Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Отчет

по лабораторной работе № 6

«Использование потоков ввода-вывода.

Обработка исключений.»

Выполнил

ст. гр. ИСб22

Серегин А.В.

Проверил:

Тимофеев И.С.

Севастополь

2015

1. Цель работы

Изучить механизм потокового ввода/вывода, обеспечивающий гибкий и эффективный с гарантией типа метод обработки символьного ввода целых чисел, чисел с плавающей точкой и символьных строк, а также простую модель его расширения для обработки типов, определяемых пользователем. Приобрести практические навыки написания объектно-ориентированных программ с использованием обработки исключительных ситуаций.

2. Варианты заданий

Вариант 5:

1. Написать программу решения линейного уравнения. Результаты вычислений поместить в файл. Установить ширину поля 12 символов, точность –– 4 цифры, пробелы заменить на символ “%”. Предусмотреть обработку ошибок.

2. Создать класс Вектор (float \*). Описать перехват ошибок, связанных с неверным вводом значений. Пример: ввели букву вместо цифры.

3. Текст программы

Vector.h:

#pragma once

#ifndef \_VECTOR\_H\_

#define \_VECTOR\_H\_

#include <cstring>

class Vector

{

private:

double\* arr\_;

size\_t size\_;

size\_t capacity\_;

void checkCapacity();

public:

class bad\_Vector

{

private:

char\* error\_;

public:

bad\_Vector() :error\_("unknown") {};

bad\_Vector(char\* error)

{

error\_ = error;

}

bad\_Vector(const bad\_Vector& e)

{

error\_ = e.what();

}

char\* what() const

{

return error\_;

}

};

Vector();

Vector(const size\_t& \_size);

Vector(const Vector& \_copy);

void clear();

size\_t capacity() const;

size\_t size() const;

void resize(const size\_t& \_size);

void reserve(const size\_t& \_capacity);

void push\_back(const double& \_val);

void pop\_back();

void operator=(const Vector& \_right);

double& operator[](const size\_t& \_n);

double& operator[](const size\_t& \_n) const;

~Vector();

};

#endif

Vector.cpp:

#include "Vector.h"

Vector::Vector()

{

arr\_ = nullptr;

size\_ = 0;

capacity\_ = 0;

}

Vector::Vector(const size\_t& \_size):Vector()

{

this->resize(\_size);

}

Vector::Vector(const Vector& \_copy)

{

size\_ = \_copy.size\_;

capacity\_ = \_copy.capacity\_;

arr\_ = new double[capacity\_];

for (size\_t i = 0; i < size\_; ++i)

arr\_[i] = \_copy.arr\_[i];

for (size\_t i = size\_; i <= capacity\_; ++i)

arr\_[i] = 0.0;

checkCapacity();

}

void Vector::clear()

{

delete[] arr\_;

capacity\_ = 0;

size\_ = 0;

}

void Vector::checkCapacity()

{

if (capacity\_ < size\_)

capacity\_ = size\_;

}

size\_t Vector::capacity() const

{

return capacity\_;

}

size\_t Vector::size() const

{

return size\_;

}

void Vector::resize(const size\_t& \_size)

{

if (capacity\_ >= \_size)

{

for (size\_t i = \_size; i < size\_; ++i)

arr\_[i] = 0.0;

size\_ = \_size;

return;

}

double\* newArr = new double[\_size];

for (size\_t i = size\_; i < \_size; ++i)

newArr[i] = 0.0;

capacity\_ = \_size;

for (size\_t i = 0; i < size\_; ++i)

newArr[i] = arr\_[i];

delete[] arr\_;

arr\_ = newArr;

size\_ = \_size;

checkCapacity();

}

void Vector::reserve(const size\_t& \_capacity)

{

checkCapacity();

if (\_capacity <= capacity\_)

return;

size\_t temp = size\_;

resize(\_capacity);

capacity\_ = \_capacity;

size\_ = temp;

}

void Vector::push\_back(const double& \_val)

{

this->resize(size\_ + 1);

arr\_[size\_ - 1] = \_val;

}

void Vector::pop\_back()

{

if (size\_ == 0)

throw bad\_Vector("pop\_back(): vector is empty");

this->resize(size\_ - 1);

}

void Vector::operator=(const Vector& \_right)

{

size\_ = \_right.size\_;

capacity\_ = \_right.capacity\_;

arr\_ = new double[capacity\_];

for (size\_t i = 0; i < size\_; ++i)

arr\_[i] = \_right.arr\_[i];

for (size\_t i = size\_; i <= capacity\_; ++i)

arr\_[i] = 0.0;

}

double& Vector::operator[](const size\_t& \_n)

{

if (\_n >= size\_)

throw bad\_Vector("out of range");

return arr\_[\_n];

}

double& Vector::operator[](const size\_t& \_n) const

{

if (\_n >= size\_)

throw bad\_Vector("out of range");

return arr\_[\_n];

}

Vector::~Vector()

{

this->clear();

}

main.cpp:

#include <iostream>

#include <fstream>

#include <iomanip>

#include <exception>

#include "Vector.h"

void printArr(const Vector& \_arr)

{

for (size\_t i = 0; i < \_arr.size(); ++i)

std::cout << \_arr[i] << ' ';

std::cout << std::endl;

}

int main()

{

try

{

double a, x, b;

std::ofstream fout("output.txt");

fout.setf(std::ios::fixed);

fout.width(12);

fout.precision(4);

fout.fill('%');

std::cout << "Input a, b, x:" << std::endl;

std::cin >> a >> b >> x;

fout << a\*x + b << std::endl;

fout.close();

Vector arr(10);

for (size\_t i = 0; i < arr.size(); ++i)

arr[i] = rand() % 10;

std::cout << "Random array:" << std::endl;

printArr(arr);

arr.resize(1);

std::cout << "Array size set 1:" << std::endl << "Array: ";

printArr(arr);

arr.pop\_back();

std::cout << "Pop last element from array" << std::endl << "Array: ";

printArr(arr);

arr.pop\_back();

std::cout << "Pop last element from array" << std::endl << "Array: ";

printArr(arr);

}

catch (const Vector::bad\_Vector& e)

{

std::cerr << "Error: " << e.what() << std::endl;

}

catch (const std::exception& e)

{

std::cerr << "Error: " << e.what() << std::endl;

}

catch (...)

{

std::cerr << "Error: " << "unknown error" << std::endl;

}

system("pause");

return 0;

}

4. Тестовые примеры.

На рисунках ниже приведен результат выполнения программы.

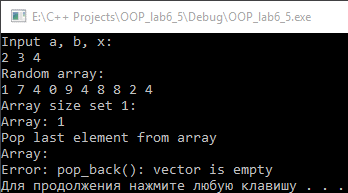


Рисунок 1 – результат вывода на консоль.

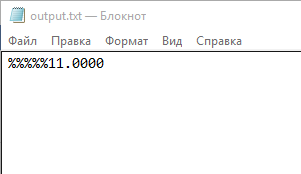


Рисунок 2 – результат вывода в файл.

Выводы

В ходе лабораторной работы был изучен механизм потокового ввода-вывода, с помощью которого можно удобно форматировать вывод данных как в консоль, так и в файлы. Также был изучен механизм обработки исключений.