



Nutan Maharashtra Vidya Prasarak Mandal's
**NUTAN MAHARASHTRA INSTITUTE OF
ENGINEERING AND TECHNOLOGY**



Department of Computer Engineering



LABORATORY MANUAL

SUBJECT: OOP and Computer Graphics Laboratory

[SUBJECT CODE: 210248]

CLASS: S.E

YEAR: 2023-24

PREPARED BY:

Prof. Renuka R. Kajale

APPROVED BY:

H.O.D. [Computer]

Vision and Mission of the Institute

1. **Vision of the Institute** be a notable institution for providing quality technical education, ensuring ethical, moral, and holistic development of students.
2. **Mission of the Institute**- To nurture engineering graduates with highest technical competence, professionalism and problem solving skills to serve the needs of industry and society.

Vision and Mission of the Department

Department Vision:

Imbibing Quality Technical Education and Overall Development by Endowing Students with Societal and Ethical skills in Computer Engineers

Department Mission:

- To impart engineering knowledge and skills by adopting effective teaching learning processes..
- To develop professional, entrepreneurial & research competencies encompassing continuous intellectual growth
- To produce educated students to exhibit societal and ethical responsibilities in the working environment.

Program Outcomes

1. Engineering knowledge:

Graduates can apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to Civil Engineering related problems.

2. Problem analysis:

An ability to identify, formulate, review research literature, and analyse Civil engineering problems reaching substantiated conclusions using principles of mathematics and engineering sciences.

3. Design/development of solutions:

An ability to plan, analyse, design, and implement engineering problems and design system components or processes to meet the specified needs.

4. Conduct investigations of complex problems:

An ability to use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage:

An ability to apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

6. The engineer and society:

An ability to apply contextual knowledge to assess societal, legal issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability:

An ability to understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics:

An ability to apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and teamwork:

An ability to function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings to accomplish a common goal.

10. Communication:

An ability to communicate effectively on engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation and make effective presentations.

11. Project management and finance:

Ability to demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning:

An ability to engage in independent and life-long learning in the broadest context of technological change.

Program Educational Objectives **(PEOs)**

PEO1: To produce globally competent graduates having Excellent fundamentals, domain knowledge, updated with modern technology to provide effective solutions for computer engineering problems

PEO2: To prepare the graduates to work as a committed professional with professional ethics and values, sense of responsibilities, understanding of legal, safety, health, societal, cultural and environmental issues.

PEO3: To prepare committed and motivated graduates with research attitude, lifelong learning, investigative approach, and multidisciplinary thinking

PEO4: To produce the graduates with well-built managerial and communication skills to work effectively as individual as well as in teams

Program Specific Outcomes (PSOs)

PSO 1 - Employ knowledge to write programs and design algorithms to integrate them with the hardware/software products in the domains of embedded systems, data Science, networking and web technology.

PSO 2 - Apply standard practices and strategies in project development using open-ended programming environments to create innovative career paths to be an entrepreneur, and a zest for higher studies.

Course Outcomes (CO)

Computer Graphics

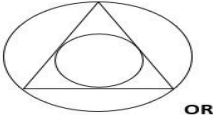
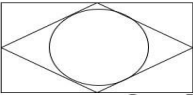
Course Outcome	Statement
	At the end of the course, a student will be able to
210248.1	Apply mathematics to develop Computer programs for elementary graphic operations.
210248.2	Define the concept of windowing and clipping and apply various algorithms to fill and clip polygons.
210248.3	Explain the core concepts of computer graphics, including transformation in two and three dimensions, viewing and projection.
210248.4	Explain the concepts of color models, lighting, shading models and hidden surface elimination.
210248.5	Describe the fundamentals of curves, fractals, animation and gaming.

Object Oriented Programming

Course Outcome	Statement
	At the end of the course, a student will be able to
210248.1	Analyze the strengths of object oriented programming.
210248.2	Design and apply OOP principles for effective programming.
210248.3	Develop the application using object oriented programming language(C++).
210248.4	Apply object-oriented concepts for advanced programming.

Computer Graphics

List of Experiments with Mapping

Sr. No.	Title of Assignment	CG CO	OOP CO	Level of Mapping
1	Write C++ program to draw a concave polygon and fill it with desired color using scan fill algorithm. Apply the concept of inheritance.	CO2	CO2	2,2
2	Write C++ program to implement Cohen Southerland line clipping algorithm.	CO2	-	2
3	<p>a) Write C++ program to draw the following pattern. Use DDA line and Bresenham 's circle drawing algorithm. Apply the concept of encapsulation.</p>  <p>b) Write C++ program to draw the following pattern. Use DDA line and Bresenham's circle drawing algorithm. Apply the concept of encapsulation.</p> 	CO1	CO1	3,2
4	<p>a) Write C++ program to draw 2-D object and perform following basic a) transformations, Scaling b) Translation c) Rotation. Apply the concept of operator overloading.</p> <p>OR</p> <p>b) Write C++ program to implement translation, rotation and scaling transformations on equilateral triangle and rhombus. Apply the concept of operator overloading.</p>	CO3	CO1	3,2
5	<p>a) Write C++ program to generate snowflake using concept of fractals.</p> <p style="text-align: center;">OR</p> <p>b) Write C++ program to generate Hilbert curve using concept of fractals.</p> <p style="text-align: center;">OR</p> <p>c) Write C++ program to generate fractal patterns by using Koch curves.</p>	CO5	-	2

6	<p><i>a) Design and simulate any data structure like stack or queue visualization using graphics. Simulation should include all operations performed on designed data structure. Implement the same using OpenGL.</i></p> <p style="text-align: center;"><i>OR</i></p> <p><i>b) Write C++ program to draw 3-D cube and perform following transformations on it using OpenGL i) Scaling ii) Translation iii) Rotation about an axis (X/Y/Z).</i></p> <p style="text-align: center;"><i>OR</i></p> <p><i>c) Write OpenGL program to draw Sun Rise and Sunset</i></p>	CO2	-	2
---	--	-----	---	---

Object Oriented Programming

List of Experiments with Mapping

Sr. No.	Title of Assignment	CO	Level of Mapping
1	Implement a class Complex which represents the Complex Number datatype. 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.	CO1	1
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 .	CO2	2
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.	CO3	2
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.	CO4	2
5	Write a function template for selection sort that inputs, sorts and outputs an integer array and a float array.	CO2	1
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. 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.	CO6	2

7	<i>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.</i>	CO6	2
---	--	-----	---

Rubrics for Evaluation

Sr. No	Evaluation Criteria	Marks for each Criteria	Rubrics
1	Timely submission	5 or 10	Punctuality reflects the work ethic. Students should reflect that work ethic by completing the lab assignments and reports in a timely manner.
2	Journal Presentation	5 or 10	Students are expected to prepare the journal. The journal presentation of the course should be complete, clear, and understandable.
3	Performance	5 or 10	After the performance, the students should have good knowledge of the experiment.
4	Understanding	5 or 10	The student should be able to explain the methodology used for designing and developing the program/solution. Students should clearly understand the purpose of the assignment and its outcome.
5	Oral	5 or 10	The student should be able to answer the questions related to the lab assignments.

