RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

INTRODUCTION TO OOPs USING C++

CLASS: FYBSc CS

Mr. ARIF PATEL

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE

Practical: 01

Question: Write a program in C++ to display 'Hello World!' on the console/monitor

Code:

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

int main(){
	cout<<"\nHello World!\n";
	return 0;
}
```

Output:

```
Hello World!
```

Practical: 02

Question: Write a program in C++ which takes the first name from the user and displays Hello A, Hello Ab,.. each in a new line till the entire name is displayed.

Code:

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

int main(){
	string fname;
	cout<<"\nEnter your first name: ";
	cin>>fname;
	int len = fname.length();
	for(int i = 0; i < len; ++i){
		int limit = i;
		cout<<"\nHello ";
		for(int j = 0; j <= limit; ++j){
			cout<<fname[j];
		}
	}
	return 0;
}
```

Output:

```
Enter your first name: Stroustrup

Hello S
Hello St
Hello Str
Hello Stro
Hello Strou
Hello Strous
Hello Stroust
Hello Stroustr
Hello Stroustru
Hello Stroustrup
```

Practical: 03

Question: Write a program in C++ which takes an integer number greater than zero and prints its reverse.

Code:

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

int main(){
	int num, rev = 0;
	cout<<"\nPlease enter a positive integer number ";
	cin>>num;
	while(num){
		rev = rev * 10 + num%10;
		num /= 10;
	}
	cout<<"\nThe reverse of the given number is "<<rev<<endl;
	return 0;
}
```

Output:

```
Please enter a positive integer number 12345

The reverse of the given number is 54321
```

Practical: 04

Question: Write a program in C++ which accepts a positive integer number from the user displays whether it is a palindrome or not.

Code:

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

int main(){
	int num, temp, rev = 0;
	cout<<"\nPlease enter a positive integer number ";
	cin>>num;
	temp = num;
	while(temp){
		rev = rev * 10 + temp%10;
		temp /= 10;
	}
	if(num == rev){
		cout<<"\nThe given number is a palindrome"<<endl;
	}
	else{
		cout<<"\nThe given number is not a palindrome"<<endl;
	}
	return 0;
}
```

Output:

```
Please enter a positive integer number 1221

The given number is a palindrome

Please enter a positive integer number 1234

The given number is not a palindrome
```

Practical: 05

Question: Write a program in C++ which accepts a positive integer number from the user and displays the number in words of its digits.

Code:

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

int main(){
	int num, digit;
	string words ="";
	cout<<"\nPlease enter a positive integer number ";
	cin>>num;
	while(num){
		digit = num%10;
		switch(digit){
			case 0:	words = "ZERO " + words;
				break;
			case 1:	words = "ONE " + words;
				break;
			case 2:	words = "TWO " + words;
				break;
			case 3:	words = "THREE " + words;
				break;
			case 4:	words = "FOUR " + words;
				break;
			case 5:	words = "FIVE " + words;
				break;
			case 6:	words = "SIX " + words;
				break;
			case 7:	words = "SEVEN " + words;
				break;
			case 8:	words = "EIGHT " + words;
				break;
			case 9:	words = "NINE " + words;
				break;
		}
		num /= 10;
	}
	cout<<"\n"<<words<<endl;
	return 0;
}
```

Output:

```
Please enter a positive integer number 29387

TWO NINE THREE EIGHT SEVEN
```

Practical: 06

Question: Write a program in C++ which has a function 'isPrime' which takes one integer argument and returns true if the argument is a prime number, else returns false. Use this to display all prime numbers between two integers, which are taken from the user.

Code:

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

bool isPrime(int n){
	bool flag = true;
	int limit = static_cast<int>(sqrt(n));
	for(int divisor = 2; divisor<= limit; ++divisor){
		if(!(n%divisor)){
			flag = false;
			break;
		}
	}
	return flag;
}

int main(){
	int s, e;
	cout<<"\nEnter the first integer : ";
	cin>>s;
	cout<<"\nEnter the second integer : ";
	cin>>e;
	cout<<endl;

	for(int i = s; i <= e ; ++i){
		if(isPrime(i)){
			cout<<i<<endl;
		}
	}
	return 0;
}
```

Output:

```
Enter the first integer : 800

Enter the second integer : 850

809
811
821
823
827
829
839
```

Practical: 07

Question: Write a program in C++ which generates 500 random numbers between 0 and 9 and prints a horizontal frequency bar for the same.

Code:

```
#include <iostream>
#include <iomanip>
#include <ctime>
#include <cstdlib>
#define LIMIT 500
using namespace std;

int main(){
	int freq[10] = {0,0,0,0,0,0,0,0,0,0};
	char ch = '*';

	srand(time(NULL));
	for(int i = 0;i < LIMIT; ++i){
		++freq[rand()%10];
	}

	//Display the frequency
	for(int i = 0; i < 10; ++i){
		cout<<endl<<setw(2)<<left<<i<<setw(3)<<right<<freq[i]<<" | ";
		for(int j = 0; j < freq[i]; ++j){
			cout<<ch;
		}
	}
	return 0;
}
```

Output:

```
 0  37 | *************************************
 1  40 | ****************************************
 2  51 | ***************************************************
 3  48 | ************************************************
 4  56 | ********************************************************
 5  58 | **********************************************************
 6  60 | ************************************************************
 7  47 | ***********************************************
 8  49 | *************************************************
 9  54 | ******************************************************
```

Practical: 08

Question: Write an OOP in C++ to create a calculator class with the addition, subtraction, multiplication and division member functions.

Code:

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

class Calculator{
	private:
		double op1, op2;
	public:
		Calculator(double p1, double p2);
		double addition();
		double subtraction();
		double multiplication();
		double division();
};

Calculator::Calculator(double p1, double p2){
	op1 = p1;
	op2 = p2;
}

double Calculator::addition(){
	return op1 + op2;
}

double Calculator::subtraction(){
	return op1 - op2;
}

double Calculator::multiplication(){
	return op1 * op2;
}

double Calculator::division(){
	return op1 / op2;
}

int main(){
	Calculator c(5.0, 2.0);
	cout<<"\nAddition of 5.0 and 2.0 is "<<c.addition();
	cout<<"\nSubtraction of 5.0 and 2.0 is "<<c.subtraction();
	cout<<"\nMultiplication of 5.0 and 2.0 is "<<c.multiplication();
	cout<<"\nDivision of 5.0 and 2.0 is "<<c.division();
	return 0;
}
```

Output:

```
Addition of 5.0 and 2.0 is 7
Subtraction of 5.0 and 2.0 is 3
Multiplication of 5.0 and 2.0 is 10
Division of 5.0 and 2.0 is 2.5
```

Practical: 09

Question: Write a program in C++ to create a class to represent a circle. Define methods to return the circumference and area of a circle. Let the constructor assign a default value of 0.0 to the radius if the user does not specify it. Define a method display to display the radius of the circle. Create objects of this class and call its various functions.

Code:

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

class Circle{
            double r;
      public:
            Circle(double rr = 0.0);
            double circumference();
            double area();
            void display();
};

Circle::Circle(double rr){
      r = rr;
      cout<<"\nObject of Circle created\n";
}

double Circle::circumference(){
      return 2*3.142*r;
}

double Circle::area(){
      return 3.142*r*r;
}

void Circle::display(){
      cout<<"\nThe radius of the circle is "<<r;
}

int main(){
      Circle c1(4), c2(1), c3;
      c1.display();
      cout<<"\nThe circumference of the circle is "<<c1.circumference();
      cout<<"\nThe area of the circle is "<<c1.area()<<endl;
      c2.display();
      cout<<"\nThe circumference of the circle is "<<c2.circumference();
      cout<<"\nThe area of the circle is "<<c2.area()<<endl;
      c3.display();
      cout<<"\nThe circumference of the circle is "<<c3.circumference();
      cout<<"\nThe area of the circle is "<<c3.area()<<endl;
      return 0;
}
```

Output:

```
Object of Circle created

Object of Circle created

Object of Circle created

The radius of the circle is 4
The circumference of the circle is 25.136
The area of the circle is 50.272

The radius of the circle is 1
The circumference of the circle is 6.284
The area of the circle is 3.142

The radius of the circle is 0
The circumference of the circle is 0
The area of the circle is 0
```

Practical: 10

Question: Write an OOP in C++ program to calculate the area of circle, rectangle and square using the concept of method overloading.

Code:

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

class Area{
      public:
            double area(double r);
            double area(double s,char ch);
            double area(double l, double b);
};

double Area::area(double r){
      return 3.142 * r * r;
}

double Area::area(double s,char ch){
      return s * s;
}

double Area::area(double l, double b){
      return l * b;
}

int main(){
      Area a;
      cout<<"\nArea of circle of radius 5.0 is "<<a.area(5.0);
      cout<<"\nArea of square of side 4.0 is "<<a.area(4.0,'s');
      cout<<"\nArea of rectangle of length 7.0 and breadth 4.0 is
"<<a.area(7.0, 4.0);
      return 0;
}
```

Output:

```
Area of circle of radius 5.0 is 78.55
Area of square of side 4.0 is 16
Area of rectangle of length 7.0 and breadth 4.0 is  28
```

Practical: 11

Question: Write a program in C++ to represent a sphere. Provide overloaded constructors, one without parameter and other with the radius. The default constructor should set the radius to 0.0. Define a method 'display' to print the radius. Create arrays of objects of sphere and invoke their display methods.

Code:

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

class Sphere{
	private:
		double r;
	public:
		Sphere(double r){
			this->r=r;
			cout<<"\nObject created";
		}
		Sphere(){
			r=0.0;
			cout<<"\nObject created";
		}
		void display(){
			cout<<"\nRadius = "<<r;
		}
};

int main(){
	Sphere s[3] = {Sphere(33), Sphere(44), Sphere(55) };
	for(int i = 0; i < 3; ++i) {
		s[i].display();
	}

	Sphere s1[3] = {30, 40, 50};
	for(int i = 0; i < 3; ++i) {
		s1[i].display();
	}

	Sphere s2[3] = {70, 90};
	for(int i = 0; i < 3; ++i) {
		s2[i].display();
	}

	Sphere s3[3];
	for(int i = 0; i < 3; ++i) {
		s3[i].display();
	}
	return 0;
}
```

Output:

```
Object created
Object created
Object created
Radius = 33
Radius = 44
Radius = 55
Object created
Object created
Object created
Radius = 30
Radius = 40
Radius = 50
Object created
Object created
Object created
Radius = 70
Radius = 90
Radius = 0
Object created
Object created
Object created
Radius = 0
Radius = 0
Radius = 0
```

Practical: 12

Question: Write a program in C++ to create a class to represent a cylinder. Define methods to return the volume and surface area of a cylinder. Define a constructor to assign the radius and height of a cylinder. Create object of this class and call its various functions using a pointer to the object.

Code:

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

class Cylinder{
            double r;
            double h;
      public:
            Cylinder(double rr, double hh);
            double volume();
            double sarea();
            void display();
};

Cylinder::Cylinder(double rr, double hh) : r(rr), h(hh){
      cout<<"\nObject created";
}

double Cylinder::volume(){
      return 3.142*r*r*h;
}

double Cylinder::sarea(){
      return 2*3.142*r*h + 2*3.142*r*r;
}

void Cylinder::display(){
      cout<<"\nThe radius of the cylinder is "<<r;
      cout<<"\nThe height of the cylinder is "<<h;
}

int main(){
      Cylinder c(6,7);
      Cylinder* ptr;
      ptr = &c;
      ptr->display();
      cout<<"\nThe volume of the cylinder is "<<ptr->volume();
      cout<<"\nThe surface area of the cylinder is "<<ptr->sarea();
      return 0;
}
```

Output:

```
Object created
The radius of the cylinder is 6
The height of the cylinder is 7
The volume of the cylinder is 791.784
The surface area of the cylinder is 490.152
```

Practical: 13

Question: Write a program in C++ which uses dynamic memory allocation. Provide copy constructor and destructor. Write a driver program for it.

Code:

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

class DMem{
	private:
		int* ptr;
		int sz;
	public:
		DMem(int n);
		~DMem();
		DMem(DMem& obj);
		void display();
};

DMem::DMem(int n){
	sz = n;
	ptr = new int[sz];
	for(int i = 0; i < sz; ++i){
		ptr[i] = i * 10;
	}
}

DMem::~DMem(){
	cout<<"\nMemory being freed for "<<sz<<" integers and object about to
be destroyed";
	delete [] ptr;
}

void DMem::display(){
	for(int i = 0; i < sz; ++i){
		cout<<"\n"<<ptr[i];
	}
}

void test(DMem obj){
	obj.display();
}

DMem::DMem(DMem& obj){
	cout<<"\nCopy constructor invoked";
	sz = obj.sz;
	ptr = new int[sz];
	for(int i = 0; i < sz; ++i){
		ptr[i] = obj.ptr[i];
	}
}
```

```
int main(){
	DMem d1(4);
	d1.display();
	test(d1);

	DMem d2(5);
	test(d2);

	return 0;
}
```

Output:

```
0
10
20
30
Copy constructor invoked
0
10
20
30
Memory being freed for 4 integers and object about to be destroyed
Copy constructor invoked
0
10
20
30
40
Memory being freed for 5 integers and object about to be destroyed
Memory being freed for 5 integers and object about to be destroyed
Memory being freed for 4 integers and object about to be destroyed
```

Practical: 14

Question: Write a program in C++ to represent a point in a Cartesian co-ordinate system. Provide a constructor which sets the values of the two co-ordinates, and if they are not provided, set it to 0.0. Define accessor functions. Define functions to return the distance of the calling point from another point passed as an argument. Overload it to return the distance of the calling point from the origin. Define method 'display' to display the point in the standard form. Write a driver program.

