**Семахин Андрей Михайлович**

Родился в 1963 году в г. Кургане. Окончил среднюю школу №31 г. Кургана, Курганский машиностроительный институт, аспирантуру Московского государственного технологического университета «СТАНКИН», кандидат технических наук, доцент.

Работает на кафедре программного обеспечения автоматизированных систем с 2006 года. Преподаёт дисциплины: основы программирования, языки программирования, алгоритмы и структуры данных, компьютерная графика, web-программирование, теория информации, тестирование и управление качеством ПО.

Сфера научных интересов: моделирование систем.

Автор 78 научных и учебно-методических работ. Научные труды опубликованы в США, Германии, Чехии, Польше и Болгарии.

ISBN 978-5-4217-0369-3

Курганский государственный университет

редакционно-издательский центр

65-48-12

А.М. Семахин

# ОСНОВЫ ПРОГРАММИРОВАНИЯ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

УЧЕБНОЕ ПОСОБИЕ

*МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ*

федеральное государственное бюджетное образовательное учреждение
высшего образования
«Курганский государственный университет»

А.М. Семахин

# ОСНОВЫ ПРОГРАММИРОВАНИЯ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Учебное пособие

Курган 2016

УДК 004.4(075.8)
ББК 32.973я73
С30

**Рецензенты**

**В.Ю. Пирогов** – кандидат физико-математических наук, профессор, заведующий кафедрой прикладной информатики и экономики Шадринского государственного педагогического института;

**В.Г. Коуров** – доктор технических наук, профессор, директор Шадринского филиала Московского педагогического государственного университета им М.А. Шолохова, заведующий кафедрой прикладной информатики и математики.

Печатается по решению методического совета Курганского государственного университета.

**Семахин А. М.**

Основы программирования. Лабораторный практикум : учебное пособие. – Курган : Изд-во КГУ, 2016. 84 с.

В лабораторном практикуме рассматриваются примеры программ, формализующие варианты задач лабораторных работ по дисциплине «Основы программирования» и приводятся теоретические сведения по структурному, объектно-ориентированному, визуальному и обобщённому методам программирования.

Для закрепления теоретических знаний и приобретения практических навыков программирования представлены вопросы и задания для самостоятельной работы.

Рис. – 11, библиограф. – 7.

УДК 004.4(075.8)
ББК 32.973я73

ISBN 978-5-4217-0369-3

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Объектно-ориентированный язык программирования C++ является самым распространенным языком программирования в формализации решения задач на ЭВМ. Операционные системы Windows и Linux написаны на языке C++. Преимуществом языка C++ являются гибкость, переносимость и универсальность. Язык C++ используется для решения задач любой степени сложности.

Язык C++ создан Бьярном Страуструпом (Bjarne Stoustrup) в 1979 году в компании Bell Laboratories (г. Муррей Хилл, штат Нью-Джерси). В 1989 году язык C++ прошел стандартизацию ANSI (American National Standards Institute, Национальный институт стандартизации США). Результатом стандартизации стало существование двух версий C++: традиционная версия C++ и стандартная (Standard C++). Традиционная версия базируется на исходной разработке Бьярна Страуструпа. Версия Standard C++ создана Бьярном Страуструпом совместно с комитетом по стандартизации ANSI/ISO (International Standards Organization, Международная организация по стандартам).

Лабораторный практикум включает примеры программ выполнения заданий лабораторных работ и теоретические положения по дисциплине «Основы программирования».

Для закрепления теоретических знаний и приобретения практических навыков программирования в конце подразделов приводятся вопросы и задачи для самостоятельного решения.

# 1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ЯЗЫКА C++

## 1.1 Базовые средства языка C++

### *1.1.1 Тип данных*

Тип данных (data type) – множество допустимых значений данных с набором операций, которые применимы к этим данным.

Тип данных определяет внутреннее представление данных в памяти компьютера, множество значений, которые могут принимать величины этого типа, операции и функции, которые можно применять к величинам

этого типа. Подразделяются на два вида: основные (fundamental) и производные (derived).

Основные типы данных: целый (int), символьный (char), расширенный символьный (wchar_t), логический (bool), вещественный (float), вещественный с двойной точностью (double). Существуют четыре спецификатора, уточняющие внутреннее представление и диапазон значений стандартных типов: короткий (short), длинный (long), знаковый (signed), беззнаковый (unsigned).

Производные типы данных: массивы (arrays), функции (functions), указатели (pointers), ссылки (reference), классы (classes), структуры (structures), объединения (unions), указатели на члены классов (pointers to class-members).

В таблице 1.1 приведены диапазоны значений и размер в байтах стандартных типов данных [1].

*Таблица 1.1 – Диапазоны значений и размер стандартных типов данных*

| Тип | Диапазон значений | Размер (байт) |
|---|---|---|
| Bool | true, false | 1 |
| signed char | -128…127 | 1 |
| unsigned char | 0…255 | 1 |
| signed short int | -32768…32767 | 2 |
| unsigned short int | 0…65535 | 2 |
| signed long int | -2147483648…2147438647 | 4 |
| unsigned long int | 0…4294967295 | 4 |
| Float | 3.4e-38…3.4e+38 | 4 |
| Double | 1.7e-308…1.7+308 | 8 |
| long double | 3.4e-4932…3.4e+4932 | 10 |

Имя переменной служит для обращения к области памяти, в которой хранится значение. Формат объявления переменной имеет вид

*[класс памяти] [const] тип имя [инициализатор];*

1 Необязательный параметр класс памяти принимает значения auto, extern, static, register.

2 Модификатор const показывает, что значение переменной изменять нельзя.

3 Необязательный параметр инициализатор присваивает переменной начальное значение.

Область действия – часть программы, в которой его можно использовать для доступа к связанной с ним области памяти.

Время жизни – существование переменной в течение выполнения программы или блока.

Область видимости идентификатора – часть текста программы, из которой допустим доступ к связанной с идентификатором областью памяти.

Тип void (пустой) применяется для определения функций, которые не возвращают значения [1].

## *1.1.2 Выражения и операции*

Выражение (expression) – конструкция языка (формула) для вычисления значения в соответствии со значениями операндов.

Операнд (operand) – часть команды, определяющая аргумент, над которым выполняется операция.

Приоритет операций (operation precedence) – очередность выполнения операций в выражении.

Операция (operation) –действие, выполняемое над операндами.

Арифметические операции приведены в таблице 1.2.

*Таблица 1.2 – Арифметические операции*

| Операция | Знак | Пример | Пояснения |
|---|---|---|---|
| Сложение | + | y=x+9; | Присваивает y значение, равное x плюс 5 |
| Вычитание | - | y=x-6; | Присваивает y значение, равное x минус 6 |
| Умножение | * | y=x*2; | Присваивает y значение, равное x, умноженное на 2 |
| Деление | / | y=x/12; | Присваивает y значение, равное x, деленное на 12 |
| Деление по модулю | % | y=x%15; | Присваивает y остаток от деления на 15 |
| Инкремент | ++ | ++x; | Префиксный инкремент. Увеличивает x на единицу, а затем использует |
| | | x++; | Постфиксный инкремент. Использует x, а затем увеличивает |
| Декремент | -- | --x; | Префиксный декремент. Уменьшает x на единицу, а затем использует |
| | | x--; | Постфиксный декремент. Использует x, а затем уменьшает на единицу |

Операции отношения приведены в таблице 1.3.

*Таблица 1.3 – Операции отношения*

| Название | Знак | Пояснения |
|---|---|---|
| Равенство | == | Истина, если операнды равны, иначе – ложь |
| Неравенство | != | Истина, если операнды не равны, иначе – ложь |
| Меньше | < | Истина, если левый операнд меньше правого, иначе – ложь |
| Больше | > | Истина, если левый операнд больше правого, иначе – ложь |
| Меньше или равно | <= | Истина, если левый операнд меньше или равен правому, иначе – ложь |
| Больше или равно | >= | Истина, если левый операнд больше или равен правому, иначе – ложь |

Перечень логических операций приведен в таблице 1.4.

К побитовым операциям относятся операции И (&), ИЛИ (||), исключающее ИЛИ (^), отрицание (~), сдвиг влево (<<), сдвиг вправо (>>). В побитовых операциях действия происходят над двоичным представлением целых чисел [2].

*Таблица 1.4 – Логические операции*

| Назначение | Знак | Пояснение |
|---|---|---|
| Конъюнкция | && | Истина, если оба операнда истинны, иначе – ложь |
| Дизъюнкция | \|\| | Истина, если хотя бы один операнд истинен, иначе – ложь |
| Отрицание | ! | Значение операнда меняется на противоположное |

### *1.1.3 Операторы языка* C++

Оператор (statement) – установленное синтаксисом языка действие в программе. В языке C++ выражение, которое заканчивается символом точки с запятой, является оператором. Операторы языка C++ имеют строгие правила написания. Если правила будут нарушены, компилятор сообщит об ошибке.

Существуют следующие группы операторов.

**1 Оператор-выражение:**

- оператор присваивания;

- оператор вызова функции;
- пустой оператор.

**2 Составной оператор:**

- блок, состоящий из одного или нескольких операторов, заключенных в фигурные скобки.

**3 Операторы ветвления:**

- условный оператор *if...else*;
- оператор переключатель *switch*.

**4 Операторы цикла:**

- оператор пошагового цикла *for (выраженние1; выражение2; выражение3) оператор*;
- оператор цикла с предусловием *while (выражение) оператор*;
- оператор цикла с постусловием *do оператор while (выражение);*.

**5 Операторы передачи управления:**

- оператор безусловного перехода *goto*;
- оператор выхода из цикла *break*;
- оператор перехода к следующей итерации цикла *continue*;
- оператор возврата из функции *return* [2].

## *1.1.4 Пример программы*

Разработайте программу, рассчитывающую и выводящую на экран в виде таблицы значения функции, разложенной в бесконечный ряд Тейлора на интервале $[X_н, X_к]$ с шагом h и точностью $\varepsilon$.

$$\ln(\frac{1+x}{1-x}) = 2 * \sum_{n=0}^{\infty} \frac{x^{2*n+1}}{2*n+1} = 2 * (x + \frac{x^3}{3} + \frac{x^5}{5} + ...), |X| < 1.$$

Программный код приведён в листинге 1.1.

Листинг 1.1. Разложение в ряд Тейлора.

*ЛабРаб1.cpp.*

```
#include "stdafx.h"
#include<iostream>
#include<math.h>
using namespace std;
double logon(double x, int n)
{  double res;
```

```
    res=2.0*(pow(x, 2.0*n+1.0)/(2.0*n+1.0)); return res;}
int _tmain(int argc, _TCHAR* argv[])
{  setlocale(LC_ALL, "russian");
    double Xn, Xk, h, e;
    double x, res, sum, q;
    int n;
    for(int i=1; i==1; i) {
        for(int i=1; i==1; i) {
            cout<<"Введите X начальное: "; cin>>Xn;
            if(Xn>-1&&Xn<1) {i=0;}
            else {cout<<"Введено недопустимое значение. Повторите ввод!"<<endl;}}
        for(int i=1; i==1; i) {
            cout<<"Введите X конечное: "; cin>>Xk; if(Xk>-1&&Xk<1) {i=0;}
            else {cout<<"Введено недопустимое значение. Повторите ввод!"<<endl;}}
        if(Xn<=Xk) {i=0;}
        else {cout<<"X начальное больше, чем X конечное. Повторите ввод!"<<endl;}}
    for(int i=1; i==1; i)
    { cout<<"Введите шаг h: "; cin>>h; if(h>0) {i=0;}
    else {cout<<"Введено недопустимое значение. Повторите ввод!"<<endl;}}
    for(int i=1; i==1; i) {cout<<"Введите точность E "; cin>>e; cout<<endl;
    if(e<=1&&e>0) {i=0;} else {cout<<"введено недопустимое значение. Повторитеп ввод!"<<endl;}
    }
    puts("-----------------------------------------------------------------");
    printf("| %-4.1s  |          %-23.15s  |  \n", "X", "ln((1+x)/(1-x))");
    puts("-----------------------------------------------------------------");
    for(x=Xn; x<=Xk; x+=h) {
        for(n=0, sum=0, q=1; q>e||fabs(sum)>e; n++, q=e)
        {sum=+logon(x, n);
    }
```

```
        printf("|    %5.1f   |   %34.30f        \n", x,sum);
        }
    puts("-----------------------------------------------------------------");
    system("pause");
    return 0;}
```

На рисунке 1.1 приведён результат работы программы.

## *1.1.5 Контрольные вопросы*

1 Что называется переменной?

2 Какие действия выполняет компилятор языка С++ при выполнении программной инструкции объявления переменной?

3 Какие типы данных применяются в языке С++?

4 Какие операции используются в языке С++?

5 Какие операторы используются в языке С++?

## *1.1.6 Задания для самостоятельной работы*

1 Разработайте программу, вычисляющую по формуле $S_n = \frac{2 * a_1 + d * (n-1)}{2} * n$ сумму n членов арифметической прогрессии $S_n$. Входные данные программы: $a_1$ - первый член прогрессии, d – разность, n – число членов прогрессии.

2 Напишите программу, рассчитывающую и выводящую на экран в виде таблицы значения функции $y$ на интервале от $X_н$ до $X_к$ с шагом $h$.

$$y = \begin{cases} a * x^2 + 25 * b * x, при\ x < 0, \text{b} \neq 0 \\ \frac{\text{x - a}}{\text{c}}, при\ x \geq 0\ и\ c \neq 0 \\ \frac{16 * x}{c + 8}, в\ остальных\ случаях \end{cases},$$

где a, b, c – действительные числа.

3 Напишите программу для решения квадратного уравнения, которое имеет вид $a * x^2 + b * x + c = 0$. Параметры a, b и c вводятся пользователем.

4 Разработайте программу, которая вычисляет по формуле $S_n = \frac{2 * a_1 + d * (n-1)}{2} * n$ сумму $n$ членов арифметической прогрессии $S_n$.

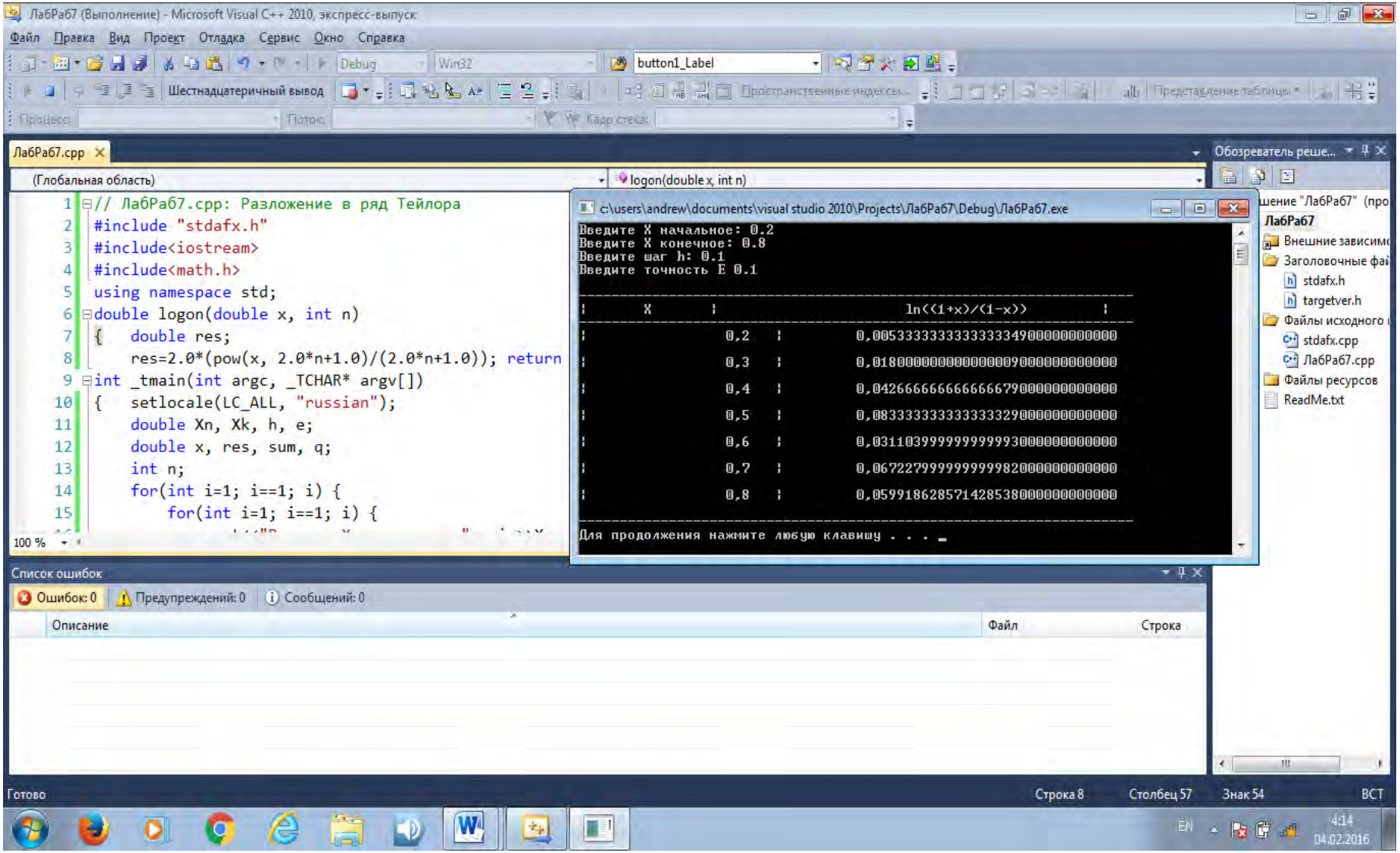


*Рисунок 1.1 – Разложение в ряд Тейлора*

Входные данные программы: $a_1$ – первый член прогрессии, $d$ – разность, $n$ – число членов прогрессии.

5 Создайте программу расчета объема $V$, массы $m$ и площади $S$ основания металлического слитка. Известны плотность $\rho$, высота $h$ и радиус основания $R$ цилиндрического слитка, полученного в металлургической лаборатории. Расчетные формулы $S = 2 * \pi * R$, $V = \pi * R^2 * h$, $m = \rho * V$.

