Министерство образования и науки Украины
Севастопольский национальный технический университет

Методические указания к лабораторным работам по дисциплине “Объектно-ориентированное программирование”

для студентов дневной и заочной формы обучения направления 050101 — “Компьютерные науки”

Часть 2

Севастополь
2013

УДК 004.42 (075.8)

Методические указания к лабораторным работам по дисциплине “Объектно-ориентированное программирование” для студентов дневной и заочной формы обучения направления 050101 — “Компьютерные науки”: в 2 ч. / СевНТУ ; сост. **Ю.В. Доронина**, **Т. И. Сметанина,** А. **К. Забаштанский —** Севастополь: Изд-во СевНТУ, 2012. ч. 2. — 40 с.

Цель указаний: оказать помощь студентам направления “Компьютерные науки” в выполнении лабораторных работ по дисциплине “Объектно-ориентированное программирование” на языке C++.

Методические указания составлены в соответствии с требованиями программы дисциплины “Объектно-ориентированное программирование” для студентов дневной и заочной формы обучения направления 050101 — “Компьютерные науки” и утверждены на заседании кафедры Информационных систем протоколом

Допущено учебно-методическим центром СевНТУ в качестве методических указаний.

Рецензент:
Кожаев Е.А., канд. техн. наук, доцент кафедры кибернетики и вычислительной техники.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цели и задачи лабораторных работ

Основная цель выполнения лабораторных работ — приобретение практических навыков написания программ на объектно-ориентированном языке программирования — C++. В результате выполнения лабораторных работ студенты углубляют знания основных теоретических положений дисциплины «Объектно-ориентированное программирование», решая практические задачи на ЭВМ.

По окончанию курса студенты должны овладеть объектно-ориентированным подходом решения практических задач, овладеть инструментами, сопровождающими разработку программных систем с использованием этого подхода.

Выбор вариантов и график выполнения лабораторных работ

При изучении курса «Объектно-ориентированное программирование» выполняется 8 лабораторных работ. Каждая работа выполняется в течение 4 академических часа.

В лабораторных работах студент решает заданную индивидуальным вариантом задачу. Варианты заданий приведены в каждой лабораторной работе и уточняются с преподавателем. Номер варианта выбирается в соответствии с номером зачетной книжки студента. Вариант вычисляется как остаток от деления числа, соответствующего двум последним цифрам в номере зачетной книжки, на 15. Например, две последние цифры зачетки 42. Тогда остаток деления 42 на 15 будет равен 12. Следовательно, студент выбирает вариант 12.

Лабораторная работа выполняется в два этапа. На первом этапе, выполняемом самостоятельно дома, студенту нужно выполнить следующее:

- проработать по конспекту и рекомендованной литературе, приведенной в конце настоящих методических указаний, основные теоретические положения лабораторной работы и ответить на контрольные вопросы;
- разработать алгоритм решения задачи;
- оформить результаты первого этапа в виде заготовки отчета по выполнению лабораторной работы (студенты-заочники первый этап выполнения лабораторных работ оформляют в виде контрольной работы, которую сдают на кафедру до начала зачетно-экзаменационной сессии).

На втором этапе, выполняемом в лабораториях кафедры, студенту следует:

- написать программу на языке Си ++, реализующий разработанный алгоритм;
- отладить программу;
- разработать и выполнить тестовый пример;
- подготовить и защитить отчёт по лабораторной работе.

Студенты должны выполнять и защищать работы строго по графику.

График выполнения лабораторных работ:

- лабораторная работа №5 — 2-ая неделя весеннего семестра;
- лабораторная работа №6 — 6-ая неделя весеннего семестра;
- лабораторная работа №7 — 10-ая неделя весеннего семестра;
- лабораторная работа №8 — 14-ая неделя весеннего семестра;

Требования к оформлению отчета

Отчёты по лабораторной работе оформляются каждым студентом индивидуально. В общем случае отчёт должен включать: название и номер лабораторной работы; цель работы; вариант задания и постановку задачи; разработку и описание алгоритма решения задачи; разработку и описание программы; выводы по работе; приложения. Содержание отчета указано в методических указаниях к каждой лабораторной работе.

Постановка задачи представляет изложение содержания задачи, цели её решения. На этом этапе должны быть полностью определены исходные данные, перечислены задачи по преобразованию и обработке входных данных, описаны выходные данные, дана характеристика результатов.

Разработка и описание программы включает описание вычислительного процесса на языке C++. Кроме текста программы, в отчёте представляется её пооператорное описание, даётся характеристика конструкций языка, обеспечивающих решение задачи конкретной лабораторной работы.

В приложении приводится текст программы, результат выполнения тестового примера.

ЛАБОРАТОРНАЯ РАБОТА №5

ШАБЛОНЫ.

1. ЦЕЛЬ РАБОТЫ

Приобретение практических навыков при написании объектно-ориентированных программ с использованием шаблонов функций и классов. Освоение особенностей отладки объектно-ориентированных программ.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Шаблоны и их применение

Шаблоны позволяют давать обобщенные, в смысле произвольности используемых типов данных, определения функций и классов. Поэтому их часто называют параметризованными функциями и параметризованными классами. Часто также используются термины “шаблонные функции” и “шаблонные классы”.

Шаблон функции представляет собой обобщенное определение функции, из которого компилятор автоматически создает представитель функции для заданного пользователем типа (или типов) данных. Когда компилятор создает по шаблону функции конкретную ее реализацию, то говорят, что он создал порожденную функцию. Синтаксис объявления шаблона функции имеет следующий вид:

```
template < class T_1|T_1 идент_1>, . . . , <class Tn |Tn идент_n >
возвр_тип имя_функции (список параметров)   // Заголовок функции
{       /* Тело функции */
}
```

За ключевым словом *template* следуют один или несколько параметров, заключенных в угловые скобки, разделенные между собой запятыми. Каждый параметр является либо ключевым словом *class*, за которым следует имя типа, либо именем типа, за которым следует идентификатор.

Для задания параметризованных типов данных вместо ключевого слова *class* может также использоваться ключевое слово *typename*. Параметры шаблона, следующие за ключевыми словами *class* или *typename*, называют ***параметризованными типами***. Они информируют компилятор о том, что некоторый (пока что неизвестный) тип данных используется в шаблоне в качестве параметра (в момент вызова шаблона на место такого параметризованного типа станет тип, заданный программистом). Параметры шаблона, состоящие из имени типа и следующего за ним имени идентификатора, информируют компилятор о том, что параметром шаблона является константа указанного типа. Шаблонную функцию можно вызывать как обычную, никакого специального синтаксиса не требуется.

Рассмотрим пример определения и вызова шаблонных функций:

```
#include <iostream>
template <class T>          // Шаблон функции, вычисляющей квадрат
                            // введенного значения
T Sqr(T x)
{
   return x*x;
}

template < class T>         // Шаблон функции обмена двух элементов
                            // в массиве по значению их индексов
T* Swap(T* t, int indl, int ind2)
{
 T tmp = t[indl];           // Собственно обмен
 t[indl] = t[ind2];
 t[ind2] = tmp;
 return t;
}
int main ( )
{  int n=10,                // Для возведения в квадрат
   i = 2, j = 5;            // Индексы в массиве
   double d = 10.21;        // Для возведения в квадрат
 char* str = "Шаблон";      // Массив букв
 cout << "Значение n = " << n << endl
 << "Его квадрат = " << Sqr (n) << endl      // Вызов шаблона для возведения
 << "Значение d = " << d << endl             // в квадрат целого числа
 <<"Его квадрат = " << Sqr(d) << endl        // Вызов шаблона для возведения
 << "Исходная строка = " << str << endl      // в квадрат вещественного числа
                                             // Преобразование массива
 << "Преобразованная строка = " << Swap(str, i, j) << endl;
return 0;
}
```

При выполнении программа выводит на экран следующее:

```
Значение n = 10
Его квадрат = 100
Значение d = 10.21
Его квадрат = 104.244
Исходная строка = Шаблон
Преобразованная строка = Шанлоб
```

Обратите внимание на вызовы шаблонных функций, сделанные в предыдущем примере: *Sqr(n)* и *Sqr(d)*. Каждый такой вызов приводит к построению конкретной ***порожденной функции***. В данном случае первый вызов приводит к построению

компилятором функции *Sqr()* для целого типа данных, во втором — для типа *double*. В связи с этим процесс построения порожденной функции компилятором называют ***конкретизацией шаблонной функции***.

