Министерство образования и науки, молодежи и спорта Украины
Севастопольский национальный технический университет

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

**к лабораторному практикуму
по дисциплине
«Вычислительная техника и программирование»**

для студентов дневной и заочной форм обучения
направления 6.050901 — «Радиотехника»

Часть 3

Севастополь
2013

УДК 004.43+621.37 (076.5)

Методические указания к лабораторному практикуму по дисциплине «Вычислительная техника и программирование» для студентов дневной и заочной форм обучения направления 6.050901 — «Радиотехника» : в 4 ч. / СевНТУ ; сост. С.Н. Бердышев. — Севастополь : Изд-во СевНТУ, 2013. — Ч.3. — 48 с.

Целью методических указаний является оказание помощи студентам дневной и заочной форм обучения направления 6.050901 — «Радиотехника» при выполнении лабораторных работ по дисциплине «Вычислительная техника и программирование».

Рассмотрены и утверждены на заседании кафедры радиотехники и телекоммуникаций СевНТУ (протокол № 5 от 23 января 2013 г.).

Допущено учебно-методическим центром СевНТУ в качестве методических указаний.

Рецензент:
Савочкин А.А., канд. техн. наук, доцент кафедры радиотехники и телекоммуникаций.

Ответственный за выпуск:
Гимпилевич Ю.Б., доктор техн. наук, проф., зав. кафедрой радиотехники и телекоммуникаций.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Дисциплина «Вычислительная техника и программирование» изучается студентами дневной формы обучения направления 6.050901 «Радиотехника» в первом и втором семестрах, студентами заочной формы обучения — в третьем или шестом семестрах.

При изучении первой части дисциплины студенты выполняют и защищают цикл из шести лабораторных работ, которые направлены на получение практических навыков программирования на языке *C*/*C*++ и изучение таких базовых структур как алгоритмы линейной, разветвляющейся и циклической структур, алгоритмы обработки одномерных и двумерных массивов. Итоговой формой контроля в первом семестре является зачет.

Вторая часть дисциплины предусматривает выполнение и защиту шести лабораторных работ, направленных на углубление и закрепление знаний, полученных при изучении первой части. При выполнении второй части лабораторного практикума рассматриваются вопросы программной реализации алгоритмов работы со структурами данных, строками и файлами, а также использование графического режима. В данных методических указаниях представлены лабораторные работы №1 — №3 второй части дисциплины. Итоговой формой контроля во втором семестре является экзамен.

После изучения дисциплины «Вычислительная техника и программирование» студент должен знать: основные понятия вычислительной техники и программирования, архитектурные особенности различных типов ЭВМ, способы разработки алгоритмов при решении технических задач, порядок работы на персональном компьютере с одной из существующих операционных систем, методику работы с программными пакетами, необходимыми в профессиональной деятельности.

В результате изучения данной дисциплины студент должен уметь: выбирать методы необходимые для решения поставленной задачи, составлять алгоритмы, разрабатывать и тестировать программы на языке *C*/*C*++, оформлять техническую документацию на программу в соответствии с действующими стандартами, читать специальную техническую литературу, осваивать новые языки программирования и типы программных комплексов.

Выполнение лабораторных работ осуществляется фронтальным методом каждым студентом индивидуально в соответствии с вариантом задания, который выдается преподавателем.

На выполнение лабораторных работ №1 — №3 отводится по четыре часа, а на выполнение работ №4 — №6 отводится по шесть часов. Выполняются лабораторные работы в два этапа. На первом этапе студенты самостоятельно должны:

— проработать по конспекту лекций и рекомендованной литературе, приведенной в библиографическом списке, основные теоретические положения лабораторной работы и подготовить ответы на контрольные вопросы;

— разработать алгоритм решения задания, составить его структурную схему, и описать его;

— написать программу на языке *C*/*C*++ и описать её;

— оформить результаты выполнения первого этапа в виде черновика отчета по лабораторной работе.

На втором этапе, выполняемом в лаборатории университета, студенты должны:

— представить результаты выполнения первого этапа преподавателю;

— отладить и выполнить написанную программу;

— распечатать полученные результаты;

— оформить окончательный вариант отчета по лабораторной работе;

— защитить отчет по лабораторной работе.

Выполнять и защищать лабораторные работы студенты должны строго по графику, в соответствии с расписанием учебных занятий.

В отчет должны включаться: титульный лист, цель работы, текст задания, теоретические сведения и расчеты, схема алгоритма с описанием, текст программы с комментариями и результаты выполнения программы. Содержание отчета перечислено в методических указаниях к каждой лабораторной работе.

На титульном листе обязательно требуется указать фамилию, имя и отчество полностью, группу, номер варианта, название и номер работы. Образец оформления титульного листа приведен в методических указаниях [1].

Отчет по лабораторной работе оформляется каждым студентом индивидуально на стандартных листах формата А4 с использованием персонального компьютера (ПК) и печатается с помощью принтера. Тексты разработанных программ и результаты программных расчетов следует приводить только в распечатанном виде. По краям листа должны быть оставлены поля: левое — 25 мм; верхнее и нижнее — 20 мм; правое — 15 мм.

Отчет следует оформлять в соответствии с методическими указаниями по оформлению текстовых работ [2].

Схемы и графики следует печатать на стандартных листах белой бумаги. Рисунки и таблицы должны быть пронумерованы и сопровождаться подписями и пояснениями. Схемы алгоритмов необходимо выполнять в соответствии с требованиями единой системы программной документации (ЕСПД) и ГОСТ 19.701-90 [3]. Правила оформления схем алгоритмов изложены в методических указаниях [1].

При выполнении лабораторных работ и подготовке к защите рекомендуется использовать учебную и справочную литературу [1, 4 — 15].

1. ЛАБОРАТОРНАЯ РАБОТА №1
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ РЕКУРСИВНЫХ ФУНКЦИЙ

1.1. Цель работы

Изучение особенностей составления и использования рекурсивных функций в языках программирования *C*/*C*++.

Получение практических навыков реализации алгоритмов, использующих рекурсивные функции, средствами языков программирования *C*/*C*++.

1.2. Варианты заданий

Составить схему алгоритма и написать на языке *C*/*C*++ программу вычисления суммы n элементов ряда, полученного при разложении функции $f(x)$, используя рекурсивные функции. Варианты задания выбираются по указанию преподавателя из таблицы 1.1. Значения аргумента x выбираются произвольно. Результаты вычисления суммы элементов ряда вывести на экран в формате с плавающей точкой и сравнить со значением функции $f(x)$.

Таблица 1.1 — Исходные данные к лабораторной работе № 1

Вариант	Функция $f(x)$	Представление в виде ряда	n
1	$\sin x$	$\sum_{k=0}^{\infty}(-1)^k \frac{x^{2k+1}}{(2k+1)!}$, при $0 \le x < \pi$	5
2	$\operatorname{ch} x$	$\sum_{k=0}^{\infty}\frac{x^{2k}}{(2k)!}$	6
3	e^{-x^2}	$\sum_{k=1}^{\infty}(-1)^k \frac{x^{2k}}{k!}$, при $x^2 < 1$	7
4	$(1+x)^{-2}$	$\sum_{k=1}^{\infty}(-1)^{k-1} k x^{k-1}$, при $x^2 < 1$	8
5	$e^x(1+x)$	$\sum_{k=0}^{\infty}\frac{x^k(k+1)}{k!}$	9
6	$\frac{x}{(1-x)^2}$	$\sum_{k=1}^{\infty} k x^k$, при $x^2 < 1$	10
7	$\operatorname{th} x$	$1 + 2\sum_{k=1}^{\infty}(-1)^k e^{-2kx}$, при $0 < x$	5
8	$\frac{x}{1-x}$	$\sum_{k=1}^{\infty}\frac{x^{2^{k-1}}}{1-x^{2^k}}$, при $x^2 < 1$	6

Продолжение таблицы 1.1

Вариант	Функция $f(x)$	Представление в виде ряда	n
9	$\sin^2 x$	$\sum_{k=1}^{\infty}(-1)^{k+1}\frac{2^{2k-1}x^{2k}}{(2k)!}$, при $-\pi<x<\pi$	7
10	$\frac{1}{x-1}$	$\sum_{k=1}^{\infty}\frac{2^{k-1}}{x^{2^{k-1}}+1}$, при $x^2>1$	8
11	$\cos^3 x$	$\frac{1}{4}\sum_{k=0}^{\infty}(-1)^k\frac{(3^{2k}+3)x^{2k}}{(2k)!}$, при $-\frac{\pi}{2}<x<\frac{\pi}{2}$	9
12	$\cos x$	$\sum_{k=0}^{\infty}(-1)^k\frac{x^{2k}}{(2k)!}$, при $-2\pi<x<2\pi$	10
13	$\operatorname{sh} x$	$\sum_{k=0}^{\infty}\frac{x^{2k+1}}{(2k+1)!}$	5
14	$\ln x$	$\sum_{k=1}^{\infty}(-1)^{k+1}\frac{(x-1)^k}{k}$, при $\lvert x-1\rvert<1$	6
15	$\ln(1+x)$	$\sum_{k=1}^{\infty}(-1)^{k+1}\frac{x^k}{k}$, при $-1<x\le 1$	7
16	$\ln\frac{1+x}{1-x}$	$2\sum_{k=1}^{\infty}\frac{x^{2k-1}}{2k-1}$, при $x^2<1$	8
17	$\ln\frac{x+1}{x-1}$	$2\sum_{k=1}^{\infty}\frac{1}{(2k-1)x^{2k-1}}$, при $x^2>1$	9
18	$\frac{1-x}{x}\ln\frac{1}{1-x}$	$1-\sum_{k=1}^{\infty}\frac{x^k}{k(k+1)}$, при $x^2<1$	10
19	$\arcsin x$	$\sum_{k=0}^{\infty}\frac{x^{2k+1}(2k)!}{2^{2k}(k!)^2(2k+1)}$, при $x^2<1$	5
20	$\operatorname{arctg} x$	$\sum_{k=0}^{\infty}\frac{(-1)^k x^{2k+1}}{2k+1}$, при $x^2\le 1$	6
21	$\ln\left(1+\sqrt{1+x^2}\right)$	$\ln 2-\sum_{k=1}^{\infty}(-1)^k\frac{(2k-1)!}{2^{2k}(k!)^2}x^{2k}$, при $x^2\le 1$	7
22	$\frac{(1-x)^2}{2x^3}\ln\frac{1}{1-x}$	$\frac{1}{2x^2}-\frac{3}{4x}+\sum_{k=1}^{\infty}\frac{x^{k-1}}{k(k+1)(k+2)}$, при $x^2<1$	8
23	$\operatorname{arctg} x$	$\frac{\pi}{2}-\sum_{k=0}^{\infty}(-1)^k\frac{1}{(2k+1)x^{2k+1}}$, при $x^2\ge 1$	9

Продолжение таблицы 1.1

Вариант	Функция $f(x)$	Представление в виде ряда	n
24	$\ln\frac{x}{x-1}$	$\sum_{k=1}^{\infty}\frac{1}{kx^k}$, при $x^2>1$	10
25	$\ln\frac{1}{1-x}$	$\sum_{k=1}^{\infty}\frac{x^k}{k}$, при $x^2<1$	5
26	$\mathrm{tg}\frac{\pi x}{2}$	$\frac{4x}{\pi}\sum_{k=1}^{\infty}\frac{1}{(2k-1)^2-x^2}$, при $x^2<1$	6
27	$\sin^3 x$	$\frac{1}{4}\sum_{k=1}^{\infty}(-1)^{k+1}\frac{(3^{2k+1}-3)x^{2k+1}}{(2k+1)!}$, при $-\frac{\pi}{2}<x<\frac{\pi}{2}$	7
28	$\cos^2 x$	$1-\sum_{k=1}^{\infty}(-1)^{k+1}\frac{2^{2k-1}x^{2k}}{(2k)!}$, при $-\pi<x<\pi$	8
29	$(1+x)^{-1}$	$\sum_{k=1}^{\infty}(-1)^{k-1}x^{k-1}$, при $x^2<1$	9
30	e^x	$\sum_{k=0}^{\infty}\frac{x^k}{k!}$	10

1.3. Теоретические сведения

1.3.1. Рекурсивные функции

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

Когда функция вызывает себя, в аппаратном или программном стеке создается копия значений её параметров, как и при вызове обычной функции, после чего управление передается первому исполняемому оператору функции. При повторном вызове этот процесс повторяется. Для завершения вычислений каждая рекурсивная функция должна содержать хотя бы одну нерекурсивную ветвь алгоритма, заканчивающуюся оператором возврата. При завершении функции соответствующая часть стека освобождается, и управление передается вызывающей функции, выполнение которой продолжается с оператора, следующего за рекурсивным вызовом. Стеком называют специальную область памяти или регистры, из которых сохраненные данные считываются в обратном порядке: первыми считываются последние записанные данные, а данные, записанные первыми, считываются последними.