## 1.2 Массивы

### *1.2.1 Описание массивов*

Массив – пользовательский тип данных, элементы которого имеют одинаковый тип данных.

Динамический массив – массив переменной длины, память под который выделяется в процессе выполнения программы.

Выделение памяти для динамического массива производится оператором new [ ], освобождение памяти – оператором delete [ ].

Индекс первого элемента массива – нуль, индекс последнего элемента – n-1, где n количество элементов массива [2].

### *1.2.2 Операции над массивами*

Основная операция массива – индексация элемента. Доступ к элементу массива производится посредством операции индекса [ ] или разыменования *.

В листинге 1.2 приводится пример массива.

Листинг 1.2. Динамический массив

```
#include “stdafx.h”
#include <iostream>
#include <ctime>
using namespace std;
int _tmain(int argc, _TCHAR *argv[]) {
setlocale(LC_ALL, “russian”);
```

```
int i, n; cout<<”Введите размерность массива ”; cin>>n; cout<<endl;
int *mas=new int[n]; srand(time(NULL));
for(i=0; i<n; i++)
mas[i]=50+rand()%100;
for(i=0; i<n; i++)
cout<< “Элемент массива mas[”<<i<<”]->”<<*(mas+i);
delete [] mas;
system(“pause”);
return;}
```

## *1.2.3 Пример программы*

Разработайте программу, определяющую в квадратной матрице $[A]_{10*10}$ величины:

- количество элементов матрицы, содержащих простые и совершенные числа;
- простые и совершенные числа матрицы;
- максимум элементов, расположенных выше главной диагонали матрицы.

Элементы двумерного массива, расположенные выше главной диагонали, упорядочите по убыванию простым методом выбора.

Программа, формализующая двумерный динамический массив и операции над элементами массива, приведена в листинге 1.3.

Листинг 1.3. Двумерный массив

```
//LabPrak2.cpp: Динамический двумерный массив.
#include "stdafx.h"
#include <iostream>
#include <ctime>
using namespace std;
unsigned int pros(unsigned int);
unsigned int sove(unsigned int);
int _tmain(int argc, _TCHAR* argv[])
{  setlocale(LC_ALL, "russian");
   int i, j, n, m, Kpr, Ksv, g, f, z, max, v, buf, num;
```

```
cout<<"Введите количество строк двумерного массива n: "; cin>>n; cout<<endl;
cout<<"Введите количество столбцов двумерного массива m: "; cin>>m; cout<<endl;
int **mas=new int *[n];
for(i=0; i<n; i++)
    mas[i]=new int [m];
//Инициализация элементов двумерного массива случайными числами
srand(time(NULL));
    for(i=0; i<n; i++)
        for(j=0; j<m; j++)
        mas[i][j]=1+rand()%100;
cout<<"Исходный двумерный массив"<<endl;
for(i=0; i<n; i++) {
    for(j=0; j<m; j++)
    cout<<mas[i][j]<<'\t';
    cout<<endl; }
//Определение количества простых и совершенных чисел в двумерном массиве
Kpr=0;
for(i=0; i<n; i++)
    for(j=0; j<m; j++)
        if(pros(mas[i][j])) Kpr++;
g=0;
int *pro=new int[Kpr];
    for(i=0; i<n; i++)
            for(j=0; j<m; j++)
                if(pros(mas[i][j])) {pro[g]=mas[i][j]; g++;}
cout<<"\nКоличество простых чисел в двумерном массиве "<<Kpr<<endl;
cout<<"Простые числа:"<<endl;
    for(i=0; i<g; i++) cout<<pro[i]<<'\t'; cout<<endl;
Ksv=0;
for(i=0; i<n; i++)
    for(j=0; j<m; j++)
        if(sove(mas[i][j])) Ksv++;
```

```
f=0;
int *sov=new int[Ksv];
   for(i=0; i<n; i++)
         for(j=0; j<m; j++)
            if(sove(mas[i][j])) {sov[f]=mas[i][j]; f++;}
cout<<"Количество совершенных чисел в двумерном массиве "<<Ksv<<endl;
cout<<"Совершенные числа:"<<endl;
for(i=0; i<f; i++) cout<<sov[i]<<'\t'; cout<<endl;
//Максимальный элемент, расположенный выше главной диагонали
z=0; for(i=0; i<n; i++)
   for(j=0; j<m; j++)
      if(i<j) z++;
cout<<"Количество элементов, расположенных выше главной диагонали матрицы "<<z<<endl;
cout<<"Элементы, расположенные выше главной диагонали матрицы"<<endl;
v=0; int *elem=new int[z];
   for(i=0; i<n; i++)
         for(j=0; j<m; j++)
            if(i<j) {elem[v]=mas[i][j]; v++;}
            for(i=0; i<z; i++) cout<<elem[i]<<'\t'; cout<<endl;
      max=elem[0]; for(i=0; i<z; i++) if(max<elem[i]) max=elem[i];
      cout<<"Максимальный элемент, расположенный выше главной диагонали матрицы: "<<max<<endl;
//Сортировка элементов, расположенных выше главной диагонали матрицы
      cout<<"Сортировка простым методом выбора"<<endl;
         num=0; for(i=1; i<z; buf=elem[z-i], elem[z-i]=elem[num], elem[num]=buf, i++)
               for(max=elem[0], num=0, j=1; j<=z-i; j++)
                  if(elem[j]>max) {max=elem[j]; num=j;}
                     for(j=0; j<z; j++) cout<<elem[j]<<'\t'; cout<<endl;
               for(i=0; i<n; i++)
      delete [] mas[i]; delete [] mas; delete [] pro; delete [] sov; delete [] elem;
```

```
    system("pause");
    return 0;
}
unsigned int pros(unsigned int Y) {
int k, l;
    for(l=1, k=2; k<=Y/2; k++)
        if(Y%k==0) {l=0; break;}
        return l;
}
unsigned int sove(unsigned int Y) {
    unsigned int a, b;
        for(b=0, a=1; a<=Y/2; a++)
            if(Y%a==0) b+=a;
        if(b==Y) return 1;
        else return 0;}
```

Результат работы программы, формализующей динамический двумерный массив целых чисел, приведен на рисунке 1.2.

### 1.2.4 Контрольные вопросы

1 Что называется массивом?

2 В чём отличие статического массива от динамического массива?

3 Как осуществляется доступ к элементам массива?

4 Что такое размерность массива?

5 Какие операторы используются при создании динамического массива?

### 1.2.5 Задания для самостоятельной работы

1 Напишите программу, определяющую в одномерном динамическом массиве, состоящем из *n* вещественных элементов:

- среднее значение элементов, расположенных между первым и последним нулевыми элементами.

Поменяйте местами максимальный и минимальный элементы.

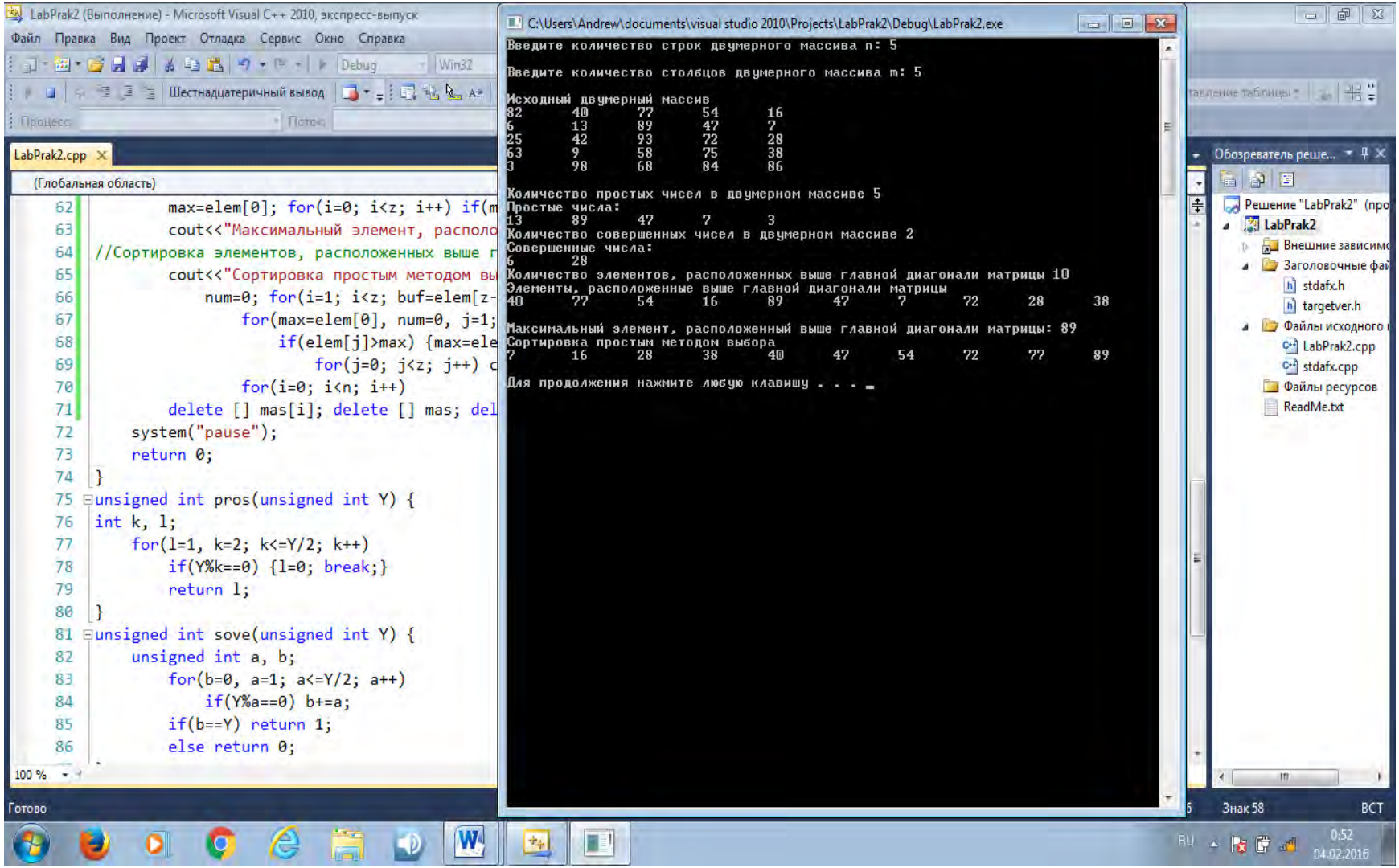


*Рисунок 1.2 – Результат работы программы, формализующей двумерный динамический массив и операции над элементами массива*

Упорядочите по возрастанию элементы, стоящие на четных местах, и по убыванию элементы, стоящие на нечетных местах.

2 Напишите программу, определяющую в целочисленной прямоугольной матрице $[A]_{10*13}$ величины:

- количество элементов матрицы, содержащих нечетные числа;
- максимум элементов, расположенных по периметру прямоугольной матрицы;
- сумму элементов, находящихся над главной диагональю матрицы.

Элементы первого столбца матрицы упорядочите по убыванию методом обмена.

3 Разработайте программу, определяющую в квадратной матрице $[A]_{12*12}$ величины:

- количество элементов матрицы, содержащих простые числа;
- максимум среди элементов главной и побочной диагоналей матрицы;
- средние квадратичные значения четных столбцов и нечетных строк матрицы.

Средние квадратичные значения четных столбцов и нечетных строк матрицы упорядочите по возрастанию простым методом обмена.

4 Разработайте программу, определяющую в квадратной матрице $[A]_{15*15}$ величины:

- количество элементов матрицы, содержащих простые числа;
- минимум элементов, расположенных ниже главной диагонали матрицы;
- средние квадратичные значения нечетных столбцов и четных строк матрицы.

Средние квадратичные значения нечетных столбцов и четных строк матрицы упорядочите по возрастанию простым методом выбора.

5 Разработайте программу, определяющую в квадратной матрице $[A]_{13*13}$ величины:

- количество элементов матрицы, содержащих простые числа;
- максимум элементов, расположенных выше главной диагонали матрицы;
- средние арифметические значения столбцов и строк матрицы.

Средние арифметические значения столбцов и строк матрицы упорядочите по возрастанию простым методом вставки.

## 1.3 Структуры

### *1.3.1 Структуры и операции над ними*

Структура (structure) объединяет логически связанные данные разных типов и представляет набор переменных, объединенных общим именем.

Объявление структуры имеет вид

struct идентификатор {объявления_элементов_структуры};

Идентификатор задает имя структуры и применяется при объявлении структурных переменных такого типа. Структуры могут быть вложенными [2].

### *1.3.2 Массивы структур*

Массив структур – множество элементов пользовательского типа, каждый элемент которого является структурой. Доступ к элементам массива структур производится по индексу [2].

В листинге 1.4 приводится пример объявления массива структур.

Листинг 1.4. Массив структур

```
#include “stdafx.h”
#include <iostream>
#include <ctime>
using namespace std;
int _tmain(int argc, _TCHAR *argv[]) {
setlocale(LC_ALL, “russian”);
struct SpaceShutle {char Name[10]; double Weight; int Number};
SpaceShutle Ship[5];
system(“pause”);
return;}
```

### *1.3.3 Указатели на структуры*

Для обращения к элементам структуры применяется указатель. Обращение производится посредством оператора «стрелка» ->. Формат объявления указателя на структур имеет вид

*тип_структуры *идентификатор;*

Формат объявления ссылки на структуру представляется в виде

*тип_структуры &ссылка=переменная;*

Ссылка на структуру инициализируется именем объекта, на который указывает. Массив указателей объявляется следующим образом

*тип_структуры *идентификатор[размерность];* [2].

### *1.3.4 Пример программы*

Разработайте программу на языке C++, описывающую массив структур AEROFLOT. Структура содержит поля:

- название пункта назначения;
- номер рейса;
- тип самолёта.

В программе предусмотрите выполнение следующих действий:

- ввод с клавиатуры данных в массив, состоящий из трёх элементов типа AEROFLOT;
- вывод на экран видеомонитора введённых данных;
- сортировка данных по названию пункта назначения;
- вывод на экран видеомонитора упорядоченных данных по названию пункта назначения;
- поиск данных по типу самолёта;
- вывод на экран пунктов назначения и номеров рейсов, обслуживаемых самолётом, тип которого введён с клавиатуры;
- выдача сообщения на экран видеомонитора в случае отсутствия таких рейсов.

Пример программного приложения приведён в листинге 1.5.

Листинг 1.5. Массив структур AEROFLOT

```
//AEROFLOT.cpp: Массив структур.
#include "stdafx.h"
```

```
#include<iostream>
#include<windows.h>
#include<math.h>
#include<stdio.h>
#include<string.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{  setlocale(LC_ALL, "russian");
   const int n=3;
   struct AEROFLOT {
      char dest[20];       //Название пункта назначения
      char number[10];  //Номер рейса
      char type[15];       //Тип самолета
   };
    AEROFLOT samolety[n];
    AEROFLOT stack;
    int i, j, f;
    char TypePlane[10];
//Ввод данных
    for(i=0; i<n; i++) {
    cout<<"Введите пункт назначения самолета: ";
    int cp=GetConsoleCP();
      SetConsoleCP(1251);
      cin>>samolety[i].dest;
          SetConsoleCP(cp);
      cout<<"Введите номер рейса: ";
      int cp1=GetConsoleCP();
      SetConsoleCP(1251);
      cin>>samolety[i].number;
      SetConsoleCP(cp1);
      cout<<"Введите тип самолета: ";
      int cp2=GetConsoleCP();
      SetConsoleCP(1251);
      cin>>samolety[i].type;
      SetConsoleCP(cp2);
    }
 //Вывод данных
    cout<<"\nИсходный массив: ";
```

```
        for(i=0; i<n; i++) {
        cout<<"\nПункт назначения самолета: "<<samolety[i].dest<<endl;
        cout<<"Номер рейса: "<<samolety[i].number<<endl;
        cout<<"Тип самолета: "<<samolety[i].type<<endl;
         }
      //Сортировка данных
        for(i=1; i<n; i++)
           for(j=n-1; j>=i; j--) {
              if(strcmp(samolety[j-1].dest, samolety[j].dest)>0) {
                 stack=samolety[j-1];
                 samolety[j-1]=samolety[j];
                 samolety[j]=stack;}}
           cout<<"\nОтсортированный массив:";
           for(i=0; i<n; i++) {
        cout<<"\nПункт назначения самолета: "<<samolety[i].dest<<endl;
        cout<<"Номер рейса: "<<samolety[i].number<<endl;
        cout<<"Тип самолета: "<<samolety[i].type<<endl;}
      //Поиск
           cout<<"\nВведите тип самолета для поиска: ";
           int cp3=GetConsoleCP();
           SetConsoleCP(1251);
           cin>>TypePlane;
           SetConsoleCP(cp3);
           f=0;
           for(i=0; i<n; i++)
           if(strcmp(samolety[i].type, TypePlane)==0)
           {f++; cout<<"Направление: "<<samolety[i].dest<<", рейс № "<<samolety[i].number<<", тип самолета "<<samolety[i].type; cout<<endl;}
           cout<<endl;
          if(f==0) cout<<"Не найдено рейсов с данным типом самолета!"<<endl;
              system("pause");
          return 0;}
```

Результат работы программы приведён на рисунке 1.3.

