Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет инфокоммуникаций

Кафедра инфокоммуникационных технологий

Е. Г. Макейчик, В. В. Чепикова, О. Г. Шевчук

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для специальности 1-45 01 01 «Инфокоммуникационные технологии (по направлениям)»

Минск БГУИР 2020

УДК 004.42(076)
ББК 32.973я73
М15

Р е ц е н з е н т ы:

кафедра автоматики, телемеханики и связи учреждения образования
«Белорусский государственный университет транспорта»
(протокол №10 от 14.11.2019);

начальник кафедры связи учреждения образования
«Военная академия Республики Беларусь»
кандидат технических наук, доцент А. А. Пилюшко

Макейчик, Е. Г.

М15 Объектно-ориентированное программирование : учеб.-метод. пособие / Е. Г. Макейчик, В. В. Чепикова, О. Г. Шевчук. – Минск : БГУИР, 2020. – 80 с.
ISBN 978-985-543-568-7.

Даны краткие теоретические сведения и порядок выполнения восьми лабораторных работ по объектно-ориентированному программированию с использованием среды разработки C++.

УДК 004.42(076)
ББК 32.973я73

ISBN 978-985-543-568-7

Содержание

Лабораторная работа №1
Среда разработки Visual Studio. Введение в тип данных "класс"

Цель работы: изучить основы программирования в среде Visual Studio 2017, научиться создавать простейшие проекты, познакомиться со структурой объектно-ориентированной программы.

Теоретические сведения

Visual Studio – это интегрированная среда разработки программ, объединяющая текстовый редактор, компилятор, компоновщик и отладчик. Visual Studio выполняет компиляцию и запуск программ в соответствии с проектом. Проект – это структура данных, содержащая всю информацию, необходимую для компиляции исходных файлов программы и ее компоновки со стандартными библиотеками (например, библиотекой ввода/вывода).

Компиляция и компоновка исходных файлов называется сборкой проекта. В результате успешной сборки Visual Studio создает приложение (двоичный исполняемый файл программы).

Объектно-ориентированное программирование (ООП) – методология, основанная на представлении программ в виде совокупности объектов, которые между собой взаимодействуют, причем каждый объект является реализацией конкретного класса. То есть основной элемент ООП – класс.

Класс – это тип данных, вводимый пользователем. Классы, как правило, организованы иерархически. Основное назначение класса – описание состава, основных свойств и поведения будущих объектов этого типа данных (введение в проекте некоей абстракции). При этом каждый класс имеет общедоступную часть – интерфейс, а также недоступную, скрытую от пользователя личную часть (или реализацию), которая и представляет внутреннее строение будущих объектов данного типа. Таким образом, в ООП абстрагирование – процесс введения типа данных "класс", т. е. таких существенных характеристик некоторых будущих объектов, которые и будут отличать их от других видов объектов. Общий формат декларации типа данных "класс" следующий:

вид_класса ID_класса (идентификатор определенного типа данных)
{элементы-данные; // Далее – просто "данные" элементы-функции;
// Далее – просто "методы" };

Рассмотрим краткую характеристику элементов, составляющих описание класса.

Атрибут "вид_класса". Виды типа данных "класс": union, struct, class.

Атрибуты доступа "Элементы", входящие в шаблон класса, могут иметь следующую степень защищенности:

1) public – глобальный, общедоступный атрибут доступа; в этом случае элементы общедоступны из любой функции проекта;

2) private – локальный, частный; элементы закрыты от общего доступа, и с ними могут работать только методы класса;

3) protected – защищенные элементы класса, к ним имеют доступ методы данного класса и методы классов, производных от данного класса.

Как правило, защищенные элементы – это внутреннее строение объектов, которое нас не должно интересовать.

Виды типа данных "класс" имеют степень защиты элементов, которую компилятор установит по умолчанию:

1) вид union – степень защиты public, управлять ею нельзя;

2) вид struct – степень защиты public, управлять защитой можно;

3) вид class – степень защиты private, ею также можно управлять.

Для управления защитой элементов, входящих в состав класса, необходимо явно указать нужный атрибут, после которого добавить символ ":" (двоеточие).

Пример:

```
class X
{
int a, b; //Поля a и b будут иметь доступ private по умолчанию
public:
double a1, a2;; //Поля a1 и a2 будут иметь доступ public
protected:
char p, l; //Поля p и l будут иметь доступ protected
};
```

Существует два способа создания объектов с таким составом данных:

1) Между символами "}" и ";" помещается список идентификаторов объектов, и уже на этапе компиляции будут созданы объекты с указанным типом данных. При таком способе задания атрибут ID_класса можно опускать.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Test
{       public:
        int a, b;
        double c; };
m, *t = &m; //Инициализируем объект m типа Test и указатель этого типа t, который указывает на наш объект m
int main()
```

```
{
//Поля объекта m в данном пример не инициализированы, так как в классе не объявлен конструктор для их инициализации
// Инициализировать поля можно вручную
t->a = 5; //Не забываем про то, что t – это указатель и сейчас мы даем значение 5 полю "a" для объекта m через косвенное обращение
m.b = 3; //Даем значение 3 полю "b" объекта m прямым обращением
cout<<t->a; //Косвенное обращение к полю "a" объекта m и вывод его значения
cout << endl << m.b; //Прямое обращение к полю "b" объекта m и вывод его значения
}
```

2) Декларация объектов класса осуществлятся в любом месте программы по мере надобности: ID_класса список ID_объектов;

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Test
{
public:
int a, b;
double c;
};
int main()
{
//Поля переменных t и m в данном пример не инициализированы, так как в классе не объявлен конструктор для их инициализации
Test x; //Инициализируем объект x
X.a = 4; //Даем значение полю a объекта x
Test *y; //Создаем указатель
y = &x; //Указатель теперь указывает на место расположения объекта x в памяти
cout<<y->a; //Косвенное обращение к полю "a" объекта x, так как y хранит указатель на адрес объекта x в памяти
cout<<endl<<x.a; //Прямое обращение к полю "a"
}
```

Данные, входящие в состав класса, – любой допустимый вид данных языка C++, за исключением файлов.

Методы, входящие в состав класса, можно разделить на три группы.

1) Методы, определяющие операции над данными.

2) Функции-конструкторы, основная задача которых – создание и инициализация объектов данного класса разнообразными способами.

3) Функции для уничтожения объектов после их использования – деструкторы.

Внешние функции, не входящие в состав класса, могут работать только с элементами класса, имеющими атрибут доступа public.

Декларация методов класса возможна в двух формах:

1) в форме определения, при которой приводится полный текст метода;

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Test
{
public:
int a, b;
double c;
void show() //Полный текст метода написан внутри класса
{        cout << a<< b;
}
};
int main()
{
Test x; //Инициализируем объект x
x.a = 5;
x.b = 9;
x.c = 0;
x.show();
}
```

2) в форме описания (прототип метода): в данном случае вне шаблона должно быть полное определение метода, при этом необходимо указать, к какому классу принадлежит данный метод, для этого используется операция привязки "::", назначение этой операции – привязать функцию к конкретному классу.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Test
{
public:
int a, b;
```

```
double c;
void show(); //Инициализируем метод в классе
};
```

void Test::show() //Пишем код метода, Test:: – говорит о том, что данный код должен выполниться при вызове метода show(), инициализированного в классе Test

```
{
cout << a << b;
}
void show() //Мы можем создать метод с таким же названием и такой же сигнатурой, но для объекта код этого метода не выведется, так как нет привязки к классу
{
cout << "User metods.";
}
int main()
{
Test x; //Инициализируем объект x
x.a = 5;
x.b = 9;
x.c = 0;
x.show(); //Output: 59
show(); //Output: User metods.
}
```

Использование этой операции вводит новое определение: полное квалификационное имя X::a означает, что элемент **a** принадлежит классу **X**. Запись X::f1() означает, что метод **f1** принадлежит классу **X**.

И в заключение рассмотрим некоторые приемы работы с объектами и методами.

Пусть в процессе работы программы создан объект класса ID_класса ID_объекта и пусть в составе класса есть метод с идентификатором ID_метода, тогда существует два способа обращения к методам:

1) Прямой вызов метода ID_объекта.ID_метода; – использована операция привязки "точка".

```
#include "pch.h"
#include <iostream>
using namespace std;
class Test
{
public:
int a, b;
double c;
```

```
void show() //Полный текст метода написан внутри класса
{                cout << a << b;        }};
int main()
{
Test x; //Инициализируем объект x
//прямой вызов полей
x.a = 5;
x.b = 9;
x.c = 0;
//Прямой вызов метода
x.show();
}
```

2) Косвенная адресация – объявлен указатель с типом ID_класса *ID_указателя. Теперь указатель типа ID_класса можно устанавливать на существующие объекты этого класса: ID_указателя = & ID_объект; – указатель идентифицирован адресом объекта.

Косвенный вызов метода: ID_указателя -> ID_метода; – использована операция привязки "стрелка".

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Test
{
public:
int a, b;
double c;
void show() //Полный текст метода написан внутри класса
{                cout << a << b;        }
};
int main()
{
Test x; //Инициализируем объект x
Test *y; //Инициализируем указатель типа Test
y = &x; //Теперь указатель y указывает на адрес объекта x и теперь мы можем косвенно к нему обращаться
//Косвенный вызов полей
cout << y->a << y->b << y->c;
//Косвенный вызов метода
y->show();
}
```

Создание объектов (безымянных) в динамической области памяти ID_класса * ID_указателя = new ID_класса; Компилятор создает объект на этапе работы проекта. Косвенный вызов метода: ID_указателя -> ID_метода;

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Test
{
public:
int a, b;
double c;
void show() //Полный текст метода написан внутри класса
{
        cout << a << b;
}
};
int main()
{
Test *y = new Test(); //Создаем динамический объект в памяти, на расположение которого указывает указатель y
//Косвенный вызов полей
cout << y->a << y->b << y->c;
//Косвенный вызов метода
y->show();
}
```

Задания к лабораторной работе

Уровень 1

Упражнение: напишите объектную программу, иллюстрирующую прямой и косвенный способы обращения к методам. Пользовательский класс должен содержать необходимые элементы: данные, метод установки их начальных значений, метод просмотра текущего состояния полей класса, метод, решающий поставленную задачу. Описание метода должно быть вне класса.

Варианты:

1) Вы свободный фрилансер, к вам обратилась компания "BestGameFromEarth" и предложила работу.

Ваша задача: создать класс Player, в котором будет храниться HP, MagicPower, Stamina (все поля типа int), и написать метод, который будет выводить крутость игрока. Крутость игрока рассчитывается по формуле

$$(HP \cdot MagicPower)/Stamina + (HP \cdot Stamina)/MagicPower - (Stamina \cdot MagicPower)/HP;$$

Не забывайте про деление на нуль. В этом случае должно быть выведено сообщение об ошибке с указанием на поле, которое было == 0).

2) Вы младший научный сотрудник научно-практического центра исследований "Explore all you want", и сегодня вы изучаете mildew. Цель вашего исследования – определить количество mildew через определенный промежуток времени. Вам известны: количество начальной mildew, ее прирост в день и количество дней, которое она будет размножаться с такой скоростью.

Ваша задача: создать класс Mildew, в котором будут 3 поля (типа int) – speed, day, kol – написать метод, который вернет количество mildew через промежуток времени, введенный в поле day.

3) Сегодня вы заступили на должность главврача больницы "Healthy like a bull". И первое, что бы вы хотели узнать, – это процент выздоровевших, больных и симулянтов от общего числа пациентов.

Ваша задача: создать класс с 3 полями (типа int) – convalescents, sick, simulators – и метод, который вернет количество выздоровевших, больных и симулянтов в процентах от общего числа пациентов. Общее количество пациентов: $(convalescents + sick + simulators)$.

4) В банке "We are the most honest", где вы проходите стажировку, хотят поставить новый автомат для оплаты услуг. Механизм работы автомата очень простой: клиент вставляет свою карточку, выбирает услугу, которую он хочет оплатить, и, если у него хватает денег на карточке, услуга оплачивается. Однако есть одно "но". Начисляется комиссия в размере 5 % от цены услуги (необходимо внимательно читать правила использования автомата, которые приклеены позади него).

Ваша задача: создать класс Machine, в котором будет 3 поля (2 int, 1 double): money, price, residue. Пользователь вводит значения для полей money и price (residue изначально == 0), поле residue заполняется при выполнении метода, который посчитает остаток после оплаты операции (не забываем про списываемые автоматом 5 %). В случае проведения операции метод выводит сообщение об успешном ее завершении, в случае нехватки денег – сообщение о несостоявшемся действии.

Уровень 2

Условие такое же, как и в упражнении уровня 1. Выполнение задания уровня 2 должно происходить в классах, созданных для выполнения уровня 1, весь функционал и поля уровня 1 должны быть сохранены.

Варианты:

1) Вы успешно выполнили заказ, но заказчик вспомнил, что персонаж еще имеет физический и магический урон, и просит срочно добавить эти поля, и вывести значения этого урона у персонажа с бонусом, получаемым от Hp, MagicPower и Stamina. Вы приступаете к работе.

Ваша задача: добавить в ранее написанный класс 2 поля (тип double) – MagicDamage, PhysicalDamage – и еще один метод, который будет выводить магический и физический урон по формулам:

Магический урон: $(MagicDamage \cdot MagicPower) - Stamina$;

Физический урон: $(PhysicalDamage \cdot Stamina + Hp) - MagicPower$.

2) Вы успешно написали программу и, вычислив предполагаемое количество mildew через N дней, ушли в отпуск, но, вернувшись, обнаружили, что количество mildew не совпадает с тем значением, которое прогнозировала ваша программа. Ошибку в программе вы исключили сразу (вы же лично ее писали – в ней не может быть ошибок), затем поняли, в чем дело: каждый день, помимо прибавления mildew, шло и отмирание старой. Вычислив количество, которое умирает в день, вы готовы доделать программу.

Ваша задача: добавить поле (тип int) DeathSpeed и в методе, который должен выводить количество mildew через промежуток времени day, учесть эту переменную.

3) Вы получили статистику пациентов. А что насчет персонала? Сколько в целом людей в вашей организации? Это необходимо подсчитать в процентном отношении.

Ваша задача: добавить 3 поля (тип int) – Doctor, Nurse, MedicalOrderly – и метод, который выведет количество сотрудников каждого типа в процентах от общего количества сотрудников.

4) Вашему начальнику очень нравится работа автомата для оплаты каких-то услуг, и он хочет добавить туда новые опции. Теперь пользователь должен иметь возможность пополнить счет на своей карте, взяв деньги у банка, а потом в течение месяца вернуть ровно столько, сколько он указал в автомате (аналогично обещанному платежу на телефоне). Однако за выполнение операции на автомате взимается 5 % от суммы. То есть, если пользователь возьмет 100 рублей у банка в долг, то автомат возьмет комиссию с этих денег 5 % и перечислит пользователю 95 рублей, а вернуть пользователь должен будет 100 рублей.

Ваша задача: добавить поле (тип int) – Debt – и метод, который увеличит значение money на Debt с вычетом 5 % комиссии и выведет старое значение money, значение money после пополнения и сумму долга перед банком.

Контрольные вопросы и задания

1) Что является основным элементом ООП? Что входит в его состав?

2) Перечислите виды классов и укажите их различия.

3) Укажите способы защиты элементов, входящих в состав класса.

4) Что такое операция привязки? Приведите пример.

5) Дайте понятие полного квалификационного имени.

Лабораторная работа №2
Использование конструктора и деструктора. Работа с динамической памятью

Цель работы: изучить способы использования элементов управления в среде Visual Studio 2017, а также основные способы работы по созданию конструкторов с захватом динамической памяти и деструкторов для ее освобождения.

Теоретические сведения

Для создания и инициализации объекта используется функция, получившая название конструктор. Она определяет (контролирует), как создается и инициализируется объект.

После использования объектов их необходимо уничтожить. Для этого предназначена функция деструктор, которая определяет, как и когда уничтожить объект.

Если этих методов явно в программе нет, компилятор при создании экземпляра класса вызовет стандартный конструктор, который и создаст объект после его декларации. После выхода из функции (из блока), в которой был создан объект, компилятор вызовет деструктор "по умолчанию", который его уничтожит. Если же объект создавался как глобальный, деструктор будет вызван по завершении работы программы.

Некоторые особенности конструктора и деструктора:

1) Эти функции не имеют возвращаемого значения (даже void).

2) Они не наследуются производными классами, хотя и вызываются из них.

3) Нельзя работать с адресами этих функций.

4) К конструктору нельзя обратиться напрямую как к обычному методу.

5) Деструктор можно вызывать, используя его полное квалификационное имя.

