Week 6

Bugsrobot.h

#pragma once

#include"BugsRobot.h"

class BugsRobot

{

public:

BugsRobot(void);

~BugsRobot(void);

virtual void walk()const;

virtual void fly()const;

virtual void jump()const;

};

Bugsrobot.cpp

#include "stdafx.h"

#include<iostream>

#include<string>

using namespace std;

#include "BugsRobot.h"

BugsRobot::BugsRobot(void)

{

}

BugsRobot::~BugsRobot(void)

{

}

void BugsRobot::walk()const

{

return;

}

void BugsRobot::jump()const

{

return;

}

void BugsRobot::fly()const

{

return;

}

AntRobot.h

#pragma once

#include"AntRobot.h"

#include"BugsRobot.h"

class AntRobot

{

public:

AntRobot(void);

~AntRobot(void);

virtual void walk()const;

};

AntRobot.cpp

#include "stdafx.h"

#include<iostream>

#include<string>

using namespace std;

#include "AntRobot.h"

AntRobot::AntRobot(void)

{

}

AntRobot::~AntRobot(void)

{

}

void AntRobot::walk()const

{

return;

}

BeeRobot.h

#pragma once

#include"BeeRobot.h"

#include"BugsRobot.h"

class BeeRobot

{

public:

BeeRobot(void);

~BeeRobot(void);

virtual void fly()const;

virtual void walk()const;

};

BeeRobot.cpp

#include "stdafx.h"

#include<iostream>

#include<string>

using namespace std;

#include "BeeRobot.h"

BeeRobot::BeeRobot(void)

{

}

BeeRobot::~BeeRobot(void)

{

}

void BeeRobot::fly()const

{

return;

}

void BeeRobot::walk()const

{

return;

}

ButterflyRobot.h

#pragma once

#include"ButterflyRobot.h"

#include"BugsRobot.h"

class ButterflyRobot

{

public:

ButterflyRobot(void);

~ButterflyRobot(void);

virtual void fly()const;

virtual void walk()const;

};

ButterflyRobot.cpp

#include "stdafx.h"

#include<iostream>

#include<string>

using namespace std;

#include "ButterflyRobot.h"

ButterflyRobot::ButterflyRobot(void)

{

}

ButterflyRobot::~ButterflyRobot(void)

{

}

void ButterflyRobot::fly()const

{

return;

}

void ButterflyRobot::walk()const

{

return;

}

GrasshopperRobot.h

#pragma once

#include"GrasshopperRobot.h"

#include"BugsRobot.h"

class GrasshopperRobot

{

public:

GrasshopperRobot(void);

~GrasshopperRobot(void);

virtual void jump()const;

virtual void walk()const;

};

GrasshopperRobot.cpp

#include "stdafx.h"

#include<iostream>

#include<string>

using namespace std;

#include "GrasshopperRobot.h"

GrasshopperRobot::GrasshopperRobot(void)

{

}

GrasshopperRobot::~GrasshopperRobot(void)

{

}

void GrasshopperRobot::jump()const

{

return;

}

void GrasshopperRobot::walk()const

{

return;

}

Rational.h

#pragma once

#include"Rational.h"

class Rational

{

public:

Rational(int=0, int=1);

~Rational(void);

void setnumerator(int);

void setdenominator(int);

Rational add(Rational);

Rational sub(Rational);

Rational mul(Rational);

Rational div(Rational);

void simplify();

void print();

private:

int numerator;

int denominator;

};

Rational.cpp

#include "stdafx.h"

#include "Rational.h"

#include<iostream>

#include<string>

using namespace std;

Rational::Rational(int num, int deno)

{

numerator=num;

denominator=deno;

}

Rational::~Rational(void)

{

}

void Rational::setnumerator(int num)

{

numerator=num;

}

void Rational::setdenominator(int deno)

{

denominator=deno;

}

Rational Rational::add(Rational R)

{

Rational RR(R.numerator\*denominator+numerator\*R.denominator, R.denominator\*denominator);

RR.simplify();

return RR;

}

Rational Rational::sub(Rational R)

{

Rational RR(numerator\*R.denominator-R.numerator\*denominator, R.denominator\*denominator);

RR.simplify();

return RR;

}

Rational Rational::mul(Rational R)

{

Rational RR(R.numerator\*numerator, R.denominator\*denominator);

RR.simplify();

return RR;

}

Rational Rational::div(Rational R)

{

Rational RR(R.denominator\*numerator, R.numerator\*denominator);

RR.simplify();

return RR;

}

void Rational::simplify()

{

for( int a=(denominator>numerator)?denominator:numerator; a>=1; a--)

{

if(numerator%a==0 && denominator%a==0)

{

numerator=numerator/a;

denominator=denominator/a;

break;

}

}

}

void Rational::print()

{

cout<<numerator<<"/"<<denominator;

if (numerator==0)

cout<<"=0";

else if (numerator==1 && denominator==1)

cout<<"=1";

cout<<endl;

}

Template Array class

ArrayT.h

#include "stdafx.h"

#include<iostream>

#include<string>

using namespace std;

template <class T>

class ArrayT

{

public:

ArrayT(T &size);

void add(T a);

void remove(T &a);

void show();

protected:

T \*arr;

T size;

};

template <class T>

ArrayT<T>::ArrayT(T &a)

{

size=a;

arr=new T[size];

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

arr[i]=NULL;

}

template <class T>

void ArrayT<T>::add(T a)

{

if (arr[0]==NULL)

arr[0]=a;

else

{

for (int i=1; i<size; i++)

{

if (arr[i]==NULL)

{

arr[i]=a;

break;

}

}

}

}

template <class T>

void ArrayT<T>::remove(T &a)

{

T ptr;

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

{

if (arr[i]==a)

ptr=i;

}

while(ptr<size)

{

if(arr[ptr]!=NULL)

{

arr[ptr]=arr[ptr+1];

ptr++;

}

else

arr[ptr]=NULL;

}

}

template <class T>

void ArrayT<T>::show()

{

for(int a=0; a<size; a++)

cout<<arr[a]<<" ";

cout<<endl;

}

Main.cpp

// W6 Practical.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include<iostream>

#include<string>

using namespace std;

#include"Rational.h"

#include"ArrayT.h"

#include"AntRobot.h"

#include"BeeRobot.h"

#include"ButterflyRobot.h"

#include"GrasshopperRobot.h"

template<class T>

void Swap(T &a, T &b){

T c;

c=a;

a=b;

b=c;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

Rational R1(1,3);

Rational R2(2,6);

Rational R3;

cout<<"Simplify R2(2/6): ";

R2.simplify();

R2.print();

cout<<"For add: ";

R3=R1.add(R2);

R3.print();

cout<<"For Subtract: ";

R3=R1.sub(R2);

R3.print();

cout<<"For multiple: ";

R3=R1.mul(R2);

R3.print();

cout<<"For divdide: ";

R3=R1.div(R2);

R3.print();

cout<<endl;

//template function

int a=2, c=4;

Swap(a,c);

double p=1.2, d=3.1;

Swap(p,d);

char b='b', z='z';

Swap(b,z);

//template class

int u=4,i=5,y=6,r=7;

ArrayT <int> a1(u);

a1.add(u);

a1.add(i);

a1.add(y);

a1.add(r);

a1.show();

a1.remove(i);

a1.show();

return 0;

}