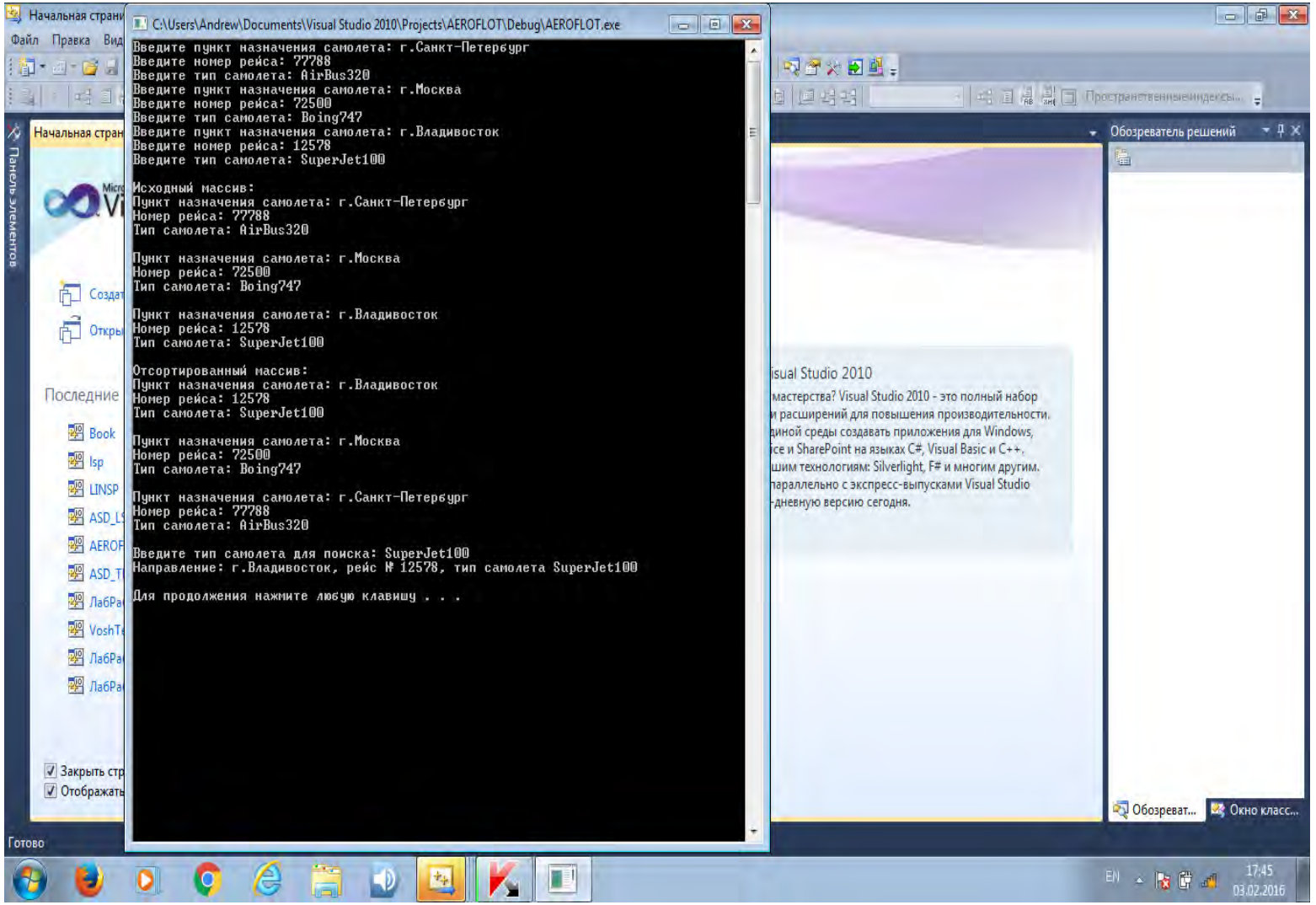


*Рисунок 1.3 – Результат работы программы, описывающей массив структур*

## *1.3.5 Контрольные вопросы*

1 Что называется структурой?

2 Какие операции выполняются над массивом структур?

3 Как осуществляется доступ к элементам структуры при помощи указателя?

4 Как объявляется ссылка на структуру?

5 Как объявляется массив указателей на структуру?

## *1.3.6 Задания для самостоятельной работы*

1 Создайте структуру с именем STUDENT, содержащую поля: фамилия и инициалы, номер группы, курс, успеваемость (массив из четырех элементов).

Напишите программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из шести структур типа STUDENT;
- записи упорядочить по возрастанию номера группы;
- вывод на дисплей фамилий и номеров групп для всех студентов, включенных в массив, если средний балл студента больше 4.0;
- если таких студентов нет, вывод соответствующего сообщения.

2 Создайте структуру с именем STUDENT, содержащую поля: фамилия и инициалы, номер группы, специальность, успеваемость (массив из четырех элементов).

Напишите программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из пяти структур типа STUDENT;
- записи упорядочить по возрастанию среднего балла;
- вывод на дисплей фамилий и номеров групп для всех студентов, имеющих оценки 4 и 5;
- если таких студентов нет, вывод соответствующего сообщения.

3 Создайте структуру с именем STUDENT, содержащую следующие поля: фамилия и инициалы, номер группы, адрес проживания, успеваемость (массив из четырех элементов).

Напишите программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из пяти структур типа STUDENT;

- записи упорядочить в алфавитном порядке по адресу проживания;

- вывод на дисплей студентов, имеющих оценку 2;

- если таких студентов нет, вывод соответствующего сообщения.

4 Создайте структуру с именем TRAIN, содержащую поля: пункт назначения, номер поезда, время отправления, время следования в пути.

Напишите программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из четырех элементов типа TRAIN;

- записи упорядочить по названию пункта назначения;

- вывод упорядоченных по названию пункта назначения записей;

- поиск по номеру поезда и вывод на экран видеомонитора пункта назначения, номера поезда, время отправления и время следования в пути в случае успешного поиска;

- если таких поездов нет, вывод на дисплей соответствующего сообщения.

5 Создайте структуру с именем BUSABROAD, содержащую поля: пункт назначения, номер маршрута, марка автобуса, стоимость проезда.

Напишите программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из пяти элементов типа BUSABROAD;

- записи разместить в алфавитном порядке по названиям пунктов назначения;

- вывод на экран пунктов назначения и номеров маршрутов, обслуживаемых автобусом, марка которого введена с клавиатуры;

- если таких рейсов нет, вывод на дисплей соответствующего сообщения.

## *1.4 Функции*

Функция – это именованная последовательность описаний и операторов, выполняющая действие. Функция может принимать параметры и возвращать значение.

Программа на языке C++ состоит из функций. Главная функция – функция main().

Объявление функции (прототип, заголовок, сигнатура) задает ее имя, тип возвращаемого значения и список передаваемых параметров. Опреде-

ление функции содержит, кроме объявления, тело функции, представляющее собой последовательность операторов и описаний в фигурных скобках:

[ класс ] тип имя ([ список_параметров ])[throw ( исключения )]

{ тело функции }

Составные части определения.

1 С помощью необязательного модификатора класс можно явно задать область видимости функции, используя ключевые слова extern и static:

- extern – глобальная видимость во всех модулях программы (по умолчанию);

- static – видимость только в пределах модуля, в котором определена функция.

2 Тип возвращаемого функцией значения может быть любым, кроме массива и функции (но может быть указателем на массив или функцию). Если функция не должна возвращать значение, указывается тип void.

3 Список параметров определяет величины, которые требуется передать в функцию при ее вызове. Элементы списка параметров разделяются запятыми. Для каждого параметра, передаваемого в функцию, указывается его тип и имя (в объявлении имена можно опускать).

В определении, объявлении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать [2].

### *1.4.1 Механизм передачи параметров в функцию*

Механизм передачи параметров является основным способом обмена информацией между вызываемой и вызывающей функциями. Параметры, перечисленные в заголовке описания функции, называются формальными параметрами или просто параметрами, а записанные в операторе вызова функции — фактическими параметрами или аргументами.

Существует два способа передачи параметров в функцию: по значению и по адресу.

При передаче по значению в стек заносятся копии значений аргументов, и операторы функции работают с этими копиями. Доступа к исходным значениям параметров у функции нет, а следовательно нет и возможности их изменить.

При передаче по адресу в стек заносятся копии адресов аргументов, а функция осуществляет доступ к ячейкам памяти по этим адресам и может изменить исходные значения аргументов.

При передаче по ссылке в функцию передается адрес указанного при вызове параметра, а внутри функции все обращения к параметру неявно разыменовываются [1].

### *1.4.2 Виды функций*

Рекурсивной называется функция, которая вызывает саму себя. Такая рекурсия называется прямой. Существует еще косвенная рекурсия, когда две или более функций вызывают друг друга.

Перегруженные функции (function overloading) – функции с разными прототипами, но одинаковыми названиями. Прототипы отличаются числом и типом аргументов, типом возвращаемого значения.

Шаблонная функция (template function) – функция, контролирующая соответствие типов данных, которые задаются ей как параметры. Объявление шаблонной функции имеет вид: template <class имя_типа1, …, имя_типаN> [2].

### *1.4.3 Пример программы*

Разработайте программу с перегруженными функциями, выполняющими действия:

- вывод на экран линии в 50 символов '#';
- вывод заданного символа заданное число раз;
- печать 50 символов с возможностью задания символа.

Программный код представлен в листинге 1.6.

Листинг 1.6. Перегруженная функция

```
// ЛабРаб9.cpp.
#include "stdafx.h"
#include<iostream>
using namespace std;
void simvol() {
   cout<<"Линия в 50 символов '#': "<<endl;
   for(int i=0; i!=50; i++) {cout<<"#";}
   cout<<endl;
}
char simvol(char c1, int n) {
```

```
    for(int i=0; i!=n; i++) printf("%c", c1);
    cout<<endl;
    return 0;
}
char simvol(char c2) {
    for(int i=0; i!=50; i++) printf("%c", c2);
    cout<<endl;
    return 0;
}
int _tmain(int argc, _TCHAR* argv[])
{  setlocale(LC_ALL, "russian");
    simvol();
    int n;
    char c1, c2;
    cout<<"Вывод заданного символа заданное число раз"<<endl;
    cout<<"Введите n= "; cin>>n;
    cout<<"Введите символ: "; cin>>c1;
    simvol(c1, n);
    cout<<"Вывод 50 символов с возможностью задания симво-
ла"<<endl;
    cout<<"Введите символ: "; cin>>c2;
    simvol(c2);
    system("pause");
    return 0;}
```

Результат работы программы приводится на рисунке 1.4.

## *1.4.4 Контрольные вопросы*

1 Что называется функцией?

2 Что такое объявление и определение функции?

3 Какие виды функций существуют?

4 Какие способы передачи параметров в функцию существуют? В чём отличие?

5 Что такое формальные и фактические параметры?

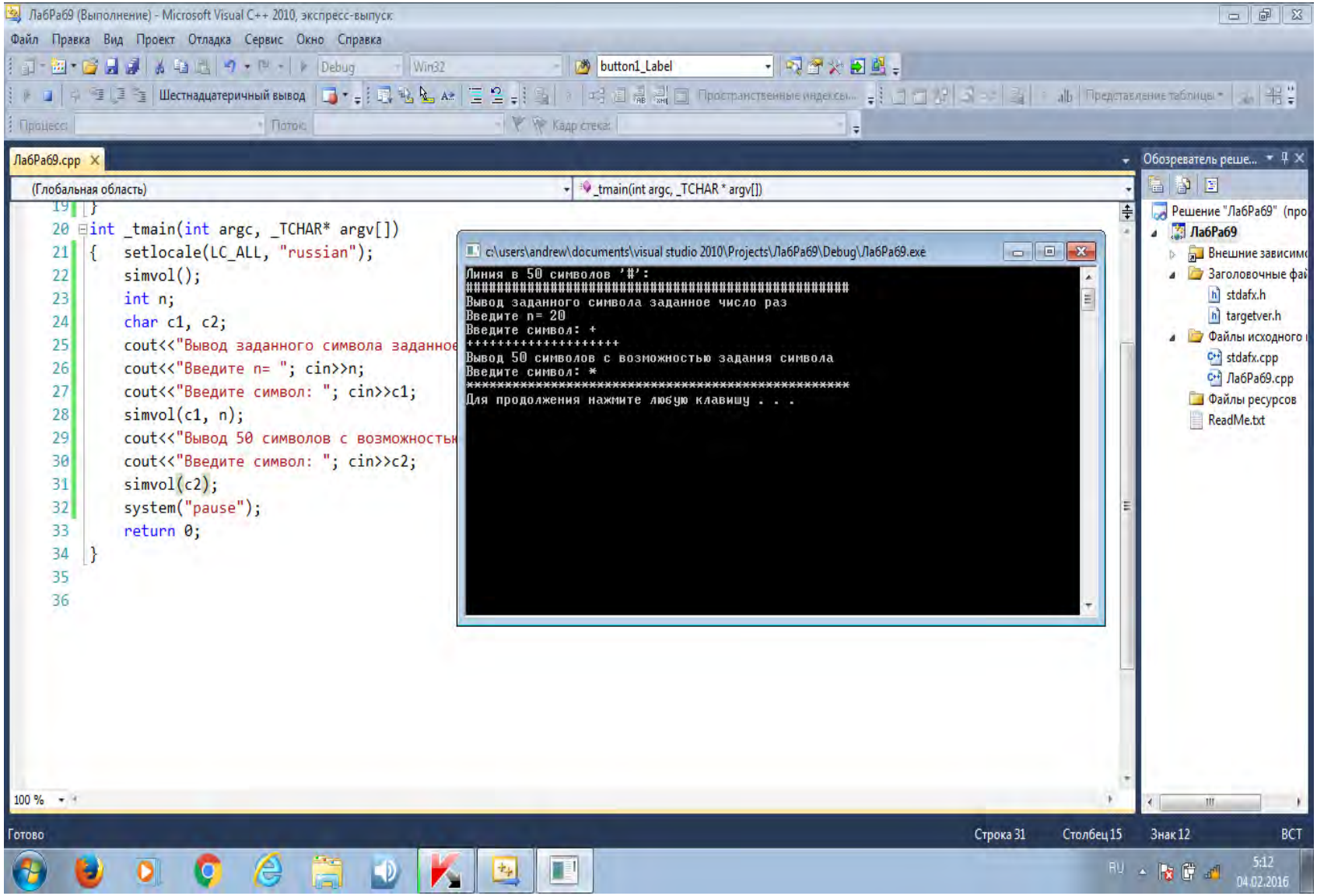


*Рисунок 1.4 – Перегруженная функция*

### *1.4.5 Задания для самостоятельной работы*

1 Удалите из массива вещественных чисел элемент, наименее отличающийся от среднего арифметического значения элементов массива. Создайте функции для вычисления среднего значения элементов массива и удаления элемента, наименее отличающегося от среднего арифметического значения.

2 Замените минимальный элемент массива вещественных чисел на среднее арифметическое значение элементов массива. Создайте функции для вычисления среднего значения элементов массива и определения его минимального элемента.

3 В одномерном массиве, состоящем из *n* элементов, вычислите:

- сумму отрицательных элементов массива;
- произведение элементов массива, расположенных между максимальным и минимальным элементами.

4 В одномерном массиве, состоящем из *n* элементов, вычислите:

- сумму положительных элементов массива;
- произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.

5 Дана целочисленная квадратная матрица $[A_{11*11}]$. Напишите программу, определяющую величины:

- количество ненулевых элементов матрицы;
- максимальный и минимальный четные элементы столбцов матрицы;
- суммы элементов матрицы, находящихся ниже побочной диагонали.

## *1.5 Объектно-ориентированное программирование*

Основные принципы объектно-ориентированного программирования: инкапсуляция, наследование и полиморфизм.

Инкапсуляция — объединение данных с функциями их обработки в сочетании со скрытием ненужной для использования этих данных информации.

Наследование — это возможность создания иерархии классов. Потомки наследуют свойства предков, могут изменять и добавлять новые.

Полиморфизм — возможность использовать в различных классах иерархии одно имя для обозначения сходных по смыслу действий и гибко выбирать требуемое действие во время выполнения программы [1; 2; 3].

## *1.5.1 Классы*

### *1.5.1.1 Конструкторы и деструкторы*

Класс является абстрактным типом данных, определяемым пользователем, и представляет собой модель реального объекта в виде данных и функций для работы с ними. Данные класса называются полями, а функции класса — методами. Поля и методы называются элементами класса. Описание класса выглядит следующим образом:

*class <имя>{*
*[ private:]*
*<описание скрытых элементов>*
*public:*
*<описание доступных элементов>*
*}; // Описание заканчивается точкой с запятой*

Объявление класса (class declaration) представляет собой описание членов класса: данных и методов. Объявление класса называют спецификацией.

Определение объявленных в спецификации класса методов располагается в отдельном файле с расширением .cpp, называемом файлом реализации класса. Когда файлы спецификации и реализации класса находятся в разных файлах, заголовок функции-члена должен включать область видимости согласно формату:

*тип_функции-члена имя_класса :: имя_функци-члена (список параметров)*

Объявление объекта в классах (object definition) – создание переменной класса на основании заданного типа. При объявлении объекта выделяется память и осуществляется инициализация членов-данных. Это выполняет конструктор – специальный метод класса.

Форматы объявления объекта имеют вид:

*имя_класса имя_объекта;*
*имя_класса имя_объекта (список параметров);*

*имя_класса  имя_объекта (имя_объекта_копирования);*

Объект – переменная типа имени класса. Доступ к членам объекта обеспечивается операторами точка и стрелка.

Конструктор (constructor) – специальный метод, имеющий имя, совпадающее с именем класса, и предназначенный для инициализации данных. При создании объекта автоматически вызывается конструктор.

Деструктор (destructor) – специальный метод класса, используемый для разрушения объектов класса. Имя деструктора совпадает с именем конструктора (именем класса), которому предшествует символ тильда ~. Деструктор имеет открытый спецификатор доступа и не имеет ни типа, ни параметров. Деструктор управляет уничтожением объекта из оперативной памяти.

Указатель this – неявно определенный указатель на объект. Является скрытой внутренней переменной. Каждый объект имеет свой указатель this [1;2;3].

В листинге 1.7 представлено объектно-ориентированное приложение.