Computer Graphics

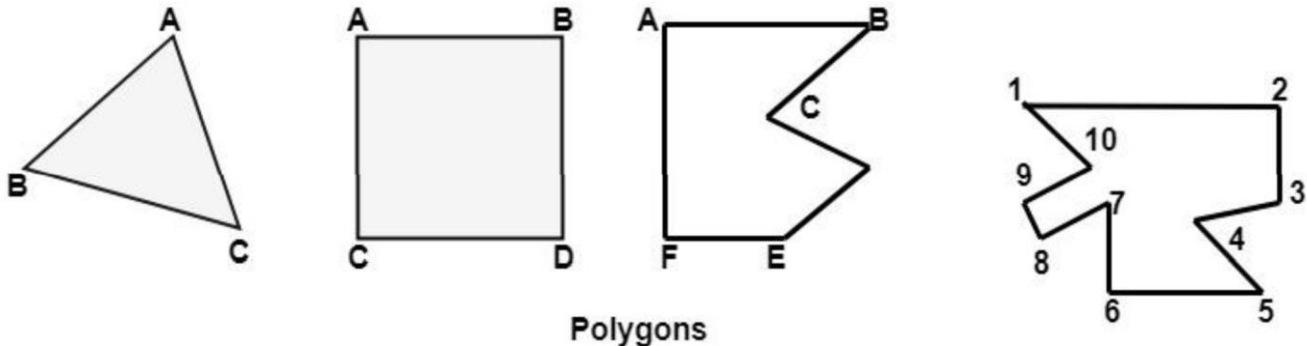
Experiments No.1

Title	A concave polygon filling using scan fill algorithm
Aim/Problem Statement	Write C++ program to draw a concave polygon and fill it with desired color using scan fill algorithm. Apply the concept of inheritance.
Pre-requisite	1. Basic programming skills of C++ 2. 64-bit Open-source Linux 3. Open-Source C++ Programming tool like G++/GCC
Learning Objective	To understand and implement scanline polygon fill 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

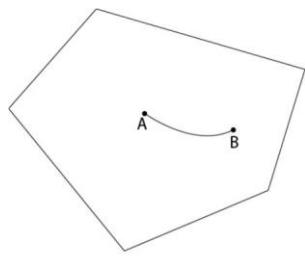
A convex polygon is a simple polygon whose interior is a convex set. In a convex polygon, all interior angles are less than 180 degrees.

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

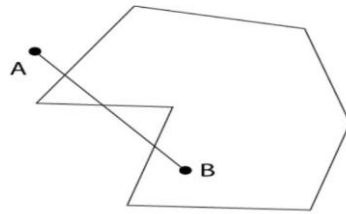
- Every internal angle is less than or equal to 180 degrees.
- Every line segment between two vertices remains inside or on the boundary of the polygon.

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 cross 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.

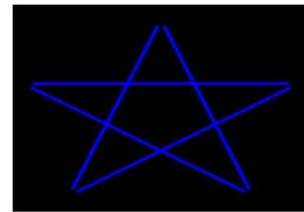
Complex polygon is a polygon whose sides cross over each other one or more times.



Convex polygon



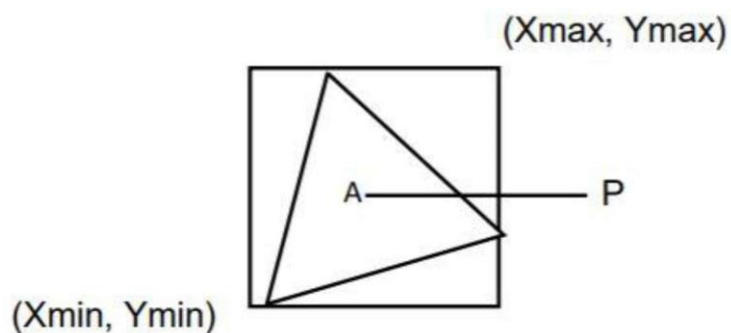
Concave polygon



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 X min and X max and Y min and Y max. 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 $C + V$.

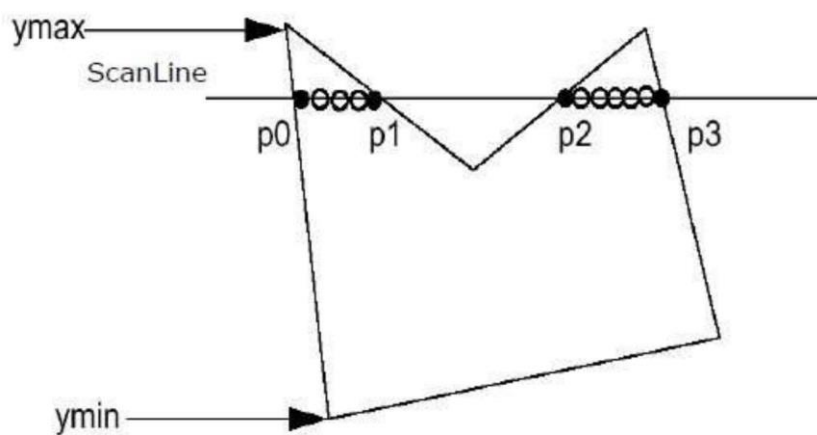
Polygon Filling:

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 S1 enter color intensity as I1 into refresh buffer
4. When two or image surface flag are on, sort the surfaces according to depth and use intensity value S_n 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: Implements a concave polygon and fill it with desired color using scan fill algorithm with concept of inheritance.

Questions:

