	This data type represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files.

To perform file processing in C++, header files <iostream> and <fstream> must be included in your C++ source file.

Opening a File:

A file must be opened before you can read from it or write to it. Either the ofstream or fstream object may be used to open a file for writing and ifstream object is used to open a file for reading purpose only.

Following is the standard syntax for open() function, which is a member of fstream, ifstream, and ofstream objects.

```
void open(const char *filename, ios::openmode mode);
#include <iostream>
 #include <fstream>
using namespace std;
int main ()
 {
 ofstream myfile;
myfile.open ("example.txt");
myfile << "Writing this to a file.\n";
myfile.close();
return 0;
}
```

Here, the first argument specifies the name and location of the file to be opened and the second argument of the open() member function defines the mode in which the file should be opened.

ModeFlag	Description
ios::app	Append mode. All output to that file to be appended to the end.
ios::ate	Open a file for output and move the read/write control to the end of the file.
ios::in	Open a file for reading.
ios::out	Open a file for writing.
ios::trunc	If the file already exists, its contents will be truncated before opening the file.

You can combine two or more of these values by ORing them together. For example if you want to open a file in write mode and want to truncate it in case it already exists, following will be the syntax:

```
    ofstream outfile;
    outfile.open("file.dat", ios::out | ios::trunc );
```

Similar way, you can open a file for reading and writing purpose as follows:

```
    fstream afile;
```

```
afile.open("file.dat", ios::out | ios::in );
```

Closing a File

When a C++ program terminates it automatically closes flushes all the streams, release all the allocated memory and close all the opened files. But it is always a good practice that a programmer should close all the opened files before program termination.

Following is the standard syntax for close() function, which is a member of fstream, ifstream, and ofstream objects.

```
void close();
```

Writing to a File:

While doing C++ programming, you write information to a file from your program using the stream insertion operator (<<) just as you use that operator to output information to the screen. The only difference is that you use an ofstream or fstream object instead of the cout object.

Reading from a File:

You read information from a file into your program using the stream extraction operator (>>) just as you use that operator to input information from the keyboard. The only difference is that you use an ifstream or fstream object instead of the cin object.

Algorithm:

1. Include required header file like iostream (for keyboard input and output), fstream (for file reading and writing), and cstdlib (for exit()).
2. In main(), read name of file from user.
3. Open file using ofstream for writing.
4. If file does not exist, new file will be created or existing file will be overwritten.
5. If any error occurs in creating or opening file, exit
6. Otherwise, open file.
7. Read string from user.
8. Check if given termination string is entered. (Here we will use ^D to end reading contents from keyboard).
9. If ^D is entered, stop reading from keyboard.
10. Otherwise, read line from keyboard and write in file using <<.
11. Close the file.
12. Open the same file for reading.
13. Check if file opened for reading or display error message and exit.

14. In while loop, start reading from file line by line and display output on the terminal.
15. If end of file is reached, exit loop.
16. End program.

Conclusion: Thus, students will be able to apply various operations on files.

Questions:-

1. Explain fstream, ifstream, and ofstream.
2. What are the modes in file handling?
3. Advantages and Disadvantages of File Handling?

Experiment No. 5 (OOP)

Aim: Write a function template selection Sort. Write a program that inputs, sorts and outputs an integer array and a float array.

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool like G++

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

1) To learn and understand templates.
2) To demonstrate function template for selection sort.

Outcomes:

1) Students will be able to learn and understand working and use of function template.
2) Students will be able to demonstrate function template for selection sort.

Theory:

Function templates

Function templates are special functions that can operate with generic types. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type. In C++ this can be achieved using template parameters. A template parameter is a special kind of parameter that can be used to pass a type as argument: just like regular function parameters can be used to pass values to a function, template parameters allow to pass also types to a function. These function templates can use these parameters as if they were any other regular type.

The format for declaring function templates with type parameters is:

template<classidentifier>function_declaration;

template<typenameidentifier>function_declaration

For example, to create a template

we could use:

```
template <class myType>
myType GetMax (myType a, myType b)
{
```

```
return (a>b?a:b);
}
```

Here we have created a template function with myType as its template parameter. This template parameter represents a type that has not yet been specified, but that can be used in the template function as if it were a regular type. As you can see, the function template GetMax returns the greater of two parameters of this still-undefined type.

Here in this program we have written a function for Selection sort and we have added a template tag before the function so that, the parameter will be of the data type Name. Everything is same except some variable data types. Take a look at the below program, you‟ll get a clear idea.

In the main function we are passing some predefined values into the Selection function by calling the function Selection(a,6) where a is the array containing integers and 6 is the size of array. After passing the values into the function we are displaying the sorted order. You can also rewrite the main function in a way that the user will enter the data and size of the array.

Selection sort

The algorithm divides the input list into two parts: the sublist of items already sorted, which is built up from left to right at the front (left) of the list, and the sublist of items remaining to be sorted that occupy the rest of the list. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sublist, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

```
int i, j;
for (j = 0; j < n-1; j++)
{
int iMin = j;
for ( i = j+1; i < n; i++) { if (a[i] <a[iMin])
{
iMin = i;
}
}
if(iMin != j)
{
swap (a[j], a[iMin]);
}
}
```

Selection sort is not difficult to analyze compared to other sorting algorithms since none of the loops depend on the data in the array. Selecting the lowest element requires scanning all n elements (this takes n − 1 comparisons) and then swapping it into the first position. Finding the next lowest element requires scanning there n−1 elements and soon, for (n−1)+(n−2)+...+ 2 + 1 = n(n - 1) / 2, Θ(n2) comparisons. Each of these scans requires one swap for n − 1 elements (the final element is already in place).

Algorithm Selection Sort:

```
Selection (A, N)
Step 1 − Set MIN to location 0
Step 2 − Search the minimum element in the list Step 3 − Swap with value at location MIN
Step 4 − Increment MIN to point to next element
Step 5 − Repeat until list is sorted
```

Algorithm:

1. Start
2. Read the numbers as integers or characters.
3. Sort them according to ascending order.
4. Print values as output.

Input:

Integer values, Float values.

Output:

Sorted order of integers as well as floats.

Conclusion: Thus, students will implement sorting of integer and float numbers.

Questions:

1. What is Function Template?
2. What is Sorting? Enlist Sorting Methods.
3. What is Container, Algorithm ,Iterator? Advantages of STL?

Experiment No. 6 (OOP)

Aim: Write C++ program using STL for sorting and searching user defined records such as personal records (Name, DOB, Telephone number etc) using vector container. OR

Write C++ program using STL for sorting and searching user defined records such as Item records (Item code, name, cost, quantity etc) using vector container.

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool like G++

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

To learn the concept STL, searching, sorting and vector container.

Outcomes:

Students will able to use STL for various applications.

Theory:

STL:

The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized.

A working knowledge of template classes is a prerequisite for working with STL.

STL has four components

- Algorithms
- Containers
- Functions
- Iterators

Algorithms

- The algorithm defines a collection of functions especially designed to be used on ranges of elements. They act on containers and provide means for various operations for the contents of the containers.

- Algorithm
- Sorting
- Searching

Important STL Algorithms

- Useful Array algorithms
- Partition Operations
- Numeric

Containers

Containers or container classes store objects and data. There are in total seven standard “first- class” container classes and three container adaptor classes and only seven header files that provide access to these containers or container adaptors.

Sequence Containers: implement data structures which can be accessed in a sequential manner.

- vector
- list
- deque
- arrays
- forward_list (Introduced inC++11)

Container Adaptors: provide a different interface for sequential containers.

- queue
- priority_queue
- stack

Associative Containers: implement sorted data structures that can be quickly searched(O(log complexity). set

- multiset
- map
- multimap

Unordered Associative Containers: implement unordered data structures that can be quickly searched

- unordered_set
- unordered_multiset
- unordered_map
- unordered_multimap

Functions

The STL includes classes that overload the function call operator. Instances of such classes are called function objects or functors. Functors allow the working of the associated function to be customized with the help of parameters to be passed.

Iterators

As the name suggests, iterators are used for working upon a sequence of values. They are the major feature that allow generality in STL.

Utility Library

- Defined in header <utility>.
- pair

Sorting:

It is one of the most basic functions applied to data. It means arranging the data in a particular fashion, which can be increasing or decreasing. There is a built in function in C++ STL by the name of sort(). This function internally uses IntroSort. In more details it is implemented using hybrid of QuickSort, Heap Sort and Insertion Sort.By default, it uses Quick Sort but if Quick Sort is doing unfair partitioning

and taking more than N*log N time, it switches to Heap Sort and when the array size becomes really small, it switches to Insertion Sort.