Листинг 1.7. Реализация класса

```
// Book.cpp..
#include "stdafx.h"
#include<iostream>
#include<string>
#include"Book.h"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{  setlocale(LC_ALL, "russian");
   char author[50];
   char *title;
   title=new char[50];
   int year;
      cout<<"Введите автора книги: "; cin>>author; cout<<endl;
      cout<<"Введите название книги: "; cin>>title; cout<<endl;
      cout<<"Введите год издания: "; cin>>year; cout<<endl;
   CBook obj1(author, title, year);
   cout<<"Автор книги: "<<obj1.getAuthor()<<endl;
   cout<<"Название книги: "<<obj1.getTitle()<<endl;
   cout<<"Год издания:"<<obj1.getYear()<<endl;
```

```
    CBook obj2(obj1);
    cout<<"\nАвтор книги: "<<obj2.getAuthor()<<endl;
    cout<<"Название книги: "<<obj2.getTitle()<<endl;
    cout<<"Год издания:"<<obj2.getYear()<<endl;
    char av[15];
    char * nazv;
    int god;
    nazv=new char[15];
    //CBook obj3;
    cout<<"Введите автора книги: "; cin>>av; cout<<endl;
    cout<<"Введите название книги: "; cin>>nazv; cout<<endl;
    cout<<"Введите год издания книги: "; cin>>god; cout<<endl;
    obj1.setAuthor(av);
    obj1.setTitle(nazv);
    obj1.setYear(god);
    cout<<"Автор книги: "<<obj1.getAuthor()<<endl;
    cout<<"Название книги: "<<obj1.getTitle()<<endl;
    cout<<"Год издания:"<<obj1.getYear()<<endl;
    obj2.setAuthor("Пушкин");
    obj2.setTitle("Бородино");
    obj2.setYear(2010);
    cout<<"\nАвтор книги: "<<obj2.getAuthor()<<endl;
    cout<<"Название книги: "<<obj2.getTitle()<<endl;
    cout<<"Год издания:"<<obj2.getYear()<<endl;
    system("pause"); return 0; }
  //Конструктор по умолчанию
  CBook::CBook(): m_year(0), m_pTitle(" ")
  {
    m_author[0]='\0';
  }
  //Конструктор с параметрами
  CBook::CBook(char* author, char* title, int year):m_year(year),
m_pTitle(title)
  {
    strncpy_s(m_author, 50, author, 49);
    if(strlen(author)>49) m_author[49]='\0';
```

```
}
//Конструктор копирования
CBook::CBook(CBook &obj):m_year(obj.m_year)
{
    strcpy_s(m_author, strlen(obj.m_author)+1, obj.m_author);
    m_pTitle=new char[strlen(obj.m_pTitle)+1];
    strcpy_s(m_pTitle, strlen(obj.m_pTitle)+1, obj.m_pTitle);
}
//Деструктор
CBook::~CBook() {delete [] m_pTitle;}
//Установить автора
void CBook::setAuthor(char* author)
{
    strncpy_s(m_author, 50, author, 49);
    if(strlen(author)>49) m_author[49]='\0';
}
void CBook::setTitle(char* title)
{
    delete [] m_pTitle;
    m_pTitle=new char[strlen(title)+1];
    strcpy_s(m_pTitle,strlen(title)+1, title);
}
//Установить год издания
void CBook::setYear(int year)
{m_year=year;}
//Вернуть автора
const char* CBook::getAuthor()
{return m_author;}
//Вернуть названиен
const char* CBook::getTitle(void)
{return m_pTitle;}
//Вернуть год издания
const int CBook::getYear(void)
{return m_year;}
```

### *1.5.1.2 Статические элементы класса*

С помощью модификатора static можно описать статические поля и методы класса.

Статические члены-данные – общие для всех объектов одного класса. Ключевое слово static помещается перед типом данного, которое объявляется статическим. Чтобы статические данные класса стали доступны за пределами класса, необходимо назначить открытый спецификатор доступа public. Если данные объявить закрытыми, то потребуются открытые методы для получения значений.

Статические методы предназначены для обращения к статическим полям класса. Они могут обращаться только к статическим полям и вызывать только другие статические методы класса, потому что им не передается скрытый указатель this. Обращение к статическим методам производится так же, как к статическим полям – через имя класса или через имя объекта [1; 2; 3].

### *1.5.1.3 Дружественные функции и классы*

Дружественные функции применяются для доступа к скрытым полям класса и представляют собой альтернативу методам. Метод, как правило, используется для реализации свойств объекта, а в виде дружественных функций оформляются действия, не представляющие свойства класса, но концептуально входящие в его интерфейс и нуждающиеся в доступе к его скрытым полям, например, переопределенные операции вывода объектов.

Правила описания и особенности дружественных функций:

- дружественная функция объявляется внутри класса, к элементам которого ей нужен доступ, с ключевым словом friend. В качестве параметра ей должен передаваться объект или ссылка на объект класса, поскольку указатель this ей не передается;
- дружественная функция может быть обычной функцией или методом другого ранее определенного класса. На нее не распространяется действие спецификаторов доступа, место размещения ее объявления в классе безразлично;
- одна функция может быть дружественной сразу нескольким классам [1; 2; 3].

### *1.5.1.4 Пример программы*

Создайте класс **Bill** (счет), представляющий собой разовый платёж за телефонный разговор. Класс должен включать поля:

- номер телефона;
- тариф за минуту разговора;
- скидки в процентах;
- время разговора в минутах;
- сумма к оплате.

Реализуйте метод вычисления суммы к оплате. В программе продемонстрируйте создание, инициализацию и обработку массива объектов типа **Bill** с различными исходными данными для вычисления сумм к оплате. Вычислите общую сумму к оплате.

Программный код приведен в листинге 1.8.

Листинг 1.8. Инкапсуляция. Конструктор и деструктор

```
// ЛабРаб11.cpp.Платеж за телефонный разговор
#include "stdafx.h"
#include <iostream>
using namespace std;
class Bill{private:
    long long nomer;
    double tarif;
    double skidka;
    double vremia;
    double oplata;
    friend double input();
public:
    Bill(): nomer(0), tarif(0), skidka(0), vremia(0) {}
    Bill(long long _nomer, double _tarif, double _skidka, double _vremia){
    nomer=_nomer; tarif=_tarif; skidka=_skidka; vremia=_vremia;}
    ~Bill() {}
    double suma() {
    double tmp_oplata, tmp_skidka;
    tmp_oplata=tarif*vremia;
    tmp_skidka=skidka*tmp_oplata/100;
    oplata=tmp_oplata-tmp_skidka;
```

```
    return oplata;
    }
    double input() {
    cout<<"["<<nomer<<"|";
    cout<<tarif<<" | ";
    cout<<skidka<<" | ";
    cout<<vremia<<" | ";
    cout<<oplata<<" ]"<<endl;
        return 0;    }};
int _tmain(int argc, _TCHAR* argv[])
{   setlocale(LC_ALL, "russian");
    Bill mas[2];
    long long _nomer;
    double _tarif, _skidka, _vremia, sum;
    for(int i=0; i<2; i++) {
    cout<<"Введите номер телефона: "; cin>>_nomer; cout<<endl;
    cout<<"Введите тариф: "; cin>>_tarif; cout<<endl;
    cout<<"Введите скидку: "; cin>>_skidka; cout<<endl;
    cout<<"Введите время: "; cin>>_vremia; cout<<endl;
    Bill obj(_nomer, _tarif, _skidka, _vremia);
    mas[i]=obj; mas[i].suma();
    }
    cout<<"номер|тариф|скидка|время|оплата"<<endl;
    for(int i=0; i<2; i++) mas[i].input();
    sum=0;
    for(int i=0; i<2; i++) sum+=mas[i].suma();
    cout<<"Итого: "<<sum<<endl;
    system("pause");
    return 0; }
```

Результат работы программы приведён на рисунке 1.5.

### *1.5.1.5 Контрольные вопросы*

1 Что такое абстракция данных?

2 Какие принципы объектно-ориентированного программирования существуют?

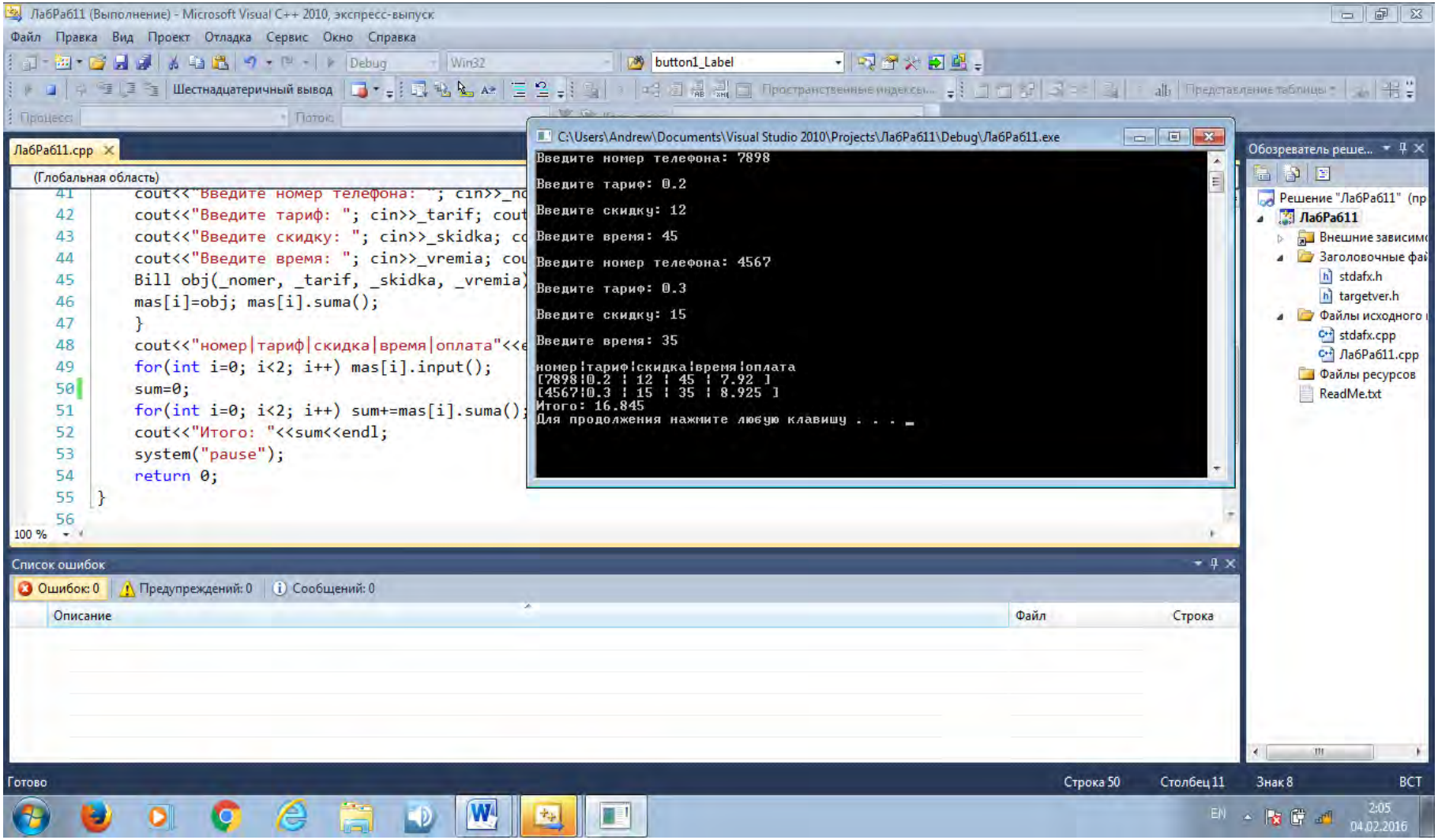


*Рисунок 1.5 – Платёж за телефонный разговор*

3 Что такое конструктор и деструктор?

4 Какие свойства конструктора и деструктора?

5 Что такое объект класса?

### *1.5.1.6 Задания для самостоятельной работы*

1 Создайте класс **Goods** (товары). В классе должны быть представлены поля: наименование товара, дата оформления, цена товара, количество единиц товара, номер накладной, по которой товар поступил на склад. Реализуйте методы изменения цены товара, изменения количества товара (увеличения и уменьшения), вычисления стоимости товара. Метод **toString()** должен выдавать в виде строки стоимость товара.

2 Создайте класс **Payment** (зарплата). В классе должны быть представлены поля: фамилия-имя-отчество, оклад, год поступления на работу, процент надбавки, подоходный налог, количество отработанных дней в месяце, количество рабочих дней в месяце, начисленная и удержанная суммы. Реализуйте методы:

- вычисления начисленной суммы;
- вычисления удержанной суммы;
- вычисления суммы, выдаваемой на руки;
- вычисления стажа.

Стаж вычисляется как полное количество лет, прошедших от года поступления на работу до текущего года. Начисления представляют собой сумму, начисленную за отработанные дни, и надбавки, то есть доли от первой суммы. Удержания представляют собой отчисления в пенсионный фонд (1% от начисленной суммы) и подоходный налог. Подоходный налог составляет 13% от начисленной суммы без отчислений в пенсионный фонд.

3 Создайте класс **Bill** (счет), представляющий собой разовый платеж за телефонный разговор. Класс должен включать в себя поля: номер телефона, тариф за минуту разговора, скидка (в процентах), время разговора (в минутах) и сумма к оплате. Реализуйте метод вычисления суммы к оплате. В программе продемонстрируйте создание, инициализацию и обработку массива объектов типа **Bill** с различными исходными данными для вычисления сумм к оплате. Вычислите общую сумму к оплате.

4 Создайте класс **AvtoVlad**, хранящий информацию о владельце автомобиля: имя, номер автомобиля, номер техпаспорта, дата рождения, телефон, отделение регистрации ГИБДД. Доступ к данным класса организуйте посредством соответствующих методов. Разработайте программу, в которой создается массив объектов данного класса. В программе организуйте:

- ввод данных с клавиатуры;
- вывод данных массива на экран в виде таблицы;
- поиск и вывод информации о владельцах автомобилей, зарегистрированных в отделении ГИБДД;
- поиск и вывод информации об автовладельце по номеру автомобиля.

5 Создайте класс **Abonent**, хранящий информацию о телефонном абоненте: фамилия, имя, дата рождения, адрес и телефонный номер. Доступ к полям класса организуйте посредством соответствующих методов. Разработайте программу, в которой создается массив объектов данного класса. В программе организуйте:

- ввод данных с клавиатуры;
- вывод данных массива на экран в виде таблицы;
- поиск и вывод информации о телефонных абонентах по номеру телефона.

## *1.5.2 Наследование*

### *1.5.2.1 Одиночное и множественное наследование*

Наследование (inheritance) – механизм C++, с помощью которого класс приобретает свойства (данные и методы) другого класса.

Наследование реализует обобщение (generalization) – отношение типа is a. Обобщение указывает на существование общности между объектами и позволяет строить иерархию классов, переходя от общих классов к специализированным версиям.

Базовый класс (base) или родительский (parent) – класс, от которого наследуются данные и методы класса.

Производный класс (derived) или дочерний (child) – класс, объекты которого наследуют данные и методы базового класса.

Доступ к членам базового класса из производного класса управляется посредством ключей доступа: private, protected и public. В таблице 1.5 приведена характеристика ключей доступа к членам базового класса из производного класса [2].

*Таблица 1.5 – Ключи доступа в производном классе к членам базового класса*

<table>
<tr><td rowspan="2">Спецификатор наследования</td><td colspan="3">Ключ доступа базового класса</td></tr>
<tr><td>Public</td><td>Protected</td><td>Private</td></tr>
<tr><td>Public</td><td>в производном классе public</td><td rowspan="2">в производном классе protected</td><td rowspan="3">в производном классе невидим</td></tr>
<tr><td>Protected</td><td>в производном классе protected</td></tr>
<tr><td>Private</td><td>в производном классе private</td><td>в производном классе private</td></tr>
</table>

Одиночное наследование – наследование данных и методов производным классом от одного базового класса. Производный класс может быть базовым классом для других классов программы. Иерархическая организация наследования может иметь несколько уровней.

Множественное наследование – иерархическая структура, в которой производный класс имеет два или более базовых класса. При таком наследовании получается новый класс из других классов, которые никак не связаны между собой. Этот класс наследует от базовых классов открытые и защищенные данные и методы, при необходимости может их дополнить своими собственными.

### *1.5.2.2 Пример программы «Одиночное наследование»*

Создайте класс **Man** (человек) с полями: имя, возраст, пол и вес. Определите методы переназначения имени, изменения возраста и изменения веса. Создайте производный класс **Student**, имеющий поле год обучения. Определите методы переназначения и увеличения года обучения.

Программный код приведён в листинге 1.9.

Листинг 1.9. Наследование: иерархия классов

```
// ЛабРаб13.cpp. Базовый класс Man. производный класс Student
#include "stdafx.h"
```

```
#include <iostream>
#include <string.h>
using namespace std;
class Man {
protected:
    char name[30];
    double age;
    char pol[3];
    double ves;
public:
    Man(char _name[], double _age, char _pol[], double _ves)
    {strcpy(name, _name); age=_age; strcpy(pol, _pol); ves=_ves;}
    ~Man() {}
    void out() {
        cout<<name<<endl;
        cout<<age<<endl;
        cout<<pol<<endl;
        cout<<ves<<endl;}};
class Student: public Man {
private:
    int god;
public:
    Student(double _god, char _name[], double _age, char _pol[], double
_ves):
      Man(_name, _age, _pol, _ves) {god=_god;}
      void out() {cout<<"Имя: "<<name<<endl; cout<<"Возраст:
"<<age<<endl;
      cout<<"Пол: "<<pol<<endl; cout<<"Вес: "<<ves<<endl;
cout<<"Год: "<<god<<endl;}
      void ren() {cout<<"-------------------------"<<endl;
    cout<<"Введите имя: "; gets(name); cin.sync();
    cout<<"Введите возраст: "; cin>>age; cin.sync();
    cout<<"Введите пол: "; gets(pol); cin.sync();
    cout<<"Введите вес: "; cin>>ves; cin.sync();
    cout<<"Введите год: "; cin>>god;
    cout<<"-------------------------"<<endl;}};
```

```
int _tmain(int argc, _TCHAR* argv[])
{  setlocale(LC_ALL, "russian");
   Student abc(1995, "Ivanov", 20, "M", 75);
   abc.out();
   abc.ren();
   abc.out();
   system("pause");
   return 0;}
```

Результат работы программы приведён на рисунке 1.6.

### *1.5.2.3 Чистые виртуальные функции и абстрактные классы*

Виртуальная функция (virtual function) – функция-член, объявленная в базовом классе и переопределенная в производном классе. В производном классе эта функция будет виртуальной.

Полиморфный класс – класс, содержащий хотя бы одну виртуальную функцию.

Для создания виртуальной функции применяется ключевое слово virtual.

Чистая виртуальная функция (pure virtual function) – это виртуальная функция, которая не имеет реализации в базовом классе. Представляет собой общий прототип для определений функций, которые располагаются в производных классах.

Абстрактный класс (abstract class) – класс, в котором объявлена хотя бы одна чистая виртуальная функция. Для абстрактного класса невозможно создать объекты. Используются в качестве базовых для других классов. Если в производном классе, который наследует от абстрактного базового класса, отсутствует реализация чистой виртуальной функции, то этот производный класс будет являться абстрактным.

