**Question#1**

#include<iostream>

using namespace std;

class Quads {

private:

float s;

float s1;

float s2;

float s3;

public:

Quads() {

s = 0;

s1 = 0;;

s2 = 0;

s3 = 0;

}

void setS(float S) {

s = S;

}

void setS1(float S1) {

s1 = S1;

}

void setS2(float S2) {

s2 = S2;

}

void setS3(float S3) {

s3 = S3;

}

float getS() {

return s;

}

float getS1() {

return s1;

}

float getS2() {

return s2;

}

float getS3() {

return s3;

}

virtual float Area() = 0;

};

class Rectangle : public Quads {

public:

float Area() {

return getS()\*getS1() ;

}

};

float Area(Quads& obj) {

float area = obj.Area();

return area;

}

int main() {

Rectangle obj;

float s = 2, s1 = 2;

obj.setS(s);

obj.setS1(s1);

obj.setS2(s);

obj.setS3(s1);

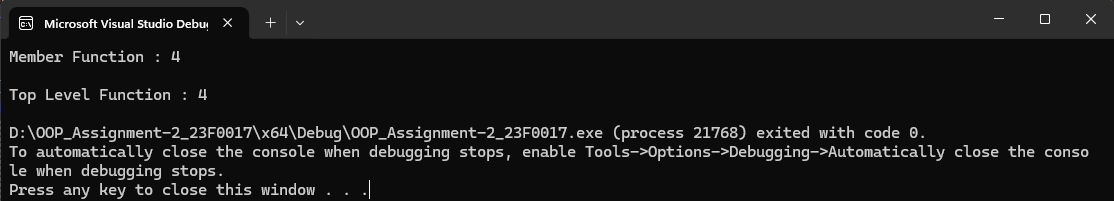
cout <<"Member Function : " << obj.Area() << endl;

cout << endl;

cout << "Top Level Function : " << Area(obj) << endl;

return 0;

}

****

**Question#2**

#include<iostream>

#include<string>

using namespace std;

class Furniture {

private:

string type;

string size;

double height;

public:

Furniture() {

type = " ";

size = " ";

height = 0;

}

void setType(string t) {

type = t;

}

void setSize(string s) {

size = s;

}

void setHeight(double h) {

height = h;

}

string getType() {

return type;

}

string getSize() {

return size;

}

double getHeight() {

return height;

}

virtual void printDescription() = 0;

};

class Bed :public Furniture {

public:

Bed(string s) {

setType("Bed");

setSize(s);

}

void printDescription() {

cout << "Type : " << getType() << endl;

cout << "Size : " << getSize() << endl;

}

};

class Chairs : public Furniture {

public:

Chairs(string s) {

setType("Chair");

setSize(s);

}

Chairs(string s,double h) {

setType("Chair");

setSize(s);

setHeight(h);

}

void printDescription() {

cout << "Type : " << getType() << endl;

cout << "Size : " << getSize() << endl;

cout << "Height : " << getHeight() << endl;

}

};

class Personalized :public Furniture {

private:

string name;

public:

Personalized(string s, double h, string n) {

setType("Personalized Chair");

name = n;

setSize(s);

setHeight(h);

}

void printDescription() {

cout << "Name : " << name << endl;

cout << "Type : " << getType() << endl;

cout << "Size : " << getSize() << endl;

cout << "Height : " << getHeight() << endl;

}

};

int main() {

int size = 5;

Furniture\*\* Inventory = new Furniture \* [size];

Inventory[0] = new Bed("Small");

Inventory[1] = new Chairs("Medium");

Inventory[2] = new Chairs("Medium",3);

Inventory[3] = new Bed("Large");

Inventory[4] = new Personalized("Small", 2, "Ali");

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

Inventory[i]->printDescription();

cout << endl;

delete Inventory[i];

}

delete[] Inventory;

return 0;

}

**A computer screen shot of a black screen

Description automatically generated**

**Question#3**

#include<iostream>

#include<cstring>

using namespace std;

class Media {

protected:

char\* type;

public:

Media() {

type = nullptr;

