Министерство образования и науки Российской Федерации
Ярославский государственный университет
им. П. Г. Демидова
Кафедра вычислительных и программных систем

Н. С. Лагутина

C++. Примеры и задачи

Практикум

Рекомендовано
Научно–методическим советом университета
для студентов, обучающихся по направлениям Прикладная математика и информатика, Фундаментальная информатика и информационные технологии

Ярославль 2011

УДК 004.5
ББК 3 973.2 — 018.1я73
Л 14

Рекомендовано
Редакционно-издательским советом университета в качестве учебного издания. План 2010/2011 учебного года

Рецензент
кафедра вычислительных и программных систем

Лагутина, Н. С. C++. Примеры и задачи: практикум/
Л 14 Н. С. Лагутина; Яросл. гос. ун–т им. П. Г. Демидова. — Ярославль: ЯрГУ, 2011. — 48 с.

Практикум содержит материалы для изучения объектно ориентированной технологии программирования на C++. Основные конструкции C++ и его стандартной библиотеки рассмотрены на примерах и сопровождаются заданиями для лабораторных работ.

Предназначен для студентов I и II курсов факультета ИВТ ЯрГУ, обучающихся по направлениям 010400 Прикладная математика и информатика, 010300 Фундаментальная информатика и информационные технологии (дисциплины«Основы программирования», «Высокоуровневое программирование», «Языки и методы программирования», профессиональный цикл), очной формы обучения. Может быть полезен для преподавателей, ведущих занятия по программированию.

Библиогр.: 7 назв.

УДК 004.5
ББК 3 973.2 — 018.1я73

Введение

Практикум обеспечивает проведение лабораторных занятий по программированию и является дополнением к учебному пособию Н. С. Лагутиной «Объектно ориентированное программирование на C++».

Каждая из первых пяти глав посвящена отдельной теме объектно ориентированной части C++. Реальные примеры демонстрируют особенности создания классов с описанием наиболее важных методов и приемов программирования. Подробно рассматривается использование некоторых классов из стандартной библиотеки C++, а именно классов ввода/вывода и обработки строк. Большое количество простых примеров позволяет студенту лучше усвоить основы такого сложного языка программирования, как C++, а также получить навыки создания объектно ориентированных программ.

В последней главе практикума помещены задания для лабораторных работ. Выполнение этих заданий на лабораторных занятиях под руководством преподавателя и во время самостоятельной работы студентов необходимо, чтобы научиться использовать язык C++ для создания пользовательских приложений.

1. Классы и объекты

Технология объектно ориентированного программирования является дальнейшим развитием идей структурного и процедурного (модульного) программирования. В ее основе лежит использование понятия «класс» для построения моделей предметной области. Класс – это тип, определяемый программистом, в котором объединяются структуры данных (поля класса) и функции их обработки (методы класса). Конкретные переменные типа данных «класс» называются экземплярами класса, или объектами.

Определение классов и объектов — одна из самых сложных задач объектно ориентированного проектирования. Понятия класса и объекта настолько тесно связаны, что невозможно говорить об объекте безотносительно к его классу. Однако существует важное различие этих двух понятий. В то время как объект обозначает конкретную сущность, определенную во времени и в пространстве, класс определяет лишь абстракцию существенного в объекте.

Таким образом, фундаментальные понятия объектно ориентированного программирования — «класс» и «объект», а также основные принципы: инкапсуляция, полиморфизм, наследование.

Инкапсуляция (encapsulation) – слияние данных и функций, работающих с этими данными в одном элементе программы (классе, структуре) одновременно с сокрытием отдельных деталей внутреннего устройства классов от внешних по отношению к ним объектов или пользователей. Это означает, что информация о данных и реализации работающих с ними методов не требуется для написания программы или модуля, использующих этот класс. Функциональность класса используется только через его интерфейс, определяемый прототипами методов, расположенных в открытой части класса.

Наследование (inheritance) – порождение нового объекта (или класса) на основе уже имеющегося. То есть, если родительский класс обладает фиксированным набором свойств

и поведения, то любой наследуемый от него класс (потомок) должен содержать тот же набор свойств и поведения, а возможно и иметь дополнительные, которые будут характеризовать его уникальность. Наследование структур данных вместе с поведением дает возможность подклассам совместно использовать общий код.

Полиморфизм (polymorphism) – перегрузка операций, функций, шаблоны классов, свойство одноименных методов или операций выполнять разные действия или обладать различным поведением в зависимости от того, к какому классу они относятся. Это значит, что одна и та же операция подразумевает разное поведение в разных классах. При вызове операции не нужно беспокоиться о том, сколько реализаций этой операции существует в системе. Полиморфизм перекладывает ответственность за выбор подходящей реализации с вызывающего кода на иерархию классов. Каждый объект выбирает подходящую процедуру согласно своему классу. Это облегчает поддержку программ, потому что добавление нового класса не требует изменения вызывающего кода. Полиморфизм позволяет переопределять в производном классе некоторые из методов базового, называемые виртуальными.

Рассмотрим пример создания класса для хранения и обработки данных о точке на плоскости. Программный код принято располагать в двух файлах. В первом заголовочном (он может называться так : `Tochka.h`) содержится объявление класса:

```
class Tochka {
public:
   Tochka():x(0),y(0) {}
   Tochka(double x1, double y1){x=x1;y=y1;}
   void SetX(double x1){x=x1;}
   void SetY(double y1){y=y1;}
   double GetX(){return x;}
   double GetY(){return y;}
   void Print();
```

```
   double dist(Tochka t);
private:
   double x,y;
};
```

а во втором записывается реализация методов (он может называться `Tochka.cpp`):

```
#include "Tochka.h"
#include <iostream>
#include <cmath>
void Tochka::Print() {
   std::cout<<"("<<x<<","<<y<<")"<<std::endl;
}
double Tochka::dist(Tochka t) {
   return sqrt((x-t.x)*(x-t.x)+(y-t.y)*(y-t.y));
}
```

Рассмотрим процесс создания класса подробно. Сначала надо определить поля класса, они должны содержать всю необходимую информацию об объекте и в то же время не включать в себя ничего лишнего. Точка на плоскости однозначно характеризуется двумя действительными координатами, назовем их `x` и `y`. Поля класса, как правило, помещают в закрытую часть класса.

В открытой части располагаются методы, относящиеся к интерфейсу класса. Именно они будут использоваться в других модулях программы при создании объектов этого класса и работе с ними. Эти методы определяют функциональность класса, т. е. то, что можно делать с объектами. В нашем примере – это в первую очередь – конструкторы: по умолчанию:

```
 Tochka():x(0),y(0) {}
```

и с параметрами:

```
Tochka(double x1, double y1){x=x1;y=y1;}
```

Затем идут методы, позволяющие изменять поля класса:

```
void SetX(double x1){x=x1;}
```

и

```
void SetY(double y1){y=y1;}
```

В такие методы можно поместить код, проверяющий корректность введенных параметров. Например, если координаты точки ограничены некоторыми значениями, код одного из методов может выглядеть так:

```
void SetX(double x1)  {
   if(x1>-10&&x1<10)
     x=x1;
}
```

Следующая пара методов позволяет получить значения координат точки (`double GetX() и double GetY()`).

Метод `void Print()` выводит координаты точки на консоль, а метод `double dist(Tochka t)` определяет расстояние между точками. Аргументом метода является только одна внешняя точка, так как в нем рассчитывается расстояние от текущего объекта до какого-то внешнего. Как правило, аргументы методов — это именно внешние объекты и переменные, необходимые для его работы с полями класса, определяющими параметры текущего объекта.

Ввод и вывод информации для объекта принято располагать в отдельных методах, а код, производящий вычисления и другие преобразования объекта, — в других.

