**Черкасов Александр 1 вариант Лабораторная работа №2-3 А-08-19**

**Содержание**

[**Задание (2.1)** 1](#_Toc51623285)

[**Задание (2.2)** 1](#_Toc51623286)

[**1. Постановка задачи** 1](#_Toc51623287)

[**2. Разработка программы** 1](#_Toc51623288)

[2.1 Разработка функций и методов классов 1](#_Toc51623289)

[2.2 Разработка интерфейса 3](#_Toc51623290)

[**3. Реализация и тестирование программы** 4](#_Toc51623291)

[3.1 Описание разработанной программы 4](#_Toc51623292)

[3.2 Тестирование программы 5](#_Toc51623293)

[**Вывод** 9](#_Toc51623294)

[**Приложение. Код программы** 9](#_Toc51623295)

# **Задание (2.1)**

Описать класс, реализующий десятичный счетчик, который может увеличивать или уменьшать свое значение на единицу в заданном диапазоне.

Предусмотреть инициализацию счетчика значениями по умолчанию и произвольными значениями. Счетчик имеет два метода: увеличения и уменьшения, — и свойство, позволяющее получить его текущее состояние.

При выходе за границы диапазона выбрасываются исключения. Написать программу, демонстрирующую все разработанные элементы класса.

# **Задание (2.2)**

Создать дочерний класс Счетчик1, который может увеличивать или уменьшать свое значение на любое введенное число в заданном диапазоне.

# **1. Постановка задачи**

Разработать объектно-ориентрованную программу в соответствии с заданиями 2.1 и 2.2

**Функции**: Ввод и вывод данных о показаниях счётчиков, реализация методов по увеличению или уменьшению показаний в заданном диапазоне.

**Входные данные**:

Для конструктора: нет/начальное состояние счетчика

Для функции увеличения/уменьшения: значение для увеличения/уменьшения счетчика

**Выходные данные**: Измененные показания счетчиков

**Ограничения**:

Показания счетчиков – целое число в диапазоне [-2048 2048]. При увеличении или уменьшении нельзя выходить за рамки диапазона.

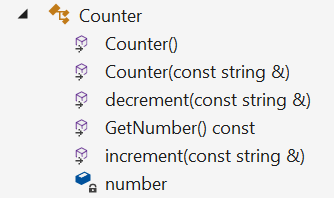
**Вид приложения** - консольное приложение на языке C++.

**Среда разработки** – CLion

# **2. Разработка программы**

2.1 Разработка функций и методов классов

Для работы программы необходимо создать базовый класс Counter, который будет представлять собой методы увеличения и уменьшения показаний счетчиков, а также получение показаний. Диаграмма классов приведена на *рис. 2.1.* для визуальной связи между классами. В таблицах *2.1*, *2.2*, *2.3* описаны конструкторы, поля и методы, а также функции программы.



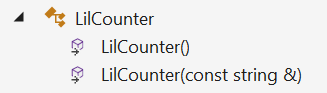


Рис. 2.1. Диаграмма классов

Табл. 2.1. Конструкторы

|  |  |  |
| --- | --- | --- |
| Название | Конструкторы | Описание |
| **Counter**  Счетчик  (базовый класс) | **Counter();** | Конструктор по умолчанию. Поле number (показания счетчика) = 000 |
| **explicit Counter(const string& new\_number);** | Параметризированный конструктор. Принимает на вход строку и обрабатывает ее внутри конструктора. |
| **LilCounter**  Счетчик  (наследник класса Counter) | **LilCounter();** | Конструктор по умолчанию. Поле number берет из класса Counter, равное 000 |
| **explicit LilCounter(const string& n) : Counter(n) {}** | Параметризированный конструктор. Принимает на вход строку и обрабатывает ее в классе Counter. |

Табл. 2.2. Поля и методы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Поля | Методы | Входные данные | Выходные данные |
| **Counter**  Счетчик  (базовый класс) | String number – показания счетчика | string **GetNumber()** const – получение показаний счетчика. | Нет | Строка – показания счетчика |
| void **Increment(const string& i)** – увеличение показаний на вводимое значение | Строка – число, на которое нужно увеличить | Нет |
| void **Decrement(const string& i)** – уменьшение показаний на вводимое число | Строка – число, на которое нужно уменьшить | Нет |
| **LilCounter**  Счетчик  (наследник класса Counter) | Нет полей, так как он использует поле класса Counter | Нет методов, так как он использует те же самые методы, что и Counter |  |  |

