МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М.Є. Жуковського “ХАІ”

Кафедра 603

**Лабораторна робота № 5**

**ПАРАМЕТРИЗОВАННЫЙ КЛАСС. СПЕЦИАЛЬНЫЙ ПОЛИМОРФИЗМ. ОБЪЕКТНООРИЕНТИРОВАННОЕ ПРЕДСТАВЛЕНИЕ ЧИСЛОВЫХ МАССИВОВ С ЭЛЕМЕНТАМИ РАЗНЫХ ТИПОВ**

з дисципліни “Об’єктно орієнтоване програмування”

ХАІ.603.612ПСТ.16О.050103.166380.ПЗ

Виконав студент: гр.612пст Лезновский В.А.

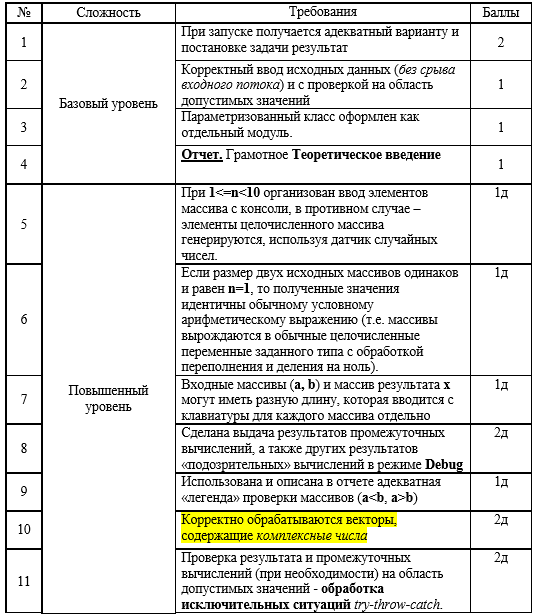
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Перевірив:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

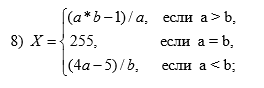
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2016

**Порядок выполнения работы (базовый и повышенный уровень)**

Взяв за основу анализ и решение задачи в лабораторной работе № 4, организовать параметризованный класс, считая, что a, b и x - одномерные векторы, содержащие элементы одного из всевозможных арифметических базовых типов данных (все массивы или целочисленные, или вещественные), длиной 1<=n<=N=1000. Предусмотреть корректный потоковый ввод-вывод информации с использованием динамической идентификации типов элементов массива (RTTI – Run Time Type Identification).

**Постановка задачи**



**Теоретическое введение**

**Параметризированный класс** (Шаблоны в С++) – Средство предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию). В C++ возможно создание шаблонов функций и классов. Шаблоны позволяют создавать параметризованные классы и функции.

Параметром может быть любой тип или значение одного из допустимых типов (целое число, enum, указатель на любой объект с глобально доступным именем, ссылка). Хотя шаблоны предоставляют краткую форму записи участка кода, на самом деле их использование не сокращает исполняемый код, так как для каждого набора параметров компилятор создаёт отдельный экземпляр функции или класса. Как следствие, исчезает возможность совместного использования скомпилированного кода в рамках разделяемых библиотек.

**Динамическая идентификация типа данных** (англ. run-time type information, run-time type identification, RTTI) — механизм в некоторых языках программирования, который позволяет определить тип данных переменной или объекта во время выполнения программы.

**Конструктор** – это метод класса, который отвечает за инициализацию переменных, то есть задает элементам класса начальные значения, обычно у класса уже есть конструктор по умолчанию, но программист может описать свой конструктор с необходимой ему инициализацией.

**Деструктор** – как и конструктор это метод класса, однако он выполняет роль отчистки в случае если была выделена динамическая память.

**Перегрузка операторов** — один из способов реализации полиморфизма, заключающийся в возможности одновременного существования в одной области видимости нескольких различных вариантов применения оператора, имеющих одно и то же имя, но различающихся типами параметров, к которым они применяются.

**inline-функция** — это такая функция, чье тело подставляется в каждую точ­ку вызова, вместо того, чтобы генерировать код вызова. Это подобно использованию параметризованных макросов в С. Имеются два способа создания inline-функции. Первый заключа­ется в использовании модификатора inline. Причина использования inline-функции заключается в их эффективности. Всякий раз, когда вы­зывается функция, необходимо выполнить серию инструкций для формирования вызова функции, вставки аргументов в стек и возврата значения из функции. В некоторых случаях для этого прихо­дится использовать много тактов центрального процессора. При использовании inline-функции нет необходимости в таких дополнительных действиях и скорость выполнения программы возрастает.