Приведем пример создания и использования объектов класса `Tochka`:

```
#include "Tochka.h"
using namespace std;
int main() {
  Tochka a, b(2,5);
```

```
  a.SetY(3);
  a.SetX(-3);
  b.Print();
  a.Print();
  cout<<"Расстояние между точками равно"<<b.dist(a);
  return 0;
}
```

Здесь создаются два объекта класса `Tochka`: `a` с использованием конструктора по умолчанию, таким образом точка `a` получает координаты `(0,0)`, и `b` с использованием конструктора с параметрами, при этом мы получаем объект с координатами `(2,5)`.

Затем координаты точки `a` изменяются на `3` и `-3`. После чего параметры объектов выводятся на печать вызовом метода `void Print()`. Предпоследний оператор программы выводит на экран сообщение о расстоянии между точками.

Объекты одного класса могут быть полями другого. Это называется композицией классов. Создадим класс «прямоугольник на плоскости», параметрами которого являются концы его диагонали (две точки), будем считать, что стороны прямоугольника параллельны осям координат. В класс включим метод (`bool contains(Tochka& p);`), определяющий, попадает ли некоторая внешняя точка внутрь прямоугольника.

```
#include "Tochka.h"
class Rectangle {
private:
  Tochka p1, p2;
public:
  Rectangle(Tochka _p1, Tochka _p2) {p1 = _p1;p2 = _p2;};
  Rectangle(double x1, double y1, double x2, double y2);
  void SetP1(Tochka _p1) { p1 = _p1; }
  void SetP2(Tochka _p2) { p2 = _p2; }
  Tochka GetP1() { return p1; }
  Tochka GetP2() { return p2; }
```

```
  bool contains(Tochka& p);
  void Print();
};
Rectangle::Rectangle(double x1, double y1, double x2,
  double y2): p1(x1, y1), p2(x2, y2){}
bool Rectangle::contains(Tochka& p) {
  if(p1.GetX() <= p.GetX() && p.GetX() <= p2.GetX() ||
      p2.GetX() <= p.GetX() && p.GetX() <= p1.GetX())
    if(p1.GetY() <= p.GetY() && p.GetY() <= p2.GetY() ||
      p2.GetY() <= p.GetY() && p.GetY() <= p1.GetY())
        return true;
  return false;}
void Rectangle::Print() {
   p1.Print();
   cout<< " - ";
   p2.Print();
}
```

В главной функции программы создадим объекты – точки и прямоугольники и выведем на экран сообщение: попадает ли точка внутрь прямоугольника:

```
#include "Rectangle.h"
using namespace std;
int main() {
  Tochka a, b(2,5);
  a.SetY(3);
  a.SetX(-3);
  Rectangle r1(a,b), r2(Tochka(9,7),b);
  if(r2.contains(a)==true)
     cout<<"Да!"<<endl;
  else
     cout<<"Нет!"<<endl;
  return 0;
}
```

2. Методы класса

Методы класса определяют поведение объектов этого класса. При создании и уничтожении объекта автоматически вызываются специальные методы класса: конструктор и деструктор. Конструктор формально отличается от других методов класса тем, что его имя совпадает с именем класса. Имя деструктора образуется путем добавления символа ~(тильда) в начало имени класса. Конструктор и деструктор являются открытыми методами класса. Деструктор может быть только один, а конструкторов несколько. Выделяют несколько важных для работы с классом и его объектами конструкторов: конструктор по умолчанию, конструктор копирования, конструкторы с параметрами.

Другие важные методы класса — перегруженные операции. Они позволяют при создании модулей программы использовать удобный и привычный синтаксис для объектов созданных классов, к тому же это облегчает понимание кода программы.

Рассмотрим пример создания класса, хранящего данные о двумерном массиве. Полями класса являются размеры этого массива и указатель на хранилище данных. Заголовочный файл `Matrix.h`:

```
#include <iostream>
class Matrix {
private:
// количество строк и столбцов
  int rows, cols;
// хранилище данных - указатель на двумерный массив
  double** a;
public:
// конструктор с параметрами
  Matrix(int _rows, int _cols);
// конструктор для создания квадратной матрицы
  Matrix(int _rows);
```

```
// конструктор копирования
  Matrix(const Matrix& m);
// деструктор
  ~Matrix();
// метод, возвращающий (i,j)-й элемент хранимого массива
  double GetElement(int i, int j);
// метод, изменяющий значение (i,j)-го элемента массива a
  void SetElement(int i, int j, double x);
// метод, возвращающий количество строк
  int GetRows() { return rows; }
// метод, возвращающий количество столбцов
  int GetCols() { return cols; }
// переопределение операции включения данных в поток
  friend std::ostream& operator<<(std::ostream& os,
  const Matrix& m);
// переопределение операции сложения матриц
  Matrix operator+(const Matrix& m) const;
// переопределение операции +=
  Matrix& operator+=(const Matrix& m);
// переопределение операции присваивания
  Matrix& operator=(const Matrix& m);
};
```

Описание методов располагаем в отдельном файле Matrix.cpp:

```
#include "Matrix.h"
Matrix::Matrix(int _rows, int _cols) {
  rows = _rows;
  cols = _cols;
  a = new double*[rows];
  for(int i = 0; i < rows; ++i)
    a[i] = new double[cols];
}
Matrix::Matrix(int _rows) {
  rows = cols = _rows;
```

```
  a = new double*[rows];
  for(int i = 0; i < rows; ++i)
    a[i] = new double[cols];
}
Matrix::Matrix(const Matrix& m) {
  rows = m.rows;
  cols = m.cols;
  a = new double*[rows];
  for(int i = 0; i < rows; ++i) {
    a[i] = new double[cols];
    for(int j = 0; j < cols; ++j)
      a[i][j] = m.a[i][j];
  }
}
Matrix& Matrix::operator=(const Matrix& m) {
  if(&m==this) return *this;
  for(int i = 0; i < rows; ++i)
    delete[] a[i];
  delete[] a;
  rows = m.rows;
  cols = m.cols;
  a = new double*[rows];
  for(int i = 0; i < rows; ++i) {
    a[i] = new double[cols];
    for(int j = 0; j < cols; ++j)
      a[i][j] = m.a[i][j];
  }
  return *this;
}
Matrix::~Matrix() {
  for(int i = 0; i < rows; ++i)
    delete[] a[i];
  delete[] a;
}
double Matrix::GetElement(int i, int j)
```

```
{  return a[i][j]; }
void Matrix::SetElement(int i, int j, double x)
{  a[i][j] = x; }
std::ostream& operator<<(std::ostream& os,
  const Matrix& m) {
  for(int i = 0; i < m.rows; ++i) {
    for(int j = 0; j < m.cols; ++j)
      os << m.a[i][j] << " ";
    os << std::endl;
  }
  return os;
}
Matrix Matrix::operator+(const Matrix& m) const {
  Matrix res(rows, cols);
  for(int i = 0; i < rows; ++i)
    for(int j = 0; j < cols; ++j)
      res.a[i][j] = a[i][j] + m.a[i][j];
  return res;
}
Matrix& Matrix::operator+=(const Matrix& m) {
  for(int i = 0; i < rows; ++i)
    for(int j = 0; j < cols; ++j)
      a[i][j] += m.a[i][j];
  return *this;
}
```

Обратим внимание на конструктор копирования и перегруженную операцию присваивания. Если создание объекта предусматривает динамическое выделение памяти, то почти всегда необходимо определить эти методы класса, так как иначе их создаст по умолчанию компилятор, формально скопировав только поля класса (в данном примере – это размеры массива и указатель).

Любая операция в языке C++ рассматривается как функция, аргументами которой служат операнды, а возвращаемым значением — результат. Если операция переопределяется как

метод класса, текущий объект класса рассматривается как левый операнд. Для операции вставки в поток левым операндом является ссылка на поток вывода, а не матрица, поэтому операция может быть переопределена только как внешняя дружественная (чтобы обеспечить возможность работы с членами класса) функция.

