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

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

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

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

Часть 1

Севастополь
2013

УДК 004.43+621.37 (076.5)

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

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

Рассмотрены и утверждены на заседании кафедры радиотехники и телекоммуникаций СевНТУ (протокол № 3 от 16 ноября 2012 г.).

Допущено учебно-методическим центром СевНТУ в качестве методических указаний.

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

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

СОДЕРЖАНИЕ

Введение 4
1. Лабораторная работа №1
Изучение интегрированной среды разработки *Borland C++* 6
1.1. Цель работы 6
1.2. Теоретические сведения 6
1.3. Порядок выполнения работы 9
1.4. Оформление отчета 13
1.5. Контрольные вопросы 14
2. Лабораторная работа №2
Программная реализация алгоритмов линейной
и разветвляющейся структуры 15
2.1. Цель работы 15
2.2. Варианты заданий 15
2.3. Теоретические сведения 18
2.4. Порядок выполнения работы 33
2.5. Оформление отчета 34
2.6. Контрольные вопросы 35
3. Лабораторная работа №3
Программная реализация алгоритмов циклической структуры 36
3.1. Цель работы 36
3.2. Варианты заданий 36
3.3. Теоретические сведения 37
3.4. Порядок выполнения работы 44
3.5. Оформление отчета 44
3.6. Контрольные вопросы 45
Библиографический список 46
Приложение А
Правила оформления схем алгоритмов 47
А.1. Общие положения 47
А.2. Описание символов 47
А.3. Соотношение геометрических размеров символов 52
А.4. Правила применения символов и выполнения схем 52
А.5. Специальные условные обозначения 55
А.6. Примеры изображения схем алгоритмов 55
Приложение Б
Пример оформления титульного листа отчета
по лабораторной работе 58

ВВЕДЕНИЕ

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

При изучении первой части дисциплины студенты выполняют и защищают цикл из шести лабораторных работ, которые направлены на получение практических навыков программирования на языке *C*/*C*++ и изучение таких базовых структур как алгоритмы линейной, разветвляющейся и циклической структур, алгоритмы обработки одномерных и двумерных массивов. В данных методических указаниях представлены лабораторные работы №1 — №3 первой части дисциплины. Лабораторные работы №4 — №6 представлены в методических указаниях [1]. Итоговой формой контроля в первом семестре является зачет.

Вторая часть дисциплины предусматривает выполнение и защиту лабораторных работ, направленных на углубление и закрепление знаний, полученных при изучении первой части. При выполнении лабораторного практикума второй части дисциплины рассматриваются вопросы программной реализации алгоритмов работы со структурами данных, строками и файлами, а также использование графического режима. Итоговой формой контроля во втором семестре является экзамен.

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

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

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

На выполнение лабораторных работ №1 — №4 и №6 отводится по четыре часа, а на выполнение работы №5 — шесть часов. Выполняются лабораторные работы в два этапа. На первом этапе студенты самостоятельно должны:

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

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

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

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

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

— представить результаты выполнения первого этапа преподавателю;

— отладить и выполнить написанную программу;

— распечатать полученные результаты;

— оформить чистовик отчета по лабораторной работе;

— защитить отчет по лабораторной работе.

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

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

На титульном листе обязательно требуется указать фамилию, имя и отчество полностью, группу, номер варианта, название и номер работы. Образец оформления титульного листа приведен в приложении Б.

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

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

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

При выполнении лабораторных работ и подготовке к защите рекомендуется использовать учебную и справочную литературу [4 — 13].

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 языке *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 языке *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 на экране отображается рабочее окно среды разработки *Borland C++*, содержащее четыре основных части:

— строка меню,

— окно редактирования,

— окно сообщений,

— строка состояния.

Строка меню предоставляет доступ к командам ИСР. Для активизации строки меню следует нажать клавишу *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*) ИСР;
Copy	— копировать блок текста и поместить его в буфер ИСР;
Paste	— вставить блок текста из буфера ИСР;
Clear	— удалить блок текста, не занося его в буфер ИСР;
Show clipboard	— показать содержимое буфера ИСР.

