Министерство науки и высшего образования РФ

Федеральное бюджетное государственное образовательное учреждение высшего профессионального образования

Пермский национальный исследовательский политехнический университет (ПНИПУ)

Факультет: Электротехнический (ЭТФ)

Направление: 09.03.01 «Информатика и вычислительная техника (Автоматизированные системы обработки информации и управления)»

Кафедра: «Информационные технологии и автоматизированных систем» (ИТАС)

Основы алгоритмизации

Лабораторная работа №9

Обработка исключительных ситуаций

Студент: Балтаев Э. А.

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

(дата, подпись)

Группа: АСУ-20-1бзу

Работу проверил:

доцент кафедры ИТАС

Полякова О. А.

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_\_\_

Пермь 2021г.

Постановка задачи:

1. Реализовать класс, перегрузить для него операции, указанные в варианте.

2. Определить исключительные ситуации.

3. Предусмотреть генерацию исключительных ситуаций

Задание варианта

Класс-контейнер МНОЖЕСТВО с элементами типа int

Реализовать операции

[] – доступа по индексу;

Int() – определение размера вектора;

\* - пересечение множеств;

-- - удаление элемента из множества.

Реализация ошибок: 1, 2

Описание исключений в операциях:

[] – неверность индекса (отрицательно или больше длины множества)

-- - возможно пустое множество, удалять нечего

<< - возможно пустое множество, выводить нечего

Описание класса

1) Исключение

#pragma once

#include <iostream>

using namespace std;

class error

{

private:

string str;

public:

error(string s) { str = s; }

void what() { cout << "Ошибка: " << str << endl; }

};

2) Множество

#pragma once

#include <iostream>

#include "error.h"

using namespace std;

class Set

{

private:

int size;

int\* data;

public:

Set(int s);

Set(const Set& s);

~Set();

Set& operator= (const Set& v);

friend ostream& operator<< (ostream& out, const Set& v);

friend istream& operator>> (istream& in, Set& v);

int& operator[] (int index); // доступ по индексу

int operator() (); // определение размера

Set operator\* (const Set& s); // пересечение множеств

Set& operator-- (); // префикс

Set operator-- (int); // постфикс

private:

void reset(); // убрать одинаковые

};

Определение компонентных функций

1) Множество

#include "Set.h"

Set::Set(int s)

{

size = s;

data = new int[size];

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

data[i] = i + 1;

}

Set::Set(const Set& s)

{

size = s.size;

data = new int[size];

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

data[i] = s.data[i];

}

Set::~Set()

{

delete[] data;

data = 0;

}

Set& Set::operator=(const Set& v)

{

if (this == &v)

return \*this;

if (size != 0)

delete[] data;

size = v.size;

data = new int[size];

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

data[i] = v.data[i];

return \*this;

}

void Set::reset()

{

int ok, k = 0;

int\* arr = new int[size];

// идем по элементами

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

ok = 0;

// идем по доп. массиву

for (int j = 0; j < k; j++)

// если он там есть

if (data[i] == arr[j]) {

ok = 1; // отмечаем, что копировать не надо

break;

}

// если его там нет

if (ok == 0) {

arr[k] = data[i];

k++;

}

}

// копируем

if (size != 0)

delete[] data;

size = k;

data = new int[size];

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

data[i] = arr[i];

delete[] arr;

}

int& Set::operator[](int index)

{

if (index < 0 || index >= size)

throw 1;

return data[index];

}

int Set::operator()()

{

return size;

}

Set Set::operator\*(const Set& s)

{

int count, k = 0;

if (size >= s.size)

count = size;

else

count = s.size;

int\* arr = new int[count];

// идем по элементами

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

// идем по элементами другого множества

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

// если они равны

if (data[i] == s.data[j]) {

arr[k] = data[i]; // копируем

k++;

break;

}

// копируем

Set temp(k);

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

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

delete[] arr;

return temp;

}

// ИСПОЛЬЗОВАНИЕ 2-го варианта реализации оисключений

// префикс

Set& Set::operator--()

{

if (size == 0)

throw error("пустое множество");

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

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

arr[i] = data[i];

delete[] data;

data = arr;

size--;

return \*this;

}

// постфикс

Set Set::operator--(int)

{

if (size == 0)

throw error("пустое множество");

Set temp(\*this);

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

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

arr[i] = data[i];

delete[] data;

data = arr;

size--;

return temp;

}

ostream& operator<<(ostream& out, const Set& v)

