Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет

Кафедра **«Информационные технологии и автоматизированные системы»**

направление подготовки: 09.03.04 - «Программная инженерия»

Лабораторная работа №9

По теме **«Обработка исключительных ситуаций»**

Вариант №10

Выполнял:

студент группы РИС-24-1б

Морозова Н.С.

Проверял:

Доцент кафедры ИТАС

Полякова О.А.

Постановка задачи:

1. Реализовать класс, перегрузить для него операции, указанные в варианте.
2. Определить исключительные ситуации.
3. Предусмотреть генерацию исключительных ситуаций.

Задание:

Класс-контейнер МНОЖЕСТВО с элементами типа int.

Реализовать операции:

[] доступа по индексу;

() определение размера вектора;

- разность множеств;

-- удаление элемента из множества.

Анализ задачи:

1. Исключительные ситуации генерируются:
2. В операции [] при попытке обратиться к элементу с номером меньше 0 или больше текущего размера множества;
3. В операции - при попытке реализовать вычесть из меньшего множества большее;
4. В операции -- при попытке удалить элемент из пустого множества или удалить несуществующий элемент (неверный индекс).
5. В лабораторной работе 6 реализованы почти все методы, что даны здесь.
6. Осталось только добавить исключения и описать функцию удаления элемента из множества.
7. Для второго метода нужно добавить файл error.h с классом error, с помощью которого будут выводиться сообщения об ошибках.

Решение

Код

Файл Lab\_9.cpp

#include "setValuse.h"

#include <Windows.h>

#include <iostream>

using namespace std;

int main(){

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

//контролируемый блок

try {

SetValue x(5);

cout << "Длина множества x = " << x() << endl;

cout << "Введите значения множества x:" << endl;

cin >> x;

cout << "Множество x: " << x << endl;

cout << "Индекс: ";

int i;

cin >> i;

cout << "x[i] = " << x[i] << endl;

//удалить один элемент

--x;

cout << "Множество без последнего элемента: " << x << endl;

SetValue z(5);

cout << "Введите значения множества z:" << endl;

cin >> z;

cout << "Множество z: " << z << endl;

cout << "Разность множеств z - x: " << z - x << endl;

SetValue y(3);

cout << "Длина множества y = " << y() << endl;

cout << "Введите значения множества y:" << endl;

cin >> y;

cout << "Множество y: " << y << endl;

cout << "Разность множеств y - x: " << y - x << endl;

}

//обработчик исключения

catch(int e)

{ cout << "ERROR!!! Код ошибки: " << e << endl; }//сообщение об ошибке

catch (...) // Ловим все остальные исключения

{ cout << "Неизвестная ошибка!" << endl; }

return 0;

}

Файл setValuse.h

#pragma once

#include <iostream>

using namespace std;

class Iterator{

friend class SetValue;//дружественный класс

public:

int\* elem; //указатель на элемент типа int

Iterator() { elem = 0; } //конструктор без параметров

Iterator(const Iterator& it) { elem = it.elem; } //конструктор копирования

//перегруженные операции сравнения

bool operator==(const Iterator& it) { return elem == it.elem; }

bool operator!=(const Iterator& it) { return elem != it.elem; }

void operator++() { ++elem; }

void operator--() { --elem; }

//перегруженная операция разыменования

int& operator\*() { return \*elem; }

};

class SetValue{

int size; //размер множества

int\* data; //указатель на динамический массив значений

Iterator beg;//указатель на первый элемент вектора

Iterator end;//указатель на элемент следующий за последним

public:

SetValue() : size(0), data(nullptr) {}

SetValue(int s, int k = 0);

//конструктор копирования

SetValue(const SetValue& a);

//деструктор

~SetValue();

//оператор присваивания

SetValue& operator=(const SetValue& a);

//операция доступа по индексу

int& operator[](int index);

//операция, возвращающая длину вектора

int operator()();

//операция разности множеств

friend SetValue operator-(const SetValue& a, const SetValue& b);

//операция удаления последнего элемента

void operator--();

//перегруженные операции ввода-вывода

friend ostream& operator<<(ostream& out, const SetValue& a);

friend istream& operator>>(istream& in, SetValue& a);

Iterator first() { return beg; }//возвращает указатель на первый элемент

Iterator last() { return end; } //возвращает указатель на элемент следующий за последним