6) Если деструктор или конструктор объявлены в поле private или protect, то они не смогут быть вызваны.

Рассмотрим функции конструктора и деструктора более подробно.

Конструктор – функция-член класса, определяющая способ создания объекта. Имя конструктора должно совпадать с именем класса. Существует несколько форм конструктора. Наиболее часто используется конструктор с параметрами,применяемыми для одновременной инициализации создаваемых объектов. Конструктор как функция может иметь умалчиваемые значения параметров. В программе может быть несколько конструкторов. Их часто называют конструкторами с перекрытием. Здесь работает механизм перегрузки функций.

Пример (для лучшего понимания целесообразно запустить этот код с пошаговым выполнением при помощи клавиши f11):

```
#include "pch.h"
```

```
#include <iostream>
using namespace std;
class Test
{
public:
int a, b;
double c;
Test() //Конструктор без параметров
{
	a = 5;
	b = 6;
	c = 7;
}
Test(int r, int t) //Конструктор с параметрами
{
	a = r;
	b = t;
	c = 0;
}
void show()
{
	cout <<" "<< a << " " << b << " " << c;
}
};
int main()
{
Test a; //Вызовет конструктор без параметров
Test b(12,34); //Вызовет конструктор с параметрами
a.show(); //Output: 5 6 7
b.show(); //Output: 12 34 0
}
```

Конструктор – это специальная функция класса, поэтому допускается вторая форма его декларации: в виде прототипа и его определения вне пространства класса.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Test
{
public:
int a, b;
double c;
```

```
Test(); //Конструктор без параметров
Test(int r, int t); //Конструктор с параметрами
void show()
{
        cout <<" "<< a << " " << b << " " << c;
}
};
Test::Test() //Пишем код конструктора без параметров
{
a = 2;
b = 42;
c = 90;
}
Test::Test(int r, int t) //Пишем код конструктора с данной сигнатурой
{
a = r;
b = t;
c = 23;
}
int main()
{
Test a; //Вызовет конструктор без параметров
Test b(12,34); //Вызовет конструктор с параметрами
a.show(); //Output: 2 42 90
b.show();//Output: 12 34 23
}
```

Деструктор – специальная функция класса, которая отвечает за уничтожение объекта. Имя деструктора совпадает с именем класса, перед которым ставится символ тильда – "~".

Если декларирован class X {...}; то деструктор для него ~X(){}.

Пример:

```
class Test
{
~Test() //Деструктор класса Test
{
};
};
class ABCD
{
~ABCD() //Деструктор класса ABCD
{
};
};
```

То есть если элементы-данные класса имеют известный размер в байтах, например, int t; (2 байта), double x; (4 байта) и т. д., компилятор при создании объектов выделит соответствующие участки памяти, и в этом случае деструктор не нужен, так как он будет иметь пустое определение. Если же объект создается в динамической области памяти, например, с помощью операции захвата памяти new, то деструктор необходимо задавать явно, потому что он должен уничтожить объект после его использования с помощью операции delete. Рассмотрим кратко эти 2 операции.

Операции new и delete (C++)

Управление программным размещением объектов в памяти, т. е. выделение памяти в процессе работы программы под переменные и массивы в языке C++, осуществляется с помощью операций new и delete.

Операция new автоматически определяет размер выделяемой памяти для указанного типа, а также позволяет инициализировать вновь создаваемый объект.

Формат операций:

ID_указателя = new тип (значение); // Захват памяти (работа с динамическим объектом через косвенную адресацию – операция "*")

deleteID_указателя; //Освобождение памяти после работы с объектом, где ID_указателя – идентификатор переменной-указателя на заданный тип

тип – тип значений, для которых выделяется память;

значение – константа (необязательный атрибут), определяющая начальное значение динамической переменной.

Таким образом, операция new устанавливает указатель на участок свободной динамической оперативной памяти размером sizeof(num).

Если выделить память не удается, то возвращается нулевой указатель, что дает возможность проконтролировать процесс захвата участка динамической памяти. Объект существует до тех пор, пока память не будет освобождена при помощи операции delete (или до окончания работы программы). Следующий участок программы демонстрирует работу с целочисленной динамической переменной.

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{
int *p = new int(6); //Выделяем динамическую память под тип int и определяем значение 6
if (.p) //Проверка на случай невыделения памяти, в этом случае в указатель вернется null
{
        cout << "Eror";
}
```

```
cout << *p << " "; //Output: 6
*p = 45; //Даем новое значение в динамической памяти
cout << *p; //Output: 45
delete p; //Освобождаем память
}
```

Указатели, которые хранят адрес освобожденной памяти, называются висячими указателями.

В нашем примере после применения delete p; указатель p стал висячим.

Оператор delete на самом деле ничего не удаляет. Он просто возвращает память, которая была выделена ранее, обратно в операционную систему. Затем операционная система может переназначить эту память другому приложению (или этому же снова).

Хотя может показаться, что мы удаляем переменную, но это не так. Переменная-указатель по-прежнему имеет ту же область видимости, что и раньше, и ей можно присвоить новое значение, как и любой другой переменной.

Вывод или повторное удаление такого указателя может привести к неожиданным результатам.

Теперь можно видоизменить рассмотренный ранее пример, включив в него работу с динамической памятью и увеличив все поля класса в 3 раза при помощи метода Run:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Test
{
public:
int *a, *b;
double *c;
Test(); //Конструктор без параметров
Test(int r, int t); //Конструктор с параметрами
~Test();
void show()
{
	cout << " " << *a << " " << *b << " " << *c;
}
void run()
{
	*a *= 3;
	*b *= 3;
	*c *= 3;
}
};
```

```
Test::Test() //Пишем код конструктора без параметров
{
a = new int(2);
b = new int(42);
c = new double(90);
}
Test::Test(int r, int t) //Пишем код конструктора с данной сигнатурой
{
a = new int(r);
b = new int(t);
c = new double(0);
}
Test::~Test()
{
delete(a, b, c);
}
int main()
{
Test a; //Вызовет конструктор без параметров
Test b(12, 34); //Вызовет конструктор с параметрами
a.show(); //Output: 2 42 90
b.show(); //Output: 12 34 0
a.run(); //Увеличиваем все поля объекта a в 3 раза
b.run(); //Увеличиваем все поля объекта b в 3 раза
//Выводим новые значения
a.show(); //Output: 6 126 270
b.show(); //Output: 36 102 0
}
```

Задания к лабораторной работе

Уровень 1

Упражнение: продемонстрируйте свое умение работать с конструкторами и деструкторами, а также с динамической памятью. За основу возьмите класс, созданный для решения предыдущей лабораторной работы (уровень 1 вашего варианта). Все поля, описанные в нем, сделайте указателями. Добавьте в те классы конструктор с параметрами, который будет служить для инициализации полей, и конструктор без параметров, в котором значения полям будут даваться такие, какие вы сами захотите. Также создайте деструктор, который освободит всю память, занятую полями. Описание конструкторов и деструктора должно быть выполнено за пределами класса. Также добавьте функцию set(), которая будет давать новые значения полям класса. Функция set() тоже должна быть описана за пределами класса.

Уровень 2

Условие такое же, как и в упражнении уровня 1. За основу возьмите класс, созданный для решения предыдущей лабораторной работы (уровень 2 вашего варианта). И также добавьте все то, что вас просят добавить в упражнении.

Контрольные вопросы и задания

1) Что такое конструктор, какие его основные особенности?
2) Что такое деструктор, какие его основные особенности?
3) Когда наличие деструктора обязательно?
4) Объяните назначение операций new и delete.
5) Сколько существуют динамические объекты?
6) Что такое висячий указатель?
7) Сколько конструкторов и деструкторов может быть в классе?

Лабораторная работа №3
Работа с динамической памятью. Одномерный динамический массив

Цель работы: изучить способы работы с мышью в среде Visual Studio, а также изучить создание одномерных массивов при помощи конструктора с захватом динамической памяти и деструктора для их уничтожения.

Теоретические сведения

Как вы уже знаете, при стандартной декларации массивов компилятор воспользуется векторной организацией памяти и выделит фиксированный участок памяти, достаточный для хранения всех его элементов. Использование динамической памяти и списковой ее организации позволяет создавать массивы переменной длины. Помимо уже известных операций по захвату и освобождению динамической памяти new и delete для этих целей используют стандартные библиотечные функции, декларацию которых содержит заголовочный файл. Но все эти функции уже включены в библиотеку "pch.h".

Приведем сведения о наборе функций манипулирования памятью:

1) Sizeof(type) – функция, которая возвращает количество байтов, необходимых для хранения переменной типа, указанного в сигнатуре этой функции;

2) void *malloc (unsigned n) – функция выделения памяти для размещения блока размером n байтов. Возвращает указатель на распределенную область или NULL при неудаче.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{
//setlocale(LC_CTYPE, "rus"); //Если кто-то захочет использовать русский язык
double *x; //Создаем указатель на будущий массив
int n = 5; //Создаем переменную, которая будет указывать количество элементов массива
x = (double*)malloc(n*sizeof(double)); //Выделяем необходимое количество байтов для 5 элементов типа double при помощи malloc(n*sizeof(double))
for (int i=0;i<5;i++) //Заполняем наш массив
{
x[i] = i+0.2;
}
for (int i = 0; i < 5; i++) //Выводим наш массив
{
cout << endl<<x[i];
```

```
}
/*
Output:
0.2
1.2
2.2
3.2
4.2
*/
free(x); //Освобождаем память
}
```

3) void *calloc (unsigned n, unsigned size); – функция выделения памяти для размещения n объектов размером size байтов и заполнение полученной области нулями. Возвращает указатель на захваченную область памяти или NULL при неудаче.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{
//setlocale(LC_CTYPE, "rus"); //Если кто-то захочет использовать русский язык
double *x; //Создаем указатель на будущий массив
int n = 5; //Создаем переменную, которая будет указывать количество элементов массива
x = (double*)calloc(n, sizeof(double)); //Выделяем 5 ячеек размером double при помощи calloc(n, sizeof(double))
for (int i=0; i<5;i++) //Заполняем наш массив
{
x[i] = i+0.2;
}
for (int i = 0; i < 5; i++) //Выводим наш массив
{
cout << endl<<x[i];
}
/*
Output:
0.2
1.2
2.2
3.2
4.2
*/
```

```
free(x); //Освобождаем память
}
```

4) /*unsigned coreleft (void); – функция получения размера неиспользованной памяти в куче в байтах;

5) void free(void *b); – функция освобождения блока памяти, адресуемого указателем b.;

6) void *realloc (void *b, unsigne n); – функция изменения размера, размещенного по адресу b блока на новое значение n и копирование (при необходимости) содержимого блока. Возвращает указатель на перераспределенную область памяти или NULL при неудаче.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{
//setlocale(LC_CTYPE, "rus"); //Если кто-то захочет использовать русский язык
double *x; //Создаем указатель на будущий массив
int n = 5; //Создаем переменную, которая будет указывать количество элементов массива
x = (double*)malloc(n*sizeof(double)); //Выделяем необходимое количество байтов для 5 элементов типа double при помощи malloc(n*sizeof(double))
for (int i=0;i<5;i++) //Заполняем наш массив
{
x[i] = i+0.2;
}
x = (double*)realloc(x, (n+2) * sizeof(double)); //Заново выделяем блок памяти для x, добавляя ему байт для хранения еще 2 переменных, при этом содержимое старого блока копируется в новый блок и информация не теряется
for (int i = 5; i < 7; i++) //Добавляем новые значения элементам
{
x[i] = i + 0.2; }
for (int i = 0; i < 7; i++) //Выводим наш массив
{ //cout << endl << x[i];
}
/*
Output:
0.2
1.2
2.2
3.2
4.2
5.2
```

```
6.2
*/
free(x); //Освобождаем память
}
```

Общий формат операторов new и delete для работы с динамической памятью следующий:

```
type *name; //Декларировали указатель на начало массива
name = new type[size]; //Захватили память
delete [ ] name; //Освободили память
```

где type – тип элементов;

size – максимальное количество элементов массива name.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{
int *n; //Указатель на массив
int lenght = 10; //Размер будущего массива
n = new int[lenght]; //Выделяем память указателю n для хранения элементов типа int количеством lenght
delete[] n; //Освобождаем память, выделенную под массив
}
```

Рассмотрим пример создания динамического массива в классе:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Test
{
public:
//Указатели на массивы
int *IntMass;
double *DoubleMass;
int IntLenght, DoubleLenght; //Переменные для хранения длины массивов
Test(int r, int t); //Конструктор с параметрами
~Test();
void Show();
};
Test::Test(int r, int t) //Пишем код конструктора с данной сигнатурой
{
IntMass = new int[r]; //Выделяем память при помощи new
```

```
IntLenght = r; //Сохраняем размер массива int
for (int i=0;i<r;i++) //Заполняем наш массив int
{
cout <<endl<< "Enter Int element[" << i << "]: ";
cin >> IntMass[i];
}
DoubleMass = (double*)calloc(t, sizeof(double)); //Выделяем память при помощи
calloc
DoubleLenght = t; //Сохраняем размер массива double
for (int i = 0; i < t; i++) //Заполняем наш массив double
{
cout << endl << "Enter Double element[" << i << "]: ";
cin >> DoubleMass[i];
} }
Test::~Test()
{
//Освобождаем память
delete[] IntMass;
free(DoubleMass);
}
void Test::Show()
{
cout << endl << "IntMass:\n";
//Вывод массива INT
for (int i=0;i<IntLenght;i++)
{
cout << endl << IntMass[i];
}
cout << endl << "DoubleMass:\n";
//Вывод массива double
for (int i = 0; i < DoubleLenght; i++)
{
cout << endl << DoubleMass[i];
} }
int main()
{
Test mass(5,4); //Передаем размеры массивов в качестве параметров
mass.Show(); //Выводим массивы
}
```

Задания к лабораторной работе

Уровень 1

Упражнение: создайте пользовательский класс, который будет содержать конструктор с параметром для создания динамических массивов (оператор new или стандартная библиотечная функция calloc) и установки начальных значений элементов (размер массива передается как аргумент для конструктора). Также класс должен иметь метод для просмотра текущего состояния массива и деструктор, который освободит память.

Варианты:

1) На вашу почту пришел заказ от компании "Cool developer". Вас просят разработать инвентарь персонажа, где будут храниться предметы. Так как игра в стадии предварительной разработки, то названий у предметов нет, есть только ID предметов.

Ваша задача: создать в классе Inventory массив размером Size (Size – количество ячеек в инвентаре) и заполнить его случайными значениями от 0 до 100 000. Затем реализовать метод Run(), который выведет количество элементов с нечетным ID.

2) После того как вы показали своему начальству разработанную вами программу для наблюдения за mildew, он решил, что программе платить не надо, а работу она делает не хуже вас, поэтому вас уволили. Вы решили попробовать себя в астрофизике. Первым заданием стала запись расстояний от Солнца до звезд, попадающих в область видимости телескопа "Hubble".

Ваша задача: создать класс Stars, а в нем – массив размером Size (Size – количество звезд, которое наблюдает "Hubble") и заполнить его случайными значениями от 0 до 999 999. Затем реализовать метод Run(), который выведет разность между самой дальней и самой близкой звездой (разумеется, в космосе для оценки межзвездных расстояний используются парсеки).

3) Завтра в вашу больницу приезжает проверка с целью проконтролировать заполнение палат больницы. Чтобы не водить проверяющих в каждую палату, вы решили написать программу, в которой будет храниться информация о количестве человек в каждой палате.

Ваша задача: создать класс Patients, в котором будет массив размером Size (Size – количество палат в больнице) и заполнить его случайными значениями от 0 до 10. Затем реализовать метод Run (), который выведет номера палат, где количество пациентов >5 (номера палат начинаются с нуля).

4) Ваш начальник хочет знать, сколько каждый клиент, вошедший в его банк, потратил денег.

Ваша задача: создать класс Customers, а в нем – массив размером Size (Size – количество клиентов, посетивших банк за день) и заполнить его случайными значениями от 0 до 100 000. Затем реализовать метод Run(), который выведет общее количество денег, полученное банком от всех клиентов, посетивших его за день, а

также номера клиентов, которые потратили более 90 000 (начальник лично хочет выдать им карточкиVIP-клиентов в благодарность за свою новую Tesla Model X).

