**9.1**

**UML Diagram**

|  |
| --- |
| **Rectangle** |
| - width : double  - height : double |
| + Rectangle()  + Rectangle(newWidth : double, newHeight : double)  + getWidth() : double  + getHeight() : double  + setWidth(newWidth : double) : void  + setHeight(newHeight : double) : void  + getArea() : double  + getPerimeter() : double |

**Header**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

class Rectangle

{

public:

Rectangle();

Rectangle(double newWidth, double newHeight);

double getWidth();

double getHeight();

void setWidth(double newWidth);

void setHeight(double newHeight);

double getArea();

double getPerimeter();

private:

double width;

double height;

};

#endif

**Implementation**

#include "rectangle.h"

Rectangle::Rectangle()

{

width = 1;

height = 1;

}

Rectangle::Rectangle(double newWidth, double newHeight)

{

width = newWidth;

height = newHeight;

}

double Rectangle::getWidth()

{

return width;

}

double Rectangle::getHeight()

{

return height;

}

void Rectangle::setWidth(double newWidth)

{

width = newWidth;

}

void Rectangle::setHeight(double newHeight)

{

height = newHeight;

}

double Rectangle::getArea()

{

return width \* height;

}

double Rectangle::getPerimeter()

{

return (2 \* width) + (2 \* height);

}

#include <iostream>

#include <iomanip>

#include "rectangle.h"

using namespace std;

int main()

{

// make a table to simplify display

cout << left << setw(14) << "Rectangle"

<< setw(10) << "Width" << setw(10) << "Height"

<< setw(8) << "Area" << "Perimeter\n";

// create the first object and display the results

Rectangle obj1;

obj1.setWidth(4);

obj1.setHeight(40);

cout << setw(14) << "1" << setw(10) << obj1.getWidth() << setw(10) << obj1.getHeight()

<< setw(8) << obj1.getArea() << obj1.getPerimeter() << endl;

// create the second object and display the results

Rectangle obj2(3.5, 35.9);

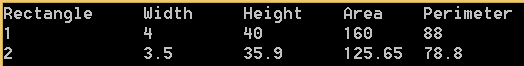
cout << setw(14) << "2" << setw(10) << obj2.getWidth() << setw(10) << obj2.getHeight()

<< setw(8) << obj2.getArea() << obj2.getPerimeter() << endl;

system("pause");

return 0;

}

****

**9.3**

**UML Diagram**

|  |
| --- |
| **Account** |
| - id : int  - balance : double  - annualInterestRate : double |
| + Account()  + Account(newId : int, newBalance : double, newRate : double)  + getId() : int  + setId(int newId) : void  + getBalance() : double  + setBalance(newBalance : double) : void  + getRate(): double  + setRate(newRate : double) : void  + getMonthlyInterestRate(): double  + withdraw(double amount) : void  + deposit(double amount) : void |

**Header**

#ifndef ACCOUNT\_H

#define ACCOUNT\_H

class Account

{

public:

Account();

Account(int newId, double newBalance, double newRate);

int getId();

void setId(int newId);

double getBalance();

void setBalance(double newBalance);

double getRate();

void setRate(double newRate);

double getMonthlyInterestRate();

void withdraw(double amount);

void deposit(double amount);

private:

int id;

double balance;

double annualInterestRate;

};

#endif

**Implementation**

#include "account.h"

Account::Account()

{

id = 0;

balance = 0;

annualInterestRate = 0;

}

Account::Account(int newId, double newBalance, double newRate)

{

id = newId;

balance = newBalance;

annualInterestRate = newRate;

}

int Account::getId()

{

return id;

}

void Account::setId(int newId)

{

id = newId;

}

double Account::getBalance()

{

return balance;

}

void Account::setBalance(double newBalance)

{

balance = newBalance;

}

double Account::getRate()

{

return annualInterestRate;

}

void Account::setRate(double newRate)

{

annualInterestRate = newRate;

}

double Account::getMonthlyInterestRate()

{

return annualInterestRate / 12;

}

void Account::withdraw(double amount)

{

balance -= amount;

}

void Account::deposit(double amount)

{

balance += amount;

}

**Main**

#include <iostream>

#include "account.h"

using namespace std;

int main()

{

// create an Account object with the properties

Account acc;

acc.setId(1122);

acc.setBalance(20000);

acc.setRate(4.5);

// run the withdraw and deposit functions, then display balance and monthly interest rate

acc.withdraw(2500);

acc.deposit(3000);

cout << "The balance is $" << acc.getBalance()

<< " and the monthly interest rate is " << acc.getMonthlyInterestRate() << "%.\n";

system("pause");

return 0;

