Министерство образования и науки Украины
Севастопольский национальный технический университет

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторному практикуму
по дисциплине
«Информатика и вычислительная техника»

для студентов дневной формы обучения
направления 6.050901 — «Радиотехника»

Часть 1

«Программирование на языке *C*/*C*++»

Севастополь
2008

УДК 004.42+621.37 (076.5)

Методические указания к лабораторному практикуму по дисциплине «Информатика и вычислительная техника» для студентов дневной формы обучения направления 6.050901 — «Радиотехника»: в 2 ч. / Сост. С.Н. Бердышев. — Севастополь: Изд-во СевНТУ, 2008. — Ч.1: Программирование на языке *C*/*C*++. — 116 с.

Целью методических указаний является оказание помощи студентам дневной формы обучения направления «Радиотехника» при выполнении лабораторных работ по дисциплине «Информатика и вычислительная техника».

Рассмотрены и утверждены на заседании кафедры радиотехники СевНТУ (протокол № 2 от 26 сентября 2007 г.).

Допущено учебно-методическим центром СевНТУ в качестве методических указаний.

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

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

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Дисциплина «Информатика и вычислительная техника» изучается студентами дневной формы обучения направления 6.050901 — «Радиотехника» в первом и втором семестрах.

При изучении первой части дисциплины, изучаемой в первом семестре, по учебному плану студенты выполняют и защищают цикл из шести лабораторных работ, которые направлены на получение практических навыков программирования на языке *C*/*C*++ на основе программной реализации 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 пяти лабораторных работ, направленных на углубление и закрепление знаний, полученных при изучении первой части. При выполнении второй части лабораторного практикума, выполняемого во втором семестре, рассматриваются вопросы программной реализации алгоритмов численной математики, применяемые для расчета параметров радиоэлектронных схем: нахождение корней нелинейных уравнений, решение систем линейных уравнений, интерполяция, дифференцирование и интегрирование. Итоговой формой контроля во втором семестре является экзамен.

После изучения дисциплины «Информатика и вычислительная техника» 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 … № 5 отводится по два часа, а работы № 6 — четыре часа.

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 её;

— 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, студент должен:

— представить результаты выполнения первого этапа преподавателю;

— отладить написанную программу;

— разработать и выполнить тестовый пример;

— защитить отчет по лабораторной работе.

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

— лабораторная работа № 1 на 2…3 неделях осеннего семестра;

— лабораторная работа № 2 на 4…5 неделях осеннего семестра;

— лабораторная работа № 3 на 6…7 неделях осеннего семестра;

— лабораторная работа № 4 на 8…9 неделях осеннего семестра;

— лабораторная работа № 5 на 10…11 неделях осеннего семестра;

— лабораторная работа № 6 на 12…15 неделях осеннего семестра.

Отчет по лабораторной работе оформляется каждым студентом индивидуально на стандартных листах формата А4 с использованием персонального компьютера (ПК) и печатается с помощью принтера. По краям листа должны быть оставлены поля: левое — 25 мм; верхнее и нижнее — 20 мм; правое — 15 мм.

В отчет должны включаться: титульный лист, цель работы и выполненные задания. 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 приложении В.

Отчет следует оформлять в соответствии с методическими указаниями по оформлению текстовых работ [1].

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 [2]. Правила оформления схем алгоритмов изложены в приложении А.

При выполнении лабораторных работ и подготовке к защите рекомендуется использовать учебную и справочную литературу [3 — 15].

Тексты разработанных программ и результаты программных расчетов следует приводить только в распечатанном виде.

1. ЛАБОРАТОРНАЯ РАБОТА №1
ИЗУЧЕНИЕ ИНТЕГРИРОВАННОЙ СРЕДЫ РАЗРАБОТКИ *BORLAND C++*

1.1. Цель работы

Приобретение практических навыков работы в интегрированной среде разработки *Borland C++*.

1.2. Теоретические сведения

1.2.1. Общее описание интегрированной среды разработки *Borland C++*

Интегрированная среда разработки (ИСР) (*Integrated Development Environment — IDE*) объединяет текстовый редактор, компилятор, отладчик и справочную систему. Все эти составные части необходимы для успешной работы по созданию исходного текста программы, его компиляции, запуску и поиску возможных ошибок на этапе выполнения.

Для того чтобы выполнить программу на *C*/*C++*, необходимо пройти следующие этапы: редактирование, предварительную (препроцессорную) обработку, компиляцию и компоновку.

Редактирование файла осуществляется с помощью редактора программ. Программист набирает с помощью этого редактора программу на *C*/*C++* и, если это необходимо, вносит в нее исправления. Программа запоминается на диске в файле с расширением *cpp*.

Перед началом этапа компиляции выполняется программа предварительной (препроцессорной) обработки. Эта программа подчиняется специальным командам, называемым директивами препроцессора, которые указывают, что в программе перед ее компиляцией нужно выполнить определенные преобразования. Обычно эти преобразования состоят во включении других текстовых файлов в файл, подлежащий компиляции, и выполнении различных текстовых замен. Препроцессорная обработка инициируется компилятором перед тем, как программа преобразуется в машинный код.

Компилятор переводит программу в машинный код (называемый также объектным кодом). Объектный код сохраняется в файле с расширением *obj*.

Программы на *C*/*C++* обычно содержат ссылки на функции, определенные где-либо вне самой программы, например, в стандартных библиотеках. Объектный код, созданный компилятором, обычно содержит пустые области из-за этих отсутствующих частей. Компоновщик связывает объектный код с кодами отсутствующих функций, чтобы создать исполняемый загрузочный модуль (файл с расширением *exe*).

1.2.2. Запуск ИСР *Borland C++*

Для запуска ИСР *Borland C++* необходимо запустить на исполнение файл *bc.exe*, находящийся в папке *BIN*. После запуска появляется рабочий экран *Borland C++*, содержащий четыре основных части:

— строка меню,

— окно редактирования,

— окно сообщений,

— строка состояния.

Строка меню предоставляет доступ к командам ИСР. Для активизации строки меню следует нажать *F*10.

Окно редактирования предназначено для ввода и редактирования текста исходного файла программы.

Ниже представлено краткое описание каждого элемента строки меню:

? — системное меню;

File — операции с файлами, выход из системы;

Edit — редактирование текста в активном окне;

Search — поиск фрагментов текста, местоположения ошибок;

Run — компиляция, компоновка и запуск программы на выполнение;

Compile — компиляция программы;

Debug — средства отладки программ;

Project — организация проектов (многофайловых программ);

Options — управление параметрами компиляции, компоновки и среды *Borland C++*;

Window — управление окнами ИСР;

Help — обращение к системе оперативной подсказки.

1.2.3. Работа с меню

Для управления файлами предназначено меню *File*. Данное меню содержит следующие основные команды:

New — открыть окно для нового файла;

Open... — открыть существующий файл;

Save — сохранить файл с прежним именем;

Save as... — сохранить файл с новым именем;

Save all — сохранить файлы всех окон;

Change dir... — изменить текущую директорию;

Quit — выйти из ИСР *Borland C++*.

Редактирование файлов выполняется с использованием меню *Edit*:

Undo — отменить действие предыдущей команды;

Redo — повторить действие последней команды;

Cut — вырезать блок текста и поместить его в буфер (*clipboard*);

Copy — копировать блок текста и поместить его в буфер;

Paste — вставить блок текста из буфера;

Clear — удалить блок текста, не занося его в буфер;

Show clipboard — показать содержимое буфера.

Переключение раскладки клавиатуры на русский язык производится однократным нажатием правой клавиши *Ctrl*, на символы псевдографики — правой клавишей *Alt*, обратное переключение на английскую раскладку клавиатуры — правой клавишей *Ctrl*.

Поиск и замена текста осуществляется при помощи меню *Search*:

Find... — найти текст;

Replace... — найти текст и заменить его новым текстом;

Search again — повторить команду *Find* или *Replace*.

Выполнение программы запускается из меню *Run*:

Run — компиляция, компоновка и выполнение;

Program reset — прервать трассировку программы;

Go to cursor — выполнить программы до инструкции, перед которой установлен курсор;

Trace into — построчное выполнение программы с заходом в тело функций и построчным выполнением инструкций внутри функций;

Step over — построчное выполнение программы; если встречается вызов функции, то функция выполняется как одна инструкция (без захода внутрь тела функции);

Arguments... — формирование аргументов командной строки.

Компиляция программы выполняется командами из меню *Compile*:

Compile — компилировать программу из активного окна;

Information... — выдать информацию о программе и системе.

Отладка программ осуществляется средствами меню *Debug*:

Evaluate/*modify…* — вычислить/модифицировать значение выражения или переменной в процессе отладки;

Call stack... — вызвать стек активных функций;

Watches — открыть окно просмотра текущих значений переменных программы;

Toggle breakpoint — добавить/удалить точку прерывания программы;

Breakpoints... — список точек прерывания.

Управление ИСР *Borland C++* производится в меню *Options*:

Compiler > — установка параметров компилятора;

Linker > — установка параметров компоновщика;

Debugger... — установка параметров отладчика;

Directories... — установка путей для каталогов (папок) ИСР;

Environment > — установка параметров ИСР;

Save... — сохранение настроек ИСР.

Особое внимание следует обратить на команду *Options* > *Directories*, поскольку с помощью нее задаются пути к заголовочным файлам (*Include Directories*), библиотекам (*Library Directories*), а также каталогу, в который будут помещаться файлы с расширением *obj* и *exe* (*Output Directory*).

Для использования отладчика необходимо в диалоговом окне *Debugger* (*Options* > *Debugger...*) установить переключатель *Source Debugging* в положение *On*.

Управление окнами выполняется командами из меню *Window*:

Size/*Move* — изменить размер окна или переместить окно;

Zoom — распахнуть или свернуть окно;

Cascade — разместить окна каскадом;

Tile — разместить окна мозаикой;

Next — активизировать следующее окно;
Close — закрыть активное окно;
Close all — закрыть все окна;
List all... — показать список всех окон.
Система помощи расположена в меню *Help*:
Contents — содержание (перечень тем системы помощи);
Index — тематический указатель;
Topic search — помощь по заданной теме;
Previous topic — помощь по предыдущей теме;
About — информация о версии системы.

1.3. Порядок выполнения работы

1.3.1. Изучить алгоритм решения задачи определения траектории полета снаряда, запущенного под углом к горизонту.

Уравнения, описывающие движение снаряда, т. е. зависимость его координат x и y от времени t, прошедшего с момента выстрела, выглядят следующим образом:

$$x(t) = x_0 + V_{0x}t; \tag{1.1}$$

$$y(t) = y_0 + V_{0y}t - \frac{gt^2}{2}; \tag{1.2}$$

$$V_{0x} = V_0 \cos\alpha; \tag{1.3}$$

$$V_{0y} = V_0 \sin\alpha. \tag{1.4}$$

Здесь введены следующие обозначения:

x_0, y_0 — координаты начальной точки траектории полета снаряда, м;

V_0 — начальная скорость снаряда, м/с;

α — угол наклона начального участка траектории полета снаряда, рад;

V_{0x}, V_{0y} — проекции вектора начальной скорости V_0 на координатные оси x и y, соответственно, м/с;

g — ускорение свободного падения, $g = 9{,}81$ м/с^2.

Будем считать, что необходимо определить положение (координаты) снаряда в моменты времени 0 с, Δt, $2\Delta t$, ..., $(n-1)\Delta t$, причем

$$\Delta t = \frac{T}{n-1}, \tag{1.5}$$

где T — время полета снаряда, с,

n — число точек, в которых рассчитываются координаты траектории полета снаряда.

Величину T можно определить по формуле

$$T = \frac{2V_{0y}}{g}. \tag{1.6}$$

Исходя из записанных уравнений алгоритм решения задачи можно представить в следующем виде:

1) Ввести начальные значения скорости снаряда V_0, угла наклона начального участка траектории α и число точек n.
2) Вычислить значение V_{0x} по формуле (1.3).
3) Вычислить значение V_{0y} по формуле (1.4).
4) Присвоить $g = 9{,}81$ м/с^2.
5) Вычислить T по формуле (1.6).
6) Вычислить Δt по формуле (1.5).
7) Присвоить $i = 1$.
8) Присвоить $t = 0$.
9) Вычислить $x(t)$ по формуле (1.1).
10) Вычислить $y(t)$ по формуле (1.2).
11) Вывести координаты снаряда.
12) Присвоить $t = t + \Delta t$.
13) Присвоить $i = i + 1$.
14) Если $i < n$, то перейти к шагу 9, иначе остановить выполнение программы.

Также алгоритм может быть представлен в виде структурной схемы. Пример схемы алгоритма изображен на рис. 1.1.

1.3.2. Написать программу, вычисляющую координаты траектории полета снаряда. Ниже представлен текст такой программы на языке *C++*.

```
#include <conio.h>     // подключение библиотеки управления
                       // вводом-выводом средствами консоли MSDOS
#include <iomanip.h>   // подключение библиотеки средств
                       // манипулирования потоками
#include <iostream.h>  // подключение библиотеки стандартных
                       // объектов и операций с потоками
                       // ввода-вывода
#include <math.h>      // подключение библиотеки математических
                       // функций

const float pi=3.14;
const float g=9.81; // ускорение свободного падения

main()
// определение траектории полета снаряда,
// запущенного под углом к горизонту
{
    float x0, y0,    // координаты начальной точки траектории
    x, y,            // текущие координаты снаряда
    v0,              // начальная скорость снаряда
    v0_x, v0_y,      // проекции вектора начальной скорости снаряда
    alpha,           // угол запуска снаряда
    t,               // текущее время
    dt,              // шаг дискретизации по времени
    T,               // время полета снаряда
```

```
    n,              // число точек траектории полета снаряда
    i;              // счетчик цикла

    clrscr ();      // очистка экрана

    // ввод значений v0, alpha, n
    cout<<"Введите начальную скорость: ", cin>>v0 ;
    cout<<"Введите угол запуска снаряда (в градусах): ";
    cin>>alpha;
    alpha*=pi/180; // перевод угла запуска из градусов в радианы
    cout<<"Введите число точек траектории: ", cin>>n;
    v0_x=v0*cos(alpha), v0_y=v0*sin(alpha); // проекции v0
    x0=y0=0; // начальная точка траектории полета снаряда
    T=2*v0_y/g; // время полета снаряда
    dt=T/(n-1); // шаг дискретизации по времени
    cout<<"Координаты траектории полета снаряда: "<<endl;
    cout.precision(2), cout.setf(ios::showpoint);
    cout.setf(ios::left,ios::adjustfield);
    cout.setf(ios::fixed,ios::floatfield);
    for(i=0;i<n;i++){
       t=i*dt; // текущее время
       x=x0+v0_x*t; // x
       y=y0+v0_y*t-g*t*t/2; // y
       cout<<setw(10)<<x<<setw(10)<<y<<endl;
    }

    cout<<"Нажмите любую клавишу... ";
    getch();
    return 0;

}
```

1.3.3. Выполнить компиляцию программы (*Compile* > *Compile* или *Alt*+*F*9).

1.3.4. Исправить синтаксические ошибки, если такие имеются, и повторить компиляцию программы.

1.3.5. Запустить программу (*Run* > *Run* или *Ctrl*+*F*9) и проанализировать результаты ее работы.

1.3.6. Добавить в окно просмотра *Watch* переменные **t**, **T**, **n**, **i**. Для этого следует использовать команду *Debug* > *Watches* > *Add watch*… или *Ctrl*+*F*7.

1.3.7. Выполнить программу в пошаговом режиме (*Run* > *Step over* или *F*8). При переключении программы на экран пользователя ввести произвольные значения начальной скорости полета, угла наклона начального участка траектории и количество рассчитываемых точек траектории полета снаряда (рекомендуется 7…10 точек).

1.3.8. После достижения инструкции **for** (строка программы **for(i=0;i<n;i++){**) проанализировать значения переменных **t**, **T**, **n**, **i**, представленных в окне *Watch*.

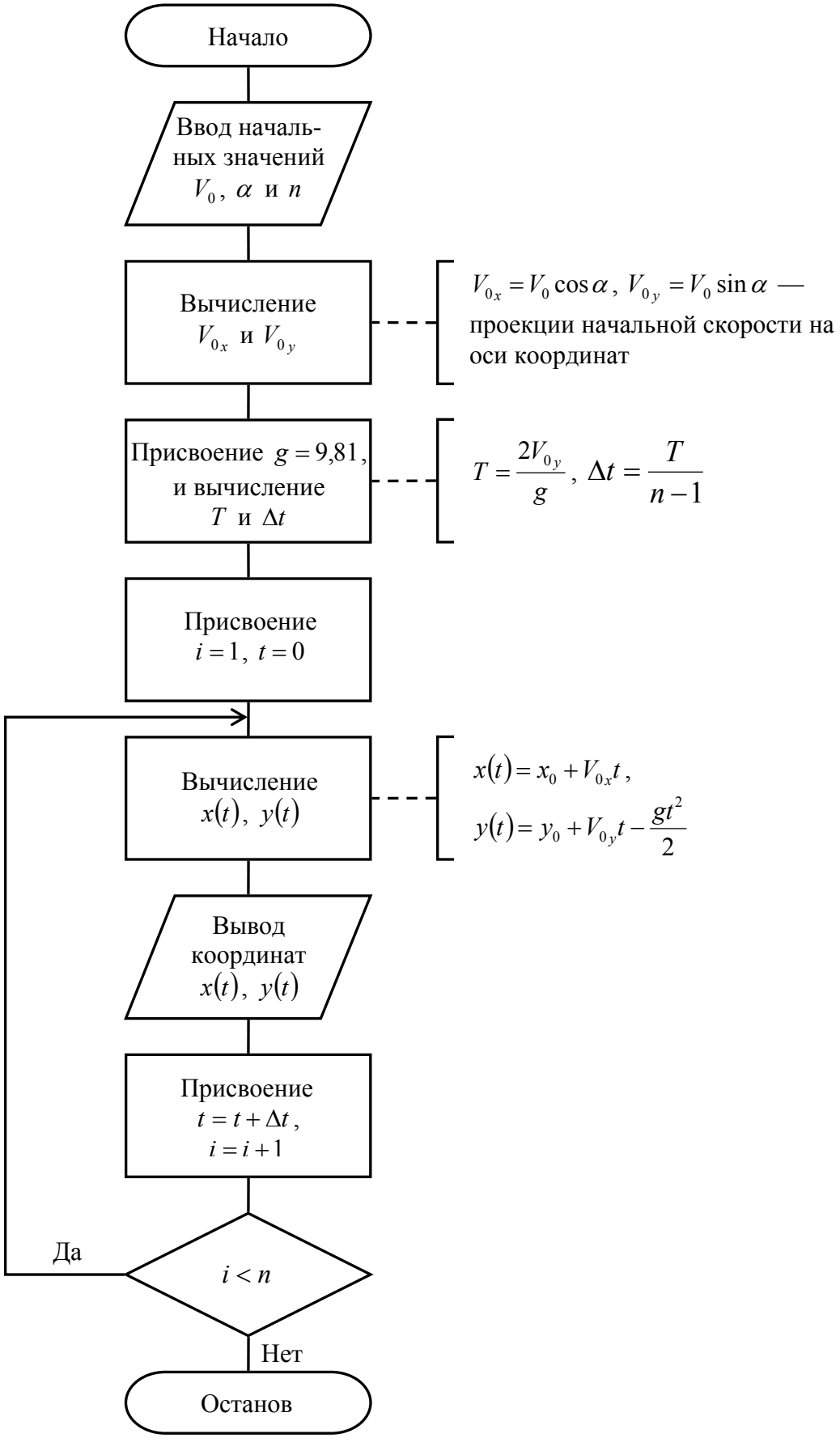


Рис. 1.1 — Схема алгоритма решения задачи

1.3.9. Зафиксировать те значения переменных **i** и **t**, при которых произойдет выход из цикла. Сравнить эти значения со значениями переменных **n** и **T**, соответственно. С помощью команды *Debug* > *Evaluate*/*modify*… (*Ctrl*+*F*4) определить значения переменных **x** и **y**.

1.3.10. Установить две точки прерывания: одну перед инструкцией

```
x0=y0=0; // начальная точка траектории полета снаряда
```

а другую — перед инструкцией

```
for(i=0;i<n;i++){
```

Для этого следует использовать команду *Debug* > *Toggle breakpoint* (*Ctrl*+*F*8). Выполнить программу до первой точки прерывания (*Run* > *Run* или *Ctrl*+*F*9). Определить значения переменных программы **x0**, **y0**, **T**, **dt** с помощью команды *Debug* > *Evaluate*/*modify*… (*Ctrl*+*F*4). Выполнить программу до второй точки прерывания (*Ctrl*+*F*9). Снова определить значения переменных **x0**, **y0**, **T**, **dt** (*Ctrl*+*F*4).

1.4. Оформление отчета

1.4.1. Сформулировать цель работы.

1.4.2. Описать задачу определения траектории полета снаряда, запущенного под углом к горизонту.

1.4.3. Изобразить алгоритм решения задачи (правила оформления схем алгоритмов см. в приложении А).

1.4.4. Привести текст программы.

1.4.5. Привести таблицу, содержащую имена используемых в программе переменных, тип и назначение переменных.

1.4.6. Описать работу с отладчиком. Провести анализ значений отслеживаемых переменных.

1.4.7. Привести результаты расчета программы.

1.4.8. В выводах следует отразить следующее:

1) Оценить достоинства и недостатки ИСР;
2) По полученным результатам оценить достоверность координат траектории полета снаряда, рассчитанных программой;
3) Провести анализ значений переменных **i** и **t**, при которых произошел выход из цикла.

1.5. Контрольные вопросы

1.5.1. Что такое препроцессорная обработка?

1.5.2. В чем состоит компиляция программы?

1.5.3. Что такое компоновка?

1.5.4. Охарактеризуйте меню *File*.

1.5.5. Охарактеризуйте меню *Edit*.

1.5.6. Охарактеризуйте меню *Search*.

1.5.7. Охарактеризуйте меню *Run*.

1.5.8. Охарактеризуйте меню *Compile*.

1.5.9. Охарактеризуйте меню *Debug*.

1.5.10. Охарактеризуйте меню *Options*.

1.5.11. Охарактеризуйте меню *Window*.

1.5.12. Охарактеризуйте меню *Help*.

1.5.13. Как добавить переменную в окно просмотра *Watch*?

1.5.14. Каким образом можно определить и модифицировать значение переменной?

1.5.15. Как добавить или удалить точку прерывания?

2. ЛАБОРАТОРНАЯ РАБОТА №2

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМОВ ЛИНЕЙНОЙ И РАЗВЕТВЛЯЮЩЕЙСЯ СТРУКТУРЫ

2.1. Цель работы

Изучение структуры *C*-программы.

Формирование навыков программирования алгоритмов линейной и разветвляющейся структуры на языке *C*.

Исследование особенностей ввода-вывода значений стандартных типов в языках *C*/*C*++.

2.2. Варианты заданий

Составить схему алгоритма и написать на языке *C* программу вычисления функции $y = f(x)$. Варианты функций выбирать по указанию преподавателя. Значения параметров a, b и аргумента x выбираются произвольно и вводятся с клавиатуры. Результаты вычислений выводятся на дисплей в формате с плавающей точкой.