Объявление прототипа чистой виртуальной функции в базовом классе имеет вид:

virtual тип_функции имя_функции (список_параметров_функции)=0;.

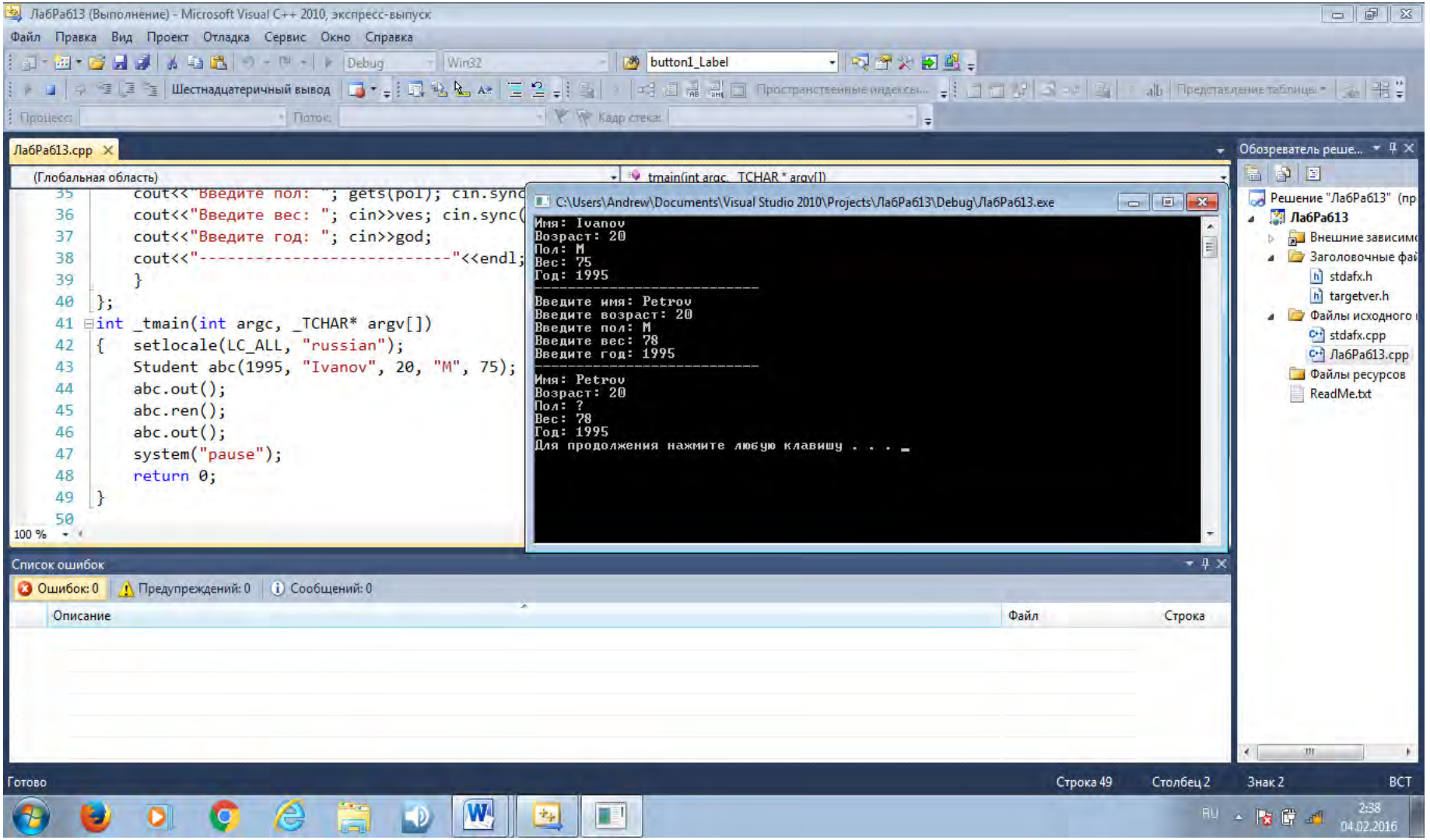


*Рисунок 1.6 – Одиночное наследование*

### *1.5.2.4 Пример программы «Абстрактный класс»*

Создайте абстрактный класс **Curves** (кривые) вычисления координаты y для значения x. Создайте производные классы: прямая, эллипс, гипербола со своими функциями вычисления y в зависимости от входного параметра x.

Уравнение прямой: $y = ax + b$, эллипса: $y = x^2 / a^2 + y^2 / b^2 = 1$, гиперболы: $y = x^2 / a^2 - y^2 / b^2 = 1$.

Программный код приведён в листинге 1.10.

Листинг 1.10. Полиморфизм и виртуальные функции

```
// ЛабРаб14.cpp. Абстрактный класс
#include "stdafx.h"
#include <iostream>
#include <math.h>
using namespace std;
class curves {
public:
    void virtual calculation(double x)=0;
};
class straight: public curves {
public:
    void calculation(double x){
    double a=10;
    double b=15;
    cout<<"Straight: y=a*x+b"<<endl<<"y="<<x*a+b<<endl;
    }};
class ellipse: public curves {
public:
        void calculation(double x){
        double a=10;
        double b=15;
        cout<<"Ellipse:
1=(x*x)/(a*a)+(y*y)/(b*b)"<<endl<<"y="<<sqrt((1-
(x*x)/(a*a))*b*b)<<endl;
        }
```

```
    };
class hyperbole: public curves {
public:
    void calculation(double x) {
    double a=10;
    double b=15;
    cout<<"Hyperbole:                    1=(x*x)/(a*a)-(y*y)/(b-
b)"<<endl<<"y="<<sqrt(((x*x)/(a*a)-1)*b*b)<<endl;
    }
};
int _tmain(int argc, _TCHAR* argv[])
{  setlocale(LC_ALL, "russian");
    hyperbole hyper;
    ellipse ell;
    straight str;
    hyper.calculation(20);
    ell.calculation(1);
    str.calculation(20);
    system("pause");
    return EXIT_SUCCESS;}
```

Результат работы программы приведён на рисунке 1.7.

### *1.5.2.5 Контрольные вопросы*

1 Что такое наследование?
2 Какие виды наследования существуют?
3 Что такое виртуальная функция?
4 Что такое абстрактный класс?
5 Что такое полиморфный класс?

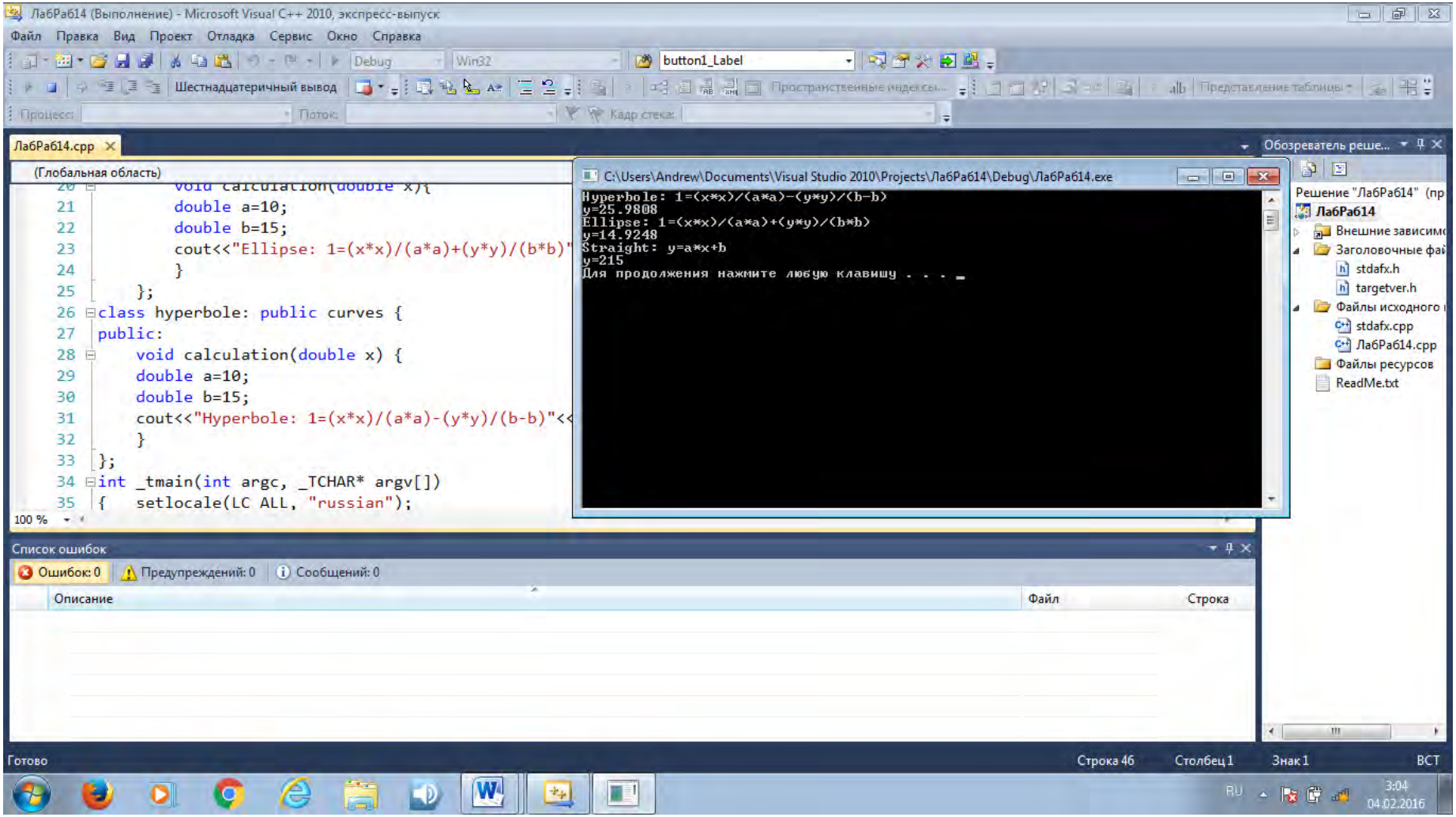


*Рисунок 1.7 – Абстрактный класс Curves*

### *1.5.2.6 Задания для самостоятельной работы*

1 Создайте базовый класс **Complex** (комплексное число) для реализации комплексных чисел в алгебраической форме и основных операций с ними: сложения, вычитания, умножения и деления. Создайте производный класс, в котором определите операции вычисления модуля комплексного числа и комплексно спряженного.

2 Создайте класс **Liquid** (жидкость), имеющий поля названия и плотности. Определите методы переназначения и изменения плотности. Создайте производный класс **Alcohol** (спирт), имеющий поле крепости. Определите методы переназначения и изменения крепости.

3 Создайте базовый класс **Complex** (комплексное число) для реализации комплексных чисел в тригонометрической форме и основных операций с ними: сложения, вычитания, умножения и деления. Создайте производный класс, в котором определите операции вычисления действительной и мнимой частей комплексного числа, а также комплексно спряженного.

4 Создайте класс **Man** (человек) с полями: имя, возраст, пол и вес. Определите методы переназначения имени, изменения возраста и изменения веса. Создайте производный класс **Student**, имеющий поле года обучения. Определите методы переназначения и увеличения года обучения.

5 Создайте класс **Triangle** (треугольник) с полями-сторонами. Определите методы изменения сторон, вычисления углов, вычисления периметра. Создайте производный класс **Equilateral** (равносторонний), имеющий поле площади. Определите метод вычисления площади.

## *1.5.3 Перегрузка операторов*

### *1.5.3.1 Операторные функции*

В языке C++ большинство операторов возможно перегружать. Нельзя перегружать операторы препроцессора, .(точка), ::, .*, ?. Для перегрузки оператора создается операторная функция (operator function).

Форма записи операторной функции имеет вид

*тип имя-класса::operator перегруженный_оператор (список_аргументов) { /*тело функции*/}.*

### *1.5.3.2 Перегрузка бинарных операторов*

Формат операторной функции бинарного оператора, объявленной нестатической функцией-членом, имеет вид

*тип operatorОперация(тип_параметра параметр);*

В *параметр* передается объект, стоящий справа от оператора. Объект, стоящий слева от оператора, передается неявно с помощью указателя this.

Формат операторной функции бинарного оператора, объявленной глобальной, представляется в виде

*тип operatorОперация(тип_параметра1 параметр1, тип_параметра2 параметр2);*

Один из параметров должен иметь тип класса, для которого перегружается оператор [2].

### *1.5.3.3 Перегрузка унарных операторов*

Формат операторной функции унарного оператора, объявленной нестатической функцией-членом, имеет вид

*тип operatorОперация();*

Формат операторной функции унарного оператора, объявленной глобальной, представляется в виде

*тип operatorОперация(тип_параметра параметр);* [2].

### *1.5.3.4 Пример программы*

Создайте класс **Complex** (комплексное число) в алгебраической форме z=x+i*y, включающий два поля класса: действительную часть (x) и мнимую часть (y) числа. Реализуйте методы вычисления модуля комплексного числа, возведения комплексного числа в степень и вывода комплексного числа. Перегрузите операции сложения, вычитания, деления и умножения комплексных чисел.

Программный код приведён в листинге 1.11.

Листинг 1.11. Полиморфизм: перегрузка операций

```
//ЛабРаб13.cpp. Комплексные числа
#include "stdafx.h"
```

```
#include <iostream>
using namespace std;
class Complex {
    double x;
    double y;
public:
    Complex(): x(0), y(0) {}
    Complex(double x1, double y1) {x=x1; y=y1;}
    ~Complex() {}
    Complex operator+(const Complex& aComplex) const;
    Complex operator-(const Complex& aComplex) const;
    Complex operator*(const Complex& aComplex) const;
    Complex operator/(const Complex& aComplex) const;
    double modul() {return hypot(x, y);}
    double step(int n) {
    Complex aComplex(x, y);
    Complex tmp;
    tmp.x=pow(aComplex.modul(), n)*cos(n*x/aComplex.modul());
    tmp.y=pow(aComplex.modul(), n)*sin(n*y/aComplex.modul());
    tmp.Cout();
    return 0;
    }
    void Cout() const {
    cout<<x<<" ";
    if(y>=0) cout<<"+i";
    else cout<<"-i";
    cout<<fabs(y)<<endl;
    }
};
int _tmain(int argc, _TCHAR* argv[])
{  setlocale(LC_ALL, "russian");
    int x1, y1, x2, y2;
    cout<<"Введите действительную часть первого числа; "; cin>>
x1; cout<<endl;
    cout<<"Введите мнимую часть первого числа; "; cin>> y1;
cout<<endl;
```

```
    cout<<"Введите действительную часть второго числа; "; cin>> x2; cout<<endl;
    cout<<"Введите мнимую часть второго числа; "; cin>> y2; cout<<endl;
    Complex z1(x1, y1); cout<<"z1="; z1.Cout();
    Complex z2(x2, y2); cout<<"z2="; z2.Cout();
    Complex aComplex;
    aComplex=z1+z2; cout<<"z1+z2="; aComplex.Cout(); cout<<endl;
    aComplex=z1-z2; cout<<"z1-z2="; aComplex.Cout(); cout<<endl;
    aComplex=z1*z2; cout<<"z1*z2="; aComplex.Cout(); cout<<endl;
    aComplex=z1/z2; cout<<"z1/z2="; aComplex.Cout(); cout<<endl;
    cout<<"Модуль z1=";
    cout<<z1.modul()<<endl;
    int n;
    cout<<"Введите n="; cin>>n; cout<<endl;
    cout<<"z1 в степени n="<<n<<" = "; z1.step(n);
    system("pause");
    return 0;
}
Complex Complex::operator+(const Complex& aComplex) const {
    Complex tmpComplex;
    tmpComplex.x=x+aComplex.x;
    tmpComplex.y=y+aComplex.y;
    return tmpComplex;
}
Complex Complex::operator-(const Complex& aComplex) const
{return Complex(x-aComplex.x, y-aComplex.y);}
Complex Complex::operator*(const Complex& aComplex) const
{return Complex(x*aComplex.x-y*aComplex.y, x*aComplex.y+y*aComplex.x);}
Complex Complex::operator/(const Complex& aComplex) const
{ return Complex((x*aComplex.x-y*aComplex.y)/(pow(aComplex.x, 2)+pow(aComplex.y, 2)), (aComplex.x*y-aComplex.y*x)/(pow(aComplex.x, 2)+pow(aComplex.y,2)));}
```

Результат работы программы приведён на рисунке 1.8.

### *1.5.3.5 Контрольные вопросы*

1 Что называется перегрузкой операторов?
2 Какие операции не перегружаются?
3 Что называется операторной функцией?
4 Как перегружаются бинарные операторы?
5 Как перегружаются унарные операторы?

### *1.5.3.6 Задания для самостоятельной работы*

1 Создайте класс **Matrix** (матрица), включающий поля: количество строк, количество столбцов, элементы матрицы. Реализуйте методы вывода матрицы и вычисления определителя. Перегрузите операции сложения, вычитания, деления, умножения матриц, умножения матрицы на число.

2 Создайте класс **StraightLine** (прямая линия), включающий координаты двух точек $(x_1, y_1)$ и $(x_2, y_2)$. Реализуйте методы вывода уравнения прямой $y = a * x + b$. Перегрузите операции проверки параллельности прямых (||) и определения угла между двумя прямыми (%).

3 Создайте класс **Complex** (комплексное число) в алгебраической форме $z = x + i * y$, включающий два поля класса: действительную часть (x) и мнимую часть (y) числа. Реализуйте методы вычисления модуля комплексного числа, возведения комплексного числа в степень и вывода комплексного числа. Перегрузите операции сложения, вычитания, деления и умножения комплексных чисел.

4 Создайте класс **Fraction** (обыкновенная дробь), включающий поля: числитель и знаменатель. Реализуйте методы определения обратной дроби и вывода дроби. Перегрузите операции сложения, вычитания и умножения дробей.

5 Создайте класс **Vector** (вектор), включающий поле координат вектора. Реализуйте методы определения направляющих косинусов вектора и вывода всех характеристик вектора. Перегрузите операции сложения (+), скалярного (%) и векторного (*) произведения векторов.