Табл. 2.3. Функции

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Назначение | Входные данные | Выходные данные |
| int **Perevod(const string& num)** | Перевод шестнадцатеричного числа в десятичное | Строка – число в 16сс, которое надо перевести | Число в 10сс типа int |
| char\* **Upper(char\* numb)** | Перевод строки в верхний регистр | Указатель на массив символов (строка которую надо перевести) | Указатель на массив символов (строка в верхнем регистре) |
| bool **ParseCondition(const string& s)** | Проверка входных данных на корректность | Строка – число в 16сс | Если строка корректна – return true; Иначе false |
| string **ParseString(string& s)** | Обработка входной строки на пробелы | Строка – число в 16сс | Строка без лишних пробелов – число в 16сс |

2.2 Разработка интерфейса

Разработано консольное приложение, которое предоставляет пользователю меню (*рис. 2.2*), где он может выбрать с какими значениями счетчика работать (по умолчанию или вводить самому), а затем выбрать что делать с ними (увеличивать или уменьшать). Программа будет подсказывать на сколько можно увеличить или уменьшить счетчик, чтобы не выйти за границы диапазона. В случае выхода за границы выбросится исключение.

Исключения выбрасываются с помощью команды *throw runtime\_error(“Причина ошибки”);* Далее они ловятся в блоке *try-catch.*

По окончанию работы со счетчиком начнется работа со счетчиком-наследником, чтобы показать, что он также обрабатывает команды.

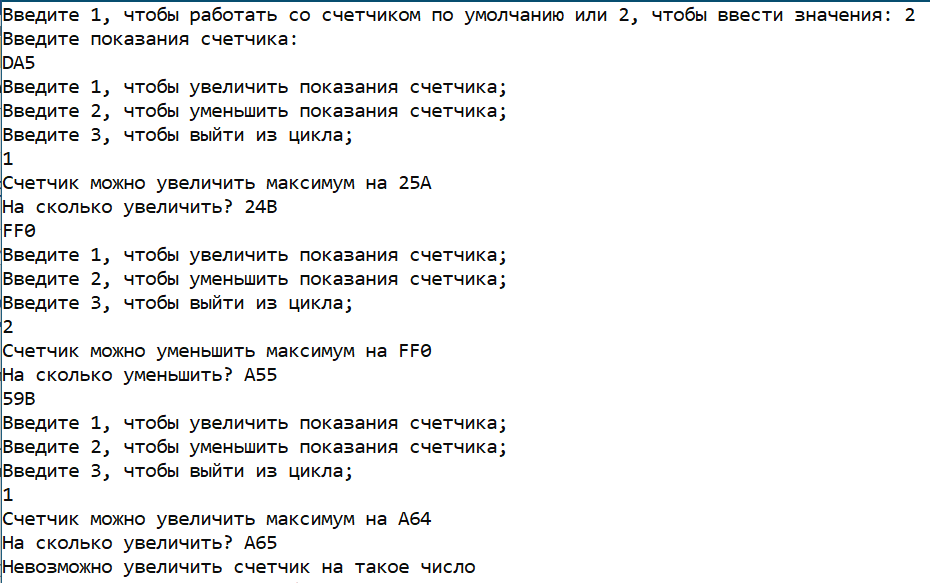


Рис. 2.2. Пример диалога

# **3. Реализация и тестирование программы**

3.1 Описание разработанной программы

Для реализации программы в качестве подхода выбрано объектно-ориентированное программирование.

Разработано консольное приложение на языке С++

Среда программирования: Microsoft Visual Studio 2019

В заголовочных файлах .h содержится интерфейс классов, в .cpp файлах – реализация классов

*Класс Counter*. Содержит два конструктора: параметризированный и по умолчанию. Первый принимает строку и проверяет ее на корректность. Второй присваивает полю number значение 000. В классе реализовано три метода: GetNumber, increment и decrement. Первый метод будет возвращать текущее значение счетчика. Второй и третий будут изменять приватное поле number, в котором хранятся показания счетчика.

