



*Томский межвузовский центр
дистанционного образования*

С.И. Борисов

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Учебное методическое пособие

ТОМСК – 2002

Министерство образования Российской Федерации

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

**Кафедра компьютерных систем в управлении
и проектировании (КСУП)**

С.И. Борисов

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Учебное методическое пособие

2002

Борисов С.И.

Объектно-ориентированное программирование: Учебное методическое пособие. – Томск: Томский межвузовский центр дистанционного образования, 2002. – 38 с.

© Борисов С.И., 2002

© Томский межвузовский центр
дистанционного образования, 2002

СОДЕРЖАНИЕ

Необходимый инструментарий.....	5
Общие положения	5
MS-DOS	6
Windows	7
Unix.....	8
Заключение	8
Лабораторная работа №1. «Простейшие программы»	9
Цель работы	9
Предварительные сведения.....	9
Задания	13
Комментарии	15
Лабораторная работа №2. «Реализация простейшего класса»	16
Цель работы	16
Предварительные сведения.....	16
Задания	19
Комментарии	21
Контрольная работа №1	27
Цель работы	27
Предварительные сведения.....	27
Задания	27
Лабораторная работа №3. «Наследование».....	29
Цель работы	29
Предварительные сведения.....	29
Задания	29
Комментарии	31
Лабораторная работа №4. «Шаблоны»	33
Цель работы	33
Предварительные сведения.....	33
Задания	35
Контрольная работа №2	38
Цель работы	38
Предварительные сведения.....	38
Задания	38

НЕОБХОДИМЫЙ ИНСТРУМЕНТАРИЙ

Общие положения

Прежде чем приступать к выполнению лабораторных работ, необходимо решить один важный вопрос: «Какой инструментарий понадобится для выполнения работы?» Ответ на данный вопрос неоднозначен, ибо разброс как технического, так и программного обеспечения, удовлетворяющего минимальным требованиям данных лабораторных работ, велик, а предположить, каким именно Вы можете воспользоваться в ваших условиях, затруднительно. Абсолютно точно, что вам необходим компьютер и компилятор C++, точнее сказать, вам понадобится среда программирования. Желательно, чтобы среда объединяла в себе редактор исходных текстов, компилятор с используемого языка и отладчик программ. Кроме того, в среду должен быть включен инструментарий для разработки программ на языке C/C++. В него входят: библиотечные модули, заголовочные файлы, подробная справочная система, набор работающих примеров, ряд инструментальных программ. Как правило, все это вы можете получить в одном дистрибутивном пакете.

На сегодняшний день существует огромное множество систем разработки программного обеспечения на базе языка C++. Часть этих систем бесплатная (свободная), и их можно использовать в любых целях без ограничений. Другая часть – коммерческая, и за их использование необходимо платить деньги фирме-разработчику. Ниже приведены наиболее популярные средства разработки на языке C++ для нескольких наиболее популярных платформ.

Платформой обычно называют связку компьютер + операционная система (ОС). Наиболее вероятно предположить, что у вас есть персональный компьютер типа PC под управлением операционной системы Windows 95, Windows 98, Windows ME, Windows NT 4.0 или Windows 2000. Годится также компьютер, работающий под управлением MS-DOS. Не исключено, что Вы используете на своем компьютере ОС типа Unix или имеете возможность работать с удаленной системой на базе этой ОС. Разделим все множество используемых компьютеров на 3 класса: MS DOS, Windows и Unix. Рассмотрим подробнее данные платформы.

MS-DOS

Для ОС MS-DOS существовало и существует огромное количество компиляторов С и С++. Однако большинство из них в настоящее время являются устаревшими и зачастую не отвечают современному стандарту языка С++. Тем не менее, для изготовления данных лабораторных работ их можно использовать. Перечислим наиболее известные системы программирования на С++.

Borland C++ (рекомендуется использовать версию 3.1, выпущенную в 1991 году). Весьма точно отвечает стандарту того времени, кроме обработки исключительных ситуаций (блоки `try ... catch ... throw`). Не поддерживаются: пространство имен (`namespace`), динамическое приведение типов (`dynamic_cast`), контроль имен времени выполнения (RTTI). Достоинством следует считать удобную многооконную оболочку со встроенным символьным отладчиком. Есть возможность создания приложений для Windows 3.x. Полная установка (включая среду для Windows) занимает 56 MB. Если не ставить среду для Windows (которая сейчас, как правило, уже не нужна), это займет около 12-14 MB. Хорошая online-справка. *Коммерческий.*

Microsoft Quick C/C++. Данная система в основном похожа на предыдущую. Есть небольшие отличия, касающиеся распределения памяти. Нет собственной оболочки, однако есть неплохой экраный отладчик. *Коммерческий.*

Symantec C/C++. (Предпочтительно версии 6.0-7.0.) Удобная многооконная среда разработчика, оконный отладчик. Работа по распределению памяти аналогична компиляторам Microsoft. Не поддерживает обработки исключительных ситуаций и прочие нововведения стандарта языка С++ (аналогично Borland C++ 3.1). *Коммерческий.*

Watcom C/C++. (Рекомендуется версия 9.0-11.0.) Отличается высокой степенью корректности кода. Обладает хорошим интерфейсом для DOS и Windows, а также позволяет компилировать как 16-битные, так и 32-битные программы для DOS, Windows и QNX. Включает огромное количество библиотек. Дистрибутив версии 9.0 занимает более 60 дискет. *Коммерческий.*

GNU C++ for DOS. В настоящее время поддержка и развитие этой системы прекращена. Обладает только инструментами командной строки (отсутствует оболочка). Весьма хорошо соответствует современному стандарту языка С++. Для инсталляции требует примерно 14MB. Автоматически использует специальные расширители DOS благодаря чему генерируются 32-разрядные приложения (`sizeof(int)=4`). Однако такие программы могут не совсем корректно

работать под Windows, особенно под Windows NT 4.0/2000. *Свободный.*

Windows

Если у Вас компьютер под управлением MS-Windows, то вы можете использовать все системы программирования, перечисленные выше для DOS. Однако надо заметить, что некоторые из интегрированных сред программирования, такие как Borland C++ 3.1, не совсем корректно работают в среде виртуальной машины Windows, в связи с чем время от времени происходит неожиданный сбой и сброс оболочки. Поэтому рекомендуется как можно чаще в процессе работы сохранять на диске текущие редактируемые файлы. Кроме сред программирования для DOS, существуют и специальные среды, ориентированные изначально на работу в Windows. Среди них имеются системы, оснащенные полноценной графической интегрированной средой разработки, и есть инструменты командной строки. Среди последних преобладают свободные (не коммерческие) системы. Все эти компиляторы позволяют формировать как оконные Windows приложения, так и консольные приложения. Нас будут интересовать только консольные приложения.

Borland C++ (4.0-5.02, рекомендуется 5.02). Удобная среда разработки, возможность разработки 16-разрядных приложений для DOS, 16- и 32-разрядных приложений для Windows. Хороший оконный отладчик (версии 4.x – только 16-разрядных). Поставляется вместе с библиотеками быстрой разработки приложений для Windows (OWL, MFC) и для DOS (TurboVision). Как и во всех пакетах от Borland, большая и хорошо организованная online справка по операторам, функциям и классам C/C++. Есть некоторые ошибки в компиляторе по работе с шаблонами. *Коммерческий.*

Microsoft Visual C++ (рекомендуются версии 5.0-6.0). Мощная среда разработки, в особенности то, что касается создания развитых приложений для Windows с использованием библиотеки MFC. К недостаткам следует отнести громоздкость среды разработчика, к которой трудно приноровиться на первых порах работы. *Коммерческий.*

Borland C++ Compiler 5.5 (весьма рекомендуется). Предоставляет только инструменты командной строки, однако в настоящее время является наиболее полно соответствующим стандарту языка C++ среди компиляторов для Windows. Генерирует только 32-разрядные приложения. Отсутствие интегрированной среды и экранных отладчиков является некоторым затруднением для освоения языка. *Свободный.*

Unix

Если у вас на компьютере установлена какая-нибудь версия UNIX (например, FreeBSD Unix или Linux любого дистрибутива), то у вас почти наверняка уже есть компилятор C++, причем компилятор достаточно мощный как для выполнения лабораторных работ, так и для создания реальных современных программных систем. Как правило, с современными версиями подобных ОС поставляется компилятор GNU C++. Данный компилятор является полноценным ANSI и ISO стандартным компилятором, то есть соответствует данным стандартам.