}

****

**9.7**

**UML Diagram**

|  |
| --- |
| **StopWatch** |
| - startTime : long  - endTime : long |
| + StopWatch()  + getStartTime() : long  + getEndTime() : long  + start() : void  + stop() : void  + getElapsedTime() : long |

**Header**

#ifndef STOPWATCH\_H

#define STOPWATCH\_H

class StopWatch

{

public:

StopWatch();

long getStartTime();

long getEndTime();

void start();

void stop();

long getElapsedTime();

private:

long startTime;

long endTime;

};

#endif

**Implementation**

#include "stopwatch.h"

#include <ctime>

StopWatch::StopWatch()

{

startTime = time(0);

}

long StopWatch::getStartTime()

{

return startTime = time(0);

}

long StopWatch::getEndTime()

{

return endTime = time(0);

}

void StopWatch::start()

{

getStartTime();

}

void StopWatch::stop()

{

getEndTime();

}

long StopWatch::getElapsedTime()

{

return (endTime - startTime) \* 1000;

}

**Main**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include "stopwatch.h"

using namespace std;

int main()

{

// create the random 100000 numbers

srand(time(0));

const int NUM = 100000;

long a[NUM];

for (int i = 0; i < NUM; i++)

{

a[i] = rand() % 100001;

}

// create a StopWatch object and start the time

StopWatch b;

b.start();

// selection sort

for (int i = 0; i < NUM; i++)

{

long currentMin = a[i];

int currentMinIndex = i;

for (int j = i + 1; j < NUM; j++)

{

if (currentMin > a[j])

{

currentMin = a[j];

currentMinIndex = j;

}

}

if (currentMinIndex != i)

{

a[currentMinIndex] = a[i];

a[i] = currentMin;

}

}

// stop the time it takes to run the sort and display the elapsed time

b.stop();

cout << "The elapsed time is " << b.getElapsedTime() << " milliseconds\n";

system("pause");

return 0;

}

****

**10.2**

#include <iostream>

#include <string>

using namespace std;

// function that returns the common characters of two strings

string commonChars(const string& s1, const string&s2)

{

string s3;

int n;

// to avoid subscript being out of range if # of characters in s2 is longer than s1

if (s1.size() > s2.size())

n = s2.size();

else

n = s1.size();

// finding the common characters of the two strings

for (int i = 0; i < n; i++)

{

if (s1[i] == s2[i])

{

s3 += s1[i];

}

}

return s3;

}

int main()

{

// prompt user to enter 2 strings

string a, b;

string c;

cout << "Enter a string: ";

getline(cin, a);

cout << "Enter a string: ";

getline(cin, b);

// display the results

c = commonChars(a, b);

if (c.empty() == true)

cout << "No common characters\n";

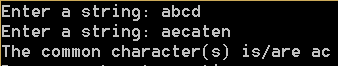
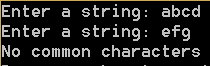
else

cout << "The common character(s) is/are " << c << endl;

system("pause");

return 0;

}

** **

**10.11**

**UML Diagram**

|  |
| --- |
| **Loan** |
| - annualInterestRate : double  - numberOfYears : int  - loanAmount : double |
| + Loan()  + Loan(rate : double, years: int, amount : double)  + getAnnualInterestRate() : double  + getNumberOfYears() : double  + getLoanAmount() : double  + setAnnualInterestRate(rate : double) : void  + setNumberOfYears(years: int) : void  + setLoanAmount(amount : double) : void  + getMonthlyPayment(rate : double, years: int, amount : double) : double  + getTotalPayment(rate : double, years: int, amount : double) : double |

**Header**

#ifndef LOAN\_H

#define LOAN\_H

class Loan

{

public:

Loan();

Loan(double rate, int years, double amount);

double getAnnualInterestRate();

double getNumberOfYears();

double getLoanAmount();

void setAnnualInterestRate(double rate);

void setNumberOfYears(int years);

void setLoanAmount(double amount);

static double getMonthlyPayment(double rate, int years, double amount);

static double getTotalPayment(double rate, int years, double amount);

private:

double annualInterestRate;

int numberOfYears;

double loanAmount;

};

#endif

**Implementation**

#include "loan.h"

#include <iostream>

using namespace std;

Loan::Loan()

{

annualInterestRate = 9.5;

numberOfYears = 30;

loanAmount = 100000;

}

Loan::Loan(double rate, int years, double amount)

{

annualInterestRate = rate;

numberOfYears = years;

loanAmount = amount;

}

double Loan::getAnnualInterestRate()

{

return annualInterestRate;

}

double Loan::getNumberOfYears()

{

return numberOfYears;

}

double Loan::getLoanAmount()

{

return loanAmount;

}

void Loan::setAnnualInterestRate(double rate)

