Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

Дисциплина: «Основы алгоритмизации и программирования»

Лабораторная работа №9

Тема : "Обработка исключительных ситуаций"

Выполнил работу

Студент группы РИС-22-1Б

Лихачев Д.А.

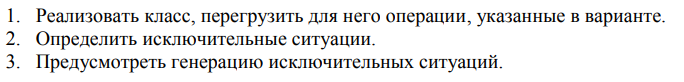
Проверила

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**

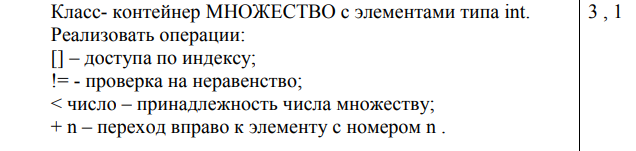




ВАРИАНТ РЕАЛИЗАЦИИ 3.

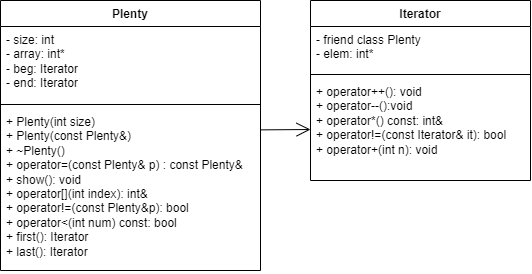


Вариант 9

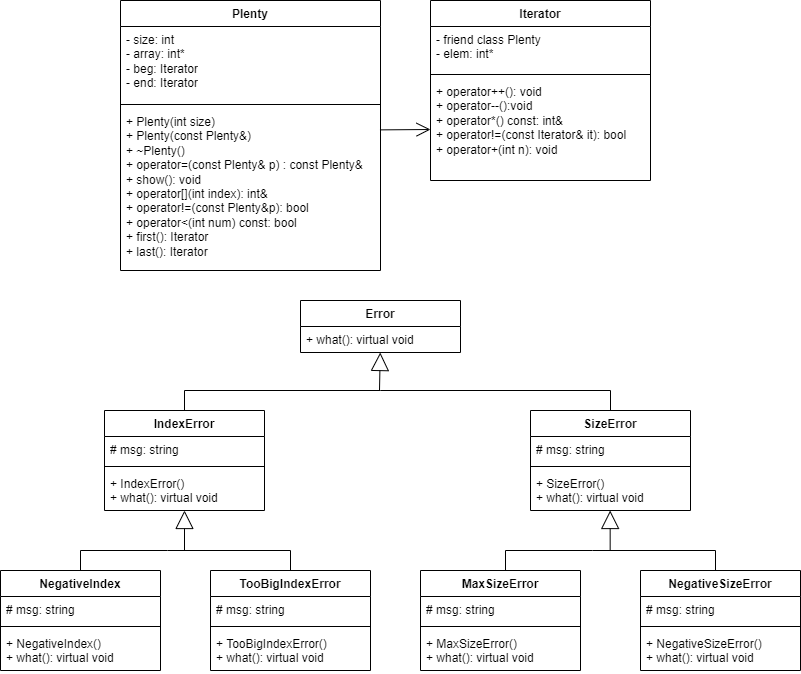


**Диаграмма классов**

**9.1**



**9.3**



**Error.h**

class Error

{

public:

virtual void what() {}

};

class IndexError : public Error // недопустимый индекс множества

{

protected:

string msg;

public:

IndexError() { msg = "Недопустимый индекс множества."; }

virtual void what() { cout << msg; }

};

class SizeError : public Error // недопустимый размер множества

{

protected:

string msg;

public:

SizeError() { msg = "Недопустимый размер множества."; }

virtual void what() { cout << msg; }

};

class MaxSizeError : public SizeError

{

protected:

string msg\_;

public:

MaxSizeError() { SizeError(); msg\_ = "Превышен максимальный размер."; }

virtual void what() { cout << msg << msg\_; }

};

class NegativeSizeError : public SizeError

{

string msg\_;

public:

NegativeSizeError() { SizeError(); msg\_ = "Отрицательный размер недопустим."; }

virtual void what() { cout << msg << msg\_; }

};

class NegativeIndex: public IndexError

{

string msg\_;

public:

NegativeIndex() { IndexError(); msg\_ = "Отрицательный индекс недопустим."; }

virtual void what() { cout << msg << msg\_; }

};

class TooBigIndexError : public IndexError

{

string msg\_;

public:

TooBigIndexError() { IndexError(); msg\_ = "Индекс больше текущего размера множества."; }

virtual void what() { cout << msg << msg\_; }

};

**Описание класса-контейнера**

const int MAX\_SIZE = 6;

class Plenty

{

private:

int size;

int\* array;

Iterator beg;

Iterator end;

public:

Plenty(int size);

Plenty(const Plenty& p);

~Plenty();

const Plenty& operator=(const Plenty& p);

void show();

int& operator [](int index);

Iterator first();

Iterator last();

bool operator!=(const Plenty& rhs);

bool operator<(int num) const;

};

**Определение компонентных функций**

// конструктор

Plenty::Plenty(int size)

{

if (size > MAX\_SIZE) throw MaxSizeError(); // если размер больше максимального, то генерируется исключение

if (size <= 0) throw NegativeSizeError(); // размер должен быть больше 0

this->size = size;

array = new int[size];

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

{

array[i] = rand() % 100 + 1;

}

beg.elem = &array[0];

end.elem = &array[size];

}

// конструктор копирования

Plenty::Plenty(const Plenty& p)

{

if (p.size > MAX\_SIZE) throw MaxSizeError(); // если размер больше максимального, то генерируется исключение

size = p.size;

array = new int[size];

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

{

array[i] = p.array[i];

}

beg.elem = &array[0];

end.elem = &array[size];

}

// деструктор

Plenty::~Plenty() { delete[] array; size = 0; }

// операция присваивания

const Plenty& Plenty::operator=(const Plenty& p)

{

if (this == &p) return \*this;

delete[] array;

size = p.size;

array = new int[size];

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

{

array[i] = p.array[i];

}

return \*this;

}

void Plenty::show()

{

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

{

cout << array[i] << " ";

}

}

// операция проверки на неравенство

bool Plenty::operator!=(const Plenty& rhs) {

bool f = false;

int i = 0;

while (!f) {

if (array[i] != rhs.array[i]) f = true;

i++;

}

return f;

}

// операция доступа по индексу

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

{

if (index < 0) throw NegativeIndex(); // если индекс отрицательный, то генерируется исключение

if (index > size) throw TooBigIndexError(); // если индекст больше размера множества, то генерируется исключение

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

{

return array[index];

}

else

{

cout << endl << "Error" << endl;

}

}

bool Plenty::operator<(int num) const

{

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

if (array[i] == num) {

return true;

}

}

return false;

}

Iterator Plenty::first() { return beg; }

Iterator Plenty::last() { return end; }

**Функция main (9.3)**

int main()

{

setlocale(0, "Rus");

// недопустимый размер множества, превышен максимальный размер

try

{

Plenty a(10);

}

catch (Error& e)

{

e.what();

}

cout << endl;

// недопустимый размер множества, отрицательный размер

try

{

Plenty b(-1);

}

catch (Error& e)

{

e.what();

}

cout << endl;

// недопустимый индекс, слишком большой индекс

try

{

Plenty c(5);

int n = c[10];

}

catch (Error& e)

{

e.what();

}

cout << endl;

// недопустимый индекс, отрицательный индекс недопустим

try

{

Plenty d(5);

int f = d[-1];

}

catch (Error& e)

{

e.what();

}

return 0;

}

**Функция main (9.1)**

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(0));

int size = 10;

try

{

Plenty a(size); // ошибка с кодом 1

}

catch (int num)

{

cout << "Ошибка с кодом: " << num << endl;

cout << "1 - выбранный размер множества недопустим. 2 - выбранный индекс недопустим." << endl;

}

try

{

Plenty a(5);

int n = a[6];

}

catch (int num)

{

cout << "Ошибка с кодом: " << num << endl;

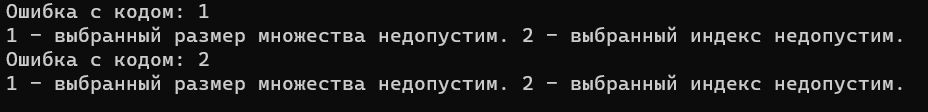
cout << "1 - выбранный размер множества недопустим. 2 - выбранный индекс недопустим." << endl;

}

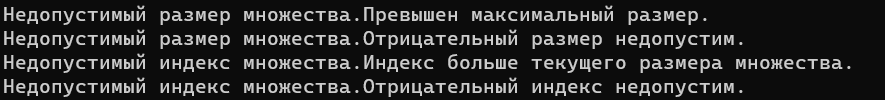
return 0;

}

**Результат работы программы (9.1)**



**Результат работы программы (9.3)**



**Ответы на контрольные вопросы**

1. Что представляет собой исключение в С++?

