Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации

Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное учреждение

высшего образования

Московский технический университет связи и информатики

(МТУСИ)

Кафедра «Информационная безопасность»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №10

по дисциплине

«Программирование в системах информационной безопасности»

на тему

«Исключения»

Вариант №19

Выполнил:

студент группы БСУ1801

Плотников П.С

Проверил:

старший преподаватель кафедры ИБ

Барков В. В.

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

Москва 2021

# **Цель**

Изучить механизм обработки исключительных ситуаций в языке C++.

# **Задание**

Исключение – это непредвиденное или аварийное событие. В С++ исключение – это объект, который система должна генерировать при возникновении исключительной ситуации. Генерация такого объекта и создает исключительную ситуацию. Исключения позволяют разделить вычислительный процесс на 2 части:

1. обнаружение аварийной ситуации (неизвестно как обрабатывать);
2. обработка аварийной ситуации (неизвестно, где она возникла).

Исключение генерируется оператором throw <выражение>, где <выражение> является либо константой, либо переменной некоторого типа, либо выражением некоторого типа. Тип объекта-исключения может быть как встроенным, так и определяемым пользователем. Для представления исключений часто используют пустой класс: classZeroDevide{}; classNegativeArg{}; Генерация исключения будет выглядеть: throwZeroDevide();//вызывается конструктор без параметровили thrownewZeroDevide();

Исключение надо перехватить и обработать. Для проверки возникновения исключения используется контролируемый блок try{}, с которым связана одна или несколько секций-ловушек catch. Все переменные, объявленные в этом блоке, являются локальными для этого блока.

Форма записи секции-ловушки следующая:

catch( спецификация исключения ),

где спецификация исключения может иметь три формы:

1) (тип имя)

2) (тип)

3) (...)

Тип – это встроенный тип или тип, определенный программистом. Формы 1 и 2 обрабатывают конкретные исключения, а форма 3 перехватывает все исключения, такую ловушку надо помещать последней, тогда она будет обрабатывать все исключения, которые еще не были обработаны. Форма 1 означает, что объект передается в блок обработки, чтобы его каким-то образом там использовать, например, для вывода информации в сообщении об ошибке.

Примеры:

catch( exceptione) // по значению

catch( exception &e) // поссылке

catch( const exception &e) // по ссылке на константу

catch( exception \*e) //по указателю

Лучше всего передавать объект по ссылке, т. к. при этом не создается временный объект-исключение.

Для каждой функции, метода, конструктора или деструктора можно в заголовке указать спецификацию исключений. Если в заголовке спецификация исключений не указана, считается, что функция может порождать любое исключение, если указана, то считается, что функция генерирует те исключения, которые явно указаны в этом списке.

Примеры:

void f1() throw(int,double);

void f2() throw(ZeroDivide);

Если спецификация имеет вид: void f() throw(); то считается, что функция исключений не генерирует. Наличие спецификаций исключения не является ограничением при реальной генерации исключений. Но если функция генерирует неспецифицированное исключение, то запускается стандартная функция unexpected(), которая вызывает функцию terminate(), что приводит к аварийному завершению программы. При отсутствии подходящей секции-ловушки осуществляется вызов стандартной функции завершения terminate(). Эта функция вызывается из функции unexpected() при нарушении спецификации завершения. Обе функции можно подменить собственными реализациями. Для этого необходимо

1. Подключить заголовок #include <expection>

2. Определить собственную функцию с прототипом void F() для подмены стандартной функции terminate().

3. Указать имя этой функции в вызове функции set\_terminate(F);

После этого вместо terminate() будет вызываться наша функция F(). Такая функция не должна возвращать управление оператором return или генерировать исключение throw(), она может только завершить программу функцией exit() или abort().

Аналогично реализуется подмена стандартной функции unexpected(): set\_ unexpected(F); Функция может сгенерировать неспецифицированное исключение, в этом случае, если в спецификации исключений не указано исключение bad\_exception, вызывается функция terminate(), в противном случае сгенерированное исключение подменяется на bad\_exception и начинается поиск его обработчика.

Преобразовать разработанный в практикуме №9 класс.

Разработать классы исключениий OverflowException и NoElementsException.

Функции Push, PushFront и PushBack не должны возвращать никакого значения и должны выбрасывать исключение OverflowException в случае нехватки места.

Функции Pop, PopBack, PopFront, Peek, PeekBack, PeekFront, должны возвращать копию элемента и выбрасывать исключение NoElementsException в случае если в контейнере нет элементов

# **Индивидуальное задание**

|  |  |  |
| --- | --- | --- |
| Вариант №19 | Очередь на основе двунаправленного циклического списка | QueueBasedOnBidirectionalCyclicLinkedList |

# **4. Выполнение**

Выполнение задания:

1. **Класс с целыми значениями элементов**

Листинг 1 – Исходный код заголовочного файла, который содержит описание класса

#include <iostream>

using namespace std;

struct Element

{

int number;

Element\* next;

Element\* prev;

};

class Queue

{

Element\* head;

Element\* tail;

public:

Queue();

Queue(const Queue& other);

Queue(Queue&& other)noexcept;

~Queue();

void push(int element);

int pop();

int peek();

int get\_size();

Queue& operator=(const Queue& other);

Queue& operator=(Queue&& other)noexcept;

friend ostream &operator<<(ostream& stream, const Queue& that)