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

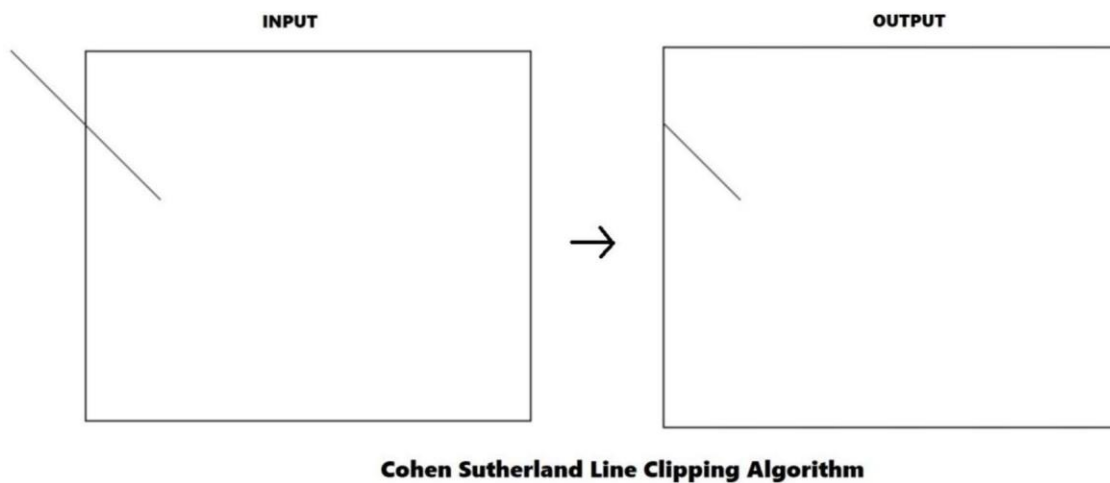
Experiments No.2

Title	Polygon clipping using Cohen Southerland line clipping algorithm
Aim/Problem Statement	Write C++ program to implement Cohen Southerland line clipping algorithm.
Pre -requisite	Basic programming skills of C++ 64-bit Open source Linux Open Source C++ Programming tool like G++/GCC
Learning Objective	To learn Cohen Southerland line clipping algorithm.

Theory:

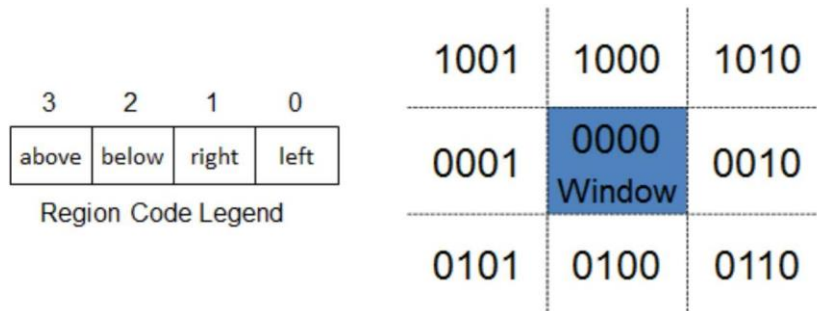
Cohen Sutherland Algorithm is a line clipping algorithm that cuts lines to portions which are within a rectangular area. It eliminates the lines from a given set of lines and rectangle area of interest (view port) which belongs outside the area of interest and clips those lines which are partially inside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

Example:



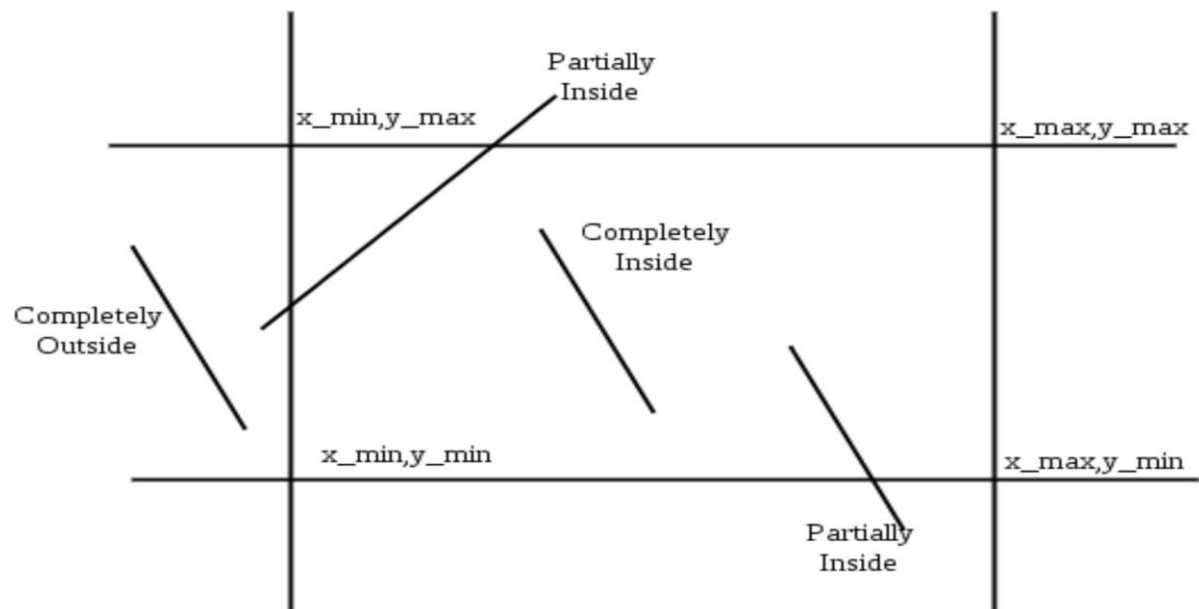
Algorithm: -

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:



As 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.

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: Implement Cohen Sutherland line clipping algorithm.

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

Experiments No.3

Title	Pattern drawing using line and circle.
Aim/Problem Statement	Write C++ program to draw a given pattern. Use DDA line and Bresenham's circle drawing algorithm. Apply the concept of encapsulation.
Pre-requisite	1. Basic programming skills of C++ 2. 64-bit Open-source Linux 3. Open-Source C++ Programming tool like G++/GCC
Learning Objective	To learn and apply DDA line and Bresenham's circle drawing algorithm.

Theory:

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:

Step 1: Get coordinates of both the end points (X1, Y1) and (X2, Y2) from user.

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

$dx = X2 - X1$; $dy = Y2 - Y1$;

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.

$X_{\text{increment}} = dx / (\text{float}) \text{ steps}$;

$Y_{\text{increment}} = dy / (\text{float}) \text{ steps}$;

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++)

{

$X1 = X1 + X_{\text{increment}}$;