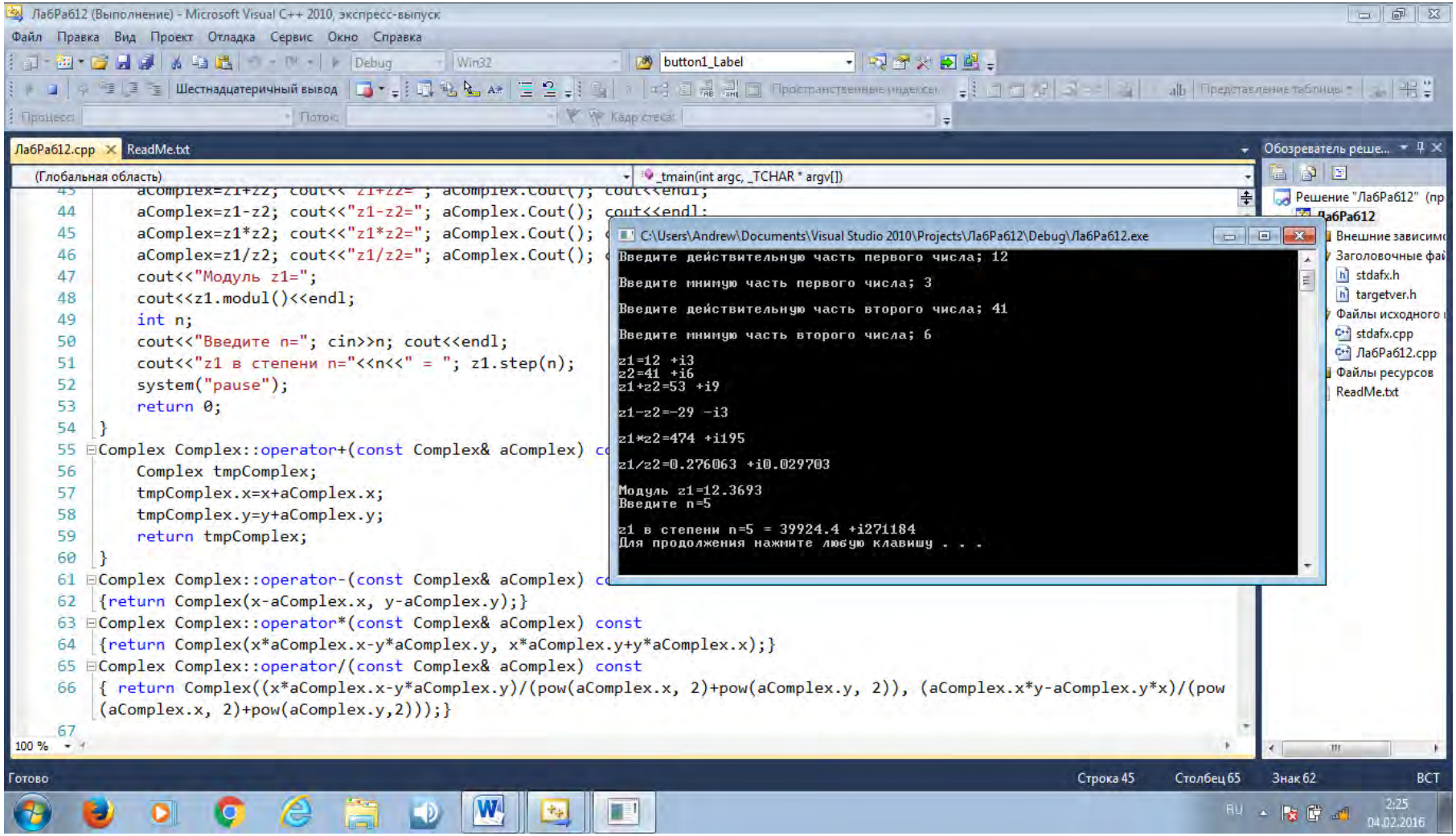


*Рисунок 1.8 – Комплексное число*

# 2 ПРИМЕНЕНИЕ ЯЗЫКА C++ В РЕШЕНИИ ПРИКЛАДНЫХ ЗАДАЧ

## 2.1 Разработка визуальных приложений в среде программирования Microsoft Visual C++

Среда программирования Microsoft Visual C++ – среда программирования быстрой разработки, в основе которой лежит технология визуального проектирования (создание диалоговых окон) и событийного программирования (разработка функций обработки событий).

Среда программирования Microsoft Visual C++ позволяет разрабатывать .NET-приложения. Технология Microsoft .NET основана на идее универсального программного кода, который может быть выполнен любым компьютером вне зависимости от операционной системы.

Универсальность программного кода обеспечивается за счет выполняемой на этапе разработки компиляции исходной программы в универсальный промежуточный код (CIL-код, Common Intermediate Language), который во время запуска программы транслируется в выполняемый программный код. Преобразование промежуточного кода в выполняемый производит JIT-компилятор (Just In Time, в тот же момент), являющийся элементом виртуальной выполняющей системы (Virtual Execution System, VES). Выполнение .NET-приложений в операционной системе Microsoft Windows обеспечивает Common Language Runtime (CLR, общеязыковая исполняющая среда).

Процесс создания программы включает два этапа: создание формы и разработку функции обработки события.

Форма создается путем помещения компонентов и их настройки. К базовым компонентам относятся компоненты: label – поле отображения информации, TextBox – поле ввода-редактирования данных, Button – командная кнопка, CheckBox – флажок, RadioButton – радио-кнопка, ListBox – список выбора, ComboBox – поле редактирования со списком выбора.

Вид компонента и его поведение определяют значения свойств компонента. Функции обработки событий выполняют основную работу в программе [4; 5; 6].

## *2.1.1 Основные компоненты*

Компонент (component) – объект, предназначенный для решения задачи.

Компоненты обеспечивают ввод данных, отображение результатов, доступ к базам данных, решение других задач [3, с. 61].

К базовым (основным) компонентам относят:

- Label – поле отображения информации;
- TextBox – поле ввода-редактирования текста (данных);
- Button – командная кнопка;
- CheckBox – флажок;
- RadioButton – радио-кнопка;
- ListBox – список выбора;
- ComboBox – поле редактирования со списком выбора [4].

Компоненты располагаются в палитре компонентов.

## *2.1.2 Пример программы*

**Задача.** Разработать визуальное приложение, формализующее алгоритм расчета. Компоненты TextBox, Label, Button.

Решение:

- запустить среду программирования **Microsoft Visual Studio 2010 Professional**;
- команда **Файл-> Создать-> Проект**. Появится окно **Создать окно**;
- среда **CLR**. Узел **Windows Forms Application Visual C++**. Ввести Имя. Нажать кнопку **ОК**;
- задать свойства формы;
- перенести на форму компоненты **TextBox** , **Label** и **Button**;
- создать функцию обработчика события;
- откомпилировать. Ввести данные. Получить результат (рисунок 2.1).

В листинге 2.1. приведен программный код.

Листинг 2.1. Визуальное приложение

```
#pragma endregion
        // щелчок на кнопке ОК
    private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
```

```
{  double  mile;  // расстояние в милях
   double km;     // растояние в километрах
   // Если в поле редактирвания нет данных,
   // то при попытке преобразовать пустую
   // строку в число возникает исключение.
   try {  mile=Convert::ToDouble(textBox1->Text);
          km=mile*1.609344;
          label2->Text=mile.ToString("n") + " мили - " +
             km.ToString("n") + " км";              }
   catch (System::FormatException^ ex)
   {  // обработка исключений
      // - сообщение
      MessageBox::Show (
         "Надо ввести исходные данные", "Мили-километры",
         MessageBoxButtons::OK,
         MessageBoxIcon::Exclamation);
      // установить курсор в поле редактирования
      textBox1->Focus();}   }
private:  System::Void  textBox1_KeyPress(System::Object^    sender,
System::Windows::Forms::KeyPressEventArgs^  e)
{  // правильными символами считаются цифры,
   // запятая, <Enter> и <Backspace>.
   // Будем считать правильным символом
   // точку, но заменим ее запятой.
   // Остальные символы запрещены.
   // Чтобы запрещенный символ не отображался
   // в поле редактирования, присвоим
   // значение true свойству Handled параметра e
    if ((e->KeyChar>='0')&&(e->KeyChar<='9'))
    {  // цифра
       return;              }
    if (e->KeyChar=='.')
    {  // точку заменим запятой
       e->KeyChar=',';
    }
    if (e->KeyChar==',') {
```

```
    {  if (textBox1->Text->IndexOf(',')!=-1)
       // запятая уже есть в поле редактирования
       e->Handled=true;
    } return;
}
   if (Char::IsControl (e->KeyChar))
   {  //  <Enter>, <Backspace>, <Esc>
      if (e->KeyChar==(char) Keys::Enter)
         // нажата клавиша <Enter>
         // установить "фокус" на кнопку OK
         button1->Focus();
         return;} }};}
```

Результат работы программы приведён на рисунке 2.1.

### *2.1.3 Контрольные вопросы*

1 Какова сущность технологии визуального проектирования и событийного программирования?

2 Какие этапы разработки визуального приложения?

3 Что такое компонент?

4 Какие базовые компоненты?

5 Что такое универсальный промежуточный код?

### *2.1.4 Задания для самостоятельной работы*

1 Судоходная компания «Балтика» занимается перевозками грузов между континентами. В ее собственности несколько десятков судов различного класса и грузоподъемности. К услугам компании обращаются клиенты разных стран мира. На судне может находиться несколько партий грузов для различных грузополучателей из разных стран и городов. Одна партия груза может состоять из нескольких разновидностей грузов. У одной партии груза может быть только один отправитель и один получатель. Маршрут следования судна разрабатывается главным менеджером компании и проходит через несколько портов. В порту назначения производится частичная погрузка и выгрузка грузов, и судно следует дальше.

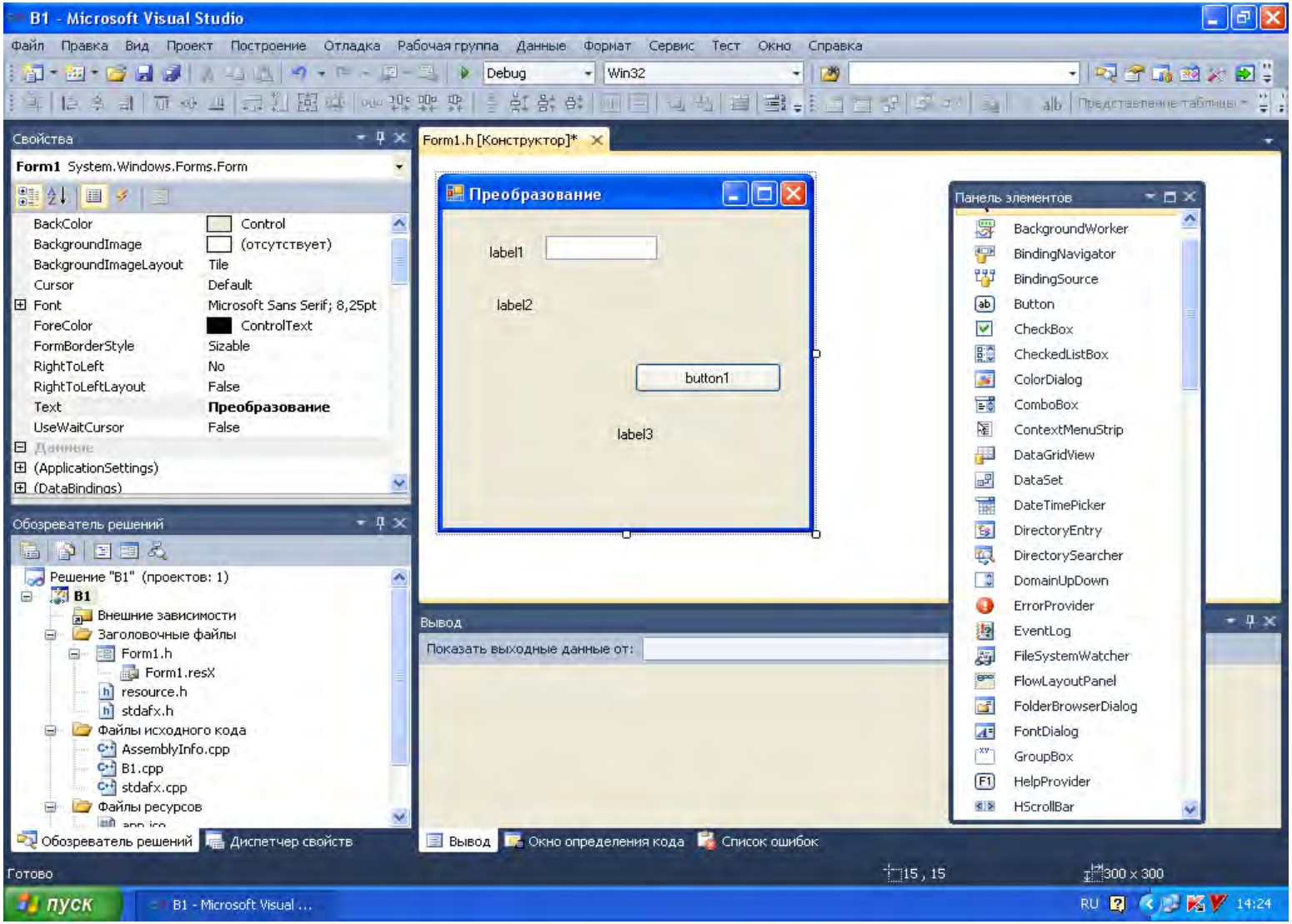


*Рисунок 2.1 – Размещение компонентов (элементов) на форме*

2 Учреждение юстиции регистрирует права юридических и физических лиц на недвижимое имущество (здания, квартиры, земельные участки). Разработайте программу регистрации прав граждан на квартиры. В здании несколько квартир. В одной квартире несколько собственников.

3 Малое научно-внедренческое предприятие «Квадро» занимается прокладкой компьютерных сетей и разработкой программных комплексов для организаций. Численность работников в «Квадро» – 80 человек. Одновременно находится в разработке до 30 проектов. Один разработчик может участвовать в нескольких проектах одновременно, но зарплата его от этого не зависит. Одна организация может заказать в «Квадро» несколько разработок. Стоимость каждого проекта оговаривается отдельно. При досрочном выполнении работы заказчик перечисляет научно-внедренческому предприятию определенный, заранее оговоренный процент премии.

4 Общество с ограниченной ответственностью «Киновидеопрокат» контролирует демонстрацию кинофильмов в кинотеатрах города. Отдел маркетинга принимает решение о покупке кинолент. Отдел закупок претворяет решения в жизнь. Лента покупается у производителя и посредника. Отдел аренды киновидеопроката сдает фильмы кинотеатрам города в аренду.

5 Предприятие LADA-сервис занимается продажей автомобилей марки ВАЗ. Покупатель может заказать модель, цвет, тюнинг и договориться о сроках поставки автомобиля. Одновременно с новыми авто на площадках компании имеется большой выбор подержанных автомобилей отечественных и иностранных моделей.

**Вариант 10.** Торгово-посредническая фирма «Столица» делает закупки продуктов питания в регионах страны и за рубежом. Часть продукции закупается у местных продавцов. Фирма получает прибыль за счет того, что крупные партии товара стоят дешевле, чем мелкие. Товар не может быть продан дешевле, чем он куплен.

## *2.2 Основы структур данных*

Динамические структуры данных – структуры, размер которых изменяется во время выполнения программы. Память выделяется по мере необходимости отдельными блоками, связанными друг с другом с помо-

щью указателей. Например, линейные списки, стеки, очереди и бинарные деревья. Они различаются способами связи отдельных элементов и допустимыми операциями. Динамическая структура может занимать несмежные участки оперативной памяти.

Программа предназначена для обработки данных. Алгоритм зависит от способа организации данных. Выбор структур данных предшествует созданию алгоритмов.

### *2.2.1 Линейные структуры данных*

Линейные структуры данных – линейный список, очередь, стек и дек.

Линейный список – структура данных, состоящая из элементов одного типа, связанных между собой.

Линейные списки реализуются при помощи массивов и связных списков.

Элемент списка содержит две части: основное поле и вспомогательное поле. Основное поле содержит информацию. Например, символ. Вспомогательное поле содержит указатель на следующий элемент. Головной элемент содержит указатель Start. Последний элемент линейного списка хранит указатель NULL.

Линейный список имеет следующие характеристики.

1 Длина списка. Количество элементов в списке.

2 Списки могут быть типизированными или нетипизированными. Если список типизирован, то тип его элементов задан, и все его элементы должны иметь типы, совместимые с заданным типом элементов списка. Обычно списки, реализованные при помощи массивов, являются типизированными.

3 Список может быть сортированным или несортированным;

4 В зависимости от реализации может быть возможен произвольный доступ к элементам списка.

Над списками выполняются операции.

1 создание первого элемента;

2 добавление элемента в конец списка;

3 чтение элемента с заданным ключом;

4 вставка элемента в заданное место списка;

5 удаление элемента с заданным ключом;

6 упорядочивание списка по ключу.

В листинге 2.2 приведена программа, формализующая универсальную очередь ограниченного размера с использованием шаблонного класса.