Вариант 1

$$y = \begin{cases} \sqrt{1{,}57 - x^3 \sin^2 x} + 4{,}1\mathrm{e}^{2x}, & \text{если } x \le a; \\ \dfrac{1+x^2}{2-7\sin x} + \mathrm{e}^{\mathrm{sh}\,x}, & \text{если } a < x < b; \\ (\arcsin x + \arccos x + \ln x)^{\mathrm{tg}\,x}, & \text{если } x \ge b. \end{cases}$$

Вариант 2

$$y = \begin{cases} (\cos^3 x + 0{,}9\,\mathrm{tg}\,x)^{2,3}, & \text{если } x \le a; \\ \dfrac{\mathrm{e}^x}{\sqrt{\mathrm{sh}\,x} + 8{,}9x + 9}, & \text{если } a < x < b; \\ \left|\ln x + \cos x^{2,1} - 1{,}2x\right| x^{4,6}, & \text{если } x \ge b. \end{cases}$$

Вариант 3

$$y = \begin{cases} \sqrt[3]{\mathrm{e}^{x - \frac{1}{\mathrm{arctg}\,x}}} + \mathrm{tg}^2 x + 6, & \text{если } x \le a; \\ (\arcsin x + \mathrm{sh}\,x)^{\cos x + x^2 + \mathrm{e}}, & \text{если } a < x < b; \\ \left| x^{\frac{1,3}{x}} - \lg(1+x) \right|^{1,3+x^2} + x^5, & \text{если } x \ge b. \end{cases}$$

Вариант 4

$$y = \begin{cases} \sqrt{\sqrt[3]{x+1{,}1} + \sqrt[5]{x^3 - 4{,}4}}, & \text{если } x \le a; \\ \mathrm{e}^{\cos 3x - 1} + (x-3)^{\mathrm{ch}\,3x+1}, & \text{если } a < x < b; \\ \dfrac{\mathrm{tg}^2 x}{0{,}5 + \mathrm{sh}^{\pi} x + \ln \pi x}, & \text{если } x \ge b. \end{cases}$$

Вариант 5

$$y = \begin{cases} \ln(x^2 + x + \mathrm{e}) + 2\cos(x-1), & \text{если } x \le a; \\ 4{,}1\sin\left(x + \dfrac{\pi}{3}\right) + 1{,}3^{-0,2x-1}, & \text{если } a < x < b; \\ 8x^4 + 9x^3 + 7x^2 + 3x + \dfrac{1}{\mathrm{ch}\,x}, & \text{если } x \ge b. \end{cases}$$

Вариант 6

$$y = \begin{cases} 3\mathrm{e}^{-2\cos x} + 2\log_3(1 + x + x^2), & \text{если } x \le a; \\ \dfrac{1+x^3}{1{,}2 - \sqrt{\mathrm{ch}\,x}}, & \text{если } a < x < b; \\ \sqrt[5]{x^3 + \sqrt{1{,}9 + x^{1,7} + \sqrt{\mathrm{tg}^5 x}}}, & \text{если } x \ge b. \end{cases}$$

Вариант 7

$$y=\begin{cases}\lg\left(\sqrt[3]{x^2}+\sqrt{x}+\sin x+\cos x\right), & \text{если } x\le a;\\ \left(\operatorname{tg}^2 x+1{,}3\,\mathrm{e}^{\operatorname{ch}x+\operatorname{th}x}\right)^{x^2}, & \text{если } a<x<b;\\ \left|\sin x-0{,}11\right|\arccos x+x^{\arcsin x}, & \text{если } x\ge b.\end{cases}$$

Вариант 8

$$y=\begin{cases}\sqrt{3{,}7^{x-\operatorname{arctg}x}}+\operatorname{tg}^{-2}x, & \text{если } x\le a;\\ \ln(x+5{,}7)+3{,}8\dfrac{\mathrm{e}^{0{,}1x}}{\sin x}, & \text{если } a<x<b;\\ \left|\cos x-0{,}5\right|+\arccos 5x, & \text{если } x\ge b.\end{cases}$$

Вариант 9

$$y=\begin{cases}\left|\cos x\right|^{1+3\operatorname{tg}^2 x}+x^{1{,}31}, & \text{если } x\le a;\\ 8^{x+1}\sqrt{\operatorname{th}x+\sqrt[7]{\left|1-x\right|}}, & \text{если } a<x<b;\\ \lg\left(\sqrt[3]{x}+\sqrt{x}+x+2\right), & \text{если } x\ge b.\end{cases}$$

Вариант 10

$$y=\begin{cases}\left[1+\operatorname{sh}^2\left(x^2-x+1\right)\right]x^{-2}, & \text{если } x\le a;\\ 4{,}3\ln x+\dfrac{3{,}5x}{\mathrm{e}^x}, & \text{если } a<x<b;\\ \left(1+2\operatorname{tg}^{-4}x+3\operatorname{tg}^{-5}x\right)^{-\sqrt{x}}, & \text{если } x\ge b.\end{cases}$$

Вариант 11

$$y=\begin{cases}\left(1+\operatorname{ch}x+\operatorname{th}^2 x\right)^{\sin x}, & \text{если } x\le a;\\ 7x^{\pi x+6}+\operatorname{arctg}(\lg x), & \text{если } a<x<b;\\ \left|\arcsin x\right|+\sqrt{\left|\mathrm{e}^{4\sin^2 x}-x\right|}, & \text{если } x\ge b.\end{cases}$$

Вариант 12

$$y=\begin{cases}\log_2\left(2\pi+\operatorname{tg}x\right)+\arcsin 2x, & \text{если } x\le a;\\ 6{,}3\,\mathrm{e}^{2x+7{,}3}\sin\left(5x+\dfrac{\pi}{6}\right)+3, & \text{если } a<x<b;\\ \left|\sin 2x-2{,}5\right|\operatorname{ch}^2 3x+x^{-5{,}4}, & \text{если } x\ge b.\end{cases}$$

Вариант 13

$$y=\begin{cases}\left(\operatorname{ch}x+\operatorname{sh}x+x^{0{,}5}\right)^{1{,}1x^{2{,}2}}, & \text{если } x\le a;\\ \left|x^{\frac{1{,}3}{x}}-\sqrt[3]{\dfrac{2{,}6}{1{,}1x}}+\mathrm{e}^{0{,}221x}\right|, & \text{если } a<x<b;\\ \lg\sqrt{\mathrm{e}^{x+25}}+\ln\sqrt[3]{100x}+\log_4 x, & \text{если } x\ge b.\end{cases}$$

Вариант 14

$$y=\begin{cases}\ln\left(x-\operatorname{sh}x\right)+\arccos 5{,}1x^3, & \text{если } x\le a;\\ \dfrac{1+x^2}{\left(1-x^2\right)\left(\operatorname{ch}^2 x^3+x^{2x-9}\right)}, & \text{если } a<x<b;\\ \sin^2 2{,}45x+\pi^{x^{\mathrm{e}}+x+1}, & \text{если } x\ge b.\end{cases}$$

Вариант 15

$$y=\begin{cases}\mathrm{e}^{-3\sin 3x}+\log_5\left(\operatorname{arctg}x\right), & \text{если } x\le a;\\ \dfrac{\sqrt{x+\operatorname{th}x+x^5}}{\cos x^2}\operatorname{ch}x\operatorname{sh}x, & \text{если } a<x<b;\\ x^{3,7}+\sin|x|+|\cos x|, & \text{если } x\ge b.\end{cases}$$

Вариант 16

$$y=\begin{cases}\left|\dfrac{1}{x^{4,3}}-\sqrt[3]{\operatorname{arctg}x}+\sin x^3\right|, & \text{если } x\le a;\\ \log_7\left(\mathrm{e}^{x^2}+\cos x\right)+x\lg x, & \text{если } a<x<b;\\ \left(x^3+2x^2-1\right)\operatorname{sh}x+5^{\sqrt[3]{x}+1}, & \text{если } x\ge b.\end{cases}$$

Вариант 17

$$y=\begin{cases}\left(\cos x+\sin x\right)^{\arccos 2x}, & \text{если } x\le a;\\ 11{,}3+3{,}3\cos\left(3{,}6x-\dfrac{\pi}{4}\right), & \text{если } a<x<b;\\ 2{,}8\lg 3x+8x^{3,1}+4{,}3x^{2,7}+1{,}1, & \text{если } x\ge b.\end{cases}$$

Вариант 18

$$y=\begin{cases}\sqrt{\sqrt{\pi}-x^{3,7}\cos x^{2,1}}+\ln x, & \text{если } x\le a;\\ \mathrm{e}^{\pi x}\sin\left(2x-\dfrac{\pi}{5}\right), & \text{если } a<x<b;\\ \left|x^{5,1}\operatorname{ch}x-1{,}6\right|\operatorname{arctg}x^{2,6}, & \text{если } x\ge b.\end{cases}$$

Вариант 19

$$y=\begin{cases}\ln(\sin 9{,}5x)+\cos^2 x^{1,2}, & \text{если } x\le a;\\ 3^{x^{4,4}}+\operatorname{ch} x^{-3,33}+\lg x, & \text{если } a<x<b;\\ \sqrt{\operatorname{sh} x^2\operatorname{arctg} x^{3,94}+9\operatorname{tg}^{1,1} x}, & \text{если } x\ge b.\end{cases}$$

Вариант 20

$$y=\begin{cases}\ln(1{,}2x+2{,}3)+\mathrm{e}^{3x}\operatorname{sh} x, & \text{если } x\le a;\\ 5^{x^2+1}\sqrt{x^{3,3}+\sqrt{|1-x|}}, & \text{если } a<x<b;\\ \left(1+\operatorname{tg}^2 x\right)^{\operatorname{arctg} x+\lg x+1}, & \text{если } x\ge b.\end{cases}$$

Вариант 21

$$y=\begin{cases}2{,}57\,\mathrm{e}^{\operatorname{sh}\frac{\pi}{x}-11}+\operatorname{ch}\sqrt{\dfrac{\pi}{x}}+\dfrac{\pi}{3x}, & \text{если } x\le a;\\ \ln(2{,}5\log_7 x+3{,}9)-\sin x, & \text{если } a<x<b;\\ \left|x^{1,2}-\sqrt[5]{1-x^3}\right|+\dfrac{1}{\operatorname{arctg} x^{2,11}}, & \text{если } x\ge b.\end{cases}$$

Вариант 22

$$y=\begin{cases}\sqrt[4]{3x^2+\sqrt[3]{1+\sin^2 x^7+\sqrt{\mathrm{e}^{2,1x}}}}, & \text{если } x\le a;\\ \left(1+\operatorname{sh} x+\lg^2 x\right)^{x^2-x+9}, & \text{если } a<x<b;\\ \left|x-\operatorname{arctg} x+\operatorname{ctg}^{2\mathrm{e}^x+14x+3} x\right|, & \text{если } x\ge b.\end{cases}$$

Вариант 23

$$y=\begin{cases}9{,}6x^3-4{,}8x^2+2{,}4x-1, & \text{если } x\le a;\\ \sqrt{\operatorname{ch}^2\left(\operatorname{arctg} x^5\right)+1{,}32}, & \text{если } a<x<b;\\ \log_6\left(\sin x+\cos x+\operatorname{tg}^{2\mathrm{e}} 2\pi^x\right), & \text{если } x\ge b.\end{cases}$$

Вариант 24

$$y=\begin{cases}1-7x^2-\sin\left(8x+\dfrac{\pi}{6}\right), & \text{если } x\le a;\\ \sqrt{\cos^3 x^2+\operatorname{arctg}^2\mathrm{e}^{3x+10}}, & \text{если } a<x<b;\\ \ln\left(\arcsin\dfrac{3}{x}+\arccos\dfrac{x}{3}+x^{-3,1}\right), & \text{если } x\ge b.\end{cases}$$

Вариант 25

$$y=\begin{cases}\arccos^2 x+\operatorname{arctg} x^2-x^4+1, & \text{если } x\le a;\\ \mathrm{e}^{\pi x}+\pi^{\mathrm{e}x}+x^{\mathrm{e}\pi}\cos x^{x-1}, & \text{если } a<x<b;\\ \lg\left(\sqrt[5]{x^{2,3}}+\sqrt{|3{,}3-x|}+1{,}22\right), & \text{если } x\ge b.\end{cases}$$

Вариант 26

$$y=\begin{cases}\sqrt{1{,}414-x^2\sin^3 x^2}+\lg^{1,21} x, & \text{если } x\le a;\\ \operatorname{tg}^2 x+\operatorname{tg} x+4^x-2, & \text{если } a<x<b;\\ \pi-2x-\operatorname{ch} x+\operatorname{sh} x-x^3, & \text{если } x\ge b.\end{cases}$$

Вариант 27

$$y=\begin{cases}\dfrac{1+\arcsin x}{1-\arccos x}, & \text{если } x\le a;\\ \sqrt{\operatorname{sh}^2 x+\operatorname{th} x^2+\cos x^{2,8}}, & \text{если } a<x<b;\\ \dfrac{\mathrm{e}^x-\mathrm{e}^{-x}}{\mathrm{e}^{x-2,11}}\ln\sin x^2, & \text{если } x\ge b.\end{cases}$$

Вариант 28

$$y=\begin{cases}\dfrac{\mathrm{e}^{2x^2-3x+5}}{\lg x+3\sin x-1{,}54}, & \text{если } x\le a;\\ \left|\operatorname{ch}^{2,1} x-1{,}1\right|\arcsin^2 x^{1,1}, & \text{если } a<x<b;\\ \left(\operatorname{ctg} x-\ln x\right)^{\cos x}+\dfrac{\pi}{2\operatorname{arctg} x}, & \text{если } x\ge b.\end{cases}$$

Вариант 29

$$y=\begin{cases}\left|\operatorname{ch} x-\operatorname{sh} x\right|^{\operatorname{tg}^2 x+9}+\lg x, & \text{если } x\le a;\\ (6\pi)^{-x}\sqrt{\sin x+\left|1-x-x^2\right|}, & \text{если } a<x<b;\\ \mathrm{e}^x\left(1+\operatorname{arctg}^2 x\right)^{\sqrt{|x|+3}}-\ln^{\mathrm{e}} x, & \text{если } x\ge b.\end{cases}$$

Вариант 30

$$y=\begin{cases}\dfrac{x^{1,3}}{2\sin x^2-7\cos x^2}, & \text{если } x\le a;\\ \sqrt{|0{,}5-\operatorname{sh} x|}+\mathrm{e}^{2x}\operatorname{ch} 4{,}3x, & \text{если } a<x<b;\\ \dfrac{1-\operatorname{arctg} x}{1+x^{x+2}}, & \text{если } x\ge b.\end{cases}$$

2.3. Теоретические сведения

Для выполнения лабораторной работы необходимо внимательно изучить структуру *C*-программы, основные типы данных и операторы языка *C*, управляющие инструкции, позволяющие реализовывать программы линейной и разветвляющейся структуры, а также ознакомиться со средствами ввода-вывода языков *C*/*C*++ и математическими функциями, входящих в состав библиотеки **<math.h>**.

2.3.1. Структура *C*-программы

Любая *C*-программа состоит из функций и переменных. Функции содержат инструкции, описывающие вычисления, которые необходимо выполнить, а переменные хранят значения, используемые в процессе этих вычислений. Любая программа начинает свои вычисления с первой инструкции функции **main**. Таким образом, каждая *C*-программа должна содержать функцию **main**. Эта функция может не иметь параметров, и тогда ее заголовок записывается как **main()**. Если функция **main** имеет параметры, то эти параметры выбираются из строки вызова (командной строки).

Левая фигурная скобка (**{**) должна начинать тело каждой функции. Соответствующая правая фигурная скобка (**}**) должна заканчивать каждую функцию. Так что тело функции представляет собой блок, ограниченный фигурными скобками **{}**. Блок содержит выражения, заканчивающиеся точкой с запятой (**;**), которые в языке *C* называют инструкциями. Общая структура функции **main** такова:

```
main()
{
    инструкция_1
    инструкция_2
    ............
    инструкция_n
}
```

Помимо функции **main** в тексте программы могут располагаться определения вспомогательных функций.

В системах программирования *C* перед началом этапа компиляции выполняется программа предварительной (препроцессорной) обработки. Эта программа подчиняется специальным командам (директивам препроцессора), которые указывают, что в программе перед ее компиляцией нужно выполнить определенные преобразования. Обычно эти преобразования состоят во включении других текстовых файлов в файл, подлежащий компиляции, и выполнении различных текстовых замен. Так, например, появление директивы

```
#include <stdio.h>
```

приводит к тому, что препроцессор подставляет на место этой директивы текст

файла **<stdio.h>**, т.е. происходит включение информации о стандартной библиотеке ввода-вывода средствами языка *C*.

2.3.2. Переменные и основные типы данных языка *C*

В *C* любая переменная должна быть описана раньше, чем она будет использована; обычно все переменные описываются в начале функции перед первой исполняемой инструкцией. В декларации (описании) объявляются свойства переменных. Декларация состоит из названия типа и списка переменных, например:

```
int width,
    height;
float distance;
int exam_score;
char c;
```

В первом описании имеется список переменных, содержащий два имени (**width** и **height**). Обе переменные описываются как целые (**int**). Целочисленная переменная **exam_score** описана отдельно, хотя ее можно добавить к первому списку целых переменных.

Переменные можно инициализировать в месте их описания, например:

```
float distance=15.9;
```

Здесь переменной **distance** присваивается начальное значение **15.9**.

Имя переменной — это любая последовательность символов, содержащая буквы, цифры и символы подчеркивания (_), которая не начинается с цифры. Язык *C* чувствителен к регистру — прописные и строчные буквы различаются.

В *C* существует всего несколько базовых типов:

char — единичный байт, который может содержать одну литеру;

int — целое число;

float — число с плавающей точкой;

double — число с плавающей точкой повышенной точности.

Имеется также несколько квалификаторов, которые можно использовать вместе с указанными базовыми типами. Например, квалификаторы **short** (короткий) и **long** (длинный) применяются к целым числам:

```
short int counter;
long int amount;
```

В таких описаниях слово **int** можно опускать, что обычно и делается.

Квалификаторы **signed** (со знаком) и **unsigned** (без знака) можно применять к типу **char** и любому целому типу. Тип **long double** предназначен для арифметики с плавающей точкой повышенной точности.

Если операнды оператора принадлежат разным типам, то они приводятся к общему типу. Автоматически производятся лишь те преобразования, которые

без какой-либо потери информации превращают операнды с меньшим диапазоном значений в операнды с большим диапазоном значений.

2.3.3. Основные операторы языка *C*

В табл. 2.1 показаны приоритеты и порядок вычисления всех операторов языка *C*. Операторы, перечисленные на одной строке, имеют одинаковый приоритет; строки упорядочены по убыванию приоритетов. Унарные операторы **+**, **–** и ***** имеют более высокий приоритет, чем те же операторы в бинарном варианте.

Таблица 2.1 — Приоритеты и порядок выполнения операторов

Приоритет	Операторы	Порядок выполнения
1	`() [] . ->`	Слева направо
2	`! ~ ++ -- + - * & sizeof` (приведение типов)	Справа налево
3	`* / %`	Слева направо
4	`+ -`	
5	`<< >>`	
6	`< > <= >=`	
7	`== !=`	
8	`&`	
9	`^`	
10	`|`	
11	`&&`	
12	`||`	
13	`? :`	
14	`= += -= *= /= %= &= ^= |= <<= >>=`	Справа налево
15	`,`	Слева направо

Арифметические операторы

К арифметическим операторам относятся сложение (**+**), вычитание (**-**), деление (**/**), умножение (*****) и получение остатка от деления (**%**). Все операции (за исключением остатка) определены для переменных типа **int**, **char** и **float**. Остаток не определен для переменных типа **float**. Деление целых чисел сопровождается отбрасыванием дробной части.

Все арифметические операции с плавающей точкой производятся над операндами двойной точности (**double**). После того, как получен результат с двойной точностью, он приводится к типу левой части выражения, если левая часть присутствует.

Операторы отношения

В языке *C* определены следующие операторы отношения: проверка на равенство (**==**), проверка на неравенство (**!=**), меньше (**<**), меньше или равно (**<=**), больше (**>**), больше или равно (**>=**).

Все перечисленные операторы вырабатывают результат целого типа (**int**). Если данное отношение между операндами ложно, то значение этого целого — ноль. Значения, отличные от нуля, интерпретируются как истинные.

Логические операторы

К логическим операторам относятся:
&& — логическое И (*and*);
|| — логическое ИЛИ (*or*);
! — логическое НЕ (*not*).

Операндами логических операторов могут быть любые числа различных типов. Результат логического выражения — единица, если истина, и ноль, если ложь. Вычисление выражений, содержащих логические операторы, производится слева направо и прекращается, как только удается определить результат.

Операторы присваивания

К операторам присваивания относятся **=**, **+=**, **−=**, ***=** и **/=**, а также префиксные и постфиксные операторы **++** и **−−**. Все операторы присваивания присваивают переменной результат вычисления выражения.

В одном выражении оператор присваивания может встречаться несколько раз. Вычисления производятся справа налево. Например:

```
a=(b=c)*d;
```

Вначале переменной **b** присваивается значение переменной **c**, затем выполняется операция умножения на **d**, и результат присваивается переменной **a**.

Типичный пример использования многократного присваивания:

```
a=b=c=d=e=f=0;
```

Операторы **+=**, **−=**, ***=**, **/=** являются укороченной формой записи оператора присваивания:

```
a+=b; // a=a+b;
a-=b; // a=a-b;
a*=b; // a=a*b;
a/=b; // a=a/b;
```

Многие компиляторы генерируют код, который выполняется быстрее при использовании таких операторов присваивания.

Префиксные и постфиксные операторы присваивания **++** и **−−** используют для увеличения (инкремент) и уменьшения (декремент) на единицу значения

переменной. Эти операторы можно использовать как в префиксной форме (помещая перед переменной, например, **++n**), так и в постфиксной форме (помещая после переменной, например, **n++**). В префиксной форме сначала изменяется операнд, а затем его значение становится результирующим значением выражения, а в постфиксной форме значением выражения является исходное значение операнда, после чего он изменяется. Приведенный ниже пример поясняет префиксную и постфиксную формы инкремента:

```
#include <stdio.h> // стандартная библиотека ввода-вывода
#include <math.h>  // библиотека математических функций

main()
{
    int x=5, y=5;
    ..................................................
    prinf("Значение префиксного выражения: %d\n", ++x);
    prinf("Значение постфиксного выражения: %d\n", y++);
    prinf("Значение x после приращения: %d\n, x");
    prinf("Значение y после приращения: %d\n, y");
    ..................................................
    return 0;
}
```

При выполнении приведенного фрагмента программы на экран будет выведен следующий результат:

```
Значение префиксного выражения: 6
Значение постфиксного выражения: 5
Значение x после приращения: 6
Значение y после приращения: 6
```

2.3.4. Основные средства ввода-вывода языков *C*/*C*++

Основные средства ввода-вывода языка *C*: функции `printf` и `scanf`

В языке *C* ввод-вывод данных реализуется с помощью внешних функций. Одними из наиболее универсальных и полезных функций ввода и вывода являются соответственно функции **`scanf`** и **`printf`**, описанные в головном файле **`<stdio.h>`**.

Функцию **`printf`** можно использовать для вывода любой комбинации символов, целых и вещественных чисел, строк, беззнаковых целых, длинных целых и беззнаковых длинных целых.

Типичный пример использования функции **`printf`**:

```
#include <stdio.h>          // подключение библиотеки стандартных
                            // средств ввода-вывода языка C
main()
{
    int age=22;             // возраст Максима
```

```
    float income=534.72;    // доход Максима

    printf("\nВозраст Максима: %d. Его доход: $%.2f.\n",
           age,income);

    return 0;
}
```

Функция **printf** выводит на экран значения своих аргументов **age** и **income** в формате, который определяется управляющей строкой, являющейся первым параметром этой функции. Все символы этой строки, кроме спецификаций формата (**%d** и **%.2f**) и управляющей последовательности (**\n**), выводятся на экран без изменений.

Таким образом, при выполнении функции **printf** произойдет следующее. Последовательность символов **\n** переведет курсор на новую строку. В результате последовательность символов «**Возраст Максима:** » будет выведена с начала новой строки.

Символы **%d** — это спецификация формата для целой переменной. Вместо **%d** будет подставлено значение переменной **age**. Далее будет выведена литеральная строка «**. Его доход: $**».

Символы **%.2f** — это спецификация формата для вещественной переменной, а также указание формата для вывода только двух цифр после десятичной точки. Вместо **%.2f** будет выведено значение переменной **income** в указанном формате. В заключение управляющая последовательность **\n** вызовет переход на новую строку.

Как видно из рассмотренного примера, спецификации формата помещаются внутри печатаемой строки. Вслед за этой строкой должен стоять ноль или более переменных, разделенных запятыми. Каждой спецификации в управляющей строке функции **printf** должна соответствовать переменная адекватного типа. Если используется несколько спецификаций, то всем им должны соответствовать переменные того типа, который задается спецификацией.

Формально спецификацию формата можно определить следующим образом:

```
%[флаг][ширина][.точность][h|l|L]символ_формата
```

где **ширина** — минимальное количество позиций, отводимых под выводимое значение;

точность — количество позиций, отводимых под дробную часть числа;

h, **l**, **L** — модификаторы, информирующие о типе аргумента.

Модификатор **h** указывает, что соответствующий аргумент должен печататься как **short** или **unsigned short**; **l** сообщает, что аргумент имеет тип **long** или **unsigned long**; **L** информирует, что аргумент принадлежит типу **long double**. Возможные значения полей **флаг** и **символ_формата** приведены в табл. 2.2 и 2.3.

Таблица 2.2 — Описание значений поля **флаг**

флаг	Назначение
-	Выравнивание по левому краю поля
+	Печать числа всегда со знаком
пробел	Если первая литера — не знак, то числу должен предшествовать пробел

Таблица 2.3 — Описание значений поля **символ_формата**

символ_формата	Тип выводимого объекта
c	**char;** единичная литера
s	**char *;** строка
d,i	**int;** знаковая десятичная запись
o	**int;** беззнаковая восьмеричная запись
U	**int;** беззнаковое десятичное целое
x,X	**int;** беззнаковая шестнадцатеричная запись
f	**double;** вещественное число с фиксированной точкой
e,E	**double;** вещественное число с плавающей точкой
g,G	**double;** вещественное число в виде **%f** или **%e** в зависимости от значения
P	**void *;** указатель
%	знак процента **%**

Функция **scanf** служит для ввода данных. Подобно функции **printf**, **scanf** использует спецификацию формата, сопровождаемую списком вводимых переменных. Каждой вводимой переменной в функции **scanf** должна соответствовать спецификация формата. Перед именами переменных следует ставить символ **&**. Этот символ означает «взять адрес переменной». Ниже приведен пример программы, использующей функцию **scanf**:

```
#include <stdio.h>

main()
{
    int weight, // вес
        height; // рост

    printf("Введите Ваш вес:\n");
    scanf("%d",&weight);
    printf("Введите Ваш рост:\n");
    scanf("%d",&height);

    printf("\nВаш вес = %d;\nВам рост = %d.\n", weight,height);

    return 0;
}
```

Здесь **&weight** и **&height** — это адреса переменных **weight** и **height**

соответственно.

Основные средства ввода-вывода языка *C*++: объекты `cin` и `cout`

В *C*++ вывод на экран выполняется с помощью объекта стандартного потока вывода **`cout`**, а ввод с клавиатуры осуществляется с помощью объекта стандартного потока ввода **`cin`**. Для использования этих объектов необходимо подключить головной файл **`<iostream.h>`**. Для обработки форматного ввода-вывода при помощи так называемых манипуляторов потока необходимо подключить головной файл **`<iomanip.h>`**.

Рассмотрим пример использования объекта **`cout`**:

```
#include <iomanip.h>  // подключение библиотеки средств
                      // манипулирования потоками языка C++
#include <iostream.h> // подключение библиотеки стандартных
                      // объектов и операций с потоками
                      // ввода-вывода средствами языка C++

main()
{
    int age=22;           // возраст Максима
    float income=534.72; // доход Максима

    cout<<'\n'
        <<"Возраст Максима: "<<age<<'.'
        <<" Его доход: $"<<setprecision(2)
        <<income<<'.'<<endl;

    return 0;
}
```

Результат работы программы:

```
Возраст Максима: 22. Его доход: $534.72.
```

Вывод в поток выполняется с помощью операции поместить в поток **`<<`**. В рассматриваемом примере объекту **`cout`** с помощью операции **`<<`** передаются значения, которые необходимо вывести на экран. Операция **`<<`** является «более интеллектуальной» по сравнению с функцией **`printf`**, так как определяет тип выводимых данных. Для вывода нескольких объектов операция **`<<`** используется в сцепленной форме.

Манипулятор потока **`endl`** вызывает переход на новую строку и очищает буфер вывода, т.е. заставляет буфер немедленно вывести данные, даже если он полностью не заполнен. Очистка (сброс) буфера вывода может быть также выполнена манипулятором **`flush`**.