Объявление, описание и вызов рекурсивных функций выполняется по тем же правилам, что обычных функций (см. лабораторную работу №5 [4]).

Примером рекурсивной функции является вычисление факториала. Для того чтобы получить значение факториала числа n, требуется умножить на n факториал числа $(n-1)$:

```
long factorial (long n)
{
    if (n==0 || n==1) return 1;     // 0! = 1, 1! = 1,
                                    // нерекурсивная ветвь функции
    return (n * factorial(n - 1)); // рекурсивный вызов функции
}
```

В функции выполняется проверка значения аргумента и если аргумент равен **1** или **0**, то функция возвращает значение **1**:

```
if (n==0 || n==1) return 1;
```

Данная ветвь функции не является рекурсивной и предназначена для выхода из рекурсии. Рекурсивный вызов функции выполняется до тех пор, пока аргумент не равен **0** или **1**.

Рекурсивный вызов функции вычисления факториала записан в строке

```
return (n * factorial(n - 1));
```

Здесь выполняется умножение числа n на факториал числа $(n-1)$, т.е. вызывается функция вычисления факториала от декрементированного на единицу аргумента и полученное значение факториала умножается на значение аргумента.

Рекурсивную функцию чаще всего применяют для компактной реализации рекурсивных алгоритмов, а также для работы со структурами данных, описанных рекурсивно. Любую рекурсивную функцию можно реализовать без применения рекурсии, для чего необходимо в программе обеспечить хранение всех необходимых данных самостоятельно.

Достоинством использования рекурсии является компактность программного кода, а недостатком — повышенный расход ресурсов вычислительной системы и опасность переполнения стека.

1.3.2. Пример вычисления суммы ряда

Рассмотрим пример использования рекурсивной функции для вычисления суммы заданного количества членов ряда

$$\sum_{k=1}^{\infty}\left(1+(-1)^k\frac{\sqrt{1+k^2x}}{(k-1)!}\right).$$

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
```

```
// объявление прототипов функций
// функция вычисления суммы ряда
double sum_of_series(double x, int k, int n);
// функция вычисления члена ряда
double term_of_series(double x, int k);
// функция вычисления факториала
long int factorial (long int k);

void main()
{
    int    n;   // количество суммируемых членов ряда
    double x,   // значение аргумента
           sum; // сумма членов ряда

    clrscr();

    cout << "Введите значение аргумента 'x' : "; cin >> x;
    cout << "Введите количество членов ряда 'n' : "; cin >> n;

    // вычисление суммы ряда
    sum = sum_of_series(x, 1, n);
    cout << "Сумма членов ряда : " << sum << endl;

    getch();

    return;
}

// рекурсивная функция вычисления суммы ряда
double sum_of_series(double x, int k, int n)
{
    if (k <= n)
    {   // вычисление суммы
        return (  term_of_series(x, k       )
                + sum_of_series(x, k + 1, n)  );
    }
    else
    {   // прекращаем вычисления суммы, если количество членов
        // ряда превысило заданное количество
        return 0;
    }
}

// функция вычисления члена ряда
double term_of_series(double x, int k)
{
    double result = 0;

    result = 1 + ((k % 2) ? -1 : 1)
               * sqrt(1 + k * k * x)
               / factorial(k - 1);
```

```
    cout << " Член ряда : " << k
         << " Значение  : " << result << endl;

    return result;
}

// рекурсивная функция вычисления факториала
long int factorial (long int k)
{
    return ((k == 0 || k == 1) ? 1 : k * factorial(k - 1));
}
```

Здесь рекурсивная функция

```
double sum_of_series(double x, int k, int n);
```

используется для вычисления суммы ряда и имеет две ветви. Для того чтобы получить значение суммы членов ряда, требуется сложить член ряда с порядковым номером k с суммой членов от элемента с порядковым номером $(k+1)$ до последнего элемента с номером n:

```
term_of_series(x, k) + sum_of_series(x, k + 1, n);
```

Суммирование продолжается до тех пор, пока порядковый номер члена ряда не превысит заданное количество членов ряда.

Вторая рекурсивная функция предназначена для вычиления значения факториала:

```
long int factorial (long int k);
```

В функции предусмотрен выход из рекурсии при вычислении значений факториалов 0! или 1!.

1.4. Порядок выполнения работы

Вариант задания определяется преподавателем. Перед выполнением работы необходимо ознакомится с разделами 1.1 — 1.3 данных методических указаний. Индивидуальные задания для выполнения работы см. в разделе 1.2.

1.4.1. Внимательно изучить индивидуальное задание.

1.4.2. Разработать схему алгоритма решения задачи.

1.4.3. Написать программу вычисления суммы ряда используя рекурсивные функции. Предусмотреть ввод аргумента x и количества членов ряда n с клавиатуры.

1.4.4. Выполнить тестирование и отладку написанной программы.

1.4.5. Получить результаты работы программы для произвольного значения аргумента и заданного количества членов ряда.

Рекомендуется скопировать в виде изображения сообщения, выводимые программой, для последующего включения в отчет.

1.4.6. Вычислить значение функции $f(x)$ для выбранного значения аргумента. Для вычислений рекомендуется использовать инженерно-

математическую программу *MathCAD* [15].

1.4.7. Подготовить отчет по работе.

1.5. Оформление отчета

1.5.1. Сформулировать цель работы.

1.5.2. Привести постановку задачи и текст индивидуального задания.

1.5.3. Привести краткие теоретические сведения о рекурсивных функциях.

1.5.4. Изобразить схему алгоритма решения задачи. Записать краткие пояснения к алгоритму.

1.5.5. Привести текст программы. Программа обязательно должна содержать комментарии.

1.5.6. Составить таблицу, в которой указать имена всех переменных используемых в программе и их тип, а также кратко описать назначение переменных.

1.5.7. Привести результаты выполнения программы.

1.5.8. Привести результат вычисления функции $f(x)$, полученный в инженерно-математической программе *MathCAD*, и сравнить его с результатом выполнения программы.

1.5.9. Сделать выводы по работе.

1.6. Контрольные вопросы

1.6.1. Какую функцию называют рекурсивной?

1.6.2. Что такое прямая рекурсия?

1.6.3. Что такое косвенная рекурсия?

1.6.4. Каким образом в языках *C*/*C*++ объявляются рекурсивные функции?

1.6.5. Каким образом в языках *C*/*C*++ описываются рекурсивные функции?

1.6.6. Каким образом в языках *C*/*C*++ вызываются рекурсивные функции?

1.6.7. Какие требования предъявляются к рекурсивным функциям?

1.6.8. Для чего используют рекурсивные функции?

1.6.9. Перечислите достоинства и недостатки использования рекурсивных функций.

1.6.10. На примере поясните алгоритм работы рекурсивной функции.

2. ЛАБОРАТОРНАЯ РАБОТА №2

ПОСТРОЕНИЕ ГРАФИКОВ ФУНКЦИЙ СРЕДСТВАМИ ЯЗЫКА *C*

2.1. Цель работы

Изучение основных возможностей графической библиотеки языка *Borland C*.

Приобретение практических навыков по разработке и отладке программ, использующих графические режимы дисплеев.

2.2. Варианты заданий

Вычислить и вывести на экран в виде графика функции $y = f(x)$. График снабдить координатными осями. На координатные оси нанести разметку. Варианты функций выбирать по указанию преподавателя из таблицы 2.1. Пример вывода графика показан на рисунке 2.1 в разделе 2.3.8.

Таблица 2.1 — Исходные данные к лабораторной работе № 2

Вариант	Функция y	Вариант	Функция y
1	$y = \lvert\sin(x+4)\rvert$	14	$y = \operatorname{tg}(0{,}05x-1)/x$
2	$y = 6x^2 + 3x$	15	$y = \cos\lvert x-3\rvert$
3	$y = x^3 + 2x^2 + x$	16	$y = 1/(x-1)$
4	$y = \sin\lvert x\rvert$	17	$y = (x+2)/(x-4)$
5	$y = 1/x$	18	$y = \sin\lvert 2-x\rvert$
6	$y = 1/(x^2 - 2x + 1)$	19	$y = \sin(x)/x$
7	$y = (x+3)/(x-2)$	20	$y = x^3 + x^2 - \sin x$
8	$y = \cos(x-1) + \lvert x\rvert$	21	$y = (x^2 + 3x - 2)/x$
9	$y = 1/(3x^2 + 2x + 1)$	22	$y = 2/(x^2 - 1{,}5)$
10	$y = x^5 - 3x^2$	23	$y = \cos\lvert x+1\rvert$
11	$y = (3-x)/(3+x)$	24	$y = \sin x \cos(0{,}1x)$
12	$y = 3 - 2/x - 2/x^2$	25	$y = x^3 - 5x^2 + 3x - 1$
13	$y = 1/(x^2 + 3x - 1)$	26	$y = x^2 + 2x - 1$

Продолжение таблицы 2.1

Вариант	Функция y	Вариант	Функция y
27	$y = 1/\left(x^2 - 3\right)$	29	$y = 1/\left(x + 0{,}75\right)^2$
28	$y = 1/\left(x^3 - 1\right)$	30	$y = \sin\lvert x\rvert - \cos\lvert 0{,}1x\rvert$

2.3. Теоретические сведения

2.3.1. Графический режим видеоадаптера

Видеоадаптер персонального компьютера может работать в одном из двух режимов — текстовом или графическом. В текстовом режиме на экране дисплея отображаются только символы. В графическом режиме минимальным элементом изображения является пиксель — графическая точка. Отметим, что *Windows*-программы работают только в графическом режиме, а использование текстового режима является характерным только для *DOS*-программ. Когда программу, которая работает в текстовом режиме, запускают из среды *Windows*, режим видеоадаптера может оставаться графическим, а текстовый режим будет эмулироваться только в пределах окна программы. Такой режим выполнения *DOS*-программ называется оконным. Полноэкранный режим выполнения *DOS*-программ предполагает установку необходимого видеорежима во время ее запуска и возврат к исходному режиму после завершения её работы. *DOS*-программа, которая работает с графикой, может быть выполнена только в полноэкранном режиме.

Программа, работающая в графическом режиме, использует графические драйверы — файлы, которые обеспечивают взаимодействие с видеоадаптером. Для разных типов видеоадаптеров используют различные графические драйверы. Основными характеристиками видеоадаптера являются разрешающая способность и количество цветов. Разрешающая способность задается парой чисел, первое из которых показывает количество пикселей в одной строке, а второе — число строк. Современные дисплейные адаптеры принадлежат к классу *SVGA* (*Super Video Graphics Array*). Такие адаптеры характеризуются разрешающей способностью более 640×480 пикселей и позволяют использовать не менее 256 цветов. Для работы с *SVGA* адаптерами используются драйверы *svga256.bgi* и *egavga.bgi*.

Графическая система языка *Borland C* состоит из обширной библиотеки *GRAPHICS.LIB*. Компилятор должен иметь доступ к этой библиотеке, иначе компиляция программы, содержащей вызовы функций графической библиотеки, будет заканчиваться выдачей множественных сообщений об ошибках. Для этого необходимо подключить графическую библиотеку:

1) В меню оболочки выбирать *Options* > *Linker* > *Libraries*…

2) На открывшейся вкладке *Libraries* в поле *Libraries* напротив строки

Graphic library поставить отметку выбора (курсором «мыши» или клавишей «пробел»).

3) На вкладке *Libraries* нажать кнопку-надпись «*Ok*».

При создании программы, обращающейся к функциям графической библиотеки, также необходимо подключить заголовочный файл **<graphics.h>**.

Графическая система содержит внутреннюю переменную, предназначенную для хранения кода завершения графических функций. В случае ошибки код завершения функции является отрицательным и определяется константой перечислимого типа **graphics_errors**. Текущее значение переменной возвращает функция **graphresult**.

Прежде чем обращаться в прикладной программе к функциям графической библиотеки, необходимо выбрать графический драйвер, соответствующий установленному на персональном компьютере видеоадаптеру. Для выбора драйвера можно воспользоваться функцией **detectgraph**, которая тестирует аппаратуру видеоадаптера и автоматически выбирает необходимый драйвер и графический режим:

```
void detectgraph( int far *graphdriver, int far *graphmode);
```

Эта функция через свои аргументы возвращает номер графического драйвера, обслуживающего обнаруженный ею дисплейный адаптер, и номер графического режима, обеспечивающего максимальное для данного адаптера разрешение. Возвращенные функцией **detect_graph** значения в дальнейшем могут быть переданы функции инициализации графической системы **initgraph**. Функция **initgraph** ищет на диске *bgi*-файл, содержащий требуемый драйвер, загружает и инициализирует соответствующий драйвер, переводит систему в графический режим и передает контроль вызванной программе. Прототип этой функции:

```
void initgraph (int far *driver, int far *mode, char far
*pathtodriver);
```

Аргументами функции являются указатели на переменные, содержащие номер графического драйвера, номер графического режима этого драйвера и путь к *bgi*-файлу.

Если по указателю ***driver** перед вызовом было записано значение **DETECT** (или **0**), то сначала запускается процедура автоматического тестирования аппаратуры видеоадаптера. В этом случае нет необходимости вызывать функцию **detectgraph**. Всю работу, связанную с переходом в графический режим, выполнит функция **initgraph**.

Ниже приведен фрагмент кода, позволяющий перевести систему в графический режим:

```
# include <graphics.h>  // подключение библиотеки графических
                        // средств
main()
{
    int Driver,Mode;
    Driver=DETECT;
    initgraph(&Driver,&Mode," ");
    ..............................
    closegraphQ;
    return 0;
}
```

Здесь функция **initgraph** вызывается в режиме автоматического тестирования аппаратуры. Поиск файла с драйвером выполняется в текущей директории.

Функция **closegraph()** освобождает память от драйвера и восстанавливает исходный видеорежим.

Используя функции **restorecrtmode** и **setgraphmode**, можно осуществлять переход в текстовый режим и обратно. При этом графический драйвер и графический буфер сохраняются в оперативной памяти. Синтаксис функций:

```
void far restorecrtmode(void);
void far setgraphmode(int mode);
```

Восстановить прежний графический режим можно, если запомнить его номер. Это можно сделать с помощью функции **getgraphmode**:

```
int far getgraphmode(void);
```

Эта функция возвращает текущее значение номера графического режима для активизированного драйвера.

2.3.2. Функции управления графическим окном

После установки графического режима по умолчанию графическим окном является весь экран. Левый верхний угол экрана имеет координаты **(0,0)**, а правый нижний имеет координаты **(getmaxx(),getmaxy())**, где **getmaxx** и **getmaxy** — функции, возвращающие значения максимальных координат.

Смена графического окна может быть выполнена с помощью функции **setviewport**:

```
void far setviewport(int left, int top, int right, int bottom, int
clip)
```

Её аргументами являются координаты окна, а также логическое выражение (**clip**), определяющее требуется ли отсекать части изображения, выходящие за пределы окна или нет. Все графические операции касаются текущего графического окна, а координаты графического курсора всегда отсчитываются от верх-

него левого угла экрана. Графический курсор невидим, но его текущие координаты можно определить с помощью функций **getx** и **gety**. Переместить курсор в требуемую позицию можно с помощью функций:

```
void far moveto (int x, int y);
void far moverel (int dx, int dy);
```

Первая функция перемещает курсор в позицию, заданную координатами **x** и **y**. Вторая функция перемещает курсор относительно текущей позиции на вектор **(dx,dy)**. Для очистки графического окна используется функция:

```
void far clearviewport(void);
```

При построении графических образов необходимо помнить, что на экране дисплея содержится разное количество пикселей по вертикали и горизонтали, а также то, что вертикальный и горизонтальный размеры пикселя неодинаковы. Это приводит к тому, что горизонтальный и вертикальный отрезки, содержащие одинаковое количество пикселей, на экране будут выглядеть как отрезки разной длины. Для правильного отображения требуется знать коэффициент пропорциональности (*aspect ratio*). Его можно определить с помощью функции:

```
void far getaspectratio(int far *xasp, int far *yasp);
```

Эта функция через свои аргументы возвращает искомое значение, причем ***yasp** всегда устанавливается равным 10000, величина ***xasp<=*yasp**. Отношение ***xasp** к ***yasp** и дает коэффициент пропорциональности.

2.3.3. Управление цветом и стилем заполнения фигур

При инициализации графического режима определяется доступная палитра цветов. Палитрой называют максимальный набор цветов, поддерживаемых одновременно *bgi*-драйвером.

Цвета нумеруются от **0** до **getmaxcolor()**. Выбор номера цвета для изображения обеспечивает функция **setcolor(int color)**. Цвет фона может быть изменен функцией **setbkcolor(int color)**. После выполнения функции **setcolor(n)** все объекты изображаются в цвете, связанном с позицией **n** палитры цветов. Числа от **0** до **15**, которые используются для обозначения цветов, определяют цветовые атрибуты пикселя или, как их ещё называют, «программные цвета». Каждому программному цвету присваивается аппаратный цвет из полной палитры в 256 цветов.

В программе для указания цвета можно вместо его номера использовать константы перечислимого типа **COLORS**, обозначающие цвета:

```
enum COLORS {BLACK, BLUE, GREEN, RED, MAGENTA, BROWN, LIGHTGRAY,
DARKGRAY, LIGHTBLUE, LIGHTGREEN, LIGHTCYAN, LIGHTRED,
LIGHTMAGENTA, YELLOW, WHITE};
```

Для управления соответствием между программными и аппаратными цветами используется ряд функций. Например, вызов **setpalette(0,RED)** присваивает первому цвету палитры красный цвет.

Для закрашивания геометрических фигур используются функции, заполняющие контур в соответствующем стиле:

```
setfillstyle(int pattern, int color);
floodfill(int x, int y, int border);
```

Первая функция устанавливает стиль и цвет заполнения (заливки), а вторая закрашивает замкнутую область, в которой содержится точка с координатами **(x,y)**. Параметр **border** задает цвет границы. Параметр **pattern** определяет стиль заполнения. Существует 12 заранее определенных стилей заполнения. Например:

EMPTY_FILL — заливка цветом фона;

SOLID_FILL — заливка заданным цветом;

LINE_FILL — штриховка горизонтальными линиями;

LTSLASH_FILL — диагональная штриховка и др.

Их символические имена задаются перечислимым типом **fill_patterns**.

2.3.4. Графические примитивы

Графические примитивы — это геометрические фигуры, которые можно отобразить на экране с помощью специальных функций. Все функции, описанные ниже, имеют тип **void far** и выполняют следующие действия:

line(int x1, int y1, int x2, int y2) — изображение отрезка прямой;

linerel(int dx, int dy) — изображение отрезка прямой относительно текущей позиции;

lineto(int x, int y) — изображение отрезка прямой из текущей точки в точку с заданными координатами;

circle(int x, int y, int radius) — изображение окружности;

arc(int x, int y, int stangle, int endangle, int radius) — изображение дуги радиуса **radius** с координатами **(x, y)** от угла **stangle** до **endangle**;

ellipse(int x, int y, int stangale, int endangale, int xradius, int yradius) — изображение эллиптической дуги с аналогичными параметрами;

rectangle(int x1, int y1, int x2, int y2) — изображение прямоугольника;

rawpoly(int numpoints, int far *polypoints) — изображение ломанной, заданной набором координат некоторого множества точек.

В последней функции параметр **numpoint** — это количество точек ломанной. Параметр-указатель ***polypoints** указывает на массив, каждый эле-

мент которого есть структура из двух полей, которые интерпретируются как координаты **(x,y)** очередной точки. Функцию **drawpoly** часто используют для вывода графиков функций, заданных таблично.

Доступ к отдельным пикселям активной страницы обеспечивают две функции:

```
unsigned far getpixel(int x, int y);
void far putpixel(int x, int y, int color);
```

Первая функция возвращает номер цвета пикселя с координатами **(x,y)**. Вторая выводит пиксель с координатами **(x,y)** цветом **color**.

Стилем и толщиной линий можно управлять с помощью функции

```
setlinestyle( int lynestyle, unsigned upattern, int thickness);
```

Аргумент **lynestyle** устанавливает стиль рисования линий, **upattern** — определяет пользовательский шаблон, а **thickness** — толщину линий. Доступные значения для толщины линий: **NORM_WIDTH** (нормальная) и **THICK_WIDTH** (толстая). При этом аргумент **thickness** воздействует на все контурные графические примитивы, а аргументы **lynestyle** и **upattern** — только на кусочно-линейные. Стиль рисования линий задается константами перечислимого типа:

```
enum line_styles {
      SOLID_LINE=0,     //сплошная линия
      DOTTED_LINE=1,    //точечная
      CENTER_LINE=2,    //штрихпунктирная
      DASHED_LINE=3,    //пунктирная
      USER_BIN=4};      //пользовательский стиль
```

При выборе значений указанных констант от **0** до **3** параметр **upattern** в функции **setlinestyle** игнорируется. Шаблон **upattern** используется, если значение **lynestyle** равно **4**. Он представляет собой 16-битовое слово. Если бит шаблона равен **1**, то соответствующий пиксель шаблона рисуется, в противном случае — нет. Так, пунктирная линия задается шаблоном **OxOFOF** или **0x3333**.

2.3.5. Отображение текстовой информации в графическом режиме

Текст в графическом режиме выводится с помощью шрифтов, которые сохраняются в файлах с расширением *chr*. Параметры вывода текста определяются функцией

```
settextstyle(int font, int direction, int charsize);
```

Аргумент **direction** задает направление вывода текста (**HORIZ_DIR**, **VERT_DIR**), аргумент **charsize** — размер символов (от **1** до **10**). Вид шриф-

та определяется значением аргумента **font** (от **0** до **4**). Все шрифты, кроме шрифта **DEFAULT_FONT=0**, являются векторными. Т.е. их элементы формируются как совокупность векторов, которые характеризуются направлением и длиной.

Расположением выводимой строки текста относительно опорной точки управляет функция