У бинарных операций + и += левым операндом является объект матрица, поэтому они переопределяются как методы класса и единственным аргументом служит внешняя матрица, которая становится правым операндом. Результатом этих операций является ссылка на объект класса `Matrix`, хранящий в своем двумерном массиве результат сложения. По смыслу операции += это ссылка на текущий объект.

Рассмотрим пример работы с объектами созданного класса. В нем создадим две квадратные матрицы, заполним одну единицами, другую – двойками и выведем на экран результат сложения этих матриц:

```
#include "Matrix.h"
using namespace std;
int main(int argc, char* argv[])
{
  Matrix m1(3);
  Matrix m2(3);
  for(int i = 0; i < m1.GetRows(); ++i)
    for(int j = 0; j < m1.GetCols(); ++j)
      m1.SetElement(i, j, 1);
  for(int i = 0; i < m2.GetRows(); ++i)
    for(int j = 0; j < m2.GetCols(); ++j)
      m2.SetElement(i, j, 2);
  cout << m1 << endl;
  cout << m2 << endl;
  cout << m1 + m2 << endl;
  m1 += m2;
  cout << m1 << endl;
  return 0;}
```

3. Потоки ввода–вывода

Ввод и вывод в C++ основан на концепции потоков. Поток – это последовательность байтов. При вводе читаются байты из потока ввода, при выводе вставляются байты в поток вывода. Понятие потока позволяет абстрагироваться от того, с каким устройством идет работа. Например, байты потока ввода могут поступать с клавиатуры или из файла на диске. Обычно ввод/вывод осуществляется через буфер – область оперативной памяти, накапливающую какое-то количество байтов, прежде чем они пересылаются по назначению. При работе программы на C++ буфер ввода обычно очищается при нажатии клавиши `Enter`. При выводе очистка буфера происходит либо при появлении в потоке символа перехода на новую строку `\n`, либо при выполнении программой очередной операции ввода с клавиатуры. Потоки бывают неформатированные и форматированные. В первом случае передача информации осуществляется блоками байтов данных без какого-либо преобразования. Во втором – байты группируются для представления различных элементов данных: целых чисел, чисел с плавающей точкой, символов, строк и т. д.

Библиотека `<iostream>` предоставляет объектно ориентированную версию потоков. Стандартному потоку ввода соответствует класс `istream`, стандартному потоку вывода — класс `ostream`. Оба класса — наследники класса `ios`. Следующим в иерархии является класс `iostream`, наследующий классы `istream` и `ostream`.

При наличии директивы `#include<iostream>` в программе автоматически становятся доступны объекты

- `cin` — объект класса `istream`, соответствующий стандартному потоку ввода;
- `cout` — объект класса `ostream`, соответствующий стандартному потоку вывода.

Форматированный ввод/вывод реализуется через две перегруженные операции: `<<` — операция вставки (включения,

помещения) в поток, >> — операция извлечения из потока. Как и для других операций следует обращать внимание на приоритет. Например, арифметические операции имеют больший приоритет и можно написать так: `cout<<i+j;`

А операции сравнения – меньший, поэтому необходимо использовать круглые скобки: `cout<<(i<j);`

Если форматирование данных по умолчанию не подходит для решения задачи, его можно изменить при помощи методов классов, этой же цели служат специальные функции класса `ios`, называемые манипуляторами, которые можно включать в поток. Манипуляторы подразделяются на простые (без аргументов) и параметризованные (с аргументом), чтобы использовать параметризованные манипуляторы, надо подключить заголовочный файл `iomanip`. За отображение данных отвечают флаги форматирования, которые изменяются описанными функциями, кроме того, любой флаг можно установить в помощью метода `flags()`:

- манипулятор `endl` включает в поток символ новой строки и очищает буфер (`cout<<endl;`)
- манипулятор `setprecision(n)` или метод `precision(n)` устанавливают количество отображаемых знаков (`cout<<setprecision(6)<<x;` или `cout.precision(6); cout<<x;`)
- манипулятор `setw(n)` или метод `width(n)` устанавливают ширину поля вывода;
- манипулятор `hex` или метод `flags(ios::hex)` позволяют перейти в шестнадцатеричную систему счисления;
- манипулятор `dec` или метод `flags(ios::dec)` позволяют перейти в десятичную систему счисления.

Большинство параметров форматирования сохраняют своё состояние до следующего вызова функций, изменяющих это состояние. Исключением является только манипулятор `setw(n)` и соответствующий ему метод `width(n)`, которые распространяются только на ближайшую операцию вывода.

Указывая количество отображаемых знаков, мы задаем количество цифр после запятой. Формат вывода чисел с плавающей точкой зависит от флагов `ios::scientific` и `ios::fixed`, которые можно установить методом `flags()`.

Рассмотрим примеры изменения формата вывода данных.

Чтобы вывести число в шестнадцатеричном виде, можно использовать метод:

```
int i=123;
cout.flags(ios::hex);
cout<<i<<endl;
```

или манипулятор:

```
int i=123;
cout<<hex<<i<<endl;
```

Чтобы вывести число в экспоненциальной форме с двумя знаками после запятой можно применить такие операторы:

```
int i=12;
cout.precision(2);
cout.flags(ios::scientific);
cout<<i.13<<endl;
```

или:

```
int i=12;
cout.flags(ios::scientific);
cout<<setprecision(2)<<i.13<<endl;
```

Для отслеживания ошибок в базовом классе `ios` определено поле `state`, отдельные биты (флаги) которого отображают состояние потока. Для анализа состояния потока более удобно использовать методы:

- флаг `ios::goodbit` — нет ошибок, следующая операция с потоком может выполняться, метод `bool good()const` возвращает значение этого флага;

- флаг `ios::eofbit` — достигнут конец ввода (например, файла), соответствующий метод —`bool eof()const`, возвращающий истинное значение в этом случае;
- флаг `ios::failbit` — ошибка форматирования или преобразования, следующая операция не выполнится, соответствующий метод —`bool fail()const`, возвращающий истинное значение в случае ошибки;
- флаг `ios::badbit` — серьезная ошибка, поток поврежден, операции с ним невозможны, соответствующий метод —`bool bad()const`.

Установить любой флаг можно методом `void setstate(iostate state)`. Сбросить все флаги в ноль можно с помощью метода `clear()`. Проще всего проверить состояние потока можно с помощью обычного логического выражения `if(!cin)`.

Рассмотрим пример программы, демонстрирующей ввод целого числа с проверкой ошибок. После ввода данных проверяется флаг `goodbit`, если все правильно, пропускаются 10 или меньше символов до `\n` (`cin.ignore(10,"\n");`), символ конца строки также исключается из потока, если пользователь вводит посторонние символы вместо цифр, в потоке происходит ошибка форматирования, после этого в программе восстанавливается флаг `goodbit` (`cin.clear();`), посторонние символы (10 или меньше) извлекаются из потока и ввод повторяется. После этих действий вводится символ, и, наконец, программа сообщает о введенных данных:

```
int main(int argc, char** argv) {
    int i; char c;
    while(1){
        cout<<"Введите число: ";
        cin>>i;
        if(cin.good()){
           cin.ignore(10,'\n');
           break;
        }
```

```
        cin.clear();
        cout<<"err"<<endl;
        cin.ignore(10,'\n');
    }
    cout<<"Введите символ: ";
    cin>>c;
    cout<<"Вы ввели "<<i<<" и "<<c<<endl<<(i<5)<<endl;
    return 0;
}
```

Кроме стандартных, есть ещё файловые и строковые потоки. Для их использования в программе следует объявлять объекты соответствующих классов.