При выводе значения переменной **`income`** используется манипулятор потока **`setprecision`**. Использование **`setprecision(2)`** — это указание формата для вывода только двух цифр после десятичной точки. Поскольку ма-

нипулятор **setprecision** принимает параметр, он называется параметризированным манипулятором потока.

Для установки ширины поля вывода может быть использован манипулятор потока **setw**, принимающий в качестве параметра число символьных позиций, в которые будет выведено значение.

Ввод потока осуществляется с помощью операции взять из потока **>>**. Эта операция применяется к объекту стандартного потока ввода **cin**. Рассмотрим пример использования объекта **cin**:

```
#include <iostream.h>

main()
{
    int weight, // вес
    height;     // рост

    cout<<"Введите Ваш Вес:"<<endl;
    cin>>weight;
    cout<<"Введите Ваш рост: "<<endl;
    cin>>height;

    cout<<"\nВаш вес = "<<weight<<';'<<endl
        <<"Ваш рост = "<<height<<'.'<<endl;

    return 0;
}
```

Здесь считываемые значения размещаются в переменных **weight** и **height**. В процессе ввода последовательность символов, набранная на клавиатуре, преобразуется в необходимое внутреннее представление (в рассматриваемом примере целочисленное).

2.3.5. Управляющие инструкции языка *C*: if-else, условное выражение, switch

Инструкция if-else

Инструкция **if-else** используется для принятия решения. Ниже представлен ее синтаксис:

```
if (выражение)
    инструкция_1
else
    инструкция_2;
```

причем **else**-часть может быть опущена. Сначала вычисляется выражение, и, если оно истинно, то выполняется **инструкция_1**. Если выражение ложно и существует **else**-часть, то выполняется **инструкция_2**. Необходимо отметить, что **else**-часть всегда относится к ближайшей **if** -части.

Условное выражение

Условное выражение, написанное с помощью тернарного оператора «**?:**», представляет собой другой способ записи инструкции **if-else**. В выражении

```
выражение1 ? выражение2 : выражение3
```

первым вычисляется **выражение1**. Если его значение отлично от нуля, то вычисляется **выражение2** и значение этого выражения становится значением всего условного выражения. В противном случае вычисляется **выражение3**, и его значение становится значением условного выражения. Таким образом, чтобы установить в **z** наибольшее из **a** и **b**, можно записать:

```
z = (a > b) ? a : b; // z=max(a,b)
```

Инструкция switch

Инструкция **switch** используется для выбора одного из многих путей. Она проверяет, совпадает ли значение выражения с одним из значений, входящих в некоторое множество целых констант, и выполняет соответствующую этому значению ветвь программы:

```
switch (выражение)
{
    case конст_выражение_1: последовательность_инструкций_1
    case конст_выражение_2: последовательность_инструкций_2
    ..........................................................
    case конст_выражение_n: последовательность_инструкций_n
    default: последовательность_инструкций_n+1
}
```

Константные выражения всех **case**-ветвей должны быть целочисленными и должны отличаться друг от друга.

Инструкция **switch** выполняется следующим образом. Вначале вычисляется **выражение**, и результат сравнивается с каждой **case**-константой. Если одна из **case**-констант равна значению выражения, управление переходит на последовательность инструкций с соответствующей **case**-меткой. Если ни с одной из **case**-констант нет совпадения, управление передается на последовательность инструкций с **default**-меткой, если такая имеется, в противном случае ни одна из последовательностей инструкций **switch** не выполняется. Ветви **case** и **default** можно располагать в любом порядке.

Для иллюстрации инструкции **switch** рассмотрим программу, которая определяет вид литеры, введенной пользователем с клавиатуры. Вид литеры — это цифра (**digit**), пробельная литера (**white space**) или другая литера (**other**). К пробельным литерам относят пробел (**' '**), литеру табуляции (**'\t'**) и литеру новая строка (**'\n'**). Ниже представлен текст программы.

```
#include <iostream.h> // подключение библиотеки стандартных
                      // объектов и операций с потоками
                      // ввода-вывода средствами языка C++
main()
// определение вида литеры, введенной с клавиатуры
// (цифра, пробельная литера или другая литера)
{
    char c;      // литера, вводимая с клавиатуры

    c=cin.get(); // ввод литеры с клавиатуры

    switch(c){
    case '0': case '1': case '2': case '3': case '4':
    case '5': case '6': case '7': case '8': case '9':
        // цифра
        cout<<"digit"<<endl;
        break;
    case ' ': case '\t': case '\n':
        // пробельная литера
        cout<<"white space"<<endl;
        break;
    default:
        // другая литера
        cout<<"other"<<endl;
        break;
    }

    return 0;
}
```

Литера, заключенная в одиночные кавычки, представляет целое значение, равное коду этой литеры (в кодировке, принятой на данной машине). Это так называемая литерная константа. Например, **'A'** есть литерная константа; в наборе *ASCII* ее значение равняется **65**. **'\t'** и **'\n'** обозначают коды литеры табуляции и литеры «новая строка» соответственно.

Функция-элемент **get** объекта **cin** вводит одиночный символ из потока и возвращает этот символ в качестве значения вызова функции. Для вывода одиночного символа в поток используется функция-элемент **put**, которая в качестве аргумента принимает значение кода символа. Следует отметить, что стандартная библиотека ввода-вывода **<stdio.h>** включает функции для чтения и записи одной литеры **getchar** и **putchar**, аналогичные функциям-элементам **get** и **put** соответственно.

Инструкция **break** вызывает немедленный выход из инструкции **switch**. Поскольку выбор ветви **case** реализуется как переход на метку, то после выполнения одной ветви **case**, если ничего не предпринять, программа «провалится вниз» на следующую ветвь. Инструкция **break** — наиболее распространенное средство выхода из инструкции **switch**.

2.3.6. Математические функции файла <math.h>

Ниже приведен перечень основных математических функций, описанных в головном файле **<math.h>**. Аргументы **x** и **y** функций имеют тип **double**; все функции возвращают значения типа **double**. Углы в тригонометрических функциях задаются в радианах:

sin(x) — синус x;
cos(x) — косинус x;
tan(x) — тангенс x;
asin(x) — арксинус x в диапазоне $[-\pi/2;\pi/2]$, $x \in [-1;1]$;
acos(x) — арккосинус x в диапазоне $[0;\pi]$, $x \in [-1;1]$;
atan(x) — арктангенс x в диапазоне $[-\pi/2;\pi/2]$;
atan2(y,x) — арктангенс y/x в диапазоне $[-\pi;\pi]$;
sinh(x) — гиперболический синус x;
cosh(x) — гиперболический косинус x;
tanh(x) — гиперболический тангенс x;
exp(x) — экспоненциальная функция e^x;
log(x) — натуральный логарифм $\ln x$, $x > 0$;
log10(x) — десятичный логарифм $\lg x$, $x > 0$;
pow(x,y) — возведение в степень x^y;
sqrt(x) — квадратный корень $\sqrt{x}$, $x \ge 0$;
ceil(x) — наименьшее целое в виде **double**, которое $\ge x$;
floor(x) — наибольшее целое в виде **double**, которое $\le x$;
fabs(x) — абсолютное значение $|x|$;
fmod(x,y) — остаток от деления x на y в виде числа с плавающей точкой.

2.3.7. Пример программы разветвляющейся структуры

Ниже представлен текст *C*-программы, вычисляющей значение функции

$$y = f(x) = \begin{cases} \sin(2x + \pi/7), & \text{если } x \le a; \\ x^{3,21} + \operatorname{tg} x, & \text{если } a < x < b; \\ \sqrt{x}\lg x, & \text{если } x \ge b. \end{cases}$$

Значения параметров a, b и аргумента x вводятся с клавиатуры. Результаты вычислений выводятся на дисплей в формате с плавающей точкой.

```
#include <conio.h>     // подключение библиотеки управления
                       // вводом-выводом средствами консоли MSDOS
#include <math.h>      // подключение библиотеки математических
                       // функций
#include <stdio.h>     // подключение библиотеки стандартных
                       // средств ввода-вывода языка C
```

```
#define PI 3.14159265

main()
// вычисление значения функции y=f(x)
{
    float a, // параметр a
    b,       // параметр b
    x,       // аргумент x
    y;       // значение функции y

    clrscr ();

    // ввод значений параметров a, Ь и аргумента x
    printf("Введите значение параметра a: ");
    scanf("%f",&a);
    printf("Введите значение параметра Ь: ");
    scanf("%f",&b);
    printf("Введите значение аргумента x: ");
    scanf("%f",&x);

    // вычисление значения функции y
    if (x<=a)
        y=sin(2*x+PI/7);
    else if (x<b) // a<x<b
             y=pow(x,3.21)+tan(x);
         else // x>=b
             y=sqrt(x)*log10(x);

    //вывод значения функции y в формате с плавающей точкой
    printf("Значение функции y = %e\n",y);

    printf("Нажмите любую клавишу...");
    getch();

    return 0;
}
```

С помощью директивы **#define PI 3.14159265** оказывается возможным указать препроцессору, чтобы он при любом появлении идентификатора **PI** в тексте программы заменял его на значение **3.14159265** ($\approx\pi$).

Функция **clrscr** библиотеки **<conio.h>** выполняет очистку экрана и перемещение курсора в левый верхний угол.

Функция **getch** библиотеки **<conio.h>** используется для ввода символов с клавиатуры без их высвечивания на экране. С помощью **getch** можно считать коды основных клавиш (ASCII-коды) и расширенные коды. Расширенные коды — это коды верхнего ряда клавиш, коды правой части клавиатуры и коды комбинаций клавиш *Alt*, *Ctrl*, *Shift* с другими клавишами. В случае считывания расширенных кодов при первом обращении функция **getch** возвращает нулевое значение, а ее повторный вызов позволяет получить расширенный код.

Используя возможности языка *C++*, вышеприведенную программу можно

переписать следующим образом:

```
#include <conio.h>      // подключение библиотеки управления
                        // вводом-выводом средствами консоли MSDOS
#include <iostream.h>   // подключение библиотеки стандартных
                        // объектов и операций с потоками
                        // ввода-вывода средствами языка C++
#include <math.h>       // подключение библиотеки математических
                        // функций

const double pi=3.14159265;

main()
// вычисление значения функции y=f(x)
{
    // объявление переменных и очистка экрана
    ..............................................
    // ввод значений параметров a, b и аргумента x
    cout<<"Введите параметр a: ";
    cin>>a;
    cout<<"Введите параметр b: ";
    cin>>b;
    cout<<" Введите аргумент x: ";
    cin>>x;
    // вычисление значения функции y
    ..............................................
    // вывод значения функции y в формате с плавающей точкой
    cout.setf(ios::scientific,
              ios::floatfield);
    cout<<"Значение функции y = f(x) = "<<y<<endl;

    cout«"Нажмите любую клавишу...";
    getch();

    return 0;
}
```

Как видно, в *C++* вместо символических констант, которые создаются директивой препроцессора **#define**, имеется возможность использования константных переменных. Строка

```
const double pi=3.14159265;
```

использует спецификацию **const** для объявления константы **pi**, имеющей значение **3.14159265**. После определения константной переменной изменить её значение нельзя. Следует отметить, что в *C++* отдается предпочтение использованию константных переменных, а не символических констант. Константные переменные, в отличие от символических констант, являются данными определенного типа, и их имена видны отладчику.

Строка

```
cout.setf(ios::scientific,
          ios::floatfield);
```

устанавливает экспоненциальный формат вывода вещественных чисел (формат с плавающей точкой). Для отображения чисел в формате с фиксированной точкой следует записать

```
cout.setf(ios::fixed,
          ios::floatfield);
```

Здесь функция-элемент **setf** объекта **cout** используется для управления установкой флагов формата вывода вещественных чисел **ios::scientific** или **ios::fixed**, которые содержатся в статическом элементе данных **ios::floatfield**.

2.4. Порядок выполнения работы

Вариант задания выдается преподавателем. Перед выполнением работы необходимо ознакомится 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 определения и области значений вычисляемой функции y. Желательным является построение графика функции.

2.4.3. Разработать схему алгоритма решения задачи.

2.4.4. Написать две программы вычисления функции y. Одна программа для ввода-вывода данных должна использовать функции **scanf** и **printf**, а другая — объекты **cin** и **cout**. Программы обязательно должны содержать комментарии.

2.4.5. Разработать тестовые примеры, которые должны включать, по крайней мере, по два значения аргумента из каждого интервала кусочно-заданной функции y. Тестовые примеры должны также отражать поведение функции y в граничных точках.

2.4.6. Выполнить тестирование и отладку написанных программ, используя разработанные тестовые примеры. Особое внимание следует обратить на значения функции y в граничных точках.

2.4.7. Получить результаты работы программы

2.4.8. Подготовить отчет по работе.

2.5. Оформление отчета

2.5.1. Сформулировать цель работы.

2.5.2. Произвести постановку задачи и текст индивидуального задания.

2.5.3. Привести краткие теоретические сведения, касающиеся использованных в программе математических функций, управляющей инструкции **if-else**, и средств ввода-вывода (функций **scanf** и **printf**, а также объектов

cin и **cout**).

2.5.4. Привести математическое обоснование задачи (анализ области определения и области значений функции, подлежащей вычислению, а также график этой функции).

2.5.5. Изобразить схему алгоритма решения задачи.

2.5.6. Привести тексты программ.

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

2.5.8. Привести результаты тестирования и отладки программ.

2.5.9. Привести результаты работы программ.

2.5.10. Сделать выводы по работе.

2.6. Контрольные вопросы

2.6.1. Опишите структуру *C*-программы.

2.6.2. В чем состоит препроцессорная обработка программы?

2.6.3. Перечислите и охарактеризуйте базовые типы языка *C*.

2.6.4. Перечислите и охарактеризуйте квалификаторы языка *C*.

2.6.5. Что такое приведение типа?

2.6.6. Перечислите и охарактеризуйте арифметические операторы.

2.6.7. Перечислите и охарактеризуйте операторы отношения.

2.6.8. Перечислите и охарактеризуйте логические операторы.

2.6.9. Перечислите и охарактеризуйте операторы присваивания.

2.6.10. Опишите средства ввода-вывода языка *C*.

2.6.11. Опишите средства ввода-вывода языка *C*++.

2.6.12. Опишите инструкцию **if-else**.

2.6.13. Что такое условное выражение?

2.6.14. Опишите инструкцию **switch**.

2.6.15. Опишите математические функции файла **<math.h>**.

3. ЛАБОРАТОРНАЯ РАБОТА №3
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМОВ ЦИКЛИЧЕСКОЙ СТРУКТУРЫ

3.1. Цель работы

Получение навыков программирования алгоритмов циклической структуры на языке *C*.

Исследование эффективности применения различных видов циклов в задаче табулирования функции.

3.2. Варианты заданий

Вычислить и вывести на экран в виде таблицы значения функции $y = f(x)$ на интервале от $x_{\text{нач}}$ до $x_{\text{кон}}$ с шагом Δx. Таблицу снабдить заголовком и шапкой. Вид функции y выбирать в соответствии с вариантами задания к лабораторной работе №2 настоящих методических указаний. Значения параметров a, b, а также $x_{\text{нач}}$, $x_{\text{кон}}$ и Δx выбираются произвольно и вводятся с клавиатуры.

Результаты вычислений выводить в формате с фиксированной точкой.

3.3. Теоретические сведения

3.3.1. Циклы в языках *C*/*C*++

Цикл представляет собой участок программы, повторяемый многократно. В языках *C*/*C*++ имеется три разновидности циклов: **`while`**, **`do while`** и **`for`**.

Цикл `while`

Цикл **`while`** имеет следующий синтаксис:

```
while (выражение)
       инструкция
```

В цикле **`while`** вначале вычисляется **`выражение`**. Если его значение отлично от нуля (т.е. имеет значение истина), то выполняется **`инструкция`** и вычисление выражения повторяется. Этот цикл продолжается до тех пор, пока **`выражение`** не станет равным нулю (примет значение ложь), после чего вычисления возобновятся с точки, расположенной сразу за инструкцией.

Перед входом в цикл **`while`** обычно инициализируют одну или несколько переменных для того, чтобы **`выражение`** имело какое-либо конкретное значение. Инструкция или последовательность инструкций (составная инструкция), составляющих тело цикла, должны, как правило, изменять значения одной или нескольких переменных, входящих в **`выражение`**, с тем, чтобы за конечное число итераций **`выражение`** приняло нулевое значение и цикл завершился. Цикл **`while`** — это цикл с предусловием; т.е. решение о выполнении еще одной итерации цикла принимается перед началом цикла. Цикл **`while`** заверша-

ется в следующих случаях:

— обращение в ноль выражения в заголовке цикла;

— выполнение в теле цикла инструкции **break**;

— выполнение в теле цикла инструкции **return**.

В первых двух случаях управление передается в точку, расположенную сразу за циклом. В третьем случае происходит выход из функции.

Цикл do while

Цикл **do while** имеет следующий синтаксис:

```
do
   инструкция
while (выражение);
```

В цикле **do while** сначала выполняется **инструкция**, затем вычисляется **выражение**. Если оно истинно (отлично от нуля), то **инструкция** выполняется снова и т.д. Когда **выражение** становится ложным (обращается в ноль), цикл заканчивает работу. Цикл **do while** завершается в тех же случаях, что и цикл **while**. Цикл **do while** — это цикл с постусловием, т.е. проверка условия продолжения цикла **(выражение)** выполняется после каждого прохождения тела цикла (**инструкция**).

Цикл for

Цикл **for** имеет следующий синтаксис:

```
for (выражение1; выражение2; выражение3)
     инструкция
```

Каждое из трех выражений **выражение1**, **выражение2**, **выражение3** можно опускать, но точку с запятой опускать нельзя. При отсутствии **выражения1** или **выражения3** считается, что их нет в конструкции цикла; при отсутствии **выражения2** полагается, что его значение всегда истинно.

Обычно **выражение1** служит для инициализации счетчика цикла, **выражение2** — для выполнения проверки на окончание цикла, **выражение3** — для модификации счетчика цикла.

Цикл **for** является циклом с предусловием; инструкция **for** эквивалентна следующей конструкции

```
выражение1;
while (выражение2){
       инструкция
       выражение3;
}
```

В языке *C++* переменную можно объявить в части инициализации (**выра-**

жение1) инструкции **for**, например:

```
for (int counter=1;counter<=10;counter++){
     // тело цикла
     ..............
}
```

Инструкции break и continue

Инструкции **break** и **continue** изменяют поток управления. Когда инструкция **break** выполняется в инструкциях **switch**, **while**, **do while** или **for**, происходит немедленный выход из соответствующей инструкции, и управление передается в точку, расположенную сразу за инструкцией. Обычное назначение инструкции **break** — досрочно прервать цикл или пропустить оставшуюся часть инструкции **switch**. Необходимо заметить, что инструкция **break** вызывает немедленный выход из самого внутреннего из объемлющих ее циклов.

Инструкция **continue** в циклах **while**, **do while** или **for** вызывает пропуск оставшейся части тела цикла, после чего начинается выполнение следующей итерации цикла. В циклах **while** и **do while** после выполнения инструкции **continue** осуществляется проверка условия продолжения цикла. В цикле **for** после выполнения инструкции **continue** выполняется выражение приращения (**выражение3**), а затем осуществляется проверка условия продолжения цикла (**выражение2**). Таким образом, инструкция **continue** вынуждает ближайший объемлющий ее цикл начать следующий шаг итерации.

Цикл for и оператор запятая

Иногда в цикле **for выражение1** и **выражение3** представляются как списки выражений, разделенных запятыми. В этом случае запятая используется как оператор запятая или операция следования, гарантирующая, что список выражений будет вычисляться слева направо. Оператор запятая имеет самый низкий приоритет (см. табл. 2.1 в лабораторной работе №2). Значение и тип списка выражений, разделенных запятыми, равны значению и типу самого правого выражения в списке.

Оператор запятая наиболее часто применяется в цикле **for**. Этот оператор дает возможность использовать несколько выражений задания начальных значений и (или) несколько выражений приращения переменных, что позволяет вести несколько индексов (управляющих переменных) параллельно, например:

```
for (int left=0,right=n-1;left<right;left++,right--){
     // тело цикла
     ..............
}
```

Кроме цикла **for**, оператор запятая можно использовать в тех случаях, когда последовательность инструкций мыслится как одна операция, например:

```
temp=a[i],a[i]=a[i+1],a[i+1]=temp;  // обмен
```

3.3.2. Пример программы табулирования функции

Постановка задачи

Написать программу табулирования (печати таблицы значений) кусочно-заданной функции $y = f(x)$ на интервале от $x_{\text{нач}}$ до $x_{\text{кон}}$ с шагом Δx. Таблицу снабдить заголовком и шапкой. Вид функции определяется формулой

$$y = f(x) = \begin{cases} a^2, & \text{если } x < 0; \\ ax, & \text{если } 0 \le x < 10; \\ 5a, & \text{если } x \ge 10. \end{cases}$$

Если $a > 10$, то значения функции должны выводиться в виде целых чисел. Значения параметра a, а также $x_{\text{нач}}$, $x_{\text{кон}}$ и Δx вводятся с клавиатуры. Результаты вычислений выводятся в формате с фиксированной точкой.

Описание алгоритма решения задачи в словесной форме

В словесной форме алгоритм решения задачи можно сформулировать следующим образом:

1) Ввести с клавиатуры исходные данные: a, $x_{\text{нач}}$, $x_{\text{кон}}$ и Δx.
2) Вывести заголовок и шапку таблицы.
3) Положить $x = x_{\text{нач}}$.
4) Вычислить значение функции y по вышеприведенной формуле.
5) Если $a > 10$, то привести значение функции y к целому типу.
6) Вывести на экран строку таблицы значений функции.
7) Увеличить значение x на величину Δx.
8) Если значение x не превышает $x_{\text{кон}}$, то перейти к пункту 4, иначе закончить выполнение программы.

Так как пункты 4…7 алгоритма выполняются многократно, то для их выполнения необходимо организовать цикл. Напишем два варианта программы с использованием циклов **`while`** и **`for`**.

Использование цикла `while`

Ниже представлена программа табулирования функции с использованием цикла **`while`**:

```
#include <conio.h>     // подключение библиотеки управления
                       // вводом-выводом средствами консоли MSDOS
#include <stdio.h>     // подключение библиотеки стандартных
                       // средств ввода-вывода языка C
#include <math.h>       // подключение библиотеки математических
                        // функций
```

```
main()
// программа табулирования функции y=f(x)
// на интервале от xn до xk с шагом dx
{
    float a,  // параметр
          x,  // аргумент функции y
          xn, // начальное значение аргумента x
          xk, // конечное значение аргумента x
          dx, // шаг
          y;  // значение функции y

    clrscr ();

    // ввод a, xn, xk, dx
    printf("Введите параметр a: "), scanf("%f",&a);
    printf("Введите xn: "), scanf("%f",&xn);
    printf("Введите xk: "), scanf("%f",&xk);
    printf("Введите шаг dx: "), scanf("%f",&dx);

    // вывод заголовка и шапки таблицы
    printf("Таблица значений функции y=f(x)\n");
    printf(" ┌──────────────┬──────────────┐\n") ;
    printf(" │      x       │   y = f(x)   │\n");
    printf(" ├──────────────┼──────────────┤\n") ;

    // табулирование функции y
    x=xn; // начальное значение аргумента x
    while (x<=xk){ // вывод строки таблицы
           printf(" │  %-9.3f│",x); // вывод аргумента x
           // вычисление значения функции y
           (x<0) ? (y=a*a) : ((x<10) ? (y=a*x) : (y=5*a));
           // вывод значения функции y
           if (a>10)
               printf(" %-10d│\n",(int)y); // целочисленное y
           else
               printf(" %-10.3f│\n",y); // вещественное y
           x+=dx; // приращение аргумента x
    }

    printf (" └──────────────┴──────────────┘\n");

    printf("Нажмите любую клавишу...");
    getch();

    return 0;
}
```

Поскольку формулы, определяющие функцию y, являются достаточно короткими, то вычисление значения функции y целесообразно осуществлять не с помощью инструкции **if else**, а с помощью условного выражения.

Вывод таблицы значений функции осуществляется средствами функции **printf** стандартной библиотеки ввода-вывода **<stdio.h>**.

Воспроизвести символ большинства кодов на экране можно, нажав соответствующую ему клавишу. Но этого нельзя сделать, например, для кодов псевдографики, с помощью которых возможно построение рамки таблицы. Любой из символов, имеющих коды от 1 до 255, можно воспроизвести на экране, дополнительно используя клавишу *Alt*. Для этого в среде *Turbo* (*Borland*) *C++* надо установить режим работы с цифровой клавиатурой (правая часть клавиатуры), нажав клавишу *Num Lock*, что фиксируется индикатором *Num Lock*. Затем надо нажать клавишу *Alt*, и, не отпуская ее, на цифровой клавиатуре набрать код символа, после чего отпустить клавишу *Alt*. В результате на экране воспроизведется символ, код которого был набран. Коды символов псевдографики представлены в табл. 3.1.