}

Media(char str[]) {

type = new char[strlen(str) + 1];

for (int i = 0; i <= strlen(str);i++) {

type[i] = str[i];

}

}

~Media() {

delete[] type;

}

void setType(char str[]) {

type = new char[strlen(str) + 1];

for (int i = 0; i <= strlen(str);i++) {

type[i] = str[i];

}

}

virtual void display() = 0;

};

class Book :public Media {

private:

char\* authorName;

char\* ISBN;

public:

Book() {

authorName = nullptr;

ISBN = nullptr;

}

Book(char type[], char name[], char isbn[]) {

setType(type);

authorName = new char[strlen(name) + 1];

for (int i = 0; i <= strlen(name); i++) {

authorName[i] = name[i];

}

ISBN = new char[strlen(isbn) + 1];

for (int i = 0; i <= strlen(isbn); i++) {

ISBN[i] = isbn[i];

}

}

~Book() {

delete[] authorName;

delete[] ISBN;

}

void display() {

cout << "Type : " << type << endl;

cout << "Author : " << authorName << endl;

cout << "ISBN : " << ISBN << endl;

}

};

class Magazine :public Media {

private:

char\* monthName;

int year;

public:

Magazine() {

monthName = nullptr;

year = 0;

}

Magazine(char type[], char name[], int y) {

setType(type);

monthName = new char[strlen(name) + 1];

for (int i = 0; i <= strlen(name); i++) {

monthName[i] = name[i];

}

year = y;

}

~Magazine() {

delete[] monthName;

}

void display() {

cout << "Type : " << type << endl;

cout << "Month : " << monthName << endl;

cout << "Year : " << year << endl;

}

};

class CD :public Media {

private:

int capacity;

public:

CD() {

capacity = 0;

}

CD(char type[], int c) {

setType(type);

capacity = c;

}

void display() {

cout << "Type : " << type << endl;

cout << "Capacity : " << capacity << endl;

}

};

class Shelf {

private:

Media\*\* obj;

int currSize;

int maxSize;

public:

Shelf(int size) {

currSize = 0;

maxSize = size;

obj = new Media \* [maxSize];

}

~Shelf() {

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

delete obj[i];

}

delete[] obj;

}

void insert(Media\* item) {

if (currSize < maxSize) {

obj[currSize] = item;

currSize = currSize + 1;

}

else {

cout << "Cannot Add Item" << endl;

}

}

void displayContents() {

cout << "Items" <<endl;

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

obj[i]->display();

}

}

};

int main() {

int n = 0;

while (n <= 0) {

cout << "Enter Number of Items : "; cin >> n;

}

Shelf obj(n);

int choice = 0;

while (true) {

cout << "Enter 1 For Book" << endl;

cout << "Enter 2 For Magazine" << endl;

cout << "Enter 3 For CD" << endl;

cout << "Enter 4 to Display Contents" << endl;

cout << "Enter 5 to Exit" << endl;

cout << "Enter Your Choice : "; cin >> choice;

if (choice == 1) {

char authorName[50], ISBN[50];

char type[] = { "Book" };

cout << "Enter Author Name : "; cin >> authorName;

cout << "Enter ISBN : "; cin >> ISBN;

obj.insert(new Book(type, authorName, ISBN));

}

else if (choice == 2) {

char monthName[50];

int year = 0;

char type[] = { "Magazine" };

cout << "Enter Month : "; cin >> monthName;

cout << "Enter Year : "; cin >> year;

obj.insert(new Magazine(type, monthName, year));

}

else if (choice == 3) {

int capacity = 0;

char type[] = { "CD" };

cout << "Enter Capacity : "; cin >> capacity;

obj.insert(new CD(type, capacity));

}

else if (choice == 4) {

obj.displayContents();

}

else if (choice == 5) {

cout << "Exiting.........." << endl;

break;

}

else {

cout << "Invalid Input" << endl;

}

}

obj.~Shelf();

return 0;

}****

**Question#4**

#include<iostream>

#include<string>

using namespace std;

class Package {

protected:

string source;

string destination;

double weight;

public:

Package() {

source = " ";

destination = " ";

weight = 0;

}

virtual double comp\_charges() = 0;

};

class package :public Package {

public:

package(string s, string d, double w) {

source = s;

destination = d;

weight = w;

}

double comp\_charges() {

return weight \* 20;

}

};

class TwoD\_package :public Package {

private:

double extracharges;

public:

TwoD\_package(string s, string d, double w, double e) {

source = s;

destination = d;

weight = w;

extracharges = e;

}

double comp\_charges() {

return weight \* 20 + extracharges;

}

};

class Urgent\_Package :public Package {

private:

double extra\_percentage;

public:

Urgent\_Package(string s, string d, double w, double e) {

source = s;

destination = d;

weight = w;

extra\_percentage = e;

}

double comp\_charges() {

double temp = weight \* 20;

return temp + (temp \* extra\_percentage / 100);

}

};

int main() {

const int size = 5;

Package\*\* pkg=new Package\*[size];

pkg[0] = new package("Lahore", "Karachi", 20);

pkg[1] = new TwoD\_package("Lahore", "Islamabad", 35, 200);

pkg[2] = new Urgent\_Package("Karachi", "Lahore", 25, 10);

pkg[3] = new TwoD\_package("Karachi", "Islamabad", 30, 250);

pkg[4] = new Urgent\_Package("Karachi", "Peshawar", 40, 25);

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

cout << "Package charges: " << pkg[i]->comp\_charges() << endl;

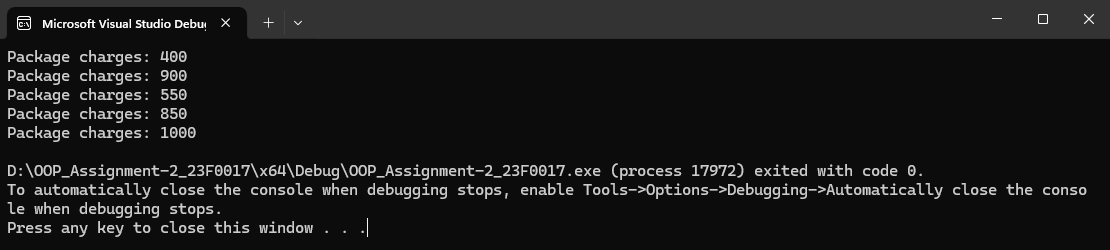
delete pkg[i];

}

delete[] pkg;

return 0;

}

****

**Question#5**

#include<iostream>

#include<cstdlib>

#include<ctime>

using namespace std;

class Matrix {

private:

int rows;

int cols;

int \*\*matrix;

public:

Matrix() {

rows = 0;

cols = 0;

matrix = nullptr;

}

Matrix(int row, int col) {

rows = row;

cols = col;

matrix = new int\*[rows];

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

matrix[i] = new int[cols];

}

}

//~Matrix() {

// for (int i = 0; i < rows; i++) {

// delete[] matrix[i];

// }

// delete[] matrix;

//}

void set() {

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

for (int j = 0; j < cols; j++) {

matrix[i][j] = rand() % 10;

}

}

}

Matrix operator+(Matrix& obj) {

if (rows != obj.rows || cols != obj.cols) {

cout << "Addition Not Possible" << endl;

return Matrix();

}

else {

Matrix temp(rows, cols);

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

for (int j = 0; j < cols; j++) {

temp.matrix[i][j] = matrix[i][j] + obj.matrix[i][j];

}

}

return temp;

}

}

Matrix operator-(Matrix& obj) {

if (rows != obj.rows || cols != obj.cols) {

cout << "Subtraction Not Possible" << endl;

return Matrix();

}

else {

Matrix temp(rows, cols);

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

for (int j = 0; j < cols; j++) {

temp.matrix[i][j] = matrix[i][j] - obj.matrix[i][j];

}

}

return temp;

}

}

Matrix operator\*(Matrix& obj){

if (cols != obj.rows || rows != obj.cols){

cout << "Multiplication Not Possible";

return Matrix();