Counter::Counter() { //Конструктор по умолчанию

number = "000";

}

Counter::Counter(const string& new\_number) { //Параметризированный конструктор

if (new\_number.size() > 3) { //Длина входящей строки не может быть больше 3

throw runtime\_error("Кол-во ячеек должно быть 3"); //Выбрасывание исключения

}

if (!ParseCondition(new\_number)) { //Проверка входных данных

throw runtime\_error("Неверный формат данных"); //Выбрасывание исключения

}

number = new\_number; //Если все верно

}

*Класс LilCounter*. Класс-наследник от класса Counter. Использует методы из класса Counter, поэтому их реализация в данном классе не требуется. Содержит два конструктора: по умолчанию и параметризированный. При работе со вторым, применяя списки инициализации, мы принимаем входящую строку и проверяем ее на корректность, с помощью конструктора, реализованного в базовом классе.

LilCounter::LilCounter() {} //Конструктор по умолчанию

LilCounter::LilCounter(const string& n) : Counter(n) { //Параметризированный конструктор

} //Здесь применили списки инициализации; Входная строка будет обрабатываться с помощью обработки в Counter

3.2 Тестирование программы

Табл. 3.1. Тесты

|  |  |  |
| --- | --- | --- |
| **№ теста** | **Смысл теста** | **Результат** |
| **1** | Проверка **стандартных конструкторов** | counter::get = 0  counter1::get = 0 |
| **2** | Проверка **конструкторов с параметром** | counter::get = 10  counter1::get = 10 |
| **3** | Проверка функции **increase** | counter::get = 1  counter::get = 5 |
| **4** | Проверка функции **decrease** | counter::get = -1  counter1::get = -4 |
| **5** | Проверка выхода за **верхний предел** | В всех случаях  std::runtime\_error  (“counter::out\_of\_range”) |
| **6** | Проверка выхода за **нижний предел** |  |
| **7** | Проверка выхода за **верхний предел** в **конструкторе** |  |
| **8** | Проверка выхода за **нижний предел** в **конструкторе** |  |
| **9** | Проверка **отрицательного параметра** для **increase** | counter1::get = -1 |
| **10** | Проверка **отрицательного параметра** для **decrease** | counte1::get = 1 |
| **11** | Попытка ввести **строку** в параметры **конструктора** | std:: runtime\_error |
| **12** | Попытка ввести **строку** в параметры **increase** | Во всех случаях  std::invalid\_argument |
| **13** | Попытка ввести **строку** в параметры **decrease** |  |

# **Вывод**

Проделанная работа помогла ознакомиться с основами ООП. Здесь затронуты важные темы: Наследование и Инкапсуляция. Наследование позволяет экономить много времени. Объекты производного класса свободно могут использовать всё, что создано и отлажено в базовом классе. Инкапсуляция ограничивает доступ к полям класса, что в итоге помогает избегать возможные ошибки и нежелательные изменения. Это неотъемлемые и очень мощные части ООП.

# **Приложение. Код программы**

//Лабораторная работа №2-3, Харитонов Андрей, А-12-19, 2 вариант

**----------------------------------------------Header.h----------------------------------------------**

#pragma once

#include <iostream>

#include <exception>

#include <windows.h>

#include <string>

using namespace std;

class Counter {

public:

Counter(); //Конструктор по умолчанию

explicit Counter(const string& new\_number); //Параметризированный конструктор (explicit - против неявного преобразования)

string GetNumber() const; //Метод для получения показаний счетчика

void increment(const string& i); //Метод - увеличение значения на i

void decrement(const string& i); //Метод - уменьшение значения на i

private: //Приватные поля

string number; //Строка, состоящая из трех символов

};

int Perevod(const string& num); //Функция перевода из 16-ричной системы в 10-тичную

char\* Upper(char\* numb); //Функция перевода строки в верхний регистр

bool ParseCondition(const string& s); //Проверка входных данных

class LilCounter : public Counter { //Класс-наследник класса Counter

public:

LilCounter(); //Конструктор по умолчанию

explicit LilCounter(const string& n); //Параметризированный конструктор

};

string ParseString(string& s); //Функция для обработки вхоядщей строки

**----------------------------------------------Source.cpp----------------------------------------------**

#include "Header.h"

Counter::Counter() { //Конструктор по умолчанию

number = "000";