ИСР *Borland C++* не имеет встроенной поддержки русских символов. В случае использования специального резидентного драйвера клавиатуры пере-

ключение 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 ИСР.

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 по-

лета снаряда.

Величина T определяется по формуле

$$T = \frac{2V_{0y}}{g}. \qquad (1.6)$$

Исходя из записанных выше уравнений, алгоритм решения задачи состоит из следующих действий:

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 рисунке 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= 3.141592654;
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
    int 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); // настройка
    cout.setf(ios::left,ios::adjustfield);        // формата выво-
    cout.setf(ios::fixed,ios::floatfield);        // да чисел
    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).

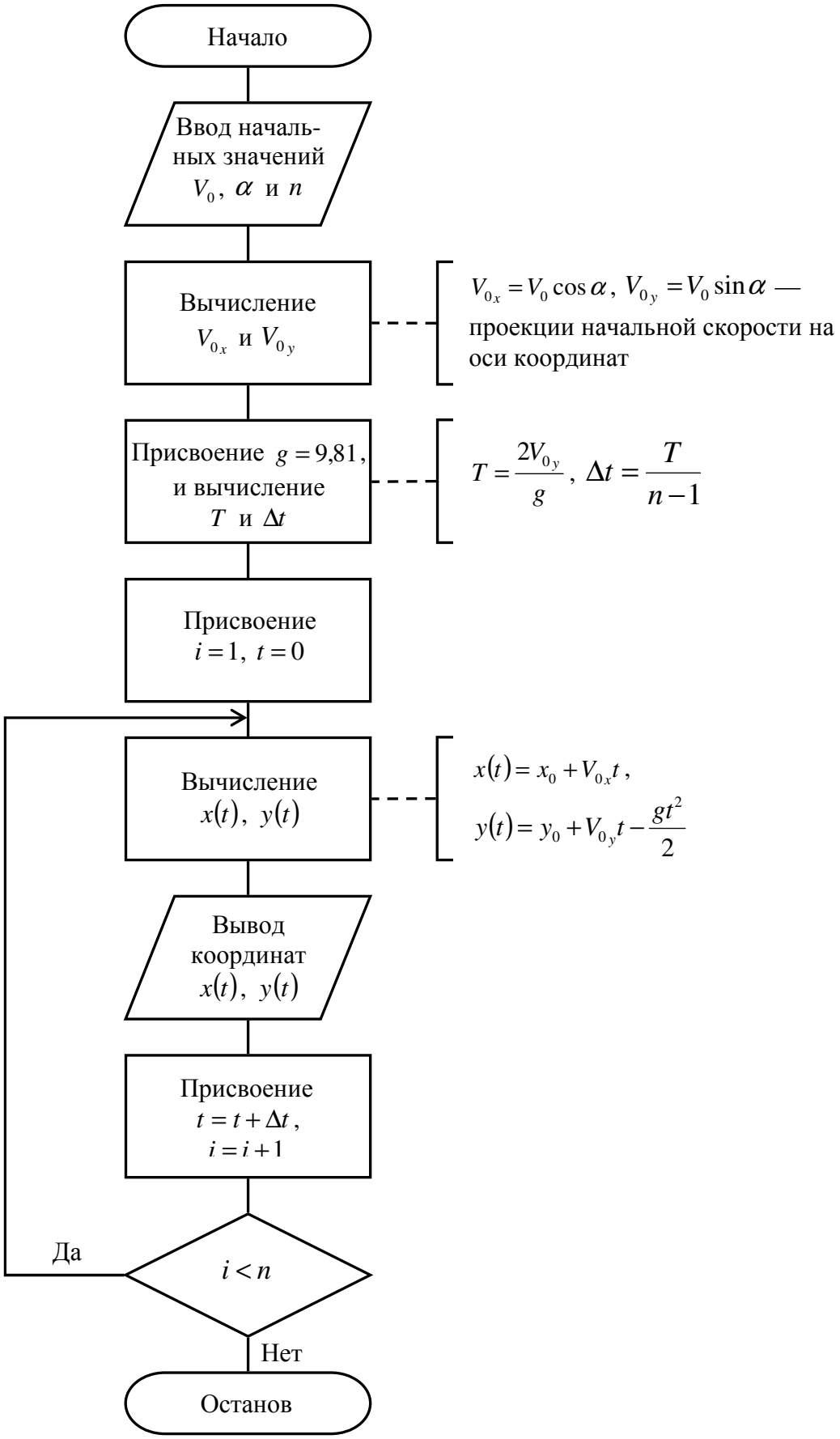


Рисунок 1.1 — Схема алгоритма решения задачи

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

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. Текст программы обязательно должен содержать комментарии.

1.4.5. Составить таблицу, содержащую имена используемых в программе переменных, тип и назначение переменных (см. таблицу 1.1).

Таблица 1.1 — Тип и назначение переменных в программе

Имя переменной	Тип переменной	Назначение переменной

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 происходит 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

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

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

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

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

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

Вариант 15

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

Вариант 16

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

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

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

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, объявление глобальных переменных и констант, а также директивы подключения библиотечных файлов и директивы препроцессора.

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 языке *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:

bool — логический тип, который может принимать либо значение **true** (истина) или **false** (ложь) (не поддерживается компиляторами, выпущенными до 1995 года);

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 при-

менять 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 таблице 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 логического типа (**bool**). 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 **=**, **+=**, **−=**, ***=**, **/=**, **%=**, **&=**, **^=**, **|=**, **<<=**, **>>=**, 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;
a %= b;  // a = a % b;
```

```
a &= b;  // a = a & b;
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 печа-