Таблица 3.1 — Коды символов псевдографики

Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ
176	▒	184	╕	192	└	200	╚	208	╨	216	╪
177	▒	185	╣	193	┴	201	╔	209	╤	217	┘
178	▓	186	║	194	┬	202	╩	210	╥	218	┌
179	│	187	╗	195	├	203	╦	211	╙	219	█
180	┤	188	╝	196	─	204	╠	212	╘	220	▄
181	╡	189	╜	197	┼	205	═	213	╒	221	▌
182	╢	190	╛	198	╞	206	╬	214	╓	222	▐
183	╖	191	┐	199	╟	207	╧	215	╫	223	▀

Использование цикла `for`

Ниже представлена программа табулирования функции с использованием цикла **`for`**:

```
#include <conio.h>      // подключение библиотеки управления
                        // вводом-выводом средствами консоли MSDOS
#include <iomanip.h>    // подключение библиотеки средств
                        // манипулирования потоками
#include <iostream.h>   // подключение библиотеки стандартных
                        // объектов и операций с потоками
                        // ввода-вывода средствами языка C++
#include <math.h>       // подключение библиотеки математических
                        // функций
main()
// программа табулирования функции y=f(x)
// на интервале от xn до xk с шагом dx
{
    // объявление переменных и очистка экрана
    ........................................
    // ввод a, xn, xk, dx
    cout<<"Введите параметр a: ", cin>>a;
```

```
    cout<<"Введите xn: " , cin>>xn;
    cout<<"Введите xk: ", cin>>xk;
    cout<<"Введите шаг dx: ", cin>>dx;

    // вывод заголовка и шапки таблицы
    cout<<"Таблица значений функции y = f(x)"<<endl
        <<"┌──────────────┬──────────────────┐"<<endl
        <<"│      x       │   y = f(x)       │"<<endl
        <<"├──────────────┼──────────────────┤"<<endl;

    // табулирование функции y
    cout.precision(3), cout.setf(ios::showpoint);
    cout.setf(ios::left,ios::adjustfield);
    cout.setf(ios::fixed,ios::floatfield);
    for(x=xn;x<=xk;x+=dx){ // вывод строки таблицы
        cout<<" │ "<<setw(9)<<x<<'│'; // вывод аргумента x
        // вычисление значения функции y
        (x<0) ? (y=a*a) : ((y<10) ? (y=a*x) : (y=5*a));
        // вывод значения функции y
        cout<<"  "<<setw(10);
        // целочисленное или вещественное y
        (a>10) ? cout<<(int)y : cout<<y;
        cout<<'│'<<endl;
    }

    cout<<"└──────────────┴──────────────────┘"<<endl;

    cout<<"Нажмите любую клавишу...";
    getch();

    return 0;
}
```

В вышеприведенной программе ввод-вывод данных осуществляется с помощью объектов **cin** и **cout**. Последовательность инструкций

```
cout.precision(3), cout.setf(ios::showpoint);
cout.setf(ios::left,ios::adjustfield);
cout.setf(ios::fixed,ios::floatfield);
```

задает формат для вывода значения аргумента x и значения функции y в строке таблицы. Для установки ширины поля используется манипулятор потока **setw**. Функция-элемент объекта **cout precision** позволяет задать точность печатаемого вещественного числа (т.е. число разрядов справа от десятичной точки). Функция-элемент **setf** используется для управления флагами состояния формата.

Флаг **ios::showpoint** устанавливается для вывода чисел с обязательной печатью десятичной точки и нулевых младших разрядов (нулей в конце числа). Так, например, значение **79.0** без установки **ios::showpoint** будет напечатано как **79**, а с установкой **ios::showpoint** — как **79.00000** (коли-

чество нулей определяется заданной точностью).

Флаги **ios::left** и **ios::right** содержатся в статическом элементе данных **ios::adjustfield** и позволяют выравнивать печать соответственно по левой или правой границам поля.

Флаги **ios::fixed** и **ios::scientific**, управляющие форматом вывода вещественных чисел, описаны в лабораторной работе №2 настоящих методических указаний (см. стр. 33).

3.4. Порядок выполнения работы

Вариант задания соответствует варианту в работе №2. Перед выполнением работы необходимо ознакомится с разделами 3.1 — 3.3 данных методических указаний. Индивидуальные задания для выполнения работы см. в разделах 2.2 и 3.2.

3.4.1. Внимательно изучить постановку задачи.

3.4.2. Выполнить анализ области определения и области значений вычисляемой функции y. Желательным является построение графика функции. Для выполнения этого пункта можно воспользоваться результатами предыдущей лабораторной работы.

3.4.3. Разработать схему алгоритма решения задачи.

3.4.4. Написать два варианта программы табулирования функции y, используя циклы **while** и **for**. В одном из вариантов ввод-вывод должен базироваться на функциях **scanf** и **printf**, а в другом — на объектах **cin** и **cout**. Программы обязательно должны содержать комментарии.

3.4.5. Разработать тестовые примеры (таблицы значений функции). При разработке тестовых примеров целесообразно использовать отлаженную программу из предыдущей лабораторной работы для вычисления значений функции y.

3.4.6. Выполнить тестирование и отладку написанных программ, используя разработанные тестовые примеры.

3.4.7. Получить результаты работы программы.

3.4.8. Подготовить отчет по работе.

3.5. Оформление отчета

3.5.1. Сформулировать цель работы.

3.5.2. Произвести постановку задачи и текст индивидуального задания.

3.5.3. Привести краткие теоретические сведения.

3.5.4. Привести математическое обоснование задачи (анализ области определения и области значений функции, подлежащей вычислению, а также график этой функции).

3.5.5. Изобразить схему алгоритма решения задачи.

3.5.6. Привести тексты программ.

3.5.7. Составить таблицу, в которой указать имена всех переменных используемых в программах и их тип, а также кратко описать назначение пере-

менных.

3.5.8. Привести результаты тестирования и отладки программ.

3.5.9. Привести результаты работы программы.

3.5.10. Сделать выводы по работе.

3.6. Контрольные вопросы

3.6.1. Что называют циклом, условием продолжения (окончания) цикла, телом цикла?

3.6.2. Что такое итерация?

3.6.3. Перечислите типы циклов в языках *C*/*C*++.

3.6.4. Охарактеризуйте цикл с предусловием.

3.6.5. Охарактеризуйте цикл с постусловием.

3.6.6. Охарактеризуйте цикл **`while`**.

3.6.7. Охарактеризуйте цикл **`do while`**.

3.6.8. Охарактеризуйте цикл **`for`**.

3.6.9. Опишите инструкции **`break`** и **`continue`**.

3.6.10. В каких случаях целесообразно применять оператор запятая?

3.6.11. Какой тип цикла лучше использовать в задаче табулирования функции? Почему?

3.6.12. Дайте характеристику флагам состояния формата, используемым в функции **`setf`** объекта **`cout`**.

4. ЛАБОРАТОРНАЯ РАБОТА №4
ОБРАБОТКА ОДНОМЕРНЫХ МАССИВОВ

4.1. Цель работы

Изучение особенностей представления и средств обработки одномерных массивов в языках *C*/*C*++ с учетом связи указателей и массивов.

Получение практических навыков реализации алгоритмов обработки одномерных массивов средствами языков *C*/*C*++.

Исследование особенностей обработки одномерных динамических массивов.

4.2. Варианты заданий

Требуется разработать программу, которая обеспечивает ввод с клавиатуры исходных данных, выполняет их обработку в соответствии с вариантом задания и выводит результаты обработки на экран. Методы обработки массивов (сортировка методами прямого обмена, прямого выбора и вставки), изложены в приложении Б. Варианты задания выбираются по указанию преподавателя.

Вариант 1

Даны натуральные числа n, p, q, a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество четных членов последовательности a_1, a_2, …, a_q. Определить значение наибольшего четного члена этой последовательности. Найти количество нечетных членов последовательности a_p, a_{p+1}, …, a_n. Определить значение наименьшего нечетного члена этой последовательности.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 2

Даны натуральные числа n, p, q, и действительные числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$. Найти:

— $\max\left[\cos a_1, \cos a_2, \ldots, \cos a_q\right]$;

— $\min\left[\sin a_p, \sin a_{p+1}, \ldots, \sin a_n\right]$;

— $\sqrt{a_p^2 + a_{p+2}^2 + \ldots + a_q^2}$.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 3

Даны натуральные числа n, p, q, и целые числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество отрицательных членов последовательности a_1, a_2, …, a_q, кратных 3 и 7, а также сумму положительных четных членов последовательности a_p, a_{p+1}, …, a_n. Определить значение наименьшего по модулю отрицательного члена последовательности a_p, a_{p+2}, …, a_q.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 4

Даны натуральные числа n, p, q, a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество членов последовательности a_1, a_2, …, a_q, удовлетворяющих условию $a_i = (a_{i-1} + a_{i+1})/2$, где $i = 2, 3, \ldots, (q-1)$, а также значение наименьшего из членов последовательности a_p, a_{p+1}, …, a_n, для которых выполняется условие $a_i = (a_{i-1} - a_{i+1})/2$, где $i = (p+1), (p+2), \ldots, (n-1)$.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 5

Даны натуральные числа n, p, q, и действительные числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$. Найти:

— $\max\left[\cos(\pi a_1), \cos(\pi a_2), \ldots, \cos(\pi a_q)\right]$;

— $\min\left[\,|a_p|, |a_{p+1}|, \ldots, |a_n|\,\right]$;

— $\sqrt{a_{p+1}^2 + a_{p+3}^2 + \ldots + a_q^2}$.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 6

Даны натуральные числа n, p, q, и целые числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Из членов последовательности a_p, a_{p+2}, …, a_q, выбрать такие числа, при подстановке которых в уравнение $5x^2 + a_i x - 7 = 0$, имеются положительные действительные корни. Здесь x — неизвестное, а $i = p, (p+2), \ldots, q$. Среди отобранных членов определить максимальный и минимальный элемент.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 7

Даны натуральные числа n, p, q, a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество членов последовательности a_1, a_2, …, a_q, удовлетворяющих условию $a_{i-1} \le a_i \ge a_{i+1}$, где $i = 2, 3, \ldots, (q-1)$, а также значение наибольшего из членов последовательности a_p, a_{p+1}, …, a_n, удовлетворяющих условию $a_{i-1} \le a_i \ge a_{i+1}$, где $i = (p+1), (p+2), \ldots, (n-1)$.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 8

Даны натуральные числа n, p, q, и действительные числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Члены последовательности a_p, a_{p+1}, …, a_q, имеющие индексы 1…4, заменить на 1, а имеющие индексы 5…9, заменить на 9. Найти те члены полученной последовательности a_1, a_2, …, a_n, значение синуса которых отрицательно.

Упорядочить члены исходной последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 9

Даны натуральные числа n, p, q, и целые числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Из членов последовательности a_{p+1}, a_{p+3}, …, a_q, выбрать такие числа, при подстановке которых в уравнение $x^2 + a_i x + 11 = 0$, имеются отрицательные действительные корни. Здесь x — неизвестное, а $i = (p+1), (p+3), \ldots, q$. Среди отобранных членов определить максимальный и минимальный элемент.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 10

Даны натуральные числа n, p, q, a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество членов последовательности a_1, a_2, …, a_q, удовлетворяющих условию $a_{i-1} \ge a_i \ge a_{i+1}$, где $i = 2, 3, \ldots, (q-1)$, а также значение наименьшего из членов последовательности a_p, a_{p+1}, …, a_n, удовлетворяющих условию $a_{i-1} \le a_i \le a_{i+1}$, где $i = (p+1), (p+2), \ldots, (n-1)$.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 11

Даны натуральные числа n, p, q, и действительные числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Среди членов последовательности a_1, a_2, …, a_q найти член, ближайший к какому-либо целому числу, а также член, который наиболее близок к нулю. Определить сумму положительных членов последовательности a_p, a_{p+1}, …, a_n, меньших π.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 12

Даны натуральные числа n, p, q, и целые числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество положительных членов последовательности a_1, a_2, …, a_q, кратных 5 и 11, а также сумму отрицательных нечетных членов последовательности a_p, a_{p+1}, …, a_n. Определить значение наибольшего по модулю отрицательного члена последовательности a_{p+1}, a_{p+3}, …, a_q.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 13

Даны натуральные числа n, p, q, a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество членов последовательности a_1, a_2, …, a_q, удовлетворяющих условию $2^i < a_i < i!$, где $i = 1, 2, \ldots, q$. Определить сумму нечетных членов последовательности a_p, a_{p+1}, …, a_n, которые имеют четные индексы.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 14

Даны натуральные числа n, p, q, и действительные числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Определить, какая из точек (a_p, a_q), (a_{p+1}, a_{q-1}), …, (a_q, a_p) расположена наиболее близко к точке $(-5, -3)$. Найти количество точек (a_1, a_n), (a_2, a_{n-1}), …, (a_n, a_1), лежащих вне круга с радиусом 6,23 и с центром, расположенным в начале координат.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 15

Даны натуральные числа n, p, q, и целые числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти сумму положительных четных членов последовательности a_1, a_2, …, a_q. Определить произведение членов последовательности a_p, a_{p+1}, …, a_n, удовлетворяющих условию $-1 \le a_i \le 1$, где $i = p, (p+1), \ldots, n$.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 16

Даны натуральные числа n, p, q, a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество членов последовательности a_1, a_2, …, a_q, удовлетворяющих условию $3^{i+1} < a_i < (i-1)!$, где $i = 1, 2, \ldots, q$. Определить сумму четных членов последовательности a_p, a_{p+1}, …, a_n, имеющих нечетные индексы.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 17

Даны натуральные числа n, p, q, и действительные числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Определить, какая из точек (a_p, a_q), (a_{p+1}, a_{q-1}), …, (a_q, a_p) расположена наиболее близко к точке $(3, 5)$. Найти количество точек (a_1, a_n), (a_2, a_{n-1}), …, (a_n, a_1), лежащих внутри квадрата со стороной равной $4{,}89$, расположенным симметрично относительно начала координат.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 18

Даны натуральные числа n, p, q, и целые числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество положительных членов последовательности a_1, a_2, …, a_q кратных 6 или 8, а также сумму отрицательных четных членов последовательности a_p, a_{p+1}, …, a_n. Определить значение наименьшего по модулю отрицательного члена последовательности a_{p+1}, a_{p+3}, …, a_q.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 19

Даны натуральные числа n, p, q, a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество четных и нечетных членов последовательности a_1, a_2, …, a_q. Определить значения наименьшего четного и наибольшего нечетно-

го члена последовательности a_p, a_{p+1}, …, a_n.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 20

Даны натуральные числа n, p, q, и действительные числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Определить, имеются ли в последовательности a_1, a_2, …, a_q три отрицательных члена, идущих подряд. Среди чисел a_p, a_{p+1}, …, a_n найти ближайшее к какому-нибудь целому числу.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 21

Даны натуральные числа n, p, q, и целые числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Из членов последовательности a_p, a_{p+1}, …, a_q, выбрать такие числа, при подстановке которых в уравнение $10x^2 + 3a_i x + 27 = 0$, имеются положительные действительные корни. Здесь x — неизвестное, а $i = p, (p+2), \ldots, q$. Среди отобранных членов определить максимальный и минимальный элемент.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 22

Даны натуральные числа n, p, q, a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество членов последовательности a_1, a_2, …, a_q удовлетворяющих условию $a_{i-1} \le a_i \ge a_{i+1}$, где $i = 2, 3, \ldots, (q-1)$, а также значение наибольшего из членов последовательности a_p, a_{p+1}, …, a_n, удовлетворяющих условию $a_{i-1} \ge a_i \le a_{i+1}$, где $i = (p+1), (p+2), \ldots, (n-1)$.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 23

Даны натуральные числа n, p, q, и действительные числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Определить, какая из точек (a_p, a_q), (a_{p+1}, a_{q-1}), …, (a_q, a_p) расположена наиболее близко к началу координат. Найти количество точек (a_p, a_q), (a_{p+1}, a_{q-1}), …, (a_q, a_p), лежащих по правую сторону от оси ординат.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 24

Даны натуральные числа n, p, q, и целые числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество отрицательных членов последовательности a_1, a_2, …, a_q, кратных 2 и 5, а также положительных четных членов последовательности a_p, a_{p+1}, …, a_n.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 25

Даны натуральные числа n, p, q, a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество членов последовательности a_1, a_2, …, a_q, удовлетворяющих условию $a_{i-1} \ge a_i \le a_{i+1}$, где $i = 2, 3, \ldots, (q-1)$, а также значение наименьшего из членов последовательности a_p, a_{p+1}, …, a_n, удовлетворяющих условию $a_{i-1} \le a_i \ge a_{i+1}$, где $i = (p+1), (p+2), \ldots, (n-1)$.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 26

Даны натуральные числа n, p, q, и действительные числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Определить, какая из точек (a_p, a_q), (a_{p+1}, a_{q-1}), …, (a_q, a_p) наиболее удалена от начала координат. Найти количество точек (a_p, a_q), (a_{p+1}, a_{q-1}), …, (a_q, a_p), лежащих по левую сторону от оси ординат.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 27

Даны натуральные числа n, p, q, a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти количество членов последовательности a_1, a_2, …, a_q, удовлетворяющих условию $2^i > a_i > i!$, где $i = 1, 2, \ldots, q$. Определить сумму четных членов последовательности a_p, a_{p+1}, …, a_n, имеющих четные индексы.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 28

Даны натуральные числа n, p, q, и действительные числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Определить, имеются ли в последовательности a_1, a_2, …, a_q три положительных члена, идущих подряд. Среди чисел a_p, a_{p+1}, …, a_n найти ближайшее к нулю.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 29

Даны натуральные числа n, p, q, и целые числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Найти сумму отрицательных членов последовательности a_1, a_2, …, a_q. Определить произведение тех членов последовательности a_p, a_{p+1}, …, a_n, которые по значению больше -5, но меньше 7.

Упорядочить члены последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 30

Даны натуральные числа n, p, q, и действительные числа a_1, a_2, …, a_n, причем $n \ge q > p \ge 1$.

Члены последовательности a_p, a_{p+1}, …, a_q, удовлетворяющие условию $-5 \le a_i \le 5$ заменить на 5, а члены, удовлетворяющие условию $10 \le a_i \le 15$, заменить на 10. В полученной последовательности a_1, a_2, …, a_n найти члены, значение косинуса которых положительно.

Упорядочить члены исходной последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом вставки.

4.3. Теоретические сведения

Для выполнения лабораторной работы необходимо изучить особенности представления и средства обработки одномерных массивов в языках *C*/*C*++ с учетом связи указателей и массивов, а также алгоритмы сортировки массивов методами прямого обмена, вставки и прямого выбора (см. приложение Б).

4.3.1. Одномерные массивы

Одномерный массив — это набор объектов одинакового типа, расположенных в последовательной группе ячеек памяти, доступ к которым осуществляется по индексу. В языке *C* одномерные массивы описываются следующим образом:

```
тип имя_массива[размер_массива];
```

В результате такого объявления компилятор отводит под массив память размером **sizeof(тип)*размер_массива** байтов. Операция (функция) **sizeof**, операндами (аргументами) которой могут являться константы, типы и переменные, в качестве результата возвращает количество байт, занимаемых ее операндом (аргументом).

Используя имя массива и индекс, можно обращаться к элементам массива: **имя_массива [индекс]**. Все элементы массива индексируются, начиная с нуля и заканчивая величиной, на единицу меньшей, чем размер массива, указанный при его описании. Например, если массив **data** объявлен как

```
double data[245];
```

то его **245** элементами типа **double** будут: **data[0]**, **data[1]**, ..., **data[244]**. Индекс массива может быть как целым числом, так и целочисленным выражением. Квадратные скобки, внутри которых записывается индекс массива, рассматриваются как оператор индексации. Оператор индексации имеет самый высокий приоритет (см. табл. 2.1 в лабораторной работе №2).

Элементам массива можно присваивать начальные значения (инициализировать их) при объявлении массива с помощью списка начальных значений, разделенных запятыми, заключенного в фигурные скобки:

```
int n[10]={32,27,64,18,95,14,90,70,60,37};
```

Если начальных значений меньше, чем элементов в массиве, то оставшиеся элементы автоматически получают нулевые начальные значения. Например, элементам массива **n** можно присвоить нулевые начальные значения с помощью объявления

```
int n[10]={0};
```

Если размер массива не указан в объявлении со списком инициализации, то количество элементов массива будет равно количеству элементов в списке начальных значений. Например, объявление

```
int n[]={1,2,3,4,5};
```

создает массив из пяти элементов.

4.3.2. Указатели

Указатели — это переменные, которые содержат в качестве своих значений адреса памяти. Их используют чаще всего при работе с динамической памятью — областью свободной памяти, в которой можно во время выполнения программы выделять место в соответствии с потребностями. Доступ к выделенным участкам динамической памяти, называемым динамическими перемен-

ными, производится только через указатели. В общем случае переменная типа указатель объявляется следующим образом:

```
тип *переменная_указатель;
```

Такая запись означает, что **переменная_указатель** является указателем на объект типа **тип**. Так, объявление

```
int *countPtr,count=5;
```

объявляет переменную **countPtr** типа **int *** (т.е. указатель на целое число) и читается как «**countPtr** является указателем на целое число» или «**countPtr** указывает на объект типа **int**». Однако переменная **count** объявлена как целое число, но не как указатель на целое число. Символ ***** в объявлении относится только к **countPtr**. Каждая переменная, объявляемая как указатель, должна иметь перед собой знак звездочки (*****).

Указатели должны инициализироваться либо при своем объявлении, либо с помощью оператора присваивания. Указатель может получить в качестве начального значения **0**, **NULL** или адрес. Указатель с начальными значениями **0** или **NULL** ни на что не указывает и часто используется как ограничитель в динамических структурах, **NULL** — это символическая константа, определенная в головном файле **<iostream.h>**, а также в нескольких головных файлах стандартной библиотеки языка *C*.

Унарный оператор адресации **&** возвращает адрес своего операнда. Например, в результате выполнения инструкции

```
countPtr=&count;
```

адрес переменной **count** будет запомнен в указателе **countPtr**.

Оператор *****, называемый оператором раскрытия ссылки или оператором разыменования (разадресации), возвращает значение объекта, на который указывает указатель. Например, инструкция

```
cout<<*countPtr<<endl ;
```

выводит на экран значение переменной **count**, а именно **5**.

Два оператора языка *C*++ **new** и **delete** позволяют управлять временем жизни какой-либо переменной. Переменная может быть создана в любой точке программы с помощью оператора **new** и затем при необходимости уничтожена с помощью оператора **delete**.

В результате выполнения инструкции

```
countPtr=new int;
```

переменной-указателю **countPtr** присваивается адрес начала участка памяти размером **sizeof(int)** байт. Память выделяется динамически из свободной

области. Оператор **delete** освобождает ранее занятую память, возвращая ее свободной области.

Наряду с операторами **new** и **delete** в языках *C*/*C*++ существуют две функции для захвата/освобождения памяти под динамическую переменную: **malloc** и **free** соответственно. Эти функции описаны в головном файле **<stdlib.h>**, а также в **<alloc.h>**. Функция **malloc (size)** запрашивает память размером **size** байт из динамической памяти и возвращает указатель на первый байт этого участка. Функция **malloc** всегда возвращает указатель на тип **void**, поэтому для получения адреса объекта определенного типа необходимо выполнить соответствующее преобразование типа, например:

```
int *x;
x=(int*)malloc(sizeof(int));
```

Здесь преобразование типа **(int*)** приводит значение, возвращаемое функцией **malloc** к адресу указателя на целочисленную переменную.

Память, выделенную в динамической памяти под динамическую переменную с помощью функции **malloc**, следует освобождать с помощью функции **free**. Единственным аргументом функции **free** является указатель, ссылающийся на освобождаемую память.

При работе с указателями возможны ошибки. Рассмотрим фрагмент программы

```
int *p=new int,  *q=new int;
*p=255,  *q=127;
p=q,  *p=63;
```

Присвоение **p=q** означает, что **p**, начиная с этого момента, указывает на ту же область памяти, что и **q**. Когда по адресу, содержащемуся в **p**, заносится значение **63** (для этого используется инструкция ***p=63**), значение, на которое ссылается **q**, также будет равно **63** (***q=63**). Кроме того, после присвоения **p=q** значение ***p=255** оказывается потерянным. Не существует доступа к этой области памяти. Она остается захваченной до конца выполнения программы. Такие явления называют утечкой памяти.

Если указатель является локальной переменной, т.е. описан в некотором блоке программы, как это имеет место во фрагменте

```
{
    char *ql=new char;
    *ql='c';
}
```

то при выходе из блока он перестанет существовать и память, на которую он ссылается, окажется недоступной. Эта память не помечается как свободная и поэтому не может быть использована в дальнейшем.

Еще один источник ошибок — неосвобождение памяти, запрошенной ра-

нее с помощью оператора **new** или функции **malloc**. Система не способна автоматически освобождать динамически выделенную память.

Неосвобождение памяти может остаться незамеченным в небольших программах, однако часто приводит к отказам в серьезных разработках и является плохим стилем программирования.

4.3.3. Массивы и указатели

Имя массива является константой-указателем и содержит адрес области памяти, которую занимает набор последовательных ячеек, соответствующих массиву.

Декларация

```
double a[10];
```

определяет массив **a**, состоящий из десяти последовательных объектов с именами **a[0]**, **a[1]**, ..., **a[9]**. Если **pa** есть указатель на **double**, т.е. определен как

```
double *pa;
```

то в результате присваивания

```
pa=a;
```

или

```
pa=&a[0];
```

pa будет указывать на нулевой элемент массива **a**; иначе говоря, **pa** будет содержать адрес элемента **a[0]**.

Если **pa** указывает на некоторый элемент массива, то **pa+1** по определению указывает на следующий элемент, **pa−1** указывает на предыдущий элемент, **pa+i** — на *i*-й элемент после **pa**, а **pa−i** — на *i*-й элемент перед **pa**. Таким образом, если **pa** указывает на **a[0]** (**pa==&a[0]**), то ***pa** есть содержимое **a[0]**, ***(pa+1)** есть содержимое **a[1]**, а ***(pa+i)** — содержимое **a[i]**. Сделанные замечания верны безотносительно к типу и размеру элементов массива **a**. Однако нельзя выходить за границы массива и тем самым ссылаться на несуществующие объекты.

Если **p** и **q** указывают на элементы одного и того же массива, то к ним можно применять операторы отношения **==**, **!=**, **<**, **<=**, **>**, **>=**. Например, отношение вида **p<q** истинно, если **p** указывает на «более ранний» элемент массива, чем **q**. Любой указатель всегда можно сравнить на равенство и неравенство с нулем.

Допускается также вычитание указателей. Например, если **p** и **q** ссылаются на элементы одного и того же массива и **p<q**, то **q−p+1** есть число элементов

от **p** до **q** включительно.

Если до начала работы программы количество элементов в массиве неизвестно, то следует использовать динамические массивы. Память под них выделяется во время выполнения программы с помощью оператора **new** или функции **malloc**, а освобождается с помощью оператора **delete** или функции **free** соответственно, например:

```
#include <stdlib.h>

main
{
    int n;
    cout<<" Введите количество элементов: ", cin>>n;
    int *a=new int[n];
    double *b=(double*)malloc(n*sizeof(double));
    // обработка массивов a и b
    ..........................
    delete []a;
    free(b);

    return 0;
}
```

4.3.4. Пример программы обработки динамических массивов

Рассмотрим пример работы с динамическими массивами. Ниже представлен текст программы, которая в целочисленном массиве, не все элементы которого одинаковы, заменяет набор элементов, расположенных между максимальным и минимальным элементами массива (максимальный и минимальный элементы не включаются в набор), на единственный элемент, равный сумме положительных элементов в заменяемом наборе. После этого выполняется сортировка полученного массива методом прямого обмена.