}

Counter::Counter(const string& new\_number) { //Параметризированный конструктор

if (new\_number.size() > 3) { //Длина входящей строки не может быть больше 3

throw runtime\_error("Кол-во ячеек должно быть 3"); //Выбрасывание исключения

}

if (!ParseCondition(new\_number)) { //Проверка входных данных

throw runtime\_error("Неверный формат данных"); //Выбрасывание исключения

}

number = new\_number; //Если все верно

}

string Counter::GetNumber() const { //Реализация метода получения показаний счетчика

return number;

}

void Counter::increment(const string& i) { //Реализация метода увеличения

char\* str = new char[number.size()+ 1]; //Вспомогательная строка для перевода в 16-рирчную систему

int num = Perevod(number); //Переводим показания счетчика в 10-тичную систему

int sl = Perevod(i); //Переводим в 10-тичнуюю систему число, на которое надо увеличить

num += sl; //Увеличиваем

number = Upper(itoa(num, str, 16)); //Переводим в 16-ричную систему num и сразу переводим в верхний регистр

if (number.size() == 2) { //Для красоты вывода добавим нолик в начало

number = "0" + number;

} else

if (number.size() == 1) { //Для красоты вывода добавим два нолика в начало

number = "00" + number;

}

if (number.size() > 3) { //Проверяем, что не переборщили с увеличением

throw runtime\_error("Невозможно увеличить счетчик на такое число");

}

}

void Counter::decrement(const string& i) { //Аналогично методу increment

char\* str = new char[number.size() + 1]; //Вспомогательная строка

int num = Perevod(number); //Перевод показаний в 10 сс

int sl = Perevod(i); //Перевод в 10 сс числа, на которое уменьшим

num -= sl; //Уменьшаем

if (num < 0) { //Проверяем, что не слишком сильно уменьшили

throw runtime\_error("Невозможно уменьшить счетчик на такое число");

}

number = Upper(itoa(num, str, 16)); //Переводим полученное число в 16 сс и в верхний регистр

if (number.size() == 2) { //Для красоты вывода добавим нолик в начало

number = "0" + number;

} else

if (number.size() == 1) { //Для красоты вывода добавим два нолика в начало

number = "00" + number;

}

}

int Perevod(const string& num) { //Функция для перевода числа в 10 сс

int p = num.size()-1; //Степень, в которую будем возводить 16

int res = 0; //Результирующая переменная

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

if (isdigit(num[i])) { //Если i - число

int j = num[i] - '0'; //Переводим из char в int

res += j \* pow(16.0, p); //Добавляем к res

p--; //Уменьшаем степень

}

else if (num[i] == 'A' || num[i] == 'a') { //Если не число, то проверяем каждую букву и делаем аналогичные действия

res += 10 \* pow(16.0, p);

p--;

}

else if (num[i] == 'B' || num[i] == 'b') {

res += 11 \* pow(16.0, p);

p--;

}

else if (num[i] == 'C' || num[i] == 'c') {

res += 12 \* pow(16.0, p);

p--;

}

else if (num[i] == 'D' || num[i] == 'd') {

res += 13 \* pow(16.0, p);

p--;

}

else if (num[i] == 'E' || num[i] == 'e') {

res += 14 \* pow(16.0, p);

p--;

}

else if (num[i] == 'F' || num[i] == 'f') {

res += 15 \* pow(16.0, p);

p--;

}

}

return res;

}

char\* Upper(char\* numb) { //Функция для перевода в верхний регистр

for (size\_t i = 0; i < sizeof(numb); i++) {

if (numb[i] >= 'a' && numb[i] <= 'z') { //В цикле проходимся по строке

numb[i] = toupper(numb[i]); //Если не число, то изменяем регистр буквы

}

}

return numb;

}

bool ParseCondition(const string& s) { //Функция для проверки входных данных

bool ok = true; //Результирующая переменная

for (size\_t i = 0; i < s.size(); i++) { //Проходимся по строке

if ((s[i] <= '9' && s[i] >= '0') || (s[i] == 'a' || s[i] == 'A'

|| s[i] == 'b' || s[i] == 'B' || s[i] == 'c'

|| s[i] == 'C' || s[i] == 'd' || s[i] == 'D'

|| s[i] == 'e' || s[i] == 'E' || s[i] == 'f'

|| s[i] == 'F')) {

}

else ok = false; //Если хоть раз условие нарушается, то ok = false;