}

else {

Matrix temp(rows, obj.cols);

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

for (int j = 0; j < obj.cols; j++) {

temp.matrix[i][j] = 0;

for (int k = 0; k < cols; k++) {

temp.matrix[i][j] = temp.matrix[i][j] + matrix[i][k] \* obj.matrix[k][j];

}

}

}

return temp;

}

}

void operator\*(int n) {

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

for (int j = 0; j < cols; j++) {

matrix[i][j] = n\*matrix[i][j];

}

}

}

friend void operator\*(int n,Matrix &obj) {

for (int i = 0; i < obj.rows; i++) {

for (int j = 0; j < obj.cols; j++) {

obj.matrix[i][j] = n \* obj.matrix[i][j];

}

}

}

void operator =(Matrix& obj) {

if (rows != obj.rows || cols != obj.cols) {

cout << "Assignment Not Possible" << endl;

}

else {

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

for (int j = 0; j < cols; j++) {

matrix[i][j] = obj.matrix[i][j];

}

}

}

}

void operator +=(Matrix& obj) {

if (rows != obj.rows || cols != obj.cols) {

cout << "Addition Not Possible" << endl;

}

else {

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

for (int j = 0; j < cols; j++) {

matrix[i][j] = matrix[i][j] + obj.matrix[i][j];

}

}

}

}

void operator -=(Matrix& obj) {

if (rows != obj.rows || cols != obj.cols) {

cout << "Subtraction Not Possible" << endl;

}

else {

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

for (int j = 0; j < cols; j++) {

matrix[i][j] = matrix[i][j] - obj.matrix[i][j];

}

}

}

}

void operator \*=(Matrix& obj) {

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

for (int j = 0; j < cols; j++) {

matrix[i][j] = matrix[i][j]\*obj.matrix[i][j];

}

}

}

bool operator ==(Matrix obj) {

if (rows != obj.rows || cols != obj.cols) {

return false;

}

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

for (int j = 0; j < cols; j++) {

if (matrix[i][j] != obj.matrix[i][j]) {

return false;

}

}

}

return true;

}

int\* operator[](int row) {

if (row < 0 || row >= rows) {

cout << "Access Violation" << endl;

return nullptr;

}

return matrix[row];

}

void display() {

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

for (int j = 0; j < cols; j++) {

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

};

int main() {

srand(time(0));

Matrix A(3,3);

Matrix B(3, 3);

A.set();

B.set();

cout << "Matrix A : " << endl;

A.display();

cout << "Matrix B : " << endl;

B.display();

Matrix temp = A + B;

cout << "A + B" << endl;

temp.display();

//temp.~Matrix();

Matrix temp1 = A - B;

cout << "A - B" << endl;

temp1.display();

temp.~Matrix();

cout << "A+=B" << endl;

A += B;

A.display();

cout << "A-=B" << endl;

A -= B;

A.display();

cout << "A\*=B" << endl;

A.display();

cout << "A=B" << endl;

A = B;

A.display();

bool flag = (A == B);

cout << "A==B : " << flag << endl;

Matrix obj = A \* B;

cout << "A\*B" << endl;

obj.display();

obj.~Matrix();

cout << "Matrix[3][3] : " << A[2][2] << endl;

A.~Matrix();

B.~Matrix();

return 0;

}

****

**Question#6**

#include<iostream>

using namespace std;

class SquareFoot;

class SquareYards;

class SquareMeters;

class SquareFoot {

protected:

double areaFoot;

public:

SquareFoot() {

areaFoot = 0;

}

SquareFoot(double a) {

areaFoot = a;

}

double getArea() {

return areaFoot;

}

SquareFoot operator +(SquareFoot& obj);

SquareFoot operator -(SquareFoot& obj);

SquareFoot operator +(SquareYards& obj);

SquareFoot operator -(SquareYards& obj);

SquareFoot operator +(SquareMeters& obj);

SquareFoot operator -(SquareMeters& obj);

SquareFoot operator +(float area);

SquareFoot operator -(float area);