Уровень 2

Перед выполнением уровня 2 посмотрите, как можно создавать массив объектов. Для этого запустите приведенный ниже пример и разберитесь в том, как он работает и как это все выглядит в памяти.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
class MassElement //Класс, объекты которого будут являться элементами массива
{
public:
double a, b; //Поля для хранения значений типа double
};
class Test //Класс, объект которого будет хранить в себе массив объектов класса MassElement, его размер и методы для работы с этим массивом
{
public:
int Size; //Поля для хранения длины массива
MassElement *mass; //Поле для хранения нашего будущего массива объектов
Test(int lenght); //Конструктор с параметрами
~Test(); //Деструктор
void Show(); //Метод для просмотра элементов массива
};
Test::Test(int lenght) //Пишем код конструктора с данной сигнатурой
{ mass = (MassElement*)calloc(lenght, sizeof(MassElement)); //Выделение памяти для массива объектов
this->Size = lenght; //Сохраняем размер нашего массива
for (int i = 0; i < lenght; i++) //Заполняем поля каждого элемента массива, которые в свою очередь являются объектами класса Test
{ cout << endl << "Enter 1 element[" << i << "]: "; cin >> mass[i].a; //Заполняем поле a объекта под номером i
cout << endl << "Enter 2 element[" << i << "]: "; cin >> mass[i].b; //Заполняем поле b объекта под номером i
} }
Test::~Test()
{
//Освобождаем память
free(mass); }
void Test::Show()
{ for (int i = 0; i < this->Size; i++) //Выводим поля каждого объекта массива
```

```
{ cout << endl << "1 element[" << i << "]: =" << mass[i].a;
cout << endl << "2 element[" << i << "]: =" << mass[i].b; } }
int main()
{ Test mass(5); //Создаем наш объект, который хранит в себе массив объектов
mass.Show(); //Просматриваем наш массив объектов
}
```

А зачем мы заводили второй класс MassElement? Нельзя ли все сделать в одном классе?

Ответ: можно.

```
#include "pch.h"
#include <iostream>
using namespace std;
class Test
{ public:
double a, b; //Поля для хранения значений типа double
int Size; //Поля для хранения размера массива
Test *mass; //Поле для хранения нашего будущего массива объектов
Test(int lenght); //Конструктор с параметрами
~Test(); //Деструктор
void Show(); //Метод для демонстрации всех элементов массива
};
Test::Test(int lenght) //Пишем код конструктора с данной сигнатурой
{       mass = (Test*)calloc(lenght, sizeof(Test)); //Выделение памяти для массива объектов
this->Size = lenght; //Созраняем размер нашего массива
for (int i = 0; i < lenght; i++) //Заполняем поля каждого элемента массива, которые в свою очередь являются объектами класса Test
{       cout << endl << "Enter 1 element[" << i << "]: "; cin >> mass[i].a; //Заполняем поле a объекта под номером i
        cout << endl << "Enter 2 element[" << i << "]: "; cin >> mass[i].b; //Заполняем поле b объекта под номером i
}}
Test::~Test()
{
//Освобождаем память
free(mass); }
void Test::Show()
{ for (int i = 0; i < this->Size; i++) //Выводим поля каждого объекта массива
{ cout << endl << "1 element[" << i << "]: =" << mass[i].a;
cout << endl << "2 element[" << i << "]: =" << mass[i].b;
} }
int main()
{ Test mass(3); //Создаем наш объект, который хранит в себе массив объектов
```

```
mass.Show(); //Просматриваем наш массив объектов
}
```

И данная реализация тоже сработает. Единственное, что можно считать недостатком, – это выделение лишней памяти. Во втором случае мы выделяем память элементу, необходимую для хранения объекта класса Test, а в первом случае – для хранения объекта класса MassElement. Наш элемент должен хранить информацию только о переменных a и b, поле для хранения Size и указателя на массив, которые прописаны в Test, ему не нужны – это лишний расход памяти. Поэтому первый пример будет экономнее в плане памяти. Но каким способом будет решена задача – определять вам.

Упражнение: создайте пользовательский класс, который будет содержать конструктор с параметром для создания динамических массивов (операция new или стандартная библиотечная функция calloc) и установки начальных значений элементов (размер массива передается как аргумент для конструктора). Также класс должен иметь метод для просмотра текущего состояния массива и деструктор, который освободит память. Для демонстрации своего умения работы с массивом объектов и хранения информации используйте массив объектов, в каждом элементе которого будут храниться поля, необходимые для выполнения вашего задания. Выполнение задания уровня 2 должно происходить в классах, созданных для выполнения уровня 1.

Варианты:

1) Как только заказчик увидел, что все ячейки заполнены непонятными ему числами, он дал команду придумать каждому предмету название и указать массу каждого предмета. Вам необходимо внести изменения в созданный класс с учетом новых данных.

Ваша задача: в существующем классе создать массив объектов этого класса и добавить поля ID, weight и name (значение weight генерируется случайно в диапазоне от 1 до 100). Метод Run() должен вывести всю информацию о каждом объекте, лежащем в инвентаре, у которого 30<weight<90. Поля ID и name вводятся вручную. Размер массива передается в конструктор в качестве параметра (как и для задания уровня 1), заполнение массива и информации о каждом объекте происходит в том же конструкторе.

2) Начальнику недостаточно информации о расстоянии до других звезд, необходимо добавить еще больше данных.

Ваша задача: в существующем классе создать массив объектов этого класса и добавить поля distance, name и size (distance и size генерируются случайно в диапазоне от 1 до 100 000). Метод Run() должен вывести всю информацию о каждом объекте, лежащем в диапазоне 400< distance <90 000. Поле name вводится вручную. Размер массива передается в конструктор в качестве параметра (как и для задания уровня 1) и заполнение массива и информации о каждом объекте происходит в том же конструкторе.

3) Проверке понравилась ваша идея с программой, в которой все хранится, но они предпочли бы получить информацию о каждом пациенте, а не о палате.

Ваша задача: в существующем классе создать массив объектов этого класса и добавить поля HowLongWillBeInHospital, Name и WardNumber (HowLongWillBeInHospital генерируется случайно в диапазоне от 1 до 90). Метод Run() должен вывести всю информацию о каждом объекте, лежащем в диапазоне 20< HowLongWillBeInHospital <70. Поля Name и WardNumber вводятся вручную. Размер массива передается в конструктор в качестве параметра (как и для задания уровня 1) и заполнение массива и информации о каждом объекте происходит в том же конструкторе.

4) Вашему начальнику необходимо знать еще возраст и пол клиентов банка.

Ваша задача: в существующем классе создать массив объектов этого класса и добавить поля spending, Age и gender (это поле для удобства можно сделать типа bool, мужчины == 0, девушки == 1). Age генерируется случайно в диапазоне от 16 до 90, spending – в диапазоне от 0 до 100 000. Метод Run() должен вывести всю информацию о каждом объекте, у которого 18<Age<30 и 25000<spending<100000 и gender==1. Поле gender вводится вручную. Размер массива передается в конструктор в качестве параметра (как и для задания уровня 1) и заполнение массива и информации о каждом объекте происходит в том же конструкторе.

Контрольные вопросы и задания

1) Объясните формат и назначение библиотечной функции malloc().

2) Объясните формат и назначение библиотечной функции calloc().

3) Объясните формат и назначение библиотечной функции free().

4) Как создать и уничтожить динамический одномерный массив при помощи операторов?

5) Как создать массив объектов класса?

6) Сколько памяти (в байтах) займет данный массив Int *mass = new int[5] ?

Лабораторная работа №4
Работа с динамической памятью.
Использование дружественных функций

Цель работы: изучить методику создания и уничтожения двумерного динамического массива при помощи конструкторов с захватом динамической памяти и деструкторов для ее освобождения, научиться работать с классом посредством функций-друзей этого класса.

Теоретические сведения

Учитывая тот факт, что имя двумерного массива – это указатель на указатель int **m; двухмерный динамический массив создается и уничтожается за 2 шага. Например, требуется создать целочисленный массив m[3][4]:

Пример 1:

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{
//Создание двумерного массива
int **mass = new int*[3]; //Создание массива указателей
for (int i=0;i<3;i++)
{ mass[i] = new int[4]; //Создание еще одного массива каждому указателю
}
//Заполнение массива
for (int i=0;i<3;i++)
{ for (int j=0;j<4;j++)
{ mass[i][j] = i+j;
//Или можно использовать запись *(*(mass + i) + j) = i+j;
} }
//Вывод массива
for (int i = 0; i < 3; i++)
{ cout << endl;
for (int j = 0; j < 4; j++)
{ cout << " " << mass[i][j];
} }
/*
Outputs:
0 1 2 3
1 2 3 4
2 3 4 5
```

```
*/
//Удаление двумерного массива
for (int i=0;i<3;i++)
{ delete[] mass[i]; //Освобождение памяти, занятой элементами
}
delete[] mass; //Освобождение памяти, занятой под массив указателей
}
```

Пример 2 (использование функций для выделения памяти):

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{
//Создание двумерного массива
int **mass = (int**)calloc(3, sizeof(int*)); //Создание массива указателей
for (int i=0;i<3;i++)
{ mass[i] = (int*)calloc(4, sizeof(int)); //Создание еще одного массива для каждого указателя
}
//Заполнение массива
for (int i=0;i<3;i++)
{ for (int j=0;j<4;j++)
{ mass[i][j] = i+j;
//Или можно использовать запись *(*(mass + i) + j) = i+j;
} }
//Вывод массива
for (int i = 0; i < 3; i++)
{ cout << endl;
for (int j = 0; j < 4; j++)
{ cout << " " << mass[i][j];
} }
/*
Outputs:
0 1 2 3
1 2 3 4
2 3 4 5
*/
//Удаление двумерного массива
for (int i=0;i<3;i++)
{
free(mass[i]); //Освобождение памяти, занятой элементами
}
free(mass); //Освобождение памяти, занятой под массив указателей
}
```

Краткая характеристика функции-друга класса

Механизм функции-друга класса обеспечивает возможность доступа к любым элементам класса тем функциям, которые не входят в состав этого класса. То есть функция-друг класса – это обычная функция пользователя, которая благодаря данному механизму имеет полные права доступа ко всем без исключения элементам класса, независимо от их степени защищенности. Чтобы обычная функция стала другом класса, в заголовке ее необходимо указать friend.

Некоторые особенности функции-друга:

1. Функцию-друга класса необходимо декларировать в описании этого класса.

В форме описания (приводится прототип), а ее полное определение – вне шаблона класса, при этом не нужно указывать операцию привязки к данному классу, так как она не является членом данного класса.

2. При обращении к функции-другу не надо использовать операцию привязки (.) или (→).

3. Так как функция-друг – обычная функция, у нее отсутствует первый скрытый параметр this. Для обращения к элементам класса из функции-друга используют или ссылку, или указатель, что и иллюстрирует следующий пример.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Test
{
private:
int b; //Защищенное поле
public:
int a; //Публичное поле
void Show(); //Метод класса для просмотра
void Set(int a,int b); //Метод класса для установки значений полей
friend void Show(Test &); //Дружественная функция для просмотра полей
friend void Set(Test &,int,int); //Дружественная функция для установки значений полей
};
void Test::Show() //Реализация метода класса для просмотра полей
{
cout<<endl << a<<" " << b; }
void Test::Set(int a, int b) //Реализация метода класса для установки новых значений
{
//Так как метод является частью класса, то ему доступен специальный указатель this
this->a = a;
this->b = b; }
void Show(Test &obj) //Реализация дружественной функции для просмотра полей
{
```

```
//Если бы эта функция не была бы указана в классе Test (Строка 15) как дружественная, то поле b вывести не удалось бы из-за нехватки доступа
cout<<endl << obj.a<<" " << obj.b; }
void Set(Test &obj, int a, int b) //Реализация дружественной функции для заполнения полей
{
//Если бы эта функция не была бы указана в классе Test (Строка 16) как дружественная, то поле b изменить не удалось бы из-за нехватки доступа
obj.a = a;
obj.b = b;
}
int main()
{
Test a;
a.Set(12, 23); //Задаем значения при помощи функции класса
a.Show(); //Выводим значения полей при помощи метода класса
/*
Output:
12 23
*/
Set(a,44,55); //Задаем значения полей при помощи дружественной функции
Show(a); //Выводим значения полей при помощи дружественной функции
/*
Output:
44 55
*/
}
```

Отметим, что функции-друзья класса кроме ссылки или указателя на объекты класса могут иметь обычные параметры, например, для инициализации данных объектов класса. Яркой демонстрацией такой функции является дружественная функция Set(Test &obj, int a, int b), которая принимала в качестве аргументов еще 2 переменные типа int для установки новых значений полям.

Итак, мы видим, что результат выполнения методов класса и дружественных функций у нас одинаковый и отсюда возникает вполне логичный вопрос: зачем вообще нужны дружественные функции? В таком простом примере с всего лишь одним классом действительно эти функции нам не слишком нужны, а вот представьте проект, где таких классов десятки. И вот чтобы не писать каждому классу свой метод для выполнения операции вывода полей, к примеру, проще у всех у них указать одну общую дружественную функцию, а в ней уже вывести все поля переданных в нее объектов. Это существенно сократит код и в нем будет легче ориентироваться.

Задания к лабораторной работе

Уровень 1

Упражнение: реализуйте дружественные функции для просмотра текущего состояния массива, установки новых значений элементов массива; выполнения поставленной задачи в соответствии с вариантом.

Пользовательский класс должен содержать конструктор с параметрами для создания динамических целочисленных массивов (операция new или стандартная библиотечная функция calloc). Размеры массива (число строк и столбцов) передается в конструктор через параметры, заполнение массива происходит в конструкторе. Класс также должен содержать деструктор для освобождения занятой памяти.

Варианты:

1) Вы очень любите игру "Heroes of Might and Magic" и поэтому решили написать свою версию этой игры. Разумеется, написать игру полностью вы не успеете в связи с наличием других заказов, только прописать поле битвы, которое будет выглядеть как квадратная матрица размером N×N (N>1). В каждую ячейку вы вводите количество солдат, стоящих на этой ячейке. Солдаты, расположенные в ячейках выше главной диагонали, – ваши, солдаты, расположенные в ячейках ниже главной диагонали, принадлежат противнику, солдаты, расположенные на главной диагонали, никому не принадлежат и не учитываются при сражении.

Ваша задача: создать класс Arena, в котором будет двумерный массив N×N, и заполнить его случайными значениями в диапазоне от 0 до 1000, после написать дружественную функцию Run, которая выведет количество ваших солдат и количество солдат противника.

2) Вас попросили помыть линзы телескопа. Когда вы уже все сделали, то решили проверить в Интернете, как правильно мыть линзы телескопа. Так как вы все сделали не так, то решили перевестись на другую кафедру. Лучшим вариантом стала кафедра растениеводства, в связи с тем что можно записаться добровольцем на недельную работу агрономом в поле. Приехав на место, вы увидели поле, разделенное на участки и очень похожее на матрицу размером N×M. На каждом участке растет разное количество растений, которое вам необходимо посчитать.

Ваша задача: создать класс Garden, в котором будет двумерный массив N×M, и заполнить его случайными значениями в диапазоне от 0 до 100, затем написать дружественную функцию Run, которая выведет количество всех растений на поле.

3) Работа главврача очень сложна и, для того чтобы отдохнуть, вы иногда выглядываете в окно с целью полюбоваться на клумбу с цветами. Вы уже давно заметили, что клумба разделена на участки и очень похожа на матрицу размером N×M. Сегодня вы решили посчитать количество цветов по краю клумбы.

Ваша задача: создать класс LowerBed, в котором будет двумерный массив размером N×M, и заполнить его случайными значениями от 0 до 10, затем написать дружественную функцию Run(), которая посчитает количество цветов по краям клумбы.

4) В вашем банке есть хранилище. Начальник просит посчитать, сколько в среднем хранится денег в ячейках и сколько ячеек, в которых количество денег больше среднего значения. Хранилище давно напоминало вам матрицу размером M×N, а с матрицей вы работать умеете.

Ваша задача: создать класс Storage, в котором будет двумерный массив размером N×M, и заполнить его случайными значениями от 0 до 100 000, затем написать дружественную функцию Run(), которая посчитает среднее количество денег во всех ячейках и количество ячеек, в которых хранится количество денег, большее среднего значения.