Code:

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

class Point{
	private:
		double x, y;
	public:
		Point(double xx = 0.0, double yy = 0.0) : x(xx), y(yy) { }
		double getX() {return x;}
		double getY() {return y;}
		double distance();
		double distance(Point p);
		void display();
};

void Point::display(){
	cout<<" ( "<<x<<" , "<<y<<" ) ";
}

double Point::distance(){
	return sqrt(x*x + y*y);
}

double Point::distance(Point p){
	return sqrt((p.x - x)*(p.x - x) + (p.y - y)*(p.y - y));
}

int main(){
	Point origin;
	cout<<"\nCo-ordinates of origin";
	origin.display();
	cout<<endl;

	Point p1(5,5);
	cout<<"\nDistance of ";
	p1.display();
	cout<<" from origin is "<<p1.distance()<<endl;

	Point p2(25,25);
	cout<<"\nDistance of ";
	p2.display();
```

```
        cout<<" from ";
        p1.display();
        cout<<" is "<<p1.distance(p2)<<endl;

        return 0;
}
```

Output:

```
Co-ordinates of origin ( 0 , 0 )
Distance of  ( 5 , 5 )  from origin is 7.07107
Distance of  ( 25 , 25 )  from  ( 5 , 5 )  is 28.2843
```

Practical: 15

Question: Create a class called Quadratic for performing arithmetic on and solving quadratic equations. A quadratic equation is an equation of the form ax^2+bx+c = 0. Use double variables to represent the values of a, b, and c and provide a constructor a is not zero, that enables objects of this class to be initialized when they are created. Give default values of a = 1, b = 0, and c = 0. Create a char variable called variable to represent the variable used in the equation and give it a default value of x. The constructor should not allow the value of a to be 0. If 0 is given, assign 1 to a. Provide public member functions that perform the following tasks.
a) add—adds two Quadratic equations by adding the corresponding values of a, b, and c. The function takes another object of type Quadratic as its parameter and adds it to the calling object.
b) subtract—subtracts two Quadratic equations by subtracting corresponding values of a, b, and c. The function takes another object of type Quadratic as its parameter and subtracts it from the calling object.
c) toString—returns a string representation of a quadratic equation in the standard form using the actual values of the data members.
d) solve—solves a quadratic equation using the quadratic formula. If b^2 - 4ac is greater than/equal to 0, displays the solutions. Otherwise, it displays “No Real Roots.”
Write a driver program to test the functionality of the Quadratic class.

Code:

```
#include<iostream>
#include<cmath>
#include<sstream>
#include<iomanip>
using namespace std;

class Quadratic{
      private:
            double a,b,c;
            char variable;
      public:
            Quadratic(double aa = 1, double bb = 0, double cc = 0, char v =
'x');
            string toString();
            void add(Quadratic q);
            void subtract(Quadratic q);
            void solve();
};

Quadratic::Quadratic(double aa, double bb, double cc, char v){
      a = (aa == 0) ? 1 : aa;
      b = bb;
      c = cc;
      variable = v;
```

```
}

string Quadratic::toString(){
	stringstream ss;
	ss<<a<<variable<<"^2"<<showpos<<b<<variable<<showpos<<c<<" = 0";
	return ss.str();
}

void Quadratic::add(Quadratic q){
	a += q.a;
	b += q.b;
	c += q.c;
}

void Quadratic::subtract(Quadratic q){
	a -= q.a;
	b -= q.b;
	c -= q.c;
}

void Quadratic::solve(){
	double dis = b*b - 4*a*c;
	double r1, r2;
	if(dis < 0){
		cout<<"\nNo real roots!";
	}
	else{
		r1 = (-b+sqrt(dis))/(2*a);
		r2 = (-b-sqrt(dis))/(2*a);
		cout<<"\nThe first root is "<<setprecision(7)<<r1;
		cout<<"\nThe second root is "<<setprecision(7)<<r2<<endl;
	}
}

int main(){
	Quadratic q1, q2(4,5,6), q3(-2,-5,-4), q4(0,5,-7,'y');
	cout<<q1.toString()<<endl;
	cout<<q2.toString()<<endl;
	cout<<q3.toString()<<endl;
	cout<<q4.toString()<<endl;
	q4.solve();
	q2.add(q3);
	cout<<q2.toString()<<endl;
	q4.subtract(q3);
	cout<<q4.toString()<<endl;
	return 0;
}
```

Output:

```
1x^2+0x+0 = 0
4x^2+5x+6 = 0
-2x^2-5x-4 = 0
1y^2+5y-7 = 0

The first root is 1.140055
The second root is -6.140055
2x^2+0x+2 = 0
3y^2+10y-3 = 0
```

Practical: 16

Question: Write a program in C++ to represent a complex number. Define appropriate constructors, destructor. Overload various operators as applicable for complex numbers. Write a driver program to test it.

Code:

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

class Complex{
      private:
            double r;
            double i;
      public:
            Complex(double rr = 0.0, double ii = 0.0);
            Complex(const Complex& obj);
            ~Complex();
            Complex operator+(const Complex& op2);
            Complex operator-(const Complex& op2);
            Complex operator*(const Complex& op2);
            Complex operator/(const Complex& op2);
            Complex operator++();
            Complex operator++(int dummy);
            Complex operator--();
            Complex operator--(int dummy);
            bool operator==(const Complex& op2);
            Complex operator-();
            double operator()();
            friend ostream& operator<<(ostream& out, Complex op2);
};


Complex::Complex(double rr, double ii){
      r = rr;
      i = ii;
}

Complex::Complex(const Complex& obj){
      r = obj.r;
      i = obj.i;
}

Complex::~Complex(){
      //cout<<"\nDestructor invoked";
}

Complex Complex::operator+(const Complex& op2){
      return Complex(r+op2.r, i+op2.i);
}

Complex Complex::operator-(const Complex& op2){
```

```
	return Complex(r-op2.r, i-op2.i);
}

Complex Complex::operator*(const Complex& op2){
	return Complex(r*op2.r - i*op2.i, r*op2.i + i*op2.r);
}

Complex Complex::operator/(const Complex& op2){
	double den = op2.r*op2.r + op2.i*op2.i;
	return Complex((r*op2.r + i*op2.i)/den, (r*op2.i - i*op2.r)/den);
}

Complex Complex::operator++(){
	r++;
	return *this;
}

Complex Complex::operator++(int dummy){
	Complex temp = *this;
	r++;
	return temp;
}

Complex Complex::operator--(){
	r--;
	return *this;
}

Complex Complex::operator--(int dummy){
	Complex temp = *this;
	r--;
	return temp;
}

bool Complex::operator==(const Complex& op2){
	return (r==op2.r && i == op2.i);
}

Complex Complex::operator-(){
	r = -r;
	i = -i;
	return *this;
}

double Complex::operator()(){
	return sqrt(r*r + i*i);
}


ostream& operator<<(ostream& out, Complex op2){
	out<<" "<<op2.r<<showpos<<op2.i<<"i "<<noshowpos;
	return out;
}
```

```
int main(){
	Complex c1(5, 7), c2(2, -3), sum, diff, prod, quot;

	sum = c1 + c2;
	cout<<"\nThe sum of "<<c1<<" and "<<c2<<" is "<<sum;

	diff = c1 - c2;
	cout<<"\nThe difference of "<<c1<<" and "<<c2<<" is "<<diff;

	prod = c1 * c2;
	cout<<"\nThe product of "<<c1<<" and "<<c2<<" is "<<prod;

	quot = c1 / c2;
	cout<<"\nThe division of "<<c1<<" by "<<c2<<" is "<<quot;

	if(c1 == c2){
		cout<<"\n"<<c1<<" is equal to "<<c2;
	}
	else{
		cout<<"\n"<<c1<<" is not equal to "<<c2;
	}

	if(c1 == c1){
		cout<<"\n"<<c1<<" is equal to "<<c1;
	}
	else{
		cout<<"\n"<<c1<<" is not equal to "<<c1;
	}

	cout<<"\n"<<c1<<" after preincrement ";
	cout<<++c1;

	cout<<"\n"<<c1<<" after postdecrement ";
	cout<<c1--;
	cout<<endl;
	cout<<-c1;
	cout<<"\nMagnitude of "<<c1<<" is "<<c1();
	return 0;
}
```