Как и для обычных функций, можно создать прототип шаблона функции в виде его предварительного объявления. Например:

```
template < class T> T* Swap(T* t, int indl, int ind2);
```

Каждый типовой параметр шаблона должен появиться в списке формальных параметров функции. Например, следующее объявление приведет к ошибке во время компиляции:

```
template < class T, class U>
T FuncName(U);
```

Шаблонная функция может быть также объявлена внешней, статической или встраиваемой (*inline*). В этом случае соответствующий спецификатор располагается вслед за списком параметризированных типов шаблона, а не перед ключевым словом *template*:

```
template extern          //Объявление внешней шаблонной функции
T* Swap(T* t, int indl, int ind2);

template static          //Объявление статической шаблонной функции
T* Swap(T* t, int indl, int ind2);

template inline          //Объявление встроенной шаблонной функции
T *Swap(T* t, int indl, int ind2);
```

2.2. Недостатки шаблонов

Шаблоны предоставляют определенные выгоды при программировании, связанные с широкой применимостью кода и легким его сопровождением. В общем, этот механизм позволяет решать те же задачи, для которых используется полиморфизм. С другой стороны, в отличие от макросов, они позволяют обеспечить безопасное использование типов данных. Однако с их использованием связаны и некоторые недостатки:

- программа содержит полный код для всех порожденных представителей шаблонного класса или функций;
- не для всех типов данных предусмотренная реализация класса или функции оптимальна.

Преодоление второго недостатка возможно с помощью специализации шаблонов.

3. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

1. В ходе самостоятельной подготовки изучить основы работы с шаблонами функций и классов.

2. Разработать программу на языке C++.

3. Разработать тестовые примеры и выполнить отладку программы.

4. Получить результаты работы программы и исследовать её свойства для различных режимов работы, сформулировать выводы.

5. Оформить отчет по проделанной работе.

4. ВАРИАНТЫ ЗАДАНИЙ

Написать функцию-шаблон, заданную по варианту. Проиллюстрировать ее корректную работу на различных по типу наборах данных (*int*, *char* и др.).

Вариант 1

Написать функцию-шаблон, вычисляющую транспонирование матрицы.

Вариант 2

Написать функцию-шаблон, меняющую местами *i*-ю строку и *j*-й столбец в матрице.

Вариант 3

Написать функцию-шаблон сортировки массива по возрастанию методом пузырька.

Вариант 4

Написать функцию-шаблон, определяющую, существуют ли в матрице повторяющиеся строки.

Вариант 5

Написать функцию-шаблон, определяющую элемент, который встречается в массиве максимальное число раз.

Вариант 6

Написать функцию-шаблон, упорядочивающую строки матрицы по возрастанию значения первого элемента в строке.

Вариант 7

Написать функцию-шаблон, вычисляющую максимальное значение элемента в массиве.

Вариант 8

Написать функцию-шаблон, переставляющую *i*-ю и *j*-ю строки в матрице.

Вариант 9

Написать функцию-шаблон, записывающую массив в обратном порядке.

Вариант 10

Написать функцию-шаблон, меняющую диагонали матрицы местами.

Вариант 11

Написать функцию-шаблон последовательного поиска в массиве по ключу. Функция возвращает индекс первого найденного элемента в массиве, равного ключу.

Вариант 12

Написать функцию-шаблон, заменяющую в матрице все значения элементов, меньшие введенного значения, на него.

Вариант 13

Написать функцию-шаблон, циклически сдвигающую одномерный массив элементов n раз (1й элемент становится 2м, 2й – 3м …*n*-й элемент становится первым).

Вариант 14

Написать функцию-шаблон, упорядочивающую строки в матрице по возрастанию элементов характеристического столбца. Элементы характеристического столбца представляют собой максимальный элемент в строке матрицы.

Вариант 15

Написать функцию-шаблон, проверяющую матрицу на симметричность.

5. СОДЕРЖАНИЕ ОТЧЕТА О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Титульный лист, цель работы, вариант задания, текст программы с комментариями, описание тестовых примеров и выводы по проделанной работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое шаблон? Привести пример.
2. Для чего применяются шаблоны?
3. Перечислить особенности работы шаблонов. Что такое порожденная функция?
4. Перечислить достоинства и недостатки шаблонов.

ЛАБОРАТОРНАЯ РАБОТА № 6

ИСПОЛЬЗОВАНИЕ ПОТОКОВ ВВОДА — ВЫВОДА. ПОТОКИ (IOSTREAM). ФАЙЛЫ — ПОТОКИ (FSTREAM). ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ

1. ЦЕЛЬ РАБОТЫ

Изучить механизм потокового ввода/вывода, обеспечивающий гибкий и эффективный с гарантией типа метод обработки символьного ввода целых чисел, чисел с плавающей точкой и символьных строк, а также простую модель его расширения для обработки типов, определяемых пользователем. Приобрести практические навыки написания объектно-ориентированных программ с использованием обработки исключительных ситуаций.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Общие положения

Поток — это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику. Потоки C++, в отличие от функций ввода/вывода в стандартном C, обеспечивают надежную работу как со стандартными, так и с определенными пользователем типами данных, а также единообразный и понятный синтаксис.

Пример:

```
#include <iostream.h>
int main()
{
 int i;
 cin >> i;                          // ввод переменной
 cout << «Вы ввели « << i;   // вывод
 return 0;
}
```

Чтение данных из потока называется извлечением, вывод в поток — помещением, или включением. Поток определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен (оперативная память, файл на диске, клавиатура или принтер). Обмен с потоком для увеличения скорости передачи данных производится, как правило, через специальную область оперативной памяти — буфер. Фактическая передача данных выполняется при выводе после заполнения буфера, а при вводе — если буфер исчерпан.

По направлению обмена потоки можно разделить на входные (данные вводятся в память), выходные (данные выводятся из памяти) и двунаправленные (допускающие как извлечение, так и включение).

По виду устройств, с которыми работает поток, можно разделить потоки на стандартные, файловые и строковые.

Стандартные потоки предназначены для передачи данных от клавиатуры и на экран дисплея, файловые потоки — для обмена информацией с файлами на внешних носителях данных (например, на магнитном диске), а строковые потоки — для работы с массивами символов в оперативной памяти.

Для поддержки потоков библиотека C++ содержит иерархию классов, построенную на основе двух базовых классов — *ios* и *streambuf.* Класс *ios* содержит общие для ввода и вывода поля и методы, класс *streambuf* обеспечивает буферизацию потоков и их взаимодействие с физическими устройствами.

Таблица 2.1. — Классы потокового ввода/вывода и буферизации потоков

ios	базовый класс потоков
istream	класс входных потоков
ostream	класс выходных потоков
iostream	класс двунаправленных потоков
istringstream	класс входных строковых потоков
ostringrstream	класс выходных строковых потоков
stringstream	класс двунаправленных строковых потоков
ifstream	класс входных файловых потоков
ofstream	класс выходных файловых потоков
fstream	класс двунаправленных файловых потоков
iomanip	класс манипуляторов двунаправленных потоков
bitset	класс для работы с двоичными числами
streambuf	базовый класс буферизации потоков
filebuf	класс буферизации файловых потоков

Основным преимуществом потоков по сравнению с функциями ввода/вывода, унаследованными из библиотеки C, является контроль типов, а также расширяемость, то есть возможность работать с типами, определенными пользователем. Для этого требуется переопределить операции потоков.

К недостаткам потоков можно отнести снижение быстродействия программы, которое в зависимости от реализации компилятора может быть весьма значительным.

Заголовочный файл *<iostream>* содержит, кроме описания классов для ввода/вывода, четыре предопределенных объекта потокового ввода/вывода (табл. 2.2).

Таблица 2.2. — Стандартные потоки для ввода/вывода

Объект	Класс	Описание
cin	istream	Связывается с клавиатурой (стандартным буферизованным вводом)
cout	ostream	Связывается с экраном (стандартным буферизованным выводом)
cerr	ostream	Связывается с экраном (стандартным не буферизованным выводом), куда направляются сообщения об ошибках
clog	ostream	Связывается с экраном (стандартным буферизованным выводом), куда направляются сообщения об ошибках

Эти объекты создаются при включении в программу заголовочного файла *<iostream>*, при этом становятся доступными связанные с ними средства ввода/вывода. Имена этих объектов можно переназначить на другие файлы или символьные буферы.

В классах *istream* и *ostream* операции извлечения из потока >> и помещения в поток << определены путем перегрузки операций сдвига.

Операции извлечения и чтения в качестве результата своего выполнения формируют ссылку на объект типа *istream* для извлечения и ссылку на *ostream* — для чтения. Это позволяет формировать цепочки операций, что проиллюстрировано последним оператором приведенного примера. Вывод при этом выполняется слева направо.

Как и для других перегруженных операций, для вставки и извлечения невозможно изменить приоритеты, поэтому в необходимых случаях используются скобки:

```
cout << i + j;      // Скобки не требуются — приоритет сложения больше, чем <<
cout << (i < j);    // Скобки необходимы — приоритет операции отношения
                    // меньше, чем <<:
cout << (i << j);   // Правая операция << означает сдвиг
```

2.2. Флаги и форматирующие методы

Флаги представляют собой отдельные биты, объединенные в поле *x_flags* типа *long* класса *ios*. Флаги перечислены в таблице 2.3. Форматирующие методы приведены в таблице 2.4.