Searching:

It is a widely used searching algorithm that requires the array to be sorted before search is applied. The main idea behind this algorithm is to keep dividing the array in half (divide and conquer) until the element is found, or all the elements are exhausted.

It works by comparing the middle item of the array with our target, if it matches, it returns true otherwise if the middle term is greater than the target, the search is performed in the left sub-array. If the middle term is less than target, the search is performed in the right sub-array.The prototype for binary search is :

binary_search(startaddress, endaddress, valuetofind)

startaddress: the address of the first element of the array.

endaddress: the address of the last element of the array.

Valuetofind: the target value which we have to search for.

```
//Searching
#include <algorithm>
 #include <iostream>
using namespace std;
void show (int a[], int arraysize)
{
   for (int i = 0; i <arraysize ; ++i)
   cout<< a[i] << " ";
}
intmain()
{
Int a[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
Int asize = sizeof(a) / sizeof(a[0]);
cout<< "\n The array is : "; show(a, asize);
cout<< "\n\nLet's say we want to search for 2 in the array";
cout<< "\n So, we first sort the array";
sort(a, a + asize);
cout<< "\n\n The array after sorting is : ";
show(a, asize);
cout<< "\n\nNow, we do the binary search";
if(binary_search(a, a + 10, 2))
cout<< "\nElement found in the array"; else
cout<< "\nElement not found in the array";
cout<< "\n\nNow, say we want to search for 10";
if(binary_search(a, a + 10, 10))
cout<< "\nElement found in the array"; else
cout<< "\nElement not found in the array";
return0;
}
```

Output:

```
The array is : 1 5 8 9 0 6 7 3 4 2 0

Let's say we want to search for 2 in the array
```

Conclusion:

Hence, we have successfully studied the concept of STL (Standard Template Library) and how it makes many data structures easy. It briefs about the predefined functions of STL and their uses such a search() and sort()

Questions:

1. What is STL? What are four components of STL?
2. What is Sorting and Searching?
3. What is vector container?

Experiment No. 7 (OOP)

Aim: Write a program in C++ to use map associative container. The keys will be the names of states and the values will be the populations of the states. When the program runs, the user is prompted to type the name of a state. The program then looks in the map, using the state name as an index and returns the population of the state

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool like G++

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

To learn the concept of map associative container.

Outcomes:

Students will be able to use map associative container for various applications

Theory:

Map associative container:

Map associative containers are associative containers that store elements in a mapped fashion. Each key value and a mapped value. No two mapped values can have same key values.

This operator is used to reference the element present at position given inside the operator. It is similar to the at() function only difference is that the at() function throws an out-of-range exception when the position is not in the bounds of the size of map, while this operator causes undefined behaviour.

Syntax :

Mapname[key]

Parameters :

Key value mapped to the element to be fetched.

Returns :Direct reference to the element at the given key value.

Examples:

```
#include <map>
#include <iostream>
#include<string>
```

```
using namespace std;
int main()
{
// map declaration
string s;
map<string,int>mymap;
map<string,int>::iterator i;
// mapping integers to strings
mymap.insert(pair<string,int>("maharastra",12));
mymap.insert(pair<string,int>("gujrat",10));
mymap.insert(pair<string,int>("goa",3));
mymap.insert(pair<string,int>("bihar",9));
mymap.erase("goa");
for(i=mymap.begin();i!=mymap.end();i++)
{
cout<<(*i).first<<"  "<<(*i).second<<endl;
}
//cout<<"enter the sate name"<<endl;
//cin>>s;
cout<<"population of state="<<"is"<<"="<<mymap["goa"];
return 0;
}
```

Algorithm:

1. Start.
2. Give a header file to map associative container.
3. Insert states name so that we get values as population of that state.
4. Use population Map.insert().
5. Display the population of states.
6. End.

Input:

Information such as state name to map associative container.

Output:

```
Size of population Map: 5
Brasil: 193 million
China: 1339 million
India: 1187million
Indonesia: 234 million
Pakistan: 170 million
Indonesia's populations is 234 million
```

Conclusion:

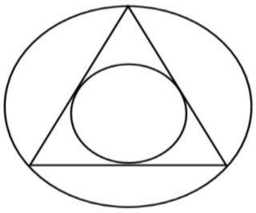
Hence, successfully studied the concept of map associative container

Questions:

1. What is an associative container in C++?
2. What is map in C++? How to do declare a map?
3. Explain Associative mapping with example?

Experiment No. 8 (CG)

Aim: Write C++ program to draw the pattern. Use DDA line and Bresenham‘s circle drawing algorithm. Apply the concept of encapsulation.



Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool like G++/GCC

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

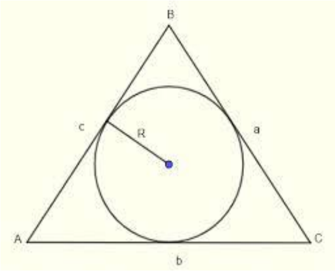
To understand and implement DDA Line Drawing algorithm and Bresenham’s Circle drawing algorithm.

Outcomes:

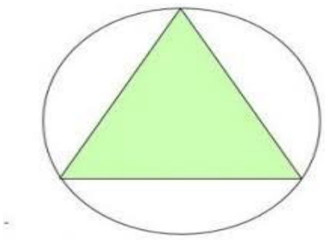
Analyze and **apply** computer graphics algorithms for line-circle drawing with the help of object oriented programming concepts

Theory:

Inscribed Circle: When the circle is drawn inside any shape is called inscribed circle.



Circumscribed Circle: Drawing any shape inside circle is called circumscribed circle. When a circle is placed outside a polygon and each vertex of the polygon lies on the circle



To implement pattern of Inscribed and circumscribed circle in triangle, following algorithms are used

1. DDA Line drawing Algorithm
2. Bresnham's Circle drawing algorithm

DDA Line Drawing Algorithm:

Line is a basic element in graphics. To draw a line, you need two end points between which you can draw a line. Digital Differential Analyzer (DDA) line drawing algorithm is the simplest line drawing algorithm in computer graphics. It works on incremental method. It plots the points from starting point of line to end point of line by incrementing in X and Y direction in each iteration.

DDA line drawing algorithm works as follows:

Algorithm-1: DDA Line drawing algorithm

Step 1: Get coordinates of both the end points (X_1, Y_1) and (X_2, Y_2) from user.

Step 2: Calculate the difference between two end points in X and Y direction.

$dx = X_2 – X_1$; $dy = Y_2 – Y_1$;

Step 3: Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If dx > dy, then you need more steps in x coordinate; otherwise in y coordinate.

```
if (absolute(dx) > absolute(dy))
    Steps = absolute(dx);
else
    Steps = absolute(dy);
```

Step 4: Calculate the increment in x coordinate and y coordinate.

```
                Xincrement = dx / (float) steps;

                Yincrement = dy / (float) steps;
```

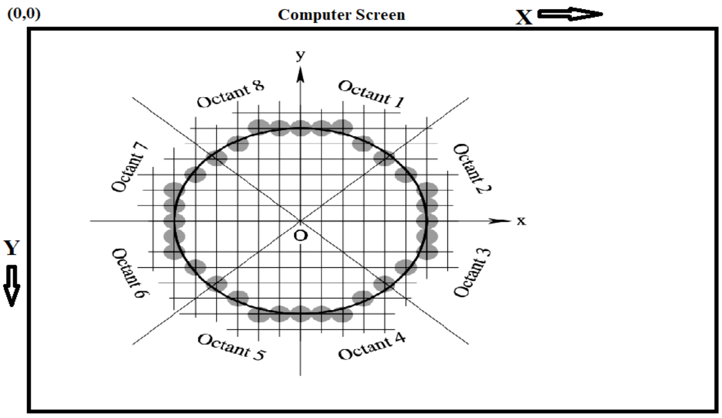
Step 5: Plot the pixels by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

```
for(int i=0; i < Steps; i++)

{

        $X_1 = X_1$ + Xincrement;

        $Y_1 = Y_1$ + Yincrement;

        putpixel(Round($X_1$), Round($Y_1$), ColorName);

}
```

Bresenham’s Circle Drawing Algorithm:

Circle is an eight-way symmetric figure. The shape of circle is the same in all quadrants. In each quadrant, there are two octants. If the calculation of the point of one octant is done, then the other seven points can be calculated easily by using the concept of eight-way symmetry.

Bresenham’s Circle Drawing Algorithm is a circle drawing algorithm that selects the nearest pixel position to complete the arc. The unique part of this algorithm is that is uses only integer arithmetic which makes it significantly faster than other algorithms using floating point arithmetic.