Output:

```
The sum of  5+7i  and  2-3i  is  7+4i
The difference of  5+7i  and  2-3i  is  3+10i
The product of  5+7i  and  2-3i  is  31-1i
The division of  5+7i  by  2-3i  is  -0.846154-2.23077i
 5+7i  is not equal to  2-3i
 5+7i  is equal to  5+7i
 5+7i  after preincrement  6+7i
 6+7i  after postdecrement  6+7i
 -5-7i
Magnitude of  -5-7i  is 8.60233
```

Practical: 17

Question: Write a program in C++ to represent a fraction. Define appropriate constructors. Overload various operators as applicable for fractions. Write a driver program to test it.

Code:

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

class Fraction{
      private:
            int num;
            int den;
      public:
            Fraction();
            Fraction(int n, int d);
            Fraction operator+(Fraction f);
            Fraction operator-(Fraction f);
            Fraction operator*(Fraction f);
            Fraction operator/(Fraction f);
            Fraction operator++();
            Fraction operator++(int dummy);
            Fraction operator--();
            Fraction operator--(int dummy);
            bool operator<(Fraction f);
            bool operator<=(Fraction f);
            bool operator>(Fraction f);
            bool operator>=(Fraction f);
            bool operator==(Fraction f);
            bool operator!=(Fraction f);
            friend Fraction operator+(int x, Fraction f);
            friend ostream& operator<<(ostream& out, Fraction f);
            double operator()();
};

Fraction::Fraction() : num(0), den(1){
}

Fraction::Fraction(int n, int d) : num(n), den(d){
}

Fraction Fraction::operator+(Fraction f){
      return Fraction(num*f.den + den*f.num, den*f.den);
}

Fraction Fraction::operator-(Fraction f){
      return Fraction(num*f.den - den*f.num, den*f.den);
}

Fraction Fraction::operator*(Fraction f){
      return Fraction(num*f.num, den*f.den);
```

```
}

Fraction Fraction::operator/(Fraction f){
      return Fraction(num*f.den, den*f.num);
}

Fraction Fraction::operator++(){
      num = num + den;
      return *this;
}

Fraction Fraction::operator++(int dummy){
      Fraction temp = *this;
      num = num + den;
      return temp;
}

Fraction Fraction::operator--(){
      num = num - den;
      return *this;
}

Fraction Fraction::operator--(int dummy){
      Fraction temp = *this;
      num = num - den;
      return temp;
}

bool Fraction::operator<(Fraction f){
      return num*f.den < den*f.num;
}

bool Fraction::operator<=(Fraction f){
      return num*f.den <= den*f.num;
}

bool Fraction::operator>(Fraction f){
      return num*f.den > den*f.num;
}

bool Fraction::operator>=(Fraction f){
      return num*f.den >= den*f.num;
}

bool Fraction::operator==(Fraction f){
     return (static_cast<double>(num) / den) ==
(static_cast<double>(f.num) / f.den);
}

bool Fraction::operator!=(Fraction f){
      return (static_cast<double>(num) / den) !=
(static_cast<double>(f.num) / f.den);
}

double Fraction::operator()(){
```

```
	return static_cast<double>(num) / den;
}

ostream& operator<<(ostream& out, Fraction f){
	out<<" "<<f.num<<"/"<<f.den<<" ";
	return out;
}

Fraction operator+(int x, Fraction f){
	return Fraction(f.den*x + f.num, f.den);
}

int main(){
	Fraction f1(2,3), f2(5,7);
	Fraction sum, diff, prod, div;
	sum = f1 + f2;
	cout<<"\nAddition of "<<f1<<" and "<<f2<<" is "<<sum;
	diff = f1 - f2;
	cout<<"\nDifference of "<<f1<<" and "<<f2<<" is "<<diff;
	prod = f1 * f2;
	cout<<"\nProduct of "<<f1<<" and "<<f2<<" is "<<prod;
	div = f1 / f2;
	cout<<"\nDivision of "<<f1<<" and "<<f2<<" is "<<div;
	cout<<"\nMagnitude of "<<f1<<" is "<<f1();
	if(f1 == f2){
		cout<<"\n"<<f1<<" is equal to "<<f2;
	}
	else{
		cout<<"\n"<<f1<<" is not equal to "<<f2;
	}

	if(f1 == f1){
		cout<<"\n"<<f1<<" is equal to "<<f1;
	}
	else{
		cout<<"\n"<<f1<<" is not equal to "<<f1;
	}
	cout<<"\nAddition of "<<1<<" and "<<f2<<" is "<<1+f2;
	Fraction temp = f1;
	++f1;
	cout<<"\n"<<temp<<" after preincrement is "<<f1;
	if(f1 < f2){
		cout<<"\n"<<f1<<" is less than "<<f2;
	}
	else{
		cout<<"\n"<<f1<<" is not less than "<<f2;
	}
	return 0;
}
```

Output:

```
Addition of  2/3  and  5/7  is  29/21
Difference of  2/3  and  5/7  is  -1/21
Product of  2/3  and  5/7  is  10/21
Division of  2/3  and  5/7  is  14/15
Magnitude of  2/3  is 0.666667
 2/3  is not equal to  5/7
 2/3  is equal to  2/3
Addition of 1 and  5/7  is  12/7
 2/3  after preincrement is  5/3
 5/3  is not less than  5/7
```

Practical: 18

Question: Write a program in C++ to represent a vector having the standard i, j, k components. Define appropriate constructors and destructor. Overload the operators suitable for the vector objects. Write a driver program for it.

Code:

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

class MathVector{
      private:
            double i,j,k;
      public:
            MathVector(double ii = 0.0, double jj = 0.0, double kk = 0.0);
            MathVector(MathVector& mv);
            ~MathVector();

            MathVector& operator+(MathVector& mv);
            MathVector& operator-(MathVector& mv);
            double operator*(MathVector& mv);
            MathVector& operator/(MathVector& mv);
            MathVector& operator-();
            double operator()();

            MathVector& operator=(MathVector& mv);
            MathVector& operator+=(MathVector& mv);
            MathVector& operator-=(MathVector& mv);
            MathVector& operator/=(MathVector& mv);
            bool operator==(MathVector& mv);
            friend ostream& operator<<(ostream& os, MathVector& mv);
};

MathVector::MathVector(double ii, double jj, double kk){
      i = ii;
      j = jj;
      k = kk;
}

MathVector::MathVector(MathVector& mv){
      //cout<<"Copy Constructor"<<endl;
      i = mv.i;
      j = mv.j;
      k = mv.k;
}

MathVector::~MathVector(){
      //cout<<"Destructor"<<endl;
}

MathVector& MathVector::operator+(MathVector& mv){
```

```
	MathVector temp;
	temp.i = i + mv.i;
	temp.j = j + mv.j;
	temp.k = k + mv.k;
	return temp;
}

MathVector& MathVector::operator-(MathVector& mv){
	MathVector temp;
	temp.i = i - mv.i;
	temp.j = j - mv.j;
	temp.k = k - mv.k;
	return temp;
}

double MathVector::operator*(MathVector& mv){
	return (i*mv.i + j*mv.j + k*mv.k);
}

MathVector& MathVector::operator/(MathVector& mv){
	MathVector temp;
	temp.i = j * mv.k - k * mv.j;
	temp.j = i * mv.k - k * mv.i;
	temp.k = i * mv.j - j * mv.i;
	return temp;
}

MathVector& MathVector::operator-(){
	i = -i;
	j = -j;
	k = -k;
	return *this;
}

double MathVector::operator()(){
	return sqrt(i*i + j*j + k*k);
}

MathVector& MathVector::operator=(MathVector& mv){
	i = mv.i;
	j = mv.j;
	k = mv.k;
	return *this;
}

MathVector& MathVector::operator+=(MathVector& mv){
	i = i + mv.i;
	j = j + mv.j;
	k = k + mv.k;
	return *this;
}

MathVector& MathVector::operator-=(MathVector& mv){
	i = i - mv.i;
	j = j - mv.j;
```

```
	k = k - mv.k;
	return *this;
}

MathVector& MathVector::operator/=(MathVector& mv){

	i = j * mv.k - k * mv.j;
	j = i * mv.k - k * mv.i;
	k = i * mv.j - j * mv.i;
	return *this;
}

bool MathVector::operator==(MathVector& mv){
	return (i==mv.i && j==mv.j && k==mv.k);
}

ostream& operator<<(ostream &os, MathVector &mv){
	stringstream ss;
	ss<<mv.i<<"i "<<showpos<<mv.j<<"j "<<mv.k<<"k "<<noshowpos;
	os<<ss.str();
	return os;
}

int main(){
	MathVector m1, m2(2), m3(4,-5), m4(-5,4,-6), temp;
	cout<<"m1 = "<<m1<<endl;
	cout<<"m2 = "<<m2<<endl;
	cout<<"m3 = "<<m3<<endl;
	cout<<"m4 = "<<m4<<endl;
	m1 = m2+m3;
	cout<<"m2 + m3 = "<<m1<<endl;
	cout<<"Dot product: m3 * m4 = "<<m3*m4<<endl;
	m1 = m3/m4;
	cout<<"Cross product: m3 / m4 = "<<m1<<endl;
	-m4;
	cout<<"-m4 = "<<m4<<endl;

	if(m1 == m2){
		cout<<m1<<" is equal to "<<m2<<endl;
	}
	else{
		cout<<m1<<" is not equal to "<<m2<<endl;
	}
	if(m4 == m4){
		cout<<m4<<" is equal to "<<m4<<endl;
	}
	else{
		cout<<m4<<" is not equal to "<<m4<<endl;
	}
	return 0;
}
```

Output:

```
m1 = 0i +0j +0k
m2 = 2i +0j +0k
m3 = 4i -5j +0k
m4 = -5i +4j -6k
m2 + m3 = 6i -5j +0k
Dot product: m3 * m4 = -40
Cross product: m3 / m4 = 30i -24j -9k
-m4 = 5i -4j +6k
30i -24j -9k  is not equal to 2i +0j +0k
5i -4j +6k  is equal to 5i -4j +6k
```

Practical: 19

Question: Write a program in C++ to represent a date. It should store the day, month, year, and a brief description of the date. The constructor should set these values. It should contain a boolean variable to store the appropriate value if it is a leap year or not, and another variable to store the day of the year. The default constructor should set the date to the current date, and descrition as "current date". Overload the '<', '>', '==', '<<' operators for the date class. Write a driver program for it.

Code:

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

class Date{
	private:
		int day, month, year, day_of_year;
		string description;
		bool leap_year;
		int calculate_day_of_year();
		bool is_leap_year();
	public:
		Date();
		Date(int d, int m, int y, string des);
		bool operator<(Date dt);
		bool operator>(Date dt);
		bool operator==(Date dt);
		friend ostream& operator<<(ostream& out, Date dt);
		int get_day_of_year();
};

Date::Date(){
	time_t now = time(0);
	tm* ct = localtime(&now);
	year = 1900 + ct->tm_year;
	month = 1 + ct->tm_mon;
	day = ct->tm_mday;
	description = "Today's date";
	leap_year = is_leap_year();
	day_of_year = calculate_day_of_year();
}

Date::Date(int d, int m, int y, string des){
	day = d;
	month = m;
	year = y;
	description = des;
	leap_year = is_leap_year();
	day_of_year = calculate_day_of_year();
```

```
}

bool Date::is_leap_year(){
	return ((year%4 == 0 && year%100 !=0) || year%400 == 0);
}

int Date::calculate_day_of_year(){
	int days = 0;
	int no_days[11] = {31,28,31,30,31,30,31,31,30,31,30};
	for(int i = 1; i < month; ++i){
		days += no_days[i-1];
		if(i == 2){
			if(leap_year){
				days += 1;
			}
		}
	}
	days += day;
	return days;
}

ostream& operator<<(ostream& out, Date dt){
	string months[12] = {"January", "February", "March", "April", "May",
"June", "July", "August", "September", "October", "November", "December"};

	stringstream ss;
	ss<<dt.description<<": "<<months[dt.month-1]<<" "<<dt.day<<",
"<<dt.year;
	out<<ss.str();
	return out;
}

bool Date::operator<(Date dt){
	if(year == dt.year){
		if(month == dt.month){
			return day < dt.day;
		}
		else
			return month < dt.month;
	}
	else
		return year < dt.year;
}

bool Date::operator>(Date dt){
	if(year == dt.year){
		if(month == dt.month){
			return day > dt.day;
		}
		else
			return month > dt.month;
	}
	else
		return year > dt.year;
}
```