Таблица 2.3. — Флаги форматирования

Флаг	Умолчание	Описание действия при установленном бите
1	2	3
skipws	+	При извлечении пробельные символы игнорируются
left		Выравнивание по левому краю поля
right	+	Выравнивание по правому краю поля
internal		Знак числа выводится по левому краю, число — по правому.
dec	+	Десятичная система счисления
oct		Восьмеричная система счисления
hex		Шестнадцатеричная система счисления
showbase		Выводится основание системы счисления (0x для шестнадцатеричных чисел и 0 для восьмеричных)
showpoint		При выводе вещественных чисел печатать десятичную точку и дробную часть
uppercase		При выводе использовать символы верхнего регистра
showpos		Печатать знак при выводе положительных чисел
scientific		Печатать вещественные числа в форме мантиссы с порядком

Продолжение таблицы 2.3.

1	2	3
fixed		Печатать вещественные числа в форме с фиксированной точкой
unitbuf		Выгружать буферы всех потоков после каждого вывода
stdio		Выгружать буферы потоков *stdout* и *stderr* после каждого вывода

ПРИМЕЧАНИЕ:

Флаги (*left*, *right* и *interna*l), (*dec*, *oct* и *he*x), а также (*scientific* и *fixed*) взаимно исключают друг друга, то есть в каждый момент может быть установлен только один флаг из каждой группы.

Таблица 2.4. — Функции форматирования

Функция	Описание действия
int width(int w);	устанавливает ширину ноля вывода в соответствии со значением параметра
int width();	возвращает значение ширины поля вывода
char fill(char);	устанавливает значение текущего символа заполнения, возвращает старое значение символа
char fill();	возвращает текущий символ заполнения
int precision(int);	устанавливает значение точности представления при выводе вещественных чисел, возвращает старое значение точности
int precision();	возвращает значение точности представления при выводе вещественных чисел
long flags(long f);	установить состояния всех флагов
long flags();	вернуть состояния всех флагов
long setf (long setbits, long ield);	присваивает флагам, биты которых установлены в первом параметре, значение соответствующих битов второго параметра
long setf(long);	устанавливает флаги, биты которых установлены в параметре
long unsetf(long);	сбрасывает флаги, биты которых установлены в параметре

Пример форматирования при выводе с помощью флагов и методов:

```
#include <iostream.h>
int main()
{ long a = 1000, b = 077;
  cout.width(7);
  cout.setf(ios::hex | ios::showbase | ios::uppercase);
```

```
  cout << a;
  cout.width(7);
  cout << b << endl;
double d = 0.12, c = 1.3e-4;
  cout.setf(ios::left);
  cout << d << endl;
  cout << c;
  return 0;
}
```

В результате работы программы в первой строке будут прописными буквами выведены переменные *a* и *b* в шестнадцатеричном представлении, под каждую из них отводится по 7 позиций (функция *width* действует только на одно выводимое значение, поэтому ее вызов требуется повторить дважды). Значения переменных *c* и *d* прижаты к левому краю поля:

```
 0X3E8 0X3F
0.12
0.00013
```

2.3. Манипуляторы

Манипуляторами называются функции, которые можно включать в цепочку операций помещения и извлечения для форматирования данных. Манипуляторы делятся на простые, не требующие указания аргументов, и параметризованные. Пользоваться манипуляторами более удобно, чем методами установки флагов форматирования (табл. 2.5.).

Таблица 2.5. — Манипуляторы

Манипулятор	Назначение	Ввод\вывод
1	2	3
dec	Вывод чисел в десятичной системе счисления	Вывод
endl	Вывод символа новой строки и флэширование	Вывод
ends	Вывод нуля (*NULL*)	Ввод\вывод
flush	Флэширование	Вывод
hex	Вывод чисел в шестнадцатеричной системе счисления	Вывод
oct	Вывод чисел в восьмеричной системе счисления	Вывод
resetiosflags(long f)	Сбросить флаги, определяемые *f*	Ввод\вывод
setbase(int base)	Установить основание системы счисления	Вывод
setfill(char ch)	Установить символ заполнения *ch*	Вывод
setiosflags(long f)	Установить флаги, задаваемые *f*	Ввод\вывод
setprecision(int p)	Установить точность, равную *p*	Вывод
setw(int w)	Установить ширину поля, равную *w*	Вывод
ws	Пропуск начальных пробелов	Ввод

2.4. Методы обмена с потоками

В потоковых классах наряду с операциями извлечения >> и включения << определены методы для неформатированного чтения и записи в поток (при этом преобразования данных не выполняются).

В таблице 2.6. приведены функции чтения, определенные в классе *istream.*

Таблица 2.6. — Функции чтения

Функция	Описание действия
get ()	возвращает код извлеченного из потока символа или *EOF*
get (c)	возвращает ссылку на поток, из которого выполнялось чтение, и записывает извлеченный символ в с
get(buf,num,lim='\n')	считывает *num-1* символов или пока не встретится символ *lim* и копирует их в символьную строку *buf.* Вместо символа *lim* в строку записывается признак конца строки (*'\0'*). Символ *lim* остается в потоке. Возвращает ссылку на текущий поток
getline(buf. num. 1im='\n')	аналогична функции *get*, но копирует в *buf* и символ *lim*;
ignore(num = 1, lim = EOF)	считывает и пропускает символы до тех пор, пока не будет прочитано *num* символов или не встретится разделитель, заданный параметром *lim.* Возвращает ссылку на текущий поток
seekg(pos)	устанавливает текущую позицию чтения в значение *pos*
seekg(offs, org)	перемещает текущую позицию чтения на *offs* байтов, считая от трех позиций, определяемых параметром *org: ios::beg* (от начала файла), *ios::cur* (от текущей позиции) или *ios::end* (от конца файла)
flushO	записывает содержимое потока вывода на физическое устройство
put (c)	выводит в поток символ с и возвращает ссылку на поток
seekg(pos)	устанавливает текущую позицию записи в значение *pos*

В классе *ostream* определены аналогичные функции для вывода (табл. 2.7.).

Таблица 2.7. — Функции вывода

Функция	Описание действия
seekg (offs, org)	перемещает текущую позицию записи на *offs* байтов, считая от трех позиций, определяемых параметром *org: ios::beg* (от начала файла), *ios::cur* (от текущей позиции) или *ios::end* (от конца файла)
write(buf, num)	записывает в поток *num* символов из массива *buf* и возвращает ссылку на поток

2.5. Файловые потоки

Чтобы открыть для вывода файл *myfile.txt* с помощью объекта *ofstream*, необходимо создать экземпляр объекта класса ofstream и передать ему имя файла в качестве параметра: *ofstream fout («myfile.txt»).*

Чтобы открыть этот файл для ввода, применяется та же методика, за исключением того, что используется объект класса *ifstream: ifstream fin («myfile.txt»).*

Обратите внимание, что *fout* и *fin* не более чем имена объектов; здесь *fout* использовался для вывода в файл подобно тому, как *cout* используется для вывода на экран; *fin* аналогичен *cin*.

Очень важным методом, используемым в файловых потоках, является функция-член *close().* Каждый раз при открытии файла для чтения или записи (или и того и другого) создается соответствующий объект файлового потока. По завершении работы файл необходимо закрыть (например: *fout.close()*;), чтобы впоследствии не повредить его и записанные в нем данные. На практике нередко бывают непредвиденные случаи, когда не закрытые файлы теряют всю внесенную в них информацию.

После того как объекты потока будут ассоциированы с файлами, они используются наравне с другими объектами потока ввода и вывода. Например:

```
#include <fstream.h>
int main()
 {  char buffer[255];            // для ввода пользователя
    cout << “File name: “;
    cin.getline (buffer,255);
    ofstream fout(buffer);        // открыть для записи
    fout << “This line written directly to the file\n”;
    cin.getline (buffer,255);     // получить данные от пользователя
    fout << buffer;               // и запись их в файл
    fout.close();                 // закрыть файл
    return 0;                     // уходя гасите свет
}
```

Обычно объект класса *ofstream*, открывая файл для записи, создает новый файл, если таковой не существует, или усекает его длину до нуля, если файл с этим именем уже существует (то есть удаляет все его содержимое). Изменить стандартное поведе-

ние объекта *ofstream* можно с помощью второго аргумента конструктора, заданного явно.

Допустимыми аргументами являются:

- *ios::app* — добавляет данные в конец файла, не усекая прежнее его содержимое;
- *ios::ate* — осуществляет переход в конец файла, но запись допускает в любом месте файла;
- *ios::trunc* — задано по умолчанию; усекает существующий файл полностью;
- *ios::nocreate* — открывает существующий файл;
- *ios::noreplace* — открывает несуществующий файл.