Скорее всего, что если вы пользуетесь ОС семейства Unix, то вы уже являетесь весьма подготовленным пользователем, администратором или программистом. Поэтому здесь приведем лишь основное направление, в котором надо вести поиск. Наберите в командной строке команду **cc**, ознакомьтесь с тем, что вам выдаст компилятор (возможно, конечно, что вы не стали ставить компилятор при инсталляции Unix, но это было бы странно). Неплохо также ознакомиться более подробно с опциями командной строки при помощи команды **man cc**, еще вам понадобится программа **make** для удобства создания многомодульных приложений; также рекомендуется ознакомиться с ее возможностями при помощи команды **man make**. Вместе с компилятором языка C++ на Unix ОС, как правило, присутствует руководство-справочник (manual) по языку C/C++. Этот справочник находится в третьем разделе руководства. Например, для получения информации о функции **printf** необходимо ввести следующую команду **man 3 printf**.

При изготовлении лабораторных работ следует учесть, что все подобные компиляторы 32-разрядные, равно как и те ОС, под которыми они работают. Поэтому размер чисел типа **int** будет 32 бита (4 байта) а не 16, как при программировании для DOS. Размер чисел типа **long** тоже, как правило, 32 бита.

Заключение

Разумеется, здесь перечислен далеко не полный перечень всех возможных компиляторов C и C++, которые вы можете использовать для выполнения данных работ.

ЛАБОРАТОРНАЯ РАБОТА №1. «ПРОСТЕЙШИЕ ПРОГРАММЫ»

Цель работы

Целью данной лабораторной работы является реализация простейших программ, изучение работы с компилятором языка C/C++, изучение стандартного ввода/вывода и операций в языке C++.

Предварительные сведения

Любая программа, как правило, осуществляет обработку данных, поэтому должна уметь вводить и выводить информацию, которая в процессе работы программы хранится в оперативной памяти компьютера.

Впервые появившиеся в языке программирования C средства форматированного ввода/вывода – функции **scanf** и **printf** – в настоящее время имеются в различного рода программных системах, например, в Delphi, API Windows, командном интерпретаторе ОС UNIX и т.д. Несколько усложненный формат записи компенсируется исключительной широтой предоставляемых возможностей. В листинге 1 приведена программа, распечатывающая на экране массив целых чисел. Строчки пронумерованы для удобства; в исходном тексте программы Вам не нужно их нумеровать. Выделение жирным шрифтом ключевых слов произведено также для удобства.

Листинг 1. «l1pre.cpp»

```

1  #include <iostream.h>
2  #include <conio.h>

3  #define N 6

4  int a[N]={5,6,3,8,1,4};
5  void main()
6  {
7      int i; // объявляем локальную переменную

8      // выводим на экран массив из N чисел:
9      for (i=0; i<N; i++)
10         cout << a[i] << endl;

11     getch();
12 }
```

В строках 1 и 2 подключаются заголовочные файлы соответственно **stdio.h** и **conio.h**. В заголовочных файлах описаны декларации функции, определены константы и макросы, предназначенные для работы с какой-то библиотекой. В данном случае мы собираемся использовать две библиотеки: библиотека стандартного ввода-вывода (STanDart Input/Output) и библиотека консольного ввода-вывода (CONsole Input/Output). Первая нам необходима для функции **printf** – стандартный вывод на экран, а вторая – для функции **getch** (ввод кода нажатой клавиши с ожиданием нажатия). Оператор **#include**, при помощи которого осуществляется подключение заголовочных файлов, называется директивой препроцессора.

В строке 3 указана еще одна директива препроцессора – директива **#define**. Эта директива в данном случае задает макрос **'N'** и устанавливает ему значение **'6'**. В общем случае в качестве значения макроса может быть указано что угодно – любая последовательность символов. Препроцессор просто заменит последовательность символов **'N'** (имя макроса) на последовательность символов **'6'** (значение макроса). В нашей программе макрос **'N'** используется два раза – в строке 4 и в строке 9. В обоих случаях этот макрос означает количество элементов в массиве. Конечно, и там и там можно было указать непосредственное значение **'6'**. Но если бы нам, например, через день после написания такой программы понадобилось работать с массивом размерностью не 6, а 10, то нам пришлось бы исправлять это в двух местах, что привело бы к увеличению вероятности возникновения ошибок. Вообще, предпочтительно всегда использовать макросы вместо непосредственных значений в тех случаях, когда данное значение используется более, чем в одном месте.

В строке 4 производится объявление глобальной переменной **'a'** как массива, состоящего из **N** целых чисел (**'int'**). О том, что этот массив глобальный, говорит то, что, он объявлен на внешнем уровне – вне процедур и функций. Помимо объявления массива, производится сразу же заполнение его конкретными значениями. Эти значения перечисляются через запятую в фигурных скобках. Здесь в конце строки обязательно ставится точка с запятой – после любого оператора и любого объявления в C++ ставится этот символ, как признак конца оператора (объявления).

В 5-ой строке записано объявление функции **main** – аналогичным образом объявляются все функции на языке C. В данном случае указывается, что данная функция не возвращает никаких значения (**'void'**) и не требует входных параметров (внутри круглых скобок пусто). Функция **main** это особая функция в языке C – именно с этой функции на-

чинается выполнение программы, то есть данная функция является точкой входа в программу.

В строке 7 записана открывающая операторная скобка – это в данном случае признак начала определения тела функции. Вообще операторные скобки ставятся везде, где необходимо указать несколько последовательных операторов, а по синтаксису можно поставить лишь один оператор. Но при определении функции (даже если она состоит из одного оператора) указывать операторные скобки необходимо.

В строке 8 определяется локальная переменная **'i'** целого типа. В языке С все определения локальных переменных должны быть в начале операторного блока ограниченного операторными скобками. В языке С++ такого ограничения нет – переменные можно определять в любом месте в программе. Время жизни и время видимости локальных переменных ограничено блоком, в котором они определены. То есть, во-первых, по выходу из функции **main** все локальные переменные, созданные в ней, будут уничтожены, а во-вторых, получить доступ к этим переменным, то есть производить какие-то манипуляции с ними можно только внутри этой функции.

В строке 9 описывается цикл **'for'**. В данном цикле производится задание начального значения для переменной цикла **'i'**. Затем указывается выражение продолжения цикла – цикл будет выполняться, пока результат этого выражения – «истинно», то есть отлично от нуля. В данном случае цикл будет выполняться пока значение переменной **'i'** будет меньше нуля. В конце каждой итерации цикла будет осуществляться инкремент переменной **'i'**: **'i++'**. В нашей программе тело цикла выполнится ровно **N** (в нашем случае шесть) раз. При этом переменная **'i'**, будет принимать значения от 0 до 5 включительно. Когда переменная **'i'** достигнет значения 6, выражение, указанное во втором поле цикла **'for'**, примет значение 0, что с точки зрения языка Си, означает 'ложь' и соответственно прекратится выполнение цикла.