Algorithm-2: Bresenham’s circle drawing algorithm

Step 1: Read the x and y coordinates of center: (centx, centy)

Step 2: Read the radius of circle: (r)

Step 3: Initialize, x = 0; y = r;

Step 4: Initialize decision parameter: p = 3 – (2 * r)

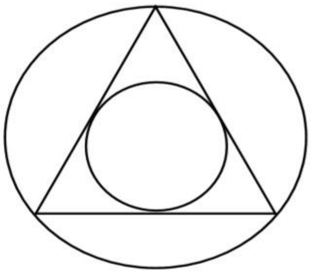
Step 5:

```
do {
        putpixel(centx+x, centy-y, ColorName);
        if (p<0)
          p = p+(4*x)+6;
        else
          {
            p=p+[4*(x-y)]+10;
              y=y-1;
            }
          x=x+1;
    } while(x<y)
```

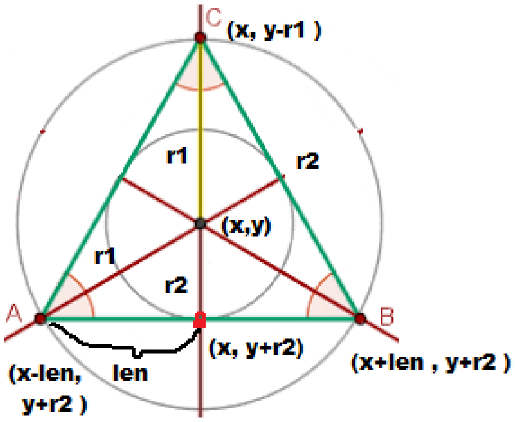
Here in step 5, putpixel() function is used which will print Octant-1 of the circle with radius r after the completion of all the iterations of do-while loop. Because of the eight-way symmetry property of circle, we can draw other octants of the circle using following putpixel() function

Octant 1: putpixel(centx+x, centy-y, ColorName);
Octant 2: putpixel(centx+y, centy-x, olorName);
Octant 3: putpixel(centx+y, centy+x, ColorName);
Octant 4: putpixel(centx+x, centy+y, ColorName);
Octant 5: putpixel(centx-x, centy+y, ColorName);
Octant 6: putpixel(centx-y, centy+x, ColorName);
Octant 7: putpixel(centx-y, centy-x, ColorName);
Octant 8: putpixel(centx-x, centy-y, ColorName);

Drawing Pattern using lines and circles:



This pattern is made up of one equilateral triangle and two concentric circles. To draw the triangle, we require coordinates of 3 vertices forming an equilateral triangle. To draw two concentric circles, coordinates of common center and radius of both the circles is required. Take coordinates of circle and radius of one of the circle from user. Then using the properties of an equilateral triangle, find all 3 vertices of equilateral triangle and radius of other circle. As shown in example, read common center coordinate (x, y) of circumscribed and inscribed circle. Input radius r1 and r2of circumscribed and inscribed circle respectively. Calculate coordinates of equilateral triangle using following formula.



C=(x1,y1) =(x,y-r1)

A=(x2,y2)=(x-len, y+r2)

B=(x3,y3)=(x+len,y+r2)

Once all these parameters are calculated, call DDA line drawing and Bresenham's circle drawing algorithms by passing appropriate parameters to get required pattern.

Main Algorithm:

1. Start
2. Input center coordinates of circle
3. Input radius r1 and r2of circumscribed and inscribed circle respectively
4. Find out coordinates of equilateral triangle.
5. Draw inscribed circle using Bresenham's circle drawing algorithm (Algorithm-2).

6. Draw circumscribed circle using Bresenham’s circle drawing algorithm (Algorithm-2).
7. Draw equilateral triangle using DDA line generation algorithm (Algorithm-1).
8. Stop

Conclusion: Thus implemented program to draw pattern using DDA line drawing algorithm and Bresenham’s circle drawing algorithm.

Questions:

1. Explain the derivation of decision parameters in Bresenham’s circle drawing algorithm.
2. What is slope? What are different types of slopes
3. What is difference between DDA and Bresenham’s line drawing algorithm?

Experiment No. 9 (CG)

Aim: Write C++ program to draw a concave polygon and fill it with desired color using scan fill algorithm. Apply the concept of inheritance.

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool like G++/GCC

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

To create concave polygon and fill it with a colour using scan line fill algorithm.

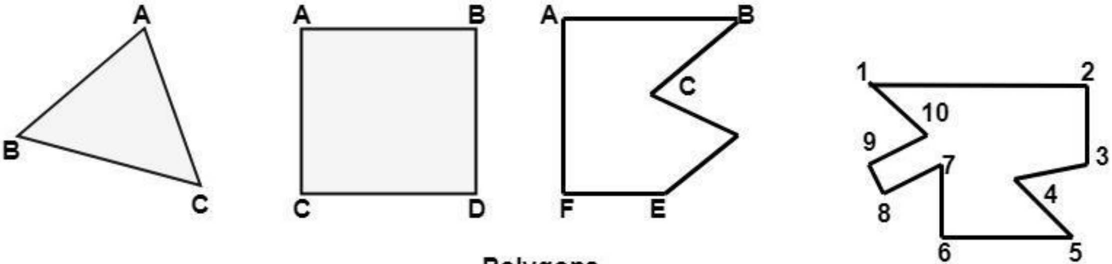
Outcomes:

Student will understand and implement filling of polygon using Scan line polygon filling algorithm.

Theory:

Polygon:

A polygon is a closed planar path composed of a finite number of sequential line segments. A polygon is a two-dimensional shape formed with more than three straight lines. When starting point and terminal point is same then it is called polygon.



Polygons

Types of Polygons

1. Concave
2. Convex
3. Complex

A convex polygon is a simple polygon whose interior is a convex set. In a convex polygon, all interior angles are less than 180 degrees.

The following properties of a simple polygon are all equivalent to convexity:

- Every internal angle is less than or equal to 180 degrees.

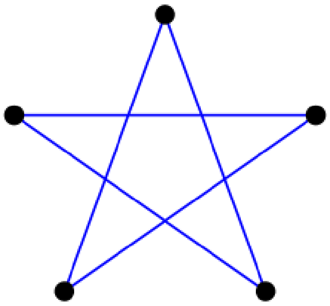
- Every line segment between two vertices remains inside or on the boundary of the polygon.

Convex Polygons: In a convex polygon, any line segment joining any two inside points lies inside the polygon. A straight line drawn through a convex polygon crosses at most two sides.

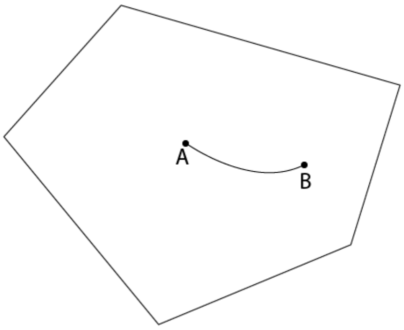
A concave polygon will always have an interior angle greater than 180 degrees. It is possible to cut

a concave polygon into a set of convex polygons. You can draw at least one straight line through a concave polygon that crosses more than two sides.

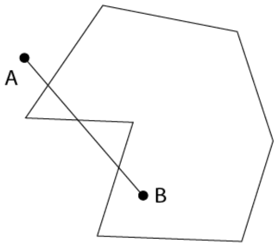
Complex polygon is a polygon whose sides cross over each other one or more times.



Complex Polygon



Convex polygon

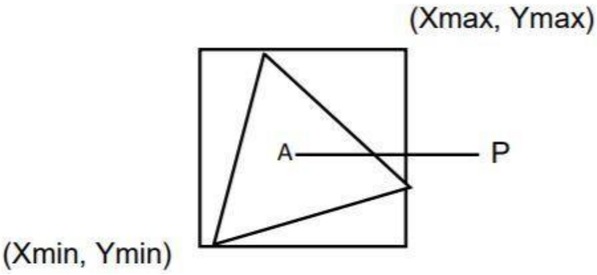


Concave polygon

Inside outside test (Even- Odd Test):

We assume that the vertex list for the polygon is already stored and proceed as follows.

1. Draw any point outside the range Xmin and Xmax and Ymin and Ymax. Draw a scan line through P up to a point A under study



2. If this scan line

i) Does not pass through any of the vertices then its contribution is equal to the number of times it intersects the edges of the polygon. Say C if

a) C is odd then A lies inside the polygon.

b) C is even then it lies outside the polygon.

ii) If it passes through any of the vertices then the contribution of this intersection say V is,

a) Taken as 2 or even. If the other points of the two edges lie on one side of the scan line.

b) Taken as 1 if the other end points of the 2 edges lie on the opposite sides of the scan- line.

c) Here will be total contribution is C + V.