Для использования файловых потоков необходимо подключить заголовочный файл `fstream`, тогда можно объявлять объекты–потоки трёх видов: входной `ifstream`, выходной `ofstream`, двунаправленный `fstream`. Эти классы являются потомками классов `istream`, `ostream` и `iostream` соответственно, поэтому они наследуют все методы указанных классов, перегруженные операции вставки и извлечения, манипуляторы, флаги состояния и т. д.

Работа с файлом предполагает следующие действия:

- создание потокового объекта;
- открытие потока и связывание его с файлом;
- обмен с потоком (ввод/вывод);
- закрытие файла.

Создать поток и связать его с файлом можно с помощью конструктора с параметрами:

```
ifstream(const char* name, int mode=ios::in)
ofstream(const char* name, int mode=ios::out|ios::trunc)
fstream(const char* name, int mode=ios::in|ios::out)
```

Первый параметр – имя файла, краткое, если файл расположен в текущем каталоге:

```
ifstream File(«test.txt»);
```

или полное:

```
ifstream File(«D:\\NetBeansProject\\lab2\\tests\\test.txt»);
```

Второй параметр конструктора задает режим открытия файла с помощью флагов:

Флаг	Режим открытия
ios::in	открыть файл для ввода
ios::out	открыть файл для вывода
ios::ate	установить указатель на конец файла
ios::app	открыть в режиме добавления данных в конец файла
ios::trunc	если файл существует, обрезать его до нулевой длины
ios::nocreate	если файл не существует, зафиксировать ошибку
ios::noreplace	если файл существует, зафиксировать ошибку
ios::binary	открыть файл в двоичном режиме

Если объект потока создан с помощью конструктора без параметров, связать его с конкретным файлом можно методом `open(const char* name, int mode)`. Рассмотрим пример открытия файла для дозаписи данных:

```
ofstream OutFile(«resalt.txt»,ios::app);
```

или:

```
ofstream OutFile;
OutFile.open(«resalt.txt»,ios::app);
```

Если файл по какой-либо причине не удается открыть (возникает ошибка), то потоковый объект становится равен нулю. Проверить это можно так:

```
ofstream OutFile(«resalt.txt»,ios::nocreate);
if(!OutFile){
  cout<< «Файл не существует или отсутствует»;
}
```

Если в процессе ввода/вывода произойдёт ошибка, то потоковый объект также станет равен нулю.

Конец файла определяется методом `eof()`. Следует обратить внимание, что в C++ после чтения из файла последнего элемента условие конца файла не возникает. Оно появляется только при попытке чтения следующих данных. Чтение последовательности чисел из файла может выглядеть так:

```
int main(int argc, char** argv) {
    int i;
    ifstream F("t.txt");
    while(1) {
       F>>i;
       if(F.eof())
          break;
       cout<<i<<" ";
    }
    F.close();
    return 0;
}
```

4. Контейнеры

В реальных задачах требуется обрабатывать группы данных большого объема. В процессе развития языков программирования, в частности C++, выработалась концепция объединения однородных данных в группу – контейнер (более функциональная и удобная, чем встроенный массив). *Контейнеры* — это объекты, содержащие другие однотипные объекты, организованные в одну из типовых структур данных: список, стек, очередь и т. п. Объекты, хранимые в контейнерах, могут быть как встроенных, так и пользовательских типов. Так как контейнер — это шаблонный класс, то необходимо, чтобы помещаемые в него объекты допускали копирование и присваивание, поэтому в пользовательском классе должны быть в открытой части конструктор копирования и операция присваивания.

Чтобы использовать контейнер в программе, необходимо включить в неё соответствующий заголовочный файл и задать тип сохраняемых объектов в аргументах шаблона. Например, для хранения набора строк можно создать список:

```
#include<list>
list <string> sentence;
```

Для всех контейнерных классов определён тип «итератор» — это более абстрактная сущность, чем указатель, но обладающая похожим поведением. Чтобы перемещаться по списку строк, можно создать объект-итератор:

```
list <string>::iterator i;
```

Доступ к элементам контейнера может осуществляться с помощью индекса. В C++ нумерацию принято начинать с нуля.

Интерфейс контейнерных классов предоставляет методы, которые обеспечивают доступ и замену элементов, добавления и удаления групп элементов, поиска и др.

Вывести на экран содержимое контейнера можно так:

```
list <string> sentence;
list <string>::iterator i;
for(i=sentence.begin();i!=sentence.end();i++)
cout<<*i<<endl;
```

или так:

```
list <string> sentence;
for(int i=0;i!=sentence.size();i++)
cout<<sentence[i]<<endl;
```

Важной частью стандартной библиотеки C++ являются *алгоритмы*, функции, выполняющие различные действия с элементами контейнера – сортировку, поиск и т. д. Стандартная библиотека предоставляет алгоритмы для выполнения большинства распространённых операций. В качестве первых двух параметров алгоритмов почти всегда выступают итераторы, определяющие диапазон элементов контейнера, к которым будет применён алгоритм. Тип итераторов определяется типом контейнера, для которого может быть использован алгоритм. Алгоритмы не проверяют выход за пределы последовательности!

Для демонстрации использования стандартной библиотеки C++ рассмотрим пример решения такой задачи: в файле записаны четверки чисел, определяющие координаты отрезка на плоскости. Необходимо определить длину каждого отрезка и найти отрезок максимальный по длине.

Для решения задачи создадим класс «отрезок», полями которого являются четыре числа – координаты начала и конца отрезка. В интерфейсе класса опишем метод, вычисляющий длину отрезка, и переопределим операцию < для использования стандартного алгоритма поиска максимума.

В основной функции программы откроем файл для чтения данных и прочитаем оттуда все четверки чисел, последовательно для каждой создадим соответствующий объект и поместим его в контейнер `vector`. Результаты работы программы выведем в файл и на экран.

Файл с описанием класса:

```
#include <iostream>
class Otrezok {
 private:
   double x1,y1,x2,y2;
 public:
   Otrezok (double _x1,double _y1,double _x2,double _y2){
     x1=_x1; y1=_y1; x2=_x2; y2=_y2;
   }
   friend std::ostream& operator<<(std::ostream& p,
     const Otrezok& t) {
   return p<<"("<<t.x1<<","<<t.y1<<")-("<<t.x2<<","<<
   t.y2<<")";
   }
   double dlina() const {
     return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));}
   bool operator<(const Otrezok& t) const {
     return this->dlina()<t.dlina();
   }
};
```

Файл с описанием главной функции программы:

```
#include <fstream>
#include <vector>
#include <algorithm>
#include "Otrezok.h"
using namespace std;
int main(int argc, char* argv[])
{
  ifstream dataFile(argv[1]);
  if(!dataFile) {
    cout<<"Error input file!"<<endl;
    return 0;
  }
  vector<Otrezok> a;
```

```
  while(!dataFile.eof()) {
    double x1,y1,x2,y2;
    dataFile>>x1>>y1>>x2>>y2;
  if(!dataFile) break;
    Otrezok x(x1,y1,x2,y2);
    a.push_back(x);
  }
  dataFile.close();
  ofstream outFile(argv[2]);
  if(!outFile) {
    cout<<"Error output file!"<<endl;
    return 0;
  }
  vector<Otrezok>::iterator i;
  for(i=a.begin();i!=a.end();i++) {
    cout<<*i<<":"<<i->dlina()<<endl;
    outFile<<*i<<":"<<i->dlina()<<endl;
  }
  vector<Otrezok>::iterator imax=max_element(a.begin(),
  a.end());
  cout<<"Maximum: "<<*imax<<endl;
  outFile<<"Maximum: "<<*imax<<endl;
  outFile.close();
  return 0;
}
```

5. Строки

Современная обработка данных во много ориентирована на работу с текстом. В языке C++ есть класс, называемый `string`, с помощью которого работа со строками символов становится удобнее, так как он имеет широкую функциональность и автоматически контролирует размер строки и выход за пределы выделенной памяти.