**Массив в объектно-ориентированном представление** – это структура, которая содержит методы для его обработки, например задание размера или переопределение операций для конкретного типа данных

**Правила переопределения операций для унарных и бинарных операций**.

I. Язык C++ не допускает определения для операций нового лексического символа, кроме уже определенных в языке. Например, нельзя определить в качестве знака операции @.

II. Не допускается перегрузка операций для встроенных типов данных. Нельзя, например, переопределить операцию сложения целых чисел:

III. int operator +(int i, int j);

IV. Нельзя переопределить приоритет операции.

V. Нельзя изменить синтаксис операции в выражении. Например, если некоторая операция определена как унарная, то ее нельзя определить как бинарную. Если для операции используется префиксная форма записи, то ее нельзя переопределить в постфиксную. Например, !а нельзя переопределить как а!

VI. Перегружать можно только операции, для которых хотя бы один аргумент представляет тип данных, определенный пользователем. Функция-операция должна быть определена либо как функция-член класса, либо как внешняя функция, но дружественная классу.

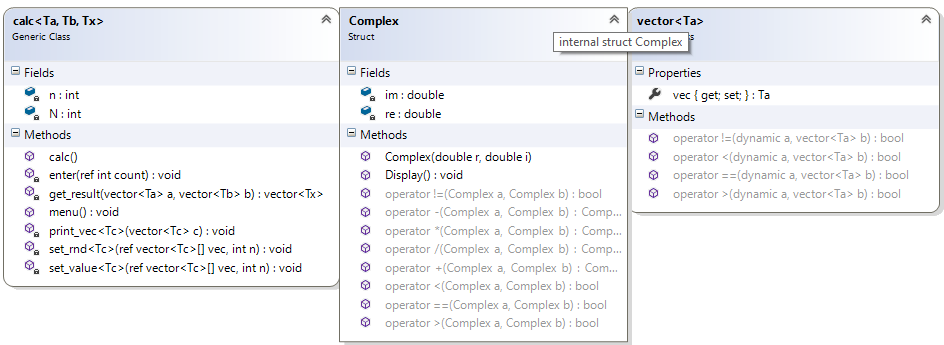
**Описание программы**

Общие сведения:

* Язык программирования – С#;
* Операционная система – Windows 10;
* Тип компилятора – Visual studio.

Графическое описание алгоритма решения задачи :

**UML-диаграмма**



**Легенда проверки массивов**

Длинна результирующего вектора Х равна длине вектора с наименьшим количеством элементов (X(min)). Сравнение массивов происходит путем сравнения каждого элемента (a[i]>b[i]). При различной длине массивов элементы берутся до длинны меньшего вектора(<=min). Комплексные числа сравниваются по действительной и мнимой части одновременно ( A[D] + B[D] ; A[M] + B[M]),

Логические операции также сравнивают действительную и мнимую часть (A[D]>B[D] and A[M]>A[M]).

**Листинг программы**

**Файл calc.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Diagnostics;

using System.Dynamic;

using ConsoleApplication1;

namespace ConsoleApplication1

{

//Пользовательское исключение

class OutOfRengeExc : Exception { }

class calc<Ta, Tb, Tx>

{

int N, n;

private void enter(ref int count)

{

while (true)

{

try

{

count = int.Parse(Console.ReadLine());

if (count > 1 | count < 1000)

break;

}

catch (Exception exc) { Console.WriteLine(exc.Message); }

}

}

public calc()

{

Console.WriteLine("Размер вектора А:");

enter(ref n);

Console.WriteLine("Размер вектора B:");

enter(ref N);

}

private void print\_vec<Tc>(vector<Tc> c)

{

if (typeof(Tc).ToString() != "ConsoleApplication1.Complex")

Console.WriteLine(c.vec);

else

{

Complex com = (dynamic)c.vec;

com.Display();

}

}

private void set\_value<Tc>(ref vector<Tc>[] vec, int n)

{

while (true)

{

try

{

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

{

vec[i] = new vector<Tc>();

Console.Write("[" + i + "]=");

if (typeof(Tc).ToString() == "ConsoleApplication1.Complex")

{

Console.Write("\r\n");

Complex com = new Complex(double.MinValue, double.MinValue);

vec[i].vec = (dynamic)com;

}

else if (typeof(Tc).ToString() == "System.Int32")

vec[i].vec = (dynamic)int.Parse(Console.ReadLine());

else if (typeof(Tc).ToString() == "System.Double")

vec[i].vec = (dynamic)double.Parse(Console.ReadLine());

else if (typeof(Tc).ToString() == "System.Decimal")

vec[i].vec = (dynamic)decimal.Parse(Console.ReadLine());

else vec[i].vec = (dynamic)float.Parse(Console.ReadLine());