2.6. Ошибки потоков

Каждый поток (*istream* или *ostream*) имеет связанное с ним состояние. Установкой и соответствующей проверкой этого состояния вылавливаются ошибки и нестандартные условия. Состояние потока вводится в *basic_ios*, базовом классе класса *basic_stream*, в <*ios*> (табл 2.8.).

Таблица 2.8. — Функции работы с ошибками потоков

Функция	Описание действия
bool good() const;	следующая операция может выполниться
bool fail() const;	следующая операция не выполнится
bool eof() const;	виден конец ввода
bool bad() const;	поток испорчен
iostate rdstate() const;	получение флагов состояния ввода/вывода
void clear(iostate f=goodbit);	сбрасываются флаги ошибок (по умолчанию - все). Обычно после возникновения ошибки нужно сбросить ошибочное состояние, чтобы дальше пользоваться потоком, но этого недостаточно – для восстановления работоспособности еще нужно вручную очистить буфер ввода/вывода.
void setstate(iostate f) {clear(rdstate() | f);}	добавление *f* к флагам состояния
operator void() const;*	не ноль, если *!fail()*
bool operator!() const {return fail();}	не *good()*

Состояние потока представляется набором флагов. Эти флаги определены в базовом классе *ios*:

```
enum io_state
{  goodbit  = 0x00.      //Нет ошибок
   eofbi t= 0x01,        //Достигнут конец файла
   failbit = 0x02,       //Ошибка форматирования или преобразования
   badbit  = 0x04.       //Серьезная ошибка, после которой
   };                    //пользоваться потоком невозможно.
```

Пример: проще всего состояние потока можно проверить как обычное логическое выражение. Обычно нужно проверить, ввел ли пользователь то, что ожидает программа:

```
double d;
cin>>d;  /* 1) 2 – преобразуется к d=2.0
            2) 2A – d=2.0 – значение сформировано, но «A» еще осталось в буфере потока и ждет приема char
            3) 3,3 вместо 3.3 – d=3.0, но запятая и вторая 3 осталась в буфере ввода и ждет ввода
            4) вводим AAA - поток испорчен, вырабатывается флаг ошибки, пользоваться d нельзя! Значение d не изменилось!!! Весь последующий ввод (cin>>) будет проигнорирован!
         */
 if( !cin )
 {          /* Сюда попадем, если только поток ввода Ваш ввод никаким образом проинтерпретировать не может (случай 4). Для того, чтобы программу можно было выполнять дальше необходимо:
               1) сбросить ошибки
               */
   cin.clear();  // иначе весь последующий ввод будет проигнорирован
 }
 else {/*используем d*/}
               // 2) очистить буфер ввода
 cin.ignore(MAX_INT,'\n');
 cin>>d;       // теперь можно вводить снова
```

2.7. Обработка исключительных ситуаций

Исключение - это некоторое событие, которое является неожиданным и зачастую прерывает нормальный процесс выполнения программы.

Стандарт языка C++ предусматривает встроенный механизм обработки исключений. Кроме того, компиляторы Visual C++ 6.0 и Borland C++ Builder предусматривают еще один механизм обработки исключений, который реализуется в тесном взаимодействии этих компиляторов с операционной системой и называется структурированной обработкой исключений (SEH - от англ. Structured Exception Handling). Этот последний механизм был реализован еще в компиляторах языка C, в связи с чем часто называется C-исключениями. Хотя структурированная обработка исключений может быть использована в C++, для программ, написанных на этом языке, рекомендуется использовать более новый механизм обработки исключений C++ (C++ EH от англ. C++ Exception Handling). Для управления исключениями в C++ используются три ключевых слова: ***try, catch*** и ***throw***.

Ключевое слово try служит для обозначения блока кода, который может генерировать исключение. При этом соответствующий блок заключается в фигурные скобки и называется *защищенным* или *try-блоком*:

```
try
{
    //Защищенный блок кода
}
```

Тело всякой функции, вызываемой из ***try***-блока, также принадлежит ***try***-блоку. Предполагается, что одна или несколько функций или инструкций из защищенного блока могут выбрасывать (или генерировать) исключение. Если выброшено исключение, выполнение соответствующей функции или инструкции приостанавливается, все оставшиеся инструкции ***try***-блока игнорируются, а управление передается вовне блока try.

Ключевое слово ***catch*** следует непосредственно за try-блоком и обозначает секцию кода, в которую передается управление, если произойдет исключение. За ключевым словом ***catch*** следует описание исключения, состоящее из имени типа исключения и необязательной переменной, заключенное в круглые скобки:

```
catch (<имя типа>[<перменная>])
 {
    //Обработчик исключения
 }
```

Имя типа исключения идентифицирует обслуживаемый тип исключений. Блок кода, обрабатывающего исключение, заключается в фигурные скобки и называется *catch-блоком* или *обработчиком* исключения. При этом говорят, что данный ***catch***-блок перехватывает исключения описанного в нем типа. Если исключение перехвачено, переменная получает его значение. Если вам не нужен доступ к самому исключению, указывать эту переменную не обязательно. Переменная исключения может иметь любой тип данных, включая созданные пользователем типы классов.

За одним ***try***-блоком могут следовать несколько ***catch***-блоков. Оператор ***catch***, с указанным вместо типа исключения многоточием, перехватывает исключения любого типа и должен быть последним из операторов ***catch***, следующих за ***try***-блоком. Рассмотрим простой пример обработки исключения:

```
#include < exception.h>
#include < iostream.h>
void func()
 {    //Функция генерирует исключение
 }
 //-------------------------------------------------------------------------------------
 int main() {
    try   //Пример использования механизма обработки исключений C++
   {
      func(); //Защищенный участок кода
      return 0;
    }
    catch (const char*)  //Обработчик исключения типа const char*
```

```
    {
        cout << "Было выброшено исключение  типа const char*\n";
         return 1;
    }
    catch (...)    //Обработчик всех необработанных исключений
    {
        cout << "Было выброшено исключение  другого типа\n";
        return 1;
    }
}
```

В данном примере функция ***func( )*** может генерировать исключения. В связи с этим она включена в защищенный блок. Перехват исключений осуществляется в двух ***catch***-блоках. Первый из них перехватывает исключения, имеющие тип ***const char****, второй — любого другого типа.

2.8. Генерирование исключений

Для обозначения в программе места и типа выбрасываемого исключения служит ключевое слово ***throw***. Местоположение инструкции ***throw*** определяет точку выброса исключения. Синтаксис его использования следующий:

throw < объект >.

Операнд в инструкции ***throw*** не является обязательным. Инструкция ***throw*** без операнда используется для повторного выбрасывания исключения того же типа, которое обрабатывается в данный момент, следовательно, она может использоваться только в catch-блоках. Рассмотрим пример, демонстрирующий выбрасывание исключений:

```
#include < stdio.h>
bool test;
class SomeException{}; \\ Класс исключений

void Func (bool bvar) \\ Функция, создающая исключительную ситуацию
{
    if (bvar) throw SomeException(); \\ Вбрасывание исключения
}

int main()
{
    try \\ Блок защищенного кода
      {
       test = true;
       Func(true) ;
    }
catch(SomeException& e)   \\ Блок обработчика исключения
    {
        test = false;
    }
    return test ? (puts("test = true.\n"),1) : (puts("test = false.\n"),0);
```

}

При выполнении программа выводит на экран: ***test = false.***

2.9. Перехватывание исключений

Если в некоторой точке выбрасывается исключение, поток выполнения программы прерывается и происходит следующее:

Если выброшена переменная встроенного типа или объект класса по значению, создается копия выброшенной переменной (причем в последнем случае используется конструктор копирования); если выброшена переменная по ссылке, копирование не производится.

1. Осуществляется поиск ближайшего обработчика, принимающего параметр, совместимый по типу с выброшенной переменной;

2. Если обработчик найден, управление программой передается найденному обработчику.

3. Если обработчик не найден, можно вызвать ***set_terminate()*** — предоставив обработчик завершения; в противном случае программа вызывает функцию завершения.

Если исключение не выброшено, программа выполняется обычным образом.

При поиске соответствующего типа исключения компилятор пользуется следующими правилами. Обработчик считается найденным, если:

a) тип выброшенной переменной совпадает с типом, ожидаемым обработчиком, либо может быть приведен к нему (другими словами, если тип выброшенной переменной есть ***T***, соответствующими ему признаются обработчики, перехватывающие исключение типа ***T***, ***const T, T& и const T&***);

b) тип выброшенной переменной представляет собой указатель, тип которого может быть преобразован к типу указателя, перехватываемого обработчиком;

c) если выброшенная переменная представляет собой объект некоторого класса, а тип перехватываемого исключения является базовым классом, наследуемым этим объектом как ***public***.