Исключение - непредвиденное или аварийное событие.

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

Исключения позволяют разделить вычислительный процесс на 2 части:

1) обнаружение аварийной ситуации (неизвестно как обрабатывать);

2) обработка аварийной ситуации (неизвестно, где она возникла).

Достоинства такого подхода:

1) удобно использовать в программе, которая состоит из нескольких модулей;

2) не требуется возвращать значение в вызывающую функцию

3. Какой оператор используется для генерации исключительной ситуации?

Исключение генерируется оператором

***throw <выражение>***, где выражение>

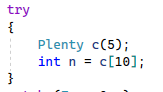
* либо константа,
* либо переменная некоторого типа,
* либо выражение некоторого типа.

Тип объекта-исключения может быть как встроенным, так и определяемым пользователем.

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

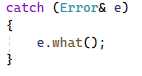
Контролируемый блок - это блок кода, в котором создаются локальные переменные, инициализирующие объекты, и при выходе из блока автоматически вызываются деструкторы объектов.

Контролируемый блок используется для гарантированного выполнения действий при выходе из блока, независимо от того, какой путь был выбран при выходе.



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

Секция-ловушка (catch-блок) - это блок кода, который используется для перехвата и обработки исключений, которые могут возникнуть в блоке try-блоке.



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

- catch без аргумента типа: catch (...). Эта форма используется для перехвата любого исключения, которое не было перехвачено ранее. Она может быть полезна, например, для записи сообщения в журнал перед завершением программы.

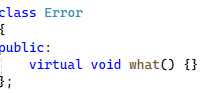
- catch с аргументом типа std::exception: catch (std::exception& e). Эта форма используется для перехвата исключений, производных от класса std::exception. Она позволяет получить доступ к информации об исключении, например, к его сообщению, которое можно использовать для вывода пользователю.

- catch с аргументом типа T, где T - это класс исключения: catch (T& e). Эта форма используется для перехвата конкретного класса исключения T. Она позволяет получить доступ к информации об исключении, которую можно использовать для его обработки.

- catch с несколькими аргументами типа: catch (T1& e1, T2& e2, ...). Эта форма используется для перехвата нескольких исключений разных типов. Она позволяет обработать каждое исключение по-разному в зависимости от его типа.

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

class Error



8. Каким образом можно создать собственную иерархию исключений?

Для создания собственной иерархии исключений необходимо определить классы исключений, которые будут наследоваться от стандартного класса Error или его производных классов.

9. Если спецификация исключений имеет вид: void fl()throw(int,double); то какие

исключения может порождать функция fl()?

Функция fl() может порождать исключения типа int и double

10. Если спецификация исключений имеет вид: void fl()throw(); то какие исключения может порождать функция f1()?

Функция fl() не может порождать никаких исключений. Это означает, что если в процессе выполнения функции возникнет исключение, то оно не будет обработано внутри этой функции и будет передано на уровень выше в стеке вызовов.

11. В какой части программы может генерироваться исключение?

В любой части программы.

12. Написать функцию, которая вычисляет площадь треугольника по трем сторонам (формула Герона).

Функцию реализовать в 4 вариантах:

- без спецификации исключений;

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

    double p = (a + b + c) / 2;

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

}

- со спецификацией throw();

double triangleArea(double a, double b, double c) throw() {

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

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

}

- с конкретной спецификацией с подходящим стандартным исключением;

#include <stdexcept>

double triangleArea(double a, double b, double c) throw(std::invalid\_argument) {

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

throw std::invalid\_argument("Invalid triangle sides");

}

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

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

}

- спецификация с собственным реализованным исключением.

#include <exception>

class InvalidTriangleException : public std::exception {

public:

virtual const char\* what() const throw() {

return "Invalid triangle sides";

}

};

double triangleArea(double a, double b, double c) throw(InvalidTriangleException) {

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

throw InvalidTriangleException();

}

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

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

}