$Y1 = Y1 + Y_{\text{increment}}$;

```
putpixel(Round(X1), Round(Y1), 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 it uses only integer arithmetic which makes it significantly faster than other algorithms using floating point arithmetic.

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

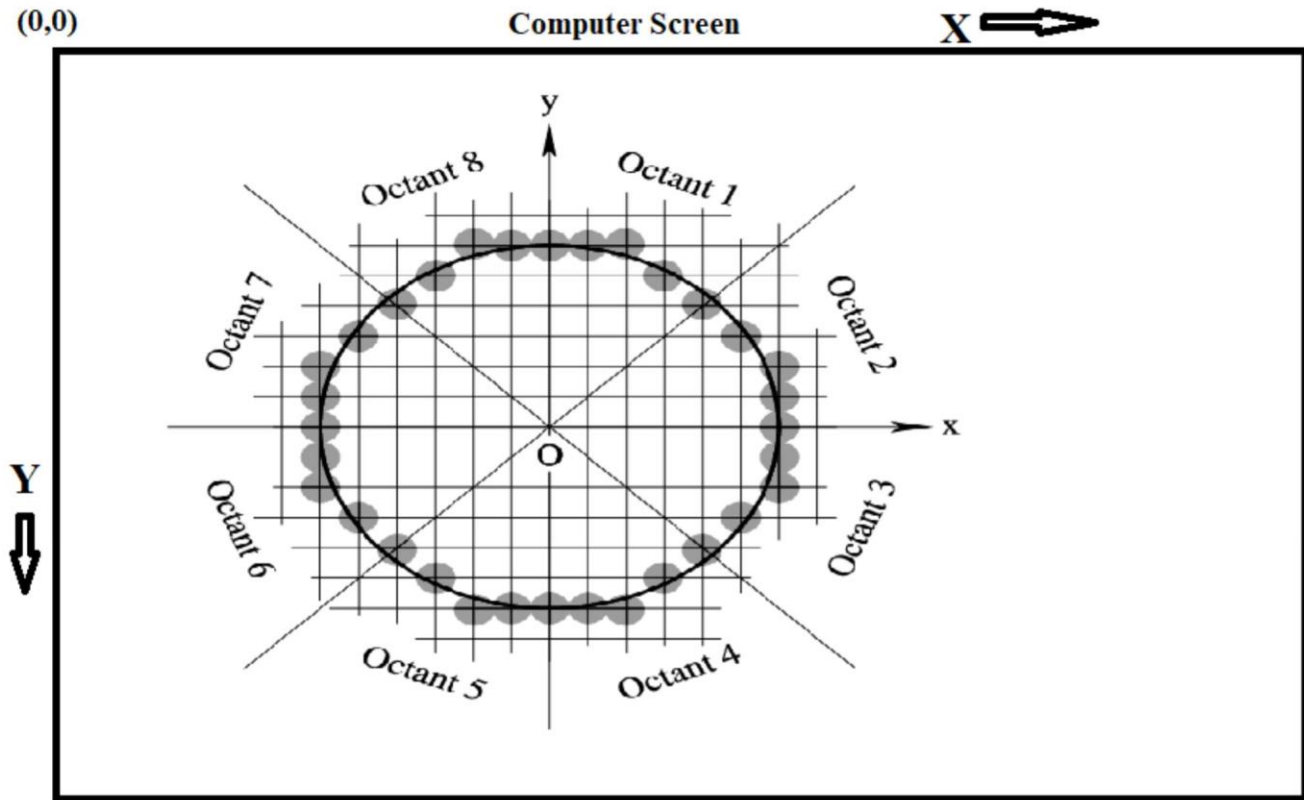
Step 2: Read the radius of circle: (r)

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()` functions:

Octant 1: `putpixel(cen x + x , cen y - y , ColorName);`

Octant 2: `putpixel(cen x + y , cen y - x , ColorName);`

Octant 3: `putpixel(cen x + y , cen y + x , ColorName);`

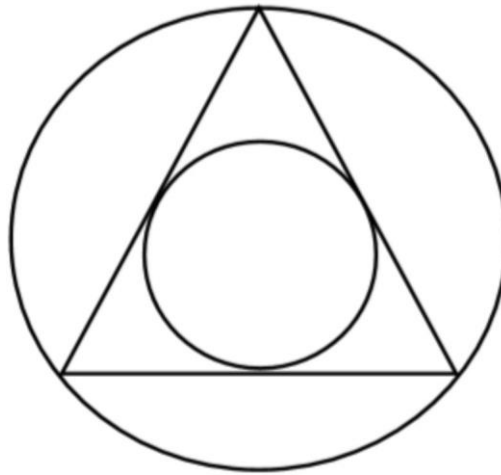
Octant 4: `putpixel(cen x + x , cen y + y , ColorName);`

Octant 5: `putpixel(cen x - x , cen y + y , ColorName);`

Octant 6: `putpixel(cen x - y , cen y + x , ColorName);`

Octant 7: `putpixel(cen x - y , cen y - x , ColorName);`

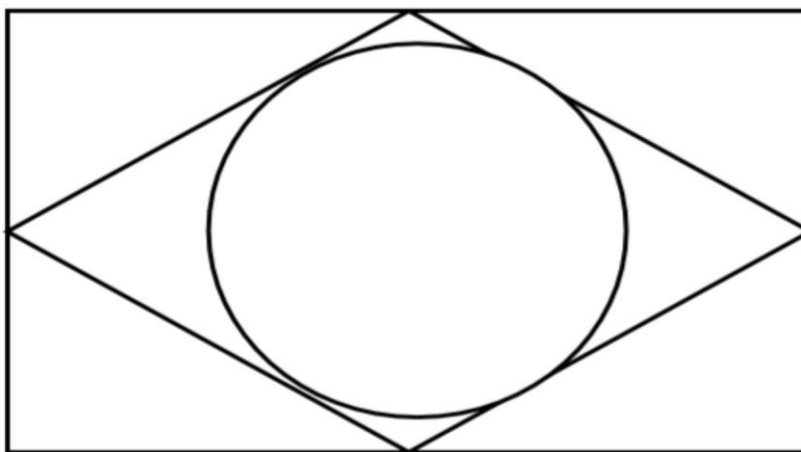
Octant 8: `putpixel(cen x - x , cen y - y , ColorName);`

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, we require coordinates of common center and radius of both the circles.

We will take coordinates of circle and radius of one of the circle from user. Then using the properties of an equilateral triangle, we can find all 3 vertices of equilateral triangle and radius of other circle. Once we get all these parameters, we can call DDA line drawing and Bresenham's circle drawing algorithms by passing appropriate parameters to get required pattern.

OR



To draw this pattern, we require two rectangles and one circle. We can use suitable geometry to get coordinates to draw rectangles and circle. Then we can call DDA line drawing and Bresenham's circle drawing algorithms by passing appropriate parameters to get required pattern.

Conclusion: Implements DDA line and Bresenham's circle drawing algorithm. Apply the concept of encapsulation.

Questions:

1. Explain the derivation of decision parameters in Bresenham's circle drawing algorithm.
2. Explain the concept of encapsulation with example.

Experiments No.4

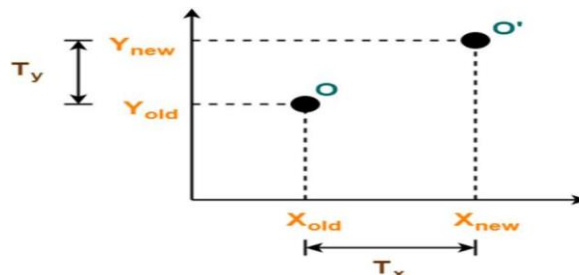
Title	Basic 2-D Transformations.
Aim/Problem Statement	<p>a) Write C++ program to draw 2-D object and perform following basic transformations: Scaling, Translation, Rotation. Apply the concept of operator overloading.</p> <p style="text-align: center;">OR</p> <p>b) Write C++ program to implement translation, rotation and scaling transformations on equilateral triangle and rhombus. Apply the concept of operator overloading.</p>
Pre-requisite	<ol style="list-style-type: none"> 1. Basic programming skills of C++ 2. 64-bit Open-source Linux 3. Open-Source C++ Programming tool like G++/GCC
Learning Objective	To learn and apply basic transformations on 2-D 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 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-

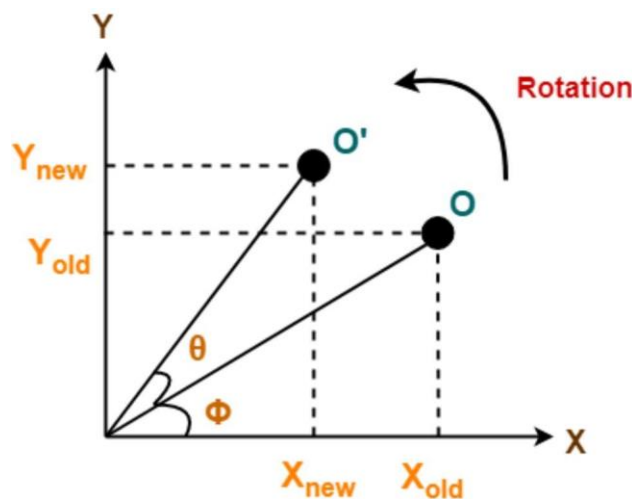
$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

Translation Matrix

2) Rotation:

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

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



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

$$X_{\text{new}} = X_{\text{old}} \times \cos\theta - Y_{\text{old}} \times \sin\theta$$

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

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

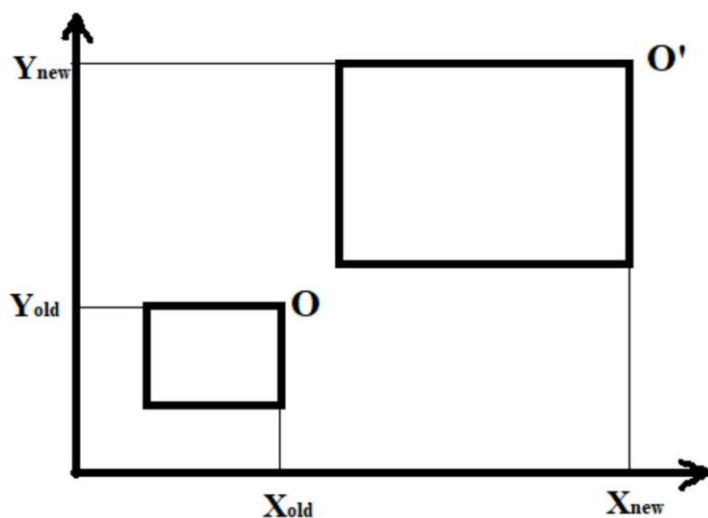
$$\begin{bmatrix} X_{\text{new}} \\ Y_{\text{new}} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} X_{\text{old}} \\ Y_{\text{old}} \end{bmatrix}$$

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-

$$X_{new} = X_{old} \times S_x$$

$$Y_{new} = Y_{old} \times S_y$$

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 to 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)

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

Rotation Matrix (Homogeneous Coordinates representation)

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

Scaling Matrix (Homogeneous Coordinates representation)

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

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: Implements C++ program to draw 2-D object and perform following basic transformations: Scaling, Translation, Rotation. Apply the concept of operator overloading.

Questions:

3. How to rotate any 2-D object about an arbitrary point? Explain in brief.
4. Explain the concept of operator overloading with example.

Experiments No.5

Title	Curves and fractals
Aim/Problem Statement	a) Write C++ program to generate snowflake using concept of fractals. OR b) Write C++ program to generate Hilbert curve using concept of fractals. OR c) Write C++ program to generate fractal patterns by using Koch curves.
Pre -requisite	1. Basic programming skills of C++ 2. 64-bit Open-source Linux 3. Open-Source C++ Programming tool like G++/GCC
Learning Objective	To study curves and fractals

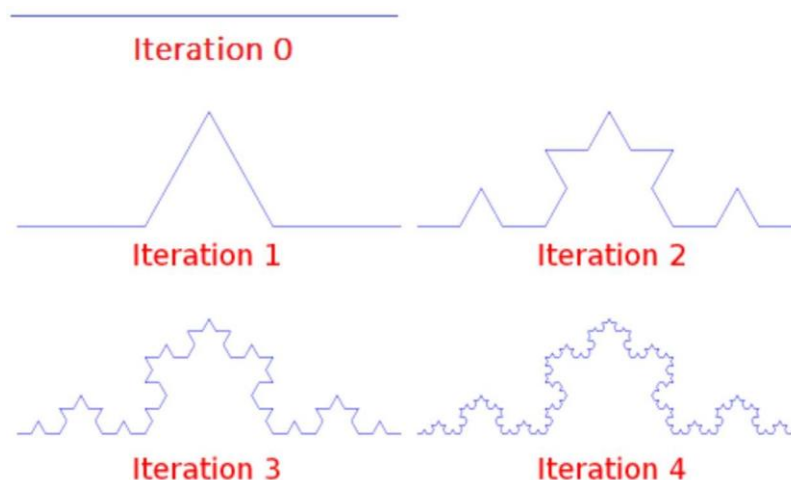
Theory:

Koch Curve:

The Koch curve fractal was first introduced in 1904 by Helge von Koch. It was one of the first fractal objects to be described. To create a Koch curve

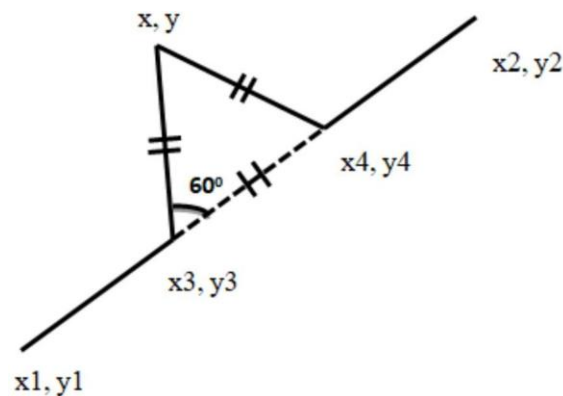
1. Create a line and divide it into three parts.
2. The second part is now rotated by 60° .