Класс `string` как стандартный библиотечный класс позволяет работать со строками, как с обычными типами данных, т. е. копировать, присваивать, складывать, сравнивать, используя привычный синтаксис соответствующих операций. Большое количество методов класса позволяют легко решать многочисленные задачи, связанные с обработкой текста.

Проиллюстрируем возможности этого класса небольшим примером решения следующей задачи: в файле содержится текстовая информация, необходимо подсчитать количество вхождения в файл набора символов, задаваемого пользователем (набор символов вводится с клавиатуры в начале работы программы), при этом игнорируются некоторые символы (в нашем примере – это пробел и знаки препинания).

Для решения задачи создан класс для обработки строки, в котором определен метод, исключающий из строки игнорируемые символы `void ud_sim(const string str)`, и метод, подсчитывающий количество вхождений заданной строки (первый аргумент метода) с учетом игнорируемых символов (второй аргумент метода) — `int kol_find_str(const MyStr t, const string str)`.
Приведем код примера полностью:

```
class MyStr{
private:
  string s;
public:
  MyStr(){s="";}
  MyStr(string _s){s=_s;}
  string GetS(){return s;}
```

```
  friend std::ostream& operator<<(std::ostream& p,
    const MyStr& t);
  void ud_sim(const string str);
  int kol_find_str(const MyStr t, const string str);
};
ostream& operator<<(ostream& p,const MyStr& t){
  return p<<t.s;
}
void MyStr::ud_sim(const string str){
  string::size_type p=s.find_first_of(str);
  while(p!=-1){
    s.erase(p,1);
    p=s.find_first_of(str);
  }
}
int MyStr::kol_find_str(const MyStr t, const string str){
  int kol=0;
  MyStr tmp=t;
  tmp.ud_sim(str);
  string::size_type p=s.find(tmp.GetS());
  while(p!=-1){
    kol++;
    p=s.find(tmp.GetS(),p+1);
  }
  return kol;
}
int main(int argc, char* argv[]){
  cout<<"Введите слово:"<<endl;
  string tmp; cin>>tmp;
  MyStr s1(tmp);
  ifstream textfile(argv[1]);
  if(!textfile) {
    cout<<"Ошибка открытия файла!"<<endl;
  return 0;
  }
```

```
  int k=0;
  while(!textfile.eof()) {
    getline(textfile,tmp);
    if(!textfile)
       break;
    MyStr x(tmp);
    x.ud_sim(" ,.?!");
    k=k+x.kol_find_str(sl," ,.?!");
  }
  cout<<sl<<" встречается в файле "<<argv[1]<<" "<<
  k<<" раз"<<endl;
  return 0;
}
```

6. Задания для лабораторных работ

6.1. Классы. Объекты. Методы

Имена входного и выходного файлов передаются через аргументы командной строки. Если имя выходного файла не задано, вывод осуществляется на экран. Если не задано имя входного файла, программа должна печатать справку об использовании и завершать свою работу. Все ошибочные ситуации должны корректно обрабатываться программой!

№ А 1.1. Создайте класс «дробь». Данные класса должны быть представлены двумя полями: числителем и знаменателем. Класс должен содержать конструктор по умолчанию, конструктор, инициализирующий числитель и знаменатель дроби заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод выводящий дробь на экран в виде 5/3. Другие методы класса должны реализовывать сложение, вычитание, умножение, деление, сравнение дробей (дополнительные методы: сокращение дроби, выделение целой части – результат распечатывается).

1. В функции main() создайте два объекта разработанного класса и выведите на консоль разность этих двух дробей.

2. Входной файл содержит произвольное количество пар чисел: числитель и знаменатель дроби. Программа должна читать описания дробей из входного файла и выводить в выходной файл: минимум среди всех дробей, максимум, общую сумму всех дробей.

№ А 1.2. Создайте класс «комплексное число». Данные класса должны быть представлены двумя полями: действительной и мнимой частью. Класс должен содержать конструктор по умолчанию, конструктор, инициализирующий поля заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий число на экран в виде 5+3i. Другие методы

класса должны реализовывать сложение, вычитание, умножение, деление, сравнение комплексных чисел, определение модуля комплексного числа. (Дополнительные методы: вывод комплексного числа в тригонометрической и экспоненциальной форме).

1. В функции main() создайте два объекта разработанного класса и выведите на консоль разность этих двух чисел.

2. Входной файл содержит произвольное количество пар чисел: действительную и мнимую часть комплексного числа. Программа должна читать описания комплексных чисел из входного файла и выводить в выходной файл: модуль каждого числа, общую сумму всех чисел, всевозможные попарные произведения.

№ А 1.3. Создайте класс «вектор целых чисел». Данные класса должны быть представлены массивом чисел фиксированной длины. Класс должен содержать конструктор по умолчанию (инициализация нулевым вектором), конструктор, инициализирующий массив заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий вектор на экран. Другие методы класса должны реализовывать сложение, вычитание, скалярное произведение векторов. (Дополнительные методы: определение линейной зависимости пары и любого количества векторов).

1. В функции main() создайте два объекта разработанного класса и выведите на консоль скалярное произведение этих двух векторов.

2. Входной файл содержит произвольное количество векторов. Программа должна читать векторы из входного файла и выводить в выходной файл: модуль каждого вектора, общую сумму всех векторов, всевозможные попарные произведения.

№ А 1.4. Создайте класс «трехмерная точка». Данные класса должны быть представлены тремя полями: координатами точки в трехмерном пространстве. Класс должен содержать конструктор по умолчанию, конструктор, инициали-

зирующий координаты точки заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий точку на экран в виде (1,-2,3). Создайте класс «параллелепипед», задаваемый тремя размерами и тремя координатами вершины параллелограмма или тремя размерами и точкой (два конструктора), данные класса должны быть представлены тремя размерами и тремя координатами вершины параллелограмма. Другие методы этого класса должны определять, находится заданная точка внутри, вне или на границе параллелепипеда. 1. В функции main() создайте два объекта разработанных классов и выведите на консоль сообщение – находится ли данная точка внутри параллелепипеда.

2. Входной файл содержит описание произвольного количества параллелограммов: шесть действительных чисел. Программа должна читать описания параллелограммов из входного файла и выводить в выходной файл объём каждого параллелограмма.

№ А 1.5. Создайте класс «точка на плоскости». Данные класса должны быть представлены двумя полями: координатами точки. Класс должен содержать конструктор по умолчанию, конструктор, инициализирующий координаты точки заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий точку на экран в виде (1,-2). Создайте класс «прямая», задаваемый двумя точками или четырьмя координатами этих точек (два конструктора). Данные класса должны быть представлены четырьмя координатами точек, определяющих прямую. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий на экран уравнение прямой. Другой метод класса должен определять, находится ли заданная точка на прямой.

1. В функции main() создайте два объекта разработанных классов и выведите на консоль сообщение – находится ли данная точка на прямой.

2. Первый входной файл содержит описание произвольного количества прямых: четыре действительных числа, определяющих две пары точек. Второй входной файл содержит набор точек. Программа должна читать описания прямых из первого входного файла и выводить в выходной файл для каждой прямой: уравнение, точки из второго файла, лежащие на ней.

№ А 1.6. Создайте класс «точка на плоскости». Данные класса должны быть представлены двумя полями: координатами точки. Класс должен содержать конструктор по умолчанию, конструктор, инициализирующий координаты точки заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий точку на экран в виде (1,-2). Создайте класс «вектор на плоскости», задаваемый двумя точками или четырьмя координатами этих точек (два конструктора). Данные класса должны быть представлены четырьмя полями: координатами начала и конца вектора. Класс должен содержать конструктор по умолчанию, конструктор, инициализирующий координаты точек заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий вектор на экран в виде (1,0). Другие методы класса должны определять длину вектора, скалярное произведение двух векторов, угол между векторами, векторное произведение, разложение вектора по единичным векторам.