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

main()
// пример программы обработки динамических массивов
{
    int n,        // количество элементов в исходном массиве
        m;        // количество элементов в результирующем массиве
    int *a,       // массив (исходный и результирующий)
        *temp_a;  // временный (промежуточный) массив
    int i,j;      // счетчики циклов
    int imax,     // индекс максимального элемента массива
        imin;     // индекс минимального элемента массива
    int ibeg,     // индекс начала заменяемого набора элементов
        iend;     // индекс конца заменяемого набора элементов
    int s_pos;    // сумма полож. элементов в заменяемом наборе
    int temp;     // буфер для сортировки методом прямого обмена
```

```
clrscr ();

// формирование исходного массива
cout<<"Введите количество элементов массива: ", cin>>n;
a=new int[n];
cout<<"Введите "<<n<<" элемента(ов) массива: ";
for(i=0;i<n;i++) cin>>a[i];
cout<<"Исходный массив: "<<endl;
for(i=0;i<n;i++) cout<<"a["<<i<<"]="<<a[i]<<" ";
cout<<endl;

// поиск индексов макс. и мин. элементов массива
for(imax=imin=0,i=1;i<n;i++){
    if(a[i]>a[imax]) imax=i;
    if(a[i]<a[imin]) imin=i;
}
cout<<"Максимальный элемент: a["<<imax<<"]="<<a[imax]<<endl
    <<"Минимальный элемент: a["<<imin<<"]="<<a[imin]<<endl;

// поиск индексов начала и конца заменяемого набора эл-тов
if(imax<imin) ibeg=imax+1, iend=imin-1;
   else ibeg=imin+1, iend=imax-1;
cout<<"Заменяемый набор элементов: "<<endl;
for (i=ibeg;i<=iend;i++) cout<<"a["<<i<<"]="<<a[i] <<" ";
cout<<endl;

// расчет суммы полож. эл-тов заменяемого набора
for(i=ibeg,s_pos=0;i<=iend;i++)
    if(a[i]>0)  s_pos+=a[i];
cout<<"Сумма положительных элементов заменяемого набора:   "
    <<s_pos<<endl;

temp_a=a;      // адрес исходного массива
m=n+ibeg-iend; // кол-во эл-тов в результирующем массиве
a=new int[m];  // результирующий массив

/*-------- формирование результирующего массива ----------*/
// запись эл-тов, расположенных до начала заменяемого набора
for(i=0,j=0;i<ibeg;i++,j++) a[j]=temp_a[i];
// запись суммы полож. эл-тов заменяемого набора
a[j++]=s_pos;
// запись эл-тов, расположенных после конца заменяемого набора
for(i=iend+1;i<n;i++,j++) a[j]=temp_a[i];

// освобождение памяти из-под исходного массива
delete []temp_a;

// вывод на экран результирующего массива
cout<<" Результирующий массив: "<<endl ;
for(i=0;i<m;i++) cout<<"a["<<i<<"]="<<a[i]<<" ";
cout<<endl;
```

```
    // сортировка массива методом прямого обмена
    for(i=m-1;i;i--)
        for(j=0;j<i;j++)
            if(a[j]>a[j+1])
               temp=a[j],a[j]=a[j+1],a[j+1]=temp;

    // вывод на экран отсортированного массива
    cout<<" Отсортированный массив: "<<endl;
    for(i=0;i<m;i++) cout<<"a["<<i<<"]="<<a[i]<<" ";
    cout<<endl;

    cout<<"Нажмите любую клавишу...";
    getch();
    delete []a; // освобождение памяти

    return 0;
}
```

Исходный целочисленный массив формируется динамически, т.е. во время выполнения программы, при этом используется значение размера массива, введенное пользователем. После того, как введены элементы массива, происходит поиск индексов максимального и минимального элементов **imax** и **imin** соответственно.

Так как порядок расположения элементов в массиве заранее не известен, то сначала может следовать как максимальный (**imax<imin**), так и минимальный элемент (**imin<imax**). С учетом этого обстоятельства осуществляется поиск индексов начала и конца заменяемого набора **ibeg** и **iend** соответственно. Далее производится расчет суммы положительных элементов заменяемого набора, т.е. тех элементов исходного массива, индексы которых лежат в диапазоне от **ibeg** до **iend** включительно.

Затем динамически происходит создание результирующего массива. При этом адрес исходного массива запоминается, чтобы после того, как формирование результирующего массива будет окончено, освободить память, которую занимает исходный массив. Для вычисления размера результирующего массива необходимо из размера исходного массива (**n**) вычесть количество элементов в заменяемом наборе (**iend-ibeg+1**) и добавить единицу, соответствующую элементу, заменяющему набор:

```
n-(iend-ibeg+1)+1==n+ibeg-iend
```

Далее происходит формирование результирующего массива. Следует обратить внимание, что при копировании элементов из исходного массива в результирующий массив используются разные счетчики цикла, так как копируемым элементам соответствуют различные индексы в соответствующих массивах. После того как результирующий массив сформирован, память из-под исходного массива освобождается. В конце результирующий массив сортируется методом прямого обмена.

4.4. Порядок выполнения работы

Вариант задания определяется преподавателем. Перед выполнением работы необходимо ознакомится с разделами 4.1 — 4.3 данных методических указаний. Индивидуальные задания для выполнения работы см. в разделе 4.2.

4.4.1. Внимательно изучить постановку задачи.

4.4.2. Разработать схему алгоритма решения задачи.

4.4.3. Написать программу, решающую поставленную задачу. Программа обязательно должна содержать комментарии.

4.4.4. Разработать тестовые примеры.

4.4.5. Выполнить тестирование и отладку написанной программы, используя разработанные тестовые примеры.

4.4.6. Получить результаты работы программы.

4.4.7. Подготовить отчет по работе.

4.5. Оформление отчета

4.5.1. Сформулировать цель работы.

4.5.2. Произвести постановку задачи и текст индивидуального задания.

4.5.3. Привести краткие теоретические сведения.

4.5.4. Изобразить схему алгоритма решения задачи.

4.5.5. Привести текст программы.

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

4.5.7. Привести результаты тестирования и отладки программы.

4.5.8. Привести результаты выполнения программы.

4.5.9. Сделать выводы по работе.

4.6. Контрольные вопросы

4.6.1. Как в языках *C*/*C*++ описываются одномерные массивы?

4.6.2. Каковы особенности инициализации массива при его объявлении?

4.6.3. Как объявить переменную типа указатель?

4.6.4. Охарактеризуйте оператор адресации **`&`** и оператор раскрытия ссылки **`*`**.

4.6.5. Охарактеризуйте операторы **`new`** и **`delete`**.

4.6.6. Опишите функции **`malloc`** и **`free`**.

4.6.7. Перечислите наиболее распространенные ошибки, связанные с использованием указателей.

4.6.8. Охарактеризуйте связь между массивами и указателями.

4.6.9. Какие действия можно выполнять над указателями?

4.6.10. Что такое динамические массивы?

4.6.11. Опишите алгоритмы сортировки массивов методами прямого обмена, вставки и прямого выбора.

5. ЛАБОРАТОРНАЯ РАБОТА №5
ОБРАБОТКА ДВУМЕРНЫХ МАССИВОВ

5.1. Цель работы

Изучить основные принципы работы с массивами в языках *C*/*C*++.
Исследовать способы передачи параметров в функции.

5.2. Варианты заданий

Вариант задания выбирается по указанию преподавателя.

Каждый пункт нижеприведенного задания оформить в виде функции. Все необходимые данные для функций должны передаваться им в качестве параметров. Ввод-вывод данных и результатов также организовать с помощью соответствующих функций.

Вариант 1

Дана целочисленная прямоугольная матрица. Определить:

— количество строк, не содержащих ни одного нулевого элемента;

— максимальное из чисел, встречающихся в заданной матрице более одного раза.

Вариант 2

Дана целочисленная прямоугольная матрица. Определить:

— количество столбцов, не содержащих ни одного нулевого элемента.

— суммы положительных четных элементов в строках матрицы, а затем переставить строки заданной матрицы, расположив их в соответствии с ростом вычисленных сумм.

Вариант 3

Дана целочисленная прямоугольная матрица. Определить:

— количество столбцов, содержащих хотя бы один нулевой элемент;

— номер строки, в которой находится самая длинная серия одинаковых элементов.

Вариант 4

Дана целочисленная квадратная матрица. Определить:

— произведение элементов в тех строках, которые не содержат отрицательных элементов;

— максимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

Вариант 5

Дана целочисленная квадратная матрица. Определить:

— сумму элементов в тех столбцах, которые не содержат отрицательных

элементов;

— наименьшую сумму модулей элементов диагоналей, параллельных побочной диагонали матрицы.

Вариант 6

Дана целочисленная прямоугольная матрица. Определить:

— сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент;

— номера строк и столбцов всех седловых точек матрицы.

Примечание. Матрица A имеет седловую точку a_{ij}, если элемент матрицы a_{ij} является минимальным в *i*-й строке и максимальным *j*-м столбце.

Вариант 7

Для заданной матрицы найти:

— значения k, при которых k-я строка матрицы совпадает с k-м столбцом;

— сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.

Вариант 8

Дана целочисленная прямоугольная матрица. Определить:

— суммы модулей отрицательных нечетных элементов в столбцах заданной матрицы, а затем переставить столбцы в соответствии с ростом вычисленных сумм;

— сумму элементов в тех столбцах, которые содержат хотя бы один отрицательный элемент.

Вариант 9

Провести сглаживание заданной вещественной матрицы размером 10 на 10. В результате операции сглаживания матрицы должна получиться матрица того же размера, каждый элемент которой получается как среднее арифметическое имеющихся соседей соответствующего элемента исходной матрицы.

В сглаженной матрице найти сумму модулей элементов, расположенных ниже главной диагонали.

Примечание. Соседними элементами для a_{ij} в матрице A считать элементы a_{mn}, у которых $i-1 \le m \le i+1$ и $j-1 \le n \le j+1$.

Вариант 10

Подсчитать количество локальных минимумов заданной матрицы размером 10 на 10.

Найти сумму модулей элементов, расположенных выше главной диагонали.

Примечание. Элемент матрицы называется локальным минимумом, если

он строго меньше всех имеющихся у него соседних элементов. Соседними элементами для a_{ij} в матрице A считать элементы a_{mn}, у которых $i-1 \le m \le i+1$ и $j-1 \le n \le j+1$.

Вариант 11

Подсчитать количество локальных максимумов заданной матрицы размером 10 на 10.

Найти сумму модулей элементов, расположенных ниже главной диагонали.

Примечание. Элемент матрицы называется локальным максимумом, если он строго больше всех имеющихся у него соседних элементов. Соседними элементами для a_{ij} в матрице A считать элементы a_{mn}, у которых $i-1 \le m \le i+1$ и $j-1 \le n \le j+1$.

Вариант 12

Уплотнить заданную матрицу, удаляя из неё строки и столбцы, заполненные нулями.

В полученной матрице найти номер первой из строк, содержащих хотя бы один положительный элемент.

Вариант 13

Осуществить циклический сдвиг всех элементов прямоугольной матрицы на n элементов вправо, либо вниз (направление выбирается по указанию оператора), n может быть больше количества элементов в строке или столбце.

Вариант 14

Осуществить циклический сдвиг элементов квадратной матрицы на k элементов таким образом, чтобы элементы 1-ой строки сдвигались слева направо в последний столбец, в последнем столбце — сверху вниз, из него — в последнюю строку справа налево, из неё — в 1-ый столбец снизу вверх, из 1-го столбца — в 1-ую строку. Для остальных элементов матрицы сдвиг выполнить аналогичным образом.

Вариант 15

Дана целочисленная прямоугольная матрица. Определить:

— номер первого из столбцов, содержащих хотя бы один нулевой элемент;

— суммы отрицательных четных элементов в каждой строке, а затем переставить строки заданной матрицы, расположив в соответствии с убыванием вычисленных сумм.

Вариант 16

Упорядочить строки целочисленной прямоугольной матрицы по возрастанию количества одинаковых элементов в каждой строке.

Найти номер первого из столбцов, не содержащих ни одного отрицательного элемента.

Вариант 17

Переставить строки и столбцы квадратной вещественной матрицы таким образом, чтобы её максимальный элемент находился в левом верхнем углу матрицы, следующий по величине — в позиции $(2, 2)$, следующий — в позиции $(3, 3)$ и т.д., заполнив таким образом всю главную диагональ матрицы.

В полученной матрице найти номер первой из строк, не содержащих ни одного положительного элемента.

Вариант 18

Дана целочисленная прямоугольная матрица. Определить:

— количество строк, содержащих хотя бы один нулевой элемент;

— номер столбца, в котором находится самая длинная серия одинаковых элементов.

Вариант 19

Дана целочисленная квадратная матрица. Определить:

— сумму элементов в тех строках, которые не содержат отрицательных элементов;

— минимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

Вариант 20

Дана целочисленная прямоугольная матрица. Определить:

— количество отрицательных элементов в тех строках, которые содержат хотя бы один нулевой элемент;

— номера строк и столбцов всех седловых точек матрицы.

Примечание. Матрица A имеет седловую точку a_{ij}, если элемент a_{ij} является минимальным элементом в *i*-й строке и максимальным в *j*-м столбце.

Вариант 21

Дана прямоугольная матрица. Выполнить:

— зеркальную перестановку столбцов матрицы относительно вертикальной оси;

— зеркальную перестановку строк матрицы, полученной в результате выполнения предыдущего пункта, относительно горизонтальной оси.

Вариант 22

Дана прямоугольная матрица. Выполнить:

— зеркальную перестановку строк матрицы относительно горизонтальной оси;

— зеркальную перестановку столбцов матрицы, полученной в результате выполнения предыдущего пункта, относительно вертикальной оси.

Вариант 23

Дана прямоугольная матрица. Выполнить:
— поиск позиций всех локальных минимумов матрицы;
— поиск позиций всех локальных максимумов матрицы.

Примечание. Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседних элементов, а локальным максимумом, если он строго больше всех имеющихся у него соседних элементов. Соседними элементами для a_{ij} в матрице A считать элементы a_{mn}, у которых $i-1 \le m \le i+1$ и $j-1 \le n \le j+1$.

Вариант 24

Дана прямоугольная матрица. Осуществить поиск позиций максимального и минимального элементов матрицы. Найти среднее арифметическое всех элементов матрицы.

Примечание. Позицией элемента матрицы называется упорядоченная пара индексов (i, j), где i — номер строки элемента, j — номер столбца.

Вариант 25

Определить, является ли заданная квадратная матрица симметрической. Найти сумму положительных и сумму отрицательных элементов матрицы.

Примечание. Матрица A является симметрической, если $A^{\mathrm{T}} = A$.

Вариант 26

Определить, является ли заданная квадратная матрица кососимметрической. Найти сумму нулевых элементов матрицы.

Примечание. Матрица A является кососимметрической (антисимметрической), если $A^{\mathrm{T}} = -A$.

Вариант 27

Упорядочить строки прямоугольной целочисленной матрицы по возрастанию сумм их элементов методом прямого обмена.

Вариант 28

Упорядочить столбцы прямоугольной целочисленной матрицы по возрастанию сумм их элементов методом прямого обмена.

Вариант 29

Упорядочить строки прямоугольной целочисленной матрицы по возрастанию сумм их элементов методом вставки.

Вариант 30

Упорядочить столбцы прямоугольной целочисленной матрицы по возрастанию сумм их элементов методом вставки.

5.3. Теоретические сведения

5.3.1. Описание и обработка двумерных массивов

Двумерный массив в *C*/*C*++ представляется как массив, состоящий из массивов. Для этого при описании массива в квадратных скобках указывают вторую размерность:

```
int a[3][5]; // матрица 3x5
```

Такой массив хранится в непрерывной области памяти по строкам (рис. 5.1).

a_{00} a_{01} a_{02} a_{03} a_{04}	a_{10} a_{11} a_{12} a_{13} a_{14}	a_{20} a_{21} a_{22} a_{23} a_{24}
← 0-ая строка →	← 1-ая строка →	← 2-ая строка →

Рис.5.1 — Хранение двумерного массива в памяти ЭВМ

В памяти сначала располагается одномерный массив **a[0]**, т.е. нулевая строка, затем массив **a[1]** — первая строка и т.д.

Для доступа к отдельному элементу массива применяется конструкция вида **a[i][j]**, где **i** — номер строки, **j** — номер столбца. Каждый индекс изменяет свои значения, начиная с нуля. Можно обратиться к элементу массива и с помощью указателей, например, ***(*(a+i)+j)** или ***(a[i]+j)**. Следует обратить внимание на то, что ***(a+i)** должно быть указателем.

При описании массивов можно задавать начальные значения его элементов. Элементы массива инициализируются в порядке их расположения в памяти:

```
int a[3][5]={1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5};
```

Если количество значений в фигурных скобках превышает количество элементов в массиве, то выдается сообщение об ошибке. Если значений меньше, то оставшиеся элементы массива инициализируются значениями по умолчанию (для основных типов это ноль). Можно задать начальные значения не для всех элементов массива. Для этого список значений для каждой строки массива заключается в дополнительные фигурные скобки:

```
int a[3][5]={{1,2}, {1,2}, {1,2}};
```

Здесь нулевой и первый столбцы матрицы заполняются единицами и двойками. Остальные элементы массива обнуляются.

При явном задании хотя бы одного инициализатора для каждой строки количество строк в описании массива можно не указывать:

```
int a[3][5]={{1, 1, 1, 1, 1}, {2, 2, 2}, {3, 3}};
```

Напишем программу, которая для целочисленной матрицы, содержащей 10 строк и 20 столбцов, определяет среднее арифметическое ее элементов и количество положительных элементов в каждой строке. Алгоритм решения этой задачи очевиден. Для вычисления среднего арифметического элементов массива требуется найти их общую сумму, после чего разделить ее на количество элементов. Порядок просмотра массива роли не играет. Определение положительных элементов каждой строки требует просмотра матрицы по строкам. Обе величины вычисляются при одном просмотре матрицы.

```
#include <iostream.h>
void main(){
     const int nrow=10, ncol=20;
     int a[nrow][ncol];
     int i,j;

     cout<<"Введите элементы массива: "<<endl;
     for(i=0;i<nrow;i++)
         for(j=0;j<ncol;j++)
             cin>>a[i][j];
     int n;      // счетчик положительных элементов
     float s=0; // сумма элементов
     for(i=0;i<nrow;i++){
         n=0;
         for(j=0;j<ncol;j++) {
             s+=a[i][j];
             if(a[i][j]>0)  n++;
         }
         cout<<"В строке "<<i<<" кол-во положительных элементов: "
             <<n<<endl;
     }
     s/=nrow*ncol;
     cout<<"Среднее арифметическое равно: "<<s<<endl;
}
```

Здесь размерности массива заданы именованными константами **nrow** и **ncol**, что позволяет легко их изменять. Для упрощения отладки рекомендуется задать небольшие значения этих констант. При вычислении количества положительных элементов для каждой строки выполняются однотипные действия: обнуление счетчика **n**, просмотр каждого элемента строки и сравнение его с нулем, при необходимости увеличение счетчика на единицу, а после окончания вычислений — вывод результирующего значения счетчика.

Рекомендуется после ввода матрицы выполнять её контрольный вывод на экран. Для того чтобы элементы матрицы располагались один за другим, следует использовать манипулятор **setw()**, устанавливающий ширину поля для выводимого значения. Для использования манипулятора следует подключить заголовочный файл **<iomanip.h>** и дополнить программу следующим кодом:

```
#include <iomanip.h>
.............................
for(i=0;i<nrow;i++){
    for(j=0;j<ncol;j++)
        cout<<setw(6)<<a[i][j];
    cout<<endl;
}
.............................
```

Здесь после вывода каждой строки матрицы выводится символ перевода строки **endl**. Необходимый форматированный вывод матрицы можно также выполнить с помощью функции **printf**.

5.3.2. Динамические массивы

В динамической области памяти можно создавать двумерные массивы с помощью операции **new** или функции **malloc**. Рассмотрим первый способ. При выделении памяти сразу под весь массив количество строк можно задавать с помощью переменной, а количество столбцов должно быть определено с помощью константы. После слова **new** записывается тип элементов массива, а затем в квадратных скобках его размерности, например:

```
int n;
const int m=5;
cin>>n;
int (*a)[m]=new int[n][m];        // 1
int **b=(int **) new int[n][m];   // 2
```

Здесь в строке **1** адрес начала выделенного с помощью **new** участка памяти присваивается переменной **a**, определенной как указатель на массив из **m** элементов типа **int**. Именно такой тип значения возвращает в данном случае операция **new**. Скобки для **(*a)** необходимы, поскольку без них конструкция интерпретировалась бы как массив из указателей.

В строке **2** адрес начала выделенного участка памяти присваивается переменной **b**, которая описана как указатель на указатель типа **int**. Поэтому требуется выполнить преобразование типа **(int **)**.

Обращение к элементам динамических массивов производится точно так же, как к элементам статических массивов, например, **a[i][j]**.

Для того чтобы понять, отчего динамические массивы описываются так, напомним, что доступ к элементу двумерного массива можно выполнить с помощью конструкции ***(*(a+i)+j)**. Поскольку здесь применяются две операции раскрытия ссылки, то переменная, в которой хранится адрес начала массива, должна быть указателем на указатель.

Когда обе размерности массива задаются на этапе выполнения программы, то память под двумерный массив можно выделить следующим образом:

```
int nrow,ncol;
cout<<"Введите количество строк и столбцов: "
cin>>nrow>>ncol;
int **a=new int *[nrow]; // 1
for(int i=0;i<nrow;i++)  // 2
    a[i]=new int[ncol];  // 3
```

В строке **1** объявляется переменная **a** типа указатель на указатель типа **int** и выделяется память под массив, каждый элемент которого есть указатель на строку матрицы. Всего элементов **nrow**. В строке **2** организуется цикл для выделения памяти под строки. В строке **3** каждому указателю на строку присваивается адрес начала области памяти, достаточной для хранения **ncol** элементов типа **int**.

5.3.3. Функции

Функция — это группа операторов, вызываемая по имени и возвращающая в точку вызова предписанное значение. Формат простейшего заголовка функции записывается следующим образом

```
<тип> <имя>(<список формальных параметров>);
```

Например, заголовок функции **main** обычно имеет вид

```
int main();
```

Это означает, что никаких параметров функции не передается, а возвращает она одно значение типа **int**. Функция может и не возвращать значения, в этом случае должен быть указан тип **void**.

Имена формальных параметров при задании прототипа функции можно не указывать. Достаточно указать их тип:

```
double sin(double);
```

Здесь записано, что функция имеет имя **sin**, вычисляет значение синуса типа **double** и для этого ей надо передать аргумент типа **double**.

Хороший стиль программирования требует, чтобы все, что передается в функцию и обратно, отражалось в ее заголовке.

Заголовок задает объявление функции. Определение функции, кроме заголовка, включает её тело, т.е. операторы, которые выполняются при вызове функции, например:

```
int sum(int a,int b){
    return a+b; // тело функции
}
```

Тело функции представляет собой блок, заключенный в фигурные скобки. Для возврата результата, вычисленного в функции, служит оператор **return**.

После него указывается выражение, значение которого и передается в точку вызова функции. Функция может иметь несколько операторов возврата.

Для вызова функции указывают ее имя, а также передают ей значения фактических параметров:

```
<имя функции>(<список фактических параметров>);
```

Порядок следования аргументов в списке фактических параметров и их типы должны строго соответствовать списку формальных параметров. Пример обращения к определенной выше функции **sum**:

```
int summa,a=5;
summa=sum(a,5);
```

Рассмотрим механизм передачи параметров в функцию. Он весьма прост. Параметры передаются в функцию как параметры-значения. Иными словами, функция работает с копией значений, которые ей передаются. Кстати, именно поэтому на месте таких параметров можно при вызове задавать выражения, например:

```
summa=sum(a*10+5,a+3);
```

Для передачи в функцию параметров способом «параметр-переменная» в языке *C* используют указатели. Определим функцию **swap**, осуществляющую обмен значениями двух переменных.