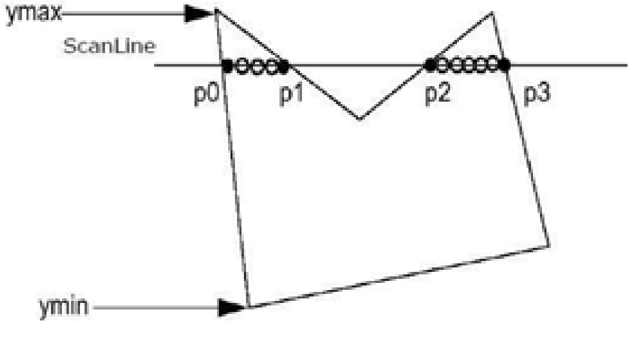
Polygon Filling:

For filling polygons with particular colors, you need to determine the pixels falling on the border of the polygon and those which fall inside the polygon.

Scan fill algorithm:

A scan-line fill of a region is performed by first determining the intersection positions of the boundaries of the fill region with the screen scan lines Then the fill colors are applied to each section of a scan line that lies within the interior of the fill region The scan-line fill algorithm identifies the same interior regions as the odd-even rule.

It is an image space algorithm. It processes one line at a time rather than one pixel at a time. It uses the concept area of coherence. This algorithm records edge list, active edge list. So accurate bookkeeping is necessary. The edge list or edge table contains the coordinate of two endpoints. Active Edge List (AEL) contain edges a given scan line intersects during its sweep. The active edge list (AEL) should be sorted in increasing order of x. The AEL is dynamic, growing and shrinking.



Algorithm

Step1: Start algorithm

Step2: Initialize the desired data structure

1. Create a polygon table having color, edge pointers, coefficients
2. Establish edge table contains information regarding, the endpoint of edges, pointer to polygon, inverse slope.
3. Create Active edge list. This will be sorted in increasing order of x.
4. Create a flag F. It will have two values either on or off.

Step3: Perform the following steps for all scan lines

1. Enter values in Active edge list (AEL) in sorted order using y as value
2. Scan until the flag, i.e. F is on using a background color
3. When one polygon flag is on, and this is for surface S1enter color intensity as I1into refresh buffer
4. When two or image surface flag are on, sort the surfaces according to depth and use intensity value Sn for the nth surface. This surface will have least z depth value
5. Use the concept of coherence for remaining planes.

Step4: Stop Algorithm

Conclusion: Hence concave polygon is implemented and filled with scan line algorithm

Questions:

1. Which are the different approaches to fill a polygon?
2. What are advantages and drawbacks of scan line polygon fill algorithm?
3. Explain flood fill algorithm .

Experiment No. 10 (CG)

Aim: Write C++ program to implement Cohen Southerland line clipping algorithm..

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool like G++/GCC

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

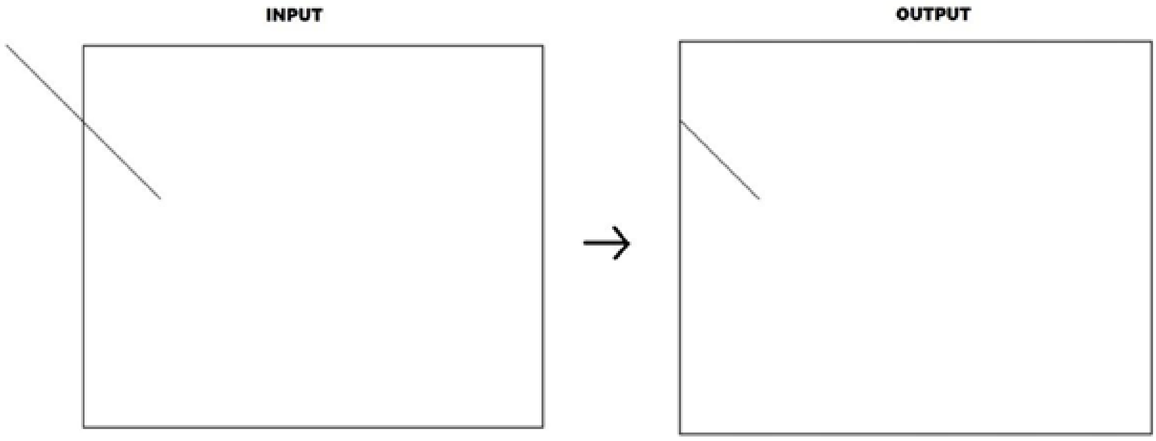
To clip lines outside window using Cohen Sutherland line clipping algorithm.

Outcomes:

Students will learn and implement Cohen Sutherland line clipping algorithm

Theory:

Cohen Sutherland Algorithm is a **line clipping algorithm** that cuts lines to portions which are within a rectangular area. It eliminates the lines from a given set of lines and rectangle area of interest (view port) which belongs outside the area of interest and clips those lines which are partially inside the area of interest.

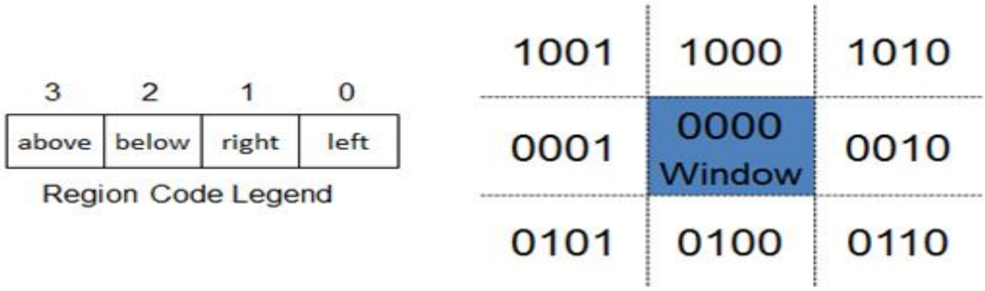


Cohen Sutherland Line Clipping Algorithm

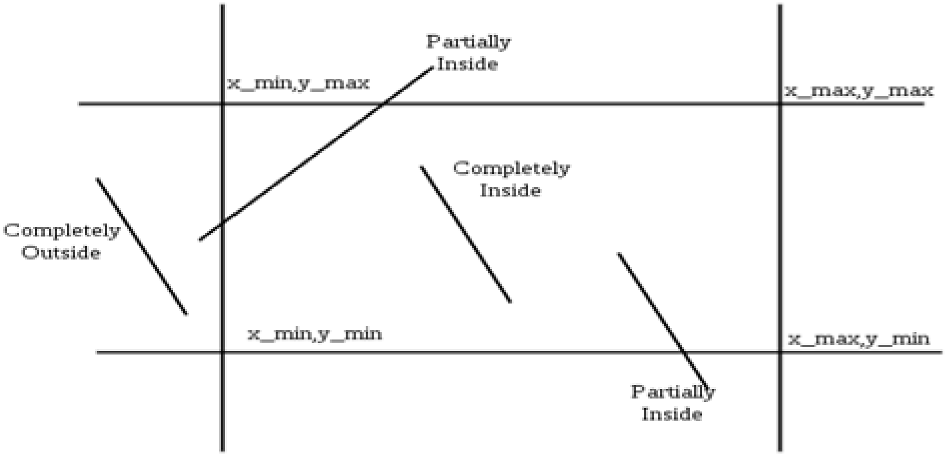
Algorithm

The algorithm divides **a two-dimensional space** into **9 regions** (eight outside regions and one inside region) and then efficiently determines the lines and portions of lines that are visible in the central region of interest (the viewport).

Following image illustrates the 9 regions: you seen each region is denoted by a 4 bit code like 0101 for the bottom right region



Four Bit code is calculated by comparing extreme end point of given line (x, y) by four co-ordinates x_min, x_max, y_max, y_min which are the coordinates of the area of interest (0000).



Calculate the four bit code as follows:

- Set First Bit if 1 Points lies to left of window (**x < x_min**)
- Set Second Bit if 1 Points lies to right of window (**x > x_max**)
- Set Third Bit if 1 Points lies to left of window (**y < y_min**)
- Set Forth Bit if 1 Points lies to right of window (y **> y_max**)

The more efficient Cohen-Sutherland Algorithm performs initial tests on a line to determine whether intersection calculations can be avoided.

Pseudocode

```
Step 1 : Assign a region code for two endpoints of given line
Step 2 : If both endpoints have a region code 0000 then given line is completely inside
and we will keep this line.
Step 3: If step 2 fails, perform the logical AND operation for both region codes.
Step 3.1: If the result is not 0000, then given line is completely outside.
        Step 3.2 : Else line is partially inside.
                Step 3.2.a : Choose an endpoint of the line that is outside the
                             given rectangle.
                Step 3.2.b : Find the intersection point of the rectangular
                             boundary(based on region code)
                Step 3.2.c : Replace endpoint with the intersection point and
                             upgrade the region code.
                Step 3.2.d : Repeat step 2 until we find a clipped line either
                             trivially accepted or rejected.
Step 4: Repeat step 1 for all lines.
```

Conclusion: Thus Cohen Sutherland line clipping algorithm is implemented

Questions:

1. What is the limitation of Cohen Sutherland Line Clipping algorithm?
2. What are the advantages of Cohen Sutherland Line Clipping?
3. What is windowing?