Строка 10 является телом цикла. Тело цикла – это такой оператор, который выполняется в каждой итерации цикла. В данном случае в качестве такого оператора выступает вывод на экран посредством точного вывода: **'cout << a[i] << endl'**. Эта последовательность операций выводит на экран значение *i*-го элемента массива **'a'** и переводит курсор в начало следующей строки. Объект **'cout'** является стандартным устройством вывода информации и, по умолчанию, соответствует экрану компьютера или текстовой консоли. Для того чтобы ввести что-либо с клавиатуры, необходимо сделать, например, следующую операцию: **cin >> v**, где **v** – это переменная, значение которой необходимо ввести с клавиатуры.

В строке 11 производится вызов функции `'getch'`. Эта функция, описанная в заголовочном файле `'conio.h'`, ожидает нажатия клавиши и, вообще говоря, возвращает код нажатой клавиши, который мы игнорируем, так как ничему не присваиваем значение, возвращаемое этой функцией.

В строке 12 стоит символ `'}'` (фигурная скобка) – это закрывающая операторная скобка, которая соответствует открывающей, находящейся в строке 7.

Кажется, с первой программой разобрались, теперь надо попробовать ее скомпилировать и запустить.

Наберите эту программу в любом текстовом редакторе, например, это может быть оболочка FAR или notepad или любой другой текстовый редактор, который сохраняет файлы в простом текстовом виде (Word в данном отношении не подойдет). Важно, чтобы расширение файла было `'сpp'` (или `'с++'` для Unix).

Теперь необходимо подать этот файл на вход вашего компилятора C++. В зависимости от того, какой системой Вы пользуетесь, это будет делаться по-разному. Например, в оболочке Borland C++ 5.02 для того, чтобы скомпилировать программу, необходимо нажать комбинацию клавиш Alt+F9 или выбрать пункт `'Compile'` из меню `'Project'`. Аналогичный пункт меню есть и в оболочке Visual Studio, в рамках которой работает Visual C++. Если вы пользуетесь компиляторами командной строки, то вам необходимо набрать в командной строке, например, такую команду: `'сc llpre.cpp'`, где `'сc'` – это название запускаемой программы вашего компилятора. Будем считать, что этот этап Вы прошли успешно, хотя здесь ожидает немало подводных камней – могут быть не настроены пути к заголовочным файлам и файлам библиотек, может быть не настроен путь к компилятору. И масса других мелочей. Кроме того, почти все интегрированные оболочки требуют перед созданием любой программы создать `'Project'` (Borland C++) или `'WorkSheet'` (Visual Studio), в который включить написанный Вами `сpp`-модуль. (Необходимо изучать инструкцию к имеющемуся у Вас компилятору.)

В результате работы компилятора у Вас появится выполняемая программа, имеющая (для MS-DOS и Windows) расширение `*.exe`. Запустите эту программу (если Вы работаете с интегрированной оболочкой, то Вы можете запустить программу непосредственно из оболочки). Посмотрите результаты работы этой программы. Она должна выдать Вам на экран содержимое массива **a**, причем каждое число на новой строке. Теперь вы должны быть готовы написать свою первую программу самостоятельно.

Задания

Номер задания вычисляется исходя из последних двух цифр Вашего кода доступа. Например, если последние две цифры равны 34, то номер варианта равен: $n=34 \cdot 10/100=3,4 = 3$ вариант.

В случае, если непонятна формулировка вопроса, выполняйте задание так, как Вы его понимаете, но предварительно объясните, как Вы поняли задание, то есть сформулируйте Ваше понимание данного задания.

Вариант	Номер задания	Задание
1	1.1	Найти минимальное число в массиве.
	1.2	Произвести сложение двух массивов 5×3 по правилам сложения матриц.
	1.3	Найти разницу в днях между двумя заданными датами.
2	2.1	Найти максимальное число в массиве.
	2.2	Создать двумерный массив 5×3 элемента, найти в этом массиве сумму всех элементов.
	2.3	Определить, является ли заданная квадратная матрица симметричной относительно главной диагонали.
3	3.1	Найти сумму чисел в массиве.
	3.2	Отсортировать массив по убыванию.
	3.3	Произвести перемножение двух массивов 5×3 и 3×5 по правилам умножения матриц.
4	4.1	Найти произведение чисел в массиве.
	4.2	Отсортировать массив по возрастанию.
	4.3	Найти определитель матрицы 3×3 по определению определителя.
5	5.1	Распечатать массив в обратном порядке.
	5.2	Заполнить массив числами Фибоначчи (от $n=1$ до $n=20$). Числа Фибоначчи: $f_n = f_{n-1} + f_{n-2}$, $f_1=1$, $f_2=1$.
	5.3	Произвести сложение двух одномерных массивов целых чисел, в каждом элементе которого записана одна десятичная цифра.

6	6.1	Заполнить массив значениями квадрата индекса.
	6.2	Заполнить массив из 20 вещественных чисел значениями факториала: $n!$, где n – индекс массива, $!$ – операция факториал: $n! = n \cdot (n-1)!$, $1! = 1$.
	6.3	Произвести вычитание двух одномерных массивов целых чисел, в каждом элементе которого записана одна десятичная цифра.
7	7.1	Найти среднее арифметическое чисел массива.
	7.2	Заполнить массив вещественных чисел случайными числами в диапазоне от 10 до 11. Произвести нормировку массива вещественных чисел: $a_i = \frac{a_i}{A}$, где A – корень квадратный из суммы квадратов значений массива: $A = \sqrt{\sum_{i=0}^n a_i^2}$.
	7.3	Произвести операцию скалярного умножения двух векторов.
8	8.1	Домножить каждый элемент массива на заданное число.
	8.2	Написать программу, которая определяет, является ли первый заданный вектор вещественных чисел обратной записью второго аналогичного вектора.
	8.3	Вычислить векторное произведение двух векторов.
9	9.1	Добавить к каждому элементу массива заданное число.
	9.2	По заданной квадратной матрице найти вектор, в котором записаны максимумы всех строк матрицы.
	9.3	Найти вектор-сумму нескольких заданных векторов.

10	10.1	Вычестъ из каждого элемента массива значение среднего арифметического.
	10.2	По заданной квадратной матрице найти вектор, в котором записаны минимумы всех столбцов матрицы.
	10.3	Назовем седловой точкой матрицы ячейку, которая является минимальной в своей строке и максимальной в столбце. Найти хотя бы одну седловую точку матрицы размером 10x10, если она есть.

Комментарии

При вычислении разницы дат возникают следующие вопросы:
 1) необходимо каким-нибудь образом ввести эти даты. Самый простой путь - задать эти даты в виде двух массивов на 3 элемента с заранее присвоенными значениями аналогично тому, как это сделано в примере программы. Можно ввести элементы этих массивов с клавиатуры, например, следующим образом: **cin >> a[i]**.

Аналогично можно задавать любой произвольный массив чисел, как целых, так и вещественных. Можно использовать для формирования массивов чисел генератор случайных чисел. В нижеследующем примере массив вещественных чисел **a** заполняется числами из диапазона от 0 до 1,0, не включая.

```

1  #include <stdlib.h>
2  #define N 10
3  double a[N];
4  void main()
5  {
6      randomize();
7      for (int i=0; i<N; i++)
8          a[i] = random(1000)/1000.0;
9  }
```

ЛАБОРАТОРНАЯ РАБОТА №2. «РЕАЛИЗАЦИЯ ПРОСТЕЙШЕГО КЛАССА»

Цель работы

Целью данной лабораторной работы является реализация простейшего абстрактного класса на языке C++.

