OOPCGI Practicals

Computer Graphics

Q. Cohen Sutherland

```
#include <graphics.h>
#include <conio.h>
#include <iostream>
// SPPU CG Group A:4
// Write C++ program to implement Cohen Sutherland line clipping

using namespace std;

int main() {
    int gd = DETECT, gm;
    int X1, Y1, X2, Y2;

    cout << "Enter the Clipping window Coordinates:" << endl;
    cin >> X1 >> Y1 >> X2 >> Y2;

    float x1, y1, x2, y2;
    cout << "Enter the Line Coordinates:" << endl;
    cin >> x1 >> y1 >> x2 >> y2;

    float op[2][4];

    // Generate outcodes
    op[0][0] = (y1 < Y1) ? 1 : 0;
    op[0][1] = (y1 > Y2) ? 1 : 0;
    op[0][2] = (x1 > X2) ? 1 : 0;
```

```
    op[0][3] = (x1 < X1) ? 1 : 0;

    op[1][0] = (y2 < Y1) ? 1 : 0;
    op[1][1] = (y2 > Y2) ? 1 : 0;
    op[1][2] = (x2 > X2) ? 1 : 0;
    op[1][3] = (x2 < X1) ? 1 : 0;

    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

    outtextxy(100, 100, "Before Clipping:");
    rectangle(X1, Y1, X2, Y2);
    line(x1, y1, x2, y2);
    delay(3000);
    cleardevice();

    float m = (x2 != x1) ? (y2 - y1) / (x2 - x1) : 0;

    // Check trivial acceptance and rejection
    if (!(op[0][0] == 0 && op[0][1] == 0 && op[0][2] == 0 && op|
          op[1][0] == 0 && op[1][1] == 0 && op[1][2] == 0 && op|
        if (((op[0][0] && op[1][0]) == 0) &&
            ((op[0][1] && op[1][1]) == 0) &&
            ((op[0][2] && op[1][2]) == 0) &&
            ((op[0][3] && op[1][3]) == 0)) {
            // Perform clipping
            if (op[0][0]) {
                x1 = x1 + (Y1 - y1) / m;
                y1 = Y1;
            }
            if (op[1][0]) {
                x2 = x2 + (Y1 - y2) / m;
                y2 = Y1;
            }
            if (op[0][1]) {
                x1 = x1 + (Y2 - y1) / m;
                y1 = Y2;
```

```
            }
            if (op[1][1]) {
                x2 = x2 + (Y2 - y2) / m;
                y2 = Y2;
            }
            if (op[0][2]) {
                y1 = y1 + m * (X2 - x1);
                x1 = X2;
            }
            if (op[1][2]) {
                y2 = y2 + m * (X2 - x2);
                x2 = X2;
            }
            if (op[0][3]) {
                y1 = y1 + m * (X1 - x1);
                x1 = X1;
            }
            if (op[1][3]) {
                y2 = y2 + m * (X1 - x2);
                x2 = X1;
            }
        } else {
            x1 = x2 = y1 = y2 = 0;  // Completely outside
        }
    }

    outtextxy(100, 100, "After Clipping:");
    rectangle(X1, Y1, X2, Y2);
    line(x1, y1, x2, y2);

    getch();
    closegraph();
    return 0;
}
```

Q.1
// Write C++ program to draw a concave polygon and fill it with desired color
// using scan fill algorithm. Apply the concept of inheritance.

```
// Write C++ program to draw a concave polygon and fill it with
// using scan fill algorithm. Apply the concept of inheritance.

#include<iostream>
#include<graphics.h>
#include<algorithm>

using namespace std;

// Function to fill the polygon using Scan-Fill Algorithm
void scanFill(int vertices[][2], int n) {
    int maxY = 0;

    // Find the maximum y-coordinate of the polygon to define th
    for (int i = 0; i < n; i++) {
        maxY = max(maxY, vertices[i][1]);
    }

    // For each scanline (y from 0 to maxY), find the intersecti
    for (int y = 0; y <= maxY; y++) {
        // Create a list to store the x-coordinates of intersect
        int intersections[100]; // Assuming no more than 100 int
        int intersectionCount = 0;

        // Loop through each edge and check if it intersects wit
        for (int i = 0; i < n; i++) {
            int x1 = vertices[i][0];
            int y1 = vertices[i][1];
            int x2 = vertices[(i + 1) % n][0];
            int y2 = vertices[(i + 1) % n][1];

            // Check if the edge crosses the current scanline
```

```
            if ((y1 <= y && y2 > y) || (y2 <= y && y1 > y)) {
                // Calculate the x-coordinate of the intersectio
                int x_intersection = x1 + (y - y1) * (x2 - x1) /
                intersections[intersectionCount++] = x_intersect
            }
        }

        // Sort the intersections in ascending order of x-coordi
        sort(intersections, intersections + intersectionCount);

        // Fill the area between pairs of intersections
        for (int i = 0; i < intersectionCount; i += 2) {
            if (i + 1 < intersectionCount) {
                // Draw a horizontal line between each pair of i
                line(intersections[i], y, intersections[i + 1],
            }
        }
    }
}

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, NULL);

    // Define vertices of the polygon (a concave example)
    int vertices[4][2] = {{100, 100}, {50, 400}, {100, 200}, {15

    // Draw the outline of the polygon
    for (int i = 0; i < 4; i++) {
        line(vertices[i][0], vertices[i][1], vertices[(i + 1) %
    }

    // Fill the polygon using Scan-Fill algorithm
    scanFill(vertices, 4);

    getch();
```

```
    closegraph();
    return 0;
}
```

CG VIVA QUESTIONE

1. What is the purpose of the Scan-Fill algorithm in computer graphics?

Answer:

The Scan-Fill algorithm is used to fill the interior of a polygon by drawing horizontal lines (scanlines) at different vertical positions (y-coordinates). For each scanline, the algorithm finds the x-coordinates of intersections with the polygon's edges and then fills the region between pairs of intersections, effectively coloring the polygon's interior.