}

return ok;

}

LilCounter::LilCounter() {} //Конструктор по умолчанию

LilCounter::LilCounter(const string& n) : Counter(n) { //Параметризированный конструктор

} //Здесь применили списки инициализации; Входная строка будет обрабатываться с помощью обработки в Counter

string ParseString(string& s) { //Функция для обработки строки на пробелы

string res;

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

if (s[i] != ' ') {

res += s[i]; //Если символ не пробел, то записываем в результирующую строку

}

}

return res; //Возвращаем res

}

**----------------------------------------------main.cpp----------------------------------------------**

#include "Header.h"

using namespace std;

int main() {

SetConsoleCP(1251); //Устанавливаем русский язык

SetConsoleOutputCP(1251);

try { //Заходим в блок try-catch для обработки исключений

int k = 0; //Переменная длявыбора счетчика

cout << "Введите 1, чтобы работать со счетчиком по умолчанию или 2, чтобы ввести значения: ";

while (k != 1 && k != 2) { //Пока не 1 и не 2

cin >> k; //Вводим k

while (cin.fail()) { //Если извлечение из потока оказалось неудачным

cout << "Нет такой команды" << endl;

cin.clear(); //Возвращаем cin в обычный режим работы

cin.ignore(32767, '\n'); //Удаляем значения предыдущего ввода

cin >> k; //Вновь вводим k

}

if (k != 1 && k != 2) {

cout << "Нет такой команды" << endl;

}

}

Counter c; //Объявляем счетчик

if (k == 1) { //Выводим значения если он поумолчанию

cout << c.GetNumber() << endl;

cin.ignore(32767, '\n'); //Перед следующим cin надо сделать этот шаг для очистки буфера

}

else if (k == 2) {

string s; //Показания счетчика запишем в строку

cout << "Введите показания счетчика:" << endl;

cin.ignore(32767, '\n');

getline(cin, s); //Получаем строку с помощью getline, чтобы пробелы также записывались

s = ParseString(s); //Обрабатываем строку и удаляем все пробелы

c = Counter(s); //Объявляем объект класса counter

}

int x; //Переменная для реализации меню

char\* str1 = new char[c.GetNumber().size() + 1]; //Строка для записи в нее 16-ричный вид 10-тичного числа

do {

cout << "Введите 1, чтобы увеличить показания счетчика;" << endl

<< "Введите 2, чтобы уменьшить показания счетчика;" << endl

<< "Введите 3, чтобы выйти из цикла;" << endl;

cin >> x; //Выбираем операцию

if(cin.fail()) { //Если извлечение из потока оказалось неудачным

cin.clear(); //Возвращаем cin в обычный режим работы

cin.ignore(32767, '\n'); //Удаляем значения предыдущего ввода

}

string i; //В i записываем на сколько уменьшаем или увеличиваем

switch (x) {

case 1:

//Вычисляем на сколько можно увеличить счетчик для удобства пользователя

cout << "Счетчик можно увеличить максимум на " << Upper(itoa(4095 - Perevod(c.GetNumber()), str1, 16)) << endl;

cout << "На сколько увеличить? ";

cin.ignore(32767, '\n'); //Перед getline удаляем символы из буфера, чтобы избежать ошибок

getline(cin, i); //Вводим 16-ричное число

i = ParseString(i); //Избавляемся от пробелов

if (ParseCondition(i)) { //Проверяем на корректность данных вводимое число

c.increment(i); //Делаем увеличение

}

else { //В случае некорректности выбрасываем исключение

throw runtime\_error("Неверный формат данных");

}

cout << c.GetNumber() << endl; //Выводим показания

break;

case 2:

//Вычисляем на сколько можно уменьшить счетчик

cout << "Счетчик можно уменьшить максимум на " << Upper(itoa(Perevod(c.GetNumber()), str1, 16)) << endl;

cout << "На сколько уменьшить? ";

cin.ignore(32767, '\n'); //Перед getline удаляем символы из буфера, чтобы избежать ошибок

getline(cin, i); //Вводим 16-ричное число

i = ParseString(i); //Избавляемся от пробелов

if (ParseCondition(i)) { //Проверяем на корректность данных вводимое число

c.decrement(i); //Делаем уменьшение

}

else { //В случае некорректности выбрасываем исключение

throw runtime\_error("Неверный формат данных");