1. В функции main() создайте два объекта разработанного класса «вектор» и выведите на консоль скалярное произведение этих двух векторов.

2. Входной файл содержит произвольное количество векторов. Программа должна читать вектора входного файла и выводить в выходной файл: длину каждого вектора, макси-

мальный угол между двумя векторами среди всевозможных пар векторов.

№ А 1.7. Создайте класс «трехмерная точка». Данные класса должны быть представлены тремя полями: координатами точки в пространстве. Класс должен содержать конструктор по умолчанию, конструктор, инициализирующий координаты точки заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий точку на экран в виде (1,-2,3). Создайте класс «прямая», задаваемый двумя точками или шестью координатами этих точек (два конструктора). Данные класса должны быть представлены шестью координатами двух точек, определяющих прямую. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения. Другой метод класса должен определять, находится ли заданная точка на прямой.

1.В функции main() создайте два объекта разработанных классов и выведите на консоль сообщение – находится ли данная точка на прямой.

2. Первый входной файл содержит описание произвольного количества прямых: шесть действительных чисел, определяющих две пары точек. Второй входной файл содержит набор точек. Программа должна читать описания прямых из первого входного файла и выводить в выходной файл для каждой прямой точки из второго файла, лежащие на ней.

№ А 1.8. Создайте класс «точка на плоскости». Данные класса должны быть представлены двумя полями: координатами точки. Класс должен содержать конструктор по умолчанию, конструктор, инициализирующий координаты точки заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий точку на экран в виде (1,-2). Создайте класс «окружность», задаваемый двумя координатами центра и величиной радиуса или двумя точками – центром и точкой на окружности (два конструктора). Данные класса должны

быть представлены двумя координатами центра и величиной радиуса. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий на экран уравнение окружности. Другие методы класса должны определять взаимное расположение двух окружностей: внешнее и внутреннее касание и точку касания, две точки пересечения, отсутствие общих точек.

1. В функции main() создайте два объекта разработанных классов и выведите на консоль сообщение – находится ли данная точка внутри окружности. В функции main() создайте два объекта разработанного класса «окружность» и выведите на консоль сообщение – пересекаются ли окружности.

2. Входной файл содержит описание произвольного количества окружностей: координаты центра и координаты точки на окружности – четыре числа. Программа должна читать описания окружностей из входного файла и выводить в выходной файл: площадь каждой окружности, её радиус, окружность с минимальным радиусом, а также все возможные пары пересекающихся окружностей.

№ А 1.9. Создайте класс «трехмерная точка». Данные класса должны быть представлены тремя полями: координатами точки в трехмерном пространстве. Класс должен содержать конструктор по умолчанию, конструктор, инициализирующий координаты точки заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий точку на экран в виде (1,-2,3). Создайте класс «сфера», задаваемый тремя координатами центра и величиной радиуса или двумя точками – центром и точкой на сфере (два конструктора), данные класса должны быть представлены тремя координатами центра и тремя координатами точки на сфере. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий на экран уравнение сферы. Другие методы класса должны определять взаимное расположение двух сфер: внешнее и внутреннее касание, пересечение,

отсутствие общих точек, когда одна из сфер полностью находится внутри другой или полностью вне ее.

1. В функции main() создайте два объекта разработанного класса «окружность» и выведите на консоль сообщение – пересекаются ли окружности.

2. Входной файл содержит описание произвольного количества сфер: координаты центра и радиус. Программа должна читать описания окружностей из входного файла и выводить в выходной файл описание взаимного расположения всевозможных пар сфер.

№ А 1.10. Создайте класс «трехмерная точка». Данные класса должны быть представлены тремя полями: координатами точки в трехмерном пространстве. Класс должен содержать конструктор по умолчанию, конструктор, инициализирующий координаты точки заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий точку на экран в виде (1,-2,3). Создайте класс «трехмерный вектор», задаваемый двумя точками или шестью координатами этих точек (два конструктора). Данные класса должны быть представлены шестью полями: координатами начала и конца вектора. Класс должен содержать конструктор по умолчанию, конструктор, инициализирующий координаты точек заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий вектор на экран в виде (1,0,3). Другие методы класса должны определять длину вектора, скалярное произведение двух векторов, угол между векторами, векторное произведение, разложение вектора по единичным векторам.

1. В функции main() создайте два объекта разработанного класса «вектор» и выведите на консоль скалярное произведение этих двух векторов.

2. Входной файл содержит произвольное количество векторов. Программа должна читать вектора входного файла и выводить в выходной файл: вектор с максимальной длиной,

общую сумму всех векторов, всевозможные попарные скалярные произведения произведения.

№ А 1.11. Создайте класс «треугольник», задаваемый либо длинами трех сторон, либо двумя сторонами и углом между ними, либо стороной и двумя прилежащими к ней углами (три конструктора). Данные класса должны быть представлены длинами трех сторон и размерами трех углов. Обязательными являются методы, возвращающие значения полей, метод, выводящий на экран данные треугольника. Другие методы класса должны определять площадь треугольника, периметр, длины его биссектрис, медиан, высот, радиусы вписанной и описанной окружности, свойства треугольника: прямоугольный, остроугольный тупоугольный, равнобедренный, равносторонний, осуществлять сравнение двух треугольников.

1. В функции main() создайте два объекта разработанного класса и выведите на консоль сообщение о равенстве этих двух треугольников.

2. Входной файл содержит описание произвольного количества треугольников: длины трёх сторон. Программа должна читать описания треугольников из входного файла и выводить в выходной файл для каждого треугольника: периметр, площадь, свойства, а также всевозможные пары равных треугольников.

№ А 1.12. Создайте класс «точка на плоскости». Данные класса должны быть представлены двумя полями: координатами точки. Класс должен содержать конструктор по умолчанию, конструктор, инициализирующий координаты точки заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий точку на экран в виде (1,-2). Создайте класс «четырехугольник», задаваемый либо восемью координатами четырех вершин, либо четырьмя соответствующими точками (два конструктора). Данные класса должны быть представлены восемью координатами четырех вершин. Обязательны-

ми являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий на экран данные четырехугольника. Другие методы класса должны определять длины всех сторон четырехугольника, его площадь, периметр, можно ли вписать в четырехугольник окружность, можно ли описать окружность около четырехугольника, свойства: параллелограмм, ромб, прямоугольник, квадрат.

1. В функции main() создайте объект разработанного класса «четырехугольник» и выведите на консоль сообщение – является ли четырехугольник ромбом.

2. Входной файл содержит описание произвольного количества четырёхугольников: координаты вершин. Программа должна читать описания четырёхугольников из входного файла и выводить в выходной файл для каждого: периметр, площадь, свойства, а также все четырёхугольники, в которые можно вписать окружность.

№ А 1.13. Создайте класс «точка на плоскости». Данные класса должны быть представлены двумя полями: координатами точки. Класс должен содержать конструктор по умолчанию, конструктор, инициализирующий координаты точки заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод, выводящий точку на экран в виде (1,-2). Создайте класс «прямоугольник», задаваемый координатами трёх вершин или тремя точками (два конструктора). Данные класса должны быть представлены четырьмя точками (вершинами). Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, методы, вычисляющие периметр прямоугольника, его площадь. Другие методы класса должны определять взаимное расположение двух прямоугольников: попадание внутрь, пересечение, отсутствие общих точек.

1. В функции main() создайте два объекта разработанного класса «прямоугольник» и выведите на консоль сообщение – есть ли у прямоугольников общие точки.