2. What is the role of inheritance in this program?

Answer:

In the current program, inheritance is not explicitly implemented, but you can modify the program to introduce inheritance. For example, a base class could be created for a generic shape with basic methods like drawing and filling, and then a derived class could represent specific polygons (e.g., concave polygons) and override the filling behavior. This would allow for more flexible code reuse and customization for different types of polygons.

3. How does the program determine the intersections with the scanline?

Answer:

For each scanline, the program checks all edges of the polygon to determine if they intersect with the scanline. If an edge's endpoints lie on opposite sides of the scanline, an intersection occurs. The x-coordinate of the intersection is calculated using linear interpolation based on the formula:

//Q.) Write C++ program to draw the following pattern. Use DDA line and Bresenham's circle drawing algorithm. Apply the concept of encapsulation.

```
#include <iostream>
#include <graphics.h>
#include <math.h>
using namespace std;

class dda {
private:
    float x1, x2, y1, y2, dx, dy, xi, yi, xn, yn, length;

public:
    void initialValues(float a, float b, float c, float d) {
        x1 = a;
        y1 = b;
        x2 = c;
        y2 = d;
        dx = x2 - x1;
        dy = y2 - y1;
        if (abs(dx) >= abs(dy))
            length = abs(dx);
        else
            length = abs(dy);
        xi = dx / length;
        yi = dy / length;
        xn = x1;
        yn = y1;
    }

    void drawLine() {
        for (int i = 1; i <= length; i++) {
            putpixel(floor(xn), floor(yn), CYAN);
            xn = xn + xi;
            yn = yn + yi;
```

```
            delay(100);
        }
    }
};

class bressenham {
private:
    int xc, yc, r, xi, yi, pi;

public:
    void initialValues(int a, int b, int c) {
        xc = a;
        yc = b;
        r = c;
        xi = 0;
        yi = r;
        pi = 3 - (2 * r);
    }

    void drawCircle() {
        while (xi <= yi) {
            putpixel(xc + xi, yc + yi, WHITE);
            putpixel(xc - xi, yc + yi, CYAN);
            putpixel(xc + xi, yc - yi, WHITE);
            putpixel(xc - xi, yc - yi, CYAN);
            putpixel(xc + yi, yc + xi, WHITE);
            putpixel(xc - yi, yc + xi, CYAN);
            putpixel(xc + yi, yc - xi, WHITE);
            putpixel(xc - yi, yc - xi, CYAN);
            xi++;
            if (pi < 0)
                pi = pi + (4 * xi) + 6;
            else {
                yi--;
                pi = pi + (4 * (xi - yi)) + 10;
            }
```

```
            delay(100);
        }
    }
};

int main() {
    int gd = DETECT, gm;
    int xc, yc, r;

    // Get user input for circle parameters
    cout << "Enter the x coordinate of circle's center: ";
    cin >> xc;
    cout << "Enter the y coordinate of circle's center: ";
    cin >> yc;
    cout << "Enter the radius of the circle: ";
    cin >> r;

    // Initialize graphics
    bressenham b1, b2;
    dda line1, line2, line3;
    initgraph(&gd, &gm, NULL);

    // Draw two concentric circles
    b1.initialValues(xc, yc, r);
    b1.drawCircle();
    b2.initialValues(xc, yc, r / 2);
    b2.drawCircle();

    // Draw equilateral triangle inscribed in the outer circle
    line1.initialValues(xc, yc - r, xc + (r / 1.154), yc + (r /
    line2.initialValues(xc, yc - r, xc - (r / 1.154), yc + (r /
    line3.initialValues(xc + (r / 1.154), yc + (r / 2), xc - (r
    line1.drawLine();
    line2.drawLine();
    line3.drawLine();
```

```
    // Wait for user input before exiting
    getch();
    closegraph();
    return 0;
}
```

1. What is the concept of encapsulation in Object-Oriented Programming (OOP)?

Answer:

Encapsulation is one of the four fundamental OOP concepts, which refers to the bundling of data (variables) and methods (functions) that operate on the data into a single unit, called a class. It restricts direct access to some of the object's components, which is a way of preventing unintended interference and misuse of the data.

2. What is the difference between DDA line drawing algorithm and Bresenham’s circle drawing algorithm?

Answer:

- **DDA (Digital Differential Analyzer) Line Drawing Algorithm**:

  The DDA algorithm uses floating-point calculations to determine the points on the line between two given points. It calculates the x and y coordinates incrementally and plots the points on the screen. The major disadvantage is that it involves floating-point arithmetic, which can be slower than integer-based algorithms.

- **Bresenham’s Circle Drawing Algorithm**:

  Bresenham’s algorithm for circle drawing is more efficient as it uses integer calculations to generate the points of the circle. It is based on the decision parameter that determines whether the next point should be plotted horizontally or vertically. It avoids floating-point operations, making it faster than DDA.