Следует обратить внимание на то, что компилятор ищет ближайший обработчик, принимающий параметр, совместимый по типу с выброшенной переменной. Этот момент не следует забывать и внимательно следить за порядком, в котором располагаются в программе обработчики исключений для данного try-блока. Обработчик, ожидающий исключение базового класса, скрывает обработчик производного класса. Точно так же обработчик для указателя типа void* скрывает обработчик для указателя любого типа.

2.10. Использование вложенных блоков try/catch

Иногда возникает необходимость использования вложенных блоков try/ catch. Для этого в языке C++ нет никаких препятствий, однако должно соблюдаться единственное требование: за каждым try-блоком обязательно должен стоять catch-блок.

Часто вложенные try/catch блоки возникают неявно, в результате создания защищенного блока в функции, которая сама находится в защищенном блоке.

К вложенным ***try/catch***-блокам приходится прибегать потому, что какая-то функция может непредвиденно привести к исключению. В этом случае простейший выход — заключить весь код в функции ***main()*** в такой блок, причем для перехвата исключения использовать обработчик вида ***catch(...)***. Затем можно использовать средства, предоставляемые компилятором для определения места в программе, вызвавшего появление исключения.

3. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

1. В ходе самостоятельной подготовки изучить основы работы с потоковым вводом/выводом и исключениями.

2. Разработать программу на языке C++.

3. Разработать тестовые примеры и выполнить отладку программы.

4. Получить результаты работы программы и исследовать её свойства для различных режимов работы, сформулировать выводы.

5. Оформить отчет по проделанной работе.

4. ВАРИАНТЫ ЗАДАНИЙ

Вариант 1

1. Написать программу — возведение числа m в степень числа n. Входные данные поступают с клавиатуры. Результат выводится в файл в 15-ти позициях, точность составляет 3 знака, пробелы необходимо заменить символом “@”. Предусмотреть обработку ошибок.

2. Создать класс массив букв. Описать перехват ошибок, связанных с недопустимым индексом массива. Пример: MyClass Mass(13); cout<<Mass.GetValue(27).

Вариант 2

1. Написать программу нахождения всех возможных делителей введенного числа. Представить результаты в виде таблицы в файле, состоящей из следующих полей: №п/п, делитель в “научном” формате, в двоичной системе, в системе счисления с основанием 16. Делители упорядочить по возрастанию значения. Предусмотреть обработку ошибок.

2. Найти значение математического выражения, описав перехват ошибок вычислений: $y = 14(\ln(x))^2 + x^2$.

Вариант 3

1. Написать программу вычисления длины и площади окружности. Входные данные поступают с клавиатуры. Результат выводится в файл в 17-ти позициях, точность составляет 5 знаков, пробелы необходимо заменить знаком “&”. Предусмотреть обработку ошибок.

2. Найти значение математического выражения, описав перехват ошибок вычислений: $y = \frac{3.8\sinh(x)}{36x^3}$.

Вариант 4

1. Найти все положительные степени двойки, значение которых не превышает величины введенного с клавиатуры числа. Входные данные поступают с клавиатуры. Результат записывается в файл в виде таблицы: №п/п, показатель степени двойки, значение в десятичной системе, значение в двоичной системе, значение в системе счисления с основанием 8. Предусмотреть обработку ошибок.

2. Найти значение математического выражения, описав перехват ошибок вычислений: $y = 99x^2 + \sinh(x)$.

Вариант 5

1. Написать программу решения линейного уравнения. Результаты вычислений поместить в файл. Установить ширину поля 12 символов, точность — 4 цифры, пробелы заменить на символ “%”. Предусмотреть обработку ошибок.

2. Создать класс Вектор (float *). Описать перехват ошибок, связанных с неверным вводом значений. Пример: ввели букву вместо цифры.

Вариант 6

1. Написать программу вычисления наибольшего общего делителя двух целых чисел. Входные данные поступают с клавиатуры. Результат записать в файл в “научном” формате и системах счисления с основаниями 10, 16. Предусмотреть обработку ошибок.

Наибольший общий делитель рекурсивно вычисляется следующим образом:

GCD(m, n) is:

if m mod n equals 0 then n;

else GCD(n, m mod n).

2. Найти значение математического выражения, описав перехват ошибок вычислений: $y = \frac{1}{x^2 + 4x + 4}$.

Вариант 7

1. Написать программу вычисления частного и остатка от деления двух целых чисел. Результаты вычислений поместить в файл. Установить ширину поля 11 символов, заменить пробелы символом “$” с помощью функций и манипуляторов. Предусмотреть обработку ошибок.

2. Найти значение математического выражения, описав перехват ошибок вычислений: $y = ctg(x) * x^2$.

Вариант 8

1. Написать программу преобразования температуры в градусах Цельсия (например: *15C*) в температуру в градусах по Фаренгейту (например: *59F*), и наоборот. 0 градусов по Цельсию соответствует 32 градусам по Фаренгейту. Изменение температуры на 1 градус по Цельсию соответствует изменению на 1.8 градуса по Фаренгейту. Результаты вычислений поместить в файл. Установить ширину поля 13 символов,

точность — 4 цифры, заменить пробелы символом “/” с помощью функций и манипуляторов. Предусмотреть обработку ошибок.

2. Найти значение математического выражения, описав перехват ошибок вычислений: $y = 10\arcsin(10x + 2.2)$.

Вариант 9

1. Написать программу решения квадратного уравнения. Ввод данных с консоли, вывод осуществить в файл. Вывод результата в “научном” формате. Установить ширину поля 12 символов, установить точность 4 цифры, заменить пробелы символом “~” с помощью функций и манипуляторов. Предусмотреть обработку ошибок.

2. Найти значение математического выражения, описав перехват ошибок вычислений: $y = \frac{10x}{\arccos(x^2)}$.

Вариант 10

1. Написать программу, печатающую символы от “*A*” до введенного с клавиатуры символа (последний возможный “*Z*”). Для каждого символа вывести номер, сам символ, шестнадцатеричный, восьмеричный и двоичный код этого символа. Сохранить результат работы программы в файле. Предусмотреть обработку ошибок.

2. Создать класс МойФайл, содержащий строку – путь к файлу. Описать перехват ошибок, связанных с некорректной работой с файлом. Пример: попытка открыть файл, не существующий по данному пути.

Вариант 11

1. Написать программу вычисления длины периметра и площади прямоугольника. Входные данные поступают с клавиатуры. Результат сохраняется в файле в формате: 15 позиций; точность 5 символов; заполняющий символ “#”. Предусмотреть обработку ошибок.

2. Найти значение математического выражения, описав перехват ошибок вычислений: $y = 12x + \sqrt{x - 8}$.

Вариант 12

1. Написать программу, которая обрабатывает все символы введенной строки. Каждый символ печатается в файл в новой строке, также печатаются его двоичный, десятичный и восьмеричный коды. Предусмотреть обработку ошибок.

2. Найти значение математического выражения, описав перехват ошибок вычислений: $y = \frac{\sqrt{x^3}}{tg(x)}$.

Вариант 13

1. Написать программу “калькулятор” (операции: сложение, вычитание, умножение и деление). Входные данные поступают с клавиатуры в формате “число операция число”. Результат сохраняется в файле в формате: 15 позиций; точность 5 символов; заполняющий символ “^”. Предусмотреть обработку ошибок.

2. Найти значение математического выражения, описав перехват ошибок вычислений: $y = \ln(\cos(1/x))$.

Вариант 14

1. Написать программу, рассчитывающую элементы ряда Фибоначчи. Каждый элемент этого ряда равен сумме двух предыдущих: $X_n=X_{(n-1)}+X_{(n-2)}$. Полагать $X_0=1$ и $X_1=1$. Вычислять до номера элемента ряда, введенного с клавиатуры. Вывести номер элемента ряда, его значение, шестнадцатеричный, восьмеричный, двоичный код. Результат работы программы сохранить в файл. Предусмотреть обработку ошибок.

2. Найти значение математического выражения, описав перехват ошибок вычислений: $y = \ln(1/\arctan(x))$.

Вариант 15

1. Написать программу вычисления длины периметра и площади прямоугольного треугольника. Входные данные поступают с клавиатуры. Результат сохраняется в файл в формате: 18 позиций; точность 6 символов; заполняющий символ “*”. Предусмотреть обработку ошибок.

2. Перехватить исключение типа переполнение значения переменной, например, при вычислении уравнения для целочисленного типа результата: $y = 10x^{10}$.