```
bool Date::operator==(Date dt){
	return day == dt.day && month == dt.month && year == dt.year;
}

int Date::get_day_of_year(){
	return day_of_year;
}

int main() {
	Date c_date;
	Date d1(22, 04, 2022, "Start of exams");
	Date d2(16, 04, 2022, "Practical exams");
	cout<<c_date<<endl;
	cout<<d1<<endl;
	cout<<d2<<endl;
	if(d1 < d2){
		cout<<"Date d1 is less than d2"<<endl;
	}
	else{
		cout<<"Date d1 is greater than or equal to d2"<<endl;
	}

	if(d1 == d1){
		cout<<"Date d1 is equal to d1"<<endl;
	}
	else{
		cout<<"Date d1 is not equal to d1"<<endl;
	}
	return 0;
}
```

Output:

```
Today's date: March 30, 2022
Start of exams: April 22, 2022
Practical exams: April 16, 2022
Date d1 is greater than or equal to d2
Date d1 is equal to d1
```

Practical: 20

Question: Write a C++ program with the following specifications. Create a class 'Vehicle'. It stores the weight and the manufacturing number (both data type is double). Provide Constructor to initialize the values. Define a destructor. Define a method 'display' to display the details of the 'Vehicle' object. Derive three classes 'Airvehicle', 'Landvehicle' and 'Seavehicle' from the above class. It contains a varible to store the speed. Provide constructors, destructors and override the 'display' method. Derive a classes 'Military_air_vehicle', 'Passenger_air_vehicle' from 'Airvehicle'. 'Military_air_vehicle' contains a variable for range of fire, and 'Passenger_air_vehicle' contains a variable for capacity. Provide constructors, destructors and override the 'display' method. Similary derive classes from 'Landvehicle' and 'Seavehicle'. Write a driver program for the classes.

Code:

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

class Vehicle{
	protected:
		double weight,manufacture;
	public:
		Vehicle(double w,double m){
			weight=w;
			manufacture=m;
			cout<<"\nThe constructor of class Vehicle invoked";
		}
		~Vehicle(){
			cout<<"\nThe Destructor of Class Vehicle invoked";
		}
		void display(){
			cout<<"\nThe weight of vehicle is "<<weight;
			cout<<"\nThe Manufacturing no is "<<manufacture;
		}
};

class Airvehicle:public Vehicle{
	protected:
		double speed;
	public:
		Airvehicle(double w,double m,double s):Vehicle(w,m){
			speed=s;
			cout<<"\nThe constructor of class Air vehicle invoked";
		}
		~Airvehicle(){
			cout<<"\nThe destructor of class Air vehicle invoked";
		}
		void display(){
			Vehicle::display();
			cout<<"\nThe speed of air vehicle is "<<speed;
```

```
            }
};

class Landvehicle:public Vehicle{
      protected:
            double speed;
      public:
            Landvehicle(double w,double m,double s):Vehicle(w,m){
                  speed=s;
                  cout<<"\nThe constructor of class land vehicle invoked";
            }
            ~Landvehicle(){
                  cout<<"\nThe destructor of class Land vehicle invoked";
            }
            void display(){
                  Vehicle::display();
                  cout<<"\nThe speed of land vehicle is "<<speed;
            }
};

class Seavehicle:public Vehicle{
      protected:
            double speed;
      public:
            Seavehicle(double w,double m,double s):Vehicle(w,m){
                  speed=s;
                  cout<<"\nThe constructor of class sea  vehicle invoked";
            }
            ~Seavehicle(){
                  cout<<"\nThe destructor of class Sea vehicle invoked";
            }
            void display(){
                  Vehicle::display();
                  cout<<"\nThe speed of Sea vehicle is "<<speed;
            }
};

class Military_air_vehicle:public Airvehicle{
      protected:
            double range_of_fire;
      public:
            Military_air_vehicle(double w,double m,double s,double
r):Airvehicle(w,m,s){
                  range_of_fire=r ;
                  cout<<"\nThe constructor of class military air vehicle
invoked";
            }
            ~Military_air_vehicle(){
                  cout<<"\nThe destructor of class Military air vehicle
invoked";
            }
            void display(){
                  Airvehicle::display();
                  cout<<"\nThe range_of_fire of Military air vehicle is
"<<range_of_fire;
```

```
            }
};

class Passenger_air_vehicle:public Airvehicle{
      protected:
            double capacity;
      public:
            Passenger_air_vehicle(double w,double m,double s,double
c):Airvehicle(w,m,s){
                  capacity=c;
                  cout<<"\nThe constructor of class passenger air vehicle
invoked";
            }
            ~Passenger_air_vehicle(){
                  cout<<"\nThe destructor of class Passenger air vehicle
invoked";
            }
            void display(){
                  Airvehicle::display();
                  cout<<"\nThe capacity of passenger air vehicle is
"<<capacity;
            }
};

class Military_land_vehicle:public Landvehicle{
      protected:
            double range_of_fire;
      public:
            Military_land_vehicle(double w,double m,double s,double
r):Landvehicle(w,m,s){
                  range_of_fire=r ;
                  cout<<"\nThe constructor of class Military land vehicle
invoked";
            }
            ~Military_land_vehicle(){
                  cout<<"\nThe destructor of class Military Land vehicle
invoked";
            }
            void display(){
                  Landvehicle::display();
                  cout<<"\nThe range_of_fire of Military land vehicle is
"<<range_of_fire;
            }
};

class Passenger_land_vehicle:public Landvehicle{
      protected:
            double capacity;
      public:
            Passenger_land_vehicle(double w,double m,double s,double
c):Landvehicle(w,m,s){
                  capacity=c;
                  cout<<"\nThe constructor of class Passenger land vehicle
invoked";
            }
```

```
            ~Passenger_land_vehicle(){
                  cout<<"\nThe destructor of class Passenger Land vehicle
invoked";
            }
            void display(){
                  Landvehicle::display();
                  cout<<"\nThe Capacity of passenger land vehicle is
"<<capacity;
            }
};

class Military_sea_vehicle:public Seavehicle{
      protected:
            double range_of_fire;
      public:
            Military_sea_vehicle(double w,double m,double s,double
r):Seavehicle(w,m,s){
                  range_of_fire=r ;
                  cout<<"\nThe constructor of class military sea vehicle
invoked";
            }
            ~Military_sea_vehicle(){
                  cout<<"\nThe destructor of class Military sea vehicle
invoked";
            }
            void display(){
                  Seavehicle::display();
                  cout<<"\nThe range_of_fire of Military sea vehicle is
"<<range_of_fire;
            }
};

class Passenger_sea_vehicle:public Seavehicle{
      protected:
            double capacity;
      public:
            Passenger_sea_vehicle(double w,double m,double s,double
c):Seavehicle(w,m,s){
                  capacity=c;
                  cout<<"\nThe constructor of class Passenger sea vehicle
invoked";
            }
            ~Passenger_sea_vehicle(){
                  cout<<"\nThe destructor of class Passenger sea vehicle
invoked";
            }
            void display(){
                  Seavehicle::display();
                  cout<<"\nThe Capacity of Passenger sea vehicle is
"<<capacity;
            }
};

int main(){
      Military_air_vehicle ma(1000,2,1000,400);
```

```
        ma.display();
        Passenger_sea_vehicle ps(1000,2,1000,500);
        ps.display();
        return 0;
}
```