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

1. What is operator overloading in C++?

Answer:

Operator overloading is a feature in C++ that allows you to redefine the way operators work for user-defined data types (e.g., classes and objects). It allows operators to have different meanings depending on their operands, making the code more intuitive and expressive. For example, you can overload the `+` operator to add two complex numbers, or `*` to scale a geometric object.

---

2. How is operator overloading implemented in the context of this 2D transformation program?

Answer:

In this program, operator overloading is used to apply transformations like scaling, translation, and rotation to a 2D object (e.g., a polygon or a point). Operators like `+`, `*`, or others might be overloaded to handle transformation operations directly on the objects. For example:

- The `+` operator might be overloaded to handle translation, where adding a vector (translation) to a point (object) results in a new translated point.
- The ` ` operator might be overloaded to handle scaling, where multiplying a point by a scalar value results in a scaled point.
- The ` ` operator could also be used for rotating a point by an angle

. How does Rotation of a 2D object work?

Answer:

Rotation in 2D is typically done around the origin `(0, 0)` using an angle `θ`. The new coordinates `(x', y')` of a point `(x, y)` after rotation are given by the following formulas:

- `x' = x * cos(θ) - y * sin(θ)`

- `y' = x * sin(θ) + y * cos(θ)`
  Where
  `θ` is the angle of rotation in radians, and `cos` and `sin` are the cosine and sine functions, respectively. This rotates the point `(x, y)` by the angle `θ` in a counterclockwise direction.

a) Write C++ program to generate snowflake using concept of fractals.
OR
b) Write C++ program to generate Hilbert curve using concept of fractals.

→

1. What are fractals, and how are they related to recursive algorithms?

Answer:

Fractals are complex geometric shapes that can be split into parts, each of which is a reduced-scale copy of the whole. They exhibit self-similarity, meaning they look similar at different levels of magnification. Fractals are often generated using recursive algorithms, where the function repeatedly calls itself with a reduced size or complexity to build up the fractal pattern.

---

2. What is the significance of recursion in generating fractals like the Snowflake or Hilbert Curve?

Answer:

Recursion is a key concept in generating fractals because it allows us to break down complex shapes into simpler parts. In the case of the Snowflake (Koch curve) or Hilbert curve, recursion enables the iterative construction of smaller sub-patterns that form the entire fractal structure. For example, the Koch snowflake involves recursively replacing line segments with smaller ones in each iteration, and the Hilbert curve uses recursive subdivision to fill a 2D space.

---

3. Can you explain the approach to generate a Snowflake (Koch curve) fractal in your C++ program?

Answer:

The Snowflake fractal is based on the Koch curve, where in each iteration, every line segment is divided into three equal parts, and the middle part is replaced with two sides of an equilateral triangle. The process is repeated recursively for each line segment. In the C++ program, we:

- Start with an equilateral triangle.
- In each recursive step, divide the edges and replace the middle section with a triangle.
- Draw the new segments and repeat the process for a specified number of iterations.

---

4. How does the recursive function work in the Koch snowflake fractal generation?

Answer:

The recursive function works by dividing each line segment into three parts, and in the middle, it adds a triangle-like shape. The function calls itself on smaller segments until the maximum depth or iteration is reached. For each line segment:

- Divide it into three equal parts.
- Form the peak of the triangle by calculating a new point.
- Draw the new segments that form the snowflake.
- Repeat the process recursively on each of the new segments.

---

5. What is the role of angle manipulation in the Koch snowflake fractal generation?

Answer:

Angle manipulation is essential in the Koch snowflake fractal generation because it helps define the direction of each line segment as the fractal evolves. When replacing a line segment with the triangular addition, angles are adjusted to ensure that the new segments are placed correctly relative to the original ones. Typically, the angle is set to 60 degrees to form an equilateral triangle.

a. 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).
OR

b. Write OpenGL program to draw Sun Rise and Sunset.

→

1. What is OpenGL, and why is it used for graphical programming?

Answer:

OpenGL (Open Graphics Library) is a cross-platform, hardware-independent API used to render 2D and 3D vector graphics. It provides a set of functions for drawing geometric shapes and manipulating their properties, such as color, transformation, and texture. OpenGL is widely used in games, simulations, and other graphical applications because of its powerful features and flexibility in handling complex rendering tasks.

---

2. Can you explain the transformations you performed (Scaling, Translation, and Rotation) on the 3D Cube in the OpenGL program?

Answer:

- **Scaling**: Scaling involves changing the size of an object. In the case of the 3D cube, scaling increases or decreases its size along the X, Y, and Z axes. This can be done using the `glScalef()` function, where you specify the scaling factors for each axis.
- **Translation**: Translation moves an object from one position to another without changing its shape or orientation. The `glTranslatef()` function is used to move the cube along the X, Y, or Z axes.
- **Rotation**: Rotation involves rotating an object around a specific axis. Using the `glRotatef()` function, you can rotate the cube around the X, Y, or Z axis by a certain angle. The rotation is performed in a counterclockwise direction by default.

Q7. Write C++ program to draw man walking in the rain with an umbrella. Apply the concept of
polymorphism.

1. What is the concept of polymorphism in C++?

Answer:

Polymorphism in C++ is the ability of an object to take on many forms. It allows one interface to be used for different data types. In this code, polymorphism is applied when the drawing of the man and umbrella, as well as the rain simulation, can be extended or modified by creating different versions of the drawing functions (e.g., different colors or shapes for the man or umbrella).