```
void swap(int *x,int *y){
     int temp;
     temp=*x;
     *x=*y;
     *y=temp;
}
```

Здесь параметры **x** и **y** являются указателями и позволяют функции осуществлять доступ к ячейкам памяти вызвавшей ее программы и менять их значения местами. Чтобы функция **swap** выполняла желаемые действия, необходимо при вызове на место параметров **x** и **y** подставить адреса соответствующих ячеек памяти. Для этого используется операция взятия адреса **&**:

```
swap(&a,&b);
```

В этом случае функция **swap** поменяет местами значения переменных **a** и **b**. Тип функции **void** используется, поскольку функция **swap** не возвращает никаких значений.

В языке *C++* для передачи параметров способом «параметр-переменная» можно использовать ссылочные переменные. Ссылка — это переменная, которая ассоциирована с другой переменной и содержит ее адрес:

```
int ivar=1234;
int &iref=ivar; // ссылка iref
```

Здесь **iref** — это ссылка, которой присваивается адрес переменной **ivar**. Ссылки должны инициализироваться при их объявлении.

Не существует операторов, непосредственно производящих действия над ссылками. Все операции выполняются над ассоциированными значениями. Так, оператор **iref++** выполняет инкремент значения **ivar**.

Сами по себе ссылки применяются редко. Их обычно используют в качестве параметров функций. Так, функцию **swap** можно переписать в виде

```
void swap(int &x,int &y){
     int temp;
     temp=x;
     x=y;
     y=temp;
}
```

При этом упрощается вызов функции:

```
swap(a,b);
```

В этом случае **x** будет ссылаться на **a**, а **y** — на **b**, и функция **swap** выполнит обмен значениями **a** и **b**.

При определении функции входные данные обычно передаются по значению, а результаты ее работы — по ссылке или указателю.

Следует иметь ввиду, что возвращение ссылки или указателя из функции может привести к проблемам, если переменная, на которую делается ссылка, вышла из области видимости, например:

```
int & func()
{
    int x;
    x=5;
    return x;
}
```

В этом случае попытка возвратить ссылку на локальную переменную приведет к ошибке.

Эффективность передачи в функцию адреса вместо самой переменной ощутима и в скорости работы, особенно если используются массивы.

Если в функцию требуется передать достаточно большой объект, однако его модификация не предусматривается, то используется константный указатель или ссылка:

```
const int * func(const int * number); // указатель
const int & func(const int & number); // ссылка
```

Здесь функция **func** принимает и возвращает указатель или ссылку на константный объект типа **int**. Любая попытка модифицировать такой объект внутри функции вызовет сообщение компилятора об ошибке.

В заключение рассмотрим передачу массивов функциям. Пусть требуется написать функцию, суммирующую элементы массива. Предположим, что имеются следующие описания:

```
#define MAX 100;
int a[MAX];
```

Тогда можно объявить функцию со следующим прототипом:

```
int SumOfA(int a[MAX]);
```

Однако будет лучше оставить квадратные скобки пустыми и добавить второй аргумент, определяющий размер массива:

```
int SumOfA(int a[],int n);
```

Необходимо помнить, что в этом случае реально в функцию передается указатель на тип элемента массива. Таким образом, изменение любого элемента массива внутри функции повлияет и на оригинал. Поэтому лучше сразу передать в функцию указатель на нулевой элемент константного массива, например:

```
int SumOfA(const int *a,int n);
```

Аналогичным образом поступают и при передаче двумерных массивов в функцию. При передаче двумерного массива в функцию в списке формальных параметров обычно указывают только количество столбцов массива. Количество строк передают в виде дополнительного параметра:

```
int SumOfMatrix (int matrix[][10],int nrow);
```

Поскольку доступ к двумерному массиву можно получить с помощью указателя на указатель, то прототип функции можно объявить и так

```
int SumOfMatrix(int **matrix,int nrow,int ncol);
```

Здесь **nrow** — количество строк матрицы, а **ncol** — количество столбцов.

5.3.4. Обработка матриц с помощью функций

Пусть требуется написать программу, которая упорядочивает строки прямоугольной целочисленной матрицы по возрастанию сумм их элементов. Начинать решение задачи необходимо с четкого описания того, что является её исходным данными и результатами, и каким образом они будут представлены в программе.

Исходные данные. Размерность матрицы будем задавать с помощью сим-

волических констант. В качестве типа значений элементов матрицы будем использовать тип **int**.

Результаты. Результатом является та же матрица, но упорядоченная. Это значит, что не следует отводить для результата новую область памяти, а необходимо упорядочить матрицу на том же месте.

Промежуточные величины. Кроме конечных результатов, в любой программе есть промежуточные, а также служебные переменные. Следует выбрать их тип и способ хранения. Очевидно, что если требуется упорядочить матрицу по возрастанию сумм элементов ее строк, эти суммы необходимо вычислить и где-то хранить. Поскольку все они потребуются при упорядочивании, их запишем в массив, количество элементов которого соответствует количеству строк матрицы, а *i*-ый элемент содержит сумму элементов *i*-ой строки. Сумма элементов строки может превысить диапазон значений, допустимых для отдельного элемента строки, поэтому для элементов этого массива необходимо выбрать тип **long**.

После того, как выбраны структуры данных, можно приступить разработке алгоритма, поскольку алгоритм зависит от того, каким образом представлены данные.

Алгоритм работы программы. Для сортировки строк воспользуемся алгоритмом прямого выбора. При этом будем одновременно с перестановкой элементов массива выполнять обмен двух строк матрицы. Вычисления в данном случае состоят из 2-х шагов: формирование сумм элементов каждой строки и упорядочивание матрицы. Упорядочивание состоит в выборе наименьшего элемента и обмена с первым из рассматриваемых. Разработанные алгоритмы полезно представить в виде блок-схемы. Функционально завершение части алгоритма отделяется пустой строкой и/или комментарием. Не следует стремиться написать всю программу сразу. Сначала рекомендуется написать и отладить фрагмент, содержащий ввод исходных данных. Затем целесообразно перейти к следующему фрагменту алгоритма.

Ниже приведен текст программы сортировки строк матрицы по возрастанию сумм элементов.

```
#include <iostream.h>  // подключение библиотеки стандартных
                       // объектов и операций с потоками
                       // ввода-вывода средствами языка C++
#include <iomanip.h>   // подключение библиотеки средств
                       // манипулирования потоками

// прототипы функций

// функция, суммирующая элементы строк
void SummaStrok(int **a,int m,int n,long int *v);
// функция, выполняющая вывод матрицы и суммы элементов в строках
void Vyvod(int **a,int m,int n);
// функция, выполняющая сортировку строк
void Sort(int **a,int m,int n,long int *v);
```

```
int main ()
{
    int m, n, i, j;
    cout<<"Введите количество строк и столбцов матрицы: ";
    cin>>m>>n;
    int **a=new int *[m];          // выделение памяти
    for(i=0;i<m;i++)               // под матрицу
        a[i]=new int[n];
    for(i=0;i<m;i++)               // ввод матрицы
        for(j=0;j<n;j++)
            cin>>a[i][j];
    long int *v=new long int[m]; // вспомогательный массив
    SummaStrok(a,m,n,v);           // суммирование элементов строк
    Vyvod(a,m,n);                  // контрольная печать
    Sort(a,m,n,v);                 // сортировка
    Vyvod(a,m,n);                  // вывод результатов
    delete []v;
    delete []a;
    return 0;
}

void SummaStrok(int **a,int m,int n,long int *v){
    int i,j;
    for(i=0;i<m;i++){
        v[i]=0;
        for(j=0;j<n;j++)
            v[i]+=a[i][j];
    }
}

void Vyvod(int **a,int m,int n){
    int i,j;
    for(i=0;i<m;i++){
        for(j=0;j<n;j++)
            cout<<setw(4)<<a[i][j]<<" ";
        cout<<endl;
    }
}

// сортировка строк матрицы методом прямого выбора
void Sort(int **a,int m,int n,long int *v){
    long buf_sum;
    int min,buf_a;
    int i,j;
    for(i=0;i<m-1;i++){
        min=i;
        for(j=i+1;j<m;j++) // поиск минимального элемента
            if(v[j]<v[min]) min=j;
        buf_sum=v[i];      // обмен значений v
        v[i]=v[min];
        v[min]=buf_sum;
        for(j=0;j<n;j++){  // перестановка строк матрицы a
            buf_a=a[i][j];
```

```
                a[i][j]=a[min][j];
                a[min][j]=buf_a;
            }
        }
}
```

В функции **Sort** используются две буферные переменные: **buf_sum**, через которую осуществляется обмен двух значений сумм (имеет такой же тип, что и сумма), **buf_a** для обмена значений элементов массива (того же типа, что и элементы массива).

5.4. Порядок выполнения работы

Вариант задания выбирается по указанию преподавателя. Перед выполнением работы необходимо ознакомится с разделами 5.1 — 5.3 данных методических указаний. Индивидуальные задания для выполнения работы см. в разделе 5.2.

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

5.4.2. Разработать алгоритм решения задачи в соответствии с вариантом.

5.4.3. Составить программу, оформив ввод данных, вывод результатов, необходимые вычисления в виде функций.

5.4.4. Разработать тестовые примеры, которые проверяют все ветви алгоритма и возможные диапазоны данных. В качестве тестового примера рекомендуется ввести значения элементов массива, близкие к максимально возможным. Дополнительно рекомендуется проверить работу программы, когда массив состоит из одной или двух строк (столбцов), поскольку многие ошибки при написании циклов связаны с неверным указанием их граничных значений.

5.4.5. Выполнить тестирование и отладку программы. Получить результаты для всех вариантов тестовых примеров.

5.4.6. Получить результаты работы программы.

5.4.7. Подготовить отчет по работе.

5.5. Оформление отчета

5.5.1. Сформулировать цель работы.

5.5.2. Произвести постановку задачи и текст индивидуального задания.

5.5.3. Привести краткие теоретические сведения.

5.5.4. Изобразить схему алгоритма решения задачи.

5.5.5. Привести текст программы.

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

5.5.7. Привести результаты тестирования и отладки программы.

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

5.5.9. Сделать выводы по работе.

5.6. Контрольные вопросы

5.6.1. Как в языках *C*/*C*++ представляются двумерные массивы?

5.6.2. Как осуществляется доступ к элементам двумерного массива?

5.6.3. Каким образом осуществляется инициализация двумерных массивов?

5.6.4. Каковы особенности работы с динамическими двумерными массивами в языках *C*/*C* ++?

5.6.5. Приведите формат заголовка функции.

5.6.6. Что является телом функции?

5.6.7. В чем состоит механизм передачи параметров в функцию?

5.6.8. Что такое ссылочные переменные? В каких случаях их целесообразно использовать?

5.6.9. Что такое аргументы по умолчанию?

5.6.10. Как осуществляется передача массивов в функции?

6. ЛАБОРАТОРНАЯ РАБОТА №6
ПОСТРОЕНИЕ ГРАФИКОВ ФУНКЦИЙ СРЕДСТВАМИ ЯЗЫКА *C*

6.1. Цель работы

Изучение основных возможностей графической библиотеки языка *Borland C*. Приобретение практических навыков по разработке и отладке программ, использующих графические режимы дисплеев.

6.2. Варианты заданий

Вариант 1

Исследовать область определения и область значений функции $y = \sin(x)/x$ и построить её график.

Вариант 2

Построить график функции $y = 6x^2 + 3x$.

Вариант 3

Построить график функции $y = x^3 + 2x^2 + x$.

Вариант 4

Построить график функции $y = x^5 - 3x^2$.

Вариант 5

Построить график функции $y = \sin|x|$.

Вариант 6

Построить график функции $y = \cos(x-1) + |x|$.

Вариант 7

Исследовать область определения и область значений функции $y = \frac{1}{x}$ и построить её график.

Вариант 8

Исследовать область определения и область значений функции $y = 3 + 2/x + 2/x^2$ и построить её график.

Вариант 9

Исследовать область определения и область значений функции $y = (x+3)/(x-2)$ и построить её график.

Вариант 10

Исследовать область определения и область значений функции $y = 1/(3x^2 + 2x + 1)$ и построить её график.

Вариант 11

Исследовать область определения и область значений функции $y = 3 - 2/x - 2/x^2$ и построить её график.

Вариант 12

Исследовать область определения и область значений функции $y = 1/(x^2 - 2x + 1)$ и построить её график.

Вариант 13

Исследовать область определения и область значений функции $y = 1/(x^2 + 3x - 1)$ и построить её график.

Вариант 14

Построить окружность радиуса r с центром в начале координат, заданную параметрически:

$$\begin{cases} x = r\cos t; \\ y = r\sin t, \end{cases}$$

где $t \in [0, 2\pi)$.

Вариант 15

Построить эллипс с большой и малой полуосями, равными соответственно r_1 и r_2, расположенными параллельно осям координат, заданный параметрически:

$$\begin{cases} x = r_1\cos t; \\ y = r_1\sin t, \end{cases}$$

где $t \in [0, 2\pi)$.

Вариант 16

Построить кривую в форме улитки Паскаля, заданную параметрически:

$$\begin{cases} x = a\cos^2 t + b\cos t; \\ y = a\cos t \sin t + b\sin t, \end{cases}$$

где $a > 0$; $b > 0$; $t \in [0, 2\pi)$.

Вариант 17

Построить кардиоиду, заданную параметрически:

$$\begin{cases} x = a\cos t\,(1 + \cos t); \\ y = a\sin t\,(1 + \cos t), \end{cases}$$

где $a > 0$; $t \in [0, 2\pi)$.

Вариант 18

Построить эпициклоиду, заданную параметрически:

$$\begin{cases} x = (a+b)\cos t - a\cos[(a+b)t/a]; \\ y = (a+b)\sin t - a\sin[(a+b)t/a], \end{cases}$$

где $a > 0$; $b > 0$; $t \in [0, 2\pi)$.

Вариант 19

Построить астроиду, заданную параметрически:

$$\begin{cases} x = a\cos^3 t; \\ y = a\sin^3 t, \end{cases}$$

где $t \in [0, 2\pi)$.

Вариант 20

Построить овалы Кассини, заданные в полярных системах координат:

$$\begin{cases} \rho^2 = c^2\cos 2\varphi + \sqrt{c^4\cos^2 2\varphi + (a^4 - c^4)}; \\ \rho^2 = c^2\cos 2\varphi - \sqrt{c^4\cos^2 2\varphi + (a^4 - c^4)}. \end{cases}$$

Рассмотреть случай, когда $0 < c\sqrt{2} < a$.

Вариант 21

Построить овалы Кассини, заданные в полярных системах координат:

$$\begin{cases} \rho^2 = c^2\cos 2\varphi + \sqrt{c^4\cos^2 2\varphi + (a^4 - c^4)}; \\ \rho^2 = c^2\cos 2\varphi - \sqrt{c^4\cos^2 2\varphi + (a^4 - c^4)}. \end{cases}$$

Рассмотреть случай, когда $0 < c < a < c\sqrt{2}$.

Вариант 22

Построить овалы Кассини, заданные в полярных системах координат:

$$\begin{cases} \rho^2 = c^2 \cos 2\varphi + \sqrt{c^4 \cos^2 2\varphi + \left(a^4 - c^4\right)}; \\ \rho^2 = c^2 \cos 2\varphi - \sqrt{c^4 \cos^2 2\varphi + \left(a^4 - c^4\right)}. \end{cases}$$

Рассмотреть случай, когда $0 < a < c$.

Вариант 23

Построить кривую Лемни-Ската, заданную в полярной системе координат:

$$\rho = a\sqrt{2\cos 2\varphi},$$

где $a > 0$.

Вариант 24

Составить программу, которая перемещает по экрану окружность. Предусмотреть ввод произвольного радиуса окружности.

Вариант 25

Составить программу, которая перемещает по экрану прямоугольник. Предусмотреть ввод произвольных длин сторон прямоугольника.

Вариант 26

Составить программу, которая перемещает по эллипс. Предусмотреть ввод произвольных полуосей эллипса.

Вариант 27

Составить программу, которая перемещает по экрану и вращает дугу окружности. Предусмотреть ввод произвольного радиуса окружности и размеров дуги.

Вариант 28

Изобразить на экране точку, пересекающую с постоянной линейной скоростью экран справа налево.

Вариант 29

Изобразить на экране прямую, вращающуюся в плоскости экрана вокруг одной из своих произвольных точек.

Вариант 30

Изобразить на экране точку, движущуюся по окружности с постоянной угловой скоростью.

6.3. Теоретические сведения

6.3.1. 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 воз-

вращает 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 тести-

рования 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.

6.3.2. 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 верхнего 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 перемещает курсор в позицию, заданную координатами **x** и **y**. Вторая 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.

6.3.3. 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**.

6.3.4. 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 ***polypoints** указывает на массив, каждый элемент 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**.

6.3.5. 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 шрифта 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**.

6.3.6. 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:

$$y\_\text{экрана} = y\_\text{центра} - \text{масштаб} \cdot f(x); \quad (7.1)$$

$$x\_\text{экрана} = x\_\text{центра} + \text{масштаб} \cdot x. \quad (7.2)$$

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 (7.2), вычисляют математическую координату x по формуле

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

— вычисляют $y_$экрана по указанной выше формуле (7.1) и получают экранную точку с координатами $(y\_\text{экрана}, x\_\text{экрана})$;

— полученные экранные точки соединяют отрезками и получают график функции.

6.3.7. 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 по-

мощью 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 прямоугольником, 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 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>**.

6.3.8. 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 7.3.6.

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.