Output:

```
The constructor of class Vehicle invoked
The constructor of class Air vehicle invoked
The constructor of class military air vehicle invoked
The weight of vehicle is 1000
The Manufacturing no is 2
The speed of air vehicle is 1000
The range_of_fire of Military air vehicle is 400
The constructor of class Vehicle invoked
The constructor of class sea  vehicle invoked
The constructor of class Passenger sea vehicle invoked
The weight of vehicle is 1000
The Manufacturing no is 2
The speed of Sea vehicle is 1000
The Capacity of Passenger sea vehicle is 500
The destructor of class Passenger sea vehicle invoked
The destructor of class Sea vehicle invoked
The Destructor of Class Vehicle invoked
The destructor of class Military air vehicle invoked
The destructor of class Air vehicle invoked
The Destructor of Class Vehicle invoked
```

Practical: 21

Question: Write a program in C++ to represent an account. It should have one pure virutal function 'withdraw'. 'SavingAccount' and 'CurrentAccount' classes inherits Account class. Write appropriate constructor and define methods to deposit and withdraw. Write a driver program. Use appropriate variables and functions for the class.

Code:

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

class Account{
	protected:
		static int id;
		int account_no;
		string name;
		double balance;
	public:
		Account(string n, double b);
		static int getId();
		void deposit(double amt);
		virtual void withdraw(double amt) = 0;
};

int Account::id = 0;

Account::Account(string n, double b){
	name = n;
	balance = b;
	account_no = ++id;
}

void Account::deposit(double amt){
	balance += amt;
}

int Account::getId(){
	return id;
}

class SavingAccount : public Account{
	protected:
		double min_balance;
	public:
		SavingAccount(string n, double b, double mb) : Account(n,b){
			min_balance = mb;
		}
		void withdraw(double amt){
			if(balance - amt >= min_balance){
				balance -= amt;
```

```
                  }
            }

            friend ostream& operator<<(ostream& out, SavingAccount& a){
                  out<<setw(25)<<left<<"Name of Account
Holder"<<setw(40)<<left<<a.name<<endl;
                  out<<setw(25)<<left<<"Account
Number"<<setw(20)<<left<<a.account_no<<endl;
                  out<<setw(25)<<left<<"Account
Balance"<<setw(20)<<left<<a.balance<<"\n"<<endl;
                  return out;
            }

};

class CurrentAccount : public Account{
      protected:
            double over_draft;
      public:
            CurrentAccount(string n, double b, double od) : Account(n,b){
                  over_draft = od;
            }
            void withdraw(double amt){
                  if(balance + amt >= over_draft){
                        balance -= amt;
                  }
            }

            friend ostream& operator<<(ostream& out, CurrentAccount& a){
                  out<<setw(25)<<left<<"Name of Account
Holder"<<setw(40)<<left<<a.name<<endl;
                  out<<setw(25)<<left<<"Account
Number"<<setw(20)<<left<<a.account_no<<endl;
                  out<<setw(25)<<left<<"Account
Balance"<<setw(20)<<left<<a.balance<<"\n"<<endl;
                  return out;
            }
};

int main(){
      SavingAccount s1("ABC", 20000, 1000);
      SavingAccount s2("EFG", 2000, 1000);
      cout<<s1;
      s1.withdraw(15000);
      cout<<s1;
      s1.deposit(1000);
      cout<<s1;
      cout<<s2;
      s2.withdraw(1500);
      cout<<s2;
      CurrentAccount c1("PQR", 20000, 10000);
      cout<<c1;
      c1.withdraw(25000);
      cout<<c1;
      cout<<"\nLast Account number : "<<Account::getId()<<endl;
```

```
        Account* ptr;
        ptr = &s1;
        ptr->deposit(1000);
        ptr = &c1;
        ptr->deposit(5000);
        cout<<s1;
        cout<<c1;
        ptr = &s1;
        ptr->withdraw(5000);
        ptr = &c1;
        ptr->withdraw(1000);
        cout<<s1;
        cout<<c1;
        return 0;
}
```

Output:

```
Name of Account Holder   ABC
Account Number           1
Account Balance          20000

Name of Account Holder   ABC
Account Number           1
Account Balance          5000

Name of Account Holder   ABC
Account Number           1
Account Balance          6000

Name of Account Holder   EFG
Account Number           2
Account Balance          2000

Name of Account Holder   EFG
Account Number           2
Account Balance          2000

Name of Account Holder   PQR
Account Number           3
Account Balance          20000

Name of Account Holder   PQR
Account Number           3
Account Balance          -5000


Last Account number : 3
Name of Account Holder   ABC
Account Number           1
Account Balance          7000
```

Practical: 22

Question: The abstract class Animal has abstract subclasses Bird and Reptile. Classes Pigeon, Kite, Vulture, Penguin and Albatross inherit Bird. Classes Anaconda and Alligator inherit Reptile. The Coldblooded and Oceanic are abstract classes. Coldblooded is inherited by Reptile and Oceanic is inherited by Penguin, Albatross and Alligator. Define and implement all the classes. Create one instances of each class and display "I am a.. ".

Code:

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

class Animal{
      public:
            virtual void display() = 0;
};

class ColdBlooded{
      public:
            virtual void display() = 0;
};

class Oceanic{
      public:
            virtual void display() = 0;
};

class Reptile : public Animal, public ColdBlooded{
};

class Bird : public Animal{
};

class Pigeon : public Bird{
      public:
            void display(){
                  cout<<"\nI am a Pigeon";
            }
};

class Kite : public Bird{
      public:
            void display(){
                  cout<<"\nI am a Kite";
            }
};

class Vulture : public Bird{
      public:
            void display(){
                  cout<<"\nI am a Vulture";
```

```
            }
};

class Penguin : public Bird, public Oceanic{
      public:
            void display(){
                  cout<<"\nI am a Penguin";
            }
};

class Albatros : public Bird, public Oceanic{
      public:
            void display(){
                  cout<<"\nI am an Albatros";
            }
};

class Alligator : public Reptile, public Oceanic{
      public:
            void display(){
                  cout<<"\nI am an Alligator";
            }
};

class Anaconda : public Reptile{
      public:
            void display(){
                  cout<<"\nI am an Anaconda";
            }
};

int main(){
      Animal* ptr;
      Animal* aptr[7];

      Pigeon pigeon;
      Kite kite;
      Vulture vulture;
      Penguin penguin;
      Albatros albatros;
      Anaconda anaconda;
      Alligator alligator;
      aptr[0] = &pigeon;
      aptr[1] = &kite;
      aptr[2] = &vulture;
      aptr[3] = &penguin;
      aptr[4] = &albatros;
      aptr[5] = &anaconda;
      aptr[6] = &alligator;

      for(int i = 0; i < 7; ++i){
            aptr[i]->display();
      }
      return 0;
}
```

Output:

```
I am a Pigeon
I am a Kite
I am a Vulture
I am a Penguin
I am an Albatros
I am an Anaconda
I am an Alligator
```

Practical: 23

Question: Write a program in C++ which reads a file 'Numbers.txt' containing positive integers. It writes all odd numbers to a file 'Odd.txt' and even numbers to a file 'Even.txt'.