Уровень 2

Перед выполнением уровня 2 посмотрите, как можно создавать двумерный массив объектов.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
class MassElement //Класс, объекты которого будут элементами нашего двумерного массива
{
public:
double a, b; //Поля для хранения значений типа double
};
class Test
{
public:
int N, M; //Переменные для хранения размера массива
MassElement **mass; //Поле для хранения нашего будущего массива объектов

Test(int N, int M); //Конструктор с параметрами
~Test(); //Деструктор
void Show(); //Метод для демонстрации всех элементов массива
};
Test::Test(int N, int M) //Пишем код для конструктора с данной сигнатурой
{
mass = (MassElement**)calloc(N, sizeof(MassElement*)); //Создаем массив, каждый элемент которого будет еще одним массивом
for (int i = 0; i < N; i++)
{
mass[i] = (MassElement*)calloc(M, sizeof(MassElement)); //Создание еще одного массива для каждого указателя
}
```

```
//Сохраняем размер нашего массива
this->N = N;
this->M = M;
for (int i = 0; i < N; i++) //Заполняем поля каждого элемента массива, которые в свою очередь являются объектами класса MassElement
{
for (int j = 0; j < M; j++)
{
cout << endl << "Enter 1 element[" << i << "][" << j << "]: "; cin >> mass[i][j].a; //Заполняем поле a объекта под номером i
cout << endl << "Enter 2 element[" << i << "][" << j << "]: "; cin >> mass[i][j].b; //Заполняем поле b объекта под номером i
}
}
}
Test::~Test()
{
//Освобождаем память
for (int i = 0; i < N; i++)
{
free(mass[i]); //Освобождение памяти, занятой элементами
}
free(mass); //Освобождение памяти, занятой под массив указателей
}
void Test::Show()
{
for (int i = 0; i < N; i++) //Выводим поля каждого объекта массива
{
for (int j = 0; j < M; j++)
{
cout << endl << "1 element[" << i << "][" << j << "]: " << mass[i][j].a;
cout << endl << "2 element[" << i << "][" << j << "]: " << mass[i][j].b;
}
}
}
int main()
{ Test mass(3, 2); //Создаем наш объект, который хранит в себе массив объектов
mass.Show(); //Просматриваем наш массив объектов
}
```

Условие такое же, как и в упражнении уровня 1. Выполнение задания уровня 2 должно происходить в классах, созданных для выполнения уровня 1. Для демон-

страции своего умения работы с двумерным массивом объектов для хранения информации создайте двумерный массив объектов, в каждом элементе которого будут храниться поля, необходимые для выполнения вашего задания.

Варианты:

1) Поиграв несколько часов в "Heroes of Might and Magic", вы вспомнили, что войска могут отличаться по уровню, поэтому необходимо это тоже учесть.

Ваша задача: в существующем классе создать двумерный массив объектов класса, в котором будут поля Level1, Level2, Level3, Level4, Level5 (тип int). Все поля заполняются случайно в диапазоне от 0 до 100. Дружественная функция Run() должна вывести количество ваших и вражеских войск: Level1, Level2, Level3, Level4, Level5.

2) Участки получились довольно большими и на каждом из них было несколько видов овощей. Необходимо узнать, сколько овощей каждого вида растет на поле.

Ваша задача: в существующем классе создать двумерный массив объектов класса, в котором будут поля Potatoes, Carrots, Cabbage, Horseradish, Onions (тип int). Все поля заполняются случайно в диапазоне от 0 до 100. Дружественная функция Run() должна вывести количество участков, на котором количество Potatoes больше, чем Carrots, в 2 раза и Horseradish>Onions.

3) После пристального просмотра вы поняли, что на клумбе растут все-таки разные виды цветов, и решили посчитать количество цветов каждого вида.

Ваша задача: в существующем классе создать двумерный массив объектов класса, в котором будут поля Tulips, Roses, Chrysanthemums, Snowdrops, Cornflowers (тип int). Все поля заполняются случайно в диапазоне от 0 до 20. Дружественная функция Run() должна вывести количество участков, на которых Tulips, Roses или Snowdrops == 0.

4) Оказалось, что в ячейках лежат не только деньги. Некоторые клиенты хранят в них ценности и ценные бумаги тоже. Вашего начальника не интересуют ценные бумаги ваших клиентов, его интересует их цена.

Ваша задача: в существующем классе создать двумерный массив объектов класса, в котором будут поля Money, Securities, Decorations, Stocks (тип int). Все поля заполняются случайно в диапазоне от 0 до 100 000. Дружественная функция Run() должна вывести среднее количество денег в ячейках (для определения цены одной ячейки == Money+Securities+Decorations+ Stocks) и количество ячеек, у которых цена больше среднего значения.

Контрольные вопросы и задания

1) Чем является имя двумерного массива?

2) Как косвенно обратиться к элементу двумерного массива?

3) Объясните понятие функции-друга класса?

4) Какие существуют способы обращения к элементам класса из функции-друга?

5) Как освободить память, занимаемую двумерным динамическим массивом?

6) Функция-друг имеет доступ ко всем полям класса?

7) Сколько памяти займет данный двумерный массив?

```
int **mass = new int*[5];
for(int i=0;i<5;i++)
{
mass[i] = new int[5];
}
```

8) Дан двумерный массив:

1 2 3 4 5
3 4 5 6 7
9 0 3 2 1

Напишите алгоритм для нахождения суммы элементов главной диагонали.

Лабораторная работа №5
Работа с динамической строкой и перегрузка операций

Цель работы: изучить методику по созданию одномерных динамических символьных массивов при помощи конструкторов с захватом динамической памяти и деструкторов для их уничтожения, а также способы работы со строковыми объектами, познакомиться с механизмом перегрузки операций.

Теоретические сведения

Напомним, что работа со строками в языке Си реализована путем использования одномерных массивов типа char, т. е. строка символов – это одномерный массив типа char, заканчивающийся нулевым байтом. Нулевой байт – это байт, каждый бит которого равен нулю, при этом для нулевого байта определена символьная константа '\0' (признак окончания строки или нуль-терминатор). Поэтому, если строка должна состоять из k символов, то в описании массива необходимо указать размер k+1, а при ручном формировании строки в ее окончание нужно явно добавить признак ее окончания.

Операции над строками выполняются только через стандартные функции. Декларации функций для работы со строками размещены в файле string.h (если вы используете Visual Studios, то string.h подключается автоматически при подключении "pch.h"). Перечислим некоторые из них, наиболее часто используемые:

1. Функция strcpy(S1, S2) – копирует содержимое строки S2 в строку S1.
2. Функция strcat(S1, S2) – присоединяет строку S2 к строке S1 и помещает ее в массив, где находилась строка S1, при этом строка S2 не изменяется. Нулевой байт, который завершал строку S1, заменяется первым символом строки S2.

Внимание! Функции strcpy и strcat на данный момент устарели и не используются, так как они были признаны небезопасными из-за неумения видеть границы, что приводило к переполнению буфера. В связи с этим были добавлены модифицированные аналоги данных функций [strcpy_s(), strcat_s()]. Данные функции могут сами определить необходимый размер буфера, а также (в случае необходимости) сам пользователь может передать в качестве аргумента размер буфера.

Рассмотрим примеры с применением данных функций.

1. Функция strcpy_s():

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{ setlocale(LC_CTYPE, "rus"); //Подключаем поддержку русских символов
char S1[] = "ООП – легко!"; //Создаем наш массив char. Так как размер в скобках мы
```

не указали, массив получится равным размеру текста "ООП – легко!" + 1 для нуль-терминатора '\0'

char S2[40]; //Создаем пустой массив на 40 символов

char S3[80]; //Создаем пустой массив на 80 символов

strcpy_s(S2,"Я – программист."); //Помещаем в массив S2 строку "Я – программист.".strcpy_s() – автоматически добавит нуль-терминатора '\0'. При попытке поместить в S2 строку большего размера, чем она может вместить, будет выдана ошибка

strcpy_s(S3,S2); //Помещаем в массив S3 строку, хранящуюся в S2

cout<<"S1= " << S1 << endl; //Output: "ООП – легко!"

cout << "S2= " << S2 << endl; //Output: "Я – программист."

cout << "S3= " << S3 << endl; //Output: "Я – программист."

}

2. Функция strcat_s():

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{ setlocale(LC_CTYPE, "rus"); //Подключаем поддержку русских символов
```

char S1[] = "ООП – легко!"; //Создаем наш массив char. Так как размер в скобках мы не указали, массив получится равным размеру текста "ООП – легко!" + 1 для нуль-терминатора '\0'

char S2[40]=""; //Создаем массив на 40 символов, в котором в данном случае будет храниться только нуль-терминатор '\0' .

char S3[80]; //Создаем массив на 80 символов

strcat_s(S2,S1); //Объединяем содержимое строки S2 с содержимым строки S1. Результат будет храниться в S2

strcat_s(S2," Я – программист."); //Объединяем содержимое строки S2 со строкой "Я – программист." Результат будет храниться в S2

//strcat_s(S3,S2); //Данная запись выдаст ошибку, так как массив S3 не содержит ничего, даже нуль-терминатора '\0'. Из-за этого функция не может определиться с позицией в S3, с которой необходимо начать запись содержимого S2

cout<<"S1= " << S1 << endl; //Output: "ООП – легко!"

cout << "S2= " << S2 << endl; //Output: "ООП – легко! Я – программист."

}

3) Функция strcmp(S1, S2) сравнивает строку S1 со строкой S2 и возвращает значение 0, если строки равны, т. е. содержит одно и то же число одинаковых символов, значение 1, если S1>S2, и значение –1 если S1<S2.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{ setlocale(LC_CTYPE, "rus"); //Подключаем поддержку русских символов
```

char S1[] = "ООП – легко!"; //Создаем наш масив char. Так как размер в скобках мы не указали, массив получится равным размеру текста "ООП – легко!" + 1 для нуль-терминатора '\0'

char S2[40]= "ООП – легко!"; //Создаем массив на 40 символов, в котором в данном случае будет храниться строка "ООП – легко!"

char S3[80] = "Я программист"; //Создаем массив на 80 символов, в котором в данном случае будет храниться строка "Я программист."

cout<<"S1==S2 выдаст: " << strcmp(S1,S2) << endl; //Output: 0

cout << "S3>S2 выдаст: " << strcmp(S3, S2) << endl; //Output: 1

cout << "S2<S3 выдаст: " << strcmp(S2, S3) << endl; //Output: -1

}

4) Функция strlen(S) возвращает длину строки, т. е. количество символов, начиная с нулевого и до нуль-терминатора (нулевой байт не учитывается).

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{ setlocale(LC_CTYPE, "rus"); //Подключаем поддержку русских символов
```

char S1[] = "ООП – легко!"; //Создаем наш массив char. Так как размер в скобках мы не указали, массив получится равным размеру текста "ООП – легко!" + 1 для нуль-терминатора '\0'

char S2[40]= "ООП – легко!"; //Создаем массив на 40 символов, в котором в данном случае будет храниться строка "ООП – легко!".

char S3[80] = "Я программист"; //Создаем массив на 80 символов, в котором в данном случае будет храниться строка "Я программист."

```
cout<<"Lenght S1 = " << strlen(S1) << endl; //Output: 10
cout << "Lenght S2 = " << strlen(S2) << endl; //Output: 10
cout << "Lenght S3 = " << strlen(S3) << endl; //Output: 15
}
```

5) Функции преобразования строки S в число:

целое: int atoi(S);

длинное целое: long atol(S);

действительное: double atof(S);

при ошибке возвращает значение 0.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{ setlocale(LC_CTYPE, "rus"); //Подключаем поддержку русских символов
```

char S1[] = "231"; //Создаем наш массив char. Так как размер в скобках мы не указали, массив получится равным размеру текста "231" + 1 для нуль-терминатора '\0'

char S2[40]="42.23"; //Создаем массив на 40 символов, в котором в данном случае будет храниться строка "42.23" .

```
char S3[80] = "45"; //Создаем массив на 80 символов, в котором в данном случае будет храниться строка "45".
int a = atoi(S1);
int b = atoi(S3);
double c = atof(S2);
if (a>b) //Output: S3>S1
{
cout << "S3>S1"<<endl;
}
if (c > a) // Output: S2>S3
{
cout << "S3>S2"<<endl;
}
else
{
cout << "S2>S3"<<endl;
}
if (atof(S2)<atof(S1)) //Output: S1>S2
{
cout << "S1>S2"<<endl;
} }
```

6) Функции преобразования числа V в строку S:

целое: itoa(int V, char S, int kod);

длинное целое: ltoa(long V, char S, int kod);

kod – это система счисления, в которой мы хотим получить результат;

2<=kod<=36, для отрицательных чисел kod=10.

Внимание! Функция itoa в данный момент также устарела и мало где поддерживается. На смену ей пришел ее более безопасный аналог _itoa_s.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{ setlocale(LC_CTYPE, "rus"); //Подключаем поддержку русских символов
int value; //Наше число
char string[6] = ""; //Массив char, в котором будет храниться наше число
cout << "Введите число: ";
cin >> value;
itoa_s(value, string, 10); //Переводим value в десятеричную систему и помещаем в string
cout << "Введенное число в decimal: " << string << endl;
itoa_s(value, string, 16); //Переводим value в шестнадцатеричную систему и помещаем в string
```

```
cout << "Введенное число в hexadecimal: " << string << endl;
itoa_s(value, string, 2); //Переводим value в двоичную систему и помещаем в string
cout << "Введенное число в binary: " << string << endl;
system("pause"); //Пауза
return 0;
}
```

Помимо традиционной декларации строки, например char stroka[40]; можно создавать динамические строки.

Пример создания с использованием библиотечной функции:

```
#include "pch.h"
#include <iostream>
using namespace std;
int main()
{
char *stroka;
int Size = 10;
```

stroka = (char*)calloc(Size, sizeof(char)); //Выделяем память для хранения 10 элементов указателю stroka. Не забываем, что в реальности последний элемент должен быть '\0', поэтому stroka в себе сможет хранить 9 символов, всегда учитывайте '\0' в конце, он всегда должен занимать последний элемент

```
//Для заполнения stroka теперь необходимо в функцию strcpy_s добавить еще один аргумент Size(размер нашей переменной stroka)
strcpy_s(stroka,Size,"123456789"); //Выполнится
cout << stroka; //Output: "123456789"
```

//strcpy_s(stroka, Size, "1234567890"); //Ошибка. Нехватка памяти для размещения строки "1234567890" в stroka (из-за того, что в конце должен стоять '\0' и символу '0' не хватает места. Не забываем про '\0'. Это распространенная ошибка

```
}
```

Можно воспользоваться и операторами new и delete для захвата и освобождения динамической памяти. Пример класса с конструктором для создания динамических строковых объектов и деструктором для их уничтожения после использования в программе:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Stroka
{
public:
char * Text; //Создаем наш указатель на строку
```

Stroka(const char *s) //Создаем конструктор, принимающий строку, которую нам необходимо сохранить

```
{
```

```
Text = new char[strlen(s) + 1]; //Мы предполагаем, что полученная строка s не будет иметь '\0' в конце, поэтому мы выделяем память для хранения строки s и добавляем +1 для записи '\0' в конец
strcpy_s(Text, strlen(s) + 1, s); //Помещаем значение s в Text
}
~Stroka() //Деструктор
{
delete Text; //Очищаем память
}
void Print() //Метод для вывода содержимого поля Text
{
cout << Text << endl; //Вывод
}
};
int main()
{
Stroka t1("12345"), t2("6789"); //Создаем 2 объекта и помещаем в них строки
t1.Print(); //Output:"12345"
t2.Print(); //Output:"6789"
}
```

Существует специальный шаблонный класс String, для которого описанные операции не требуются. Их можно реализовать при помощи операторов.

Пример:

```
#include "pch.h"
#include <iostream>
#include <string> //Подключаем наш класс String и функции для работы с классом string (к примеру, to_string и т. д.)
using namespace std;
int main()
{
String text1, text2="String – the Best.", text3 = "I'm a programmer";
//Данная запись заменяет нам strcpy. Оператор "=" в данном случае делает то же самое, что и функция strcpy(). Помещает значение text2 в text1
text1 = text2;
cout << text1<<endl; //Output: "String - the Best."
//Данная запись заменяет нам сразу strcpy и strcat. Оператор "+" в данном случае делает то же самое, что и функция strcat. Объединяет значения text2 и text3, а потом оператор "=" помещает это объединенное значение в text1
text1 = text2 + text3;
cout << text1<<endl; //Output: String – the Best.I'm a programmer
//В классе String есть функция length(), которая вернет вам длину строки, хранящейся в нашем объекте. Как вы уже догадались, это замена функции strlen();
```

```
cout << text1.length()<<endl; //Output: 34
//А как просто можно сравнивать объекты String. Функция strcmp() нам больше не нужна.
//При верности условия возвращается 1, иначе 0
if (text1>text2)
{ cout << "text1>text2: "<< (text1 > text2)<<endl; //Output:1
}
if (text1==text1)
{ cout << "text1==text1:" << (text1 == text1)<<endl; //Output:1
}
if (text2 > text1)
{
}
else
{ cout << "text2 > text1: " << (text2 > text1)<<endl; //Output:0
}
//А вот для перевода string в число будет нужна функция atoi() и ей подобные
int a;
text1 = "123";
a = atoi(text1.c_str()); //Метод c_str() – возвращает массив char из text1, с которым уже может работать наша функция atoi
cout <<"Number = "<< a<<endl; //Output:123
//А для перевода числа в string существует функция to_string();
text1 = to_string(7832.23);
cout <<"Number = "<< text1<<endl; //Output:7832.23
text1 = to_string(a);
cout << "Number = " << text1 << endl; //Output:123
}
```

Как вы видите, класс string облегчает работу со строками. Также объект класса string можно считать динамическим, так как он сам выделяет памяти столько, сколько ему нужно для хранения информации, которую мы хотим в него поместить. Давайте рассмотрим пример с использованием string в классе.