5. СОДЕРЖАНИЕ ОТЧЕТА О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Титульный лист, цель работы, вариант задания, текст программы с комментариями, описание тестовых примеров и выводы по проделанной работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Объясните понятие “поток ввода/вывода”.
2. Поясните механизм буферизации потоков ввода/вывода.
3. Каким образом осуществляется ввод в поток?
4. Какие встроенные типы данных поддерживает оператор вставки в поток?
5. Каким образом осуществляется вывод из потока?
6. Какие средства для управления форматированием потока предусматривает библиотека ввода/вывода?
7. Какие функции-члены класса *ios* используются для опроса и установки состояния потока?
8. Какие существуют режимы открытия файла?
9. Какой режим открытия файла установлен по умолчанию?
10. Какие формы конструкторов предусмотрены для создания файлового потока?
11. Что такое исключение? Когда оно возникает?
12. Пояснить механизм обработки исключений.
13. Как происходит перехват исключения?
14. Можно ли использовать вложенные конструкции ***try/catch?*** Когда и при каких условиях?

ЛАБОРАТОРНАЯ РАБОТА №7
СТАНДАРТНАЯ БИБЛИОТЕКА ШАБЛОНОВ. КОНТЕЙНЕРЫ.

1. ЦЕЛЬ РАБОТЫ

Приобретение практических навыков в написании объектно-ориентированных программ с использованием контейнеров стандартной библиотеки шаблонов. Освоение особенностей отладки объектно-ориентированных программ.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Основные концепции STL

Стандартная библиотека шаблонов STL (***Standard Template Library***) состоит из двух основных частей: набора контейнерных классов и набора обобщенных алгоритмов для этих классов.

Контейнеры — это объекты, содержащие другие однотипные объекты, и предоставляющие интерфейс для манипуляции содержащимися в них объектами. Контейнерные классы являются шаблонными, поэтому хранимые объекты могут быть и встроенных, и пользовательских типов. Эти объекты должны допускать копирование и присваивание. Встроенные типы этим требованиям удовлетворяют; то же самое относится к классам, если конструктор копирования или операция присваивания не объявлены в них закрытыми или защищенными. В контейнерных классах реализованы такие типовые структуры данных, как стек, список, очередь и т.д.

Обобщенные алгоритмы реализуют большое количество процедур, применимых к контейнерам — например, поиск, сортировку, слияние и т. п. Однако они не являются методами контейнерных классов. Наоборот, алгоритмы представлены в STL в форме глобальных шаблонных функций. Благодаря этому достигается их универсальность: эти функции можно применять не только к объектам различных контейнерных классов, но также и к массивам. Независимость от типов контейнеров достигается за счет косвенной связи функции с контейнером: в функцию передается не сам контейнер, а пара адресов first, last, задающая диапазон обрабатываемых элементов.

Реализация указанного механизма взаимодействия базируется на использовании так называемых итераторов. ***Итераторы*** — это обобщение концепции указателей: они ссылаются на элементы контейнера. Их можно инкрементировать, как обычные указатели, для последовательного продвижения по контейнеру, а также разыменовывать для получения или изменения значения элемента.

2.2. Контейнеры

Контейнеры STL можно разделить на два типа: последовательные и ассоциативные.

Последовательные контейнеры обеспечивают хранение конечного количества однотипных объектов в виде непрерывной последовательности. К базовым последовательным контейнерам относятся векторы (vector), списки (list) и двусторонние очереди (deque). Есть еще специализированные контейнеры (или адаптеры контейнеров), реализованные на основе базовых — стеки (stack), очереди (queue) и очереди с приоритетами (priority_queue).

Ассоциативные контейнеры обеспечивают быстрый доступ к данным по ключу. Эти контейнеры построены на основе сбалансированных деревьев. Существует пять типов ассоциативных контейнеров: словари (map), словари с дубликатами (multlmap), множества (set), множества с дубликатами (multiset) и битовые множества (bitset). Для использования контейнера в программе необходимо включить в нее соответствующий заголовочный файл. Тип объектов, сохраняемых в контейнере, задается с помощью аргумента шаблона, например:

```
vector<int> aVect; / / создать вектор aVect целых чисел (типа int )
list<Man> department; / / создать список department типа Man
```

2.3. Итераторы

Итераторы — это обобщение указателей, которые позволяют программисту работать с различными структурами данных (контейнерами) единообразным способом.

Для всех контейнерных классов STL определен тип iterator, однако реализация его в разных классах разная. Например, в классе vect, где объекты размещаются один за другим, как в массиве, тип итератора определяется посредством typedef T* iterator. А вот в классе list тип итератора реализован как встроенный класс iterator, поддерживающий основные операции с итераторами.

К основным операциям, выполняемым с любыми итераторами, относятся:

– Разыменование итератора: если p — итератор, то *p — значение объекта, на который он ссылается.

– Присваивание одного итератора другому.

– Сравнение итераторов на равенство и неравенство (== и !=).

– Перемещение его по всем элементам контейнера с помощью префиксного (++p) или постфиксного (p++) инкремента.

Так как реализация итератора специфична для каждого класса, то при объявлении объектов типа итератор всегда указывается область видимости в форме имя_шаблона::, например:

```
vector<int> : : iterator iter1;
list<Man> : : iterator iter2;
```

Организация циклов просмотра элементов контейнеров тоже имеет некоторую специфику. Так, если i — некоторый итератор, то вместо привычной формы

for (i=0; i < n; ++i) используется следующая: for (i = first; i!=last; ++i),

где first — значение итератора, указывающее на первый элемент в контейнере, а last — значение итератора, указывающее на воображаемый элемент, который следует за последним элементом контейнера. Операция сравнения “<” здесь заменена на операцию “!=”, поскольку операции “<” и “>” для итераторов в общем случае не поддерживаются.

Для всех контейнерных классов определены унифицированные методы begin() и end(), возвращающие адреса first и last соответственно.

Все типы итераторов в STL принадлежат пяти категорий: ***входные, выходные, прямые, двунаправленные*** итераторы и ***итераторы произвольного доступа***.

Входные итераторы (InputIterator) используются алгоритмами STL для чтения значений из контейнера, аналогично тому, как программа может вводить данные из потока cin.

Выходные итераторы (OutputIterator) используются алгоритмами для записи значений в контейнер, аналогично тому, как программа может выводить данные в поток cout.

Прямые итераторы (ForwardIterator) используются алгоритмами для навигации по контейнеру только в прямом направлении, причем они позволяют и читать, и изменять данные в контейнере.

Двунаправленные итераторы (BidirectionalIterator) имеют все свойства прямых итераторов, но позволяют осуществлять навигацию по контейнеру и в прямом, и в обратном направлениях (для них дополнительно реализованы операции префиксного и постфиксного декремента).

Итераторы произвольного доступа (RandomAccessIterator) имеют все свойства двунаправленных итераторов плюс операции (наподобие сложения указателей) для доступа к произвольному элементу контейнера.

В то время как значения прямых, двунаправленных и итераторов произвольного доступа могут быть сохранены, значения входных и выходных итераторов сохраняться не могут (аналогично тому, как не может быть гарантирован ввод тех же самых значений при вторичном обращении к потоку cin). Следствием является то, что любые алгоритмы, базирующиеся на входных или выходных итераторах, должны быть однопроходными. Для двунаправленных итераторов и итераторов произвольного доступа определены разновидности, называемые ***адаптерами итераторов***. Адаптер, просматривающий последовательность в обратном направлении, называется reverse_iterator.

2.4. Общие свойства контейнеров

В табл. 2.1. приведены имена типов, определенные с помощью typedef в большинстве контейнерных классов.

Таблица 2.1. — Унифицированные типы, определенные в STL

Поле	Пояснение
value_type	Тип элемента контейнера
size_type	(эквивалентен unsigned int) Тип индексов, счетчиков элементов и т. д.
iterator	Итератор
constiterator	Константный итератор (значения элементов изменять запрещено)
reference	Ссылка на элемент
constreference	Константная ссылка на элемент (значение элемента изменять запрещено)
keytype	Тип ключа (для ассоциативных контейнеров)
key_compare	Тип критерия сравнения (для ассоциативных контейнеров)

В табл. 2.2. представлены некоторые общие для всех контейнеров операции.

Таблица 2.2. — Операции и методы, общие для всех контейнеров

Операция или метод	Пояснение
Операции равенства (==) и неравенства (! =)	Возвращают значение true или false
Операция присваивания (=)	Копирует один контейнер в другой
Clear	Удаляет все элементы
insert	Добавляет один элемент или диапазон элементов
Erase	Удаляет один элемент или диапазон элементов
size_type size() const	Возвращает число элементов
size_type max_size() const	Возвращает максимально допустимый размер контейнера
bool empty() const	Возвращает true, если контейнер пуст
iterator begin()	Возвращают итератор на начало контейнера (итерации будут производиться в прямом направлении)
iterator end()	Возвращают итератор на конец контейнера (итерации в прямом направлении будут закончены)
reverse_iterator begin()	Возвращают реверсивный итератор на конец контейнера (итерации будут производиться в обратном направлении)
reverse iterator end()	Возвращают реверсивный итератор на начало контейнера (итерации в обратном направлении будут закончены)

2.5. Алгоритмы

Алгоритм — это функция, которая производит некоторые действия над элементами контейнера (контейнеров). Чтобы использовать обобщенные алгоритмы, нужно подключить к программе заголовочный файл <algorthm>.