Code:

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

int main(){
        ifstream reader;
        ofstream odd_writer;
            ofstream even_writer;
            int val;

        reader.open("Numbers.txt", ios::in | ios::out);
        if(!reader){
                cout<<"\nFile could not be opened";
                return EXIT_FAILURE;
        }

            odd_writer.open("Odd.txt", ios::out);
        if(!odd_writer){
                cout<<"\nFile Odd could not be opened";
                return EXIT_FAILURE;
        }

            even_writer.open("Even.txt", ios::out);
        if(!even_writer){
                cout<<"\nFile Even could not be opened";
                return EXIT_FAILURE;
        }

        while(reader>>val){
                  if(val%2){
                        odd_writer<<val<<"\n";
                  }
                  else{
                        even_writer<<val<<"\n";
                  }
        }

        reader.close();
            odd_writer.close();
            even_writer.close();

        return EXIT_SUCCESS;
}
```

Output:

```
3    2
5    4
67   78
89   90
```

Practical: 24

Question: Write a program in C++ to write all integers from 33 to 126 and their corresponding character (as per the ASCII values) to a file. Each pair will be in one line.

Code:

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

int main(){
        ifstream rfile;
        ofstream fwriter;
            char ch;
            int val;

        fwriter.open("ASCIITxt.txt", ios::out);
        if(!fwriter){
                cout<<"\nFile could not be opened";
                return EXIT_FAILURE;
        }
            for(int i = 33; i < 127; ++i){
                  fwriter<<setw(5)<<left<<i<<static_cast<char>(i)<<endl;
            }

        fwriter.clear();
        fwriter.close();

            rfile.open("ASCIITxt.txt");
        if(!rfile){
                cout<<"\nFile could not be opened";
                return EXIT_FAILURE;
        }

        while(rfile>>val>>ch){
                cout<<"ASCII value "<<val<<" Character "<<ch<<endl;
        }
        rfile.clear();
        rfile.close();

        return EXIT_SUCCESS;
}
```

Output:

```
ASCII value 33 Character !
ASCII value 34 Character "
ASCII value 35 Character #
ASCII value 36 Character $
ASCII value 37 Character %
ASCII value 38 Character &
```

Practical: 25

Question: Write a template version of a class for implementing the 'stack' data structure. Write a driver program for it using different data types.

Code:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <ctime>
#include <cstdlib>
using namespace std;

class StackUnderflowException{
            string str;
      public:
            StackUnderflowException(string m){
                  str = m;
            }
            friend ostream& operator<<(ostream& o, StackUnderflowException&
s){
                  o<<s.str;
                  return o;
            }
};

class StackOverflowException {
            string str;
      public:
            StackOverflowException(string m){
                  str = m;
            }
            friend ostream& operator<<(ostream& o, StackOverflowException&
s){
                  o<<s.str;
                  return o;
            }
};

template<typename T>
class Stack{
      private:
            vector<T> stk;
            int cap;
      public:
            Stack(int sz);
            Stack(Stack<T>& s);
            ~Stack();
            void push(T el);
            T pop();
            template<typename Tt> friend ostream& operator<<(ostream& out,
Stack<Tt>& obj);
```

```
};


template<typename T>
Stack<T>::Stack(int sz){
	cap = sz;
}


template<typename T>
Stack<T>::~Stack(){
	stk.clear();
}


template<typename T>
Stack<T>::Stack(Stack<T>& s){
	cap = s.cap;
	stk = s.stk;
}


template<typename T>
void Stack<T>::push(T el){
	if(stk.size() == cap){
		throw StackOverflowException("\nStack Overflow!");
	}
	stk.push_back(el);
}


template<typename T>
T Stack<T>::pop(){
	if(stk.empty()){
		throw StackUnderflowException("\nStack Underflow!");
	}
	T temp = stk.back();
	stk.pop_back();
	return temp;
}


template<typename Tt>
ostream& operator<<(ostream& out, Stack<Tt>& obj){
	out<<"\n";
	for(int i = 0; i < obj.stk.size(); ++i){
		out<<obj.stk[i]<<" ";
	}
	return out;
}
```

```
int main(){
	srand(time(NULL));
	Stack<int> istack(rand()%5 + 6);
	Stack<char> cstack(rand()%5 + 6);
	Stack<double> dstack(rand()%5 + 1);
	double temp = 1;
	try{
		istack.push(rand()%100);
		istack.push(rand()%100);
		istack.push(rand()%100);
		istack.push(rand()%100);
		cout<<"\nAfter four push operations";
		cout<<istack;
		istack.pop();
		istack.pop();
		cout<<"\nAfter two pop operations";
		cout<<istack;

		cstack.push(static_cast<char>(rand()%100)+28);
		cstack.push(static_cast<char>(rand()%100)+28);
		cstack.push(static_cast<char>(rand()%100)+28);
		cstack.push(static_cast<char>(rand()%100)+28);
		cstack.push(static_cast<char>(rand()%100)+28);
		cout<<"\nAfter five push operations";
		cout<<cstack;
		cstack.pop();
		cstack.pop();
		cstack.pop();
		cout<<"\nAfter three pop operations";
		cout<<cstack;

		dstack.push(temp / (rand()%10+1));
		dstack.push(temp / (rand()%10+1));
		dstack.push(temp / (rand()%10+1));
		dstack.push(temp / (rand()%10+1));
		cout<<"\nAfter four push operations";
		cout<<dstack;
		dstack.pop();
		dstack.pop();
		dstack.pop();
		cout<<"\nAfter three pop operations";
		cout<<dstack;
		dstack.pop();
		dstack.pop();
		dstack.pop();
	}
	catch(StackUnderflowException& sue){
		cout<<sue;
	}
	catch(StackOverflowException& soe){
		cout<<soe;
	}
	return 0;
}
```

Output:

```
After four push operations
3 76 90 24
After two pop operations
3 76
After five push operations
Y ] W k {
After three pop operations
Y ]
After four push operations
0.2 0.333333 0.25 0.111111
After three pop operations
0.2
Stack Underflow!
```