Предварительные сведения

Язык C++ является объектно-ориентированным языком программирования и позволяет использовать такие преимущества объектно-ориентированной парадигмы, как инкапсуляция, полиморфизм и наследование. Более подробно с этими свойствами ООП Вы можете познакомиться в курсе лекций. В данной лабораторной работе мы более подробно познакомимся с первым из этих свойств ООП. Для примера приведем декларацию класса **Digit** – длинное целое (например, на 20 десятичных знаков).

```
1  typedef unsigned int uint;
2  class Digit {
3  public:
4      Digit(const char *str);
5      Digit(int i);
6      Digit add(const Digit &s);
7      void divide(Digit denom, Digit &quot,
Digit &rem, bool RemDesired)const;
8      int compare(const Digit &y)const;
9  private:
10     char P[20];
11 };
```

Как правило, декларации класса записываются в отдельные заголовочные файлы. Все заголовочные файлы являются обычными текстовыми файлами на языке C, принято давать таким файлам расширение ‘h’ или ‘hpp’ (на сленге, используемом программистами на C/C++, такие файлы называют «хидера», что является фонетической транскрипцией от английского ‘header’). Хотя это, конечно, не догма, и подобным файлам можно давать любые имена и расширения. Однако, если вы хотите, чтобы вас поняли другие программисты, с которыми вам наверняка придется совместно работать над реализацией каких-то

проектов, необходимо придерживаться определенных стандартов. В том числе и стандартов по именованию файлов.

Кроме того, существуют общие правила или стандарты на некоторое оформление заголовочных файлов. Например, это можно сделать так:

```

1 // файл digit.h
2 #ifndef _DIGIT_H
3 #define _DIGIT_H
... // Здесь располагается декларация класса,
... // функций и глобальных переменных,
... // а также макросов и простых констант, которые
... // будут использоваться при работе с данным классом.
4 #endif // _DIGIT_H

```

В строках 2 и 4 при помощи директив условной компиляции **#ifndef** и **#endif** организуется проверка на предмет того, не был ли данный файл уже скомпилирован ранее. Это чрезвычайно актуально для больших проектов, количество которых исчисляется десятком файлов. Рассмотрим следующую ситуацию:

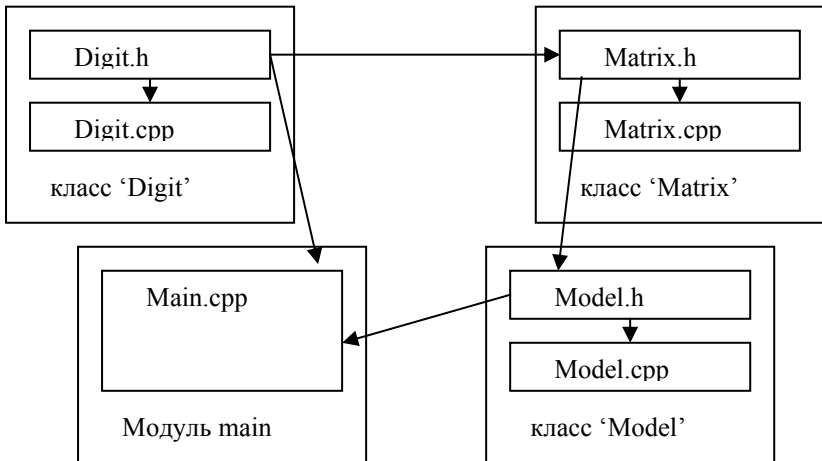


Рисунок 1. Пример схемы использования модулей в программе

Стрелками на рисунке обозначено включение какого-то файла в другой при помощи директивы **#include**. Будем понимать под «модулем» связку из заголовочного файла и «cpp» файла. В заголовочном

файле обычно содержатся декларации класса, а в 'сpp' – дефиниция (реализация) методов данного класса. Поэтому в 'сpp'-файл, как правило, включается заголовочный файл данного модуля.

На схеме, изображенной на рисунке 1, показана схема некоторого проекта программы моделирования. Декларация класса Vector используется при декларации класса Matrix и в модуле main. Класс Matrix используется при декларации класса Model. В свою очередь, класс Model используется главным модулем. Заметим, что в главном модуле будут включаться заголовочные файлы Digit.h и Model.h. В файл Model.h включается файл Matrix.h, а тот, в свою очередь, включает файл Digit.h. Таким образом, файл Vector.h включается в главный модуль дважды. Такое двойное включение имеет две отрицательные стороны: во-первых, в связи с тем, что компилятору придется дважды компилировать один и тот же текст, он (компилятор) будет работать медленнее. Вторая причина более существенна – если в данном заголовочном файле находится декларация класса, то компилятор выдаст сообщение о невозможности повторной декларации класса.

Этого можно избежать, если придерживаться вышеприведенного стандарта оформления заголовочных файлов. При компиляции модуля main первым будет включен файл Digit.h, при этом в списке макросов препроцессора появится макрос с именем **DIGIT_H**. Далее при включении файлов Model.h, Matrix.h и (повторно) Vector.h препроцессор вырежет из входного текста компилятора весь кусок кода от **#ifndef** до **#endif**, так как макрос **DIGIT_H** уже определен.

Разумеется, при оформлении заголовочного файла необходимо указать некоторые дополнительные сведения, такие как имя автора и дата разработки данного класса, информация о вносимых после основной разработки исправлениях, предназначение и/или абстрактный смысл класса, а также некоторую другую, вспомогательную информацию, которая поможет программисту, который будет использовать данный класс (возможно, Вам). Например, таким образом:

```

1 // файл digit.h
2 // декларация класса Digit – неограниченно длинное целое
3 // Определены все основные математические операции
4 // над целыми числами
5 // Дефиниция – файл digit.cpp
6 // (с) С.И. Борисов, январь 2000г.
7 // Создан в процессе реализации системы моделирования
8 //
9 // поддерживаемые операции и методы:
10 // +, -, *, /, %, |, &, ^, ~,
```

```

11 // +=, -=, *=, /=, %=, |=, &=, ^=
12 // ==, !=, <, >, <=, >=,
13 // abs,...
14 // изменения:
15 // 14 апрель 2000г. Добавлена функция поиска наименьшего
16 // общего кратного:
17 // Digit NOK(Digit&, Digit&); - делит каждый компонент
18 // вектора на его длину.
19 // 7 сентябрь 2000г., исправлена ошибка в методе %
20 //

```

Задания

В случае, если непонятна формулировка вопроса, выполняйте задание так, как Вы его понимаете, но предварительно объясните, как Вы поняли задание, то есть сформулируйте Ваше понимание данного задания.

Вариант	Задание
1	Разработать класс Vector – геометрический вектор произвольной размерности (размерность задается в конструкторе вектора). Реализовать метод доступа к элементам вектора. Реализовать операции сложения, вычитания и скалярного произведения векторов, а также нахождение модуля вектора.
2	Разработать класс Matrix – матрица. Размерность матрицы задавать в конструкторе. Реализовать метод доступа к элементам массива. Реализовать операции сложения, вычитания, умножения и транспонирования матрицы. Сделать метод определения симметричности матрицы (если матрица квадратная). Сделать метод формирования единичной матрицы (для квадратных матриц).
3	Разработать класс ArrayOfInt – массив целых чисел. Размерность массива задавать в конструкторе. Реализовать метод доступа к элементам массива. Реализовать метод Sum – вычисление суммы чисел в массиве. Метод сортировки массива по возрастанию и по убыванию, метод сравнения двух массивов (==, !=). Метод конкатенации (слияния) двух массивов.