}

break;

}

catch (Exception exc) { Console.WriteLine(exc.Message); }

}

}

private void set\_rnd<Tc>(ref vector<Tc>[] vec, int n)

{

Random rnd = new Random();

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

{

vec[i] = new vector<Tc>();

if (typeof(Tc).ToString() == "ConsoleApplication1.Complex")

{

Complex com = new Complex(rnd.NextDouble() - 0.5, rnd.NextDouble() - 0.5);

vec[i].vec = (dynamic)com;

}else if (typeof(Tc).ToString() == "System.Int32")

vec[i].vec = (dynamic)rnd.Next(0, 1000) - 500;

else if (typeof(Tc).ToString() == "System.Double")

vec[i].vec = (dynamic)rnd.NextDouble() - 0.5;

else if (typeof(Tc).ToString() == "System.Decimal")

vec[i].vec = (dynamic)Convert.ToDecimal(rnd.NextDouble() - 0.5);

else vec[i].vec = (dynamic)Convert.ToSingle(rnd.NextDouble() - 0.5);

}

}

private vector<Tx> get\_result(vector<Ta> a, vector<Tb> b)

{

vector<Tx> Left = new vector<Tx>(), Right = new vector<Tx>(), Result = new vector<Tx>();

try

{

if (typeof(Ta).ToString() == "ConsoleApplication1.Complex")

{

if ((dynamic)a > b)

{

Left.vec = (a.vec \* (dynamic)b.vec - (new Complex(1,1)));

Right.vec = (dynamic)a.vec;

if ((dynamic)Right.vec == new Complex(0, 0))

throw new DivideByZeroException();

Result.vec = Left.vec / (dynamic)Right.vec;

Console.Write("a>b");

Debug.WriteLine("Override operator >. X = A > B: " + Left.vec + " / " + Right.vec + " = " + Result);

}

else if ((dynamic)a < b)

{

Right.vec = (dynamic)b.vec;

Complex number = new Complex(5, 5);

Complex number1 = new Complex(4, 4);

Left.vec = ((dynamic)number1 \* a.vec - (dynamic)number);

if ((dynamic)Right.vec == new Complex(0, 0))

throw new DivideByZeroException();

Result.vec = (dynamic)Left.vec / Right.vec;

Console.Write("a<b");

Debug.WriteLine("Override operator <. X = A < B: " + Left.vec + " / " + Right.vec + " = " + Result);

}

else if ((dynamic)a == b)

{

Result.vec = (dynamic)(new Complex(255, 255));

Console.Write("a==b");

Debug.WriteLine("Override operator ==. X = A == B: " + Result);

}

else Console.Write(" К сожалению условия не подошли");

}

else

{

if ((dynamic)a > b)

{

Left.vec = (a.vec \* (dynamic)b.vec - (dynamic)1);

Right.vec = (dynamic)a.vec;

if (Left.vec / (dynamic)Right.vec == 0)

throw new DivideByZeroException();

Result.vec = Left.vec / (dynamic)Right.vec + (dynamic)1;

Console.Write("a>b");

Debug.WriteLine("Override operator >. X = A > B: " + Left.vec + " / " + Right.vec + " = " + Result);

}

else if ((dynamic)a < b)

{

Right.vec = (dynamic)b.vec;

Left.vec = ((dynamic)4 \* a.vec - (dynamic)5);

if (Left.vec / (dynamic)Right.vec == 0)

throw new DivideByZeroException();

Result.vec = Left.vec / (dynamic)Right.vec;

Console.Write("a<b");

Debug.WriteLine("Override operator <. X = A < B: " + Left.vec + " / " + Right.vec + " = " + Result);

}

else if ((dynamic)a == b)

{

Result.vec = (dynamic)255;

Console.Write("a==b");

Debug.WriteLine("Override operator ==. X = A == B: " + Result);

}

else Console.Write(" К сожалению ниодного условия не подошло");

}

}

catch (Exception exc) { Console.WriteLine(exc.Message); }

return Result;

}

public void menu()

{

ConsoleKeyInfo e = new ConsoleKeyInfo();

vector<Ta>[] a\_vec = new vector<Ta>[n];

vector<Ta> a = new vector<Ta>();

vector<Tb>[] b\_vec = new vector<Tb>[N];

vector<Tb> b = new vector<Tb>();

Debug.Indent();

bool enter\_value = false;