Внимание! String – это класс, в котором уже прописан деструктор. Поэтому, если вы используете string в своем классе, то создавать деструктор для освобождения памяти не нужно. String все сделает сам.

Пример:

```
#include "pch.h"
#include <iostream>
#include <string> //Подключаем наш класс string и функции для работы с классом string (к примеру, to_string и т. д.)
using namespace std;
class Stroka
{ public:
String info; //Создаем поле, в котором будет храниться строка
```

```
Stroka(string s) //Создаем конструктор, который будет принимать строку
{
info = s; //Помещаем строку, полученную конструктором в поле info
};
void Print() //Метод для вывода поля info
{ cout << info << endl; //Выводим содержимое info
} };
int main()
{ Stroka text1("12345"),text2("I know OOP."); //Создаем объекты и передаем строки в конструктор
text1.Print(); //Output: "12345"
text2.Print(); //Output: "I know OOP."
}
```

Как мы видим, результат будет точно таким же, как и при использовании класса.

Перегрузка операторов в C++ (краткая характеристика)

В языке C++ наряду с обычными конструкциями операций над операндами со стандартными типами (арифметические, операции отношений, логические операции и т. д.) существует механизм перегрузки этих операторов, позволяющий манипулировать объектами классов пользователя, используя обычный синтаксис языка, т. е. привычную (удобную) запись операций. Если компилятор обнаружил, что в программе есть перегрузка операций для объектов конкретного типа, введенных пользователем, то, найдя эти операции, компилятор анализирует их. Затем возможно 2 варианта:

1) если операнды имеют встроенный тип данных, то будет заложена стандартная операция;

2) если операнды имеют тип данных пользователя, т. е. являются объектами класса, для которого операция была перегружена, то будет заложена перегруженная операция, запрограммированная пользователем.

Операции для конкретного класса перегружаются или методом этого класса, или функцией-другом для этого класса следующего вида:

```
тип operator@(список параметров с указанием их типа)
 {
код, определяющий смысл указанной операции,
}
```

@ – символ перегружаемой операции.

Так как операция @ перегружается для объекта конкретного класса, а сам класс – это тип этого объекта, то тип – это идентификатор этого класса.

Отличия перегрузки операций при помощи метода класса или функции-друга следующие:

а) @a – унарная операция может быть перегружена методом без параметров

или функцией-другом с одним параметром, так как у метода имеется первый скрытый от нас параметр (указатель this), который автоматически устанавливается на единственный операнд унарной операции, а функция-друг такого указателя не имеет, потому при перегрузке ее параметров, ее операнд и передают;

б) al @ a2 – бинарная операция перегружается методом класса с одним параметром или функцией-другом с двумя параметрами, так как при ее перегрузке методом на первый операнд устанавливается скрытый указатель (this), а второй передается через параметр; при перегрузке функцией-другом первый операнд передается функцией через первый параметр, второй – через второй.

Для чего нужна перегрузка операторов? Давайте представим, что мы создали класс Drobi, в котором хранится 2 поля: n – для хранения числителя и d – для хранения знаменателя. Мы создали 2 объекта этого класса и положили в него значения:

Drobi a(1,3);

Drobi b(2,4);

И теперь необходимо сложить первую дробь со второй, например, Drobi c = a + b.

Но есть проблема. Drobi – это наш пользовательский класс, и компилятор не знает, что для того чтобы сложить 1/3 и 2/4, необходимо сначала привести их к общему знаменателю, а только потом сложить числители. Именно при перегрузке оператора мы пишем определенный алгоритм, который должен запускаться при использовании данного оператора с пользовательским типом данных.

В нашем случае, когда компилятор увидит, что мы хотим сложить две дроби a + b, то он обратится к оператору данного класса, где будет написан алгоритм, который будет приводить дроби к общему знаменателю, складывать числители и возвращать правильно сложенные числитель и знаменатель.

Есть 3 разных способа перегрузки операторов:

1) через дружественные функции;

2) через обычные функции;

3) через методы класса.

Пример перегрузки через дружественные функции:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Drobi
{
private:
int n, d; //n – хранит числитель, d – хранит знаменатель
public:
Drobi(int n, int d) //Конструктор для заполнения числителя и знаменателя
{       this->n = n;
        this->d = d;
```

```
}
void Print() //Вывод числителя и знаменателя
{       cout << "numerator = " << n << endl << "denominator = " << d << endl;     }
friend Drobi operator+(const Drobi &d1, const Drobi &d2); //Объявляем дружественную функцию
};
Drobi operator+(const Drobi & d1, const Drobi & d2) //Описываем нашу дружественную функцию. В качестве параметров в функцию передаем адреса наших объектов: d1 соответствует объекту, стоящему слева от "+", d2 – объекту, стоящему справа от "+"
{       return Drobi( (d1.n*d2.d + d1.d*d2.n), d1.d*d2.d ); //Приводим дроби к общему знаменателю, складываем числители, складываем знаменатели, возвращаем объект класса Drobi
}
int main()
{
Drobi a(1, 3),b(2,4); //Создаем 2 объекта класса Drobi и заполняем его поля
Drobi c = a + b; //Объект "c" теперь указывает на область памяти, которая выделилась при выполнении operator+
c.Print(); //Выводим значения объекта "c"
}
```

С оператором "+" все понятно. А что означает объект "c", указывающий на область памяти, которая выделилась при выполнении operator +?

На самом деле до того, как мы опишем оператор "=" для нашего класса, наш объект будет просто указывать на область памяти, которая стоит справа от "=".

К примеру, если мы напишем

```
Drobi obj(1, 3),obj1(2,4);
obj = obj1;
```

то значение полей объекта "obj" не поменяются. Объект "obj" будет просто указывать на ту же область памяти, где хранится наш объект obj 1(у этих переменных станет одинаковый адрес). В результате чего мы потеряем адрес, по которому хранилась информация об объекте "obj".

Пример перегрузки оператора через обычную функцию

Операторы можно перегружать, используя и обычные функции. Единственное, в чем они уступают дружественным функциям, – не имеют доступа к полям и методам с модификатором доступа **private.** Поэтому перегрузки оператора через обычные функции используют, только если этому оператору не нужны значения полей с модификатором доступа **private** или в классе предусмотрены методы, которые смогут нам вернуть значение наших полей с доступом **private**.

В нашем классе нет таких методов, которые помогут нам вернуть значение полей n и d, поэтому мы их добавим.

```
#include "pch.h"
#include <iostream>
```

```
using namespace std;
class Drobi
{ private:
int n, d; //n – хранит числитель, d – хранит знаменатель
public:
Drobi(int n, int d) //Конструктор для заполнения числителя и знаменателя
{       this->n = n;
        this->d = d;    }
void Print() //Вывод числителя и знаменателя
{       cout << "numerator = " << n << endl << "denominator = " << d << endl;     }
int ReturnN() //Функция, которая возвращает нам значение поля n
{       return n; }
int ReturnD() //Функция, которая возвращает нам значение поля d
{ return d;
} };
```

Drobi operator+(Drobi & d1, Drobi & d2) //Описываем нашу функцию. В качестве параметров в функцию передаем адреса наших объектов: d1 соответствует объекту, стоящему слева от "+", d2 – объекту, стоящему справа от "+"

```
{ return Drobi((d1.ReturnN() * d2.ReturnD() + d1.ReturnD()*d2.ReturnN()), d1.ReturnD()*d2.ReturnD()); //Приводим дроби к общему знаменателю, складываем числители, возвращаем объект класса Drobi
}
int main()
{ Drobi a(1, 3), b(2, 4); //Создаем 2 объекта класса Drobi и заполняем его поля
Drobi c = a + b; //объект "c" теперь указывает на область памяти, которая выделилась при выполнении operator+
c.Print(); //Выводим значения объекта "c"
}
```

Результат у нас получится таким же, что и при использовании дружественной функции. Единственное отличие – это способ получения значения полей n и d у наших объектов.

Пример перегрузки оператора через метод класса

Перегрузка операторов через методы класса очень похожа на **перегрузку операторов через дружественные функции**. Но при перегрузке оператора через метод класса доступ к объекту, для которого мы вызвали оператор, можно будет получить при помощи специального указателя this.

Пример:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Drobi
{ private:
```

```
int n, d; //n – хранит числитель, d – хранит знаменатель
public:
Drobi(int n, int d) //Конструктор для заполнения числителя и знаменателя
{       this->n = n;
        this->d = d;  }
void Print() //Вывод числителя и знаменателя
{ cout << "numerator = " << n << endl << "denominator = " << d << endl;    }
Drobi operator+(Drobi &d2) //Создаем метод для перегрузки оператора "+" в нашем классе
{ return Drobi((this->n*d2.d + this->d*d2.n), this->d*d2.d); //this – указывает на объект, который стоит слева от оператора "+"
};
Drobi operator+(int d2) //Создаем метод для перегрузки оператора "+" в нашем классе
{ return Drobi( this->n + (d2*this->d), this->d); //Возвращаем нашу дробь с новыми значениями
}; };
int main()
{ Drobi obj(1, 3), obj1(2, 4); //Создаем 2 объекта класса Drobi и заполняем его поля
Drobi c = obj + obj1; //В данном случае объект "obj" вызовет перегрузку метода operator+, а значит, к полям "obj" можно будет обратиться при помощи this
c.Print(); //Выводим значения объекта "c"
//Реализуем еще метод для прибавления целого числа к объекту нашего класса Drobi
Drobi obj2 = obj + 5;
obj2.Print(); //Выводим значения объекта "obj2"
}
```

Давайте рассмотрим пример решения задачи при помощи перегрузки операторов.

Задача: ввести строку символов S1, признак окончания ввода строки – нажатие клавиши Enter. Программа должна содержать перегруженную операцию "=", использование которой скопирует S1 в S2, убрав оттуда 2 первых и 2 последних символа:

Решение:

1) Использование массива Char:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Stroka
```

```
{ public:
char Text[256]; //Создаем наш массив char
void Print() //Метод для вывода содержимого поля Text
{ cout << Text << endl; //Вывод
}
void operator=(Stroka &str) //Перегружаем оператор = . str – объект, стоящий справа от "="
{for (int i = 2; i < strlen(str.Text)-2; i++) //Заполняем массив Char объекта, стоящего слева от "=".
{Text[i-2] = str.Text[i];}
Text[strlen(str.Text)-4] = '\0';} }; //Добавляем '\0' для обозначение конца строки
int main()
{       Stroka t1, t2; //Создаем 2 объекта
gets_s(t1.Text); //Помещаем строку, введенную пользователем, в поле Text объекта t1
t1.Print(); //Выводим строку
t2 = t1; //Записываем в поле Text объекта t2 строку из объекта t1 без 2 первых и последних символов при помощи перегрузки оператора "="
t2.Print(); //Выводим строку
}
```

2) Использование String

```
#include "pch.h"
#include <iostream>
#include <string>
using namespace std;
class Stroka
{ public:
string Text; //Создаем наш объект string
void Print() //Метод для вывода содержимого поля Text
{       cout << Text << endl; //Вывод
}
void operator=(Stroka &str) //Перегружаем оператор = . str – объект, стоящий справа от "="
{       for (int i=2;i<str.Text.length()-2;i++) //Идем по строке в объекте str. Обход начинаем сразу с третьего символа, не доходя до конца строки второго символа
{       Text += str.Text[i]; //Добавляем в нашу строку Text i-й символ из строки объекта str}
} };
int main()
{Stroka t1, t2; //Создаем 2 объекта
```

```
getline(cin, t1.Text); //Помещаем строку, введенную пользователем, в поле Text объекта t1
cin.clear(); //Очищаем поток для ввода
t1.Print(); //Выводим строку
t2 = t1; //Записываем в поле Text объекта t2 строку из объекта t1 без 2 первых и последних символов при помощи перегрузки оператора "="
t2.Print(); //Выводим строку
}
```

Задания к лабораторной работе

Уровень 1

Упражнение: создайте класс, в котором будет поле для хранения строки (поле может быть типа char или string на ваше усмотрение). Реализуйте метод вывода содержимого этого поля, хранящего строку, и перегрузку оператора "=" (любым из 3 способов перегрузки операторов), чтобы результат удовлетворял условию задания вашего варианта.

Варианты:

1) Вас попросили разобраться с читерами в популярной MMORPG-игре "Sword, Magic, Shield, Bow, Arrow and something else". Сервер постоянно возвращает непонятные строки, например "qqfdsdvn3u9r8329fjf239 n392hfnvowgn we 3wrfiodg" и т. д. (никто не понимает, что они означают, но если все работает, то не трогайте). Но одно разработчики знают точно: если в строке, возвращаемой сервером, встречаются символы в верхнем регистре, к примеру "fdghjHJHKLdsjf FG F ddsgsdgwfg3 dsggsege", то это значит, что кто-то использует читы в игре. Вы опытный разработчик и знаете, как выполнять такие задачи. Если символы в верхнем регистре – это читеры, то сервер просто должен перестать показывать эти символы в верхнем регистре.

Ваша задача: создать класс ServerAnswer, в котором будет поле для хранения строки и метод для просмотра этого поля, а также программа должна содержать перегруженную операцию "=", которая скопирует строку, введенную пользователем в поле, предназначенное для хранения строки в классе ServerAnswer, удалив оттуда все символы в верхнем регистре.

2) Вы решили перевестись на кафедру математики. На этой кафедре работают люди, в голове которых одни числа, из-за чего общаться они тоже начали на языке цифр, а точнее вместо символа они используют его код из ASCII Table. К примеру, фраза "Hello" у них будет записана как "72 101 108 108 111". Вы пока не выучили ASCII Table и не можете так быстро переводить предложения, поэтому решили написать программу, которая поможет вам в этом.

Ваша задача: создать класс MathematicalLanguage, в котором будет поле для хранения строки и метод для просмотра этого поля, а также программа должна со-

держать перегруженную операцию "=", которая поместит строку, введенную пользователем, в поле, предназначенное для хранения строки в классе MathematicalLanguage, заменив все символы введенной строки на ее код в ASCII Table (не забывайте, что у пробела тоже есть свой код).

3) Вчера вы прочитали журнал "IT-сфера. Если нам лень что-то делать, мы напишем программу, которая сделает это за нас", в котором была интересная статья о том, что можно отсканировать лист с рукописным вводом, и сканер переведет этот рукописный ввод в редактируемые форматы текста, с которым сможет работать компьютер. К вам сразу пришла идея запустить такую программу в аптеках вашей больницы. Пришел, приложил рецепт к сканеру, и фармацевт уже несет нужное вам лекарство. Однако программе будет сложно понять, где закончился диагноз и началось название лекарства. Поэтому вы сказали всем врачам вашей больницы писать названия лекарств в фигурных скобках "{ }". Необходимо создать программу, которая будет выводить все эти названия, заключенные между фигурными скобками.

Ваша задача: создать класс Recipe, в котором будет поле для хранения строки и метод для просмотра этого поля, а также программа должна содержать перегруженную операцию "=", которая поместит из строки, введенной пользователем, в поле, предназначенное для хранения строки в классе Recipe, строки, заключенные между фигурными скобками "{ }" (не забывайте учитывать то, что фигурных скобок может быть несколько и строки между другими фигурными скобками тоже необходимо вывести).

4) Вашему начальнику постоянно приходит большое количество счетов. Он попросил вас написать программу, которая из текста выделит только числа.

Ваша задача: создать класс BillsForCommunal, в котором будет поле для хранения строки и метод для просмотра этого поля, а также программа должна содержать перегруженную операцию "=", которая поместит из строки, введенной пользователем в поле, предназначенное для хранения строки в классе BillsForCommunal, все числа, встречающиеся в введенной строке (при этом будем считать, что числа будут только положительными и целыми, т. е. после обработки строки "afsfefe77-3 9.2 dg3 -24" должно вернуться "77 3 9 2 3 24").

Уровень 2

Упражнение: создайте класс, в котором будет поле для хранения строки (поле может быть типа char или string на ваше усмотрение). Реализуйте метод вывода содержимого этого поля, хранящего строку, и перегрузку оператора "=" (любым из 3 способов перегрузки операторов), чтобы результат удовлетворял условию задания вашего варианта.

Варианты:

1) Вашему начальнику доложили, что простое удаление информации о читерах из логов не решит проблему, а только скроет ее, и теперь от вас требуют решить

эту проблему. Вы применяете "вычисление по IP". IP игрока вычисляется следующим образом: сначала идет стандартная часть 192.168, потом вставляется сумма цифр номера позиции нашего символа в верхнем регистре в строке, затем – код этого символа в верхнем регистре, который берется из ASCII Table. К примеру, при вводе "asdHM332 23 dfs sdfG" должно вернуться "192.168.3.72 192.168.4.77 192.168.10.71". И, получив ip всех читеров, вы сможете отдать их администратору, а он их заблокирует.

Ваша задача: создать класс IPStorage, в котором будет поле для хранения строки и метод для просмотра этого поля, а также программа должна содержать перегруженную операцию "=", которая сгенерирует IP по методу, описанному в задании, на основе символов в верхнем регистре из строки, введенной пользователем в поле, предназначенное для хранения строки в классе IPStorage.

2) Вы уже неделю трудитесь вместе с математиками и отлично справляетесь со своими заданиями. В связи с этим ваш руководитель пригласил вас в элитный клуб математиков "Пифагоровы штаны". Чтобы пройти в клуб, на входе необходимо выполнить задание. Дается строка, состоящая из слов, разделенных одним или несколькими пробелами, и вам необходимо разбить сумму кодов символов самого длинного слова на общее число символов в строке (без учета пробелов). Так как считать это сложно, вы решили написать программу, которая в этом поможет.

Ваша задача: создать класс Result, в котором будет поле для хранения строки и метод для просмотра этого поля, а также программа должна содержать перегруженную операцию "=", которая вернет результат деления суммы кодов самого длинного слова на общее число символов строки (без учета пробелов) из строки, введенной пользователем в поле, предназначенное для хранения строки в классе Result.

3) Вы закупили хорошие японские сканеры, но не учли, что в Японии чтение идет справа налево и поэтому названия лекарств выводятся неправильно. Вам необходимо это исправить.

Ваша задача: создать класс Drug, в котором будет поле для хранения строки и метод для просмотра этого поля, а также программа должна содержать перегруженную операцию "=", которая вернет строки, заключенные между фигурными скобками "{ }" в перевернутом виде, из строки, введенной пользователем в поле, предназначенное для хранения строки в классе Drug. К примеру, при вводе "AEgsdgwf124 fdh{abcd45,}{ergrr}{gfdgfdg" должно вернуться ",54dcba rrgre".

4) Ваш начальник попросил вас создать программу, которая выводила бы сумму всех чисел, найденных в тексте.

Ваша задача: создать класс TotalScore, в котором будет поле для хранения строки и метод для просмотра этого поля, а также программа должна содержать перегруженную операцию "=", которая вернет сумму всех чисел из строки, введенной пользователем, в поле, предназначенное для хранения строки в классе To-

talScore. К примеру, при вводе "Afdee54 6 rg-6 PPe-5-4 4.3 3,70,3 011" должно вернуться "54+6-6-5-4+4.3+3.70+3+11=67".

Контрольные вопросы и задания

1) Как создать динамическую строку?
2) Что такое "перегрузка операций"?
3) Какие способы перегрузки операторов вы знаете?
4) Каковы отличия при перегрузке унарных и бинарных операций?
5) Какие функции для работы со строками вы знаете?
6) Что вы знаете о классе String?
7) Почему при перегрузке унарного оператора через дружественные функции требуется в качестве аргумента передавать операнд, а при перегрузке через метод он не требуется?
8) Каков будет результат выполнения приведенного кода?