Experiment No. 11 (CG)

Aim: Write C++ program to draw 2-D object and perform following basic transformations a)Scaling b) Translation c) Rotation. Apply the concept of operator overloading.

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool like G++/GCC

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

To apply mathematics for performing various types of transformations on objects.

Outcomes:

Students are able to apply translation, scaling and rotation on 2D objects.

Theory:

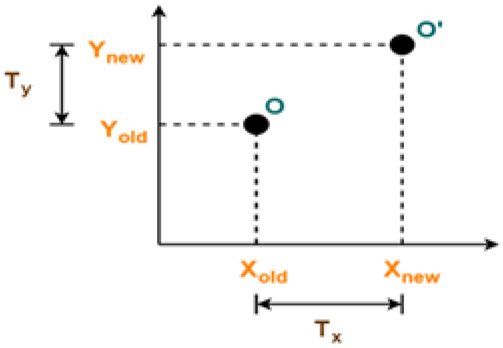
Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, reflection etc. When a

Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, reflection etc. When a transformation takes place on a 2D plane, it is called 2D transformation. Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation. Translation, Scaling and Rotation are basic transformations.

1) Translation:

A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate or translation vector (T_x, T_y) to the original coordinates. Consider Initial coordinates of the object O = (X_{old}, Y_{old})

- New coordinates of the object O after translation = (X_{new}, Y_{new})
- Translation vector or Shift vector = (T_x, T_y)



This translation is achieved by adding the translation coordinates to the old coordinates of the object as-

$X_{new} = X_{old} + T_x$ (This denotes translation towards X axis)

$Y_{new} = Y_{old} + T_y$ (This denotes translation towards Y axis)

In Matrix form, the above translation equations may be represented as-

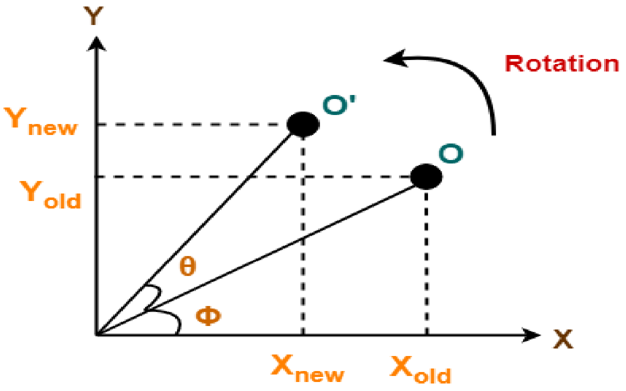
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

Translation Matrix

2) Rotation:

In rotation, we rotate the object at particular angle θ (theta) from its original position. Consider

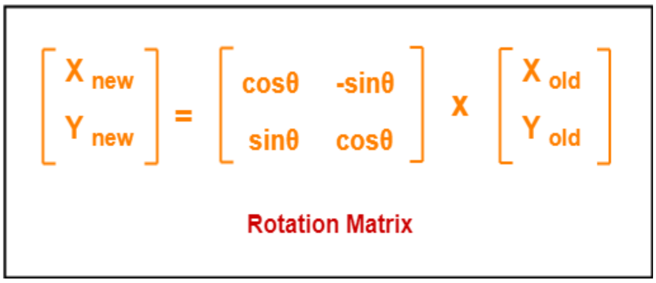
- Initial coordinates of the object O = (X_{old}, Y_{old})
- Initial angle of the object O with respect to origin = Φ
- Rotation angle = θ
- New coordinates of the object O after rotation = (X_{new}, Y_{new})



This anti-clockwise rotation is achieved by using the following rotation equations-

$X_{new} = X_{old} \text{ x } \cos\theta – Y_{old} \text{ x } \sin\theta$ $Y_{new} = X_{old} \text{ x } \sin\theta + Y_{old} \text{ x } \cos\theta$

In Matrix form, the above rotation equations may be represented as-



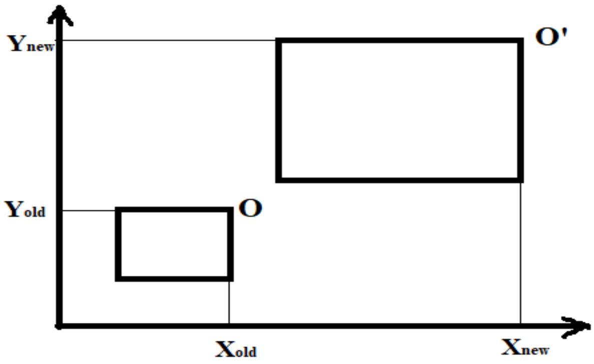
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \text{x} \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Rotation Matrix

3) Scaling:

Scaling transformation is used to change the size of an object. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor (S_x, S_y). If scaling factor > 1, then the object size is increased. If scaling factor < 1, then the object size is reduced. Consider

- Initial coordinates of the object O = (X_{old}, Y_{old})
- Scaling factor for X-axis = S_x
- Scaling factor for Y-axis = S_y
- New coordinates of the object O after scaling = (X_{new}, Y_{new})



This scaling is achieved by using the following scaling equations- $X_{new} = X_{old} \text{ x } S_x$

$Y_{new} = Y_{old} \text{ x } S_y$

In Matrix form, the above scaling equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Scaling Matrix

Homogeneous Coordinates:

Matrix multiplication is easier to implement in hardware and software as compared to matrix addition. Hence we want to replace matrix addition by multiplication while performing transformation operations. So the solution is **homogeneous coordinates,** which allows us t o express all transformations (including translation) as matrix multiplications.

To obtain homogeneous coordinates we have to represent transformation matrices in 3x3 matrices instead of 2x2. For this we add dummy coordinate. Each 2 dimensional position (x,y) can be represented by homogeneous coordinate as (x,y,1).

Translation Matrix (Homogeneous Coordinates representation)

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Rotation Matrix (Homogeneous Coordinates representation)

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Scaling Matrix (Homogeneous Coordinates representation)

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Applying transformations on equilateral triangle:

Consider that coordinates of vertices of equilateral triangle are (X1,Y1), (X2,Y2) and (X3,Y3). After applying basic transformations, we will get corresponding coordinates as (X1’,Y1’), (X2’,Y2’) and (X3’,Y3’) respectively. Following multiplication will give the translation on this equilateral triangle:

$$\begin{bmatrix} X1' & X2' & X3' \\ Y1' & Y2' & Y3' \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X1 & X2 & X3 \\ Y1 & Y2 & Y3 \\ 1 & 1 & 1 \end{bmatrix}$$

Similarly we can apply rotation and scaling on equilateral triangle.

Applying transformations on rhombus:

Consider that coordinates of vertices of rhombus are (X1,Y1), (X2,Y2), (X3,Y3) and (X4,Y4) applying basic transformations, we will get corresponding coordinates as (X1’,Y1’), (X2’,Y2’), (X3’,Y3’) and (X4’,Y4’) respectively. Following multiplication will give the translation on this rhombus:

$$\begin{bmatrix} X1' & X2' & X3' & X4' \\ Y1' & Y2' & Y3' & Y4' \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X1 & X2 & X3 & X4 \\ Y1 & Y2 & Y3 & Y4 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Similarly we can apply rotation and scaling on rhombus.

Conclusion: Hence implemented various transformation algorithms.

Questions:

1. How to rotate any 2-D object about an arbitrary point? Explain in brief.
2. Explain shearing transformation.
3. Give difference between parallel and perspective projections?

Experiment No. 12 (CG)

Aim: Write C++ program to generate Hilbert curve using concept of fractals.

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool like G++/GCC

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

To understand and implement Hilbert curve using fractals.

Outcomes:

Students are able to understand mathematics behind generation of Hilbert curve using fractals.

Theory:

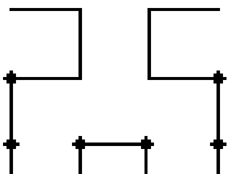
The Hilbert curve

The Hilbert curve is a space filling curve that visits every point in a square grid with a size of 2×2, 4×4, 8×8, 16×16, or any other power of 2. It was first described by David Hilbert in 1892. Applications of the Hilbert curve are in image processing: especially image compression and dithering. It has advantages in those operations where the coherence between neighbouring pixels is important. The Hilbert curve is also a special version of a quadtree; any image processing function that benefits from the use of quadtrees may also use a Hilbert curve.