таться 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 таблицах 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, в которые записываются значения, введенные с клавиатуры с помощью функции **scanf**.

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 манипулятор **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 её формат:

```
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 отме-

тить, 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* её код 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 $\sin x$;
cos(x) — косинус $\cos x$;
tan(x) — тангенс $\operatorname{tg} x$;
asin(x) — арксинус $\arcsin x$ в диапазоне $[-\pi/2;\ \pi/2]$, $x \in [-1;1]$;
acos(x) — арккосинус $\arccos x$ в диапазоне $[0;\pi]$, $x \in [-1;1]$;
atan(x) — арктангенс $\operatorname{arctg} x$ в диапазоне $[-\pi/2;\ \pi/2]$;
atan2(y,x) — арктангенс $\operatorname{arctg}\dfrac{y}{x}$ в диапазоне $[-\pi;\pi]$;
sinh(x) — гиперболический синус $\operatorname{sh} x$;
cosh(x) — гиперболический косинус $\operatorname{ch} x$;
tanh(x) — гиперболический тангенс $\operatorname{th} x$;
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 3.141592654

/* ------------------------------------------------------------>
   Определение логического типа и его значений для компиляторов,
   не поддерживающих логический тип.
   Если компилятор поддерживает логический тип, то строки следует
   закомментировать, либо исключить из программы.              */

#define true  1              // определение значения "истина"
#define false 0              // определение значения "ложь"

typedef unsigned char bool; // определение логического типа
/* <------------------------------------------------------------ */

main()
// вычисление значения функции y=f(x)
{
    float a,       // параметр a
    b,             // параметр b
    x,             // аргумент x
    y;             // значение функции y
    bool valid;    // флаг, отображающий принадлежность аргумента
                   // области допустимых значений (ОДЗ)

    // очистка экрана
    clrscr ();

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

    printf("Введите значение параметра b: ");
    scanf("%f",&b);

    printf("Введите значение аргумента x: ");
    scanf("%f",&x);

    // вычисление значения функции y
    valid = true;
    if (x<=a)      // 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
    {
        // проверка ОДЗ
        if (x<=0)          // если аргумент имеет недопустимое
            valid = false; // значение, то флаг сбрасывается
        else
            y=sqrt(x)*log10(x);
    }

    //вывод значения функции y в формате с плавающей точкой
    if (valid)
        printf("Значение функции y = %e\n",y);
    else
        printf("Недопустимое значение аргумента");

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

    return 0;
}
```

С помощью директивы **#define PI 3.141592654** оказывается возможным указать препроцессору, чтобы он при любом появлении идентификатора **PI** в тексте программы заменял его на значение **3.141592654** (≈π).