//Циклическое "Меню"

while (e.Key != ConsoleKey.Escape)

{

Console.Clear();

Console.WriteLine("1 Перезаписать размерность векторов А и В");

Console.WriteLine("2 Перезаписать векторы А и В");

Console.WriteLine("3 Вывести на экран векторы А и В");

Console.WriteLine("4 Вычислить результат");

Console.WriteLine("Esc Выход");

e = Console.ReadKey(true);

Console.Clear();

//Ввод размерности

if (e.Key == ConsoleKey.D1)

{

Console.WriteLine("Размер вектора А:");

enter(ref n);

Console.WriteLine("Размер вектора B:");

enter(ref N);

}

//Ввод данных

else if (e.Key == ConsoleKey.D2)

{

if (n > 10)

set\_rnd<Ta>(ref a\_vec, n);

else

{

Console.WriteLine("Введите вектор А типа "+typeof(Ta).ToString());

Console.Write("a");

set\_value<Ta>(ref a\_vec, n);

}

if (N > 10)

set\_rnd<Tb>(ref b\_vec, N);

else

{

Console.WriteLine("Введите вектор B типа " + typeof(Tb).ToString() + ":");

Console.Write("b");

set\_value<Tb>(ref b\_vec, N);

}

if (n == N && n == 1)

{

a = a\_vec[0];

b = b\_vec[0];

}

enter\_value = true;

Console.WriteLine("Векторы были успешно записаны");

}

//Вывод векторов

else if (e.Key == ConsoleKey.D3 & enter\_value)

{

if (n == N && n == 1)

{

Console.WriteLine("\r\n\r\nВектор A типа:" + typeof(Ta).ToString() + "\r\n");

print\_vec<Ta>(a);

Console.WriteLine("\r\n\r\nВектор B типа:" + typeof(Tb).ToString() + "\r\n");

print\_vec<Tb>(b);

}

else

{

Console.WriteLine("Вектор А типа:" + typeof(Ta).ToString() + "\r\n");

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

print\_vec<Ta>(a\_vec[i]);

Console.WriteLine("\r\n\r\nВектор B типа:" + typeof(Tb).ToString() + "\r\n");

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

print\_vec<Tb>(b\_vec[i]);

}

}

//Вычисление вектора

else if (e.Key == ConsoleKey.D4 & enter\_value)

{

if (n == N && n == 1)

{

Console.WriteLine("\r\n\r\nВектор A типа:" + typeof(Ta).ToString() + "\r\n");

print\_vec<Ta>(a);

Console.WriteLine("\r\n\r\nВектор B типа:" + typeof(Tb).ToString() + "\r\n");

print\_vec<Tb>(b);

}

else

{

Console.WriteLine("Вектор А типа:" + typeof(Ta).ToString() + "\r\n");

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

print\_vec<Ta>(a\_vec[i]);

Console.WriteLine("\r\n\r\nВектор B типа:" + typeof(Tb).ToString() + "\r\n");

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

print\_vec<Tb>(b\_vec[i]);

}

Console.WriteLine("\r\n");

if (n == N && n == 1)

{

vector<Tx> x = new vector<Tx>();

Console.WriteLine("\r\n\r\nВектор X типа:" + typeof(Tx).ToString() + "\r\n");

x = new vector<Tx>();

Console.Write("[" + 0 + "]: ");

x = get\_result(a, b);

Console.Write(" x = ");

print\_vec<Tx>(x);

}

else

{

vector<Tx>[] x = new vector<Tx>[n];

Console.WriteLine("\r\n\r\nВектор X типа:" + typeof(Tx).ToString() + "\r\n");

if (n > N)

n = N;

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

{

x[i] = new vector<Tx>();

Console.Write("[" + i + "]: ");

x[i] = get\_result(a\_vec[i], b\_vec[i]);

Console.Write(" x = ");

print\_vec<Tx>(x[i]);

}

}

}

//Выход из цикла как только нажата клавиша ескейп.

else if (e.Key == ConsoleKey.Escape) break;

if (!enter\_value)

Console.WriteLine("Введите сначала векторы");

Console.WriteLine("\r\n\r\nДля продолжения нажмите любую кнопку...");

Console.ReadKey(true);

}

}

}

}

**Program.cs**

namespace ConsoleApplication1

{

class Program

{

static void Main()

{

System.Threading.Thread.CurrentThread.Name = "Pyrhavka 612 pst.";

calc<Complex, Complex, Complex> x = new calc<Complex, Complex, Complex>();

x.menu();

Console.ReadKey(true);