};

class SquareYards {

protected:

double areaYards;

public:

SquareYards() {

areaYards = 0;

}

SquareYards(double a) {

areaYards = a;

}

double getArea() {

return areaYards;

}

SquareYards operator +(SquareYards& obj);

SquareYards operator -(SquareYards& obj);

SquareYards operator +(SquareFoot& obj);

SquareYards operator -(SquareFoot& obj);

SquareYards operator +(SquareMeters& obj);

SquareYards operator -(SquareMeters& obj);

SquareYards operator +(float area);

SquareYards operator -(float area);

};

class SquareMeters {

protected:

double areaMeters;

public:

SquareMeters() {

areaMeters = 0;

}

SquareMeters(double a) {

areaMeters = a;

}

double getArea() {

return areaMeters;

}

SquareMeters operator+(SquareMeters& obj);

SquareMeters operator-(SquareMeters& obj);

SquareMeters operator+(SquareYards& obj);

SquareMeters operator-(SquareYards& obj);

SquareMeters operator+(SquareFoot& obj);

SquareMeters operator-(SquareFoot& obj);

SquareMeters operator+(float area);

SquareMeters operator-(float area);

};

SquareFoot SquareFoot::operator +(SquareFoot& obj) {

SquareFoot temp;

temp.areaFoot = areaFoot + obj.areaFoot;

return temp;

}

SquareFoot SquareFoot::operator -(SquareFoot& obj) {

SquareFoot temp;

temp.areaFoot = areaFoot - obj.areaFoot;

return temp;

}

SquareFoot SquareFoot::operator +(SquareYards& obj) {

SquareFoot temp;

temp.areaFoot = areaFoot + (obj.getArea() \* 9);

return temp;

}

SquareFoot SquareFoot::operator -(SquareYards& obj) {

SquareFoot temp;

temp.areaFoot = areaFoot - (obj.getArea() \* 9);

return temp;

}

SquareFoot SquareFoot::operator +(SquareMeters& obj) {

SquareFoot temp;

temp.areaFoot = areaFoot + (obj.getArea() \* 10.76);

return temp;

}

SquareFoot SquareFoot::operator -(SquareMeters& obj) {

SquareFoot temp;

temp.areaFoot = areaFoot - (obj.getArea() \* 10.76);

return temp;

}

SquareFoot SquareFoot::operator +(float area) {

SquareFoot temp;

temp.areaFoot = areaFoot + area;

return temp;

}

SquareFoot SquareFoot::operator -(float area) {

SquareFoot temp;

temp.areaFoot = areaFoot - area;

return temp;

}

SquareYards SquareYards::operator +(SquareYards& obj) {

SquareYards temp;

temp.areaYards = areaYards + obj.getArea();

return temp;

}

SquareYards SquareYards::operator -(SquareYards& obj) {

SquareYards temp;

temp.areaYards = areaYards - obj.getArea();

return temp;

}

SquareYards SquareYards::operator +(SquareFoot& obj) {

SquareYards temp;

temp.areaYards = areaYards + (obj.getArea() \* 0.111);

return temp;

}

SquareYards SquareYards::operator -(SquareFoot& obj) {

SquareYards temp;

temp.areaYards = areaYards - (obj.getArea() \* 0.111);

return temp;

}

SquareYards SquareYards::operator +(SquareMeters& obj) {

SquareYards temp;

temp.areaYards = areaYards + (obj.getArea() \* 1.195);

return temp;

}

SquareYards SquareYards::operator -(SquareMeters& obj) {

SquareYards temp;

temp.areaYards = areaYards - (obj.getArea() \* 1.195);

return temp;

}

SquareYards SquareYards::operator +(float area) {

SquareYards temp;

temp.areaYards = areaYards + area;

return temp;

}

SquareYards SquareYards::operator -(float area) {

SquareYards temp;

temp.areaYards = areaYards - area;

return temp;

}

SquareMeters SquareMeters::operator +(SquareMeters& obj) {

SquareMeters temp;

temp.areaMeters = areaMeters + obj.getArea();

return temp;