2. What is the role of `X11/Xlib.h` in the program?

Answer:

The `X11/Xlib.h` library provides the necessary functions for working with the X Window System, which is a windowing system for bitmap displays. This library allows the program to create windows, handle events, and draw graphics like lines, circles, and arcs on the screen. In this code, `X11/Xlib.h` is used to create the display window, draw the man and umbrella, and simulate the rain.

3. Explain the use of `XFillArc` and `XDrawLine` functions.

Answer:

- `XFillArc`: This function is used to draw filled arcs or pieslices. In this program, it is used to draw the head of the man and the umbrella.
- `XDrawLine`: This function is used to draw straight lines between two points. It is used to draw the body, arms, legs, and the umbrella's stick in this program.

OOP

Q.10 Write C++ program using STL for sorting and searching user defined records such as
personal records (Name, DOB, Telephone number etc) using vector container.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>

using namespace std;

// Structure to store personal records
struct PersonalRecord {
    string name;
    string dob;
    string telephone;

    // Display record
    void display() {
        cout << "Name: " << name << ", DOB: " << dob << ", Telep
    }
};

// Function to sort records by name
bool compareByName(PersonalRecord a, PersonalRecord b) {
    return a.name < b.name;
}

// Function to search for a record by name
void searchByName(vector<PersonalRecord> records, string name) {
    bool found = false;
    for (int i = 0; i < records.size(); i++) {
        if (records[i].name == name) {
            cout << "Record found:" << endl;
            records[i].display();
            found = true;
            break;
        }
    }
```

```
    if (!found) {
        cout << "Record not found." << endl;
    }
}

int main() {
    vector<PersonalRecord> records;
    int choice;

    do {
        // Menu options
        cout << "\nMenu:\n1. Add Record\n2. Display All Records\
        cout << "Enter your choice: ";
        cin >> choice;
        cin.ignore(); // to clear the newline character from the

        if (choice == 1) {
            PersonalRecord record;
            cout << "Enter Name: ";
            getline(cin, record.name);
            cout << "Enter Date of Birth (DD/MM/YYYY): ";
            getline(cin, record.dob);
            cout << "Enter Telephone Number: ";
            getline(cin, record.telephone);
            records.push_back(record);
            cout << "Record added successfully." << endl;
        }
        else if (choice == 2) {
            // Display all records
            cout << "\nAll Records:" << endl;
            for (int i = 0; i < records.size(); i++) {
                records[i].display();
            }
        }
        else if (choice == 3) {
            // Sort records by name
```

```
            sort(records.begin(), records.end(), compareByName);
            cout << "Records sorted by name." << endl;

            // Display sorted records
            cout << "\nSorted Records:" << endl;
            for (int i = 0; i < records.size(); i++) {
                records[i].display();
            }
        }
        else if (choice == 4) {
            // Search for a record by name
            string searchName;
            cout << "Enter name to search: ";
            getline(cin, searchName);
            searchByName(records, searchName);
        }
        else if (choice == 5) {
            cout << "Exiting program." << endl;
        }
        else {
            cout << "Invalid choice. Please try again." << endl;
        }
    } while (choice != 5);

    return 0;
}
```

Viva Questions

1. What is STL in C++?

- **Answer**: The Standard Template Library (STL) is a collection of template-based classes and functions in C++ that provide commonly used data structures (e.g., `vector`, `list`, `map`) and algorithms (e.g., `sort`, `find`). It promotes code reusability and efficiency.

2. What are the major components of STL?

- **Answer**: STL has three main components:
  1. **Containers**: Data structures like `vector`, `list`, `set`, `map`, etc.
  2. **Iterators**: Objects used to traverse through elements in containers.
  3. **Algorithms**: Predefined functions for operations like searching, sorting, and manipulating data.

3. Why is a `vector` used in the program?

- **Answer**: The `vector` is used to dynamically store a list of `PersonalRecord` objects. It allows dynamic resizing as new records are added, which is more flexible compared to a static array.

Q. 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.

dont use &

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

class Complex {
private:
    double real;
    double imag;

public:
```

```
    // Default constructor
    Complex() {
        real = 0;
        imag = 0;
    }

    // Parameterized constructor
    Complex(double r, double i) {
        real = r;
        imag = i;
    }

    // Overload operator+ to add two complex numbers
    Complex operator+(Complex c) {
        Complex result;
        result.real = real + c.real;
        result.imag = imag + c.imag;
        return result;
    }

    // Overload operator* to multiply two complex numbers
    Complex operator*(Complex c) {
        Complex result;
        result.real = real * c.real - imag * c.imag;
        result.imag = real * c.imag + imag * c.real;
        return result;
    }

    // Overload operator >> to read a complex number
    friend istream& operator>>(istream& input, Complex& c) {
        cout << "Enter real part: ";
        input >> c.real;
        cout << "Enter imaginary part: ";
        input >> c.imag;
        return input;
    }
```

```
    // Overload operator << to print a complex number
    friend ostream& operator<<(ostream& output, Complex c) {
        output << c.real;
        if (c.imag >= 0)
            output << " + " << c.imag << "i";
        else
            output << " - " << -c.imag << "i";
        return output;
    }
};

int main() {
    Complex c1, c2, sum, product;

    cout << "Enter first complex number:\n";
    cin >> c1;

    cout << "Enter second complex number:\n";
    cin >> c2;

    // Add two complex numbers
    sum = c1 + c2;

    // Multiply two complex numbers
    product = c1 * c2;

    cout << "\nFirst Complex Number: " << c1 << endl;
    cout << "Second Complex Number: " << c2 << endl;

    cout << "Sum: " << sum << endl;
    cout << "Product: " << product << endl;

    return 0;
}
```