3. Add another part which goes from the end of part 2 to the beginning of part 3
4. Repeat step 1 to step 3 with each part of the line.

We will get following Koch curve as number of iterations goes on increasing



Step 1: In Iteration 0, we have a horizontal line.

Step 2: In Iteration 1, line is divided into 3 equal parts. Middle part of a line is rotated in 60° , because it forms a perfect an equilateral triangle as shown below:



Here, (x_1, y_1) and (x_2, y_2) is accepted from user.

Now, we can see line is divided into 3 equal segments: segment $((x_1, y_1), (x_3, y_3))$, segment $((x_3, y_3), (x_4, y_4))$, segment $((x_4, y_4), (x_2, y_2))$ in above figure.

Coordinates of middle two points will be calculated as follows:

$$x_3 = (2 \cdot x_1 + x_2) / 3;$$

$$y_3 = (2 \cdot y_1 + y_2) / 3;$$

$$x_4 = (x_1 + 2 \cdot x_2) / 3;$$

$$y_4 = (y_1 + 2 \cdot y_2) / 3;$$

In our curve, middle segment $((x_3, y_3), (x_4, y_4))$ will not be drawn. Now, in order to find out coordinates of the top vertex (x, y) of equilateral triangle, we have rotate point (x_4, y_4) with respect to arbitrary point (x_3, y_3) by angle of 60° degree in anticlockwise direction. After performing this rotation, we will get rotated coordinates (x, y) as:

$$x = x_3 + (x_4 - x_3) \cdot \cos \theta + (y_4 - y_3) \cdot \sin \theta$$

$$y = y_3 - (x_4 - x_3) \cdot \sin \theta + (y_4 - y_3) \cdot \cos \theta$$

Step 3: In iteration 2, you will repeat step 2 for every segment obtained in iteration 1. In this way, you can generate Koch curve for any number of iterations.

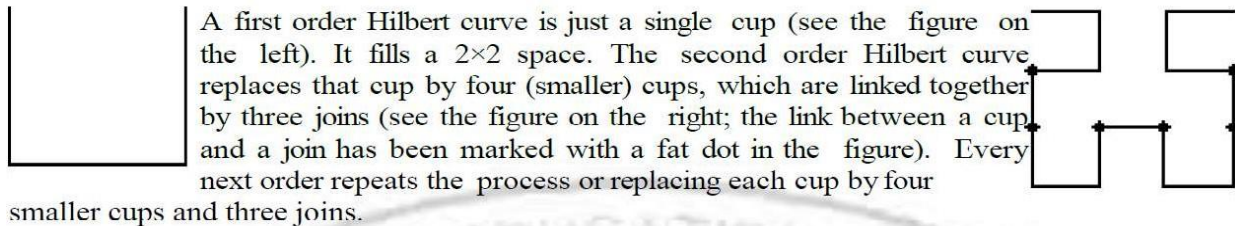
The Hilbert curves

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 neighboring pixels is important. The Hilbert curve is also a special version of a quadtree; any image processing function

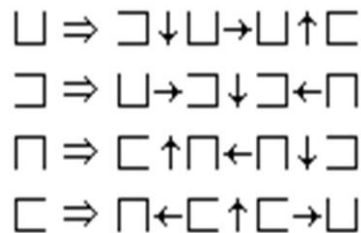
that benefits from the use of quadtrees may also use a Hilbert curve.

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.



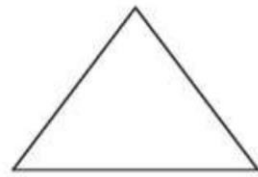
Cup subdivision rules



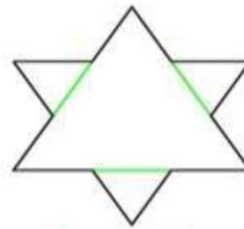
The function presented below (in the "C" language) computes the Hilbert curve. Note that the curve is symmetrical around the vertical axis. It would therefore be sufficient to draw half of the Hilbert curve.

Snowflake curve:

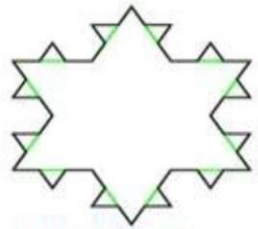
Snowflake curve is drawn using koch curve iterations. In koch curve, we just have a single line in the starting iteration and in snowflake curve, we have an equilateral triangle. Draw an equilateral triangle and repeat the steps of Koch curve generation for all three segments of an equilateral triangle.



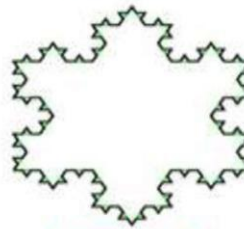
Iteration 0



Iteration 1



Iteration 2



Iteration 3

Conclusion: Implements C++ program to generate snowflake using concept of fractals.

Questions:

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

Experiments No.6

Title	Implementation of OpenGL functions
Aim/Problem Statement	<p>a) Design and simulate any data structure like stack or queue visualization using graphics. Simulation should include all operations performed on designed data structure. Implement the same using OpenGL.</p> <p style="text-align: center;">OR</p> <p>b) Write C++ program to draw 3-D cube and perform following transformations on it using OpenGL i) Scaling ii) Translation iii) Rotation about an axis (X/Y/Z).</p> <p style="text-align: center;">OR</p> <p>c) Write OpenGL program to draw Sun Rise and Sunset.</p>
Pre -requisite	<ol style="list-style-type: none"> 1. Basic programming skills of C++ and OpenGL 2. 64-bit Open-source Linux 3. Open-Source C++ Programming tool like G++/GCC, OpenGL
Learning Objective	To implement OpenGL functions.

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 like 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.

OpenGL Utility Library (GLU): built on-top of the core OpenGL to provide important utilities and more building models (such as quadric surfaces). GLU functions start with a prefix "glu" (e.g., gluLookAt, gluPerspective) 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 quadric 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 functions start with a prefix of "glut" (e.g., glutCreatewindow, glutMouseFunc). 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++.

Getting started with OpenGL

Overview of an OpenGL program

- Main
- Open window and configure frame buffer (using GLUT for example)
- Initialize GL states and display (Double buffer, color mode, etc.)

- Loop
- Check for events

if window event (resize, unhide, maximize etc.) modify the viewport and redraw else if input event (keyboard and mouse etc.) handle the event (such as move the camera or change the state) *and usually draw the scene*

- Redraw
- Clear the screen (and buffers e.g., z-buffer)
- Change states (if desired)
- Render
- Swap buffers (if double buffer)

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
 - Convert shapes into pixels on screen

OpenGL Syntax

- All functions have the form: gl*
- *glVertex3f()* – **3** means that this function take three arguments, and **f** means that the type of those arguments is float.
- *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*
- 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.
```