В табл. 2.3. приведены наиболее популярные алгоритмы STL.

Таблица 2.3. — Некоторые типичные алгоритмы STL

Алгоритм	Назначение
accumulate	Вычисление суммы элементов в заданном диапазоне
copy	Копирование последовательности, начиная с первого элемента
count	Подсчет количества вхождений значения в последовательность
count_if	Подсчет количества выполнений условия в последовательности
equal	Попарное равенство элементов двух последовательностей
fill	Замена всех элементов заданным значением
find	Нахождение первого вхождения значения в последовательность
find_first_of	Нахождение первого значения из одной последовательности в другой
find_if	Нахождение первого соответствия условию в последовательности
for_each	Вызов функции для каждого элемента последовательности
merge	Слияние отсортированных последовательностей
remove	Перемещение элементов с заданным значением
replace	Замена элементов с заданным значением
search	Нахождение первого вхождения в первую последовательность второй последовательности
sort	Сортировка
swap	Обмен двух элементов
transform	Выполнение заданной операции над каждым элементом последовательности

В списках параметров всех алгоритмов первые два параметра задают диапазон обрабатываемых элементов в виде полуинтервала [first, last), где first — итератор, указывающий на начало диапазона, а last — итератор, указывающий на выход за границы диапазона.

2.6. Использование последовательных контейнеров

К основным последовательным контейнерам относятся вектор (vector), список (list) и *двусторонняя очередь* (deque).

Чтобы использовать последовательный контейнер, нужно включить в программу соответствующий заголовочный файл:

```
#include <vector>
#include <list >
```

```
#include <deque>
using namespace std;
```

Контейнер ***вектор*** является аналогом обычного массива, за исключением того, что он автоматически выделяет и освобождает память по мере необходимости. Контейнер эффективно обрабатывает произвольную выборку элементов с помощью операции индексации [ ] или метода at. Однако вставка элемента в любую позицию, кроме конца вектора, неэффективна. Для этого потребуется сдвинуть все последующие элементы путем копирования их значений. По этой же причине неэффективным является удаление любого элемента, кроме последнего.

Контейнер ***список*** организует хранение объектов в виде двусвязного списка. Каждый элемент списка содержит три поля: значение элемента, указатель на предшествующий и указатель на последующий элементы списка. Вставка и удаление работают эффективно для любой позиции элемента в списке. Однако список не поддерживает произвольного доступа к своим элементам: например, для выборки n-го элемента нужно последовательно выбрать предыдущие n -1 элементов.

Контейнер ***двусторонняя очередь (дек)*** во многом аналогичен вектору, элементы хранятся в непрерывной области памяти. Но в отличие от вектора двусторонняя очередь эффективно поддерживает вставку и удаление первого элемента (так же, как и последнего).

Существует пять способов определить объект для последовательного контейнера.

1. Создать пустой контейнер:

```
vector<int> vec1;
list<string> list1;
```

2. Создать контейнер заданного размера и инициализировать его элементы значениями по умолчанию:

```
vector<string> vec1(100);
list<double> list1(20);
```

3. Создать контейнер заданного размера и инициализировать его элементы указанным значением:

```
vector<string> vec1 (100, "Hello!");
deque<int> dec1(300, -1);
```

4. Создать контейнер и инициализировать его элементы значениями диапазона [first, last) элементов другого контейнера:

```
int arr[7] = {15, 2, 19, -3, 28, 6, 8};
vector<int> v1(arr, arr+7);
list<int> lst(v1.beg() + 2, v1.end());
```

5. Создать контейнер и инициализировать его элементы значениями элементов другого однотипного контейнера:

```
vector<int> v1;
// добавить в v2 элементы
```

```
vector<int> v2(v1);
```

6. Для вставки и удаления последнего элемента контейнера любого из трех рассматриваемых классов предназначены методы push_back() и pop_back(). Кроме того, список и очередь (но не вектор) поддерживают операции вставки и удаления первого элемента контейнера push_front() и pop_front(). Учтите, что методы pop_back() и pop_front() не возвращают удаленное значение. Чтобы считать первый элемент, используется метод front(), а для считывания последнего элемента — метод back(). Кроме этого, все типы контейнеров имеют более общие операции вставки и удаления, перечисленные в табл. 2.4.

Таблица 2.4. — Методы insert() и erase()

Метод	Пояснение
insert (iterator position, const T& value)	Вставка элемента со значением value в позицию, заданную итератором position
insert (iterator position, size_type n, const T& value)	Вставка n элементов со значением value, начиная с позиции position
template <class InputIter> void insert(iterator position, InputIter first, InputIter last)	Вставка диапазона элементов, заданного итераторами first и last, начиная с позиции position
erase(iterator position)	Удаление элемента, на который указывает итератор position
erase(iterator first, iterator last)	Удаление диапазона элементов, заданного позициями first и last

2.7. Использование ассоциативных контейнеров

В ассоциативных контейнерах элементы не выстроены в линейную последовательность. Они организованы в более сложные структуры, что дает большой выигрыш в скорости поиска. Поиск производится с помощью ключей, обычно представляющих собой одно числовое или строковое значение.

В множестве (set) хранятся объекты, упорядоченные по некоторому ключу, являющемуся атрибутом самого объекта. Например, множество может хранить объекты класса Man, упорядоченные в алфавитном порядке по значению ключевого поля name. Если в множестве хранятся значения одного из встроенных типов, например int, то ключом является сам элемент.

Словарь (map) можно представить себе как таблицу из двух столбцов, в первом из которых хранятся объекты, содержащие ключи, а во втором — объекты, содержащие значения.

И в множествах, и в словарях все ключи являются уникальными (только одно значение соответствует ключу). Мультимножества (multiset) и мультисловари

(multimap) аналогичны своим родственным контейнерам, но в них одному ключу может соответствовать несколько значений.

Ассоциативные контейнеры имеют много общих методов с последовательными контейнерами. Тем не менее некоторые методы, а также алгоритмы характерны только для них.

3. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

1. В ходе самостоятельной подготовки изучить основы работы с контейнерами.
2. Разработать программу на языке C++ согласно выданному варианту.
3. Разработать тестовые примеры и выполнить отладку программы.
4. Получить результаты работы программы и исследовать её свойства для различных режимов работы, сформулировать выводы.
5. Оформить отчет по проделанной работе.

4. ВАРИАНТЫ ЗАДАНИЙ

Вариант 1

Написать программу, моделирующую управление каталогом в файловой системе. Для каждого файла в каталоге содержатся следующие сведения: имя файла, дата создания, количество обращений к файлу.

Программа должна обеспечивать:

– начальное формирование каталога файлов;

– вывод каталога файлов;

– удаление файлов, дата создания которых раньше заданной;

– выборку файла с наибольшим количеством обращений.

Выбор моделируемой функции должен осуществляться с помощью меню. Для представления каталога использовать контейнерный класс list из STL.

Вариант 2

Написать программу моделирования работы автобусного парка. Сведения о каждом автобусе содержат: номер автобуса, фамилию и инициалы водителя, номер маршрута.

Программа должна обеспечивать выбор с помощью меню и выполнение следующих функций:

– начальное формирование данных о всех автобусах в парке в виде списка (ввод с клавиатуры или из файла);

– имитация выезда автобуса из парка: вводится номер автобуса; программа удаляет данные об этом автобусе из списка автобусов, находящихся в парке, и записывает эти данные в список автобусов, находящихся на маршруте;

– имитация въезда автобуса в парк: вводится номер автобуса; программа удаляет данные об этом автобусе из списка автобусов, находящихся на маршруте, и записывает эти данные в список автобусов, находящихся в парке;

– вывод сведений об автобусах, находящихся в парке, и об автобусах, находящихся на маршруте.

Для представления необходимых списков использовать контейнерный класс list.

Вариант 3

Написать программу учета заявок на авиабилеты. Каждая заявка содержит: пункт назначения, номер рейса, фамилию и инициалы пассажира, желаемую дату вылета.

Программа должна обеспечивать выбор с помощью меню и выполнение следующих функций:

– добавление заявок в список;

– удаление заявок;

– вывод заявок по заданному номеру рейса и дате вылета;

– вывод всех заявок.

Для хранения данных использовать контейнерный класс list.

Вариант 4

Написать программу учета книг в библиотеке. Сведения о книгах содержат: фамилию и инициалы автора, название, год издания, количество экземпляров данной книги в библиотеке.