2. Входной файл содержит описание произвольного количества прямоугольников: координаты трёх вершин. Программа должна читать описания прямоугольников из входного файла и выводить в выходной файл координаты недостающей вершины каждого прямоугольника, его периметр, площадь, а также все такие пары прямоугольников, в которых один из прямоугольников целиком лежит внутри другого.

№ A 1.14. Разработать класс, данные которого содержат последовательность целых чисел произвольного размера, а методы необходимы для изменения данных и решения задачи из пункта 2.

1. В функции main() создайте объект разработанного класса и выведите на консоль количество и сумму чисел в последовательности.

2. Программа должна читать последовательность из входного файла и выводить в выходной файл максимальное, минимальное, среднее значение, количество появлений каждого из них в последовательности. Также должна выводиться гистограмма (из символов ASCII), показывающая распределение частот появления чисел в последовательности: по одной оси – числа, входящие в последовательность, по другой – частота встречаемости). Формат входного файла: элементы последовательности, разделённые произвольным количеством пробелов.

№ B 1.15. Создайте класс «дата». Данные класса должны быть представлены тремя полями: числом, месяцем и годом XXI века. Класс должен содержать конструктор по умолчанию, конструктор, инициализирующий поля класса заданными значениями. Обязательными являются методы, возвращающие значения полей, изменяющие эти значения, метод выводящий дату на экран в виде 05.06.07. Другие методы класса должны определять число, наступающее через заданное число дней или наступившее заданное число дней назад, по дате определять соответствующий день недели, выводить на экран

календарь месяца, соответствующего заданной дате, определять равенство дат.

1. В функции main() создайте два объекта разработанного класса и выведите на консоль сообщение о равенстве дат.

2. Входной файл содержит описание произвольного количества дат. Программа должна читать выводить в выходной файл: самую раннюю дату, самую позднюю, календарь месяца, соответствующего каждой заданной дате.

№ В 1.16. Разработать класс «матрица», данные которого содержат квадратную вещественную матрицу произвольного размера, а методы необходимы для изменения данных и решения задачи из пункта 2.

1. В функции main() создайте объект разработанного класса и выведите на консоль куб исходной матрицы.

2. Входной файл содержит квадратную вещественную матрицу. Программа должна читать описание матрицы из входного файла и выводить в выходной файл квадрат, куб исходной матрицы, её определитель, а также обратную матрицу, если последняя существует. Формат входного файла: в первой строке задаётся число строк и столбцов в матрице, в последующих – элементы, разделённые произвольным количеством пробелов.

№ В 1.17. Разработать класс «матрица», данные которого содержат квадратную целочисленную матрицу произвольного размера, а методы необходимы для изменения данных и решения задачи из пункта 2.

1. В функции main() создайте объект разработанного класса и выведите на консоль сообщение – является ли исходная матрица магическим квадратом.

2. Входной файл содержит квадратную целочисленную матрицу. Программа должна читать описание матрицы из входного файла и выводить в выходной файл матрицы, полученные из исходной путём поворота на 90, 180 и 270 градусов против часовой стрелки, а также определять, является ли исходная матрица магическим и латинским квадратом. Фор-

мат входного файла: в первой строке задаётся число строк и столбцов в матрице, в последующих – элементы, разделённые произвольным количеством пробелов.

№ В 1.18. Неориентированный граф задан матрицей смежности: если в графе существует ребро, соединяющее i-ю и j-ю вершины, то элементы i-й строки j-го столбца и j-й строки i-го столбца матрицы содержат положительное число – длину ребра, иначе – 0.

1. Разработать класс «граф», данные которого содержат матрицу смежности, а методы необходимы для изменения данных и решения задачи из пункта 2.

2. Формат входного файла: в первой строке задаётся количество вершин в графе, в последующих – элементы матрицы смежности, разделённые произвольным количеством пробелов. Программа должна читать описание графа из входного файла и выводить в выходной файл количество рёбер графа, все рёбра наибольшего и наименьшего веса, а также вектор, содержащий номера компонент связности, которым принадлежат соответствующие вершины графа.

№ В 1.19. Неориентированный граф задан матрицей смежности: если в графе существует ребро, соединяющее i-ю и j-ю вершины, то элементы i-й строки j-го столбца и j-й строки i-го столбца матрицы содержат положительное число – длину ребра, иначе – 0.

1. Разработать класс «граф», данные которого содержат матрицу смежности, а методы необходимы для изменения данных и решения задачи из пункта 2.

2. Формат входного файла: в первой строке задаётся количество вершин в графе, в последующих – элементы матрицы смежности, разделённые произвольным количеством пробелов. Программа должна читать описание графа из входного файла и выводить в выходной файл количество дуг графа, количество дуг, входящих в каждую вершину графа, кратчайшие пути от каждой вершины до всех остальных достижимых из неё.

№ B 1.20. Ориентированный граф задан набором рёбер. Каждое ребро имеет длину и вес, задаваемые целыми положительными числами.

1. Разработать класс «граф», данные которого содержат матрицу смежности, а методы необходимы для изменения данных и решения задачи из пункта 2.

2. Формат входного файла: в первой строке задаётся количество вершин в графе, в последующих – четвёрки чисел: номера вершин, образующих ребро, длину и вес ребра соответственно. Числа могут быть разделены произвольным количеством пробелов. Программа должна читать описание графа из входного файла и выводить в выходной файл матрицу длин кратчайших путей в графе, количество дуг, исходящих из каждой вершины графа и рёбра, имеющие наибольший и наименьший вес.

№ C 1.21. Входной файл содержит информацию о лабиринте в виде матрицы, в которой разными символами обозначены пустое пространство и стены (например, 0 и 1), а также информацию о начальной и конечной позициях (двумя другими символами).

1. Разработать класс для хранения, изменения, обработки информации о лабиринте.

2. Программа должна найти в лабиринте путь, если это возможно, от начальной позиции до конечной и вывести результат в файл в виде матрицы, аналогично входным данным, где найденный путь отмечен любым удобным символом. Путь не обязательно должен быть кратчайшим. Если найти путь невозможно, в файл выводится соответствующее сообщение. Формат входного файла: размеры матрицы, описывающей лабиринт, матрица.

№ C 1.22. Входной файл содержит информацию о количестве городов, которые должен объехать коммивояжер, и о расстояниях между ними. Информация хранится в виде матрицы смежности соответствующего графа (города – вершины графа, расстояния – веса ребер).

1. Разработать класс для хранения, изменения, обработки информации о графе.

2. Программа должна найти в графе гамильтонов цикл (обход всех вершин графа) наименьшей длины, если это возможно, и вывести результат в файл в виде последовательности обхода городов. Если найти путь невозможно, в файл выводится соответствующее сообщение. Формат входного файла: количество вершин (городов), матрица смежности.

№ C 1.23. Разработать калькулятор для бесконечно больших целых чисел. Входной файл содержит информацию о паре больших чисел и операции, которую необходимо над ними произвести. Операции: сложение, вычитание, умножение, целочисленное деление, нахождение наибольшего общего делителя, наименьшего общего кратного.

1. Разработать класс для хранения, изменения, обработки информации о большом числе.

2. Программа должна выполнять указанную операцию и выводить результат в файл. Формат входного файла: число, операция, число.

6.2. Строки

Работа со строками должна осуществляться только с использованием стандартного класса string. Программа должна быть объектно ориентированной.

№ A 2.1. Дан файл с текстом программы на C++. Имитировать работу директивы препроцессора define. Заменить все наборы символов на соответствующие им в директиве, удалить из текста программы саму директиву.

№ A 2.2. Дан файл с текстом на русском языке. Каждая глава начинается со слова «Глава», далее идёт её номер и название. Каждый параграф начинается с символа #, далее номер главы, точка, номер параграфа и его название. Название главы и параграфа заканчивается вместе с концом строки. Составить файл с оглавлением.