OpenGL provides a consistent interface to the underlying graphics hardware. This abstraction allows a single program to run on 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 is 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();
```

Developer-Driven Advantages

- Industry standard
- An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.
- Stable
- OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.
- Reliable and portable

- All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system.
- Evolving
- Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.
- Scalable
- OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.
- Easy to use
- OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.
- Well-documented:
- Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

Conclusion:

Implements and simulate any data structure like stack or queue visualization using graphics. Simulation should include all operations performed on designed data structure. Implement the same using OpenGL.

Questions:

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

Object Oriented Programming

Experiments No.1

Title	Implementation Constructor and Overload operator .
Aim/Problem Statement	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.
Pre -requisite	Object oriented concepts.
Learning Objective	To understand operator overloading concept.

Theory:

A complex number is a number that can be expressed in the form $a + bi$, where a and b are real numbers and i is the imaginary unit, that satisfies the equation $i^2 = -1$. In this expression, a is the real part and b is the imaginary part of the complex number.

Operations on Complex number:

- 1. Addition of Complex Numbers** Addition of two complex numbers $a + bi$ and $c + di$ is defined as follows.
 $(a+bi)+(c+di)=(a+c)+(b+d) i$

This is similar to grouping like terms: real parts are added to real parts and imaginary parts are added to imaginary parts.

2. Subtraction of Complex Numbers

The subtraction of two complex numbers $a + bi$ and $c + di$ is defined as follows. $(a + bi) - (c + di) = (a - b) + (b - d) i$

3. Multiply Complex Numbers

The multiplication of two complex numbers $a + bi$ and $c + di$ is defined as follows. $(a + bi)(c + di) = (ac - bd) + (ad + bc) i$

However you do not need to memorize the above definition as the multiplication can be carried out using properties similar to those of the real numbers and the added property $i^2 = -1$. (see the example below)

OOP Features & Concepts used in this practical :

Operator Overloading: 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.

Box operator+(const Box&);

declares the addition operator that can be used to **add** two Box objects and returns final Boxobject.

Overloadable/Non-overloadableOperators:

Following is the list of operators which can be overloaded:

+	-	*	/	%	^
&		~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	
+=	-=	/=	%=	^=	&=
=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

Operator that are not overloaded are follows

scope operator - ::

sizeof

member selector - . member

pointer selector - *ternary

operator - ?:

For calling function:

resultant = Object operator symbol Object;

resultant will vary depending upon operation done and return type of function.

Conclusions: Complex number operations are implemented with the operator overloading concept.

Experiments No.2

Title	Personal information system using constructor, destructor, static member functions, friend class, this pointer, inline code and dynamic memory allocation.
Aim/Problem Statement	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 licence 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, friend class, this pointer, inline code and dynamic memory allocation operators-new and delete.
Pre -requisite	Object oriented concepts.
Learning Objective	To understand operator Constructor,Static and Pointer concept.

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

1. **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.
2. **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.
3. **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 :

1. A static function can have access to only other static members (fun or var) declared in the same class
2. A static function can be called using the class name instead of its object name

Class_name :: function_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:

- ✓ They cannot access non static class member data using the member-selection operators(. or ->).
- ✓ They cannot be declared as **virtual**.
- ✓ They cannot have the same name as a non static function that has the same argumenttypes.

Ex. Shall we give the example.....

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:

- ✓ It is not in the scope of the class to which it has been declared as friend.
- ✓ 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.
- ✓ It can be declared in private or in the public part of the class.
- ✓ 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 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 variable's memory address rather than data.

Declaration of pointer variable is in the following form:

`Data_type * ptr_var;`

Eg. `int * ptr;`

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

We can initialize pointer variable as follows:

`* ptr ; // declaration`

`ptr = &a ; // initialization`

Pointers to objects:

Consider the following example

item P ; // where item is class & P is object

Similarly, we can define a pointer `item_ptr` of type `item` as follows:

*`*it_ptr ;`*

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

Ex. `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 licence 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

Conclusions:

Hence, we have successfully studied concept of constructor, default constructor, copy constructor, destructor, static member functions, friend class, this pointer, inline code and dynamic memory allocation operators-new and delete.

Experiments No.3

Title	Creating a class which uses the concept of inheritance, displays data and data members and uses the concept of exception handling.
Aim/Problem Statement	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.
Pre -requisite	Object oriented concepts.
Learning Objective	To learn the concept of inheritance and exception handling.

Theory:

Inheritance:

Inheritance in Object Oriented Programming can be described as a process of creating new classes from existing classes. New classes inherit some of the properties and behavior of the existing classes. An existing class that is "parent" of a new class is called a base class. New class that inherits properties of the base class is called a derived class. Inheritance is a technique of code reuse. It also provides possibility to extend existing classes by creating derived classes.

The basic syntax of inheritance is:

Class DerivedClass : accessSpecifier BaseClass

There are 3 access specifiers: Namely public, private and protected.

public:

This inheritance mode is used mostly. In this the protected member of Base class becomes protected members of Derived class and public becomes public.

protected:

In protected mode, the public and protected members of Base class becomes protected members of Derived class.

private:

In private mode the public and protected members of Base class become private members of Derived class.

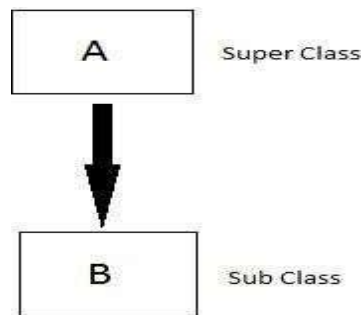
Types of Inheritance

In C++, we have 5 different types of Inheritance.

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

Single Inheritance:

In this type of inheritance one derived class inherits from only one base class. It is the most simplest form of Inheritance.



Syntax:

```
class subclass_name : access_mode base_class
```

```
{
    //body of subclass
```

Multiple Inheritance:

In this type of inheritance a single derived class may inherit from two or more than two base classes.

```
class subclass_name : access_mode base_class1, access_mode base_class2, ....
```

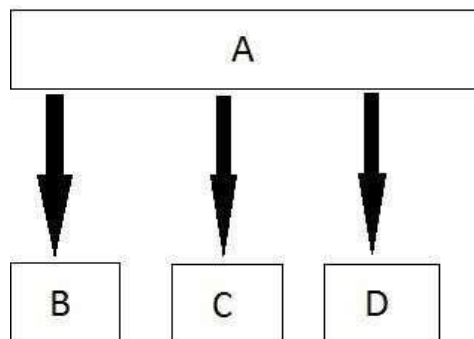
```
{
    //body of subclass
};
```

Multilevel Inheritance:

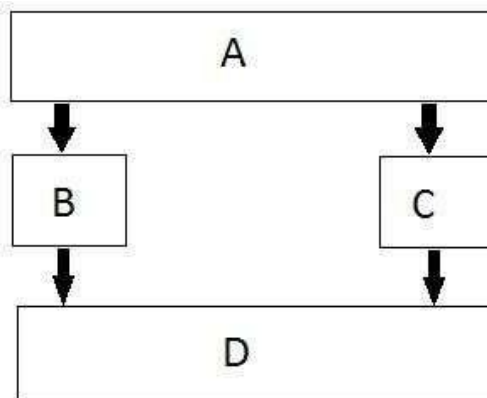
In this type of inheritance the derived class inherits from a class, which in turn inherits from some other class. The Super class for one, is sub class for the other.

Hierarchical Inheritance:

In this type of inheritance, multiple derived classes inherit from a single base class.

**Hybrid Inheritance:**

Hybrid Inheritance is combination of any 2 or more types of **Inheritance**.

**Exception Handling:**

Exception handling is part of C++ and object oriented programming. They are added in C++ to handle the unwanted situations during program execution. If we do not type the program correctly then it might result in errors. Main purpose of exception handling is to identify and report the runtime error in the program.

Famous examples are divide by zero, array index out of bound error, file not found, device not found, etc.

C++ exception handling is possible with three keywords i.e. try, catch and throw. Exception handling performs the following tasks:-

- Find the problem in the given code. It is also called as hit exception.
 - It informs error has occurred. It is called as throwing the exception.
- We receive the error info. It is called as catching the exception.
- It takes the corrective action. It is called as exception handling.

TRY:- It is block code in which there are chances of runtime error. This block is followed by one or more catch block. Most error prone code is added in try block.

CATCH:- This is used to catch the exception thrown by the try block. In catch block we take corrective action on throwing exception. If files are opened, we can take corrective action like closing file handles, closing database connections, saving unsaved work, etc.

THROW:- Program throws exception when problem occurs. It is possible with