1. What is Operator Overloading?

- **Answer**: Operator overloading is a feature in C++ that allows operators to be redefined and used in a way that is specific to user-defined types (like classes). It allows custom implementations of operators such as `+`, `-`, `*`, `<<`, `>>`, etc., to work with objects of a class. In this program, operators `+`, `*`, `>>`, and `<<` are overloaded to perform operations on `Complex` numbers.

---

2. Why are the `>>` and `<<` operators overloaded as friend functions?

- **Answer**: The `>>` and `<<` operators are overloaded as friend functions because they need access to the private members (`real` and `imag`) of the `Complex` class. Friend functions can access private and protected members of a class. Since these operators are used to input and output data for objects of `Complex`, it is necessary to define them as friend functions to directly access and manipulate the data inside the class.

---

3. Explain how the `operator+` works in this program.

- **Answer**: The `operator+` is overloaded to add two `Complex` numbers. It takes another `Complex` object as a parameter, adds the real and imaginary parts separately, and returns a new `Complex` object that represents the sum of the two numbers. For instance:
  - `result.real = real + c.real;`
  - `result.imag = imag + c.imag;`
    The
    `+` operator allows for easy addition of two `Complex` numbers in the main program.

---

4. What is the purpose of overloading the `operator*` in this program?

- **Answer**: The `operator*` is overloaded to multiply two `Complex` numbers. The formula for multiplying two complex numbers `(a + bi)` and `(c + di)` is: $(a+bi)*(c+di)=(ac-bd)+(ad+bc)i$

In the program,
`operator*` calculates the product by:

$(a+bi)*(c+di)=(ac−bd)+(ad+bc)i(a + bi) * (c + di) = (ac - bd) + (ad + bc)i$

- `result.real = real * c.real - imag * c.imag;`
- `result.imag = real * c.imag + imag * c.real;`
  The operator allows easy multiplication of two `Complex` numbers in the main program.

---

5. What is the use of the `friend` keyword in this program?

- **Answer**: The `friend` keyword is used to grant non-member functions access to the private and protected members of a class. In this program, the `operator>>` and `operator<<` are defined as friend functions because they need to directly access the `real` and `imag` variables of `Complex` objects to input and output the data. Friend functions are not member functions but can still access private and protected data of the class.

---

6. Why is the `istream` and `ostream` used in the overloaded operators?

- **Answer**: `istream` and `ostream` are used in the overloaded `>>` and `<<` operators to handle input and output operations for the `Complex` class.
  - `istream& operator>>(istream& input, Complex& c)` allows reading the `real` and `imag` parts of a complex number from the user input.
  - `ostream& operator<<(ostream& output, Complex c)` allows printing the `Complex` number to the console in a formatted manner.
    These are standard C++ stream classes used to perform input and output operations.

Q. 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.

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

class Admin; // Forward declaration for friend class

class Student {
    string name, dob, bloodGroup, address, license, className, t
    char division;
    int roll;
    static int count; // Static member for keeping track of stud

public:
    // Default constructor
    Student() : name(""), roll(0), className(""), division('A'),
                address(""), license(""), telephoneNumber(""), c
        count++;  // Increment count when a student object is cr
    }

    // Copy constructor
    Student(const Student &s) {
        *this = s; // Using this pointer
        count++;   // Increment count when a new student is crea
    }

    // Destructor
    ~Student() {
        count--;  // Decrement count when a student object is de
    }

    // Inline function to display student details
```

```
    inline void display() const {
        cout << "Name: " << name << "\nRoll: " << roll << "\nCla
             << "\nDivision: " << division << "\nDOB: " << dob <
             << bloodGroup << "\nAddress: " << address << "\nTel
             << "\nLicense: " << license << "\nOther: " << other
    }

    // Static member function to get the student count
    static int getCount() { return count; }

    // Friend function to modify student details
    friend void modifyStudent(Student &s, int rollNo);

    // Friend class to access private members of Student
    friend class Admin;

    // Function to input student details from the user
    void inputDetails() {
        cout << "Enter Name: ";
        cin >> name;  // Input name
        cout << "Enter Roll: ";
        cin >> roll;  // Input roll

        cout << "Enter Class: ";
        cin >> className;  // Input class

        cout << "Enter Division (A/B/C): ";
        cin >> division;  // Input division

        cout << "Enter DOB: ";
        cin >> dob;  // Input DOB

        cout << "Enter Blood Group: ";
        cin >> bloodGroup;  // Input blood group

        cout << "Enter Address: ";
```

```
        cin >> address;  // Input address

        cout << "Enter Telephone: ";
        cin >> telephoneNumber;  // Input telephone

        cout << "Enter License: ";
        cin >> license;  // Input license

        cout << "Enter Other details: ";
        cin >> other;  // Input other details
    }
};

int Student::count = 0; // Static member initialization

// Friend function definition to modify student based on roll nu
void modifyStudent(Student &s, int rollNo) {
    if (s.roll == rollNo) {
        cout << "Enter New Name and Roll: ";
        cin >> s.name >> s.roll;  // Modify student's name and r
    } else {
        cout << "No student found with roll number " << rollNo <
    }
}