};

Файл setValuse.cpp

#include "setValuse.h"

#include <iostream>

using namespace std;

//конструктор с параметрами

SetValue::SetValue(int s, int k)

{

size = s;

data = new int[size];

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

data[i] = k;

beg.elem = &data[0];

end.elem = &data[size - 1];

}

//конструктор копирования

SetValue::SetValue(const SetValue& a){

size = a.size;

data = new int[size];

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

data[i] = a.data[i];

beg.elem = &data[0]; // Указываем на новые данные

end.elem = &data[size - 1]; // Указываем на новый конец

}

//деструктор

SetValue::~SetValue() { delete[] data; data = 0; }

//операция присваивания

SetValue& SetValue::operator=(const SetValue& a){

if (this == &a)

return \*this;

size = a.size;

if (data != 0)

delete[] data;

data = new int[size];

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

data[i] = a.data[i];

beg.elem = &data[0];

end.elem = &data[size];

return \*this;

}

//операция доступа по индексу

int& SetValue::operator[](int index){

if (index < 0 || index >= size) throw 1;

else return data[index];

}

//операция для получения длины вектора

int SetValue::operator()() { return size; }

//операция для разности множеств

SetValue operator-(const SetValue& a, const SetValue& b){

if (a.size < b.size) throw 2;

SetValue temp(a.size);

for (int i = 0; i < a.size; ++i) {

bool flag = true;

for (int j = 0; j < b.size; ++j)

if (a.data[i] == b.data[j]) {

flag = false;

break;

}

if (flag)

temp.data[i] = a.data[i];

}

return temp;

}

//операция удаления

void SetValue::operator--(){

if (data == nullptr) throw 3;

// Создаем новый массив на 1 элемент меньше

int\* newData = new int[size - 1];

// Копируем все элементы, кроме последнего

for (int i = 0, j = 0; i < size - 1; ++i)

newData[j++] = data[i];

delete[] data;

// Обновляем данные

data = newData;

--size;

beg.elem = &data[0];

end.elem = &data[size - 1];

}

//операции для ввода-вывода

ostream& operator<<(ostream& out, const SetValue& a){

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

out << a.data[i] << ' ';

return out;

}

istream& operator>>(istream& in, SetValue& a){

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

in >> a.data[i];

return in;

}

Файл error.h

#pragma once

#include <string>

#include <iostream>

using namespace std;

class error {//класс ошибка

string str;

public:

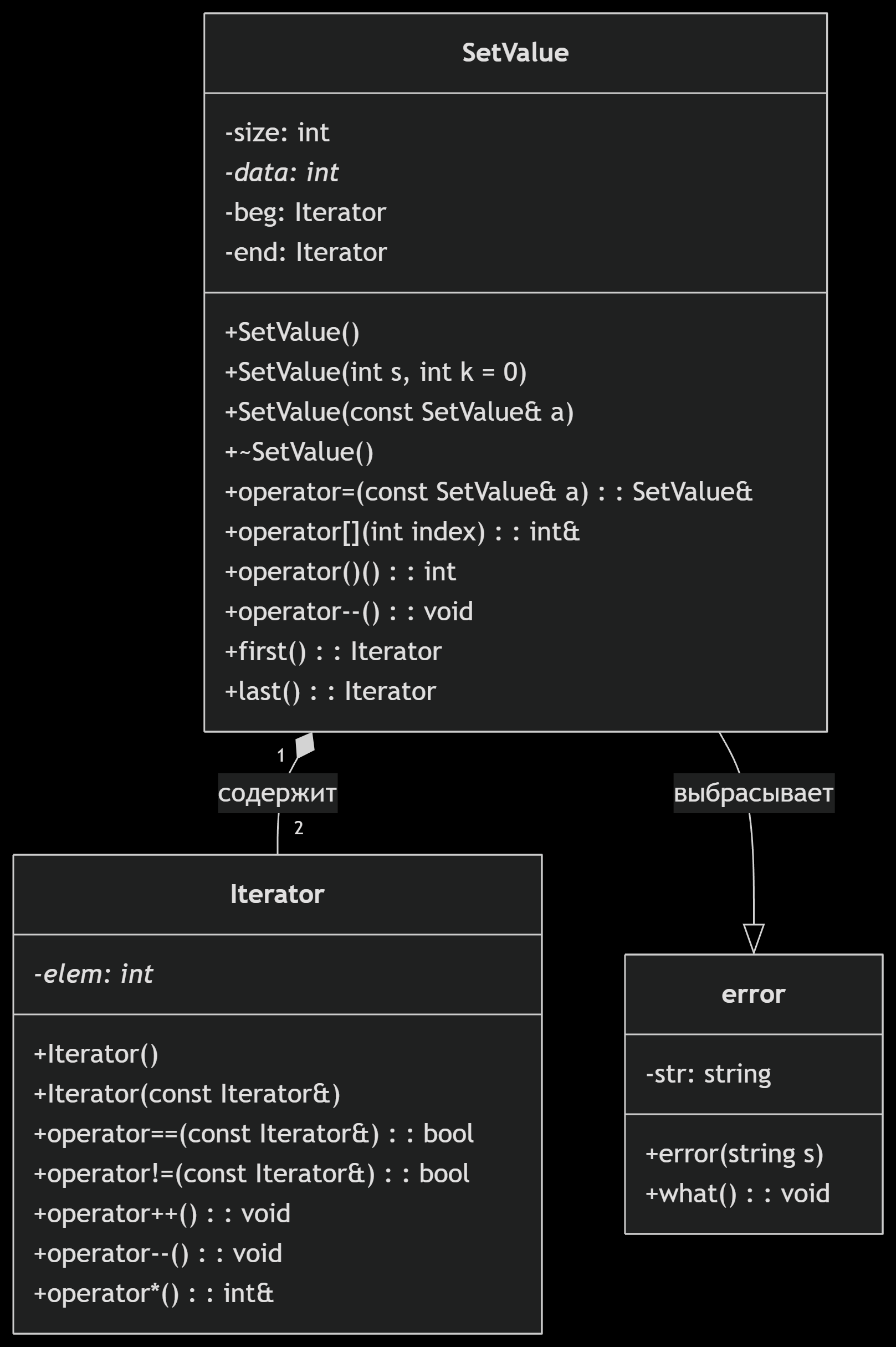
//конструктор, инициирует атрибут str сообщением об ошибке

error(string s) { str = s; }

void what() { cout << str << endl; } //выводит значение атрибута str

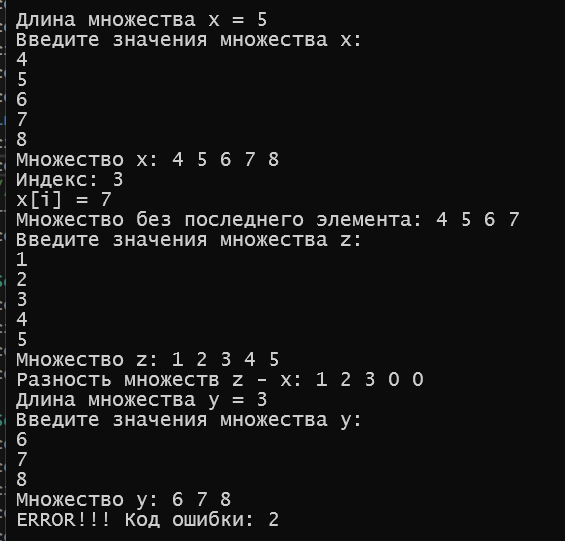
};

UML-диаграмма

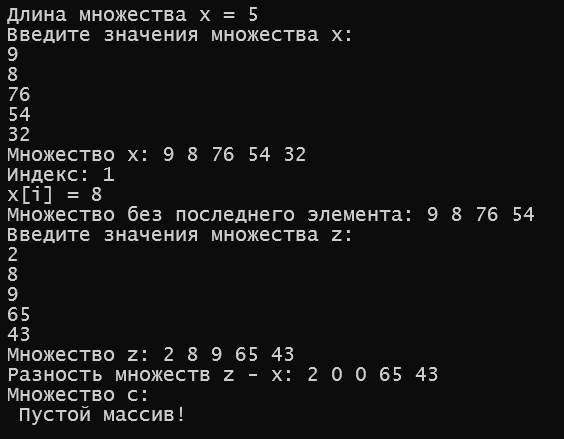


Результат работы программы:

1 метод



2 метод



Контрольные вопросы:

1. Что представляет собой исключение в С++?