```
throw keyword.SNYTAX:=
//normal program code try{
throw exception
}
catch(argument)
{
...
}
```

Algorithm:

1. Start.
2. Create classes Publication, book and tape.
3. Publication class having data members title, price.
4. Class Book having data members pages and member functions getdata() and putdata().
5. Class Tape having data members minutes and member functions getdata() and putdata().
6. Create an object b of class book and object t of class tape.
7. Stop.

Conclusion:

Hence, we have successfully studied concept of inheritance and exception handling.

Questions:

1. What is Exception handling?
2. What is Inheritance?

Experiments No.4

Title	File Handling In C++
Aim/Problem Statement	Implement a program to open, read, write and to close the file in c++. Implement the following operations: <ol style="list-style-type: none"> 1. Open a file. 2. Read data from file. 3. Write data into file. 4. Close the file.
Pre -requisite	Object oriented concepts.
Learning Objective	To learn the concept of File Handling In C++

Theory:

So far, 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.

Operations :

1. Creating a file and output some data

In order to create files we have to learn about File I/O i.e. how to write data into a file and how to read data from a file. We will start this section with an example of writing data to a file. We begin as before with the include statement for stdio.h, then define some variables for use in the example including a rather strange looking new type.

```
#include <stdio.h>
#include <stdio.h>
main( )
{
    FILE *fp; char
    stuff[25]; int index;
    fp = fopen("TENLINES.TXT", "w"); /* open for writing */
    strcpy(stuff, "This is an example line.");
    for (index = 1; index <= 10; index++) fprintf(fp, "%s Line
    number %d\n", stuff, index); fclose(fp); /* close the file
    before ending program */
}
```

2. Reading from File

When an r is used, the file is opened for reading, a w is used to indicate a file to be used for writing, and an

a indicates that you desire to append additional data to the data already in an existing file.

Most C compilers have other file attributes available; check your Reference Manual for details. Using the r indicates that the file is assumed to be a text file. Opening a file for reading requires that the file already exist. If it does not exist, the file pointer will be set to NULL and can be checked by the program.

```
#include <stdio.h>void
main()
{
    FILE *fopen(), *fp;int c;
    fp = fopen("prog.c","r");c =
    getc(fp) ;
    while (c!= EOF)
    {
        putchar(c); c =
        getc(fp);
    }
    fclose(fp);
}
```

3. Write In File

When a file is opened for writing, it will be created if it does not already exist and it will be reset if it does, resulting in the deletion of any data already there. Using the w indicates that the file is assumed to be a text file.

```
#include <stdio.h>int
main()
{
    FILE *fp;
    file = fopen("file.txt","w");
    /*Create a file and add text*/
    fprintf(fp,"%s","This is just an example :)"); /*writesdata to
the file*/
    fclose(fp); /*done!*/return
    0;
}
```

4. Close the File

To close a file you simply use the function `fclose` with the file pointer in the parentheses. Actually, in this simple program, it is not necessary to close the file because the system will close all open files before returning to DOS, but it is good programming practice for you to close all files in spite of the fact that they will be closed automatically, because that would act as a reminder to you of what files are open at the end of each program.

```
fclose(fp);
```

Conclusions:

File operation is implemented using object oriented programming language features.

Assignment Questions:

- 1) what is file?
- 2) What is database?
- 3) What is file descriptor?

Experiments No.5

Title	Selection sort using function template.
Aim/Problem Statement	Implement C++ program for Selection sort using function template
Pre -requisite	Object oriented concepts.
Learning Objective	To learn the concept of File Handling In C++

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 function that returns the greater one of two objects 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.

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]);
    }
}
```

election 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 the remaining $n - 1$ elements and so on, for $(n - 1) + (n - 2) + \dots + 2 + 1 = n(n - 1) / 2 \in \Theta(n^2)$ 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.
5. Stop

Conclusion:

Hence, we have successfully implemented Selection sort using function template.

Experiments No.6

Title	Personnel information system using sorting and searching for STL and vector container
Aim/Problem Statement	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
Pre -requisite	Object oriented concepts.
Learning Objective	To learn the concept STL, searching, sorting and vector container.

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
- Container Adaptors : provide a different interface for sequential containers.
 - queue
 - priority_queue
 - stack
- Associative Containers : implement sorted data structures that can be quickly searched ($O(\log n)$ 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 builtin function in C++ STL by the name of `sort()`. This function internally uses IntroSort. In more details it is implemented using hybrid of QuickSort, HeapSort and InsertionSort. By default, it uses QuickSort but if QuickSort is doing unfair partitioning and taking more than $N \cdot \log N$ time, it switches to HeapSort and when the array size becomes really small, it switches to Insertion Sort. The prototype for `sort` is :

```
sort(startaddress, endaddress)
```

startaddress: the address of the first element of the array

endaddress: the address of the next contiguous location of the last element of the array.

So actually `sort()` sorts in the range of `[startaddress, endaddress)`

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.

Algorithm:

1. Start.
2. Give a header file to use 'vector'.
3. Create a vector naming 'personal_records'.
4. Initialize variables to store name, birth date and telephone number.
5. Using iterator store as many records you want to store using predefined functions as `push_back()`.
6. Create another vector 'item_record'
7. Initialize variables to store item code, item name, quantity and cost.
8. Using iterator and predefined functions store the data.
9. Using predefined function `sort()`, sort the data stored according to user requirements.
10. Using predefined function `search`, search the element from the vector the user wants to check.
11. Display and call the functions using a menu.

End.

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 as search() and sort()

Questions:

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

Experiments No.7

Title	To use map associative container.
Aim/Problem Statement	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.
Pre -requisite	Object oriented concepts.
Learning Objective	To learn the concept of map associative container.

Theory:

Map associative container:

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

map::operator[]

This operator is used to reference the element present at position given inside the operator. It is similar to the at() function, the 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.

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 populationMap.insert().
5. Display the population of states.
6. End.

Conclusion:

Hence, we have successfully studied the concept of map associative container

Questions:

1. What is an associative container in C++?
2. What is map in C++?