Cups and joins

The basic elements of the Hilbert curves are what I call "cups" (a square with one open side) and "joins" (a vector that joins two cups). The "open" side of a cup can be top, bottom, left or right.

In addition, every cup has two end-points, and each of these can be the "entry" point or the "exit" point. So, there are eight possible varieties of cups. In practice, a Hilbert curve



uses only four types of cups. In a similar vein, a join has a direction: up, down, left or right.

A first order Hilbert curve is just a single cup (see the figure on the left). fills a 2×2 space. The second order Hilbert curve replaces that cup by four (smaller) cups, which are linked together by three joins (see the figure on the right; the link between a cup and a join has been marked with a fat dot in the figure). Every next order repeats the process or replacing each cup by four smaller cups and three joins.

Cup subdivision rules

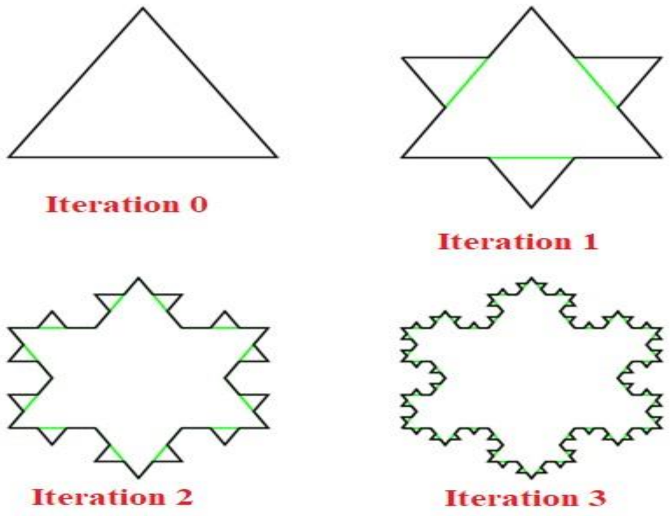
⊔ ⇒ ⊐↓⊔→⊔↑⊏
⊐ ⇒ ⊔→⊐↓⊐←⊓
⊓ ⇒ ⊏↑⊓←⊓↓⊐
⊏ ⇒ ⊓←⊏↑⊏→⊔

The function presented below (in the "C" language) computes the Hilbert curve. Note that the curve is symmetrical around the vertical axis. It would therefore be sufficient to draw half of the Hilbert curve.

Snowflake curve:

Snowflake curve is drawn using koch curve iterations. In koch curve, we just have a single line in the starting iteration and in snowflake curve, we have an equilateral triangle.

Draw an equilateral triangle and repeat the steps of Koch curve generation for all three segments of an equilateral triangle.



Conclusion: Thus implemented Hilbert curve using fractals.

Questions:

1. What is the importance of curves and fractals in computer graphics?
2. What are applications of curves and fractals?
3. What is blending function?

Experiment No. 13 (CG)

Aim: Write OpenGL program to draw Sun Rise and Sunset.

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool like OPENGL

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

To learn graphics functionality using OpenGL.

Outcomes:

Analyze and **apply OpeGL** computer graphics functions to implement Sun Rise and Sun Set.

Theory:

OpenGL Basics:

Open Graphics Library (OpenGL) is a cross-language (language independent), cross-platform (platform independent) API for rendering 2D and 3D Vector Graphics (use of polygons to represent image). OpenGL is a low-level, widely supported modeling and rendering software package, available across all platforms. It can be used in a range of graphics applications, such as games, CAD design, or modeling. OpenGL API is designed mostly in hardware.

Design: This API is defined as a set of functions which may be called by the client program. Although functions are similar to those of C language but it is language independent.

Development: It is an evolving API and Khronos Group regularly releases its new version having some extended feature compare to previous one. GPU vendors may also provide some additional functionality in the form of extension.

Associated Libraries: The earliest version is released with a companion library called OpenGL utility library. But since OpenGL is quite a complex process. So in order to make it easier other library such as OpenGL Utility Toolkit is added which is later superseded by freeglut. Later included library were GLEE, GLEW and glbinding.

Implementation: Mesa 3D is an open source implementation of OpenGL. It can do pure software rendering and it may also use hardware acceleration on BSD, Linux, and other platforms by taking advantage of Direct Rendering Infrastructure.

Installation of OpenGL on Ubuntu

We need the following sets of libraries in programming OpenGL:

1. **Core OpenGL (GL):** consists of hundreds of functions, which begin with a prefix "gl" (e.g., glColor, glVertex, glTranslate, glRotate). The Core OpenGL models an object via a set of geometric primitives, such as point, line, and polygon.
2. **OpenGL Utility Library (GLU):** built on-top of the core OpenGL to provide important utilities and more building models (such as qradric surfaces). GLU functions start with a prefix "glu" (e.g., gluLookAt, gluPerspective)
3. **OpenGL Utilities Toolkit (GLUT):** provides support to interact with the Operating System (such as creating a window, handling key and mouse inputs); and more building models (such as sphere and torus). GLUT is designed for constructing small to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits. GLUT is simple, easy, and small. Alternative of GLUT includes SDL.
4. **OpenGL Extension Wrangler Library (GLEW):** "GLEW is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform. For installing OpenGL on ubuntu, just execute the following command (like installing any other thing) in terminal: sudo apt-get install freeglut3-dev

For working on Ubuntu operating system:

- gcc filename.c -lGL -lGLU -lglut

where filename.c is the name of the file with which this program is saved.

Prerequisites for OpenGL

Since OpenGL is a graphics API and not a platform of its own, it requires a language to operate in and the language of choice is C++.

OpenGL order of operations

- Construct shapes (geometric descriptions of objects – vertices, edges, polygons etc.)
- Use OpenGL to
  - Arrange shape in 3D (using transformations)
  - Select your vantage point (and perhaps lights)
  - Calculate color and texture properties of each object

o Convert shapes into pixels on screen

OpenGL Syntax

□ All functions have the form: gl*

o *glVertex3f()* – 3 means that this function take three arguments, and f means that the type of those arguments is float.

o *glVertex2i()* – 2 means that this function take two arguments, and i means that the type of those arguments is integer

□ All variable types have the form: GL*

o In OpenGL program it is better to use OpenGL variable types (portability)

Glfloat instead of *float*

Glint instead of *int*

OpenGL primitives

Drawing two lines

glBegin(GL_LINES);

glVertex3f(_,_,_); // start point of line 1

glVertex3f(_,_,_); // end point of line 1

glVertex3f(_,_,_); // start point of line 2

glVertex3f(_,_,_); // end point of line 2 glEnd();

We can replace GL_*LINES* with *GL_POINTS, GL_LINELOOP, GL_POLYGON* etc.

OpenGL states

□ On/off (e.g., depth buffer test)

o *glEnable(GLenum)*

o *glDisable(GLenum)*

o Examples:

glEnable(GL_DEPTH_TEST);

glDisable(GL_LIGHTING);

□ Mode States

o Once the mode is set the effect stays until reset

o Examples:

glShadeModel(GL_FLAT) or glShadeModel(GL_SMOOTH)

glLightModel(…) etc.

Drawing in 3D

Depth buffer (or z-buffer) allows scene to remove hidden surfaces. Use *glEnable(GL_DEPTH_TEST)* to enable it.

glPolygonMode(Face, Mode)

Face: GL_FRONT, GL_BACK, GL_FRONT_AND_BACKMode: GL_LINE, GL_POINT, GL_FILL

o glCullFace(Mode)

□ Mode: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK

o glFrontFace(Vertex_Ordering)

o Vertex Ordering: GL_CW or GL_CCW

Viewing transformation

□ glMatrixMode (Mode)

o Mode: GL_MODELVIEW, GL_PROJECTION, or GL_TEXTURE

□ glLoadIdentity()

□ glTranslate3f(x,y,z)

□ glRotate3f(angle,x,y,z)

□ glScale3f(x,y,z)

OpenGL provides a consistent interface to the underlying graphics hardware. This abstraction allows a single program to run a different graphics hardware easily. A program written with OpenGL can even be run in software (slowly) on machines with no graphics acceleration. OpenGL function names always begin with *gl*, such as *glClear(),* and they may end with characters that indicate the types of the parameters, for example *glColor3f(GLfloat red, GLfloat green, GLfloat blue)* takes three floating-point color parameters and *glColor4dv(const GLdouble *v)* takes a pointer to an array that contains 4 double-precision floating-point values. OpenGL constants begin with *GL*, such as *GL DEPTH*. OpenGL also uses special names for types that are passed to its functions, such as *GLfloat* or *GLint*, the corresponding C types are compatible, that i*float* and *int* respectively.