{

annualInterestRate = rate;

}

void Loan::setNumberOfYears(int years)

{

numberOfYears = years;

}

void Loan::setLoanAmount(double amount)

{

loanAmount = amount;

}

double Loan::getMonthlyPayment(double rate, int years, double amount)

{

double monthlyInterestRate = rate / 1200;

return (amount \* monthlyInterestRate) /

(1 - (pow(1 / (1 + monthlyInterestRate), (years \* 12))));

}

double Loan::getTotalPayment(double rate, int years, double amount)

{

return getMonthlyPayment(rate, years, amount) \* years \* 12;

}

**Main**

#include "loan.h"

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

// Enter an annual interest rate

cout << "Enter yearly interest rate, for example 8.25: ";

double annualInterestRate;

cin >> annualInterestRate;

// Enter number of years

cout << "Enter number of years as an integer, for example 5: ";

double numberOfYears;

cin >> numberOfYears;

// Enter loan amount

cout << "Enter loan amount, for example 120000.95: ";

double loanAmount;

cin >> loanAmount;

// Display results

cout << fixed << setprecision(2);

cout << "The monthly payment is "

<< Loan::getMonthlyPayment(annualInterestRate, numberOfYears, loanAmount) << endl;

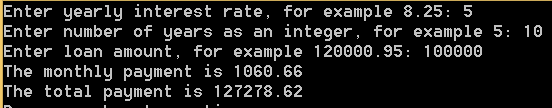
cout << "The total payment is "

<< Loan::getTotalPayment(annualInterestRate, numberOfYears, loanAmount) << endl;

system("pause");

return 0;

}

****

**10.13**

**UML Diagram**

|  |
| --- |
| **RegularPolygon** |
| - n : int  - side : double  - x : double  - y : double |
| + RegularPolygon()  + RegularPolygon(no : int, length : double, hor : double, ver : double)  + RegularPolygon(no : int, length : double)  + setN(no : int) : void  + getN() : int const  + setSide(length : double) : void  + getSide() : double const  + setX(hor : double) : void  + getX(): double const  + setY(ver : double) : void  + getY(): double const  + getPerimeter(): double const  + getArea(): double const |

**Header**

#ifndef POLYGON\_H

#define POLYGON\_H

class RegularPolygon

{

public:

RegularPolygon();

RegularPolygon(int no, double length, double hor, double ver);

RegularPolygon(int no, double length);

void setN(int no);

int getN() const;

void setSide(double length);

double getSide() const;

void setX(double hor);

double getX() const;

void setY(double ver);

double getY() const;

double getPerimeter() const;

double getArea() const;

private:

int n;

double side;

double x;

double y;

};

#endif

**Implementation**

#include "polygon.h"

#include <iostream>

#include <cmath>

using namespace std;

RegularPolygon::RegularPolygon()

{

n = 3;

side = 1;

x = 0;

y = 0;

}

RegularPolygon::RegularPolygon(int no, double length, double hor, double ver)

{

n = no;

side = length;

x = hor;

y = ver;

}

RegularPolygon::RegularPolygon(int no, double length)

{

n = no;

side = length;

x = 0;

y = 0;

}

void RegularPolygon::setN(int no)

{

n = no;

}

int RegularPolygon::getN() const

{

return n;

}

void RegularPolygon::setSide(double length)

{

side = length;

}

double RegularPolygon::getSide() const

{

return side;

}

void RegularPolygon::setX(double hor)

{

x = hor;

}

double RegularPolygon::getX() const

{

return x;

}

void RegularPolygon::setY(double ver)

{

y = ver;

}

double RegularPolygon::getY() const

{

return y;

}

double RegularPolygon::getPerimeter() const

{

return n \* side;

}

double RegularPolygon::getArea() const

{

return (n \* pow(side, 2)) / (4 \* (tan(3.14159 / n)));

}

**Main**

#include <iostream>

#include <iomanip>

#include "polygon.h"

using namespace std;

int main()

{

cout << fixed << setprecision(2);

cout << left << setw(10) << "Polygon" << setw(12) << "Perimeter" << "Area\n";

RegularPolygon p1;

cout << setw(10) << "1" << setw(12) << p1.getPerimeter() << p1.getArea() << endl;

RegularPolygon p2(6,4);

cout << setw(10) << "2" << setw(12) << p2.getPerimeter() << p2.getArea() << endl;

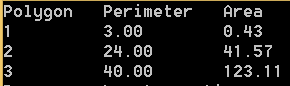
RegularPolygon p3(10, 4, 5.6, 7.8);

cout << setw(10) << "3" << setw(12) << p3.getPerimeter() << p3.getArea() << endl;

system("pause");

return 0;

}

****