// Friend class to demonstrate access to private members
class Admin {
public:
    // Method to update division using friend access
    void updateDivision(Student &s, char newDivision) {
        s.division = newDivision;
    }
};

int main() {
    Student* student = new Student();  // Dynamically allocate m
```

```
    Admin admin;  // Admin object to modify student data
    int choice;
    bool isStudentInitialized = false;

    do {
        cout << "\n--- Menu ---\n";
        cout << "1. Add Student\n2. Display Student\n3. Modify S
        cout << "Enter choice: ";
        cin >> choice;

        try {
            if (choice == 1) {
                student->inputDetails();  // Input student detai
                isStudentInitialized = true;
            }
            else if (choice == 2 && isStudentInitialized) {
                student->display();  // Display student details
            }
            else if (choice == 3 && isStudentInitialized) {
                int rollNo;
                cout << "Enter Roll No to modify: ";
                cin >> rollNo;  // Get the roll number for modif
                modifyStudent(*student, rollNo);  // Modify stud
            }
            else if (choice == 4 && isStudentInitialized) {
                char newDiv;
                cout << "Enter new division: ";
                cin >> newDiv;
                admin.updateDivision(*student, newDiv);  // Upda
            }
            else if (choice == 5) {
                cout << "Exiting.\n";  // Exit the menu loop
            }
            else {
                cout << "Invalid input or no student data.\n";
            }
```

```
        } catch (const exception &e) {
            cout << "An error occurred: " << e.what() << "\n";
        }

    } while (choice != 5);  // Repeat the menu until 'Exit' is s

    delete student;  // Deallocate memory for the dynamically al
    return 0;
}
```

1. What is the purpose of the `count` variable in the `Student` class?

Answer:
The
`count` variable is a static member of the `Student` class that keeps track of the total number of `Student` objects created. It is incremented in the constructor whenever a new object is created and decremented in the destructor whenever an object is destroyed. This ensures that `count` always reflects the number of active `Student` objects.

2. What is the significance of the `friend` keyword in C++?

Answer:
The
`friend` keyword allows a function or a class to access the private and protected members of another class. In this program, `modifyStudent` is a friend function, and `Admin` is a friend class of the `Student` class. This means that the `modifyStudent` function and `Admin` class can directly access and modify the private members of the `Student` class, such as `name`, `roll`, and `division`.

3. How does dynamic memory allocation work in this program?

Answer:
Dynamic memory allocation is used when a new
`Student` object is created via the `new` operator. This means that memory is

allocated at runtime for the `Student` object. In the program, `student = new Student();` dynamically allocates memory for a new `Student` object, and `delete student;` is used to deallocate the memory when it's no longer needed, ensuring proper memory management.

4. What is the purpose of the copy constructor in the `Student` class?

Answer:
The copy constructor is used to create a new `Student` object as a copy of an existing `Student` object. In this program, the copy constructor copies the data from one `Student` object to another, and the static `count` variable is incremented to reflect the new object. The constructor is called when a `Student` object is passed by value or returned by value.

5. What is the role of the `inline` keyword in the `display` function?

Answer:
The `inline` keyword is used to suggest to the compiler to replace the function call with the actual code of the function, thus eliminating the overhead of a function call. This is typically used for small functions like `display()`, where the overhead of calling the function is considered unnecessary. However, the compiler can ignore the `inline` suggestion if it deems the function too complex.

6. Explain the working of the `static` member `count` in the `Student` class.

Answer:
The `static` keyword makes the `count` variable shared among all instances of the `Student` class, rather than having a separate copy for each object. It means that `count` retains its value across all instances and is incremented or decremented when objects are created or destroyed. This allows the program to keep track of the number of `Student` objects in existence, regardless of the specific object.

7. Why is the `Student` class's destructor important in this program?

Answer:
The destructor is important because it ensures that when a `Student` object is destroyed (either when it goes out of scope or is explicitly deleted), the `count` variable is decremented. This prevents memory leaks and ensures that the student count reflects the actual number of active `Student` objects.

Q.

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.

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

// Base class Publication
class Publication {
protected:
    string title;
    float price;

public:
    Publication() : title(""), price(0.0) {}

    void setData() {
        // Input data
        cout << "Enter title: ";
```

```
        cin >> title;  // Using cin to input string (without spa

        cout << "Enter price: ";
        cin >> price;  // Using cin to input float

        // Error handling
        if (title.empty()) {
            throw "Title cannot be empty";  // Throwing an error
        }
        if (price < 0) {
            throw "Price cannot be negative";  // Throwing an er
        }
    }

    void display() const {
        cout << "Title: " << title << ", Price: " << price << er
    }
};

// Derived class Book
class Book : public Publication {
private:
    int pageCount;

public:
    Book() : pageCount(0) {}

    void setData() {
        Publication::setData();
        // Input data
        cout << "Enter page count: ";
        cin >> pageCount;

        // Error handling
        if (pageCount < 0) {
            throw "Page count cannot be negative";  // Throwing
```

```
        }
    }

    void display() const {
        Publication::display();
        cout << "Page Count: " << pageCount << endl;
    }
};

// Derived class Tape
class Tape : public Publication {
private:
    float playingTime;

public:
    Tape() : playingTime(0.0) {}

    void setData() {
        Publication::setData();
        // Input data
        cout << "Enter playing time (in minutes): ";
        cin >> playingTime;

        // Error handling
        if (playingTime < 0) {
            throw "Playing time cannot be negative";  // Throwin
        }
    }

    void display() const {
        Publication::display();
        cout << "Playing Time: " << playingTime << " minutes" <<
    }
};

int main() {
```

```
    Book book;
    Tape tape;

    int choice;
    bool bookAdded = false, tapeAdded = false;

    do {
        // Displaying menu
        cout << "\n--- Menu ---\n";
        cout << "1. Add Book Details\n";
        cout << "2. Add Tape Details\n";
        cout << "3. Display Book Details\n";
        cout << "4. Display Tape Details\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        try {
            if (choice == 1) {
                cout << "\nEnter Book Details:\n";
                book.setData();  // Input book data
                bookAdded = true;  // Mark book data as added
            }
            else if (choice == 2) {
                cout << "\nEnter Tape Details:\n";
                tape.setData();  // Input tape data
                tapeAdded = true;  // Mark tape data as added
            }
            else if (choice == 3 && bookAdded) {
                cout << "\nBook Details:\n";
                book.display();  // Display book data
            }
            else if (choice == 4 && tapeAdded) {
                cout << "\nTape Details:\n";
                tape.display();  // Display tape data
            }
```

```
            else if (choice == 3 && !bookAdded) {
                cout << "Book details not entered yet.\n";
            }
            else if (choice == 4 && !tapeAdded) {
                cout << "Tape details not entered yet.\n";
            }
            else if (choice == 5) {
                cout << "Exiting program.\n";  // Exit the menu
            }
            else {
                cout << "Invalid choice, please try again.\n";
            }
        }
        catch (const char* msg) {
            // Exception handling
            cout << "Error: " << msg << endl;
            // Resetting data members to default zero values
            book = Book();  // Reset book data to default values
            tape = Tape();  // Reset tape data to default values
            cout << "Data has been reset due to error.\n";
        }

    } while (choice != 5);  // Repeat the menu until 'Exit' is s

    return 0;
}
```