GLU is the OpenGL utility library. It contains useful functions at a higher level than those provided by OpenGL, for example, to draw complex shapes or set up cameras. All GLU functions are written on top of OpenGL. Like OpenGL, GLU function names begin with glu, and constants begin with GLU.GLUT, the OpenGL Utility Toolkit, provides a system for setting up callbacks for interacting with the user and functions for dealing with the windowing system. This abstraction allows a program to run on different operating systems with only a recompile. Glut follows the convention of prepending function names with glut and constants with GLUT.

Writing an OpenGL Program with GLUT

An OpenGL program using the three libraries listed above must include the appropriate headers. This requires the following three lines:

#include <GL/gl.h>

#include <GL/glu.h>

#include <GL/glut.h>

Before OpenGL rendering calls can be made, some initialization has to be done. With GLUT, this consists of initializing the GLUT library, initializing the display mode, creating the window, and setting up callback functions. The following lines initialize a full color, double buffered display: *glutInit(&argc, argv);*

glutInitDisplayMode(GLUT_DOUBLE | *GLUT_RGB);*

Double buffering means that there are two buffers, a front buffer and a back buffer. The front buffer is displayed to the user, while the back buffer is used for rendering operations. This prevents flickering that would occur if we rendered directly to the front buffer.

Next, a window is created with GLUT that will contain the viewport which displays the OpenGL front buffer with the following three lines:

glutInitWindowPosition(px, py); glutInitWindowSize(sx, sy); glutCreateWindow(name);

To register callback functions, we simply pass the name of the function that handles the event to the appropriate GLUT function.

glutReshapeFunc(reshape);

glutDisplayFunc(display);

Here, the functions should have the following prototypes:

void reshape(int width, int height);

void display();

In this example, when the user resizes the window, reshape is called by GLUT, and when the display needs to be refreshed, the display function is called. For animation, an idle event handler that takes no arguments can be created to call the display function to constantly redraw the scene with *glutIdleFunc*. Once all the callbacks have been set up, a call to *glutMainLoop* allows the program to run. In the display function, typically the image buffer is cleared, primitives are rendered to it, and the results are presented to the user. The following line clears the image buffer, setting each pixel color to the clear color, which can be configured to be any color:

glClear(GL_COLOR_BUFFER_BIT);

The next line sets the current rendering color to blue. OpenGL behaves like a state machine, so certain state such as the rendering color is saved by OpenGL and used automatically later as it is needed.

glColor3f(0.0f, 0.0f, 1.0f);

To render a primitive, such as a point, line, or polygon, OpenGL requires that a call to *glBegin* is made to specify the type of primitive being rendered.

glBegin(GL_LINES);

Only a subset of OpenGL commands is available after a call to *glBegin*. The main command that is used is *glVertex*, which specifies a vertex position. In GL LINES mode, each pair of vertices define endpoints of a line segment. In this case, a line would be drawn from the point at (x0, y0) to (x1, y1).

glVertex2f(x0, y0); glVertex2f(x1, y1);

A call to glEnd completes rendering of the current primitive. *glEnd()*; Finally, the back buffer needs to be swapped to the front buffer that the user will see, which GLUT can handle for us: *glutSwapBuffers();*

Conclusion: Thus studied OpenGL functions and implemented transformation program.

Questions:

1. What are the advantages of Open GL over other API's?
2. Explain rendering pipeline with reference to OpenGL.
3. Explain events and callback in OpenGL.

Experiment No. 14

Aim: Write a C++ program to implement bouncing ball using sine wave form. Apply the concept of polymorphism.

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

To learn and apply polymorphism to create bouncing ball animation using sine wave form..

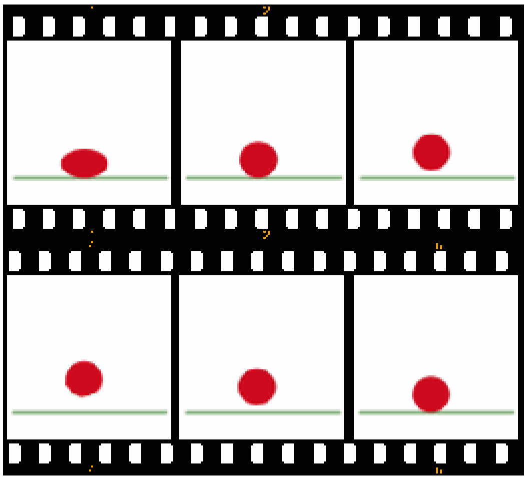
Outcomes:

Students are able to create animation of bouncing ball using graphics functions.

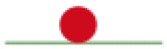
Theory:

What is animation?

Animation is the process of designing, drawing, making layouts and preparation of photographic sequences which are integrated in the multimedia and gaming products. Animation involves the exploitation and management of still images to generate the illusion of movement. Animation is the process of creating a continuous motion and shape change[Note 1] illusion by means of the rapid display of a sequence of static images that minimally differ from each other. The illusion—as in motion pictures in general—is thought to rely on the phi phenomenon.



The bouncing ball animation (below) consists of these six frames.



This animation moves at 10 frames per second.

Animations can be recorded on either analogue media, such as a flip book, motion picture film, video tape, or on digital media, including formats such as animated GIF, Flash animation or digital video. To display it, a digital camera, computer, or projector are used.

Animation creation methods include the traditional animation creation method and those involving stop motion animation of two and three-dimensional objects, such as paper cutouts, puppets and clay figures. Images are displayed in a rapid succession, usually 24, 25, or 30 frames per second. How to move an element to left, right, up and down using arrow keys?

To detect which arrow key is pressed, you can use ncurses.h header file. Arrow key's code is defined as: KEY_UP, KEY_DOWN, KEY_LEFT, KEY_RIGHT.

```
#include<ncurses.h> int main()
{
int ch;
/* Curses Initialisations */ initscr();
raw();
keypad(stdscr, TRUE); noecho();
printw("Welcome - Press # to Exit\n"); while((ch = getch()) != '#')
{
switch(ch)
{
case KEY_UP: printw("\nUp Arrow");
break;
case KEY_DOWN: printw("\nDown Arrow"); break;
case KEY_LEFT: printw("\nLeft Arrow"); break;
case KEY_RIGHT: printw("\nRight Arrow"); break;
default:
{
   printw("\nThe pressed key is ");
   attron(A_BOLD);

   printw("%c", ch); attroff(A_BOLD);

}

}

}
printw("\n\nBye Now!\n"); refresh();
   getch();

   endwin();

   return 0;

   }
```

Draw a sine wave using c++

```
    #include <math.h>
   #include <graphics.h>
    #include <iostream>
   int main() {

          int gd = DETECT, gm;
          int angle = 0; double
          x, y;

          initgraph(&gd, &gm, NULL);

          line(0, getmaxy() / 2, getmaxx(), getmaxy() / 2);

            /* generate a sine wave */

           for(x = 0; x < getmaxx(); x+=3) {

            /* calculate y value given x */ y = 50*sin(angle*3.141/180);

            y = getmaxy()/2 - y;

            /* color a pixel at the given position */ putpixel(x, y,
            15);

            delay(100);  /* increment angle */
            angle+=5;

     }
     closegraph();/* deallocate memory allocated for graphics screen */
     return 0;        }
```

Algorithm:

Step 1: Start

Step 2: select flash document.

Step 3: select circle as symbol to draw circle.

Step 4: fill the circle with desired colour.

Step 5: select the object and frames to it.

Step 6: move the object and add frames .

Step 7: repeat step 6

Step 8: add twin motion to observe the animation

Step 9: Stop.

Conclusion: Thus with the use of graphics functions, bouncing ball animation in sine wave form is created.

Questions:

1 Which packages are used for animation?

2 Explain animation of tic-tac –toe game.

3 What is segment table?

Experiment No. 15 (CG)

Aim: Design and implement game / animation clip / Graphics Editor using open source graphics library. Make use of maximum features of Object Oriented Programming.

Software Requirements:

- 64-bit Open source Linux
- Open Source C++ Programming tool

Hardware Requirement:

- C2D, 2GB RAM, 500 GB HDD.

Objectives:

To use open graphics library functions to create game /animation clip using object oriented concepts.

Outcomes:

Students are able to create game or animation clip using open library graphics functions.