Листинг 2.2. Универсальная очередь

```
//NODE.H Определение шаблона Node
#ifndef NODE_H //Исключение повторной компиляции файлов
#define NODE_H
template<class T> class Node {
public:
   T data;      //Данные элемента очереди
   Node*next;    //Указатель на следующий элемент очереди
   Node(const T &);   //Конструктор
   T getData() const;  //Получить данные
};
//Определение конструктора шаблонного класса Node
template<class T> Node<T>::Node(const T &info) {
   data=info;
   next=0;}
//Определение метода getData()  const шаблонного класса Node
template<class T> T Node<T>::getData() const { return data;}
#endif
//QUEUE.H Определение шаблона Queue
#ifndef QUEUE_H //Исключение повторной компиляции файла
#define QUEUE_H
using namespace std;
//определение шаблонного класса Queue
template<class T> class Queue {
   friend class Node<T>;
   Node<T> *pbeg, *pend;
public:
   Queue();   //Конструктор по умолчанию (без параметров)
   ~Queue();  //Деструктор
   void insertAtFront(T &); //Вставка элемента в начало очереди
   void insertAtBack(T &);      //Вставка элемента в конец очереди
   int removeFromFront(T &);//Удаление элемента из начала очереди
   int removeFromBack(T &); //Удаление элемента из конца очереди
```

```
    int isEmpty() const;    //Проверка состояния очереди (наличие элементов)
    void print() const;         //Вывод содержимого очереди
};
//Определение конструктора Queue()
template<class T> Queue<T>::Queue() { pbeg=pend=0; }
//Определение деструктора ~Queue()
template<class T> Queue<T>::~Queue() {
    if(!isEmpty()) { cout<<"Удаление..."<<endl;
    Node<T> *current=pbeg, *temp;
    while(current!=0) {
        temp=current;
        cout<<temp->data<<endl;
        current=current->next;
        delete temp;}
    } cout<<"\nВсе элементы удалены!\n"<<endl;
}
//Определение метода insertAtFront()
template<class T> void Queue<T>::insertAtFront(T &value) {
    Node<T> *pnew=new Node<T>(value);
    if(isEmpty()) pbeg=pend=pnew;
    else {
        pnew->next=pbeg;
        pbeg=pnew;}
}
//Определение метода insertAtBack()
template<class T> void Queue<T>::insertAtBack(T &value) {
    Node<T> *pnew=new Node<T>(value);
    if(isEmpty()) pbeg=pend=pnew;
    else {
        pend->next=pnew;
        pend=pnew;}}
//Определение метода removeFromFront()
template<class T> int Queue<T>::removeFromFront(T &value) {
    if(isEmpty()) return 0;
    else {
```

```
        Node<T> *temp=pbeg;
        if(pbeg==pend) pbeg=pend=0;
            else pbeg=pbeg->next;
        value=temp->data;
            delete temp;
                return 1;    }}
//Определение метода removeFromBack()
template<class T> int Queue<T>::removeFromBack(T &value) {
if(isEmpty()) return 0;
    else {
        Node<T> *temp=pend;
        if(pbeg==pend)
                pbeg=pend=0;
            else { Node<T> *current=pbeg;
            while(current->next!=pend)
                current=current->next;
            pend=current;
            current->next=0;   }
        value=temp->data;
            delete temp;
                return 1;    }}
//Определение метода isEmpty()
template<class T> int Queue<T>::isEmpty() const { return pbeg==0; }
//Определение метода print()
template<class T> void Queue<T>::print() const {
    if(isEmpty()) { cout<<" \nОчередь пуста!"<<endl<<endl; return;}
    Node<T> *current=pbeg;
    cout<<"В очереди...";
    while(current) { cout<<current->data<<" "; current=current->next;}
    cout<<endl<<endl;
}
#endif

// QueueTemplate.cpp: Универсальная очередь ограниченного размера
//с использованием шаблонного класса
#include "stdafx.h"
```

```
#include <iostream>
#include "NODE.h"
#include "QUEUE.h"

void testInteger();
void instruction();

int _tmain(int argc, _TCHAR* argv[])
{   setlocale(LC_ALL, "russian");
    testInteger();
    system("pause");
    return 0;
}
void testInteger() {
    Queue<int> integer;
    instruction();
    int choice, value;
    do { cout<<"? "; cin>>choice;
    switch(choice) {
    case 1: cout<<"\nВведите целое: "; cin>>value;
            integer.insertAtFront(value);
            integer.print();
            break;
    case 2:  cout<<"\nВведите целое: "; cin>>value;
            integer.insertAtBack(value);
            integer.print();
            break;
    case 3:  if(integer.removeFromFront(value))
                cout<<value<<" удаляется из очереди"<<endl;
            integer.print();
            break;
    case 4:  if(integer.removeFromBack(value))
                cout<<value<<" удаляется из очереди"<<endl;
            integer.print();
            break;
    }
```

```
        } while(choice!=5);
}
void instruction() {
    cout<<"Выберите: "<<endl
        <<"1 - вставить в начало"<<endl
        <<"2 - вставить в конец"<<endl
        <<"3 - удалить из начала"<<endl
        <<"4 - удалить из конца"<<endl
        <<"5 - удалить все элементы"<<endl;
}
```

Результат работы программы приведен на рисунке 2.2.

### *2.2.2 Нелинейные структуры данных*

Нелинейные структуры данных – иерархические структуры дерева.

Бинарное дерево — это динамическая структура данных, состоящая из узлов, каждый из которых содержит данные и не более двух ссылок на узлы. На каждый узел имеется ровно одна ссылка. Начальный узел называется корнем дерева. Лист – узел, не имеющий поддеревьев. Предок – исходящий узел. Входящий узел – потомок.

Высота дерева определяется количеством уровней, на которых располагаются его узлы. Если дерево организовано таким образом, что для каждого узла все ключи его левого поддерева меньше ключа этого узла, а все ключи его правого поддерева – больше, оно называется деревом поиска. Одинаковые ключи не допускаются. В дереве поиска можно найти элемент по ключу, двигаясь от корня и переходя на левое или правое поддерево в зависимости от значения ключа в каждом узле.

Для бинарных деревьев определены операции:

- включения узла в дерево;
- поиска по дереву;
- обхода дерева;
- удаления узла.

В листинге 2.3 приведена программа, формализующая бинарное дерево.

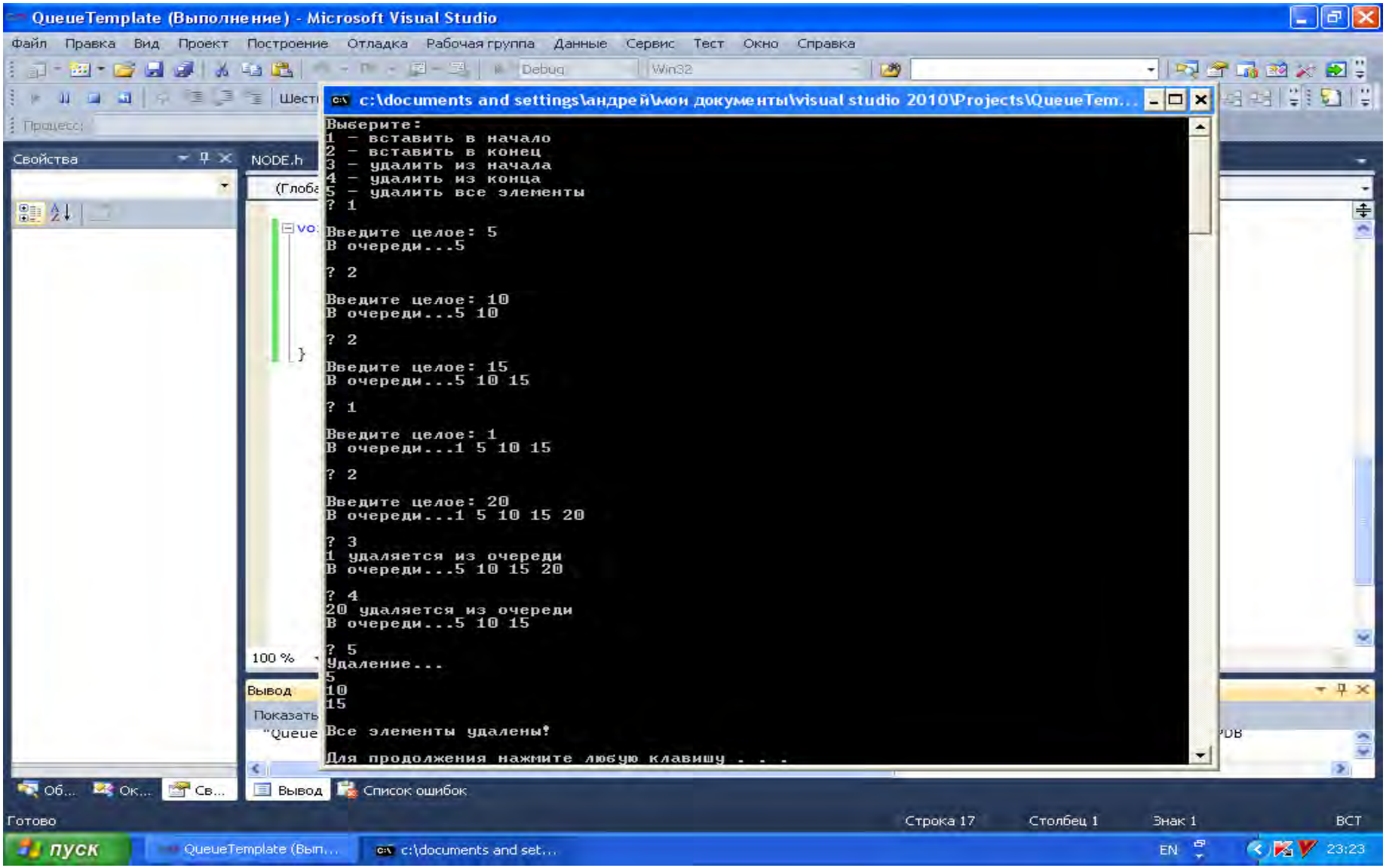


*Рисунок 2.2 – Результат работы программы*

Листинг 2.3. Построение бинарного дерева

```
// bintree.cpp.
#include "stdafx.h"
#include<iostream>
using namespace std;
struct node
{
int Key;
int Count;
node  *Left;
node *Right;
};
struct no
{node *elem;
int ch;
no *sled;};
class TREE
{private:
node *Tree;
void PushStack (no **,node **,int *);
void PopStack (no**,node **,int *);
void VyvodStack (no**);
public:
TREE () { Tree = new(node); (*Tree).Right = NULL; }
node* GetTreeRight () {return (*Tree).Right;}
void  TreeSearch (int);
void  VyvodTree (node *);};
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "russian");
TREE A;
int el;
cout<<"Вводите значения вершин: "<<endl;
cin>>el;
while  (el!=0)
{ A.TreeSearch (el); cin>>el; }
A.VyvodTree (A.GetTreeRight());
```

```
system("pause");return 0;}
void TREE::TreeSearch (int el)
// Поиск узла со значением el в дереве с
// последующим (в случае неудачного поиска!) включением
// в дерево. Tree - указатель на корень дерева.
{node *p1,*p2;
int  d;
p2 = Tree; p1 = (*p2).Right; d = 1;
while  (p1!=NULL && d!=0)
{ p2 = p1;
if  (el<(*p1).Key) {p1 = (*p1).Left; d = -1;}
else
if  (el>(*p1).Key) {p1 = (*p1).Right; d = 1;}
else  d = 0; }
if  (d==0)  (*p1).Count = (*p1).Count + 1;
else {
p1 = new(node);
(*p1).Key = el;  (*p1).Left = NULL;
(*p1).Right = NULL;  (*p1).Count = 1;
if  (d<0) (*p2).Left = p1; else  (*p2).Right = p1;}}
void TREE::VyvodTree (node *t)
//Построение дерева, заданного указателем t,
//на экране дисплея (нерекурсивный алгоритм).
{no *stk,*stk1;
node *u;
int i,n;
stk = stk1 = NULL; n = 0;
while  (t!=NULL)
{PushStack (&stk1,&t,&n);
if  ((*t).Right!=NULL)
{if  ((*t).Left!=NULL) PushStack (&stk,&((*t).Left),&n);
t = (*t).Right;}
else {if  ((*t).Left!=NULL)
{if  (stk1!=NULL)
{PopStack (&stk1,&u,&n);
for  (i=0; i<=n; i++) cout<<" ";
```

```
cout<<(*u).Key<<endl; } t = (*t).Left; } else
if  (stk==NULL) t = NULL;
else
{while  ((*stk).elem!=(*((*stk1).elem)).Left)
{PopStack (&stk1,&u,&n);
for  (i=0; i<=n; i++) cout<<" ";
cout<<(*u).Key<<endl;}
PopStack (&stk1,&u,&n);
for  (i=0; i<=n; i++) cout<<" ";
cout<<(*u).Key<<endl;
PopStack (&stk,&t,&n);}}
n = n + 1; }
VyvodStack (&stk1);}
void TREE::PushStack (no **stk,node **el,int *n)
// Помещение узлов со значениями *el и n в стек.
// *stk - указатель на стек.
{ no *q; q = new(no);  (*q).elem = *el;  (*q).ch = *n;
  (*q).sled = *stk; *stk = q;}
void TREE::PopStack (no**stk,node **t,int *n)
// Извлечение из стека узлов со значениями *t и n.
// *stk - указатель на стек.
{ no *q;  if  (*stk!=NULL)  {
    *t = (**stk).elem;    *n = (**stk).ch;    q  = *stk;    *stk = (**stk).sled;
delete q;  }}
void TREE::VyvodStack (no** stk)
// Вывод содержимого стека на экран дисплея.
// *stk - указатель на стек.
{node *k ;int i,n; while  (*stk!=NULL){ k = (**stk).elem; n = (**stk).ch;
for  (i=0; i<=n; i++) cout<<" ";
cout<<(*k).Key<<endl;*stk = (**stk).sled; }}
```

Скриншот работы программы приведен на рисунке 2.3

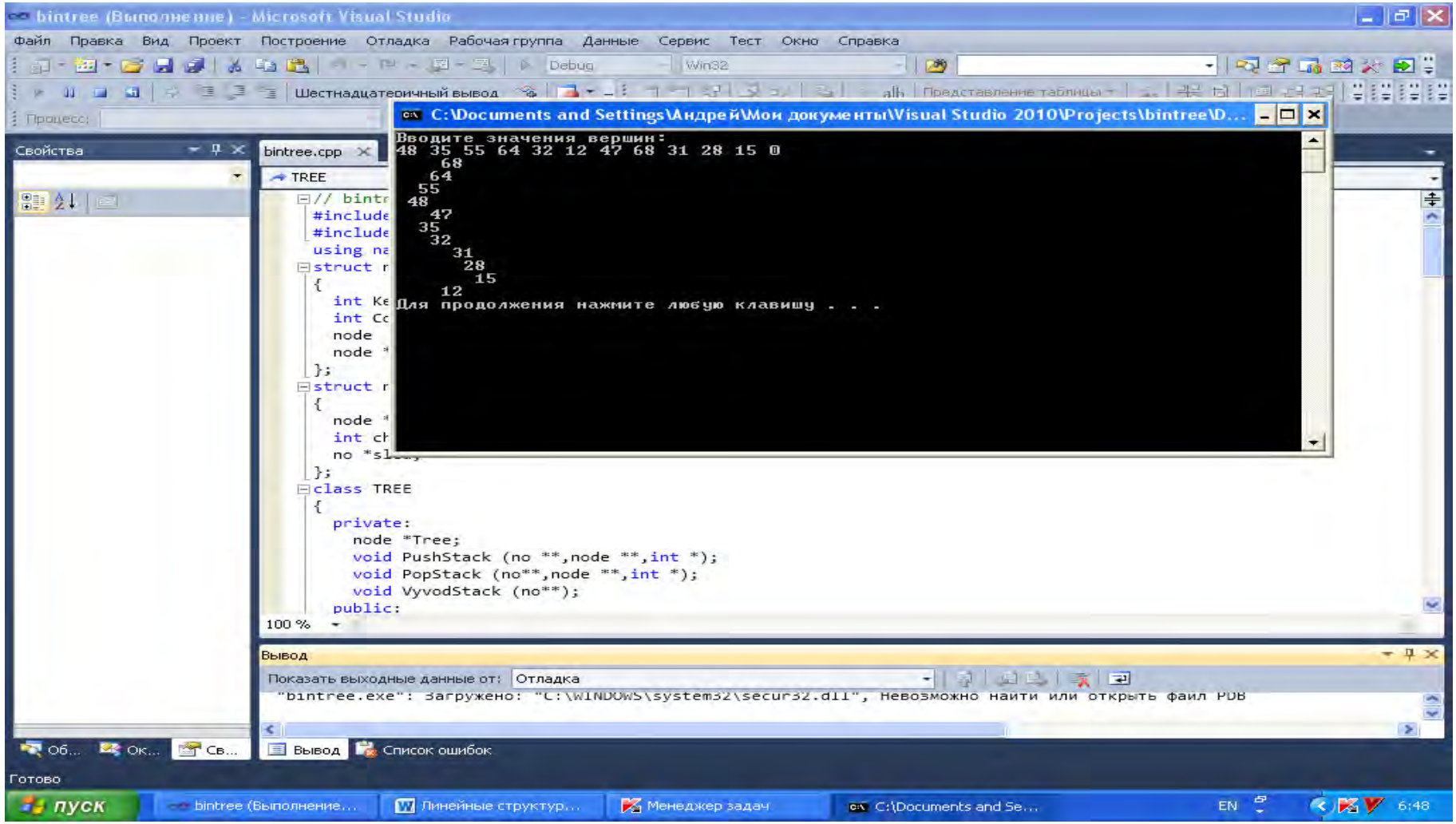


*Рисунок 2.3 – Бинарное дерево поиска*

## *2.2.3 Контрольные вопросы*

1 Что такое динамическая структура данных?
2 Что такое линейные структуры данных?
3 Какие характеристики линейных структур данных?
4 Что такое бинарное дерево?
5 Какие операции выполняются над узлами бинарного дерева?

## *2.2.4 Задания для самостоятельной работы*

1 Формализуйте на ЭВМ в виде линейного списка обслуживание покупателей билетов в кассе авиакомпании, занимающейся продажей билетов на предстоящие рейсы.

2 Формализуйте на ЭВМ в виде очереди ограниченного размера обслуживание заявок с данными от абитуриентов, осуществляемое приемной комиссией университета.

3 Разработайте программу, которая содержит информацию о сотрудниках, работающих в фирме. Сведения о сотрудниках содержат табельный номер, фамилию, имя, отчество, образование, год поступления на работу, домашний адрес, оклад. Программа должна обеспечивать начальное формирование данных обо всех сотрудниках фирмы в виде двоичного дерева, добавление данных о сотрудниках, вновь принятых на работу, удаление данных о сотрудниках, уволенных с работы, по запросу выдавать сведения о сотрудниках в штате фирмы, упорядоченные по фамилии, имени, отчеству.