Программа должна обеспечивать выбор с помощью меню и выполнение следующих функций:

– добавление данных о книгах, вновь поступающих в библиотеку;

– удаление данных о списываемых книгах;

– выдача сведений о всех книгах, упорядоченных по фамилиям авторов;

– выдача сведений о всех книгах, упорядоченных по годам издания.

Хранение данных организовать с применением контейнерного класса multimap, в качестве ключа использовать «фамилию и инициалы автора».

Вариант 5

Написать программу «Моя записная книжка». Предусмотреть возможность работы с произвольным числом записей, поиска записи по какому-либо признаку (например, по фамилии, дате рождения или номеру телефона), добавления и удаления записей, сортировки по разным полям.

Хранение данных организовать с применением контейнерного класса map или multimap.

Вариант 6

Написать программу учета заявок на обмен квартир и поиска вариантов обмена. Каждая заявка содержит сведения о двух квартирах: требуемой (искомой) и имеющейся. Сведения о каждой квартире содержат: количество комнат, площадь, этаж, район.

Программа должна обеспечивать выбор с помощью меню и выполнение следующих функций:

– ввод заявки на обмен;

– поиск в картотеке подходящего варианта: при совпадении требований и предложений по количеству комнат и этажности и различии по показателю «площадь»

в пределах 10% выводится соответствующая карточка и удаляется из списка, в противном случае поступившая заявка включается в картотеку;

– вывод всей картотеки.

Для хранения данных картотеки использовать контейнерный класс list.

Вариант 7

Написать программу «Автоматизированная информационная система на железнодорожном вокзале». Информационная система содержит сведения об отправлении поездов дальнего следования. Для каждого поезда указывается: номер поезда, станция назначения, время отправления.

Программа должна обеспечивать выбор с помощью меню и выполнение следующих функций:

– первоначальный ввод данных в информационную систему (с клавиатуры или из файла);

– вывод сведений по всем поездам;

– вывод сведений по поезду с запрошенным номером;

– вывод сведений по тем поездам, которые следуют до запрошенной станции назначения.

Хранение данных организовать с применением контейнерного класса vector.

Вариант 8

Написать программу «Англо-русский и русско-английский словарь». «База данных» словаря должна содержать синонимичные варианты перевода слов.

Программа должна обеспечивать выбор с помощью меню и выполнение следующих функций:

– Загрузка «базы данных» словаря (из файла).

– Выбор режима работы: “англо-русский” или “русско-английский”.

– Вывод вариантов перевода заданного английского слова.

– Вывод вариантов перевода заданного русского слова.

Базу данных словаря реализовать в виде двух контейнеров типа map.

Вариант 9

Составить программу учета заявок на авиабилеты. Каждая заявка содержит: пункт назначения, номер рейса, фамилию и инициалы пассажира, желаемую дату вылета.

Программа должна обеспечивать выбор с помощью меню и выполнение следующих функций:

– добавление заявок в список;

– удаление заявок;

– вывод заявок по заданному номеру рейса и дате вылета;

– вывод всех заявок, упорядоченных по пунктам назначения;

– вывод всех заявок, упорядоченных по датам вылета.

Хранение данных организовать с применением контейнерного класса multimap, в качестве ключа использовать «пункт назначения».

Вариант 10

Написать программу учета книг в библиотеке. Сведения о книгах содержат: фамилию и инициалы автора, название, год издания, количество экземпляров данной книги в библиотеке.

Программа должна обеспечивать выбор с помощью меню и выполнение следующих функций:

– добавление данных о книгах, вновь поступающих в библиотеку;

– удаление данных о списываемых книгах;

– выдача сведений о всех книгах, упорядоченных по фамилиям авторов;

– выдача сведений о всех книгах, упорядоченных по годам издания.

Хранение данных организовать с применением контейнерного класса vector.

Вариант 11

Написать программу «Моя записная книжка». Предусмотреть возможность работы с произвольным числом записей, поиска записи по какому-либо признаку (например, по фамилии, дате рождения или номеру телефона), добавления и удаления записей, сортировки по разным полям.

Хранение данных организовать с применением контейнерного класса list.

Вариант 12

Написать программу учета заявок на обмен квартир и поиска вариантов обмена. Каждая заявка содержит фамилию и инициалы заявителя, а также сведения о двух квартирах: требуемой (искомой) и имеющейся. Сведения о каждой квартире содержат: количество комнат, площадь, этаж, район.

Программа должна обеспечивать выбор с помощью меню и выполнение следующих функций:

– ввод заявки на обмен;

– поиск в картотеке подходящего варианта: при совпадении требований и предложений по количеству комнат и этажности и различии по показателю «площадь» в пределах 10% выводится соответствующая карточка и удаляется из списка, в противном случае поступившая заявка включается в картотеку;

– вывод всей картотеки.

Хранение данных организовать с применением контейнерного класса set.

Вариант 13

Написать программу «Автоматизированная информационная система на железнодорожном вокзале».

Информационная система содержит сведения об отправлении поездов дальнего следования. Для каждого поезда указывается: номер, станция назначения, время отправления. Программа должна обеспечивать выбор с помощью меню и выполнение следующих функций:

– первоначальный ввод данных в информационную систему (с клавиатуры или из файла); вывод сведений по всем поездам;

– вывод сведений по поезду с запрошенным номером;

– вывод сведений по тем поездам, которые следуют до запрошенной станции назначения.

Хранение данных организовать с применением контейнерного класса set.

Вариант 14

Написать программу «Англо-русский и русско-английский словарь». «База данных» словаря должна содержать синонимичные варианты перевода слов.

Программа должна обеспечивать выбор с помощью меню и выполнение следующих функций:

– Загрузка «базы данных» словаря (из файла).

– Выбор режима работы: “англо-русский” или “русско-английский”.

– Вывод вариантов перевода заданного английского слова.

– Вывод вариантов перевода заданного русского слова.

Базу данных словаря реализовать в виде двух контейнеров типа set.

Вариант 15

Составить программу моделирования работы автобусного парка.

Сведения о каждом автобусе содержат: номер автобуса, фамилию и инициалы водителя, номер маршрута.

Программа должна обеспечивать выбор с помощью меню и выполнение следующих функций:

– начальное формирование данных о всех автобусах в парке в виде списка (ввод с клавиатуры или из файла);

– имитация выезда автобуса из парка: вводится номер автобуса; программа удаляет данные об этом автобусе из списка автобусов, находящихся в парке, и записывает эти данные в список автобусов, находящихся на маршруте;

– имитация въезда автобуса в парк: вводится номер автобуса; программа удаляет данные об этом автобусе из списка автобусов, находящихся на маршруте, и записывает эти данные в список автобусов, находящихся в парке;

– вывод сведений об автобусах, находящихся в парке, и об автобусах, находящихся на маршруте, упорядоченных по номерам автобусов;

– вывод сведений об автобусах, находящихся в парке, и об автобусах, находящихся на маршруте, упорядоченных по номерам маршрутов.

Хранение всех необходимых списков организовать с применением контейнерного класса map, в качестве ключа использовать «номер автобуса».

5. СОДЕРЖАНИЕ ОТЧЕТА О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Титульный лист, цель работы, вариант задания, текст программы с комментариями, описание тестовых примеров и выводы по проделанной работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое контейнер? Назовите виды контейнеров.
2. Перечислите свойства контейнеров.
3. Что такое итераторы?
4. Объясните термин “обобщенные алгоритмы”.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Буч Г. Объектно-ориентированное проектирование с примерами применения / Г.Буч. — М.: Конкорд, 1992. — 519 с.

2. Дейтел Х. М. Как программировать на C++ (полное издание) / Х. М. Дейтел, П. Дж. Дейтел.; под ред. В.В.Тимофеева. — М.: Бином, 2008. — 1454 с.

3. Айра Пол. Объектно-ориентированное программирование на C++ /Айра Пол. — СПб: “Невский диалект”, 2001. — 464 с.

4. Павловская Т.А. Программирование на языке высокого уровня C/C++ / Т.А. Павловская. — Питер, 2003. — 461с.

5. Лаптев В.В. C++. Объектно-ориентированное программирование. Задачи и упражнения. / В.В. Лаптев, А.В. Морозов, А.В. Бокова.; под ред. В.В. Лаптева. — СПб.: Питер, 2007. — 288с.

6. Иванова Г.С Объектно-ориентированное программирование: учеб для студ. вузов/ Г.С. Иванова, Т.Н. Ничушкина, Е.К. Пугачев; под ред. Г.С.Ивановой. — М.: Изд-во МГТУ им. Н.Э.Баумана, 2001. — 320 с.

Заказ № _____ от «_____» _____2013г. Тираж _____экз.

Изд-во СевНТУ