4	<p>Разработать класс FileStream, инкапсулирующий работу с файлами через стандартную библиотеку Си (fopen, fclose, fprintf, fscanf, fread, fwrite и т.д.). Реализовать методы открытия и закрытия файла (отдельный метод Open и метод Create и конструктор с именем открываемого файла, закрывать – в деструкторе и отдельным методом Close) Реализовать методы Write и Read для int, double и char*. Обеспечить два режима записи – двоичный и текстовый. В текстовом режиме все числа записываются в виде текста, например, целое число 3987 записывается как последовательность символов '3987 ' (преобразование можно сделать при помощи fprintf, например), а в двоичном – в виде последовательности двух байт: 0x93, 0x0f (при помощи fwrite).</p>
5	<p>Разработать класс large – длинное целое. Для хранения одной десятичной цифры использовать одно число типа char. Количество десятичных цифр, которые должны размещаться в этом числе, передавать в конструкторе. Реализовать методы присваивания, сложения двух чисел, печати числа на экране.</p> <p>Переписать программу вычисления числа Фибоначчи для типа large. Вычислить $f_n=100$ (22 десятичных цифры). Замечание: если Вы в прошлый раз использовали рекурсивный алгоритм вычисления чисел Фибоначчи, то теперь Вам необходимо разработать нерекурсивный алгоритм, так как рекурсивное вычисление для $n=100$ займет слишком много времени (возможно, несколько миллионов лет).</p>
6	<p>Разработать класс String – строка символов. Реализовать операции присваивания, конкатенацию (слияние двух строк), сравнения строк (==, !=), метод вывода объекта на экран. Размер строки динамически увеличивается в процессе работы со строкой по мере необходимости.</p>
7	<p>Разработать класс Vector4 и Matrix4 – 4-компонентный вектор и матрица 4x4 соответственно. Реализовать операции сложения матриц, умножения матриц, сложения векторов, умножения матрицы на вектор и вектора на матрицу. Домножение матрицы и вектора на число, нормализация вектора (при данной нормализации все 4 составляющие вектора делятся на значение последней – четвертой составляющей). Отображение матрицы и вектора на экране.</p>

8	Реализовать класс Date (дата) – инкапсулирует внутри данные для работы с датой. Реализовать методы ввода и вывода этой информации. Реализовать операцию вычисления разности между двумя датами (результат в днях).
9	Реализовать класс Time (время) – инкапсулирует внутри данные для хранения времени (часы, минуты, секунды). Реализовать методы ввода и вывода этой информации. Реализовать операцию вычисления разности между двумя точками времени (результат в секундах).
10	Разработать класс Complex – комплексные числа. Реализовать операции: сложения, умножения комплексных чисел, умножения реальных чисел на комплексные и наоборот, нахождения реальной и мнимой части числа, перевода в полярные координаты.

Комментарии

Комментарий 1. Если размерность вектора/матрицы задается в конструкторе, то необходимо использовать динамическое выделение памяти. То есть в конструкторе либо в методе задания размера (SetLen, например) необходимо выделять память, например, таким образом:

```

1 void Digit::SetLen(int new_len)
2 {
3     char *tmp = new char [new_len];
4     if (value){
5         for (int i=0; i<min(len,new_len); i++)
6             tmp[i] = value[i];
7         delete [len]value;
8     }
9     value = tmp;
10    len = new_len;
11 }
```

В строке 3 выделяется место под новый массив (нового размера), в строках 4-8 происходит переприсваивание нового вектора в старый, причем в строках 5-6 старый массив копируется в новый. Обратите внимание, что количество копируемых элементов выбирается как минимальное между размером старого и нового векторов. Только после этого удаляется (строка 7) старый вектор вещественных чисел. Имея такой метод **SetLen**, можно написать следующий конструктор:

```

1  Digit::Digit(int size)
2  {
3      value = NULL;
4      len = 0;
5      SetLen(size);
6  }

```

В строках 3 и 4 обнуляются внутренние данные класса – делать так считается хорошим стилем. Тогда при вызове функции **SetLen** гарантировано, что данные члены будут нулевыми. Это является принципиальным для функции **SetLen**. Дело в том, что данная функция выделяет память под новый массив в случае, если **value == 0**. Если же это значение отлично от нуля, то сперва освобождается память от предыдущего содержимого, то есть считается, что объект уже существовал, что в корне не верно для конструктора. Конструктор потому и называется конструктором, что он создает новый объект, то есть вызывается для вновь создаваемого объекта.

В случае, если в процессе работы необходимо изменить – увеличить или уменьшить – размер массива (например, такая потребность может возникнуть при реализации операции конкатенации), то мы просто вызываем метод **SetLen**, передавая ему количество элементов в новом массиве.

Комментарий 2. Во всех классах возможны ситуации, когда часть методов должна реализовываться как члены-методы класса, а часть – как дружественные функции. Рассмотрим следующую ситуацию: пусть в классе **Digit** необходимо реализовать метод сложения. В простейшем случае можно реализовать лишь один метод сложения в следующем виде:

```

1  class Digit {
2      ...
3      Digit operator+(const Digit&);
4      ...
5  };

```

Данный оператор способен производить операцию сложения над двумя числами типа **Digit**. Предположим, что у нас возникает ситуация, когда необходимо домножить число типа **Digit** на обычное целое число. При наличии приведенного выше оператора и (обязательно!) конструктора от целого числа это можно сделать в следующем виде:

```

1  Digit a,b(31946);
2  a = b*Digit(35);

```

Однако здесь есть один подводный камень — в нашем классе **Digit** параметр конструктора от целого числа имеет смысл максимального количества десятичных цифр, которые мы можем в данном числе разместить. Стало быть, мы не можем воспользоваться данным методом. Здесь есть два пути решения.

Пусть конструктор от целого числа задает количество десятичных цифр в числе и есть оператор присваивания, который в качестве параметра принимает целое число. Тогда можно сделать так:

```

1  class Digit {
2  ...
3      Digit& operator=(int s) {
4          ConvertFromInteger(s);
5          return *this;
6      }
7      Digit operator*(const Digit& s);
8  };
9  ...
10 Digit a,b,c;
11 b = 31946;
12 c = 35;
13 a = b*c;

```

Обратите внимание, что оператор присваивания возвращает ссылку на сам объект. По парадигме, принятой в языке С и соответственно наследованной в языке С++, любой оператор, в том числе и оператор присваивания, возвращает какое-то значение, которое потом может использоваться при вычислении дальнейшего выражения. Метод **ConvertFromInteger** — некий внутренний метод, который переводит из целого числа в тип **Digit**. Однако делать так не совсем удачно с точки зрения удобства использования данного класса (см. строки 11, 12).

Другим вариантом можно предложить набор из трех перегруженных операций умножения:

```

1  class Digit {
2  ...
3      friend Digit operator*(Digit&, Digit&);
4      friend Digit operator*(Digit&, double);
5      friend Digit operator*(double, Digit&);
6  };

```

```

7   ...
8   Digit operator*(Digit& a, Digit& b)
9   {
10  Digit tmp;
11  ... // в tmp помещаем результат умножения a на b
12  return tmp;
13  }
14  Digit operator*(Digit& a, double b)
15  {
16  Digit t;
17  t.ConvertFromDouble(b);
18  return a*t;
19  }
20  Digit operator*(double a, Digit& b)
21  {
22  Digit t;
23  t.ConvertFromDouble(a);
24  return t*b;
25  }
26  ...

```

Две первых операции умножения можно было реализовать как внутренний член класса, а последнюю – только как свободную функцию. Но для единообразия они все сделаны как свободные дружественные функции. При реализации, например, класса матрицы, для которой умножение матрицы на число и умножение матрицы на матрицу есть принципиально разные операции, придется реализовывать все эти 3 операции.

Обратите внимание, что в качестве входного параметра выступает ссылка на объект, а в качестве возвращаемого значения – сам объект Digit. Передача ссылки в качестве параметра значительно экономнее по накладным расходам, чем передача объекта. При этом необходимо указывать компилятору, что данный параметр константный. Это необходимо для создания следующих конструкций:

```

1   Digit a, b, c, d;
2   a = b*c*d;

```

Данную последовательность можно записать по-другому:

```

1   a = operator+(operator+(b,c),d);

```


То есть в первую очередь вызывается метод сложения, в который передаются ссылки на объекты **b** и **c**. В результате работы этого метода в памяти создается новый объект (посредством конструктора копирования). Данный объект считается константным. И ссылка на этот объект передается вновь в метод сложения. Опять создается временный константный объект, ссылка на который передается в качестве параметра в метод присваивания. После обработки выражения компилятор заботится об удалении всех созданных временных объектов.