General Concepts:

1. **What is inheritance in C++ and how is it demonstrated in this program?**
    - **Answer:** Inheritance is a mechanism in C++ that allows a class (derived class) to inherit properties and behaviors (members and functions) from another class (base class). In this program, `Book` and `Tape` are derived from the `Publication` class, meaning they inherit the `title` and `price` attributes and the `setData` and `display` functions.

2. **Can you explain the purpose of constructors in this program?**
    - **Answer:** Constructors are used to initialize objects when they are created. In this program, constructors are used to initialize the attributes (`title`, `price`, etc.) to default values like empty string or zero. This ensures that when a new `Book` or `Tape` object is created, it starts with known values.
3. **What is polymorphism, and is it used in this program?**
    - **Answer:** Polymorphism allows objects of different classes to be treated as objects of a common base class. In this program, polymorphism is not explicitly demonstrated because there is no use of virtual functions or method overriding. However, polymorphism could be introduced by making the `setData` and `display` methods virtual, allowing different behavior based on the actual object type.
4. **What is the significance of the `try-catch` block in this program?**
    - **Answer:** The `try-catch` block is used for exception handling. It allows the program to catch and handle exceptions (like invalid input) gracefully without crashing. If an error occurs while entering data (e.g., a negative price or invalid page count), the exception is caught, and the program resets the data to default values.
5. **What is the purpose of the `throw` statement in this code?**
    - **Answer:** The `throw` statement is used to raise an exception when an error condition is met. For example, if the user enters a negative price or an invalid page count, the `throw` statement triggers an exception with an error message, which is later caught in the `catch` block.

Q.5

Write a function template for selection sort that inputs, sorts and outputs an integer array and a float array.

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

// Template function for selection sort
```

```
template <typename T>
void selectionSort(T arr[], int size) {
    for (int i = 0; i < size - 1; ++i) {
        int minIndex = i;
        for (int j = i + 1; j < size; ++j) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        // Swap the elements
        T temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}

// Template function to display an array
template <typename T>
void displayArray(T arr[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int choice;
    cout << "Selection Sort Program\n";
    cout << "1. Integer Array\n";
    cout << "2. Float Array\n";
    cout << "Enter your choice: ";
    cin >> choice;

    if (choice == 1) {
        // Input integer array
        int size;
```

```
        cout << "Enter the size of the integer array: ";
        cin >> size;

        int intArr[size];
        cout << "Enter " << size << " integers: ";
        for (int i = 0; i < size; ++i) {
            cin >> intArr[i];
        }

        cout << "Original Integer Array: ";
        displayArray(intArr, size);

        selectionSort(intArr, size);

        cout << "Sorted Integer Array: ";
        displayArray(intArr, size);