```
/*----------------------подключение библиотек--------------------*/
#include <stdio.h>     // библиотека стандартных средств
                       // ввода-вывода языка C
#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 C
/*----------------------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:\\tools\\bc\\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!=O){
            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
}
```

6.4. Порядок выполнения работы

Вариант задания выдаётся преподавателем. Перед выполнением работы необходимо ознакомиться с разделами 6.1 — 6.3 данных методических указаний. В ходе самостоятельной подготовки изучить особенности графического режима видеоадаптера и основные функции графической библиотеки языка *Borland C*. Индивидуальные задания для выполнения работы см. в разделе 6.2.

6.4.1. Внимательно изучить постановку задачи.

6.4.2. Выполнить анализ области значений логических и экранных координат, выбрать при необходимости масштаб.

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

6.4.4. Разработать программу на языке *C*/*C*++.

6.4.5. Разработать тестовые примеры, следуя указаниям, изложенным в разделе 6.3.

6.4.6. Выполнить тестирование и отладку написанной программы.

6.4.7. Получить результаты работы программы и исследовать её свойства в различных режимах работы.

6.4.8. Подготовить отчет по работе, сформулировать выводы.

6.5. Оформление отчета

6.5.1. Сформулировать цель работы.

6.5.2. Привести постановку задачи и текст индивидуального задания.

6.5.3. Привести краткие теоретические сведения и математическое обоснование задачи.

6.5.4. Изобразить схему алгоритма решения задачи и составить его описание.

6.5.5. Привести текст программы с комментариями.

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

6.5.7. Привести результаты тестирования и отладки программы.

6.5.8. Привести результаты работы программы.

6.5.9. Сделать выводы по работе.

6.6. Контрольные вопросы

6.6.1. Чем отличаются текстовый и графический режим работы дисплея?

6.6.2. Что понимают под разрешающей способностью дисплея?

6.6.3. Что такое видеопамять, видеоадаптер?

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

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

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

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

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

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

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

6.6.11. Как отобразить текст в графическом режиме?

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

6.6.13. Как управлять стилем рисования линий?

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

6.6.15. Приведите формулы преобразования логических координат в экранные.

6.6.16. Приведите формулы линейного преобразования координат.

6.6.17. Как реализовать анимацию?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Методические указания по оформлению текстовых работ для студентов дневной и заочной форм обучения направления 0907 — «Радиотехника» / Сост. В.Г. Слезкин. — Севастополь: Изд-во СевНТУ, 2006. — 15 с.

2. 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.

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

4. Павловская Т.А. *C*/*C*++. Программирование на языке высокого уровня / Т.А. Павловская. — СПб.: Питер, 2007. — 461 с.

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

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

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

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

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

10. Дуглас Т. Программирование на языке СИ для персонального компьютера IBM PC / Т. Дуглас. — М.: Радио и связь, 1991. — 172 с.

11. Керниган Б.В. Язык программирования СИ / Б.В. Керниган, Д.М. Ритчи. — М.: Финансы и статистика, 1992. — 271 с.

12. Болски М.И. Язык программирования СИ. Справочник / М.И. Болски. — М.: Радио и связь, 1988. — 96 с.

13. Белецкий Я. Энциклопедия языка *C*: [пер. с польск.] / Я. Белецкий. — М.: Мир, 1992. — 687 с.

14. Вирт Н. Алгоритмы и структуры данных: [пер. с англ.] / Н. Вирт. — М.: Мир, 1989. —360 с.

15. Шилдт Г. Самоучитель *C*++: [пер. с англ.] / Г. Шилдт. — СПб.: Издательская группа BHV, 1998. — 620 с.

ПРИЛОЖЕНИЕ А
ПРАВИЛА ОФОРМЛЕНИЯ СХЕМ АЛГОРИТМОВ

А.1. Общие положения

Данные правила составлены на основе ГОСТ 19.701-90 [2], который был разработан методом прямого применения международного стандарта *ISO* 5807-85 «Обработка информации. Символы и условные обозначения блок-схем данных, программ и систем, схем программных сетей и системных ресурсов», и принят взамен ГОСТ 19.002-80 и ГОСТ 19.003-80.

Указанный стандарт распространяется на условные обозначения (символы) в схемах алгоритмов, программ, данных и систем, а также устанавливает правила выполнения схем. Стандарт не распространяется на форму записей и обозначений, помещаемых внутри символов или рядом с ними и служащих для уточнения выполняемых ими функций.

В данном приложении рассматриваются только символы и правила, касающиеся изображения схем алгоритмов программ.

Схемой называют графическое представление анализа или метода решения задачи, в котором используются символы для отображения операций, данных и т.д. Схема программы отображает последовательность операций в программе.

При изображении схем алгоритмов различают основные и специфические символы. К основным относятся символы, используемые в тех случаях, когда точный тип процесса или носителя данных неизвестен или отсутствует необходимость в описании фактического носителя данных. Специфические символы используются в тех случаях, когда известен точный тип процесса или носителя данных. Схема программы состоит из:

— символов процесса, указывающих операции обработки данных;

— линейных символов, указывающих поток управления;

— специальных символов, используемых для облегчения написания и чтения схемы.

А.2. Описание символов

А.2.1. Символы данных

При составлении схем программ используется только один основной символ данных (рис.А.1). Символ «данные» отображает данные, расположенные на неопределенном носителе.

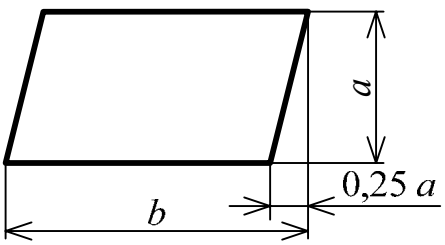


Рис. А.1 — Данные

Используется символ для блоков преобразования данных в форму, пригодную для обработки (ввод) или отображения результата обработки (вывод).

А.2.2. Символы процесса

В схемах программ применяют как основные (рис.А.2,а), так и специфические символы процесса (рис.А.2,б,в,г и рис.А.3).

Символ «процесс», изображенный на рис.А.2,а, отображает функцию обработки данных любого вида. Применяется символ для обозначения выполнения определенной операции или группы операций, приводящих к изменению значения, формы или размещения данных.

Символ «предопределенный процесс» (рис.А.2,б) отображает предопределенный процесс (отдельно описанный алгоритм или программу), состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте (в подпрограмме, модуле).

Символ «подготовка», показанный на рис.А.2,в, отображает модификацию команды или группы команд с целью воздействия на некоторую последующую функцию. Применяется для выполнения операций, меняющих команды или группы команд, изменяющих программу, для установки переключателя, модификации индексного регистра или инициализации программы.

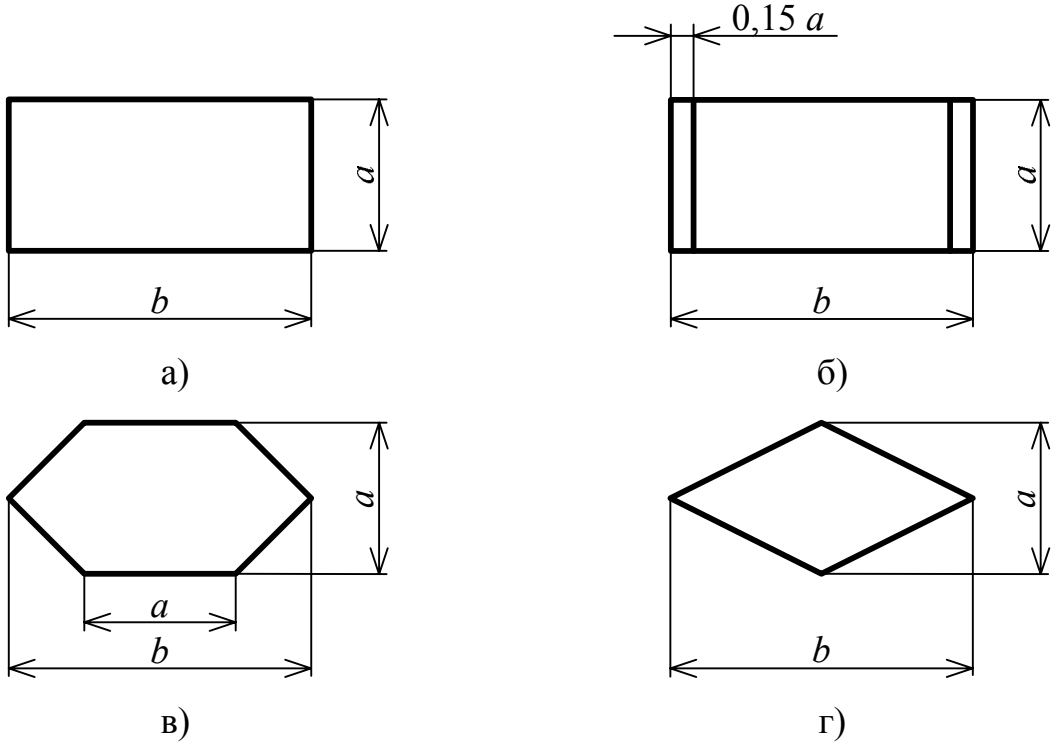


Рис. А.2 — Процесс (а), предопределенный процесс (б), подготовка (в) и решение (г)

Символ «решение» (см. рис. А.2,г) отображает решение или функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован, после вычисления условий, определенных внутри этого символа. Результаты вычисления, активизирующие выходы, могут быть записаны по соседству с линиями, отображаю-

щими пути выходов. Используется символ для отображения выбора направления выполнения алгоритма или программы в зависимости от некоторых переменных условий, т.е для описания условия.

На рис. А.3 изображен символ «граница цикла» и пример его использования. Символ состоит из двух частей и отображает начало и конец цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения, и т.д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие окончания цикла.

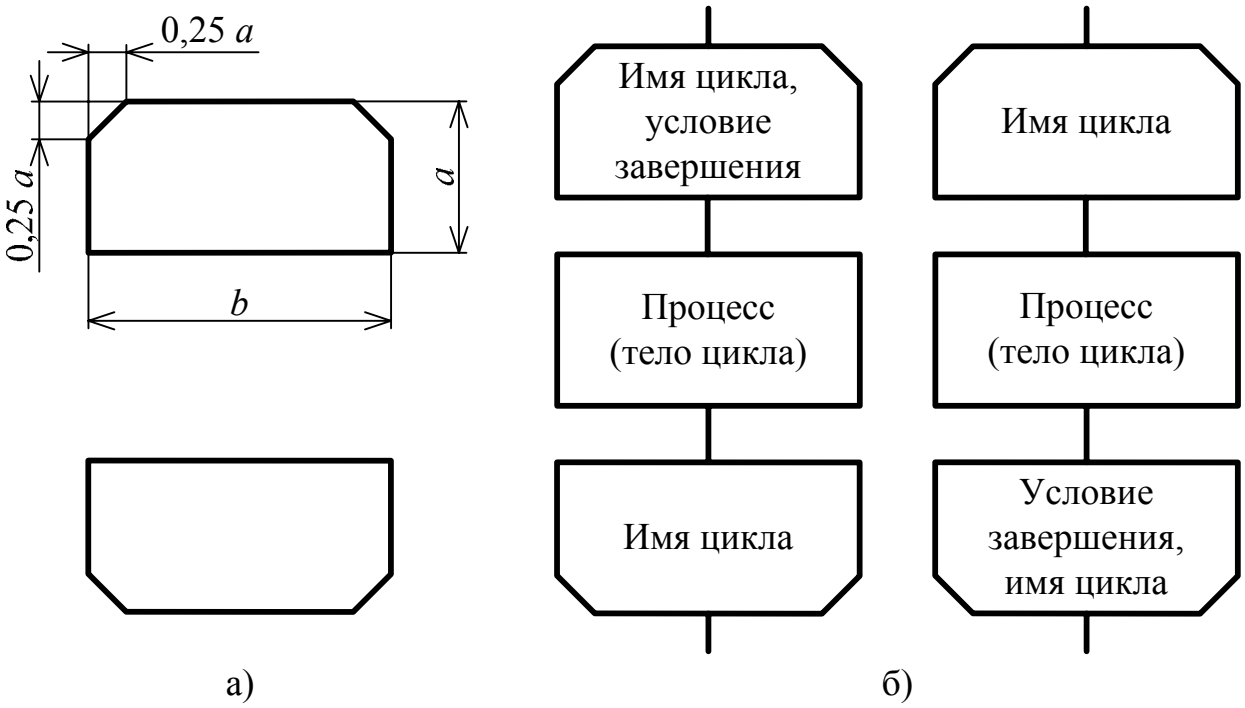


Рис. А.3 — Граница цикла (а) и примеры использования символа (б)

А.2.3. Символы линий

При изображении схем алгоритмов программ применяют символы «линия», который является основным, и «пунктирная линия», являющийся специфическим (рис.А.4).

Символ «линия» отображает поток данных или управления. Применяется для указания последовательности связей между символами. При необходимости или для повышения удобочитаемости к линии могут быть добавлены стрелки-указатели.

Символ «пунктирная линия» отображает альтернативную связь между двумя или более символов (например, транспортирование носителей данных). Кроме того, символ применяют для обведения аннотированного участка схемы (см. рис.А.6).

Рис. А.4 — Линия (а) и пунктирная линия (б)

А.2.4. Специальные символы

К специальным символам относятся «соединитель», «терминатор», «комментарий» и «пропуск».

Изображенный на рис.А.5,а, символ «соединитель» отображает выход в часть схемы или вход из другой части этой схемы и используется для обрыва линии, связывающей символы, и продолжения её в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение.

Символ «терминатор» (рис.А.5,б) отображает выход во внешнюю среду и вход из внешней среды. Используется для обозначения начала, конца, прерывания процесса обработки данных или программы, а также внешнего использования, источника или пункта назначения данных.

«Комментарий» (рис.А.5,в) используют для добавления описательных комментариев или поясняющих записей в целях объяснения или примечания. Пунктирные линии в символе комментария связаны с соответствующим символом или могут обводить группу символов (рис.А.6). Текст комментариев или примечаний должен быть помещен около ограничивающей фигуры (скобки).

Символ «пропуск» (три точки) используют в схемах для отображения пропуска символа или группы символов, в которых не определены ни тип, ни число символов. Используют символ только в символах линии или между ними (рис.А.5,г). Применяется главным образом в схемах, изображающих общие решения с неизвестным числом повторений. На рис.А.7 изображен пример использования символа «пропуск».

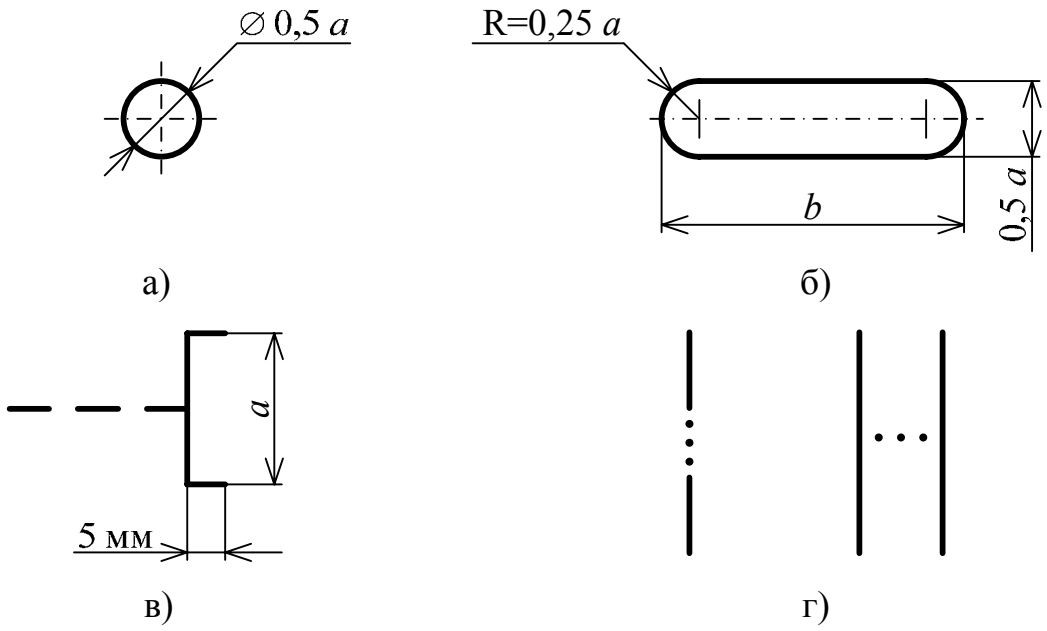


Рис. А.5 — Соединитель (а), терминатор (б),
комментарий (в) и пропуск (г)

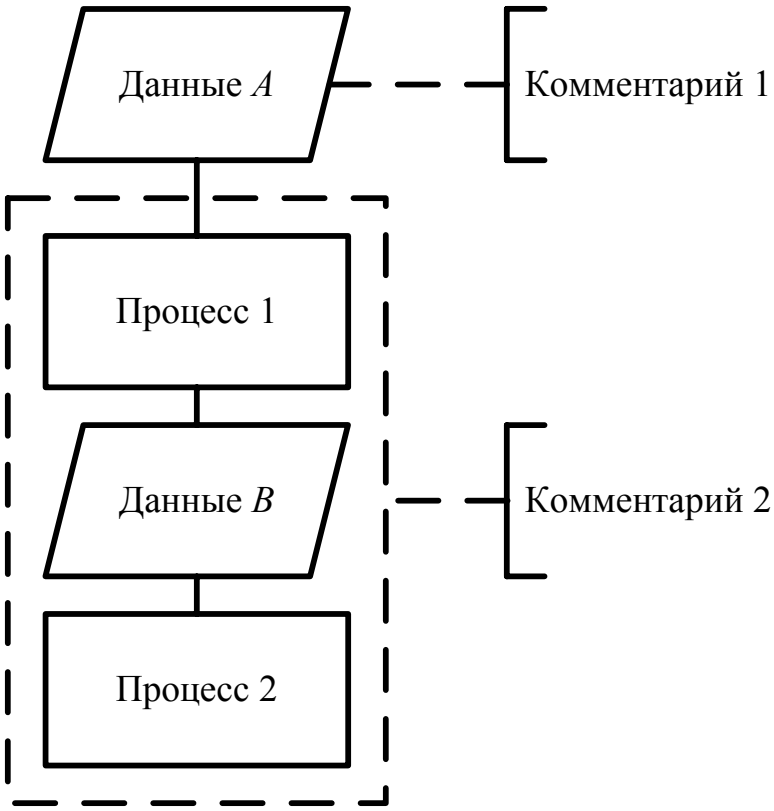


Рис. А.6 — Пример использования комментария

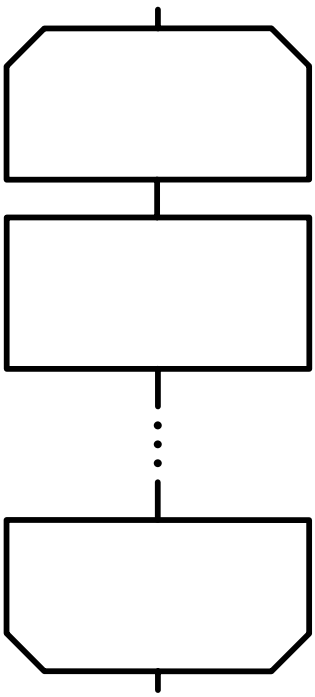


Рис. А.7 — Пример использования пропуска (троеточия)

А.3. Соотношение геометрических размеров символов

Размер a рекомендуется выбирать из ряда 10, 15, 20 мм. Допускается увеличивать размер a на величину, кратную 5 мм. Размер b равен $1{,}5a$.

При ручном выполнении схем алгоритмов и программ допускается устанавливать размер b равным $2a$.

При автоматизированном выполнении условных графических обозначений размеры символов округляют до значений, определяемых техническими возможностями используемых устройств.

А.4. Правила применения символов и выполнения схем

А.4.1. Правила применения символов

1) Символ предназначен для графической идентификации функции, которую он отображает, независимо от текста внутри этого символа.

2) Символы в схеме должны располагаться равномерно. Следует придерживаться разумной длины соединений и минимального числа длинных линий.

3) Большинство символов задумано так, чтобы дать возможность включения текста внутри символа. Формы символов, установленные стандартом, должны служить руководством для фактически используемых символов. Не должны изменяться геометрические параметры, влияющие на форму символов. Символы по возможности должны быть одного размера.

4) Символы могут быть вычерчены в любой ориентации, но, по возможности, предпочтительной является горизонтальная ориентация. Зеркальное изображение формы символа обозначает одну и ту же функцию, но не является предпочтительным.

5) Минимальное количество текста, необходимого для понимания функции символа, следует помещать внутри символа. Текст для чтения должен записываться слева направо и сверху вниз независимо от направления потока (рис.А.8).

Если объем текста превышает размеры символа, то следует использовать символ комментария. Если использование символов комментария может запутать или разрушить ход схемы, то текст комментария следует поместить на отдельном листе, указав на него перекрестную ссылку или описание (см. далее правило 7).

6) В схемах может использоваться идентификатор символов. Это связанный с данным символом идентификатор, который определяет символ для использования в справочных целях в других элементах программной документации (например, в листинге программы). Идентификатор символа должен располагаться слева над символом (рис.А.9).

7) В схемах может использоваться описание символов — любая информация, например, для отображения специального применения символа с перекрестной ссылкой, или для улучшения понимания функции как части схемы. Описание символа должно быть расположено справа над символом (рис.А.10).

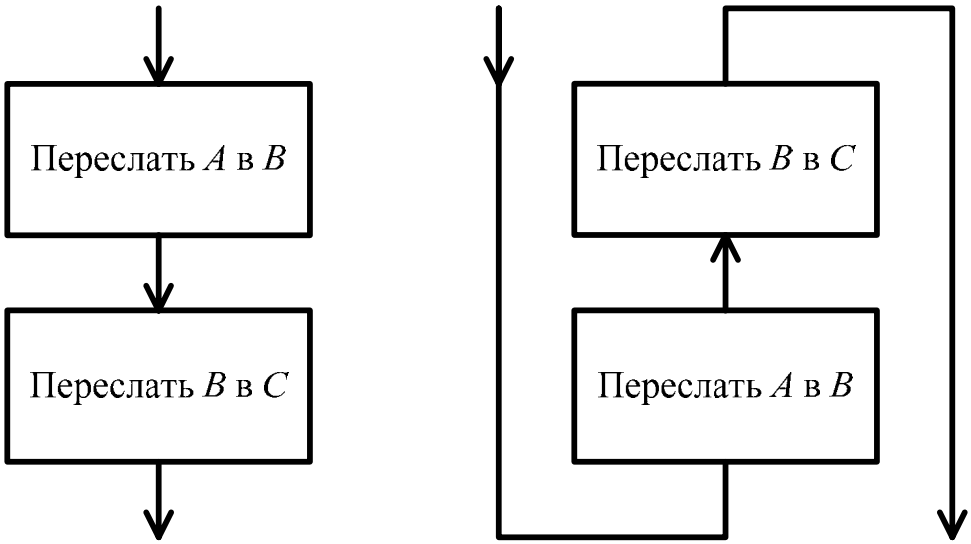


Рис. А.8 — Запись текста внутри символов

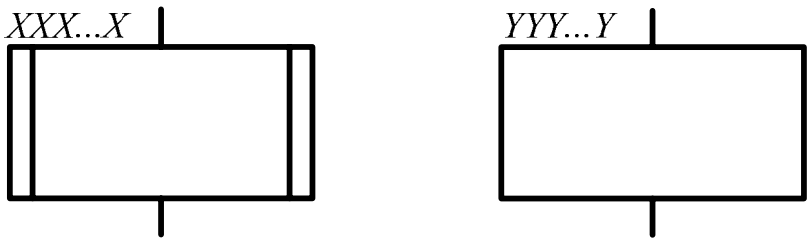


Рис. А.9 — Расположение идентификаторов

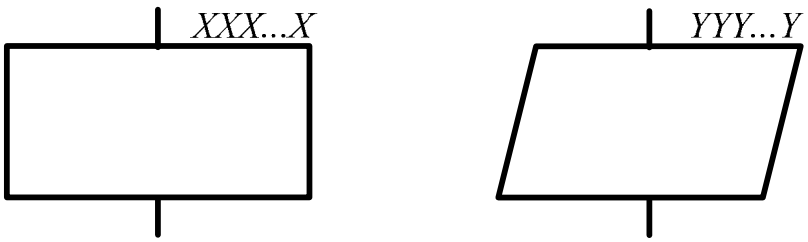
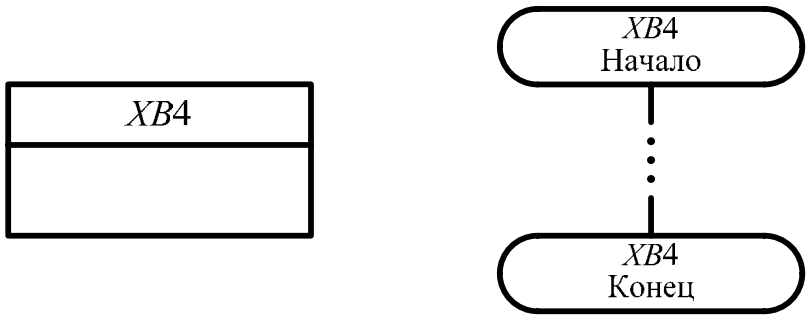


Рис. А.10 — Расположение описаний символов



а) б)

Рис. А.11 — Символ с полосой (а) и подробное преставление (б)

8) В схемах может использоваться подробное представление, которое обозначается с помощью символа с полосой для процесса или данных. Символ с полосой указывает, что в этом же комплекте документации в другом месте имеется более подробное представление.

Символ с полосой представляет собой любой символ, внутри которого в верхней части проведена горизонтальная линия. Между этой линией и верхней границей символа помещается идентификатор, указывающий на подробное представление данного символа (см. рис.А.11,а).

В качестве первого и последнего символов подробного представления должен быть использован символ «терминатор». Начальный символ подробного представления должен содержать ссылку на идентификатор, который имеется в символе с полосой (см. рис.А.11,б).

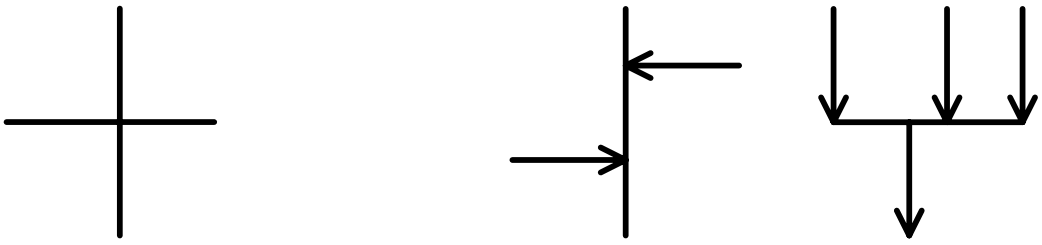
А.4.2. Правила выполнения соединений

1) Потоки данных или потоки управления в схемах показываются линиями. Направление потока слева направо и сверху вниз считается стандартным.

В случаях, когда необходимо внести большую ясность в схему (например, при соединениях), на линиях используются стрелки. Если поток имеет направление отличающееся от стандартного, стрелки должны указывать это направление.

2) В схемах следует избегать пересечения линий. Пересекающиеся на схеме линии не имеют логической связи между собой, поэтому изменения направления в точках пересечения не допускаются (рис.А.12,а).

3) Две или более входящие линии могут объединяться в одну исходящую линию. Если две или более линии объединяются в одну, то место объединения должно быть смещено (рис.А.12,б).



а) б)

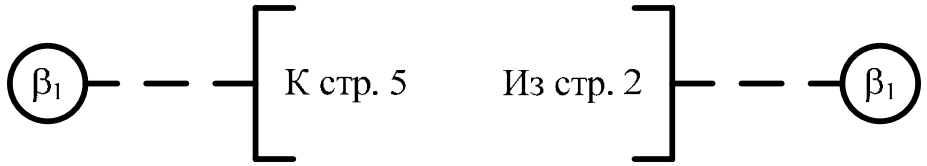
Рис. А.12 — Примеры пересечения линий (а) и объединения линий (б)

4) Линии в схемах должны подходить к символу либо слева, либо сверху а исходить либо справа, либо снизу. Линии должны быть направлены к центру символа.

5) При необходимости линии в схемах следует разрывать для уменьшения количества пересечений или слишком длинных линий, а также если схема расположена на нескольких страницах. Соединитель в начале разрыва называют внешним, а в конце разрыва — внутренним.

Ссылки к страницам могут быть приведены совместно с символом ком-

ментария для соединителей (рис.А.13).



а) б)

Рис. А.13 — Внешний соединитель (а) и внутренний соединитель (б)

А.5. Специальные условные обозначения

При составлении схем программ используется только одно специальное условное обозначение — несколько выходов.

Несколько выходов из символа следует показывать:

— несколькими линиями от данного символа к другим символам;

— одной линией от данного символа, которая затем разветвляется в соответствующее число линий.

Каждый выход из символа при таком представлении должен сопровождаться соответствующими значениями условий, чтобы показать логический путь, который он указывает, с тем, чтобы эти условия и соответствующие ссылки были однозначно идентифицированы.

Примеры использования нескольких выходов изображены на рис.А.14.

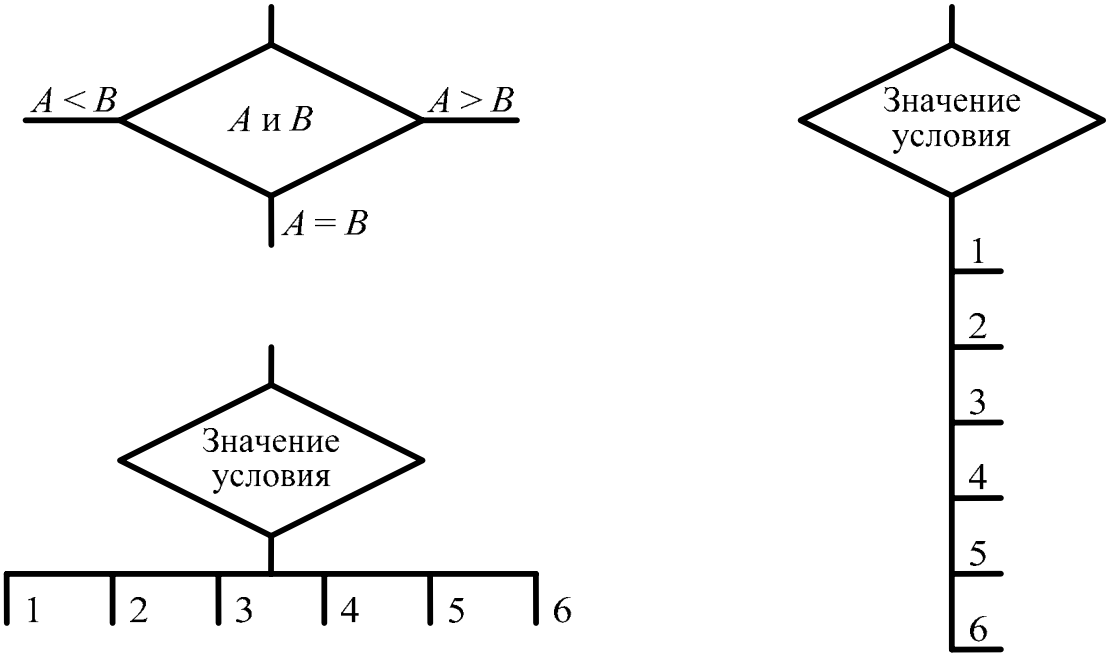


Рис. А.14 — Варианты применения нескольких выходов из символа

А.6. Примеры изображения схем алгоритмов

На рис.А.15 и А.16 изображены схемы алгоритмов программ, предлагаемые стандартом по правилам выполнения схем алгоритмов, программ, данных и систем.

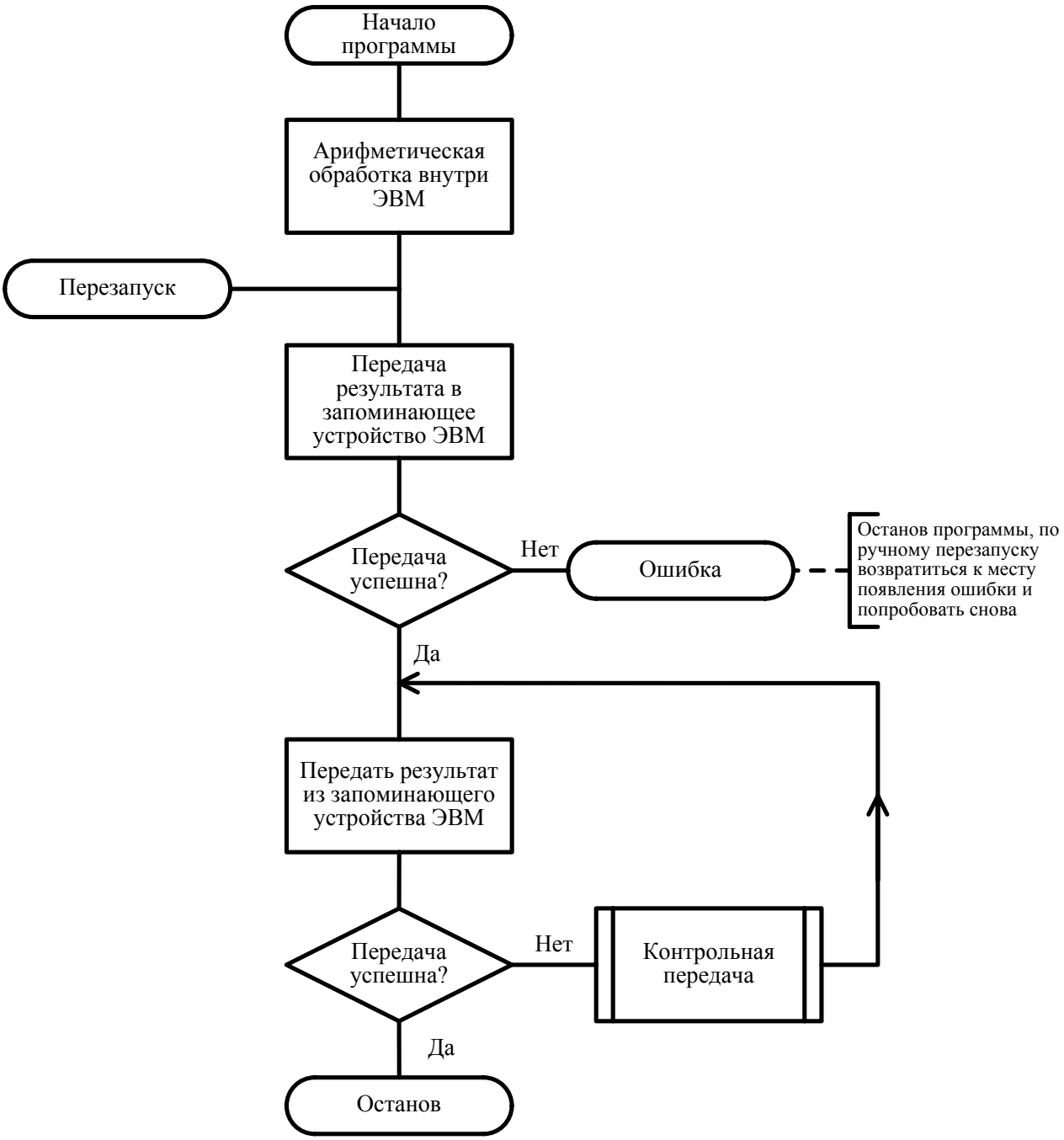


Рис. А.15 — Первый пример изображения схемы программы [2]

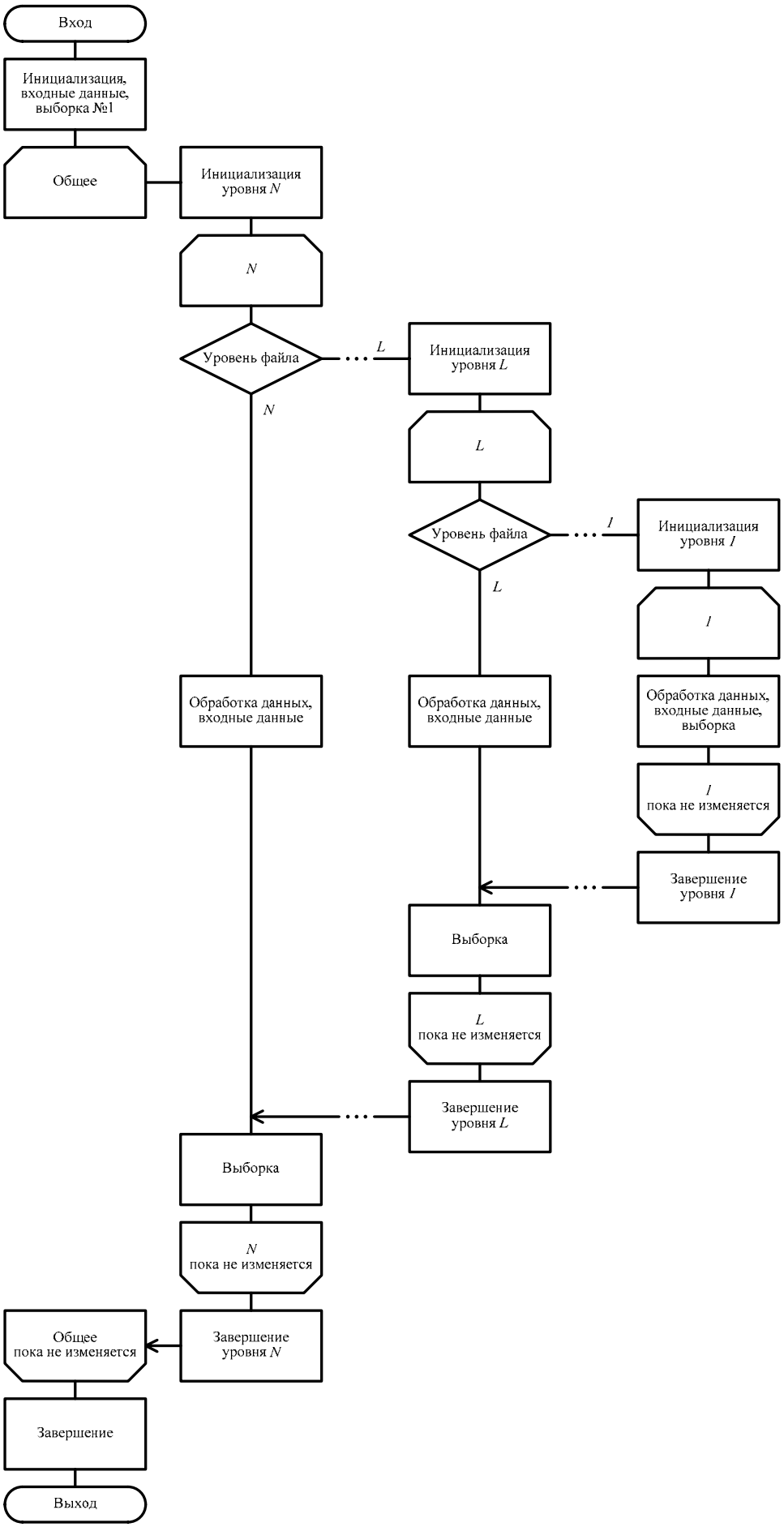


Рис. А.16 — Второй пример изображения схемы программы [2]

ПРИЛОЖЕНИЕ Б
АЛГОРИТМЫ СОРТИРОВКИ МАССИВОВ

Б.1. Сортировка Шелла

Сортировка Шелла по-другому называется сортировкой с уменьшающимся шагом. Основная идея этого алгоритма состоит в том, что на ранних стадиях сравниваются далеко отстоящие друг от друга, а не соседние элементы, как в обычных перестановочных сортировках. Это приводит к быстрому устранению массовой неупорядоченности, благодаря чему на более поздней стадии остается меньше работы. Интервал между сравниваемыми элементами постепенно уменьшается до единицы, и в этот момент сортировка сводится к обычным перестановкам соседних элементов.

Алгоритм сортировки Шелла состоит в следующем:

1) Элементы исходного массива разбиваются на непересекающиеся подмассивы таким образом, чтобы в каждый подмассив входило не более двух элементов. Элементы каждого подмассива располагаются на расстоянии d друг от друга. Расстояние d (шаг подмассива) выбирается так

$$d = 2^m > N/2,$$

где m — натуральное число;

N — количество элементов в исходном массиве.

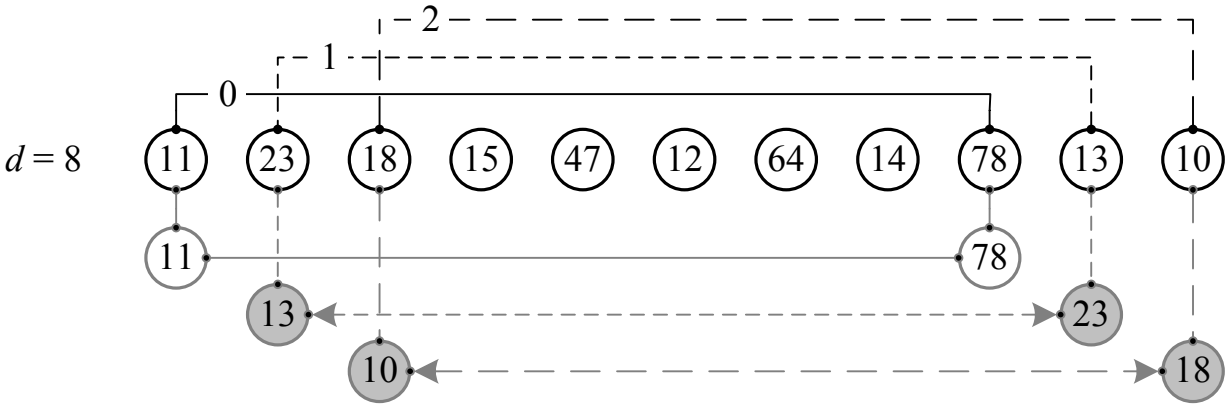
2) Производится обычная сортировка каждого подмассива (например, методом вставки или методом прямого выбора). В результате получается преобразованный массив.

3) Шаг подмассива уменьшается вдвое ($d = d/2$). Элементы преобразованного массива разбиваются на непересекающиеся подмассивы с шагом d.

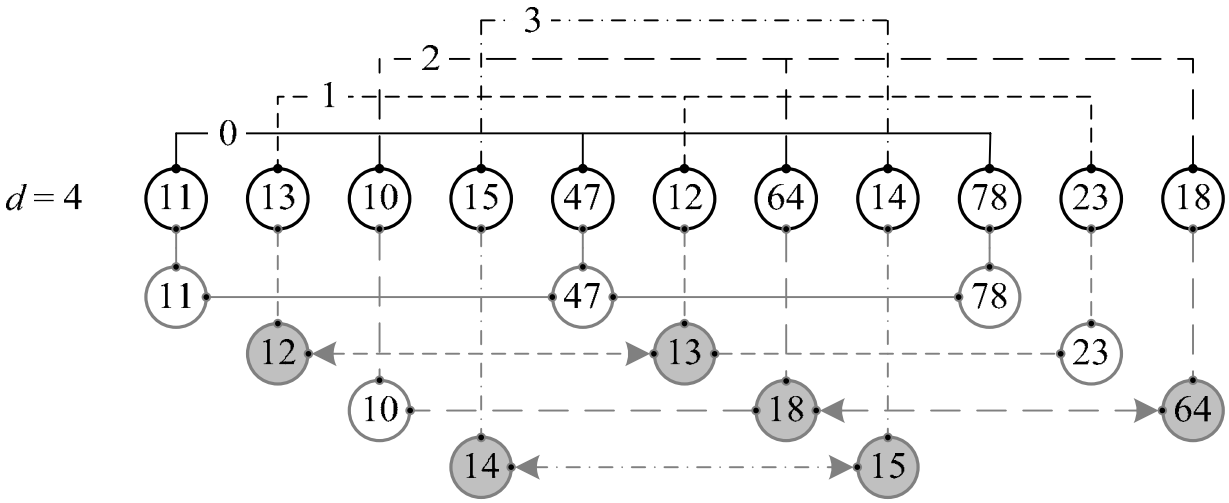
4) Выполняется пункт 2. Если шаг подмассива d равен единице, то сортировка окончена, иначе переход к пункту 3.

Иллюстрирует описанный алгоритм рис.Б.1. В примере, соответствующем рис.Б.1, сортировке по возрастанию подвергается массив, состоящий из 11 элементов. Очевидно, начальный шаг подмассива равен $d = 2^3 = 8 > 11/2$. Элементы исходного массива разбиваются на 3 подмассива по 2 элемента и 5 подмассивов по одному элементу (рис.Б.1,а). Далее выполняется сортировка элементов в каждом подмассиве методом прямого обмена (пузырька).

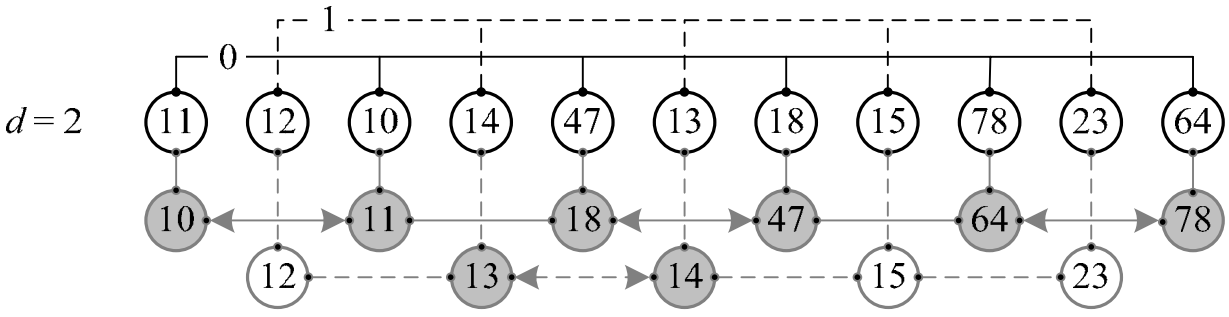
Затем шаг подмассива уменьшается в 2 раза ($d = 4$, рис.Б.1,б). Сортируемый массив разбивается на 4 подмассива, после чего каждый подмассив сортируется методом пузырька. Процесс уменьшения шага подмассива с последующей сортировкой подмассивов повторяется до тех пор, пока d не станет равным единице (рис.Б.1,г). Результатом является отсортированный массив.



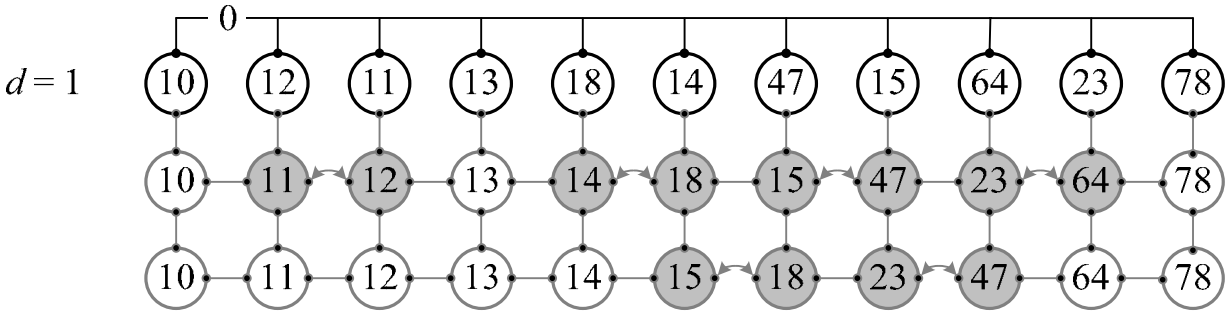
а)



б)



в)



г)

Рис.Б.1 — Пример сортировки методом Шелла

С учетом вышесказанного программу, реализующую алгоритм сортировки Шелла **n**-элементного массива **a**, можно сформулировать следующим образом:

```
// определение начального шага подмассива
for(d=1;d<(double)n/2;d*=2);
do { // сортировка непересекающихся подмассивов
    for(k=0;k<d;k++)
        // выполнить обычную сортировку k-го подмассива
}while((d/=2)>=1);
```

При программировании сортировки **k**-го подмассива в методе Шелла следует иметь ввиду, что все элементы этого подмассива отстоят друг от друга на расстоянии **d**, первый элемент **k**-го подмассива есть **a[k]**, а последний есть **a[up]**. Можно показать, что верхняя граница **k**-го подмассива равна: **up=k+((n-k-1)/d)*d**.

Б.2. Метод прямого обмена

Данный метод также носит название «пузырькового» метода и является самым простым. Заключается метод в систематическом обмене соседних элементов, не отвечающих требуемому порядку, пока такие пары элементов существуют.

Ниже представлен текст программы сортировки массива методом Шелла с использованием сортировки подмассивов методом прямого обмена.

```
main()
// сортировка массива по возрастанию методом Шелла с
// использованием сортировки подмассивов методом прямого обмена

{
    const unsigned int n=15;   // размер сортируемого массива
    int a[n];          // сортируемый массив
    int d;             // шаг подмассива (расстояние между эл-тами)
    int k;             // индекс начала k-го подмассива
    int up;            // индекс конца k-го подмассива
    int i;             // счетчик цикла (текущий индекс подмассива)
    int t;             // буфер в сортировке прямым обменом
    int was_swap;      // флаговая переменныя, принимающая
        // значение 1, если в ходе просмотра подмассива был
        // обмен эл-тов, и значение 0 в противном случае

    // определение начального шага подмассива
    for(d=1;d<(float)n/2;d*=2);

    do{ // сортировка непересекающихся подмассивов
    for(k=0;k<d;k++)
        // сортировка k-го подмассива методом прямого обмена
        for(up=k+((n-k-1)/d)*d,was_swap=1;up>k;up-=d)
            if(was_swap){ // был обмен эл-тов в ходе
                          // предыдущего просмотра подмассива
                was_swap=0;       // в начале просмотра обмена нет
```

```
                for(i=k;(i+d)<=up;i+=d)      // «всплытие пузырька»
                if(a[i]>a[i+d])
                    // обмен соседних эл-тов
                    t=a[i], a[i]=a[i+d], a[i+d]=t, was_swap=1;
            }
            else break;  // в ходе предыдущего просмотра
                         // подмассива обмена эл-тов не было
    } while((d/=2)>=1);

    return 0;
}
```

Использование флаговой переменной **was_swap** позволяет избежать лишних сравнений при сортировке подмассивов методом прямого обмена.

Б.3. Метод прямого выбора

Упорядоченный список получается из исходного многократным применением выборки из исходного списка минимального элемента, удалением этого элемента из исходного списка, и добавлением его в конец упорядоченного списка, который первоначально должен быть пустым.

Текст программы сортировки массива методом Шелла с использованием сортировки подмассивов методом прямого выбора приведен ниже.

```
main()

// сортировка массива по возрастанию методом Шелла с
// использованием сортировки подмассивов методом прямого выбора
{
    const unsigned int n=15;  // размер сортируемого массива
    int a[n];         // сортируемый массив
    int d;            // шаг подмассива (расстояние между эл-тами)
    int k;            // индекс начала k-го подмассива
    int up;           // индекс конца k-го подмассива
    int i;            // счетчик цикла (текущий индекс подмассива)
    int i_max;        // индекс максимального эл-та в подмассиве
    int t;            // буфер обмена для перестановки элементов
                      // в сортировке прямым выбором

    // определение начального шага подмассива
    for(d=1;d<(float)n/2;d*=2);

    do {   // сортировка непересекающихся подмассивов
        for(k=0;k<d;k++)
        // сортировка k-го подмассива методом прямого выбора
        for(up=k+((n-k-1)/d)*d,was_swap=1;up>k;up-=d){
            // поиск максимального эл-та
            for(i_max=k,i=k+d;i<=up;i+=d)
                if(a[i]>a[i_max]) i_max=i;
            // перестановка эл-тов
            t=a[up], a[up]=a[i_max], a[i_max]=t;
        }
```

```
    } while((d/=2)>=1);

    return 0;

}
```

Б.4. Метод вставки

Упорядоченный массив получается из исходного следующим образом: сначала он состоит из единственного элемента (первого элемента исходного списка), далее по порядку все элементы из исходного массива вставляются в упорядоченный список таким образом, чтобы он оставался упорядоченным.

Ниже представлен текст программы сортировки массива методом Шелла с использованием сортировки подмассивов методом вставки.

```
main()
// сортировка массива по возрастанию методом Шелла с
// использованием сортировки подмассивов методом вставки

{
    const unsigned int n=15;  // размер сортируемого массива
    int a[n];         // сортируемый массив
    int d;            // шаг подмассива (расстояние между эл-тами)
    int k;            // индекс начала k-го подмассива
    int up;           // индекс конца k-го подмассива
    int low;      // индекс начала отсортированной части подмассива
    int i;            // счетчик цикла (текущий индекс подмассива)
    int t;        // буферная переменная, используемая в сортировке
                  // вставкой для хранения эл-та, вставляемого в
                  // отсортированную часть

    // определение начального шага подмассива
    for(d=1;d<(float)n/2;d*=2);

    do {  // сортировка непересекающихся подмассивов
        for(k=0;k<d;k++)
            // сортировка k-го подмассива методом вставки
            for(up=k+((n-k-1)/d)*d,low=up;low>k;low-=d){
                // выделение в отсортированной части подмассива
                // места для вставки очередного эл-та
                // из неотсортированной части подмассива
                for(i=low,t=a[low-d];i<=up;i+=d)
                    if(t>a[i]) a[i-d]=a[i]; else break;
                a[i-d]=t; // вставка эл-та в отсортированную часть
            }
    } while((d/=2)>=1);

    return 0;

}
```

Б.5. Быстрая сортировка

Быстрая сортировка (1962 г.) по-другому называется сортировкой с разделением. Это самый лучший метод сортировки массивов.

Быстрая сортировка базируется на следующем алгоритме. Из массива выбирается средний элемент (назовем его **split**). Далее массив просматривается слева направо, пока не найден элемент, больший **split**. Затем массив просматривается справа налево, пока не найден элемент, меньший **split**. После этого эти два элемента меняются местами. Такой процесс «просмотра с обменом» продолжается до тех пор, пока «просмотр слева направо» и «просмотр справа налево» не встретятся. В результате массив будет разделен на две части: левую — с элементами, не большими, чем **split**, и правую — с элементами, не меньшими, чем **split**.

Ниже представлен текст программы, соответствующей описанному алгоритму.

```
main()
// разделение массива относительно среднего элемента
{
    const unsigned int n=15;  // размер массива
    int a[n];       // разделяемый массив
    int split,      // разделяющий элемент
        t;          // буфер обмена
    int l,          // индекс при просмотре массива слева направо
        r;          // индекс при просмотре массива справа налево

    // просмотр с обменом
    for(l=0,r=n-1,split=a[(l+r)/2];l<=r;){
        while(a[l]<split) ++l;   // просмотр слева направо
        while(a[r]>split) --r;   // просмотр справа налево
        if(l<=r) t=a[l] ,a[l++]=a[r] ,a[r--]=t;   // обмен
    }

    return 0;

}
```

Для сортировки массива с учетом описанного выше алгоритма разделения необходимо выполнить следующее: разделив массив, нужно сделать то же самое с обеими полученными частями, затем с частями этих частей и т.д., пока каждая часть не будет содержать только один элемент. Очевидно, разделение частей массива следует осуществлять в цикле. После каждого разделения нужно произвести два очередных разделения и лишь одно из них можно выполнить непосредственно в ходе следующей итерации. Запрос на другое разделение заносится в стек. Очевидно, что извлечение запросов на разделение из стека выполняется в обратной последовательности.

Для того чтобы необходимый размер стека запросов на разделение был как можно меньшим, в нем следует хранить запрос на разделение более длинной

части, а в цикле сразу осуществлять дальнейшее разделение коротких частей. Нетрудно показать, что в этом случае необходимый размер стека равен $2\log_2 n$, где n — размер сортируемого массива.

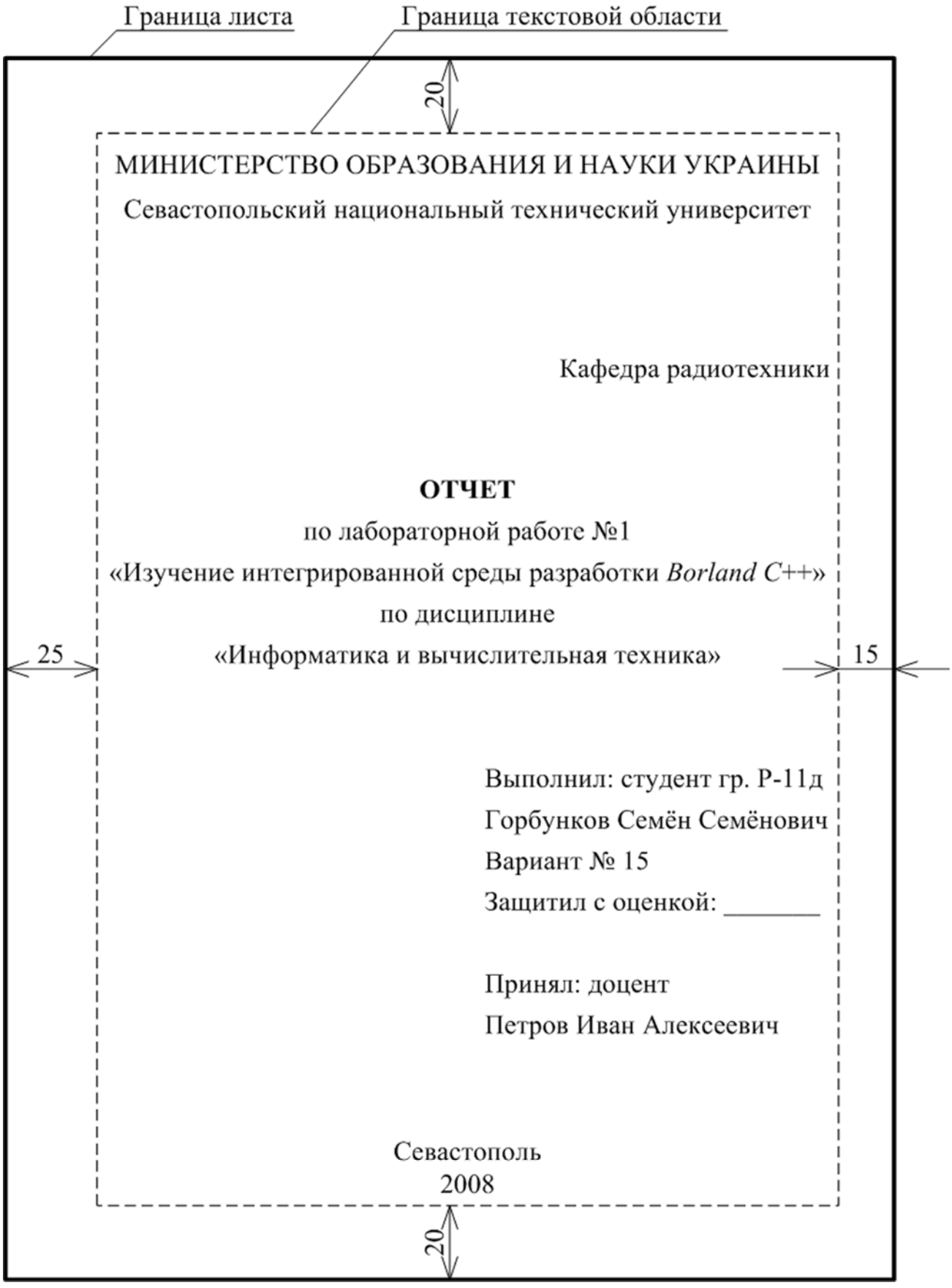
Ниже представлен текст программы быстрой сортировки.

```
#include <math.h>

main()
// быстрая сортировка массива по возрастанию

{
    const unsigned int n=15;   // размер сортируемого массива
    int a[n];          // сортируемый массив
    int split,         // разделяющий элемент
        t;             // буфер обмена
    int i,             // индекс при просмотре разделяемой части
                       // слева направо в процессе разделения
        j,             // индекс при просмотре разделяемой части
                       // справа налево в процессе разделения
        l,             // индекс левой границы разделяемой части
        r;             // индекс правой границы разделяемой части
    int *stack;        // стек для хранения запросов на разделение
                       // хранятся запросы на разделение более
                       // длинных частей, более короткие части
                       // разделяются в цикле
    int stack_ptr=-1; // указатель (индекс) стека

    // выделение памяти под стек
    stack=new int[2*ceil(log(n)/log(2))];
    // инициализация стека
    stack[++stack_ptr]=0;      // индекс левой границы
    stack[++stack_ptr]=n-1;    // индекс правой границы

    while(stack_ptr>=0) {      // пока есть запросы на разделение
                               // (стек не пуст)
        // извлечение из стека очередного запроса на разделение
        r=stack[stack_ptr--]; // правая граница части
        l=stack[stack_ptr--]; // левая граница части
        while(l<r) {          // пока размер разделяемой части >1
            // разделение части относительно среднего эл-та
            for(i=l,j=r,split=a[(i+j)/2];i<=j;){
                while(a[i]<split) ++i; // просмотр слева направо
                while(a[j]>split) --j; // просмотр справа налево
                if(i<=j) t=a[i],a[i++]=a[j],a[j--]=t;   // обмен
            }
            // результат разделения:
            // a[l..j] - левая часть, a[i..r] - правая часть
            if(j-l<r-i) {  // если правая часть длиннее (>) левой
                if(i<r)    // если длина правой части >1
                    // запись в стек запроса на разделение
                    // правой (более длинной) части
                    stack[++stack_ptr]=i,stack[++stack_ptr]=r;
```

```
                r=j;      // a[l..r] - левая (более короткая) часть
                          // разделяется на следующей итерации
            } else {      // если левая часть длиннее (>=) правой
                if (l<j) // если длина левой части >1
                    // запись в стек запроса на разделение
                    // левой (более длинной) части
                    stack[++stack_ptr]=l, stack[++stack_ptr]=j;
                l=i;      // a[l..r] - правая (более короткая) часть
                          // разделяется на следующей итерации
            }
        }
    }

    delete []stack;  // освобождение памяти из-под стека

    return 0;

}
```

ПРИЛОЖЕНИЕ В

ПРИМЕР ОФОРМЛЕНИЯ ТИТУЛЬНОГО ЛИСТА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ



Заказ № _____ от «____»_____ 2008 г. Тираж _____ экз.

Изд-во СевНТУ