```
settextjustify(int horiz, int vert);
```

Её аргументы принимают значения перечислимого типа:

```
enum text_just{
      LEFT_TEXT=0,      //расположить слева
      CENTER_TEXT=1,    //расположить по центру
      RIGHT_TEXT=2,     //расположить справа
      BOTTOM_TEXT=0,    //расположить внизу
      TOP_TEXT=2 };     //расположить вверху
```

Вывод текста выполняется функциями

```
outtext (char far *textstring);
outtextxy(int x, int y, char far *textstring);
```

Обе функции в качестве аргумента получают указатель **textstring** на выводимую строку символов. За одно обращение к функции выводится одна строка. Первая функция выводит текст в текущую графическую позицию, вторая — в точку, заданную с помощью аргументов **x** и **y**.

2.3.6. Построение графиков функций

График функции $f(x)$ строят по точкам $(x, f(x))$. Для этого циклически задают значение абсциссы $\{x_i\}$ и вычисляют соответствующие ординаты $y_i = f(x_i)$. Затем отрезками соединяют точки (x_{i-1}, y_{i-1}) с точками (x_i, y_i).

Для изображения графика следует перевести вычисленные математические координаты точек в их экранные эквиваленты, умножив координаты $(x, f(x))$ на масштаб. С учетом того, что центр системы координат — это центр экрана, а экранная ось ординат направлена в обратную сторону по сравнению с математической осью, получим следующие выражения для экранных координат:

$$y\_\text{экрана} = y\_\text{центра} - \text{масштаб} \cdot f(x); \quad (2.1)$$

$$x\_\text{экрана} = x\_\text{центра} + \text{масштаб} \cdot x. \quad (2.2)$$

При построении графиков функций приведенные формулы используют в следующем порядке:

— циклически изменяют значение $x_$экрана от 0 до значения, равного максимальному числу пикселей в строке;

— для заданного значения $x_$экрана, преобразуя (2.2), вычисляют математическую координату x по формуле

$$x = (x\_\text{экрана} - x\_\text{центра})/\text{масштаб};$$

— вычисляют $y_$экрана по указанной выше формуле (2.1) и получают экранную точку с координатами $(y\_\text{экрана}, x\_\text{экрана})$;

— полученные экранные точки соединяют отрезками и получают график функции.

2.3.7. Преобразование координат и анимационные эффекты

Преобразование координат используют при изменении масштаба изображения, переносе и повороте изображений. Рассмотрим линейные преобразования координат двумерных объектов. Такие преобразования выполняются с помощью соответствующих формул.

Предположим, что в декартовой системе координат задана плоская геометрическая фигура и (x, y) — координаты одной из её точек. Пусть (x_1, y_1) координаты этой же точки после преобразования. Тогда параллельное перемещение фигуры на Δx пикселей вдоль оси $0x$ и на Δy пикселей вдоль оси $0y$ можно выполнить с помощью формул:

$$x_1 = x + \Delta x;$$
$$y_1 = y + \Delta y.$$

Растяжение фигуры в k_x и k_y раз выполняется на основе соотношений

$$x_1 = x k_x;$$
$$y_1 = y k_y.$$

Поворот фигуры вокруг начала координат на угол φ:

$$x_1 = x\cos\varphi + y\sin\varphi;$$
$$y_1 = -x\sin\varphi + y\cos\varphi.$$

В ходе разработки анимационных программ необходимо создавать эффект перемещения графических объектов. Простейший способ реализации этого эффекта заключается в том, чтобы нарисовать объект соответствующим цветом, а затем скрыть его путем повторного перерисовывания цветом фона. Затем изобразить объект в новых координатах. С целью повышения быстродействия программы при обработке изображений больших размеров их сначала копируют в оперативную память, а затем выводят на экран их копии в требуемых координатах. Для этого используют функции

```
void getimage(int x1 int y1, int x2, int y2, void far *bitmap);
void putimage(int x, int y, void far *bitmap, int op).
```

Функция **getimage** сохраняет в памяти изображение, очерченное прямо-

угольником, левым верхним углом которого является точка **(x1, y1)**, а правым нижним — точка **(x2, y2)**. Изображение сохраняется в памяти по указателю **bitmap**. Объём памяти, необходимый для хранения изображения, можно определить с помощью функции

```
imagesize(int x1 ,int y1, int x2, int y2);
```

Возвращаемое ею значение можно непосредственно передавать одной из функций для выделения оперативной памяти.

Функция **putimage** восстанавливает изображение на экране в координатах относительно левого верхнего угла **(x, y)**. Параметр **op**, принимающий возможные значения от **0** до **4**, обозначает способ взаимодействия новой копии изображения с изображением, которое уже имеется на экране. В простейшем случае, когда **op=0**, происходит простое копирование изображения из памяти. Значение **op**=**1** позволяет выполнять побитовую операцию «исключающего или» над содержимым оперативной памяти и видеобуфера для каждого пикселя. Это дает возможность удалять изображение в тех точках экрана, где помещен его оригинал. Если эту операцию выполнить дважды для одних и тех же координат экрана, то изображение не изменится. Это весьма полезно при программировании движущихся по экрану объектов.

Для того, чтобы движущийся по экрану объект отображался в каждой позиции определенное заданное время необходимо предусмотреть приостановление выполнения программы после вывода объекта на экран. Выполнить приостановку программы можно с помощью следующей функции

```
delay(unsigned int milliseconds);
```

Функция **delay** приостанавливает выполнение программы на время, определяемое переменной **milliseconds**, заданное в миллисекундах. Для использования функции **delay** необходимо подключить библиотеку **<dos.h>**.

2.3.8. Пример программы построения графика функции

Напишем программу, которая строит график функции

$$f(x)=|\sin x|+\cos|x|.$$

Область определения этой функции — вся ось абсцисс, а область значений $[-1,2]$. Отрезок оси ординат для удобства отображения следует растянуть. Для этого выполним масштабирование. Масштаб будем задавать целым числом от **20** до **100**. Значения масштаба будем хранить в переменной **scale**, координаты центра экрана — в переменных **cx** и **cy**. Алгоритм построения графика функций описан в разделе 2.3.6.

График строится с помощью функции **draw_function()**. Функции **axis()** и **razmetka()** строят координатные оси и размечают их. При разметке координатных осей используется функция **floattoa**, обеспечивающая

преобразование вещественного значения в строку.

Функция инициализации графического режима

```
initgraph(&dr,&mode,"c:\\borlandc\\bgi");
```

осуществляет поиск драйвера в директории **"c:\borlandc\bgi"**. В случае, если необходимые драйвера находятся в другом месте, то следует указать актуальный путь к директории, содержащей необходимые драйвера.

```
/*----------------------подключение библиотек--------------------*/
#include <stdio.h>     // библиотека стандартных средств
                       // ввода-вывода языка С
#include <graphics.h> // библиотека графических средств
#include <conio.h>     // библиотека управления вводом-выводом
                       // средствами консоли MSDOS
#include <math.h>      // библиотека математических функций
#include <stdlib.h>    // библиотека определения типов, переменных,
                       // и функций для работы с символами
#include <string.h>    // библиотека функций для работы со строками
                       // средствами языка С
/*----------------------глобальные переменные--------------------*/
float scale;               // масштаб
int cx,cy;                 // координаты центра экрана
int maxX,maxY;             // максимальное число пикселей по осям
const float PI=3.14159265;
/*-----------------------прототипы функций----------------------*/
float f(float x);          // отображаемая функция
void axis();               // вычерчивание осей координат
void razmetka();           // разметка осей координат
void draw_function();      // построение графика функции f(x)
char* floattoa(float j); // преобразование вещ. значения в строку
/*-----------------------основная функция----------------------*/
main() {
    int dr,mode;           // номер графического драйвера и режима
    int errorcode;         // код ошибки

    //установка графического режима
    dr=DETECT;               // автоматическое определение режима
    initgraph(&dr,&mode,"c:\\borlandc\\bgi");
    errorcode=graphresult(); // определение кода ошибки
    if (errorcode!=grOk) {
     printf("Graphic system error: %s\n",grapherrormsg(errorcode));
     exit(1);}               // выход, если не установлен граф. режим

    // определение максимального числа пикселей по осям
    // и центра экрана
    maxX=getmaxx();
    maxY=getmaxy();
    cx=maxX/2;
    cy=maxY/2;
```

```
    // установка цвета
    setbkcolor(WHITE);    // цвет фона белый
    setcolor(BLUE);       // основной цвет синий

    // ввод масштаба
    outtext("Input scale (20..100):");
    gotoxy(27,1);
    scanf("%f",&scale);
    cleardevice();        // очистка экрана
    axis();               // построение осей
    razmetka();           // разметка осей
    outtextxy(10,30,"y=|sin(x)+cos(x)|");
    draw_function();      // построение графика f(x)
    getch();
    closegraph();         // закрытие графического режима
    return 0;
}
/*--------------------вычисление функции f(x)-------------------*/
float f(float x){
       return fabs(sin(x))+cos(fabs(x));
}
/*-------------------построение осей координат-----------------*/
void axis(){
    setcolor(BLUE);
    line(cx,0,cx,maxY);             // ось y
    line(0,cy,maxX,cy);             // ось x
    line(maxX-10,cy-5,maxX,cy);     // стрелки на оси x
    line(maxX-10,cy+5,maxX,cy);
    line(cx,0,cx-5,10);             // стрелки на оси y
    line(cx,0,cx+5,10);
    outtextxy(maxX-10,cy+20,"X");   // надпись X
    outtextxy(cx+10,10,"Y");        // надпись Y
}
/*-------------------------разметка осей-----------------------*/
void razmetka(){
    float i,          // экранная координата оси
          j;          // логическая координата оси
    char *s;

    setcolor(BLUE);  // цвет рисок на осях
    // разметка оси x
    j=0;              // '0' в центре координатной системы
    do{
        // разметка левой части x
        i=cx-j*scale;           // вычисление экранной координаты
        line(floor(i),cy-5,floor(i),cy+5);  // простановка рисок
                                            // на оси
        if(j!=0){
            s=floattoa(-j);      // преобразование числа в строку
            outtextxy(floor(i)-10,cy+10,s); // вывод чисел на оси
        }
        // разметка правой части x
        i=cx+j*scale;
```

```
        line(floor(i),cy-5,floor(i),cy+5);
        if (j!=0){
            s=floattoa(j);
            outtextxy(floor(i)-10,cy+10,s);
        }
        j=j+PI;          // j изменяем с шагом Pi
    }
    while (j<=8*PI);     // вся ось X от-8*Pi до +8*Pi
    // разметка оси y
    j=0;
    do{
        // разметка верхней части оси y
        i=cy-j*scale;
        line(cx+3,floor(i),cx-3,floor(i));
        if (j!=0) {
            s=floattoa(j);
            outtextxy(cx+15,floor(i)-2,s);
        }
        // разметка нижней части оси y
        i=cy+j*scale;
        line(cx+3,floor(i),cx-3,floor(i));
        if (j!=0) {
            s=floattoa(-j);
            outtextxy(cx+15,floor(i)-2,s);
        }
        j=j+0.5;
    }
    while(j<=2);         // вся ось Y от-2 до +2
}
/*-------------------построение графика функции-----------------*/
void draw_function() {
    int x0, y0, x1, y1;       // экранные координаты концов отрезка
    float x;                  // аргумент функции

    setcolor(RED);
    for(x0=0;x0<=maxX;x0++){
        x=(x0-cx)/scale;              // вычисление аргумента x по x0
        y0=floor(cy-scale*f(x));      // вычисление экран. коорд. y0
        if(x0>0) line(x0,y0,x1,y1); // построение отрезка
        x1=x0;                        // копирование экранных
        y1=y0;                        // координат
    }
}
/*-----------преобразование вещественного числа в строку--------*/
char* floattoa(float j) {
    int i, l;
    int dec, sign;
    const int ndig=2;   // число цифр в строке будет 2+1
    char *s;