Исключение в C++ — это механизм обработки ошибок, который позволяет прервать нормальное выполнение программы при возникновении исключительной ситуации и передать управление специальному блоку кода (обработчику исключения).

1. На какие части исключения позволяют разделить вычислительный процесс? Достоинства такого подхода?

Вычислительный процесс можно разделить на:

1. Блок кода, где может возникнуть исключение (try).
2. Обработчик исключения (catch).

Достоинства:

1. Разделение нормального потока выполнения и обработки ошибок.
2. Возможность централизованной обработки ошибок.
3. Упрощение кода (не нужно проверять возвращаемые значения после каждого вызова).
4. Какой оператор используется для генерации исключительной ситуации?

Оператор throw: throw some\_exception;

1. Что представляет собой контролируемый блок? Для чего он нужен?

Контролируемый блок — это блок кода, заключённый в try, в котором могут возникать исключения. Он нужен для того, чтобы определить область, где исключения будут перехватываться.

1. Что представляет собой секция-ловушка? Для чего она нужна?

Секция-ловушка — это блок catch, который обрабатывает исключение, выброшенное в try. try { /\* код \*/ } catch (const Exception& e) { /\* обработка \*/ }

1. Какие формы может иметь спецификация исключения в секции ловушке? В каких ситуациях используются эти формы?

catch (тип) — ловит исключения определённого типа.

catch (тип&) — ловит исключение по ссылке (эффективнее для объектов).

catch (const тип&) — ловит константную ссылку (рекомендуется для объектов).

catch (...) — ловит любое исключение (используется, если тип исключения неизвестен).

1. Какой стандартный класс можно использовать для создания собственной иерархии исключений?

Базовый класс exception из <stdexcept>.

1. Каким образом можно создать собственную иерархию исключений?

Наследованием от exception или других стандартных исключений:

class MyException : public runtime\_error {

public:

MyException(const tring& msg) : runtime\_error(msg) {}

};

1. Если спецификация исключений имеет вид: void f1() throw(int,double); то какие исключения может прождать функция f1()?

Функция может выбрасывать только int и double. Если выброшено другое исключение, вызовется unexpected().

1. Если спецификация исключений имеет вид: void f1() throw(); то какие исключения может прождать функция f1()?

Функция не должна выбрасывать никаких исключений. Если исключение возникнет, вызовется unexpected().

1. В какой части программы может генерироваться исключение?

В любой части программы, включая:

1. Функции.
2. Конструкторы/деструкторы.
3. Операторы.
4. Блоки try.
5. Написать функцию, которая вычисляет площадь треугольника по трем сторонам (формула Герона).
   * без спецификации исключений:

#include <cmath>

#include <stdexcept>

using namespace std;

double heron(double a, double b, double c) {

if (a <= 0 || b <= 0 || c <= 0) {

throw invalid\_argument("Sides must be positive");

}

if (a + b <= c || a + c <= b || b + c <= a) {

throw invalid\_argument("Triangle inequality violated");

}

double p = (a + b + c) / 2;

return sqrt(p \* (p - a) \* (p - b) \* (p - c));

}

* + со спецификацией throw():

double heron\_noexcept(double a, double b, double c) throw() {

return 0; // Нереалистично, но спецификация требует отсутствия исключений

}

* + с конкретной спецификацией с подходящим стандартным исключением:

double heron\_standard(double a, double b, double c) throw(invalid\_argument) {

if (a <= 0 || b <= 0 || c <= 0) {

throw invalid\_argument("Sides must be positive");

}

if (a + b <= c || a + c <= b || b + c <= a) {

throw invalid\_argument("Triangle inequality violated");

}

double p = (a + b + c) / 2;

return sqrt(p \* (p - a) \* (p - b) \* (p - c));

}

* + спецификация с собственным реализованным исключением:

class TriangleException : public runtime\_error {

public:

TriangleException(const std::string& msg) : runtime\_error(msg) {}

};

double heron\_custom(double a, double b, double c) throw(TriangleException) {

if (a <= 0 || b <= 0 || c <= 0) {

throw TriangleException("Sides must be positive");

}

if (a + b <= c || a + c <= b || b + c <= a) {

throw TriangleException("Triangle inequality violated");

}

double p = (a + b + c) / 2;

return sqrt(p \* (p - a) \* (p - b) \* (p - c));

}