Некоторые компиляторы, выпущенные до 1995 года не поддерживают логический тип переменных констант и функций. В этом случае их необходимо объявить самостоятельно. Строки программы

```
#define true  1
#define false 0
```

предназначены для определения значения «истина» (**true**), которое препроцессор заменит значением 1, и для определения значения «ложь» (**false**), которое препроцессор заменит значением 0.

В строке

```
typedef unsigned char bool;
```

объявляется (создается) тип переменной **bool**, который создается из стандартного типа **unsigned char** при помощи команды объявления типа **typedef**.

В случае, если компилятор поддерживает логический тип **bool**, то вышеперечисленные строки программы следует закомментировать, либо исключить из текста программы.

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

Функция **getch** библиотеки **<conio.h>** используется для ввода симво-

лов 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

/* ---------------------------------------------------------->
   Определение логического типа и его значений для компиляторов,
   не поддерживающих логический тип.
   Если компилятор поддерживает логический тип, то строки следует
   закомментировать, либо исключить из программы.              */

const unsigned char true=1;  // определение значения "истина"
const unsigned char false=0; // определение значения "ложь"

typedef unsigned char bool;  // определение логического типа
/* <---------------------------------------------------------- */

const double pi=3.141592654;

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);
```

```
    if (valid)
        cout<<"Значение функции y = f(x) = "<<y<<endl;
    else
        cout<<"Недопустимое значение аргумента"<<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=3.141592654;
```

использует спецификацию **const** для объявления константы **pi**, имеющей значение **3.141592654**. 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. С основными сведениями о функциях и методах построе-

ния графиков функций можно ознакомиться в математическом справочнике [13].

2.4.3. Построить графики ветвей функции (для каждого отрезка значений аргументов). Для построения графиков рекомендуется использовать инженерно-математическую программу *MathCAD* [14].

2.4.4. Используя графики ветвей функций и область определения выбрать значения параметров функции a и b.

2.4.5. Для выбранных значений параметров a и b построить график функции.

2.4.6. Разработать схему алгоритма решения задачи.

2.4.7. Написать две программы вычисления функции y. Одна программа для ввода-вывода данных должна использовать функции **`scanf`** и **`printf`**, а другая — объекты **`cin`** и **`cout`**.

2.4.8. Выполнить тестирование и отладку написанных программ. Особое внимание следует обратить на значения функции y в граничных точках.

2.4.9. Получить результаты работы двух программ для пяти значений аргумента: по одному значению для каждого интервала ($x_1 < a$, $a < x_2 < b$, $x_3 > b$) и для значений в граничных точках ($x_4 = a$ и $x_5 = b$).

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

2.4.10. Подготовить отчет по работе.

2.5. Оформление отчета

2.5.1. Сформулировать цель работы.

2.5.2. Привести постановку задачи и текст индивидуального задания.

2.5.3. Привести краткие теоретические сведения, касающиеся 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 допустимых значений аргумента;

— графики ветвей функции;

— выбор граничных точек a и b;

— график функции с учетом выбранных граничных точек a и b;

— выбор значений аргумента для ввода в программы.

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

2.5.6. Привести тексты программ. Тексты программ обязательно должны содержать комментарии.

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 работы программ для пяти значений аргумента.

2.5.9. 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. Варианты функций y выбирать по указанию преподавателя из таблицы 3.1. Граничные значения аргумента $x_{\text{нач}}$, $x_{\text{кон}}$ и шаг Δx выбираются произвольно и вводятся с клавиатуры.

Результаты вычислений выводить в формате с фиксированной точкой.

Таблица 3.1 — Исходные данные к лабораторной работе № 3

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

Продолжение таблицы 3.1

Вариант	Функция y	Вариант	Функция y
25	$y = 1/\left(\mathrm{e}^{x}\cos x\right)$	28	$y = \mathrm{tg}\, x/x$
26	$y = 1/\sin x$	29	$y = \cos(3x)/(2x-1)$
27	$y = \sin^2 x/x^2$	30	$y = \sin(3x+1)/(3x+1)$

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

Оператор цикла **while** имеет следующий формат:

```
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** завершается 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

Оператор цикла **do while** имеет следующий формат:

```
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

Оператор цикла **for** имеет следующий формат:

```
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 (**выражение1**) 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. таблицу 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

$$y = f(x) = \frac{1}{2x - 5}.$$

Значения границ интервала $x_{\text{нач}}$, $x_{\text{кон}}$ и шаг 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: $x_{\text{нач}}$, $x_{\text{кон}}$ и Δx.
2) Вывести заголовок и шапку таблицы.
3) Положить $x = x_{\text{нач}}$.
4) Проверить область допустимых значений (ОДЗ) аргумента.
5) Если аргумент находится в области ОДЗ, то вычислить значение функции y по вышеприведенной формуле.
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…8 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 x,            // аргумент функции y
          xn,           // начальное значение аргумента x
```

```
        xk,           // конечное значение аргумента x
        dx,           // шаг
        y;            // значение функции y
    clrscr ();

    // ввод xn, xk, dx
    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
           if ((2*x-5) != 0)        // проверка ОДЗ
           {   // вычисление значения функции y
               y = 1/(2*x-5);
               // вывод значения функции y
               printf(" %-10.3f│\n",y); // вещественное y
           }
           else
           {   // вывод сообщения о выходе аргумента из ОДЗ
               printf(" Нет решений │\n");
           }
           x+=dx; // приращение аргумента x
    }
    printf (" └──────────┴──────────────┘\n");