{

if (v.size == 0)

throw error("пустое множество");

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

out << v.data[i] << " ";

return out;

}

istream& operator>>(istream& in, Set& v)

{

cout << "Введите " << v.size << " элементов через пробел: ";

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

in >> v.data[i];

}

v.reset();

return in;

}

Определение главного файла

#include <iostream>

#include "Set.h"

int main()

{

system("chcp 1251");

cout << "\nСоздание двух множеств\n";

int n1, n2;

cout << "n1? "; cin >> n1;

Set s1(n1);

cin >> s1;

cout << "n2 "; cin >> n2;

Set s2(n2);

cin >> s2;

cout << "s1: " << s1 << endl;

cout << "s2: " << s2 << endl;

cout << "\nГенерация исключений с помощью стандартных типов\n";

try {

cout << "s1[2] = " << s1[2] << endl;

cout << "s1[6] = " << s1[6] << endl;

cout << "s1[10] = " << s1[10] << endl;

}

catch (int) {

cout << "Ошибка: не верный индекс" << endl;

}

cout << "\nГенарция исключений с помощью собственного типа\n";

try {

cout << "before s2--: " << s2 << endl;

s2--;

cout << "after s2--: " << s2 << endl;

s2--;

cout << "after s2--: " << s2 << endl;

}

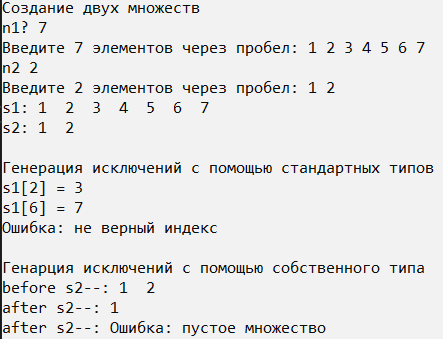
catch (error& err) {

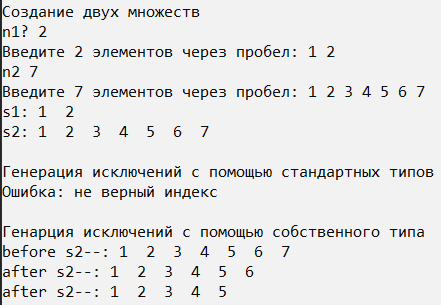
err.what();

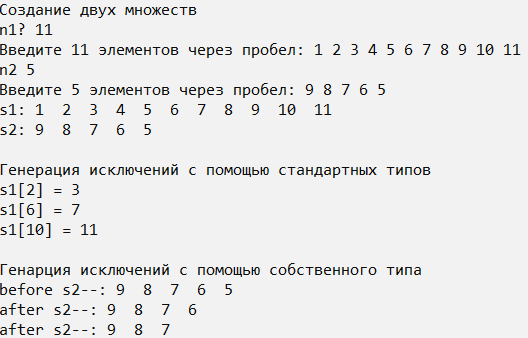
}

}

Результаты работы программы







Ответы на контрольные вопросы:

1) Исключение – объект, который система должна генерировать при возникновении исключительной ситуации. Генерация такого объекта и создает исключительную ситуацию

2) На 2 части: обнаружение аварийной ситуации и обработка аварийной ситуации. Достоинство: удобно в модульных программах и не требуется возвращение функций.

3) throw

4) try { … } – безопасное выполнение кода внутри блока

5) catch(ошибка) { … } – обработка исключения

6) (тип имя) и (тип) – конкретные исключения; (…) – все исключения

7) exception

8) BaseException – ChildrenException: public BaseException …

9) Исключение с типом int, double

10) Все искоючения

11) Во всех частях

12)

void square(float a, float b, float c)

{

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

throw exception();

float p = a + b + c;

return pow(p\*(p-a)\*(p-b)\*(p-c) ,0.5)

}

void square(float a, float b, float c) throw()

{

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

throw exception();

float p = a + b + c;

return pow(p\*(p-a)\*(p-b)\*(p-c) ,0.5)

}

void square(float a, float b, float c) throw(runtime\_error)

{

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

throw runtime\_error(“error”);

float p = a + b + c;

return pow(p\*(p-a)\*(p-b)\*(p-c) ,0.5)

}

void square(float a, float b, float c) throw(BaseError)

{

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

throw SquareError(“Not triangle”);

float p = a + b + c;

return pow(p\*(p-a)\*(p-b)\*(p-c) ,0.5)

}