Комментарий 3. При разработке классов особое внимание следует уделить разработке конструкторов. Вообще, как правило, реализуется несколько конструкторов, с максимумом возможностей для пользователя. Конструкторов делается тем больше, чем сложнее структура данных, которую данный класс инкапсулирует. Но есть определенный минимум конструкторов, которые необходимо всегда или почти всегда реализовывать – это, например, конструктор копирования. Приведем список рекомендуемых конструкторов для класса **Digit**, описанного выше. В отличие от примера данного в разделе предварительных сведений, теперь будем считать, что количество десятичных цифр в числе не фиксировано, а должно задаваться в конструкторе:

```

1  class Digit {
2  ...
3  public:
4      Digit(char* s, int n=0);
5      Digit(long a=0L, int n=0);
6      Digit(Digit& d);
7      ...
8  };

```

В строке 4 записан конструктор, преобразующий строку с записью числа во внутренний формат. При этом вторым параметром указывается максимальная размерность (количество десятичных цифр). Предполагается, что если указан «0», то количество цифр подбирается автоматически по длине входной строки. Именно это значение и указывается как значение по умолчанию, это сделано для удобства программиста, который будет использовать данный класс – он может не указывать требуемую размерность.

В строке 5 записан конструктор, в который в качестве первого параметра передается начальное значение числа в виде длинного целого. Вторым параметром опять же следует необязательный параметр – количество десятичных цифр.

В строке 5 записан конструктор копирования. Это очень важный конструктор, который является обязательным, если Вы внутри класса работаете с динамически создаваемыми данными. Данный конструктор, в принципе, похож на оператор присваивания, но конструктор копирования вызывается при создании нового объекта, а оператор присваивания вызывается для существующего объекта. (Смотрите также предыдущий комментарий.)

Комментарий 4. Информацию по библиотечным функциям, таким, например, как `open`, `fclose`, ... и так далее, можно найти в справочной системе той системы программирования, которую Вы используете.

Комментарий 5. В классе «large» сложение необходимо производить аналогично тому, как Вы это делаете при сложении чисел в столбик (справа налево с переносом в старший разряд).

Комментарий 6. Различают статическое и динамическое выделение памяти. В первом случае память выделяется в момент запуска программы и освобождается после завершения. При этом размер выделенной памяти остается неизменным. Во втором случае память выделяется тогда, когда это необходимо, и освобождается сразу же по завершении работы с данным участком памяти. При этом размер выделенной памяти может быть произвольным и даже может изменяться в процессе работы. Этот механизм предоставляет программисту более гибкие средства для работы с оперативной памятью компьютера. В C++ существует несколько способов динамического распределения памяти. Однако наиболее удобным является использование операторов ***new*** и ***delete***, а в объектно-ориентированном программировании этим операторам нет альтернативы. Более подробно о работе с этими операторами смотрите в курсе лекций.

КОНТРОЛЬНАЯ РАБОТА №1

Цель работы

Целью данной контрольной работы является контроль знаний и умений, полученных в процессе изучения предыдущего материала.

Предварительные сведения

В предыдущих лабораторных работах внутри классов Вы использовали только встроенные в компилятор типы данных. Данная работа будет отличаться лишь тем, что, помимо стандартных типов данных, Вы будете использовать еще и созданные Вами в предыдущей лабораторной работе классы.

Задания

В случае, если непонятна формулировка вопроса, выполняйте задание так, как Вы его понимаете, но предварительно объясните, как Вы поняли задание, то есть сформулируйте Ваше понимание данного задания.

Вариант	Задание
1	На основе класса <code>Vector</code> разработать класс <code>Matrix</code> (матрица, физически представляющая собой вектор, состоящий из заданного числа векторов). Реализовать метод сравнения двух матриц (<code>==</code> , <code>!=</code>).
2	Разработать класс <code>AggrOfMatrix</code> – массив матриц. Размерность массива и матриц, входящих в него, задавать в конструкторе. Реализовать метод доступа к элементам массива, метод сравнения двух массивов (<code>==</code> , <code>!=</code>). Реализовать операции конкатенации массивов.
3	Разработать класс <code>SqAggrOfInt</code> – двумерный массив целых чисел. Физически данный массив должен состоять из вектора массивов типа <code>AggrOfInt</code> . Размерность массива задавать в конструкторе. Реализовать метод доступа к элементам массива. Реализовать метод <code>Sum</code> – вычисление суммы чисел в массиве. Метод сортировки массива по возрастанию и по убыванию, метод сравнения двух массивов (<code>==</code> , <code>!=</code>). Метод конкатенации (слияния) двух массивов.
4	Разработать класс <code>SortedFile</code> – сортированный файл, в данном файле хранятся строки одинаковой длины в порядке возрастания. Имя файла задается в конструкторе. Реализовать метод слияния двух файлов (результатирующий файл должен остаться сортированным).

5	Разработать класс NaturalFraction (обыкновенная дробь) два числа типа large (числитель и знаменатель). Реализовать методы сложения, умножения и деления дробей. Метод вывода на экран.
6	Разработать класс Strings – массив строк символов. Реализовать операции доступа к строкам (оператор «квадратная скобка»), присваивания, конкатенации (слияния двух массивов).
7	Разработать класс ArrayOfVector4 – массив 4-компонентных векторов. Размер массива не ограничен и изменяется динамически в зависимости от потребностей. Реализовать операции доступа к элементам массива, конкатенации, добавления к массиву элемента. Метод-итератор – в данный метод передается функция, которую необходимо выполнить для каждого элемента массива.
8	Реализовать класс Person (персона) – инкапсулирует персональные данные о человеке (фамилию, имя, отчество и дату рождения типа Date). Для этого класса реализовать конструктор, позволяющий записать данные в объект и метод вывода персоны на экран. Реализовать класс Persons – массив персон. Реализовать операции доступа к элементам массива, конкатенации, добавления к массиву элемента. Метод, выдающий список лиц, до дня рождения которых осталось менее 5 дней (возможно придется модифицировать класс Date).
9	Реализовать класс Event (событие) инкапсулирует время прихода события (типа Time) и значение события (вещественное число). Реализовать класс Events – последовательность событий. Размер массива задается в конструкторе и автоматически изменяется в случае необходимости. Реализовать методы сортировки по времени и по значению события (два метода). Метод поиска среднего значения между двумя точками времени. Метод поиска среднего времени между двумя последовательными событиями.
10	Разработать класс ComplexMatrix – матрица комплексных чисел. Размер матрицы задается в конструкторе. Реализовать операции: сложения, умножения матриц, умножения матрицы на число (комплексное).

ЛАБОРАТОРНАЯ РАБОТА №3. «НАСЛЕДОВАНИЕ»

Цель работы

Целью данной лабораторной работы является изучение механизма наследования.

Предварительные сведения

Понятие наследования подробно рассмотрено в курсе лекций.

Задания

В случае, если непонятна формулировка вопроса, выполняйте задание так, как Вы его понимаете, но предварительно объясните, как Вы поняли задание, то есть сформулируйте Ваше понимание данного задания.