№ A 2.3. Дан файл с текстом на языке HTML. Преобразовать все теги («спецификаторы» в угловых скобках –

<body> или </body>), записанные большими буквами в такие же теги, но записанные маленькими буквами (<style> из <STYLE>). Учесть, что в тег могут входить атрибуты, которые надо оставить без изменения (<html id="A1"> из <HTML id="A1">).

№ А 2.4. Дан файл с текстом. Слово – последовательность символов латинского языка, остальное разделители. Определить, с какой буквы начинается наибольшее количество слов (маленькие и большие буквы не различаются). Результат вывести в отдельный файл: сначала в отдельной строке буква, затем слова.

№ А 2.5. Дан файл с текстом. Слово – последовательность символов латинского языка, остальное разделители. Определить, сколько раз повторяется в файле каждое слово первого предложения. Предложение заканчивается одним из трёх символов – '.', '?', '!'. Результат вывести в отдельный файл.

№ А 2.6. Дан файл с текстом. Слово – последовательность символов латинского языка, остальное разделители. Предложение заканчивается одним из трёх символов – '.', '?', '!'. Вывести в новый файл все предложения в обратном порядке.

№ А 2.7. Дан файл с текстом. Слово – последовательность символов латинского языка, остальное разделители. Предложение заканчивается одним из трёх символов – '.', '?', '!'. Вывести в новый файл все предложения, в которых более заданного количества знаков препинания.

№ А 2.8. Дан файл с текстом. Слово – последовательность символов латинского языка, остальное разделители. Предложение заканчивается одним из трёх символов – '.', '?', '!'. Вывести в новый файл все предложения, в которых более заданного количества слов.

№ А 2.9. Дан файл с текстом. Слово – последовательность символов латинского языка, остальное разделители. Определить, сколько слов содержат один слог, два слога и т. д. Результат вывести в отдельный файл в следующем виде: снача-

ла через запятую все односложные слова, затем все двусложные и т. д.

№ A 2.10. Дан файл с текстом. Слово – последовательность символов латинского языка, остальное разделители. Определить, сколько раз повторяется в файле каждое слово. Результат вывести в отдельный файл.

№ B 2.11. Дан файл с текстом программы на C++. Записать в выходной файл все идентификаторы, количество появлений каждого в тексте и номера строк, где появляется соответствующий идентификатор. Информация о новом идентификаторе должна располагаться с новой строки.

№ B 2.12. Дан файл с текстом программы на C++. Записать в выходной файл все переменные, указать, в какой строке расположено определение переменной и номера строк, где значение переменной изменяется. Информация о новой переменной должна располагаться с новой строки.

№ B 2.13. Дан файл с произвольным текстом. Преобразовать все числа (целые и десятичные дроби) в денежную сумму в рублях и копейках, остальной текст оставить без изменения. Тройки цифр в рублях разделить пробелом, в конце добавить слово «рублей», «рубля» или «рубль», после копеек «коп.» (123 456 рублей 24 коп. из 123456,2389).

№ B 2.14. Дан файл с произвольным текстом. Преобразовать все даты текущего века, записанные в виде трёх чисел, разделённых точками, в строки, где число месяц и год являются словами (первое мая две тысячи седьмого года из 01.05.07). Преобразовывать только корректные даты.

№ B 2.15. Дан файл с произвольным текстом. Преобразовать время, записанное в виде 1 час 5 минут 7 секунд во время в секундах, остальной текст оставить без изменения. Преобразовывать только корректное время (3907 секунд из 1 час 5 минут 7 секунд).

№ B 2.16. Дан файл с произвольным текстом. Преобразовать время, записанное в виде трёх чисел, разделённых пробелом, добавив слова «час», «минута», «секунда» в нужном

падеже, остальной текст не изменять. Преобразовывать только корректное время (1 час 5 минут 7 секунд из 1 5 7).

№ B 2.17. Дан файл с произвольным текстом. Разделители слов – пробелы. Отформатировать текст по ширине по следующим правилам: длина строки — 60 символов, отступ абзаца – 3 символа. Результат вывести в отдельный файл.

№ C 2.18. Преобразовать файл с текстом программы на C++ к каноническому виду. Текст программы должен начинаться с первого символа строки. Все операторы линейной части программы должны иметь слева одно и то же число пробелов. При определении функции, цикла или условного оператора открывающая скобка должна располагаться в той же строке, что и соответствующее служебное слово, а закрывающая фигурная скобка должна располагаться на уровне первой буквы первого слова той синтаксической конструкции, к которой она относятся. В случае условного оператора или цикла с одним оператором последний следует располагать либо на одной строке с управляющим оператором, либо на следующей строке с отступом слева от него.

№ C 2.19. Преобразовать файл с описанием класса на C++ в два: заголовочный, содержащий только описание полей и прототипы методов, и файл с определением методов, с названием, соответствующим заголовочному и расширением cpp. Оба файла должны соответствовать каноническому виду. Все операторы линейной части программы должны иметь слева одно и то же число пробелов. При определении функции, цикла или условного оператора открывающая скобка должна располагаться в той же строке, что и соответствующее служебное слово, а закрывающая фигурная скобка должна располагаться на уровне первой буквы первого слова той синтаксической конструкции, к которой она относятся.

№ C 2.20. Создать на основе файла с описанием класса на C++ файл с документационными комментариями (исходный текст должен содержать соответствующие комментарии для каждого члена класса, если комментарий отсутствует, то

его не будет и в создаваемом файле). Файл должен содержать: комментарий (одна строка), описание (не полное определение!) члена класса (следующая строка) и т. д. В начале файла следует указать название класса.

№ С 2.21. Разработать класс для преобразования текста, форматирования текста и сбора статистики. Преобразование текста: замена слов и словосочетаний; удаление лишних пробелов, запятых, добавление пробела после запятой; проверка правильности расстановки скобок всех видов. Форматирование текста: отступ в начале абзаца с заданным количеством символов; ширина текста с возможностью выравнивания влево, вправо и по ширине; перенос слов или его отключение. Статистика: количество предложений; количество слов; количество всех символов. Формат файла с текстом: сначала правила форматирования в любом удобном виде, затем текст.

Литература

[1] Давыдов, В. Г. Технологии программирования C++/ В. Г. Давыдов. — СПб.: БХВ–Петербург, 2005.

[2] Лаптев, В. В. C++. Объектно ориентированное программирование. Задачи и упражнения/В. В. Лаптев, А. В. Морозов, А. В. Бокова. — СПб.: Питер, 2007.

[3] Лагутина, Н. С. Основы объектно ориентированного программирования на языке C++/Н. С. Лагутина; Яросл. гос. ун–т. — Ярославль: ЯрГУ, 2008.

[4] Лафоре, Р. Объектно ориентированное программирование в C++/ Р. Лафоре. — СПб.: Питер, 2004.

[5] Павловская, Т. А. C++. Объектно ориентированное программирование: практикум/ Т. А. Павловская, Ю. А. Щупак. — СПб.: Питер, 2006.

[6] Павловская, Т. А. C/C++. Программирование на языке высокого уровня/Т. А. Павловская. — СПб.: Питер, 2003.

[7] Пол, А. Объектно ориентированное программирование на C++/ А. Пол. — СПб.: Бином, 2001.

Учебное издание

Лагутина Надежда Станиславовна

C++. Примеры и задачи

Практикум

Редактор, корректор И. В. Бунакова
Компьютерный набор, верстка Н. С. Лагутина

Подписано в печать 27.01.2011 г. Формат 60×84/16.
Бумага тип. Усл. печ. л. 2,79 Уч.-изд. л. 2,0.
Тираж 50 экз. Заказ

Оригинал-макет подготовлен в редакционно-издательском
отделе Ярославского государственного университета
им. П. Г. Демидова.

Отпечатано на ризографе
Ярославский государственный университет
им. П. Г. Демидова
150000 Ярославль, ул. Советская, 14.