Theory:

Animation is the process of designing, drawing, making layouts and preparation of photographic sequences which are integrated in the multimedia and gaming products. Animation involves the exploitation and management of still images to generate the illusion of movement. A person who creates animations is called animator. He / she use various computer technologies to capture the still images and then to animate these in desired sequence.A multimedia product can be sets of texts, graphic arts, sounds, animations and videos. Etc

Types of Animation

- **Traditional animation (cel animation or hand-drawn animation)**
- **Stop motion animation (Claymation, Cut-outs)**
- **Motion Graphics (Typography, Animated logo)**
- **Computer animation**

  2D animation

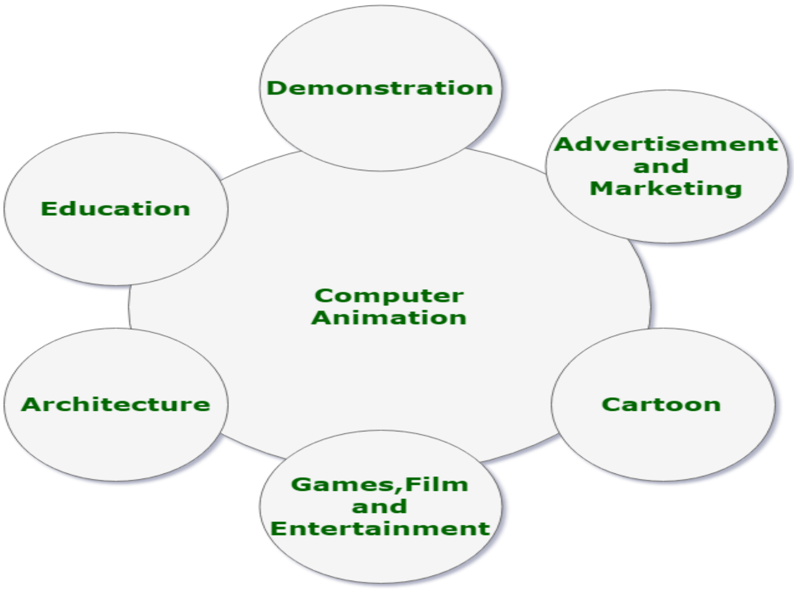
  3D animation

Generally, **Computer animation** is a visual digital display technology that processes the moving images on screen. In simple words, it can be put or defined as the art or power of giving **life, energy and emotions etc.** to any non-living or inanimate object via computers.

It can be presented in form of any video or movie. Computer animation has the ability to make any dead image alive. The key/main concept behind computer animation is to play the defined images at a faster rate to fool the viewer so that the viewer should interpret those images as a continuous motion of images.

Nowadays, animation can be seen in many area around us. It is used in a lot of movies, films and games, education, e-commerce, computer art, training etc. It is a big part of entertainment area as most of the sets and background is all build up through VFX and animation.

Applications of Computer Animation:



Methods/Techniques:

1. **Frame by Frame (Traditional Method):**

   Earlier, in traditional method, animation was done by hands because of the absence of the computer-aided drawing facilities. And, these traditional method required a lot of effort for even making a short video because of the fact that every second of animation requires 24 frames to process.

2. **Procedural:**

   In Procedural method, set of rules are used to animate the objects. Animator defines or specify the initial rules and procedure to process and later runs simulations. Many of the times rules or procedure are based on real world.s physical rule which are shown by mathematical equations.

3. **Behavioral:**

   According to this method/technique, to a certain extent the character or object specifies/determines it's own actions which helps / allows the character to improve later, and in turn, it frees the animator in determining each and every details of the character's motion.

4. **KeyFraming:**

A key frame in computer animation is a frame where we define changes in an animation. According to key framing, a storyboard requirement is must as the animator/artist draws the major frames (frames in which major/important changes can be made later) of animation from it. In key framing, character's or object's key position are the must and need to be defined by the animator, because the missing frames are filled in those key position via computer automatically.

5. **MotionCapture:**

This method of animation uses the live action/motion footage of a living human character which is recorded to the computer via video cameras and markers and later, that action or motion is used/applied to animate the character which gives the real feel to the viewers as if the real human character has been animated. Motion Capture is quite famous among the animators because of the fact that the human action or motion can be captured with relative ease.

6. **Dynamics:**

In this method, simulations are used in order to produce a quite different sequence while maintaining the physical reality. Physics's laws are used in simulations to create the motion of pictures/characters. High level of interactivity can be achieved in this method, via the use of real-time simulations, where a real person performs the action or motions of a simulated character.

Game Programming

Before we actually jump into game programming, we need to know something called event **driven programming**. Event driven programming refers to that style of programming wherein the user of the application is free to choose from several options rather than be confined to a predetermined sequence of interactions with the program. Game programming is one common example of event driven programming. A game is a closed, i.e., complete and self sufficient formal system that represents a subset of reality. A game is a perfect combination of actions-reactions or event-responses where every response is based on the most-recently occurred event.

Elements of Game Programming

In general, a computer game has five elements: Graphics, Sound, Interface, Gameplay, Story

Graphics

Graphics consists of any images that are displayed and any effects that are performed on them. This includes 3D objects, textures, 2D tiles, 2D full screen shots, Full Motion Video (FMV) and anything else that the player will see.

Sound

Sound consists of any music or sound effects that are played during the game. This includes starting music, CD music, MIDI or MOD tracks, Foley effects (environment sounds), and sound effects.

Interface

Sound consists of any music or sound effects that are played during the game. This includes starting music, CD music, MIDI or MOD tracks, Foley effects (environment sounds), and sound effects.

Gameplay

It encompasses how fun the game is, how immense it is, and the length of playability.

Story

The game's story includes any background before the game starts, all information the player gains during the game or when they win and any information they learn about character in the game. A story is an element of a game. The difference between a story and a game is that a story represents the facts in an immutable (i.e., fixed) sequence, while a game represents a branching tree of sequences and allows the player to create his own story by making choice at each branch point.

Game Design Sequence

Since game design requires one to explore one's artistic abilities, it cannot be formulated in a step by step process. However, there are certain technical steps that one needs to follow in one way or another. These are:

1. Determining Initial requirements.
2. Develop Interface.
3. Develop Interface.
4. Develop Logic for Scoring points.

Interface is composed of input and output. While developing interface, the programmer should develop the **static display screens** and **dynamic display screen**. **Static display** is the screen which remains unaffected by the player's actions i.e., the input by the player. The **dynamic display**, on the other hand, is the screen which is governed by the player's actions i.e., the input by the player. Examples of some static display screens are:

Game selection screens

What options are available to the player on the game startup? This describes what options are on the menu, how and where it appears on what screen, how the player gets there, and how he gets out.

Game start screen

What does the screen looks like at the beginning of the game, what are the startup parameters, where are the characters, etc? What messages, if any are on screen, and where? Intro music? etc.Since the dynamic screen vary as per the input given by the player, their descriptions are too many to be listed here. Some examples:

Message Screens

While developing interfaces, you also need to work on screens in response to legal actions of the player, by intimating that he/she is on the right track. Also, you need to work on the screens required to warn the player in case he/she commits an illegal move or action.

End of game message

These screens include messages and responses to questions like: What happens when the player loses? What happens when the player wins? What happens when the player get the high score? Where does the player go when the game is over? How does he start a new game?

This step involves developing a proper logic for gameplay. This requires the game-programmer to answer many questions in the form of program-code. These questions include: How is game played? What are the controls? What is the Game Goal? How is the player going to achieve the game goal? etc. In other words, we must say that since game represents an event-driven situation, the game-programmer i.e., you must specify or program everything that includes:

1. **Determining Initial requirements**

   While writing a game program, after selecting the goal-of-game, one needs to determine its initial requirements. For instance, to write a game program for guessing a number, you need to decide about a way to generate the number, number of players involved, number of chances allowed to the player, a scoring methodology etc. Here we are not aiming at making you a professional game programmer, rather we are concentrating more at giving you an idea of writing simple or elementary game programs.

General Description of Game: The general description of a game involves the general overview of the game, how it works, what happens on each level, etc. It describes **all parts** of the game from the player's perspective:

- What he's supposed to know before he starts playing.
- What he sees.
- What he does.
- His intended reaction to what he sees and does.

2. **Develop Interface**

3. **Develop Logic of Gameplay**

   - responses to the user/player's action.
   - responses to system events.
   - rules of the game.
   - if it's a two player game (if the computer is a player), then the computer's moves and actions.

4. **Develop Logic for Keeping Scores**

Developing logic for the scoring purposes is a subset of developing logic for the game play. For this, you must first decide the scoring policy that you're going to follow in your game. You're going to decide the maximum number of chances allowed, the scoring mechanism, whether it is tied to time or not, etc. During this phase, the milestone events are worked out and accordingly scoring (positively or negatively) is carried out.

Conclusion: Thus, game/animation clip is created using object oriented concepts.

Questions:

1 Which basic functions used to design animation?

2 What is game board? How to create it?

3 Which are open source tools for gaming and animation