Вариант	Задание
1	Разработать класс <code>NormVector</code> – вектор единичной длины, наследованный от класса <code>Vector</code> . Замечание: нормирование необходимо производить после любого изменения составляющих вектора. Соответственно необходимо перегрузить метод доступа к компонентам вектора.
2	Разработать класс <code>NormMatrix</code> (матрица с единичным определителем), наследованный от класса <code>Matrix</code> . Замечание: для упрощения пусть матрица будет размерностью до 3×3 . Нормирование матрицы необходимо производить после любого изменения составляющих. Соответственно необходимо перегрузить метод доступа к ее компонентам.
3	Разработать класс <code>SortedArrayOfInt</code> – упорядоченный массив целых чисел, наследованный от класса <code>ArrayOfInt</code> . Замечание: сортировку массива необходимо производить после любого изменения его составляющих. Соответственно необходимо перегрузить метод доступа к его компонентам.

4	Разработать класс EncryptedFile – зашифрованный файл, наследованный от файла FileStream. В конструктор данного класса передавать ключ – последовательность из нескольких символов, все операции чтения и записи производить через виртуальную процедуру шифрования. Об алгоритме шифрования читайте в комментариях. Все остальные методы сохраняются аналогично классу FileStream.
5	Разработать класс fixpoint (число с фиксированной точкой), наследованный от класса large. В конструктор класса передавать количество цифр до запятой и количество цифр после запятой – их сумма как раз и будет количеством цифр в числе large. Операция сложения остается неизменной. Изменится вывод числа на экран.
6	Разработать класс LowString (строка из прописных символов), наследованный от класса String. Замечание: замену больших букв на маленькие необходимо производить при любом изменении строки.
7	Разработать класс NormedVector4 – 4-компонентный нормализованный вектор. Нормализацию вектора необходимо производить при каждом изменении составляющих вектора. Обеспечить методы «поворот», «смещение», «масштабирование» (см. комментарии).
8	Реализовать класс ValidDate (корректная дата), наследованный от класса Date. При каждом изменении даты необходимо производить контроль корректности даты. Реализовать метод вывода печати в различных форматах.
9	Реализовать класс ValidTime – корректное время. При каждом изменении времени необходимо производить контроль корректности времени. Реализовать методы форматного вывода этой информации.
10	Разработать класс PolarComplex (комплексные числа, представленные в полярных координатах), наследованный от класса Complex. Ввод и вывод этих чисел осуществляется в полярных координатах, хранение и все остальные методы оставить родительские.

Комментарии

Комментарий 1. Для варианта №7. Четырехкомпонентный вектор является представлением трехмерной точки в памяти компьютера, причем первые три компоненты являются координатами x,y,z точки, а четвертая компонента равна 1:

$$P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Матрица 4x4 – матрица преобразования координат точки в пространстве. Любое движение (то есть преобразование пространства, сохраняющее расстояние между точками) в трехмерном пространстве, согласно теореме Шаля, может быть представлено в виде суперпозиции поворота и параллельного переноса, то есть последовательного выполнения поворота и параллельного переноса.

Перенос точки в пространстве осуществляется умножением матрицы $M = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$ на четырехкомпонентный вектор, где dx, dy,

dz – расстояние переноса по каждой из осей.

Поворот точки относительно оси OZ на угол alpha осуществляется умножением на матрицу $\begin{bmatrix} \cos \alpha & -\sin \alpha & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

Поворот точки относительно оси OX на угол alpha осуществляется умножением на матрицу $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

Масштабирование осуществляется умножением на матрицу $\begin{bmatrix} kx & 0 & 0 & 0 \\ 0 & ky & 0 & 0 \\ 0 & 0 & kz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, где kx, ky, kz – коэффициенты масштабирования по соответствующим осям.

В процессе манипуляции над вектором значение последней составляющей может измениться, поэтому после каждого изменения вектора необходимо производить его нормализацию.

Комментарий 2. Для варианта №4. реализовать функцию шифрования можно очень простую, здесь от Вас не требуется создавать мощные криптостойкие алгоритмы, а сделать простенький алгоритм шифрования. Один из вариантов реализации такого алгоритма:

```

1  начальная инициализация
2  пусть k – строка ключа, n = длина ключа, i=0

3  процедура записи в файл
4  пусть s = входная цепочка, m = длина входной цепочки
5  пусть j=0
6  пока j<m делаем
7      запишем в файл k[i] xor s[j]
8      i++, j++
9      если i==n тогда пусть i=0
10  конец пока
11  конец процедуры

12  процедура чтения из файла
13  пусть s = выходная цепочка, m = длина выходной цепочки
14  пусть j=0
15  пока j<m делаем
16      считаем из файла символ ch
17      s[j] = k[i] xor ch
18      i++, j++
19      если i==n тогда пусть i=0
20  конец пока
21  конец процедуры

```

Такой алгоритм называется симметричным – один и тот же ключ используется и для шифрования и для дешифровки данных, более того, в данном случае для этого используется даже один и тот же алгоритм. При реализации такого класса не надо забывать про начальную инициализацию, она должна производиться ровно один раз при открытии файла (на чтение или на запись).

ЛАБОРАТОРНАЯ РАБОТА №4. «ШАБЛОНЫ»

Цель работы

Целью данной лабораторной работы является изучение шаблонов классов в языке C++.

Предварительные сведения

Зачастую в процессе разработки программы возникают ситуации, когда хочется использовать одну и ту же разработанную функцию для разных типов данных. Например, разработаем функцию нахождения числа Фибоначчи для целых чисел:

```
22  int fib_int(int n)
23  {
24      int f1=1,f2=1,f;
25      if (n<3) return 1;
26      for (int i=3; i<=n; i++) {
27          f = f1+f2;
28          f1 = f2;
29          f2 = f;
30      }
31      return f;
32  }
```

Данная функция весьма ограничена в применении. При каком-то значении **n** происходит переполнение целого числа, в результате чего получается некорректный результат. Следующим шагом будет реализовать данную функцию для типа **long**. Можно просто скопировать кусок текста и заменить все **int** на **long**. Точнее сказать не все, а только тип возвращаемого значения и типы переменных **f1**, **f2**, **f**. Но и данный вариант неудовлетворителен, так как при некотором значении **n** мы получим переполнение и для типа **long**. В данном случае следует использовать тип, например, **double**. Однако у этого варианта есть одна проблема: значение получается приближенным, округленным. Хорошо бы разработать класс **large** – бесконечно длинное целое и затем переписать данную функцию для этого класса.

Так или иначе, но нам пришлось переписать данную функцию как минимум трижды. При этом при каждом переписывании функции есть достаточно высокая вероятность внесения ошибок. А если спустя некоторое время мы обнаружим, что изначально ошиблись в реализации функции, то нам придется исправлять код функции везде, где сделана,

что приведет к вероятности появления еще большего количества ошибок. Здесь нам на помощь приходят шаблоны. Шаблон – это описание функции или класса, по которому генерируется уже конкретный класс или функция. С какой-то точки зрения шаблон можно считать строковой подстановкой, аналогичной препроцессору. Запишем шаблон приведенной выше функции:

```

1 template<class T> T fib(int n)
2 {
3     T f1=1,f2=1,f;
4     if (n<3) return 1;
5     for (int i=3; i<=n; i++) {
6         f = f1+f2;
7         f1 = f2;
8         f2 = f;
9     }
10    return f;
11}
```

В данном случае описывается шаблон функции **fib**, который имеет один формальный параметр **T** – тип данных, с которыми будет оперировать данная функция. В данном случае в качестве фактического параметра можно указывать любой встроенный тип или определенный пользователем класс. У функции по-прежнему остался один параметр **n** причем строго целого типа, а в качестве типа возвращаемого значения указан тип **T**, то есть данный тип еще не определен. Он будет определен при использовании шаблона, то есть в тот момент, когда компилятор сгенерирует по данному шаблону функцию. В качестве типа локальных переменных **f1**, **f2**, **f** также используется формальный параметр шаблона **T**.