}

SquareMeters SquareMeters::operator -(SquareMeters& obj) {

SquareMeters temp;

temp.areaMeters = areaMeters - obj.getArea();

return temp;

}

SquareMeters SquareMeters::operator +(SquareYards& obj) {

SquareMeters temp;

temp.areaMeters = areaMeters + (obj.getArea() \* 0.836);

return temp;

}

SquareMeters SquareMeters::operator -(SquareYards& obj) {

SquareMeters temp;

temp.areaMeters = areaMeters - (obj.getArea() \* 0.836);

return temp;

}

SquareMeters SquareMeters::operator +(SquareFoot& obj) {

SquareMeters temp;

temp.areaMeters = areaMeters + (obj.getArea() \* 0.09);

return temp;

}

SquareMeters SquareMeters::operator -(SquareFoot& obj) {

SquareMeters temp;

temp.areaMeters = areaMeters - (obj.getArea() \* 0.09);

return temp;

}

SquareMeters SquareMeters::operator +(float area) {

SquareMeters temp;

temp.areaMeters = areaMeters + area;

return temp;

}

SquareMeters SquareMeters::operator -(float area) {

SquareMeters temp;

temp.areaMeters = areaMeters - area;

return temp;

}

int main() {

SquareFoot obj(1);

SquareYards obj1(1);

SquareMeters obj2(1);

float area = 1;

SquareFoot temp = obj + obj;

cout << "Square Foot + Square Foot : " << temp.getArea() << endl;

temp = obj - obj;

cout << "Square Foot - Square Foot : " << temp.getArea() << endl;

temp = obj + obj1;

cout << "Square Foot + Square Yards : " << temp.getArea() << endl;

temp = obj - obj1;

cout << "Square Foot - Square Yards : " << temp.getArea() << endl;

temp = obj + obj2;

cout << "Square Foot + Square Meters : " << temp.getArea() << endl;

temp = obj - obj2;

cout << "Square Foot - Square Meters : " << temp.getArea() << endl;

temp = obj + area;

cout << "Square Foot + Float : " << temp.getArea() << endl;

temp = obj - area;

cout << "Square Foot - Float : " << temp.getArea() << endl;

cout << endl;

SquareYards temp1 = obj1 + obj1;

cout << "Square Yards + Square Yards : " << temp1.getArea() << endl;

temp1 = obj1 - obj1;

cout << "Square Yards - Square Yards : " << temp1.getArea() << endl;

temp1 = obj1 + obj;

cout << "Square Yards + Square Foot : " << temp1.getArea() << endl;

temp1 = obj1 - obj;

cout << "Square Yards - Square Foot : " << temp1.getArea() << endl;

temp1 = obj1 + obj2;

cout << "Square Yards + Square Meters : " << temp1.getArea() << endl;

temp1 = obj1 - obj2;

cout << "Square Yards - Square Meters : " << temp1.getArea() << endl;

temp1 = obj1 + area;

cout << "Square Yards + Float : " << temp1.getArea() << endl;

temp1 = obj1 - area;

cout << "Square Yards - Float : " << temp1.getArea() << endl;

cout << endl;

SquareMeters temp2 = obj2 + obj2;

cout << "Square Meters + Square Meters : " << temp2.getArea() << endl;

temp2 = obj2 - obj2;

cout << "Square Meters - Square Meters : " << temp2.getArea() << endl;

temp2 = obj2 + obj1;

cout << "Square Meters + Square Yards : " << temp2.getArea() << endl;

temp2 = obj2 - obj1;

cout << "Square Meters - Square Yards : " << temp2.getArea() << endl;

temp2 = obj2 + obj;

cout << "Square Meters + Square Foot : " << temp2.getArea() << endl;

temp2 = obj2 - obj;

cout << "Square Meters - Square Foot : " << temp2.getArea() << endl;

temp2 = obj2 + area;

cout << "Square Meters + Float : " << temp2.getArea() << endl;

temp2 = obj2 - area;

cout << "Square Meters - Float : " << temp2.getArea() << endl;

return 0;

}

****