```
string text = "If you do not know the answer, you will get 2";

for(int i = text.length() - 1; i > -1; i--)
{
        cout << text[i];
}
```

Лабораторная работа №6
Наследование и механизм виртуальных функций

Цель работы: изучить одну из базовых концепций ООП – наследование классов в C++, заключающуюся в построении цепочек классов, механизмов виртуальных функций, связанных иерархически.

Теоретические сведения

Наследование – такое соотношение между классами, когда один класс использует часть другого, добавляя ему нечто свое и таким образом расширяя его возможности. При этом первый класс, который описывает наиболее общие свойства ряда объектов, называется базовым, второй класс – производным.

Определение производного класса имеет следующий синтаксис:

class ID_производного_класса: ID_базового_класса

{ код_производного_класса };

Одна из особенностей порожденного класса – видимость унаследованных компонент базового класса со степенью защиты protected и public (атрибуты базового класса). Компоненты базового класса с ключевым словом private производному классу недоступны.

Производный класс может служить базовым классом для создания следующего, производного, класса на более низком уровне иерархии классов. Наследование называется простым, если производный класс имеет одного родителя. В C++ наследование может быть множественным. Множественное наследование позволяет одному классу обладать свойствами двух и более родительских классов.

Видимые компоненты базового класса со степенью защиты protected и public в производном классе становятся private, а значит, не могут быть наследованы производными классами на более низких уровнях иерархии классов. Это связано с тем, что в заголовке производного класса перед ID базового класса по умолчанию компилятор указывает атрибут доступа private. Если же указать явно общедоступный атрибут

class ID_производного_класса: public ID_базового_класса

то унаследованные компоненты базового класса сохранят степень защиты в производном классе и могут быть унаследованы производным классом на следующем уровне иерархии.

Перед активизацией конструктора производного класса производится вызов конструктора базового класса. Если базовый класс является производным классом на более высоком уровне иерархии, то процесс вызова конструктора производится по своей цепи иерархии классов, пока не доберется до базового класса самого верхнего уровня иерархии. Таким образом, объекты базового класса всегда существуют в составе объектов производного класса. В противоположность этому деструктор производного класса вызывается перед вызовом деструктора базового класса. Это

объясняется тем, что уничтожение объекта базового класса влечет за собой уничтожение и объекта производного класса.

Если конструктор базового класса имеет параметры для инициализации своих объектов, то конструктор производного класса обязан обеспечить передачу ему исходных данных. Для этой цели используется форма конструктора со списком инициализаторов, в который включаются идентификаторы конструкторов базового класса (явный вызов конструктора базового класса). При этом последние должны быть объявлены с атрибутом минимум protected.

Продемонстрируем данный механизм на конкретном примере:

```
#include "pch.h"
#include <iostream>
#include <string>
using namespace std;
class Basic //Базовый класс
{ public:
int BasicInt; //Поле типа public
Basic(int a) //Конструктор
{ BasicInt = a + 5;    };
protected:
int BasicProtectedInt; //Защищенное поле protected
private:
int BasicPrivateInt; //Защищенное поле private
};
class First:protected Basic //Наследуем с доступом protected
{ public:
int x;
First(int a,int b):Basic(a), x(b), y(b + 2), z(a + 2) //Передаем в конструктор Basic значение, которое было введено в конструктор First, и инициилизируем поля x, y, z
{       BasicProtectedInt = b;
        //BasicPrivateInt;
};
private:
int y;
protected:
int z; };
class Second:public Basic //Наследуем с доступом public
{ public:
int x;
Second(int a,int b):Basic(a),x(b),y(b+2),z(a+2) //Передаем в конструктор Basic значение, которое было введено в конструктор Second, и инициализируем поля x, y, z
{       BasicProtectedInt = b;
        //BasicPrivateInt;
};
```

```
private:
int y;
protected:
int z; };
int main()
{ First obj(1,3);
cout << obj.BasicInt; //Нет доступа к полю BasicInt, так как First наследуется с доступом protected, из-за чего полю BasicInt присвоился доступ protected вместо public
Second obj2(1, 3);
cout << obj2.BasicInt; //Имеем доступ к полю BasicInt
}
```

Как правило, коды методов выносят из пространства класса. Тогда для предыдущего примера:

```
#include "pch.h"
#include <iostream>
#include <string>
using namespace std;
class Basic //Базовый класс
{ public:
Basic(int a);
int BasicInt; //Поле типа public
protected:
int BasicProtectedInt; //Защищенное поле protected
private:
int BasicPrivateInt; //Защищенное поле private
};
Basic::Basic(int a)
{ BasicInt = a + 5; }
class First:protected Basic //Наследуем с доступом protected
{ public:
int x;
First(int a, int b);
private:
int y;
protected:
int z; };
First::First(int a, int b):Basic(a), x(b), y(b + 2), z(a + 2)
{       BasicProtectedInt = b; }
class Second:public Basic //Наследуем с доступом public
{ public:
int x;
Second(int a,int b);
private:
```

```
int y;
protected:
int z; };
Second::Second(int a, int b):Basic(a), x(b), y(b + 2), z(a + 2)
{ BasicProtectedInt = b; }
int main()
{ First obj(1,3);
cout << obj.BasicInt; // Нет доступа к полю BasicInt, так как First наследуется с доступом protected, из-за чего полю BasicInt присвоился доступ protected вместо public
Second obj2(1, 3);
cout << obj2.BasicInt; //Имеем доступ к полю BasicInt
}
```

Использование косвенной адресации с установкой указателей на базовый класс

Механизм косвенной адресации рассмотрим на конкретном примере:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Basic //Базовый класс
{ public:
Basic(int a);
int BasicInt; //Поле типа public
protected:
int BasicProtectedInt; //Защищенное поле protected
private:
int BasicPrivateInt; //Защищенное поле private
};
Basic::Basic(int a)
{       BasicInt = a + 5; }
class First:protected Basic //Наследуем с доступом protected
{ public:
int x;
First(int a, int b);
private:
int y;
protected:
int z; };
First::First(int a, int b):Basic(a), x(b), y(b + 2), z(a + 2)
{       BasicProtectedInt = b; }
class Second:public Basic //Наследуем с доступом public
{ public:
int x;
Second(int a,int b);
```

```
private:
int y;
protected:
int z; };
Second::Second(int a, int b):Basic(a), x(b), y(b + 2), z(a + 2)
{ BasicProtectedInt = b; }
int main()
{First f(1,2); // Конструктор класса First создает объект f
Second s(3,4); // Конструктор класса Second создает объект S
Basic *b; //Указатель на базовый касс
b = &s; //Указатель Базового класса указывает на объект производного Second
b = &f; //Указатель Базового класса не может указать на объект производного First из-за того, что First наследуется с доступом protected
//Хоть b сейчас и указывает на объект класса Second, без преобразования указателя C++ не позволит нам сейчас использовать поля объекта наследника
//cout << b->x; //Не сработает...
int i = b->BasicInt;    //Базовый класс виден напрямую
//Стиль языка C для явного преобразования типов с использованием круглых скобок:
//int j = ((Second*)(b))->x;
//Использование кастов c++ для преобразования типов
//Введенные в C++ касты (приемы преобразований типов) не дают программе собраться, если преобразование слишком опасное. Эти ограничения повышают надежность вашей программы
int j = static_cast<Second*>(b)->x;
// Через переменные печатаем их текущее состояние
cout << " BasicInt = " << i << endl;
cout << " x = " << j << endl;}
```

Механизм виртуальных функции

Механизм виртуальных функций – одна из основных концепций объектно-ориентированного программирования. Данный механизм предполагает использование идеи "один интерфейс – множество методов реализации". Эта идея заключается в том, что базовый класс обеспечивает для производных классов все элементы, которые эти классы могут использовать непосредственно, а также содержит набор функций, которые производные классы должны реализовать путем их переопределения (создание собственного кода функции в производном классе, которые позволяют решить поставленную перед производным классом задачу). Виртуальная функция – это функция, объявленная с ключевым словом virtual в базовом классе и переопределенная в одном или нескольких производных от этого класса. Обязательное требование: заголовок функции должен быть точно такой же, как в базовом классе (и имя, и список параметров должны быть одинаковые). Тогда при создании объекта или базового, или одного из производных классов компилятор определяет,

какую из функций требуется вызвать, основываясь на типе (ID класса) объекта. Таким образом, компилятор на этапе компиляции, обнаружив виртуальные функции с одинаковыми заголовками, но с разными кодами, не конкретизирует, какой из виртуальных методов будет вызываться, а отводит для их адресов место в специальной таблице адресов виртуальных методов. И далее на этапе выполнения зарезервированное в таблице место инициализируется адресом соответствующей виртуальной функции конструктором того класса, объект которого создается в данный момент выполнения программы. Такой процесс в C++ получил название "позднее связывание".

Как правило, на практике в базовом классе приводят только прототип виртуального метода, который определяет общий интерфейс, указывающий, с какими данными необходимо работать. А в производном классе приводят полные коды, которые определяют способы обработки указанных данных.

Если требование полного сохранения заголовка виртуального метода в производном классе нарушено, то компилятор, обнаружив это, механизм виртуальных функций проигнорирует и произведет на этапе компиляции его перегрузку.

Давайте решим задачу с использованием виртуальных функций, чтобы лучше разобраться с ними.

Создадим базовый класс Weapons, в котором будет храниться количество патронов в магазине для каждого производного от него класса и 3 класса оружия pistol, MachineGun и WaterGun. Каждое оружие должно стрелять по-своему и у каждого оружия свое количество патронов в магазине. Давайте посмотрим, как удобно решается такая задача с использованием виртуальных функций:

```
#include "pch.h"
#include <iostream>
using namespace std;
class Weapons //Базовый класс
{ public:
int AmmunitionStore; //Храним количество патронов в магазине
Weapons(int a);
virtual void ReloadingWeapons(); //Создаем прототип метода перезарядки оружия
virtual void Shoot(); //Создаем прототип метода стрельбы оружия
};
Weapons::Weapons(int a) //Заряжаем оружие
{ AmmunitionStore = a; }
void Weapons::ReloadingWeapons()
{
}
void Weapons::Shoot()
{
}
//Описываем пистолет
```

```
class pistol:public Weapons
{ public:
 pistol(int a);
 void ReloadingWeapons() override;
 void Shoot() override; };
pistol::pistol(int a):Weapons(a)
{
}
void pistol::ReloadingWeapons() //Описываем метод перезарядки оружия для pistol
{ cout << "\nReloadingWeapons\n";
AmmunitionStore = 8; //В пистолет заряжаем 8 патронов
}
void pistol::Shoot() //Описываем метод стрельбы оружия для pistol
{       if (AmmunitionStore==0)
{       cout << "AmmunitionStore empty.\n";
        return; }
for (int i=0;i< AmmunitionStore;i++)
{       cout << "BANG. ";  }
AmmunitionStore = 0; //Патроны закончились
cout << endl; }
//Описываем пистолет end
//Описываем автомат
class MachineGun:public Weapons
{ public:
MachineGun(int a);
void ReloadingWeapons() override;
void Shoot() override; };
MachineGun::MachineGun(int a):Weapons(a)
{
}
void MachineGun::ReloadingWeapons() //Описываем метод перезарядки оружия для
MachineGun
{ cout << "\nReloadingWeapons\n";
AmmunitionStore = 31; //В автомат заряжаем 31 патрон
}
void MachineGun::Shoot() //Описываем метод стрельбы оружия для MachineGun
{       if (AmmunitionStore == 0)
{       cout << "AmmunitionStore empty.\n";
        return; }
for (int i = 0; i < AmmunitionStore/3; i++)
{       cout << "TRATATA. "; }
AmmunitionStore = 0; //Патроны закончились
cout << endl; }
//Описываем автомат end
```

```
//Описываем водный пистолет
class WaterGun:public Weapons
{ public:
WaterGun(int a);
void ReloadingWeapons() override;
void Shoot() override; };
WaterGun::WaterGun(int a):Weapons(a)
{
}
void WaterGun::ReloadingWeapons() //Описываем метод перезарядки оружия для WaterGun
{       cout << "\nReloadingWeapons\n";
AmmunitionStore = 10; //В водный пистолет наливаем воды на 10 выстрелов
}
void WaterGun::Shoot() //Описываем метод стрельбы оружия для WaterGun
{       if (AmmunitionStore == 0)
{       cout << "AmmunitionStore empty.\n";
        return; }
for (int i = 0; i < AmmunitionStore; i++)
{       cout << "Piu. ";       }
AmmunitionStore = 0; //Патроны закончились
cout << endl; }
//Описываем водный пистолет end
int main()
{ int chose;
//Создаем объекты нашего оружия
WaterGun WG(5);
pistol P(3);
MachineGun MG(25);
Weapons *arsenal = &P; //Создаем указатель базового класса, который мы сможем направлять на каждый производный от него объект, созданый ранее. По стандарту поставим его на pistol
while (true)
{       cout << endl << "Chose Weapon: \n1:Pistol\n2:MachineGun\n3:WaterGun\nYor chose:"; cin >> chose;
        switch (chose)
        {       case 1:
        {       arsenal = &P; //Берем наш пистолет
                break; }
        case 2:
        {       arsenal = &MG; //Берем наш автомат
                break; }
        case 3:
        {       arsenal = &WG; //Берем наш водный пистолет
```

```
            break; }
        default:
            cout << endl << "Invalid input." << endl;
            break; }
        //Выполняем операции с оружием, на которое указывает arsenal
        arsenal->Shoot(); //Выстреливаем все патроны
        arsenal->Shoot(); //Получаем сообщение о нехватке патронов
        arsenal->ReloadingWeapons(); //Перезаряжаем оружие
        arsenal->Shoot(); //Выстреливаем все патроны
} }
```

Программа должна содержать базовый класс, включающий поля и виртуальные методы, необходимые для выполнения задания в соответствии с вариантом, и производные от базового классы с перезаписанными методами в соответствии с заданием выбранного варианта.