Заставим компилятор сгенерировать данную функцию и вызовем ее следующим образом:

```

long f;
f = fib<long>(35);
```

Фактическим значением для формального параметра **T** указан тип **long**. При этом компилятор формирует функцию, где везде вместо **T** будет подставлено **long**, и сразу же организует вызов этой функции. Если нам понадобится вычислить число Фибоначчи для большого **n**, то можно сгенерировать эту функцию и для вещественных чисел:

```

double f;
f = fib<double>(350);
```

Это другая, новая, функция, созданная по тому же шаблону, что и предыдущая. Если мы попробуем еще раз использовать функцию для типа **long**, то будет вызвана созданная ранее функция.

Проанализируем внимательно тело функции **fib**. Можно обнаружить, что для типа **T** используются следующие операции: конструктор копирования, операция сложения и операция присваивания. Для встроенных типов данных все эти операции определены и корректно работают. А вот для создаваемых Вами классов эти операции необходимо определить. Иначе будет невозможно использовать данный шаблон для этих классов. Вообще при разработке шаблонов желательно стремиться к минимизации и систематизации используемых операций.

Аналогично шаблону функции можно задать и шаблон класса. Например:

```

1  template <class T, int n>
2  class Array {
3  private:
4      T arr[n];
5  public:
6      T& operator[](int i);
7  };
8  template <class T, int n>
9  T& Array::operator[](int i) {
10     if (i<n) return arr[i];
11     else return arr[0];
12 }
```

В данном примере разработан шаблон класса **Array** – массив с контролем на выход за пределы массива. Что бы создать экземпляр данного класса, можно записать следующее: `Array<double,20>`. В данном случае – массив из 20 элементов типа **double**. Как развитие этого класса можно обеспечить возможность создания динамического массива и реализовать также операции присваивания (копирование массива), конструктор копирования и так далее.

Обратите внимания, что каждый метод класса является шаблоном функции.

Задания

В случае, если непонятна формулировка вопроса, выполняйте задание так, как Вы его понимаете, но предварительно объясните, как Вы поняли задание, то есть сформулируйте Ваше понимание данного задания.

Номер варианта	Задание
1	Разработать шаблон класса TVector – одномерный массив; в качестве параметра шаблона использовать тип данных, хранимых в массиве. Функциональность данного класса аналогична функциональности класса из 2-ой лабораторной работы. В качестве примера использования данного шаблона сделать его реализацию для типа NormVector.
2	Разработать шаблон класса TList – список (односвязный либо двусвязный); в качестве параметра шаблона использовать тип хранимых в списке данных. Обеспечить функциональность данного класса, аналогичную функциональности класса из 2-ой лабораторной работы. В качестве примера использования данного шаблона сделать его реализацию для типа Matrix.
3	Разработать шаблон класса TArray – массив (прямоугольный); в качестве параметра шаблона использовать тип хранимых в массиве данных. Обеспечить функциональность данного класса, аналогичную функциональности класса из 1-ой лабораторной работы. В качестве примера использования данного шаблона сделать его реализацию для типа SqArrayOfInt.
4	Разработать шаблон класса TFileOf – типизированный файл; в качестве параметра шаблона использовать тип данных, хранимых в файле (это могут быть строки фиксированной длины, целые числа, структуры и так далее). Имя открываемого файла и режим открытия (чтение/запись) передается в конструктор. Режим чтения и записи только двоичный. Реализовать методы Read и Write, в которые в качестве параметра передается объект (для Write) или ссылка на объект (для Read), который необходимо записать или прочитать.
5	Разработать шаблон класса TLarge – длинное число; в качестве параметра шаблона указать тип данных, используемых для хранения одной десятичной цифры. Обеспечить функциональность, аналогичную функциональности класса large. В качестве примера использования данного шаблона сделать его реализацию для типа char.

6	Разработать класс TString – строка символов; в качестве параметра шаблона указать тип данных, используемых для хранения одного символа. Обеспечить функциональность, аналогичную функциональности класса String. Размер строки динамически увеличивается в процессе работы со строкой по мере необходимости. В качестве примера использования данного шаблона сделать его реализацию для типа short.
7	Разработать шаблоны классов TVector4 и TMatrix4 – 4-компонентный вектор и матрица 4x4 соответственно; в качестве параметра шаблона использовать тип данных, используемых для хранения одной ячейки. Обеспечить функциональность, аналогичную функциональности классов Vector4 и Matrix4.
8	Разработать шаблон класса TVector – одномерный массив; в качестве параметра шаблона использовать тип данных, хранимых в массиве. Обеспечить метод доступа к элементам массива; дополнительно обеспечить функциональность, аналогичную классу Persons. В качестве примера использования данного шаблона сделать его реализацию для типа Person.
9	Разработать шаблон класса TVector – одномерный массив; в качестве параметра шаблона использовать тип данных хранимых в массиве. Обеспечить метод доступа к элементам массива; дополнительно обеспечить функциональность, аналогичную классу Events. В качестве примера использования данного шаблона сделать его реализацию для типа Person.
10	Разработать шаблон класса TArray – двумерный массив (матрица); в качестве параметра шаблона использовать тип данных, хранимых в матрице. Размерность матрицы задавать в конструкторе. Реализовать метод доступа к элементам массива. Реализовать операции сложения, вычитания, умножения и транспонирования матрицы. Сделать метод определения симметричности матрицы (если матрица квадратная). В качестве примера использования данного шаблона сделать его реализацию для типа Complex.

КОНТРОЛЬНАЯ РАБОТА №2

Цель работы

Целью данной контрольной работы является контроль знаний и умений, полученных в процессе изучения предыдущего материала.

Предварительные сведения

Данная контрольная работа является завершающим звеном в процессе разработки и реализации цикла лабораторных работ. Основная задача данной работы – доведение ранее созданных классов до полной функциональности.

Задания

Номер варианта	Задание
1	Доработать шаблон класса TVector. Будем считать новый шаблон полноценным геометрическим вектором; внести операции сложения, вычитания, умножения скалярного и геометрического (для 3-мерных), умножения на число, нормализацию вектора. В качестве параметра шаблона – тип чисел хранимых в векторе (могут быть float, double, complex и так далее).
2	Разработать шаблон класса TMatrix, в качестве параметра шаблона – тип чисел хранимых в векторе (могут быть float, double, complex и так далее). В данном шаблоне должны быть реализованы методы: сложения, вычитания матрицы, умножения матриц, умножения матрицы на число, транспонирования, вычисления определителя (по определению), обращения матрицы. Создание единичной матрицы.
3	Разработать шаблон класса TVector – одномерный массив; в качестве параметра шаблона использовать тип хранимых в массиве данных. Обеспечить методы: добавление элемента в начало, в конец, в произвольное место массива. Аналогично – удаление. Реализовать методы: среднее арифметическое, среднее геометрическое, максимум, минимум, сортировка по убыванию и по возрастанию. Дополнительно: написать функцию построения ряда Фибоначчи, в качестве контейнера для хранения ряда использовать реализацию шаблона TVector для вещественных чисел.

4	Добавить в шаблон класса TFileOf операцию сортировки файла.
5	Добавить в шаблон класса large операции вычитания и умножения.
6	Добавить в шаблон класса TString методы: DelDupSpaces – удаление дублированных пробелов, Tab2Space – замена табуляции на пробелы, CountWord – подсчет количества слов, CountSimb – подсчет количества символов.
7	Доработать шаблоны классов TVector4 и TMatrix4 для обеспечения полной функциональности (аналогично лабораторной работе №3).
8	Добавить к классу Date метод определения дня недели.
9	Доработать класс Time – написать метод перевода разницы секунд в часы, минуты и секунды.
10	Разработать шаблон класса TComplex, обеспечить ему полную функциональность, необходимую для комплексных чисел, в качестве параметра шаблона – тип данных, используемых для хранения составляющих комплексного числа. Обеспечить операцию деления комплексных чисел.