    // fcvt - стандартная функция библиотеки stdlib.h
    s=fcvt(j,ndig,&dec,&sign); // копирование цифр из j в s
                               // dec-позиция десятичной точки
```

```
                                    // sign>0 если j<0;
    // формирование числовой строки с правильным
    // следованием символов
    // копируем строку s в строку snew с учетом знака и
    // позиции десятичной точки
    char *snew=new char[strlen(s)+3];

    i=l=0;
    if(sign){snew[l]='-'; l++;}
    while(s[i]!='\0'){
        if(i==dec) {snew[l]='.'; l++;}
        snew[l]=s[i];
        i++;
        l++;
    }
    s[l]='\0';
    return snew;    //возвращаем snew
}
```

Приведенная программа выводит на экран график, который изображен на рисунке 2.1.

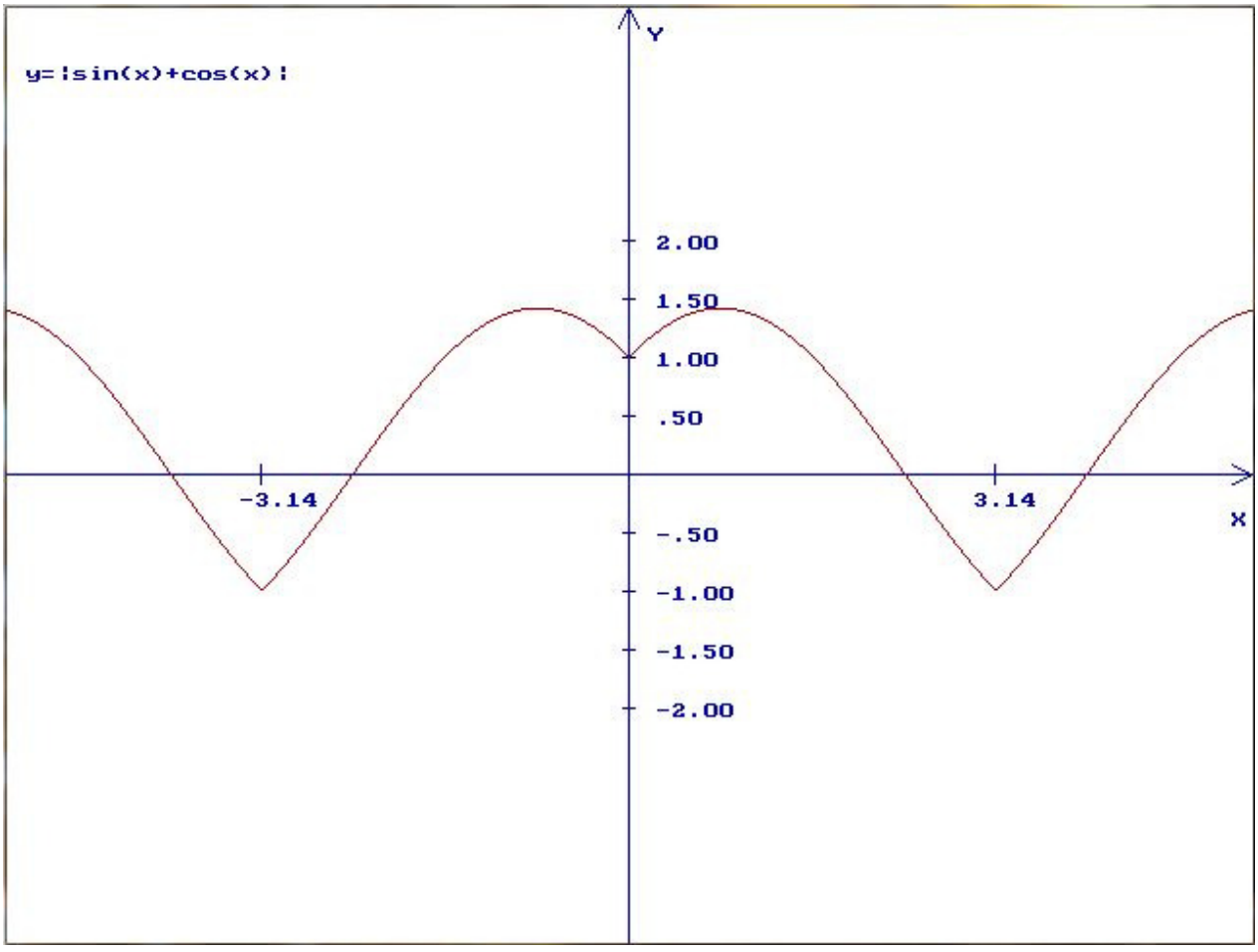

Рисунок 2.1 — Вывод графика и подписей программой
в графическом режиме

2.4. Порядок выполнения работы

Вариант задания выдаётся преподавателем. Перед выполнением работы необходимо ознакомиться с разделами 2.1 — 2.3 данных методических указаний. Индивидуальные задания для выполнения работы см. в разделе 2.2.

2.4.1. Внимательно изучить индивидуальное задание.

2.4.2. Выполнить анализ области значений аргумента, функции и экранных координат, выбрать при необходимости масштаб.

2.4.3. Построить график заданной функции. Для построения графика рекомендуется использовать инженерно-математическую программу *MathCAD* [15].

2.4.4. Разработать схему алгоритма решения задачи, разбив его при необходимости на отдельные этапы (функции).

2.4.5. Разработать программу на языке *C*.

2.4.6. Выполнить тестирование и отладку написанной программы.

2.4.7. Получить результаты работы программы. Рекомендуется скопировать в виде изображения график, выводимый программой, для последующего включения в отчет.

2.4.8. Подготовить отчет по работе.

2.5. Оформление отчета

2.5.1. Сформулировать цель работы.

2.5.2. Привести постановку задачи и текст индивидуального задания.

2.5.3. Привести краткие теоретические сведения касающиеся использования графического режима.

2.5.4. Привести математическое обоснование задачи:

— анализ области допустимых значений аргумента;

— график функции;

— анализ экранных координат.

2.5.5. Изобразить схему алгоритма решения задачи. Записать краткие пояснения к алгоритму.

2.5.6. Привести текст программы. Текст программы обязательно должен содержать комментарии.

2.5.7. Составить таблицу, в которой указать имена всех переменных используемых в программе и их тип, а также кратко описать назначение переменных.

2.5.8. Привести результаты работы программы.

2.5.9. Сделать выводы по работе.

2.6. Контрольные вопросы

2.6.1. Чем отличаются текстовый и графический режим работы дисплея?

2.6.2. Что понимают под разрешающей способностью дисплея?

2.6.3. Что такое видеопамять, видеоадаптер?

2.6.4. Для чего предназначены драйверы графической системы? Как узнать какие режимы поддерживаются видео драйвером?

2.6.5. Опишите функции **`detectgraph`** и **`initgraph`**?

2.6.6. Как вызвать функцию **`initgraph`** в режиме автоматического тестирования аппаратуры?

2.6.7. Какие функции существуют в *Borland C* для перехода из графического режима в текстовый и обратно? Опишите их?

2.6.8. Объясните систему координат графического окна? Как устанавливаются его размеры? Каким образом производится его очистка?

2.6.9. Для чего нужно знать коэффициент пропорциональности вертикальных и горизонтальных отрезков?

2.6.10. Каким образом в *Borland C* выполняется управление цветом?

2.6.11. Как отобразить текст в графическом режиме?

2.6.12. Какие графические примитивы можно отображать встроенными функциями языка *Borland C*?

2.6.13. Как управлять стилем рисования линий?

2.6.14. Какие функции языка *Borland C* обеспечивают управление отдельными пикселями?

2.6.15. Приведите формулы преобразования логических координат в экранные.

2.6.16. Приведите формулы линейного преобразования координат.

2.6.17. Как реализовать анимацию средствами *Borland C*?

3. ЛАБОРАТОРНАЯ РАБОТА №3
ПРОГРАММИРОВАНИЕ ОПЕРАЦИЙ НАД СТРОКАМИ И ФАЙЛАМИ

3.1. Цель работы

Изучить основные принципы операций над строками и файлами в языках *C*/*C*++.

Исследовать свойства файловых указателей.

3.2. Варианты заданий

Разработать алгоритм обработки данных, полученных из текстового файла, в соответствии с вариантом задания. Написать программу, которая считывает из текстового файла данные и обрабатывает их в соответсвии с заданием. Вараинт задания выбирается по указанию преподавателя.

Вариант 1

Написать программу, которая считывает из текстового файла четыре предложения и выводит их на экран в обратном порядке, т.е. первым выводит четвертое предложение, а последним — первое. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 2

Написать программу, которая считывает текст из файла и выводит на экран только те предложения, которые содержат введенное с клавиатуры слово. Ввод-вывод осуществлять с помощью средств языка *C*++.

Вариант 3

Написать программу, которая считывает текст из файла и выводит на экран только те строки, которые содержат двузначные числа. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 4

Написать программу, которая считывает текст из файла и выводит на экран слова, начинающиеся с гласных букв. Ввод-вывод осуществлять с помощью средств языка *C*++.

Вариант 5

Написать программу, которая считывает текст из файла и выводит его на экран, меняя местами каждые два соседних слова. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 6

Написать программу, которая считывает текст из файла и выводит на экран только те предложения, которые не содержат запятые. Ввод-вывод осуществлять с помощью средств языка *C++*.

Вариант 7

Написать программу, которая считывает текст из файла и определяет, сколько в нем слов, состоящих не более чем из четырех букв. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 8

Написать программу, которая считывает текст из файла и выводит на экран только цитаты, то есть предложения, заключенные в кавычки. Ввод-вывод осуществлять с помощью средств языка *C++*.

Вариант 9

Написать программу, которая считывает текст из файла и выводит на экран только те предложения, которые состоят из заданного количества слов. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 10

Написать программу, которая считывает текст из файла и выводит на экран слова, начинающиеся и оканчивающиеся на гласные буквы. Ввод-вывод осуществлять с помощью средств языка *C++*.

Вариант 11

Написать программу, которая считывает текст из файла и выводит на экран только те строки, которые не содержат трехзначные числа. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 12

Написать программу, которая считывает текст из файла и выводит на экран только предложения, начинающиеся с тире, перед которым могут находиться только пробельные символы. Ввод-вывод осуществлять с помощью средств языка *C++*.

Вариант 13

Написать программу, которая считывает текст из файла и выводит его на экран, заменив каждую первую букву слов, начинающихся с гласной буквы, на прописную. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 14

Написать программу, которая считывает текст из файла и выводит его на

экран, заменив встречающиеся в тексте цифры от 0 до 9 на слова «ноль», «один»...«девять», начиная каждое предложение с новой строки. Ввод-вывод осуществлять с помощью средств языка *C++*.

Вариант 15

Написать программу, которая считывает текст из файла, находит самое длинное слово и определяет, сколько раз оно встретилось в тексте. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 16

Написать программу, которая считывает текст из файла и выводит на экран сначала вопросительные, а затем восклицательные предложения. Ввод-вывод осуществлять с помощью средств языка *C++*.

Вариант 17

Написать программу, которая считывает текст из файла и выводит его на экран, добавляя после каждого предложения информацию о том, сколько раз встретилось в нем введенное с клавиатуры слово. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 18

Написать программу, которая считывает текст из файла и вычисляет количество открытых и закрытых скобок в каждой строке. Вывести на экран считанный текст, добавляя вычисленные значения в конец каждой строки. Ввод-вывод осуществлять с помощью средств языка *C++*.

Вариант 19

Написать программу, которая считывает текст из файла и выводит на экран предложения, содержащие максимальное количество знаков пунктуации. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 20

Считать два текстовых файла. Удалить из двух считанных текстов строки, которые имеют одинаковые номера, но сами не являются одинаковыми. Результаты вывести на экран. Ввод-вывод осуществлять с помощью средств языка *C++*.

Вариант 21

В каждой строке текстового файла найти наиболее длинную последовательность цифр. Результаты вывести на экран. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 22

Считать два текстовых файла. Вывести на экран символы, которые записа-

ны в позициях с одинаковыми номерами, но различаются между собой, и номер позиции. Ввод-вывод осуществлять с помощью средств языка *C++*.

Вариант 23

Считать файл с текстом программы на языке *C*/*C++*. Исключить комментарии из текста программы, результат вывести на экран. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 24

Написать программу, которая подсчитывает количество пустых строк в текстовом файле. Ввод-вывод осуществлять с помощью средств языка *C++*.

Вариант 25

Написать программу, которая находит максимальную длину строки текстового файла и печатает эту строку. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 26

Написать программу для подсчета количества строк текстового файла, начинающихся и оканчивающихся одной и той же литерой. Ввод-вывод осуществлять с помощью средств языка *C++*.

Вариант 27

Написать программу для подсчета количества строк текстового файла, состоящих из одинаковых литер. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 28

Написать программу, выводящую слова каждого предложения текстового файла на экран в обратном порядке, т.е. первым выводится последнее слово предложения, а последним — первое слово предложения. Ввод-вывод осуществлять с помощью средств языка *C++*.

Вариант 29

Написать программу, которая считывает текст из файла и выводит на экран сначала предложения, начинающиеся с однобуквенных слов, а затем все остальные. Ввод-вывод осуществлять с помощью средств языка *C*.

Вариант 30

Написать программу, которая считывает текст из файла и выводит на экран все его предложения в обратном порядке. Ввод-вывод осуществлять с помощью средств языка *C++*.

3.3. Теоретические сведения

3.3.1. Строки

В языке программирования *C*/*C*++ строка представляет собой массив символов, завершающийся терминатором строки — символом с нулевым значением (**'\0'**). При объявлении строкового массива необходимо принимать во внимание наличие терминатора в конце строки, отводя тем самым под строку на один байт больше. Удобно задавать длину строки именованной константой:

```
const int lenstr = 80;
char stroka[len_str];
```

В переменной **stroka** можно хранить не 80 символов, а только 79. Строки можно при описании инициализировать строковой константой. При этом нуль-символ (**'\0'**) добавляется в строку автоматически:

```
char a[10] = "Язык Си";
```

Если строка при определении инициализируется, её длину можно не указывать:

```
char a[] = "Язык Си";
```

В этом случае компилятор сам выделит память достаточную для размещения всех символов и нуль-литеры.

Для размещения строки в динамической памяти необходимо описать указатель на **char**, а затем выделить память с помощью функции **malloc()** или **new**:

```
char *s1 = (char*) malloc(m*sizeof(char));
char *s2 = new char[m];
```

Здесь **m** — переменная, определяющая длину строки. Для ввода-вывода строк можно использовать как объекты **cin** и **cout** языка *C*++, так и функции языка *C*. Рассмотрим первый способ:

```
#include <iostream.h>
void main() {
    const int n = 80;
    char s[n];
    cin >> s;
    cout << s << endl;
}
```

Таким образом, строка вводится аналогично числовым переменным. Однако, если ввести строку, состоящую из нескольких слов, разделенных пробелами, то будет введено только первое слово. Это связано с тем, что ввод выполняется до первого пробельного символа (**' '**, **'\t'**, **'\n'**). Поэтому для ввода строки лучше использовать функции **getline** или **get**:

```
#include <iostream.h>
void main() {
    const int n = 80;
    char s[n];
```

```
    cin.getline(s,n);
    cout<<s<<endl;
    cin.get(s,n);
    cout<<s<<endl;
}
```

Функция **getline** считывает из входного потока (**n–1**) символов или менее (если символ **'\n'** встретится раньше) и записывает их в строковую переменную **s**. Сам символ **'\n'** тоже считывается, но не записывается в **s**, вместо него записывается нуль-литера (**'\0'**). Если в потоке более (**n–1**) символа, то следующий ввод будет выполняться, начиная с первого несчитанного символа.

Функция **get** работает аналогично, но оставляет в потоке символ **'\n'**. Поэтому перед повторным вызовом функции **get** необходимо удалять символ **'\n'** из входного потока. Для этого нужно вызвать функцию **get** без параметров:

```
cin.get();
```

В языке программирования *C* для ввода и вывода строк используются функции **scanf** и **printf** со спецификацией **%s**:

```
#include <stdio.h>
void main()
{
    const int n = 40;
    char s[n];
    scanf("%s",s);
    printf("%s",s);
}
```

В функции **scanf** перед переменной **s** операция взятия адреса (**&**) опущена, потому что имя массива является указателем на его начало. Функция **scanf** выполняет ввод до первого пробельного символа. Чтобы ввести строку, состоящую из нескольких слов, используйте спецификацию **%c**:

```
scanf("%20c",s);
```

В этом случае количество считываемых символов может быть только константой, и короткие строки придется при вводе дополнять пробелами до требуемой длины строки.

Библиотека языка программирования *C* содержит функции, специально предназначенные для ввода (**gets**) и вывода (**puts**) строк.

Функция **gets(s)** читает символы с клавиатуры до появления символа **'\n'** и помещает их в строку **s** (сам символ **'\n'** в строку не включается, вместо него вносится **'\0'**). Функция возвращает указатель на **s**, а в случае ошибки или конца файла — **NULL**. Однако функция **gets** не контролирует количество введенных символов в **s**, что может приводить к выходу за границы массива **s**.

Функция **puts(s)** выводит строку **s** на стандартное устройство вывода, заменяя завершающий символ **'\0'** на символ новой строки.

Большинство функций работы со строками описаны в библитечном файле **<string.h>**. Некоторые из этих функций указаны в таблице 3.1. Здесь аргументы **s** и **t** принадлежат типу **char***, **cs** и **ct** — типу **const char***, **n** — типу **size_t**, **c** — типу **int**, приведенному к **char**.

Например, чтобы строке **s** присвоить значение строки **t** можно использовать функции **strcpy** или **strncpy**:

```
char t[]=" Язык Си";
char *s = new char[m];
strcpy(s,t);
strncpy(s,t,strlen(t)+1);
```

Функция **strcpy(s,t)** копирует все символы из строки **t**, включая и нуль-литеру, в строку, на которую указывает **s**. Функция **strncpy(s,t,n)** выполняет то же самое, но не более **n** символов. При этом если нуль-символ встретится раньше, копирование прекращается, и оставшиеся символы строки **s** заполняются нуль-символами. Если **n** меньше или равно длине строки **t**, завершающая нуль-литера не добавляется. Обе функции возвращают указатель на результирующую строку.

Функция **strlen(t)** возвращает фактическую длину строки **t**, не включая нуль-литеру.

Таблица 3.1 — Функции для обработки строк

Функция	Описание функции
char* strcpy(d,s)	Копирует строку **s**, включая **'\0'**,в строку **d**; возвращает **d**.
char* strcat(d,s)	Добавляет в конец строки **d** строку **s** и **'\0'**.
char* strcmp(s1,s2)	Сравнивает строки с учетом регистра символов.
char* strchr(s,ch)	Возвращает указатель на первое найденное вхождение (положение) символа **'ch'** в строке **s**.
char* strrchr(s,ch)	Возвращает указатель на последнее найденное вхождение (положение) символа **'ch'** в строке **s**.
size_t strspn(s1,s2)	Возвращает индекс первого символа строки **s1**, отсутствующего в **s2**.
size_t strcspn(s1,s2)	Возвращает индекс первого символа строки **s1**, совпадающего с любым из символов строки **s2**.
char* strpbrk(s1,s2)	Возвращает указатель на символ, являющийся первым вхождением любого из символов из **s2** в строку **s1**.
char* strstr(s1,s2)	Выполняет поиск первого вхождения подстроки **s2** в строку **s1** возвращает указатель на элемент из **s1**, с которого начинается **s2**.
size_t strlen(s)	Возвращает длину строки **s**.
char* strtok(s1,s2)	Возвращает лексему из **s1**, отделенную началом **s1** и любым символом из набора **str2**.

3.3.2. Функции файлового ввода-вывода

Язык программирования *C++* унаследовал от языка *C* библиотеку стандартных функций ввода-вывода. Функции ввода-вывода объявлены в заголовочном файле **<stdio.h>**. Операции ввода-вывода осуществляются с файлами. Файл может быть текстовым или бинарным (двоичным). Различие между ними заключается в том, что текстовый файл разбит на строки. Признаком конца строки является пара символов **'\r'** (возврат каретки) и **'\n'** (перевод строки). При вводе или выводе значений из текстового файла они требуют преобразований во внутреннюю форму хранения этих значений в соответствии с их типами, объявленными в программе. Бинарный файл — это просто последовательность символов. При вводе-выводе информации в бинарные файлы никаких преобразований не выполняется. Поэтому работа с бинарными файлами выполняется быстрее.

В языке программирования *C* реализован потоковый ввод-вывод. Термин «поток» происходит из представления процесса ввода-вывода информации в файл в виде последовательности или потока байтов. Все потоковые функции ввода-вывода обеспечивают буферизированный, форматированный или неформатированный ввод-вывод.

Перед тем, как выполнять операции ввода или вывода в файловый поток его следует открыть с помощью функции **fopen()**. Эта функция имеет следующий формат записи:

```
FILE *fopen(const char *filename, const char *mode);
```

Функция **fopen()** открывает файл с именем **filename** и возвращает указатель на файловый поток, используемый в последующих операциях с файлом. Параметр **mode** является строкой, задающей режим, в котором открывается файл. Он может принимать значения, указанные в таблице 3.2.

Таблица 3.2 — Режимы открытия файлов

Режим	Описание режима
r	Файл открывается только для чтения.
w	Файл создается только для записи (стирает предыдущий).
a	Файл открывается для добавления данных в конец файла. Чтение не разрешено.
r+	Существующий файл открывается для чтения и записи.
w+	Создается новый файл для чтения и записи (стирает предыдущий).
a+	Файл открывается для чтения и добавления данных в конец файла. Если файл не существует, то он создается.
t	Файл создается или открывается как текстовый (по умолчанию).
b	Файл создается или открывается как бинарный.

Чтобы указать, что данный файл открывается или создается как текстовый, необходимо добавить символ **t** в строку режима (например, **"rt"**, **"w+t"** и

т.п.). Аналогично можно сообщить, что файл открывается или создается как бинарный. Для этого следует добавить в строку режима символ **b** (например, **"wb"**, **"a+b"**).

В случае успеха функция **fopen** возвращает указатель на открытый поток, в случае ошибки — **NULL**. Например:

```
FILE *fptr = fopen("myfile.txt","rt");
```

Здесь объявляется переменная **fptr** — указатель на файловый поток и выполняется её инициализация с помощью функции **fopen()**.

Для завершения работы с потоком он должен быть закрыт с помощью функции **fclose()**:

```
fclose(fptr);
```

Рассмотрим функции, осуществляющие ввод-вывод в файловый поток.

Функция **fgetc()** имеет следующий формат записи:

```
int fgetc(FILE* fptr);
```

Она осуществляет ввод символа из файлового потока **fptr**. В случае успеха функция возвращает код символа. Если делается попытка прочесть конец файла или произошла ошибка, то возвращается код **EOF**.

Функция **fputc()** осуществляет вывод символа **ch** в поток:

```
int fputc(int ch, FILE *fptr);
```

Функция **fgets()** имеет следующий формат записи:

```
char *fgets(char *s,int n, FILE *fptr);
```

Она осуществляет чтение строки символов из файлового потока в строку **s**. Функция прекращает чтение, если прочитано (**n–1**) символов или встретится символ перехода на новую строку **'\n'** Если этот символ встретился, то он сохраняется в переменной **s**. В обоих случаях в переменную **s** добавляется символ **'\0'**. В случае успеха функция возвращает указатель на считанную строку **s**. Если произошла ошибка или считана метка **EOF**, то функция возвращает **NULL**.

Для записи строки в файл можно использовать функцию **fputs()**. При этом символ перехода на новую строку не добавляется, и завершающая нуль-литера **'\0'** в файл не копируется. Функция **fputs()** имеет следующий формат записи:

```
int fputs(const *char, FILE *fptr);
```

В случае успеха функция **fputs()** возвращает неотрицательное значение, в противном случае она возвращает **EOF**.

Форматированный ввод-вывод текстовых файлов организуется в языке программирования *C* с помощью функций **fscanf()** и **fprintf()**. Эти функции аналогичны функциям **scanf()** и **printf()** соответственно, с той лишь разницей, что их первым аргументом является указатель на файл, откры-

тый в соответствующем режиме:

```
int fscanf(FILE *fptr, const char *format[,a1,a2,...]);
int fprintf(FILE *fptr, const char *format[,a1,a2,...]);
```

Функция **feof()** осуществляет проверку достижения метки конца файла:

```
int feof(FILE *fptr);
```

Функция возвращает **0**, если конец файла не достигнут.

Рассмотрим пример файлового ввода и вывода с помощью функций **fscanf()** и **fprintf()**:

```
#include <stdio.h>
// определение файлов для чтения и записи
#define INFILE  "C:\\borlandc\\students\\old.dat"
#define OUTFILE "C:\\borlandc\\students\\new.dat"

int main()
{
    int i,x;
    FILE *in,*out; // описание указателей на файлы

    // открытие файла для чтения (входного потока)
    // и проверка результата открытия
    if ((in = fopen(INFILE,"rt"))== NULL)
    {
        fprintf(stderr,"Не могу открыть входной файл '%s'\n",
                       INFILE);
        return 1;
    }

    // открытие файла для записи (выходного потока)
    // и проверка результата открытия
    if ((out = fopen(OUTFILE,"wt"))== NULL)
    {
        fprintf(stderr,"Не могу открыть выходной файл '%s'\n",
                       OUTFILE);
        return 1;
    }

    i = 0;
    while (fscanf(in,"%d",&x)!=EOF)
    {
        i++;
        fprintf(out,"%d\t%d\n",i,x);
    }

    // закрытие входного и выходного потоков
    fclose(in);
    fclose(out);
    return 0;
}
```

Программа копирует целые числа из входного файла **old.dat** в выходной файл **new.dat**. При этом в выходной файл записываются также порядковые номера чисел. Копирование выполняется до тех пор, пока не будет встречена метка конца файла **EOF** во входном файле. Если входной или выходной файл нельзя открыть, то программа выводит соответствующее сообщение в стандартный поток **stderr**, который по умолчанию связан с пользовательским терминалом (экраном) и предназначен для вывода сообщений об ошибках.

Для осуществления неформатированного ввода-вывода (без преобразований) применяются функции **fread()** и **fwrite()**:

```
sizet fread(void *bptr, size_t size, size_t n, FILE *fptr);
sizet fwrite(const void *bptr, size_t size, size_t n, FILE *fptr);
```

Функция **fread()** считывает блоки данных из файлового потока, на который указывает указатель **fptr**, в буфер, доступ к которому выполняется через указатель **bptr**, а функция **fwrite()** выполняет обратную операцию, то есть переписывает блоки данных из буфера в файловый поток. При этом копируется **n** блоков данных, каждый из которых содержит **size** байтов. В случае успеха функции возвращают число скопированных блоков. В случае ошибки возвращается **0** или число полностью скопированных блоков. Если **n** больше значения, которое вернула функция **fread()**, то это говорит о том, что встретилась метка конца файла. Тип переменной **size_t** соответсвует типу **unsigned int**.

Когда файл открывается для чтения или записи, с ним на самом деле связывается структура типа **FILE**, определенная в заголовочном файле **<stdio.h>**. Эта структура для каждого открытого файла содержит счетчик положения текущей записи. Сразу после открытия файла его значение равно **0**. Каждая операция ввода-вывода вызывает приращение этого счетчика на число записанных или считанных байтов из файла. Функции позиционирования позволяют изменять или получать значение счетчика, связанного с файлом: **fseek()**, **ftell()** и **rewind()**.

Функция **ftell()** возвращает текущее значение счетчика связанного с файлом. В случае ошибки она возвращает значение **–1L**. Функция имеет следующий формат записи:

```
long int ftell(FILE *fptr);
```

Функция **fseek()** изменяет позиционирование файлового потока **fptr** на **offset** байтов относительно позиции, определяемой параметром **from**:

```
int fseek(FILE *fptr, long offset, int from);
```

Параметр **from** может принимать следующие значения:

SEEK_SET (=0) — начало файла;

SEEK_CUR (=1) — текущая позиция в файле;

SEEK_END (=2) — конец файла.

Функция **fseek()** возвращает **0**, если счетчик текущей записи успешно

изменен.

Функция **rewind()** устанавливает счетчик текущей позиции на начало потока:

```
void rewind(FILE *fptr);
```

3.3.3. Файловый ввод-вывод с использованием средств языка *C++*

Для реализации ввода-вывода с использованием средств языка *C++* в программу следует включить заголовочный файл **<fstream.h>**. Для осуществления операций с файлами библиотека ввода-вывода *C++* предусматривает три класса:

ifstream — потоковый класс для ввода из файла;

ofstream — потоковый класс для вывода в файл;

fstream — потоковый класс для ввода-вывода в файл.

Для создания потока ввода, открытия файла и связывания его с потоком можно использовать следующую форму конструктора класса **ifstream**:

```
ifstream *fptr(const char *filename, int mode);
```

Например:

```
ifstream fin("text.txt", ios::in);
```

Здесь определяется объект **fin** класса входных потоков **ifstream**. С этим объектом можно работать так же, как со стандартными объектами **cin** и **cout**, т.е. использовать операции помещения в поток **<<** и извлечения из потока **>>**, а также рассмотренные выше функции **get**, **getline**. Например:

```
const length = 80;
char s[length];
fin.getline(s, length);
```

Предполагается, что файл с именем **text.txt** находится в том же каталоге, что и текст программы, иначе следует указать полный путь к файлу, дублируя символ обратной косой черты, так как иначе он будет иметь специальное значение, например:

```
ifstream fin("C:\\borlandc\\students\\text.txt", ios::in);
```

Второй параметр этого конструктора означает режимы открытия файла. В таблице 3.3 приведены возможные режимы открытия и их значение.

Например, для открытия файла вывода в режиме добавления можно использовать инструкцию:

```
ofstream fout("out.txt",ios::out | ios::app);
```

Файлы, которые открываются для вывода, создаются, если они не существуют, за исключением тех случаев, когда используется режим **ios::nocreate**.

Таблица 3.3 — Режимы открытия файлов

Режим	Описание режима
ios::in	Файл открывается для чтения.
ios::out	Файл создается для записи.
ios::ate	Файл открывается для добавления данных. Указатель устанавливается на конец файла, но его можно смещать.
ios::app	Файл открывается для добавления данных строго в конец файла. Указатель всегда указывает в конец файла.
ios::trunc	Если файл существует, то удалить.
ios::nocreate	Если файл не существует, то выдать ошибку.
ios::noreplace	Если файл существует, то выдать ошибку.
ios::binary	Файл создается или открывается как бинарный.

Если открытие файла завершилось неудачей, объект, соответствующий потоку, будет возвращать значение **false**. Например, если предыдущая инструкция завершилась неудачей, то это можно проверить так:

```
ofstream fout("out.txt",ios::out | ios::app);
if(!fout) {
    cout << " Файл не открыт \n";}
```

Если при открытии файла не указан режим **ios::binary**, то файл открывается как текстовый. В этом случае можно использовать при работе с файлом функции ввода-вывода, принятые в языке *C*, такие, как **fprintf()** и **fscanf()**.

Для проверки достигнут ли конец файла или нет, можно использовать функцию **eof()** класса **ios**:

```
fin.eof();
```

Если конец файла не достигнут, то функция возвращает значение **0**.

Завершив операции ввода-вывода, необходимо закрыть файл, вызвав функцию **close()**:

```
fout.close();
```

Закрытие файла происходит автоматически при выходе объекта потока из области видимости или при завершении работы программы.

Произвольный доступ к записям файлов реализуется с помощью функций (методов) **seekg()** и **seekp()**, используемых, соответственно, для позиционирования входного и выходного потоков. Например:

```
fin.seekg(0, ios::end);
```

Функция **seekg(offset, arg)** перемещает текущую позицию чтения из файла на **offset** байтов относительно позиции **arg**. Параметр **arg** может принимать одно из трех значений:

ios::beg — относительно начала файла;

ios::cur — относительно текущей позиции;

ios::end — относительно конца файла.

Получить текущее значение позиции в потоке ввода и вывода можно с помощью функций **tellg()** и **tellp()**, соответственно. Например, вызов **fin.tellg()** передаст значение счетчика текущей позиции в точку вызова.

Для осуществления неформатированного ввода-вывода при работе с потоками в языке *C++* применяются функции **read()** и **write()**. Эти функции записываются в следующем формате:

```
input_file_pointer.read(char *s, streamsize n);
output_file_pointer.write(const char *s, streamsize n);
```

Здесь параметр **s** указывает на буфер, в который соответственно помещаются или из которого извлекаются символы, а **n** означает число читаемых или записываемых символов. Примеры вызова этих функций приведены ниже в тексте программы.

3.3.4. Пример программы обработки текстового файла

Рассмотрим пример программы, которая считывает текст из файла и выводит в выходной файл только вопросительные предложения из этого текста.

Исходные данные. Исходные данные хранятся в текстовом файле неизвестного размера. Предложение может занимать несколько строк текстового файла, поэтому ограничиться буфером на одну строку в данной задаче нельзя. В связи с этим выделим в программе буфер, в который поместится весь файл.

Результаты. Результаты являются частью исходных данных, поэтому дополнительно под них пространство выделять не требуется.

Промежуточные величины. Для организации вывода предложений понадобятся переменные целого типа, в которых будем хранить позиции начала и конца предложения.

Алгоритм решения задачи.

Для решения задачи необходимо выполнить следующие действия:

1) Открыть файл.

2) Определить его длину.

3) Выделить в динамической памяти соответствующий буфер.

4) Считать файл с диска в буфер.

5) Анализируя буфер посимвольно, выделять предложения. Если предложение оканчивается вопросительным знаком выводить его в файл.

Детализируем последний пункт алгоритма. Для вывода предложения необходимо хранить позиции его начала и конца. Предложение может оканчиваться точкой, восклицательным или вопросительным знаком. В двух первых случаях предложение пропускается. Это выражается в том, что значение позиции начала предложения обновляется. Она будет соответствовать позиции символа, следующего за текущим символом, и просмотр буфера продолжается.

В программе следует использовать функции неформатированного чтения блоков данных из входного файла, так как применение функций посимвольного чтения неэффективно.

Ниже приведен текст программы, позволяющей решить эту задачу с исполь-

зованием функций языка *C*. Для определения длины файла воспользуемся средствами библиотеки **<stdio.h>**. Для этого переместим указатель текущей позиции в конец файла с помощью **fseek()**, а затем с помощью **ftell()** получим значение конечной позиции (и длину файла) и запишем его в переменную **len**.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    // открытие входного файла
    FILE *fin;
    fin=fopen("C:\\borlandc\\students\\textin.txt","r");
    if(!fin) { // проверка результата открытия
        puts("Невозможно открыть входной файл");
        return 1;
    }
    // неформатированное чтение текстового файла (поблочное)
    fseek(fin,0,SEEK_END);      // указатель в конец файла
    int len=ftell(fin);         // запомнить длину файла
    const int block_len=512;    // размер считываемых блоков
    int block_num=len/block_len            //вычисление количества
                +((len%block_len) ? 1 : 0);//блоков в файле
    // выделить память под буфер
    char *buf=(char*)malloc(sizeof(char)*(len+1));
    rewind(fin);                // указатель в начало файла
    printf("Чтение данных из файла.\n");// чтение несколькими
    fread(buf,block_num,block_len,fin); // блоками
    buf[len]='\0';              //поместить в буфер нуль-литеру

    // создание выходного файла
    FILE *fout;
    fout = fopen("C:\\borlandc\\students\\textout.txt","w");
    if(!fout) { // проверка результата открытия
        puts("Невозможно открыть выходной файл");
        return 1;
    }
    int n=0,    // индекс символа начала предложения
        i=0,    // индекс символа конца предложения
        j=0;    // текущий индекс символа вопросительного
                // предложения
    printf("Запись данных в файл.\n");
    while(buf[i]) { // просмотр символов в буфере
        if(buf[i]=='?') {         // Если i-ый символ – вопрос,
            for(j=n;j<=i;j++)
                putc(buf[j],fout);// то вывод предложения в поток
            n=i+1;                // и обновление индекса начала
        }                         // предложения.
        if (buf[i]=='.'||buf[i]=='!') // Если предложение
        {                             // не вопросительное,
            n=i+1;                    // то обновление только
        }                             // индекса n.
```

```
        i++;
    }
    fclose(fin);    // закрытие входного файла
    fclose(fout);   // закрытие выходного файла
    printf("Программа завершена.\n");
    free(buf); return 0;
}
```

Вариант решения этой же задачи с использованием средсв языка *C++* представлен ниже. Здесь для определения длины файла воспользуемся функциями **seekg()** и **tellg()** класса **ifstream**. Вначале переместим указатель текущей позиции в конец файла с помощью **seekg()**, а затем с помощью **tellg()** получим значение конечной позиции и сохраним ее в переменной **len**.

```
#include <fstream.h>

int main()
{
    // создание входного потока и открытие файла
    ifstream fin("C:\\borlandc\\students\\textin.txt",
                 ios::in | ios::nocreate);
    if(!fin) { // проверка результата открытия
        cout<<"Невозможно открыть входной файл"<<endl;
        return 1;
    }
    // неформатированное чтение текстового файла
    fin.seekg(0,ios::end);          // указатель в конец файла
    int len = fin.tellg();          // запомнить длину файла
    char *buf = new char [len+1]; // выделить память под буфер
    fin.seekg(0,ios::beg);          // указатель в начало файла
    cout<<"Чтение данных из файла."<<endl; // считать len символов
    fin.read(buf,len);                     // из файла в буфер buf
    buf[len] = '\0';                // поместить в буфер нуль-литеру

    // создание выходного потока и открытие файла
    ofstream fout("C:\\borlandc\\students\\textout.txt",
                  ios::out);
     if(!fout) { // проверка результата открытия
        cout<<"Невозможно открыть выходной файл"<<endl;
        return 1;
    }
    int n=0,    // индекс символа начала предложения
        i=0,    // индекс символа конца предложения
        j=0;    // текущий индекс символа вопросительного
                // предложения
    cout<<"Запись данных в файл."<<endl;
    while(buf[i]) { // просмотр символов в буфере
        if( buf[i] == '?') {    // Если i-ый символ - вопрос,
            for(j=n;j<=i;j++)   // то вывод предложения в поток
                fout<<buf[j];   // и обновление индекса
            n=i+1;              // начала предложения.
        }
```

```
        if (buf[i]=='.'||buf[i]=='!') // Если предложение
        {                             // не вопросительное,
            n=i+1;                    // то обновление только
        }                             // индекса n.
        i++;
    }
    fin.close();     // закрытие входного файла
    fout.close();    // закрытие выходного файла
    cout<<"Программа завершена."<<endl;
    delete[] buf; return 0;
}
```

В заключение отметим, что файл с тестовым примером должен содержать предложения различной длины — от нескольких символов до нескольких строк.

Программа, написанная с использованием функций ввода-вывода языка *C*, а не классов языка *C++*, часто является более быстродействующей, но менее защищенной от ошибок ввода-вывода данных.

3.4. Порядок выполнения работы

Вариант задания выбирается по указанию преподавателя. Перед выполнением работы необходимо ознакомится с разделами 3.1 — 3.3 данных методических указаний. Индивидуальные задания для выполнения работы см. в разделе 3.2.

3.4.1. Внимательно изучить индивидуальное задание.

3.4.2. Выбрать для обработки текст и подготовить тектовый файл. Рекомендуемый размер файла — не более 25 кБайт (не более 40000 символов). Для корректного чтения русскоязычных символов следует установить в файле кодировку для кириллической кодовой страницы OEM 866, например, при помощи программы *Notepad++*, установленной на компьютерах в лаборатории.

3.4.3. Выбрать типы данных для хранения исходных данных, промежуточных величин и результатов.

3.4.4. Разработать алгоритм решения задачи в соответствии с вариантом.

3.4.5. Составить программу в соответсвии с вариантом задания.

3.4.6. Выполнить тестирование и отладку программы.

3.4.7. Получить результаты работы программы. Рекомендуется скопировать в виде изображения сообщения, выводимые программой, для последующего включения в отчет.

3.4.8. Подготовить отчет по работе.

3.5. Оформление отчета

3.5.1. Сформулировать цель работы.

3.5.2. Привести постановку задачи и текст индивидуального задания.

3.5.3. Привести краткие теоретические сведения об используемых в программе средствах файлового ввода-вывода.

3.5.4. Изобразить схему алгоритма решения задачи. Записать краткие пояснения к алгоритму.

3.5.5. Привести текст программы. Программа обязательно должна содержать комментарии.

3.5.6. Составить таблицу, в которой указать имена всех переменных используемых в программе и их тип, а также кратко описать назначение переменных.

3.5.7. Привести результаты работы программы и провести анализ работы программы.

3.5.8. Включить в отчет текст из обрабатываемого файла.

3.5.9. Сделать выводы по работе.

3.6. Контрольные вопросы

3.6.1. Приведите примеры описания и инициализации строк в языке программирования *C*.

3.6.2. Как хранится строка языка программирования *C* в памяти персонального компьютера?

3.6.3. Поясните назначение и приведите формат записи функций **`gets`** и **`puts`**.

3.6.4. Как выделить динамическую память под строку в языках программирования *C*/*C* ++?

3.6.5. Как ввести строку с помощью функций **`getline`** и **`get`** объекта **`cin`**?

3.6.6. Для чего предназначены функции **`strcpy()`** и **`strncpy()`**?

3.6.7. Для чего предназначена функция **`strcmp()`**? Как ее вызвать? Какие результаты она передает в точку вызова?

3.6.8. Поясните, как открыть файл в языке программирования *C*? Какие режимы открытия файла имеются в языке *C*?

3.6.9. Опишите функции **`fgets`** и **`fputs`**. В чем их отличие от функций **`gets`** и **`puts`**?

3.6.10. Приведите примеры чтения из файла и записи в файл с помощью функций **fscanf**() и **fprintf**().

3.6.11. Приведите примеры вызова функций **`fread()`** и **`fwrite()`**. Объясните назначение аргументов этих функций.

3.6.12. Как выполняется изменение и чтение счетчика текущей позиции файла в языке программирования *C*?

3.6.13. Поясните, как открыть файл в языке программирования *C*++? Какие режимы открытия файла имеются в языке *C*++?

3.6.14. Объясните назначение функций **`seekg()`** и **`seekp()`**. Приведите примеры использования функций.

3.6.15. Приведите примеры вызова функций **`read()`** и **`write()`** языка программирования *C*++.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Методические указания к лабораторному практикуму по дисциплине «Вычислительная техника и программирование» для студентов дневной и заочной форм обучения направления 6.050901 — «Радиотехника» : в 4 ч. / СевНТУ ; сост. С.Н. Бердышев. — Севастополь : Изд-во СевНТУ, 2013. — Ч.1. — 60 с.

2. Методические указания по оформлению текстовых работ для студентов дневной и заочной форм обучения направления 6.050901 — «Радиотехника» / СевНТУ ; сост. В.Г. Слезкин. — Севастополь : Изд-во СевНТУ, 2010. — 20 с.

3. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. — Взамен ГОСТ 19.002-80, ГОСТ 19.003-80 ; Введ. 01.01.1992. — М. : Изд-во стандартов, 1991. — 28 с.

4. Методические указания к лабораторному практикуму по дисциплине «Вычислительная техника и программирование» для студентов дневной и заочной форм обучения направления 6.050901 — «Радиотехника» : в 4 ч. / СевНТУ ; сост. С.Н. Бердышев. — Севастополь : Изд-во СевНТУ, 2013. — Ч.2. — 48 с.

5. Бердышев С.Н. Учебное пособие по дисциплине «Вычислительная техника и программирование» для студентов дневной и заочной форм обучения направления 6.050901 — «Радиотехника» / СевНТУ ; сост. С.Н. Бердышев, Е.А. Редькина. — Севастополь : Изд-во СевНТУ, 2013. — 170 с.

6. Савочкин А.А. Учебное пособие по дисциплине «Информатика» для студентов заочной формы обучения специальности 7.090701 — Радиотехника «Программирование на языке СИ» / СевНТУ ; сост. А.А. Савочкин. — Севастополь : Изд-во СевНТУ, 2002. — 62 с.

7. Павловская Т.А. *C*/*C*++. Программирование на языке высокого уровня / Т.А. Павловская. — СПб.: Питер, 2010. — 461 с.

8. Павловская Т.А. *C*/*C*++. Структурное программирование / Т.А. Павловская, Ю.А. Щупак. — СПб.: Питер, 2003. — 240 с.

9. Глушаков СВ. Язык программирования *C*++ / С.В. Глушаков, А.В. Коваль, С.В. Смирнов. — Харьков: Фолио, 2002. — 500 с.

10. Дейтел Х. Как программировать на *C*++: [пер. с англ.] / Х. Дейтел, П. Дейтел. — М.: Изд-во БИНОМ, 2000. — 1024 с.

11. Ковалюк Т.В. Основи програмування / Т.В. Ковалюк. — К.: Видавнича група BHV, 2005. — 384 с.

12. Франка П. *C*++: учебный курс / П. Франка. — СПб. : Питер, 2006. — 522 с.

13. Громов Ю.Ю. Программирование на языке СИ: учеб. пособие / Ю.Ю. Громов, С.И. Татаренко. — Тамбов: Изд-во ТГТУ, 1995. — 169 с.

14. Вирченко Н.А. Графики функций: справочник / Н.А. Вирченко, И.И. Ляшко, К.И. Швецов. — Киев : Наук. думка, 1979. — 320 с.

15. Руководство пользователя *MathCAD* / Корпорация «*Parametric Technology Corporation*». — Нидем (США) : *PTC*, 2011. — 190 с.

Заказ № _____ от «____»_____ 20___ г. Тираж _____ экз.

Изд-во СевНТУ