} }}

**Struct.cs**

namespace ConsoleApplication1

{

//комплексные числа

struct Complex

{

double re;

double im;

public Complex(double r, double i)

{

while(true)

try

{

if (r == double.MinValue & i == double.MinValue)

{

Console.Write("r="); re = double.Parse(Console.ReadLine());

Console.Write("i="); im = double.Parse(Console.ReadLine());

}

else

{

re = r; im = i;

}

break;

}

catch (Exception exc) { Console.WriteLine(exc.Message); }

}

//Вывод комплексного числа

public void Display()

{

Console.Write(re + ", " + im+"\r\n");

}

//Перегрузка операторов > < == для комплексных чисел

public static bool operator >(Complex a, Complex b)

{

if (a.re > b.re & a.im > b.im)

return true;

else return false;

}

public static bool operator <(Complex a, Complex b)

{

if (a.re < b.re & a.im < b.im)

return true;

else return false;

}

public static bool operator ==(Complex a, Complex b)

{

if (a.re == b.re & a.im == b.im)

return true;

else return false;

}

public static bool operator !=(Complex a, Complex b)

{

if (a.re != b.re & a.im != b.im)

return true;

else return false;

}

//Перегрузка операторов + - для комплексных чисел

public static Complex operator +(Complex a, Complex b)

{

Complex res = new Complex(0.0, 0.0);

res.re = a.re + b.re; res.im = a.im + b.im;

return res;

}

public static Complex operator -(Complex a, Complex b)

{

Complex res = new Complex(0.0, 0.0);

res.re = a.re - b.re; res.im = a.im - b.im;

return res;

}

//Перегрузка операторов / \*

public static Complex operator /(Complex a, Complex b)

{

Complex res = new Complex(0.0, 0.0);

res.re = a.re / b.re; res.im = a.im / b.im;

return res;

}

public static Complex operator \*(Complex a, Complex b)

{

Complex res = new Complex(0.0, 0.0);

res.re = a.re \* b.re; res.im = a.im \* b.im;

return res;

}

}

//Класс описываюший вектор

class vector<Ta>

{

public Ta vec { get; set; }

public static bool operator >(dynamic a, vector<Ta> b)

{

bool Result = false;

if (a.vec > b.vec)

Result = true;

return Result;

}

public static bool operator <(dynamic a, vector<Ta> b)

{

bool Result = false;

if (a.vec < b.vec)

Result = true;

return Result;

}

public static bool operator ==(dynamic a, vector<Ta> b)

{

bool Result = false;

if (a.vec == b.vec)

Result = true;

return Result;

}

public static bool operator !=(dynamic a, vector<Ta> b)

{

bool Result = false;

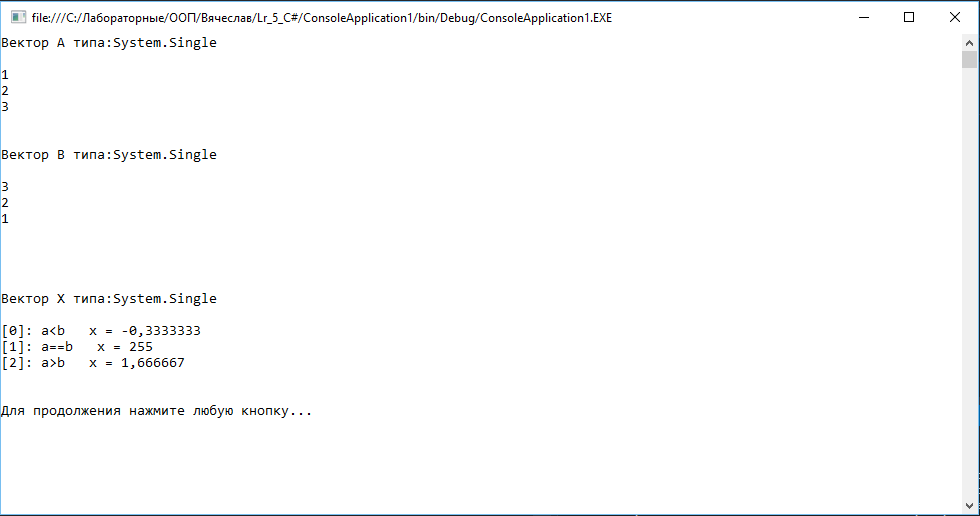
if (a.vec != b.vec)

Result = true;

return Result;

} }}

**Проверка расчетов**



Пример выполнения программы



Окно режима Debug

**Вывод:** На лабораторной работе я научился организовать параметризованный класс. Изучил специальный полиморфизм и объектно ориентированное представление числовых массивов с элементами разных типов.