Базовый класс должен содержать конструктор с параметрами для инициализации полей базового класса в динамической памяти, деструктор для освобождения занятой памяти, виртуальные методы просмотра текущего состояния и переустановки значений полей базового класса в новое состояние.

Производные классы должны иметь:

– конструктор с параметрами и списком инициализаторов, передающий данные конструктору базового класса;

– переопределенные методы просмотра текущего состояния объектов и их переустановки в новое состояние.

Задания к лабораторной работе

Уровень 1

Варианты:

1) После того как вы успешно справились с читерами в игре "Sword, Magic, Shield, Bow, Arrow and something else", вас попросили помочь с разработкой нового патча обновления для этой игры. Это глобальный патч, в котором планируют добавить очень многое: новых монстров, героев, NPC, локации, оружие, заклинания, механику и даже новое название самой игры. Конкретно вы занимаетесь разработкой новых монстров. Так как монстров необходимо добавить очень много, то вы решили описать общую структуру нового монстра и разработать 3 примера его создания на основе этой структуры, а дальше передать этот пример стажерам, чтобы они по аналогии создавали новых монстров.

Ваша задача: создать базовый класс Monsters, в котором будут поля HP, damage, armour и 2 виртуальных метода PrintInfo() и Scream(), а также 3 производных класса от класса Monsters (название классам придумайте самостоятельно), в которых будет происходить переопределение методов PrintInfo() и Scream(). Переопре-

деленное PrintInfo() должно вывести поля HP, damage, armour, умноженные на константное значение (на какое значение умножать, решайте сами), а метод Scream() должен вывести крик этого монстра (как будет кричать ваш монстр, придумайте сами) к примеру "Life for Nerzula. "

2) На кафедре математики вас попросили подготовить сообщение про число Эйлера и число Пи для рассказа студентам. Во время подготовки вы округлили число Пи (3,14…) и число Эйлера (2,718…), в результате чего пришли к выводу что число Пи == числу Эйлера. После этого вас уволили. Вы решили пойти в администрацию центра, чтобы вам помогли найти работу. Вам предложили поработать в администрации. Вашей задачей будет подсчет расходов на каждую кафедру. Вы решили написать программу, которая сама делала бы все необходимые расчеты.

Ваша задача: создать базовый класс Faculty, в котором будут поля NumberOfEmployees, EmployeeSalary, Surcharge и 2 виртуальных метода PrintInfo() и CostsWithSurcharge(), а также 3 производных класса от класса Faculty (название классам придумайте самостоятельно), в которых будет происходить переопределение методов PrintInfo() и CostsWithSurcharge (). Переопределенное PrintInfo() должно вывести поля NumberOfEmployees, EmployeeSalary и расход на кафедру, который равен NumberOfEmployees* EmployeeSalary, а метод CostsWithSurcharge () должен вывести поля NumberOfEmployees,EmployeeSalary, Surcharge и расход на кафедру, который равен NumberOfEmployees * ((EmployeeSalary + Surcharge) * 0.13).

3) Вы уже целый месяц работаете в больнице главврачом и уже многое успели сделать, но не успели выдать сотрудникам заработную плату. Вам нужно срочно посчитать, сколько и кому необходимо выплатить. Вы решили написать программу, которая все посчитает.

Ваша задача: создать базовый класс Employees, в котором будут поля HourlyPayment, WorkedHours, Surcharge и 2 виртуальных метода PrintInfo() и CostsWithSurcharge(), а также 3 производных класса от класса Employees (название классам придумайте самостоятельно), в которых будет происходить переопределение методов PrintInfo() и CostsWithSurcharge (). Переопределенное PrintInfo() должно вывести поля HourlyPayment, WorkedHours и зарплату сотрудника, которая равна HourlyPayment * WorkedHours, а метод CostsWithSurcharge () – поля WorkedHours, HourlyPayment, Surcharge и зарплату сотрудника, которая равна WorkedHours * (HourlyPayment + Surcharge * 0.13).

4) Ваш начальник предпочитает инновационные технологии, он попросил вас написать программу, которая показывала бы, сколько денег в месяц нужно будет платить человеку, который берет ипотеку в банке, с учетом процента, установленного этим банком. Ваш начальник владеет несколькими банками, у каждого из них свой процент по ипотеке, и это нужно учитывать.

Ваша задача: создать базовый класс Bank, в котором будут поля AmountOfDebt, NumberOfMonths и 2 виртуальных метода PrintInfo() и Debt(), а также 3 производных класса от класса Bank (название классам придумайте самостоятельно), в которых будет происходить переопределение методов PrintInfo() и Debt (). Переопределенное PrintInfo() должно вывести поля AmountOfDebt, NumberOfMonths, а метод Debt () – сумму, которую должен будет платить в месяц человек именно этому банку.

Данная сумма вычисляется по формуле

(AmountOfDebt+AmountOfDebt×[(процент банка придумайте сами) /100]) /NumberOfMonths

Уровень 2

Варианты:

1) Ваши стажеры добавили большое количество новых монстров и теперь необходимо проверить, сможет ли победить их наш игрок.

Ваша задача: добавить в ранее написанную программу класс Player, в котором будут поля HP, damage, armour. Поля заполняются при помощи конструктора. В классе Player должен быть также описан метод WhoWin(), принимающий в качестве аргумента указатель базового класса Monsters.Устанавливайте этот указатель на объекты ваших производных классов от Monsters и передавайте его в функцию WhoWin(), которая должна вывести сообщение "Player win." в случае если player оказался сильнее монстра и сообщение "Player lose.", если монстр оказался сильнее. Определить, кто сильнее, очень просто. Если HP(игрока)/(Урон монстра – броня игрока) > HP(Монстра)/ (Урон игрока – броня монстра), то игрок победил, а в противном случае – проиграл.

2) После того как вы показали общую сумму расходов, ваш начальник попросил вас уменьшить количество сотрудников на кафедрах на 10 %, а 50 % зарплаты уволенных поровну распределить между оставшимися сотрудниками.

Ваша задача: добавить в ранее написанную программу класс Dismisses, в котором будет описан метод Check(), принимающий в качестве аргумента указатель базового класса Faculty.Устанавливайте этот указатель на объекты ваших производных классов от Faculty и передавайте его в функцию Check (), которая должна переустановить значение NumberOfEmployees на (NumberOfEmployees – NumberOfEmployees*0.1), а значение EmployeeSalary на (EmployeeSalary + NumberOfEmployees(обновленное) / (NumberOfEmployees(не обновленное)*0.1 * EmployeeSalary) / 2).

3) Изучив информацию о работниках, вы увидели, что у вас в больнице есть сотрудники, которые работают намного больше, чем другие, поэтому вы решили поощрить их и наказать тех, кто работает меньше, путем пересчета зарплат.

Ваша задача: добавить в ранее написанную программу класс Recalculation, в котором будет описан метод Check() принимающий в качестве аргумента указатель

базового класса Employees.Устанавливайте этот указатель на объекты ваших производных классов от Employees и передавайте его в функцию Check (), которая должна переустановить значение HourlyPayment на (HourlyPayment + HourlyPayment *0.1), если WorkedHours>12, а в противном случае переустановить значение HourlyPayment на (HourlyPayment – HourlyPayment *0.1).

4) Несмотря на большие проценты много людей взяли ипотеку в банках вашего начальника, из-за чего в банке заканчиваются деньги. Чтобы не обанкротиться, ваш начальник принял решение уменьшить количество месяцев для выплаты ипотеки для тех, кто берет не очень много денег у банка.

Ваша задача: добавить в ранее написанную программу класс Recalculation, в котором будет описан метод Check(), принимающий в качестве аргумента указатель базового класса Bank. Устанавливайте этот указатель на объекты ваших производных классов от Bank и передавайте его в функцию Check (), которая должна переустановить значение NumberOfMonths на (NumberOfMonths – NumberOfMonths *0.3) если AmountOfDebt <100000.

Контрольные вопросы и задания

1) Что такое наследование и иерархия классов?

2) Какие элементы базового класса видны из производного? Как управлять степенью их защиты?

3) Какое наследование называют множественным?

4) Поясните механизм виртуальных функций.

5) Что такое чисто виртуальная функция?

6) Если указатель базового класса указывает на объект производного от него класса, что нужно сделать, чтобы получить доступ к полям и методам производного класса?

7) Какой порядок вызова имеют конструкторы и деструкторы при наследовании?

Лабораторная работа №7
Шаблоны классов

Цель работы: изучить приемы создания и использования шаблонов классов.

Теоретические сведения

Механизм шаблонов C++ – это средство построения обобщенных определений функций и классов, которые не зависят от используемых типов данных. Этот механизм позволяет сократить трудоемкость создания программ, так как повышает лаконичность текста, т. е. использование шаблонов избавляет от необходимости дублировать коды классов и функций для разных типов данных.

Компилятор по заданному типу аргументов на основе описания шаблона автоматически создает соответствующие экземпляры классов и функций, которые называются *представителями* конкретных классов и функций.

Шаблоны класса в отличие от шаблона функции позволяют параметризовать, т. е. использовать в качестве параметров не только типы элементов данных, но и константы разных типов.

Синтаксис определения шаблона класса следующий:

template <список параметров шаблона> обычное описание структуры класса;

При этом список параметров шаблона не может быть пустым. Элементы в списке разделяются запятыми. Внутри пространства класса параметры шаблона должны быть хотя бы один раз упомянуты.

В список параметров могут входить два вида параметров:

1) типированные параметры, начинающиеся со слов

class Идентификатор,

компилятор при создании экземпляра класса заменит его на конкретный тип данных;

2) нетипированные параметры шаблона:

Стандартный тип числовых данных Идентификатор

Таким образом, типированные параметры – это фиктивные имена типов данных, входящих в класс. Нетипированные параметры – это поименованные типы числовых констант. При этом им при декларировании можно присваивать умалчиваемые значения.

Каждый параметр является локальным в рамках пространства класса.

В описании класса хотя бы один раз необходимо упомянуть ID типированных и нетипированных параметров, конкретные значения для которых будут переданы в момент создания объекта этого класса посредством списка аргументов, который указывается через запятые в треугольных скобках сразу после ID класса:

ID_ класса <список аргументов> ID_объектов',

ID_объектов – идентификаторы объектов, которые создает данный класс (записанные через запятые, например a, b, c*).* При этом в списке аргументов каждому типированному параметру шаблона должен соответствовать известный конкретный тип

данных, а каждому нетипированному параметру – константное выражение соответствующего типа. Таким образом, между списком параметров шаблона и списком аргументов должно быть абсолютное соответствие по количеству, порядку их следования и типам. Если нетипированные параметры имеют умалчиваемые значения, их располагают в списке последними.

Рассмотрим пример с применением шаблонов:

```
#include "pch.h"
#include <iostream>
using namespace std;
template<typename K> //Создаем шаблон, у которого только один параметр – тип
class ArrayElement //Описываем класс для создания объектов будущего массива
{
public:
int a; //Поле типа int
double b; //Поле типа double
K c; //Поле с неизвестным типом данных
};
```

template<typename T,class AE,int MAXSIZE = 20> //Создаем шаблон, у которого первые 2 параметра типированные. Параметр 1 принимает стандартный тип данных, параметр 2 – пользовательский тип данных, параметр 3 – прямое значение (нетипированный параметр). Если параметр 3 не будет указан при создании объекта, то MAXSIZE установится значение = 20.

class Array //Описываем класс для создания хранения массива неопределенного типа и переменной неопределенного типа

```
{
public:
T k = 2.9; //Переменная неопределенного типа
AE *Mass; //Указатель неопределенного типа
Array(); //Конструктор
void Print(); //Метод для вывода массива
};
template<typename T, class AE, int MAXSIZE>
Array<T, AE, MAXSIZE>::Array() //Описываем конструктор
{       Mass = new AE[MAXSIZE]; //Выделяем память
for (int i = 0; i < MAXSIZE; i++) //Заполняем массив
{       Mass[i].a = i;
        Mass[i].b = i;
        Mass[i].c = k; } }
template<typename T, class AE, int MAXSIZE>
void Array<T, AE, MAXSIZE>::Print() //Описываем метод вывода
{       for (int i = 0; i < MAXSIZE; i++)
{       cout << Mass[i].a << endl;
        cout << Mass[i].b << endl;
```

```
        cout << Mass[i].c << endl;
        cout << endl; } }
int main()
{ Array<int,ArrayElement<int>, 15> obj; //Создаем объект obj, в котором поле K с в классе ArrayElement будет типа int и поле T k в классе Array будет типа int, MAXSIZE будет = 15 ;
    obj.Print();
    Array<double, ArrayElement<double> > obj2; //Создаем объект obj, в котором поле K с в классе ArrayElement будет типа double и поле T k в классе Array будет типа double, MAXSIZE будет = 20, так как мы не передали параметр 3 и установилось значение, прописанное в template;
    obj2.Print(); }
```

Упражнение: создайте шаблон класса, порождающего динамические одномерные массивы с элементами различных типов (вещественные, целочисленные, символьные и т. д.). Тип данных и результат являются параметрами по отношению к классу. Программа должна содержать конструктор, деструктор, метод просмотра значений созданного массива, а также метод для решения задачи в соответствии с вариантом.

Задания к лабораторной работе

Уровень 1

Варианты:

1) Вы решили изучить изготовление шаблонов.

Ваша задача: создать шаблон класса, который в качестве параметра принимает тип (int, double или char) для класса Array. В классе должен быть массив (динамический или статический (на ваш выбор)), тип которого передается в шаблон в качестве параметра, и конструктор, в котором происходит заполнение массива необходимым типом данных. Если был передан тип int, то нужно вывести среднее значение всех элементов массива, если тип double, то – сумму всех элементов массива, если тип char, то – значение первого и последнего элементов массива.

2) Вы решили устроиться работать на кафедру вычислительных методов и программирования. После собеседования выяснилось, что вы не знаете, что такое шаблон класса. Вам дали несколько часов на то, чтобы изучить тему и выполнить тестовое задание, для того чтобы вас зачислили на кафедру.

Ваша задача: создать шаблон класса, который в качестве параметра принимает тип (int, double или char) для класса Array. В классе должен быть массив (динамический или статический (на ваш выбор)), тип которого передается в шаблон в качестве параметра, и конструктор, в котором происходит заполнение массива необходимым типом данных. Если был передан тип int, то нужно вывести элементы массива, находящиеся между максимальным и минимальным элементом, если тип double, то – разницу между максимальным и минимальным элементом массива, если тип char, то – код каждого символа в соответствии с ASCII Table.

3) Вы подали заявку на участие в конкурсе "Best Hospital in the world". Для участия в конкурсе вам требуется предоставить количество выздоровевших пациентов, или расход больницы, или счастливый символ больницы. Так как каждый день эта информация была разной, вы решили создать массив для ее хранения с неопределенным типом данных.

Ваша задача: создать шаблон класса, который в качестве параметра принимает тип (int, double или char) для класса Array. В классе должен быть массив (динамический или статический (на ваш выбор)), тип которого передается в шаблон в качестве параметра, и конструктор, в котором происходит заполнение массива необходимым типом данных. Если был передан тип int, то нужно вывести максимальный элемент массива, если тип double, то – минимальный элемент массива, если тип char, то – все элементы массива.

4) Ваш начальник попросил вас написать программу, в которую можно было бы ввести 3 разных по типу массива, а она в зависимости от введенных данных выдала бы результат.