4 Разработайте программу, которая содержит информацию о моделях компьютеров, продаваемых в магазине вычислительной техники. Сведения о компьютере содержат марку компьютера, тип процессора, тактовую частоту процессора, объем памяти, объем жесткого диска, объем памяти видеокарты, цену компьютера, количество экземпляров, имеющихся в наличии. Программа должна обеспечивать начальное формирование данных обо всех компьютерах в магазине вычислительной техники в виде двоичного дерева, добавление данных о компьютерах, поступающих в магазин, удаление данных о проданных компьютерах, выдавать сведения о наличии компьютеров в магазине, упорядоченные по наименованию модели.

5 Разработайте программу, которая содержит информацию о реестре жилых помещений (купля/продажа) фирмы. Данные реестра жилья содержат район, адрес, количество комнат, общую площадь, жилую площадь, год постройки дома, стоимость. Программа должна обеспечивать хранение всех данных о жилых помещениях в виде двоичного дерева, добавление в реестр данных о жилых помещениях, удаление данных о проданных жилых помещениях из реестра фирмы, вывод данных о жилых помещениях по стоимости, вывод всех жилых помещений, занесенных в реестр.

## 2.3 Реализация методами объектно-ориентированного программирования сверхдлинной целочисленной арифметики

Диапазон чисел, используемый в задачах криптографии, доходит до нескольких сот и тысяч десятичных цифр. Такой диапазон чисел не соответствует базовым типам данных современных компьютеров. Число, которое состоит из несколько сот (тысяч) десятичных знаков, нельзя записать как единый объект в базовое устройство компьютера. Компьютерное представление таких чисел и операции над ними реализуются в виде программ.

Длинная арифметика – операции (сложение, умножение, вычитание, деление, возведение в степень и т.д.) над числами, разрядность которых превышает длину машинного слова вычислительной машины. Эти операции реализуются не аппаратно, а программно, используя базовые аппаратные средства работы с числами меньших порядков.

Представление целого числа $a$ в системе счисления с основанием $B$ имеет вид

$$a = an\text{-}1Bn\text{-}1 + an\text{-}2Bn\text{-}2 + \ldots + a1B + a0,$$

где $0 \le ai < B$.

Данное представление целого числа аналогично представлению полинома степени $n$ - 1:

$$a(x) = an\text{-}1xn\text{-}1 + an\text{-}2xn\text{-}2 + \ldots + a1x + a0$$

с коэффициентами $ai$, $0 \le ai < B$.

**Замечание.** Знак числа, как и место десятичной точки, можно запомнить в отдельной переменной и учитывать при выполнении операций.

Сверхдлинное целое число хранится в виде массива цифр. Цифры используются из системы счисления. Применяется десятичная система

счисления и её степени (десять тысяч, миллиард) либо двоичная система счисления.

Операции над числами длинной арифметики производятся с помощью «школьных» алгоритмов сложения, вычитания, умножения, деления столбиком или алгоритмов быстрого умножения: быстрое преобразование Фурье и алгоритм Карацубы.

Для поддержки отрицательных чисел вводится дополнительный флаг «отрицательности» числа или дополняющие коды.

Сформулируем основные алгоритмы для выполнения арифметических операций с большими целыми числами.

### *2.3.1 Сложение сверхдлинных целых чисел*

Рассмотрим два целых числа *a, b* по некоторому основанию *B*, т. е.
$a = an\text{-}1Bn\text{-}1 + an\text{-}2Bn\text{-}2 + \ldots + a1B + a0$,
$b = bn\text{-}1Bn\text{-}1 + bn\text{-}2Bn\text{-}2 + \ldots + b1B + b0$,
где $0 \le ai < B$, $0 \le bi < B$.
Для сложения чисел используется следующий алгоритм.
**Алгоритм 1.** Найти сумму $w = a + b$.
1 Определить $c = 0$, $i = 0$.
2 Для $i = 0, 1, \ldots, n\text{-}1$ выполнить шаги 3-5 алгоритма.
3 Вычислить $t = ai + bi + c$.
4 Положить $c = t \bmod B$.
5 $wi = c$.
6 Положить $wn = t$.
7 Результат $w = wnBn + wn\text{-}1Bn\text{-}1 + wn\text{-}2Bn\text{-}2 + \ldots + w1B + w0$.

### *2.3.2 Вычитание сверхдлинных целых чисел*

Для вычитания чисел используется следующий алгоритм.
**Алгоритм 2.** Найти разность $w = a - b$, где $0 < b \le a$.
1 Определить $c = 0$, $i = 0$.
2 Для $i = 0, 1, \ldots, n\text{-}1$ выполнить шаги 3-5 алгоритма.
3 Вычислить $t = ai - bi + c$.
4 Положить $c = t \bmod B$.
5 $wi = c$.

6 Результат $w = wn\text{-}1Bn\text{-}1 + wn+m\text{-}2Bn+m\text{-}2 + \ldots + w1B + w0$.

Заметим, что в процессе выполнения алгоритма переменная $c$ может принимать отрицательное значение, т. е. может выполняться неравенство $c < 0$.

### *2.3.3 Умножение сверхдлинных целых чисел*

Операции умножения и деления являются наиболее трудоемкими. В настоящее время существует много методов умножения больших чисел. Рассмотрим алгоритм, который моделирует школьный метод умножения в столбик.

**Алгоритм 3.** Найти произведение двух чисел $a$ ($n$ – разрядное число) и $b$ ($m$ – разрядное число): $w = a * b$.

1 Для $i = 0, 1, \ldots, n\text{-}1$.

2 Определить $wi = 0$.

3 Для $i = 0, 1, \ldots, m\text{-}1$ (выполнить шаги 4-9 алгоритма).

4 Определить $c = 0$.

5 Для $j = 0, 1, \ldots, n\text{-}1$ (выполнить шаги 6-8 алгоритма).

6 Вычислить $t = wi+j + ai \cdot bj + c$.

7 Положить $wi+j = t \bmod B$.

8 $c = t \bmod B$.

9 $wi+n = c$.

10 Результат $wn+m\text{-}1Bn+m\text{-}1 + wn+m\text{-}2Bn+m\text{-}2 + \ldots + w1B + w0$.

Одним из самых трудных алгоритмов умножения двух чисел является алгоритм, который базируется на быстром преобразовании Фурье. Недостаток этого алгоритма заключается в том, что выигрыш в скорости умножения чисел по сравнению, допустим, с изложенным алгоритмом достигается лишь тогда, когда длина сомножителей 8000-10000 двоичных разрядов.

Для реальных задач криптографии используют числа гораздо большей разрядности.

### *2.3.4 Деление сверхдлинных целых чисел*

Рассмотрим алгоритм деления «в столбик».

**Алгоритм 4.** Найти результат деления двух чисел $a$ ($n + m$ – разрядное число) и $b$ ($n$ – разрядное число). Начальные данные алгоритма: целые

числа $a = (am+n-1,\ldots, a0)B$, $b = (bn-1 ,\ldots, b0)B$, где $B$ – основание системы счисления.

Результат выполнения алгоритма: целые числа $q = (qm, qm-1 ,\ldots, q0)B$ (частное от деления двух чисел) и $r = (rn, rn-1 ,\ldots, r0)B$ (остаток от деления двух чисел), такие, что $a = qb + r$, $0 \le r < b$.

1 Определить такое целое число $d > 0$, что $d \cdot bn-1 \ge [B/2]$.

2 Положить $(rm+n , rm+n-1 ,\ldots, r0)B \leftarrow (0, am+n-1, am+n-2 ,\ldots, a0)B$.

3 Для $i = m + n, m + n - 1, ..., n$ выполнить следующие действия (шаги 4-7).

4 Определить $Q = \min([(Ri \cdot B + Ri-1)/Vn-1], B - 1)$, где $Ri = ri \cdot d$, $Vn-1 = bn-1 \cdot d$, а выражение в квадратных скобках означает целую часть числа.

5 Пока $Vn-2 \cdot Q > (RiB + Ri-1 – QVn-1)B + Ri-2$ выполнять $Q = Q – 1$.

6 Вычислить $w = r – bQ$ и, если $w < 0$, положить $Q = Q – 1$.

7 Положить $w = r – bQ$ и $qi-n = Q$.

8 Результат $q = (qm , qm-1 ,\ldots, q0)B$ , $r = (rn , rn-1 ,\ldots, r0)B$.

## *2.3.5 Контрольные вопросы*

1 Что такое длинная арифметика?

2 Какой алгоритм операции сложения сверхдлинных целых чисел?

3 Какой алгоритм операции вычитания сверхдлинных целых чисел?

4 Какой алгоритм операции умножения сверхдлинных целых чисел?

5 Какой алгоритм операции деления сверхдлинных целых чисел?

## *2.3.6 Задания для самостоятельной работы*

Разработайте программу на языке C++, используя визуальное программирование. Программа формализует работу калькулятора сверхдлинных чисел. Калькулятор позволяет выполнять следующие операции.

1 Аддитивные и мультипликативные операции:

- сложения +;
- вычитания -;
- умножения *;
- деления /.

2 Операции сравнения сверхдлинных чисел:

- равно ==;

- меньше <;
- больше >;
- меньше или равно <=;
- больше или равно >=.

3 Логические и побитовые операции:

- побитовое И (&);
- побитовое исключающее ИЛИ (^);
- побитовое ИЛИ (|);
- логическое ИЛИ (||);
- логическое И (&&).

4 Определение наибольшего общего делителя (НОД) и наименьшего общего кратного (НОК).

## 2.4 Элементы библиотеки стандартных шаблонов

Стандартная библиотека шаблонов (STL – Standard Template Library) – библиотека контейнерных классов, включающая векторы, списки, очереди, стеки и алгоритмы общего назначения.

Цель включения библиотеки STL в стандарт языка состоит в избавлении программиста от разработки рутинных общепринятых программ.

Стандартная библиотека шаблонов разработана сотрудниками фирмы Hewlet Packard А.А. Степановым и М. Ли. Она содержит более сотни различных шаблонов и алгоритмов [4].

Стандартную библиотеку C++ можно условно разделить на две части. Первая часть включает функции, макросы, типы и константы, унаследованные из библиотеки C. Вторая часть содержит классы, шаблоны, средства для ввода, вывода, хранения и обработки данных как стандартных типов, так и типов, определенных пользователем [4; 7].

### *2.4.1 Последовательные контейнеры*

Последовательный контейнер – вид контейнеров, в котором введено отношение порядка для совокупности хранимых объектов. Для элементов контейнера определены понятия первый, последний, предыдущий и следующий.

Последовательные контейнеры обеспечивают хранение конечного количества однотипных объектов в виде последовательности и имеют разновидности: векторы (vector), деки (deque), списки (list), стеки (stack), очереди (queue), очереди с приоритетом (priority_queue). Первые три последовательных контейнера являются основными, остальные – вспомогательными контейнерами.

### *2.4.2 Ассоциативные контейнеры*

Ассоциативные контейнеры обеспечивают быстрый доступ к данным. Они построены на основе сбалансированных деревьев поиска. Существуют следующие типы ассоциативных контейнеров:

- словари (map);
- словари с дубликатами (multimap);
- множества (set);
- множества с дубликатами (multiset);
- битовые множества (bitset).

Словарь построен на основе пар «ключ/значение». Ключ ассоциирован с элементом. В качестве ключа используется значение произвольного типа. Ключ имеет уникальное значение.

Словарь с дубликатами позволяет дублирование ключей.

Множество предусматривает сортировку элементов в соответствии с их значениями. Дубликаты элементов не разрешаются.

Множество с дубликатами предусматривает наличие элементов с одинаковыми значениями.

Битовые множества представляют массивы битов.

Ассоциативные контейнеры описаны в заголовочных файлах <map> и <set>. Классы шаблонов ассоциативных контейнеров имеют необязательный аргумент для передачи критерия сортировки.

Для хранения пары «ключ – элемент» используется шаблон pair, описанный в заголовочном файле <utility>.

### *2.4.3 Алгоритмы стандартной библиотеки шаблонов*

Стандартная библиотека шаблонов предоставляет алгоритмы для выполнения операций, которые требуются пользователю контейнера. Об-

щее число алгоритмов около шестидесяти. Для доступа к алгоритмам библиотеки стандартных шаблонов в программу необходимо включить заголовочный файл <algorithm>.

Алгоритмы стандартной библиотеки шаблонов подразделяются на пять категорий:

- немодифицированные операции с последовательностями;
- модифицированные операции с последовательностями;
- алгоритмы сортировки последовательностей;
- алгоритмы работы с множествами и пирамидами;
- обобщенные численные алгоритмы.

Обобщенные численные алгоритмы выполняют обработку числовых элементов. Они помещены в заголовочный файл <numeric>. В таблице 4.1 приведен список численных алгоритмов стандартной библиотеки шаблонов.

*Таблица 4.1 – Обобщенные численные алгоритмы*

| Название | Описание |
|---|---|
| accumulate() | Объединяет все значения элементов (вычисляет сумму, произведение и т.д.) |
| inner_product() | Объединяет все элементы двух интервалов |
| adjacent_difference() | Объединяет каждый элемент с его предшественником |
| partial_sum() | Объединяет каждый элемент со всеми предшественниками |

## *2.4.4 Контрольные вопросы*

1 Что такое последовательный контейнер?
2 Что такое ассоциативный контейнер?
3 Какие виды последовательного контейнера?
4 Какие виды ассоциативного контейнера?
5 Что такое обобщённые алгоритмы?

## *2.4.5 Задания для самостоятельной работы*

1 Последовательный контейнер вектор. Присваивание векторов

Разработайте законченную программу, в которой с помощью подходящих конструкторов создайте три вектора *v1*, *v2*, *v3* с элементами целого типа, размерами соответственно 5, 7, 6 и одинаковыми значениями элементов соответственно 1, 2, 3. Выведите на экран размеры векторов, значения их элементов и выполните присваивание *v3=v2-v1*. После этого вновь выведите на экран размеры векторов и значения их элементов.

Используйте только средства стандартной библиотеки языка C++ (потоковый ввод-вывод, класс *vector*).

2 Последовательный контейнер вектор. Копирование векторов

Разработайте законченную программу, в которой с помощью подходящих конструкторов создайте три вектора *v1*, *v2*, *v3* с элементами целого типа, размерами соответственно 4, 5, 7 и одинаковыми значениями элементов соответственно 1, 2, 3. Выведите на экран размеры векторов, значения их элементов. С помощью метода *assign( )* первым трем элементам *v1* присвойте значение 4, а первым двум элементам *v2* присвойте значения элементов *v3[4]* и *v3[5]*. После это этого вновь выведите на экран размеры векторов и значения их элементов.

Используйте только средства стандартной библиотеки языка C++ (потоковый ввод-вывод, класс *vector*).

3 Ассоциативные контейнеры. Множества с дубликатами (*multiset)*

Напишите законченную программу, в которой с помощью подходящего конструктора создайте множество с дубликатами и инициализируйте его четырьмя строковыми значениями. С помощью итераторов выведите содержимое созданного множества на экран. Добавьте в множество еще два строковых значения, одно из которых уже имеется в множестве. Еще раз выведите содержимое множества на экран.

Используйте только средства стандартной библиотеки языка C++ (потоковый ввод-вывод, классы *string, set*).

4 Ассоциативные контейнеры. Словари (*map)*

Напишите законченную программу, в которой создайте пустой словарь – телефонную книгу для хранения записей в лексикографическом порядке. Заполните телефонную книгу данными из файла *map.dat*. Каждая строка файла хранит фамилию абонента и телефонный номер-число, раз-

деленные пробелом. Выведите на экран содержимое телефонной книги. Дополните телефонную книгу новой записью и измените один из номеров телефонной книги. Снова выведите на экран содержимое телефонной книги. Выполните поиск в словаре существующего и несуществующего элемента и выведите результат поиска на экран.

Используйте только средства стандартной библиотеки языка C++ (потоковый ввод-вывод, классы *string, map*).

5 Напишите законченную программу, в которой создайте и инициализируйте целочисленный вектор. Выведите значения его элементов на экран. С помощью метода *find( )* выполните поиск в векторе элемента с заданным значением и выведите результаты поиска на экран. С помощью метода count( ) подсчитайте в векторе количество элементов с заданным значением и выведите результаты поиска на экран.

Используйте только средства стандартной библиотеки языка C++ (потоковый ввод-вывод, класс *vector,* объявления стандартных функциональных объектов из файла <*functional*> и объявления стандартных алгоритмов из файла <*algorithm*>).

# ЗАКЛЮЧЕНИЕ

При создании программ используют различные технологии программирования: структурная, объектно-ориентированная, обобщенная с использованием стандартной библиотеки.

Технология визуального проектирования и событийного программирования позволяет создавать эффективные приложения, с которыми связаны два аспекта: программный код для создания графического интерфейса пользователя, с которым взаимодействует пользователь, и программный код для обработки этого взаимодействия и реализации полезной функциональности приложения.

# СПИСОК ЛИТЕРАТУРЫ

1 Павловская Т. А. C/C++. Программирование на языке высокого уровня. – СПб. : Питер, 2006. – 461 с.

2 Глушаков С. В., Дуравкина Т. В. Программирование на C++. – М. : АСТ, 2008. – 685 с.

3 Лаптев В. В. C++. Объектно-ориентированное программирование : учебное пособие. – СПб. : Питер, 2008. – 464 с.

4. Культин Н. Б. Основы программирования в Microsoft Visual C++. 2010. – СПб. : БХВ-Петербург, 2010. – 384 с.

5. Культин Н. Б. Microsoft Visual C++ в задачах и примерах. – СПб. : БХВ-Петербург, 2010. – 272 с.

6. Пахомов Б. И. C/C++ и MS Visual C++ 2012 для начинающих. – СПб. : БХВ-Петербург, 2013 – 512 с.

7. Давыдов В. Г. Технологии программирования. C++. – СПб. : БХВ-Петербург, 2005. – 672 с.

Учебное издание

Семахин Андрей Михайлович

**ОСНОВЫ ПРОГРАММИРОВАНИЯ.**
**ЛАБОРАТОРНЫЙ ПРАКТИКУМ**

Учебное пособие

Редактор Г.В. Меньщикова


| | | |
|---|---|---|
| Подписано в печать 24.10.16 | Формат 60x84 1/16 | Бумага 65 г/м$^2$ |
| Печать цифровая | Усл. печ. л. 4,06 | Уч.-изд. л. 4,06 |
| Заказ № 162 | Тираж 100 | |

БИК Курганского государственного университета.
640020, г. Курган, ул. Советская, 63/4.
Курганский государственный университет.