    } else if (choice == 2) {
        // Input float array
        int size;
        cout << "Enter the size of the float array: ";
        cin >> size;

        float floatArr[size];
        cout << "Enter " << size << " floats: ";
        for (int i = 0; i < size; ++i) {
            cin >> floatArr[i];
        }

        cout << "Original Float Array: ";
        displayArray(floatArr, size);

        selectionSort(floatArr, size);

        cout << "Sorted Float Array: ";
        displayArray(floatArr, size);
```

```
    } else {
        cout << "Invalid choice! Exiting program." << endl;
    }

    return 0;
}
```

Viva Questions and Answers Related to Selection Sort and Templates

General Questions on Templates

1. **What is a function template?**
    - A function template is a blueprint for creating functions that can work with different data types without rewriting the code for each type. It allows code reusability.
2. **Why do we use templates in C++?**
    - Templates are used to achieve generic programming. They allow functions or classes to work with any data type, improving code flexibility and reducing redundancy.

Questions on Selection Sort

1. **What is selection sort?**
    - Selection sort is a simple sorting algorithm that repeatedly finds the smallest (or largest) element from the unsorted portion and moves it to the sorted portion.
2. **What is the time complexity of selection sort?**
    - The time complexity is:
        - Best case: O(n2)

          O(n2)O(n^2)
        - Worst case: O(n2)

O(n2)O(n^2)

- Average case: O(n2)

  O(n2)O(n^2)

3. **What is the space complexity of selection sort?**
   - The space complexity is O(1) because it is an in-place sorting algorithm and does not require extra space.

     O(1)O(1)
4. **Why is selection sort not preferred for large datasets?**
   - Selection sort has a time complexity of O(n2), making it inefficient for large datasets compared to more advanced algorithms like quicksort or mergesort.

     O(n2)O(n^2)

Q.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.

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

int main() {
    // Create and write to an output file
    ofstream outFile("data.txt");
    if (!outFile) {
        cout << "Error opening file for writing!" << endl;
        return 1;
    }
```

```
    // Writing sample information to the file
    outFile << "Title: C++ Programming\n";
    outFile << "Author: John Doe\n";
    outFile << "Year: 2024\n";
    outFile.close();  // Close the output file after writing

    cout << "Data written to the file successfully." << endl;

    // Open the file again as an input file to read from it
    ifstream inFile("data.txt");
    if (!inFile) {
        cout << "Error opening file for reading!" << endl;
        return 1;
    }

    // Reading and displaying the content from the file
    string content;
    cout << "\nFile Content: \n";
    while (getline(inFile, content)) {
        cout << content << endl;
    }

    inFile.close();  // Close the input file after reading

    return 0;
}
```

General Questions:

1. **What is the purpose of using `ofstream` and `ifstream` in this program?**
    - **Answer**: `ofstream` is used for writing data to a file, while `ifstream` is used for reading data from a file.
2. **What is the role of the `ofstream` and `ifstream` constructors in this program?**

- **Answer**: The constructors for `ofstream` and `ifstream` open the respective files. `ofstream outFile("data.txt")` opens the file `data.txt` for writing, and `ifstream inFile("data.txt")` opens it for reading.

3. **How does the program handle file opening errors?**
    - **Answer**: The program checks if the file is opened successfully by testing the stream objects (`outFile` and `inFile`) using `if (!outFile)` and `if (!inFile)`. If the file cannot be opened, an error message is printed, and the program exits with a return value of `1`.
4. **Why is it necessary to call `close()` after writing and reading from the file?**
    - **Answer**: Calling `close()` ensures that the file is properly closed after reading or writing. It frees up resources and ensures that all data is written to the file before the program exits.
5. **What is the significance of `getline()` function in this program?**
    - **Answer**: The `getline()` function reads an entire line from the file and stores it in the string variable `content`. It is used here to read the file content line by line and display it on the console.

Q.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.

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

```
int main() {
    // Create a map to store state names and their populations
    map<string, int> statePopulation;

    // Insert some states and their populations into the map
    statePopulation["California"] = 39538223;
    statePopulation["Texas"] = 29145505;
    statePopulation["Florida"] = 21538187;
    statePopulation["New York"] = 20201249;
    statePopulation["Pennsylvania"] = 13002700;

    string state;
    cout << "Enter the name of a state: ";
    cin >> state;

    // Search for the state in the map
    auto it = statePopulation.find(state);
    if (it != statePopulation.end()) {
        cout << "The population of " << state << " is " << it->s
    } else {
        cout << "State not found in the database." << endl;
    }

    return 0;
}
```

Viva Questions and Answers:

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

Answer:

An associative container in C++ stores elements in a way that allows quick retrieval using keys. Examples include `std::map`, `std::set`, `std::multimap`, and `std::multiset`. The `std::map` stores key-value pairs, ensuring that keys are unique and sorted.

2. What is the difference between `std::map` and `std::unordered_map`?

Answer:

- `std::map` is implemented as a balanced binary search tree (like a red-black tree). It keeps keys sorted and allows ordered traversal.
- `std::unordered_map` is implemented using a hash table. It doesn't maintain order but provides faster average access time for retrieval.

---

3. How does the `find()` method work in a map?

Answer:

The `find()` method searches for a key in the map. If the key is found, it returns an iterator pointing to the key-value pair. If the key is not found, it returns an iterator to `map.end()`.

---

4. What happens if you try to access a key that doesn't exist in a map using the subscript operator (`[]`)?

Answer:

If you use the subscript operator with a key that doesn't exist, the map will create a new key-value pair with the given key and a default value (e.g., `0` for integers, an empty string for strings).

---

5. Can two keys in a map have the same value?

Answer:

Yes, two keys in a map can have the same value. However, the keys themselves must be unique.

---

6. What is the time complexity of inserting and searching in a `std::map`?

Answer:

Both insertion and search in a `std::map` have a time complexity of O(logn)O(\log n)O(logn), where nnn is the number of elements in the map.