Ваша задача: создать шаблон класса, который в качестве параметра принимает тип (int, double или char) для класса Array. В классе должен быть массив (динамический или статический на (ваш выбор)), тип которого передается в шаблон в качестве параметра, и конструктор, в котором происходит заполнение массива необходимым типом данных. Если был передан тип int, то нужно вывести нечетные элемент массива, если тип double, то – сумму элементов массива, если тип char, то – все элементы массива с конца.

Уровень 2

Упражнение: создайте шаблон класса, порождающего динамические одномерные массивы с элементами различных типов (вещественные, целочисленные, символьные и т. д.). Тип данных и результат являются параметрами по отношению к классу. Программа должна содержать: конструктор, деструктор, метод просмотра значений созданного массива, а также метод для решения задачи в соответствии с вариантом.

Варианты:

1) Ваша задача: создать шаблон класса, который в качестве параметра принимает тип (int, double или char) для класса Array. В классе должен быть массив (динамический или статический (на ваш выбор)), тип которого передается в шаблон в качестве параметра, и конструктор, в котором происходит заполнение массива необходимым типом данных. Если был передан тип int, то нужно вывести среднее значение всех нечетных элементов массива, если тип double, то – отсортировать и вывести массив в порядке возрастания, если тип char, то – вывести значение нечетных по порядку элементов массива.

2) Вы выполнили тестовое задание и вас отправили к заведующему кафедрой, который сказал, что задание слишком простое и решил его немного усложнить.

Ваша задача: создать шаблон класса, который в качестве параметра принимает тип (int, double или char) для класса Array. В классе должен быть массив (динамический или статический (на ваш выбор)), тип которого передается в шаблон в качестве параметра, и конструктор, в котором происходит заполнение массива необходимым типом данных. Если был передан тип int, то нужно вывести все значения элементов массива с числами, поставленными в обратном порядке. То есть при вводе "12 3 4 56 78" должно вывестись "21 3 4 65 87". Если тип double, то нужно вывести среднее значение четных элементов массива, если тип char, то – отсортировать этот массив и вывести его в порядке возрастания.

3) Ваша задача: создать шаблон класса, который в качестве параметра принимает тип (int, double или char) для класса Array. В классе должен быть массив (динамический или статический (на ваш выбор)), тип которого передается в шаблон в качестве параметра, и конструктор, в котором происходит заполнение массива необходимым типом данных. Если был передан тип int, то нужно вывести произведение всех элементов массива, заканчивающихся на 5, деленных на все элементы массива, заканчивающиеся на 3. Если тип double, то нужно округлить все значения и вывести элементы, которые встречаются в массиве более 1 раза, если тип char, то – отсортировать этот массив и вывести его в порядке убывания.

4) Вашему начальнику понравилась ваша программа, но ему необходимо, чтобы программа выдавала немного другой результат.

Ваша задача: создать шаблон класса, который в качестве параметра принимает тип (int, double или char) для класса Array. В классе должен быть массив (динамический или статический (на ваш выбор)), тип которого передается в шаблон в качестве параметра, и конструктор, в котором происходит заполнение массива необходимым типом данных. Если был передан тип int, то нужно прибавить значение текущего элемента к значению следующего, к примеру, при вводе "1 2 3 4 5 6" должно вывестись "1 3 6 10 15 21". Если тип double, то нужно вывести сумму самого большого и самого малого элемента массива, если тип char, то нужно все символы, код которых заканчивается на 5, заменить на пробел (код 32).

Контрольные вопросы и задания

1) В чем в C++ заключается механизм шаблонов классов и функций?
2) Каков общий формат шаблона класса?
3) Перечислите виды параметров шаблона класса.
4) Как создать конкретный экземпляр класса, используя шаблон класса?
5) Как определить метод вне пространства шаблона класса?
6) Чем отличается шаблон класса от шаблона функции?

Лабораторная работа №8
Обработка исключительных ситуаций (C++)

Цель работы: изучить механизмы обработки исключительных ситуаций.

Теоретические сведения

Исключения – возникновение непредвиденных ошибочных условий в момент работы программы. В то же время исключение – это более общее, чем ошибка, понятие, так как может возникать и тогда, когда в программе нет ошибок (например, сбой выделения памяти при создании объекта класса). В общем исключение обозначает особую ситуацию, когда требуется безвозвратно переключить выполнение программы на блок обработки такой ситуации. Выявление особой ситуации производится только программным путем при помощи проверки нормального хода выполнения программы.

Средства обработки ошибочных ситуаций позволяют передать обработку исключений из кода, в котором возникло исключение, некоторому другому программному блоку, который выполнит в данном случае определенные действия. Таким образом, основная идея данного механизма состоит в том, что функция проекта, обнаружившая непредвиденную ошибочную ситуацию, которую она не знает, как решить, генерирует сообщение об этом (бросок исключения). А система вызывает по этому сообщению программный модуль, который перехватит исключение и отреагирует на возникшее нештатное событие. Такой программный модуль называют обработчиком или перехватчиком исключительных ситуаций. И в случае возникновения исключения в его обработчик передается произвольное количество информации с контролем ее типа. Эта информация и является характеристикой возникшей нештатной ситуации.

Обработка исключений в C++ является обработкой с завершением. Это означает, что исключается невозможность возобновления выполнения программы в точке возникновения исключения.

Для обеспечения работы такого механизма были введены следующие ключевые слова:

- *try* – проба испытания;
- *catch* – перехватить (обработать);
- *throw* – бросать.

Кратко рассмотрим их назначение.

Ключевое слово *try* открывает блок кода, в котором может произойти ошибка; это обычный составной оператор: *try* { код };

Код содержит набор операций и операторов, который и будет контролироваться на возникновение ошибки. В него могут входить вызовы функции пользователя, которые компилятор также возьмет на контроль.

Среди данного набора операторов и операций обязательно указывают операцию броска исключения *throw,* которая имеет следующий формат:

throw выражение',

где *выражение* определяет тип информации, которая и описывает исключение (например, конкретные типы данных).

Обработчик исключения *catch* перехватывает информацию:

catch (тип и *параметр)* { код }

Через этот параметр обработчику передаются данные определенного типа, описывающие обрабатываемое исключение. Код определяет те действия, которые необходимо выполнить при возникновении данной конкретной ситуации. В С++ используют несколько форм обработчиков. Рассмотренный обработчик получил название "параметризованный специализированный перехватчик".

Перехватчик должен следовать сразу же после блока контроля, т. е. между обработчиком и блоком контроля не должно быть ни одного оператора. При этом в одном блоке контроля можно вызывать исключения разных типов для различных ситуаций, поэтому обработчиков может быть несколько. В этом случае их необходимо расположить сразу же за контролирующим блоком последовательно друг за другом.

Кроме того, запрещены переходы как извне в обработчик, так и между обработчиками. Можно воспользоваться универсальным или абсолютным обработчиком:

catch (...){ код}

где (...) означают способность данного перехватчика обрабатывать информацию любого типа. Такой обработчик располагают последним в пакете специализированных обработчиков. Тогда, если исключение не будет перехвачено специализированными обработчиками, то будет выполнен последний – универсальный обработчик.

Если не возникнет исключение, набор обработчиков будет обойден, т. е. проигнорирован.

Если же исключение было "брошено" при возникновении критической ситуации, то будет вызван конкретный перехватчик при совпадении его параметра с выражением в операторе броска, т. е. управление будет передано найденному обработчику. После выполнения кода вызванного обработчика управление передается оператору, который расположен за последним перехватчиком, или проект корректно завершает работу.

Существенное отличие вызова конкретного обработчика от вызова обычной функции заключается в следующем: при возникновении исключения и передаче управления определенному обработчику система осуществляет вызов всех деструкторов для всех объектов классов, которые были созданы с момента начала контроля и до возникновения исключительной ситуации с целью их уничтожения.

Блоки *try* как составные блоки могут быть вложены друг в друга. В случае возникновения исключения в некотором текущем блоке поиск обработчика по-

следовательно продолжается в блоках предшествующих уровней вложенности деструкторов с продолжением вызова.

Пример обработки исключительных ситуаций

Функция Divide() возвращает частное от деления чисел, принимаемых в качестве аргументов (a, b). Если делитель равен нулю (b = 0), то возникает исключительная ситуация.

```
#include "pch.h"
#include <iostream>
using namespace std;
double Divide(double, double);
int main()
{
double a, b, result;
cout << " Input a, b: " << endl;
cin >> a >> b;
try
{       result = Divide(a, b);
        cout << " Nonnal Work " << endl;
        cout << " a = " << a << ", b = " << b << endl;
        cout << " Ansver: " << result << endl;     }
catch (double z) {
        cout << " Division by zero... " << endl;
        cout << " b = " << z << endl; } }
double Divide(double a1, double b1) {
if (b1 == 0) throw b1;
return a1 / b1; }
```

, Введем значения 1 и 2 и получим результаты выполнения программы:

```
 Input a, b: 1 2
Normal Work
a=1, b = 2
Answer: 0.5
```

А при вводе значений 1 и 0 получим:

```
Input a, b: 1 0
 Division by zero...
b =0
```

Оператор *throw* сигнализирует об исключительном событии (попытку деления на нуль) и генерирует объект исключительной ситуации. В данном случае он находится внутри функции Divide(). Объект перехватывается обработчиком *catch*. Этот процесс, как уже известно, и называется вызовом исключительной ситуации. В рассмотренном примере исключительная ситуация имеет форму вещественной переменной – делителя.

Упражнение: создайте пользовательский класс, в котором будут поля (название и количество полей будут даны в индивидуальном задании), конструктор для заполнения этих полей и метод, выполняющий индивидуальное задание.

Задания к лабораторной работе

Уровень 1

Варианты:

1) Компания "Baby" попросила вас разработать мини-игру для детей. Правила игры просты: родитель вводит какое-то число, оно высвечивается на экране, затем ребенок должен ввести это число. Так как игра детская, то родитель не может ввести числа меньше нуля и больше 100 (при вводе числа, выходящего за этот диапазон, должно появиться сообщение об ошибке, и родителя должны снова попросить ввести число, и так будет до тех пор, пока родитель не введет число, принадлежащее диапазону 0…100). При вводе ребенком неправильного числа появляется соответствующее сообщение и просьба ввести число заново и так до тех пор, пока число не будет введено правильно.

Ваша задача: создать класс Game, в котором будет поле number, хранящее в себе число, введенное родителем. Поле number заполняется при помощи конструктора с параметром. В классе также должен быть метод Check(), который будет проверять число, введенное ребенком, и в случае, если введено не то число, вызывать ошибку.

2) Вам нужно написать программу, которая проверяет, является ли число простым. В случае если число не простое, должна быть вызвана ошибка.

Ваша задача: создать класс PrimeNumber, в котором будет поле number для хранения будущего простого числа. Поле number заполняется при помощи метода Check(), который проверяет, является ли число простым, и если да, то записывает это число в поле number, а если нет – вызывает ошибку.

3) У продавца в аптеке сломался калькулятор, и теперь ему сложно считать сдачу для клиентов. Необходимо написать программу, которая будет помимо простого счета оповещать продавца в случае, если клиент дал денег меньше чем нужно.

Ваша задача: создать класс Calculator, в котором будут поля Price и MoneyPaid, заполняемые при помощи конструктора. Также в классе должен быть метод Count(), который выведет разность MoneyPaid – Price, в случае, если разность будет положительной. Если разность получилась отрицательной, то Count() должен вызвать ошибку.

4) Начальник попросил вас написать программу, которая будет проверять, сколько каждый сотрудник банка пожертвовал на благотворительность, и в случае, если он пожертвовал слишком мало, будет выдавать ошибку.

Ваша задача: создать класс Vacation, в котором будет поле number. Поле number заполняется при помощи метода Check(), который проверяет, является ли число > 1000, и если да , то записывет это число в поле number, а если нет, то вызывает ошибку.

Уровень 2

Варианты:

1) Компании "Baby" понравилась разработанная вами игра и они попросили вас написать новую игру, которая называется "Угадай число". Правила игры очень просты: компьютер загадывает любое число в диапазоне –50...50. Пользователь вводит число, и если оно не совпадает с загаданным, то вызывается ошибка. Если пользователь ввел число меньше загаданного, то выводится сообщение о том, что введенное число меньше загаданного. Если введенное число больше загаданного, то выводится сообщение о том, что введенное число больше загаданного. Если введенное число выходит за границы диапазона –50…50, то выводится сообщение о том, что данное число не могло быть загадано и диапазон, в котором лежит загаданное число, составляет –50…50.

Ваша задача: создать класс Game, в котором будет поле number, хранящее сгенерированное число. Поле number заполняется при помощи конструктора без параметров, в котором происходит генерация случайного значения. В классе также должен быть метод Check(), который будет проверять число, введенное пользователем, на принадлежность к диапазону –50…50 и на близость к загаданному числу.

2) Ваша задача: создать класс Palindromos, в котором будет поле number для хранения будущего палиндрома. Поле number заполняется при помощи метода Check(), который проверяет, является ли число палиндромом, и если да, то записывает это число в поле number, а если нет, то вызывает ошибку.

3) В честь дня математики у вас проходит акция. Если |MoneyPaid – Price| является простым числом, то ошибка не вызывается. Вам нужно срочно переделать вашу программу под это условие.

Ваша задача: создать класс Calculator, в котором будут поля Price и MoneyPaid. Поля Price и MoneyPaid заполняются при помощи конструктора. Также в классе должен быть метод Count(), который выведет разность MoneyPaid – Price в случае, если разность будет положительной, или выведет |MoneyPaid – Price|, если модуль разности является простым числом. Если разность получилась отрицательной, то Count() должен вызвать ошибку.

4) Ваша задача: создать класс Vacation, в котором будет поле number. Поле number заполняется при помощи метода Check(), который проверяет, является ли число больше 1000 и кратно ли оно 5 и если да , то записывает это число в поле number, а если нет, то вызывает ошибку.

Контрольные вопросы и задания

1) Что называют исключением?

2) Что такое "блок с контролем"?

3) Дайте характеристику обработчику исключений. Какие бывают виды обработчиков?

4) Какие правила налагаются на соотношения между блоком контроля и обработчиками?

5) Чем отличается вызов обработчика от вызова обычной функции?

6) Что делает оператор throw?

Литература

1. Jesse, Russell Объектно-ориентированное программирование / Jesse Russell. – М. : VSD, 2012. – 117 с.
2. Васильев, А. C#. Объектно-ориентированное программирование / А. Васильев. – М. : Питер, 2017. – 320 с.
3. Визуальные средства разработки приложений : учеб.-метод. пособие / В. Н. Комличенко [и др.]. – Минск : БГУИР, 2004. – 68 с.
4. Иванова, Г. С. Объектно-ориентированное программирование / Г. С. Иванова, Т. Н. Ничушкина, Е. К. Пугачев. – М. : МГТУ им. Н. Э. Баумана, 2014. – 368 с.
5. Иванова, Г. С. Объектно-ориентированное программирование / Г. С. Иванова. – М. : МГТУ им. Н. Э. Баумана, 2014. – 149 с.
6. Кьоу, Дж. Объектно-ориентированное программирование / Дж. Кьоу, М. Джеанини. – М. : Питер, 2015. – 240 с.
7. Комлев, Н. Ю. Объектно Ориентированное Программирование. Хорошая книга для Хороших Людей / Н. Ю. Комлев. – М. : СОЛОН-Пресс, 2014. – 297 с.
8. Лафоре, Р. Объектно-ориентированное программирование в C++ / Р. Лафоре. – М. : Мир, 2015. – 928 с.
9. Лесневский, А. С. Объектно-ориентированное программирование для начинающих / А. С. Лесневский. – М. : Бином. Лаборатория знаний, 2005. – 232 с.
10. Луцик, Ю. А. Объектно-ориентированное программирование на языке C++ : учеб. пособие / Ю. А. Луцик, А. М. Ковальчук, И. В. Лукьянова. – Минск : БГУИР, 2003. – 203 с.
11. Хорев, П. Б. Объектно-ориентированное программирование / П. Б. Хорев. – М. : Академия, 2017. – 448 с.
12. Пол, И. Объектно-ориентированное программирование с использованием C++/ И. Пол. – Киев : НПИФ «ДиаСофт», 1995. – 462 с.
13. Павловская, Т. C/C++. Процедурное и объектно-ориентированное программирование : учебник / Т. Павловская. – М. : Питер, 2015. – 496 с.

Св. план 2020, поз. 37

Учебное издание

Макейчик Екатерина Геннадьевна
Чепикова Виолетта Викторовна
Шевчук Оксана Геннадьевна

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *М. А. Зайцева*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *О. И. Толкач*

Подписано в печать 10.09.2020. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 4,77. Уч.-изд л. 5,0. Тираж 60 экз. Заказ 53.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
Ул. П. Бровки, 6, 220013, г. Минск