}

cout << c.GetNumber() << endl; //Выводим показания

break;

case 3:

break;

default:

cout << "Нет такой команды" << endl;

break;

}

} while (x != 3);

//----------------------------------ВТОРАЯ ЧАСТЬ (РАБОТА С ОБЪЕКТОМ-НАСЛЕДНИКОМ) ------------------------

int k1 = 0; //Переменная для выбора счетчика

cout << "Введите 1, чтобы работать со счетчиком по умолчанию или 2, чтобы ввести значения: ";

while (k1 != 1 && k1 != 2) { //Пока не 1 и не 2

cin >> k1; //Выбираем

while (cin.fail()) { //Если извлечение из потока оказалось неудачным

cout << "Нет такой команды" << endl;

cin.clear(); //Возвращаем cin в обычный режим работы

cin.ignore(32767, '\n'); //Удаляем значения предыдущего ввода

cin >> k1;

}

if (k1 != 1 && k1 != 2) {

cout << "Нет такой команды" << endl;

}

}

LilCounter lc; //Объявляем объект-наследник

if (k1 == 1) { //Если k1 = 1 то по значения по умолчанию

cout << lc.GetNumber() << endl;

cin.ignore(32767, '\n');

}

else if (k1 == 2) { //Иначе вводим

string s1; //Показания счетчика запишем в строку

cout << "Введите показания счетчика:" << endl;

cin.ignore(32767, '\n'); //Очищаем буфер

getline(cin, s1); //Получаем строку с помощью getline, чтобы пробелы также записывались

s1 = ParseString(s1); //Обрабатываем строку и удаляем все пробелы

lc = LilCounter(s1); //Объявляем объект класса LilCounter

}

int x1; //Переменная для реализации меню

char\* str2 = new char[c.GetNumber().size() + 1]; //Вспомогательная строка

do {

cout << "Введите 1, чтобы увеличить показания счетчика;" << endl

<< "Введите 2, чтобы уменьшить показания счетчика;" << endl

<< "Введите 3, чтобы выйти из цикла;" << endl;

cin >> x1; //Выбираем операцию

if (cin.fail()) { //В случае неудачного извлечения

cin.clear(); //Возвращаем cin в обычный режим

cin.ignore(32767, '\n'); //Очищаем буфер

}

string i; //В i записываем на сколько уменьшаем или увеличиваем

switch (x1) {

case 1:

//Вычисляем на сколько можно увеличить счетчик для удобства пользователя

cout << "Счетчик можно увеличить максимум на " << Upper(itoa(4095 - Perevod(lc.GetNumber()), str2, 16)) << endl;

cout << "На сколько увеличить? ";

cin.ignore(32767, '\n'); //Перед getline удаляем символы из буфера, чтобы избежать ошибок

getline(cin, i); //Вводим 16-ричное число

i = ParseString(i); //Избавляемся от пробелов

if (ParseCondition(i)) { //Проверяем на корректность данных вводимое число

lc.increment(i); //Делаем увеличение

}

else { //В случае некорректности выбрасываем исключение

throw runtime\_error("Неверный формат данных");

}

cout << lc.GetNumber() << endl; //Выводим показания

break;

case 2:

//Вычисляем на сколько можно уменьшить счетчик

cout << "Счетчик можно уменьшить максимум на " << Upper(itoa(Perevod(lc.GetNumber()), str2, 16)) << endl;

cout << "На сколько уменьшить? ";

cin.ignore(32767, '\n'); //Перед getline удаляем символы из буфера, чтобы избежать ошибок

getline(cin, i); //Вводим 16-ричное число

i = ParseString(i); //Избавляемся от пробелов

if (ParseCondition(i)) { //Проверяем на корректность данных вводимое число

lc.decrement(i); //Делаем уменьшение

}

else { //В случае некорректности выбрасываем исключение

throw runtime\_error("Неверный формат данных");

}

cout << lc.GetNumber() << endl; //Выводим показания

break;

case 3:

break;

default:

cout << "Нет такой команды" << endl;

break;

}

} while (x1 != 3);

}

catch (const exception& e) { //Если было выброшено исключение

cout << e.what() << endl; //Выводим причину ошибки

}

system("pause"); //Задерживаем консоль перед выходом

return 0;

}