{

if (that.head == NULL) return stream << "Очередь пуста";

Element\* temp = that.head;

while (temp != that.tail)

{

stream << temp->number << " ";

temp = temp->prev;

}

return stream << temp->number;

}

};

Листинг 2 - Исходный код основного файла, который содержит описание методов класса и функцию main

#include "Header1.h"

Queue::Queue()

{

head = tail = NULL;

}

Queue::Queue(const Queue& other)

{

if (other.head == NULL)

{

head = tail = NULL;

return;

}

Element\* temp = other.head;

while (temp != other.tail)

{

this->push(temp->number);

temp = temp->prev;

}

this->push(temp->number);

}

Queue::Queue(Queue&& other)noexcept

{

if (other.head == NULL)

{

head = tail = NULL;

return;

}

head = other.head;

tail = other.tail;

other.head = other.tail = NULL;

}

Queue::~Queue()

{

if (head == NULL) return;

while (this->get\_size() > 0)

{

this->pop();

}

head = tail = NULL;

}

void Queue::push(int element)

{

Element\* temp = new Element;

temp->number = element;

if (head != NULL)

{

temp->next = tail;

temp->prev = head;

tail->prev = temp;

head->next = temp;

tail = temp;

}

else temp->next = temp->prev = head = tail = temp;

}

int Queue::pop()

{

if (head == tail)

{

if (head == NULL) return 0;

int res = head->number;

delete head;

head = tail = NULL;

return res;

}

int res = head->number;

head = head->prev;

delete head->next;

head->next = tail;

tail->prev = head;

return res;

}

int Queue::peek()

{

if (head == NULL) return 0;

return head->number;

}

int Queue::get\_size()

{

int i = 0;

Element\* temp = tail;

if (head == NULL) return 0;

while (temp != head)

{

i++;

temp = temp->next;

}

return i + 1;

}

Queue& Queue::operator=(const Queue& other)

{

if (this == &other || other.head == NULL) return \*this;

Element\* temp = other.head;

while (temp != other.tail)

{

this->push(temp->number);

temp = temp->prev;

}

this->push(temp->number);

return \*this;

}

Queue& Queue::operator=(Queue&& other)noexcept

{

if (this == &other || other.head == NULL) return \*this;

head = other.head;

tail = other.tail;

other.head = other.tail = NULL;

return \*this;

}

int main()

{

setlocale(LC\_ALL, "ru");

cout << "Выберите действие с очередью:" << endl;

cout << "0. Выйти из программы" << endl;

cout << "1. Вставить элемент в конец" << endl;

cout << "2. Удалить и вывести головной элемент" << endl;

cout << "3. Вывести головной элемент" << endl;

cout << "4. Вывести длину очереди" << endl;

cout << "5. Вывести все элементы очереди" << endl;

cout << "6. Создать новую очередь с копированием и вывести ее элементы" << endl;

cout << "7. Создать новую очередь с перемещением и вывести ее элементы" << endl;

cout << "8. Копировать 1 очередь во 2 и вывести ее элементы" << endl;

cout << "9. Копировать 1 очередь во 2 с перемещением и вывести ее элементы" << endl;

Queue queue;

int opt;

cin >> opt;

while (opt != 0)

{

switch (opt)

{

case 1:

{

int a;

cout << "Введите число для вставки: ";

cin >> a;

queue.push(a);

break;

}

case 2:

{

cout << "Значение удаленного головного элемента: " << queue.pop() << endl;

break;

}

case 3:

{

cout << "Значение головного элемента: " << queue.peek() << endl;

break;

}

case 4:

cout << "Длина очереди: " << queue.get\_size() << endl;

break;

case 5:

cout << queue << endl;

break;

case 6:

{

Queue queue2 = queue;

cout << queue2 << endl;

break;

}

case 7:

{

Queue queue3 = move(queue);

cout << queue3 << endl;

break;

}

case 8:

{

Queue queue2;

queue2 = queue;

cout << queue2 << endl;

break;

}

case 9:

{

Queue queue3;

queue3 = move(queue);

cout << queue3 << endl;

break;

}

default:

cout << "Введено недопустимое значение" << endl;

}

cin >> opt;

}

return 0;

}

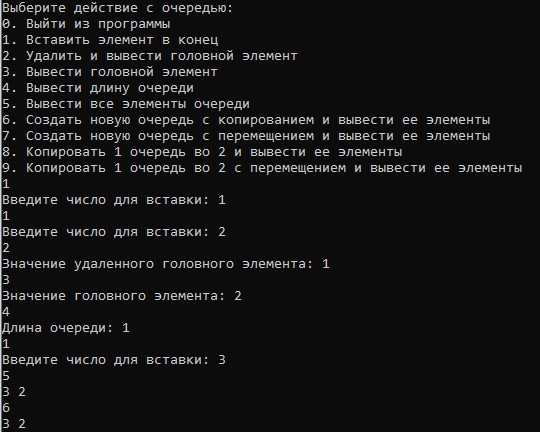


Рисунок 1 – Пример работы программы

**<https://github.com/Pavel-Plotnikov-BSU1801/Laboratornai-10-Plotnikov-P.S.-BSU1801>**

# **Вывод**

В этой лабораторной работе я изучил динамические структуры данных, овладел навыками создания конструкторов копирования, перемещения, деструкторов, перегрузки операций копирования и перемещения.