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

    return 0;
}
```

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 псевдогра-

фики представлены в таблице 3.2.

Таблица 3.2 — 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 xn, xk, dx
    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
        cout<<" "<<setw(13);
        if ((2*x-5) != 0)              // проверка ОДЗ
        {   // вычисление значения функции y
            y = 1/(2*x-5);
            // вывод значения функции y
            cout << y; // вещественное y
        }
        else
        {   // вывод сообщения о выходе аргумента из ОДЗ
            cout <<" Нет решений ";
        }
        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** (количество 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 ме-

тодических указаний (см. раздел 2.3.4).

3.4. Порядок выполнения работы

Вариант задания выдается преподавателем. Перед выполнением работы необходимо ознакомится с разделами 3.1 — 3.3 данных методических указаний. Индивидуальные задания для выполнения работы см. в разделе 3.2.

3.4.1. Внимательно изучить индивидуальное задание.

3.4.2. Выполнить анализ области определения и области значений вычисляемой функции y.

3.4.3. Построить график заданной функции. Для построения графиков рекомендуется использовать инженерно-математическую программу *MathCAD* [14].

3.4.4. Используя построенный график и область определения функции выбрать значения границ интервала табуляции функции и шаг изменения аргумента таким образом, чтобы на экран выводилось от 10-ти до 20-ти значений.

3.4.5. Разработать схему алгоритма решения задачи.

3.4.6. 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`**. Ширину полей, отводимых для вывода значений функции y (целой и дробной части) следует определить самостоятельно, в соответствии с величинами функции y.

3.4.7. Выполнить тестирование и отладку написанных программ. Особое внимание следует обратить на вычисление функции в граничных точках области допустимых значений.

3.4.8. Получить результаты работы двух программ. Рекомендуется скопировать в виде изображения сообщения, выводимые программами, для последующего включения в отчет.

3.4.9. Подготовить отчет по работе.

3.5. Оформление отчета

3.5.1. Сформулировать цель работы.

3.5.2. Привести постановку задачи и текст индивидуального задания.

3.5.3. Привести краткие теоретические сведения касающиеся использования операторов цикла.

3.5.4. Привести математическое обоснование задачи:

— анализ области допустимых значений аргумента;

— график функции;

— выбор граничных значений интервала табулирования функции и шага табулирования.

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

3.5.6. Привести тексты программ. Тексты программ обязательно должны

содержать комментарии.

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

3.5.8. Привести результаты работы программ.

3.5.9. 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`**.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ПРИЛОЖЕНИЕ А
ПРАВИЛА ОФОРМЛЕНИЯ СХЕМ АЛГОРИТМОВ

А.1. Общие положения

Данные правила составлены на основе ГОСТ 19.701-90 [3], 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 (рисунок А.1). Символ «данные» отображает данные, расположенные на неопределенном носителе.

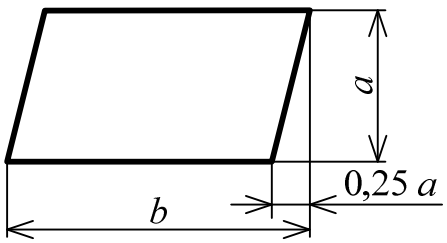


Рисунок А.1 — Данные

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 (рисунок А.2,а), так и специфические символы процесса (рисунок А.2,б,в,г и рисунок А.3).

Символ «процесс», изображенный на рисунке 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» (рисунок 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 рисунок 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.

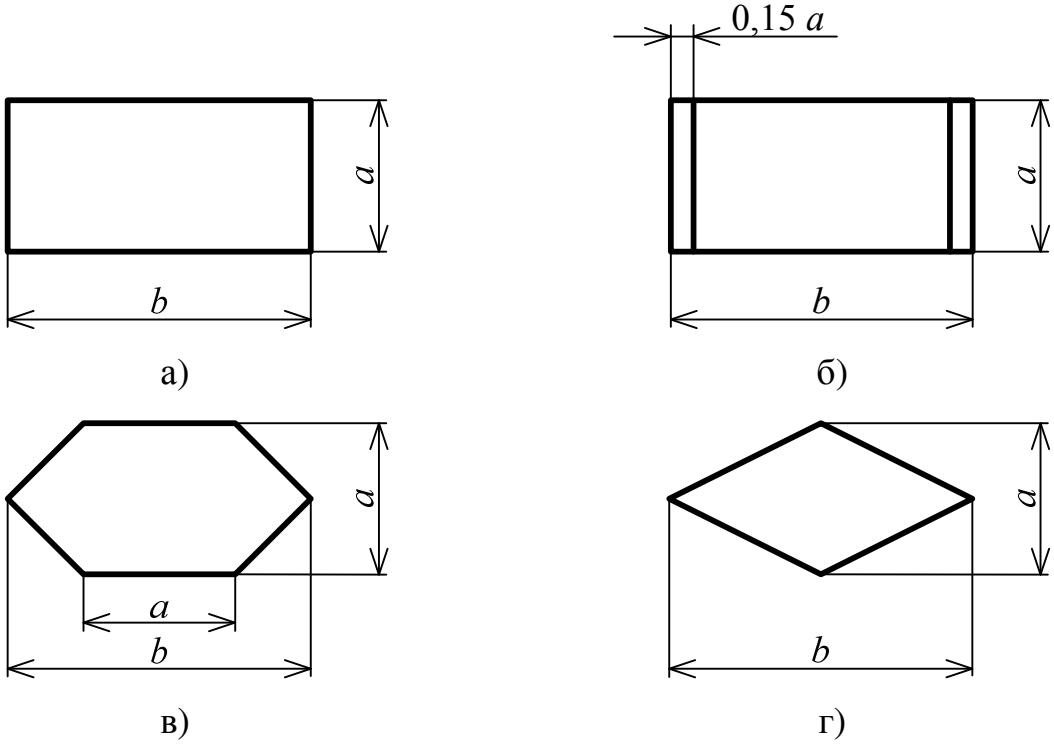


Рисунок А.2 — Процесс (а), предопределенный процесс (б), подготовка (в) и решение (г)

Символ «решение» (см. рисунок 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, активи-

зирующие выходы, могут быть записаны по соседству с линиями, отображающими пути выходов. Используется символ для отображения выбора направления выполнения алгоритма или программы в зависимости от некоторых переменных условий, т.е для описания условия.

На рисунке 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.

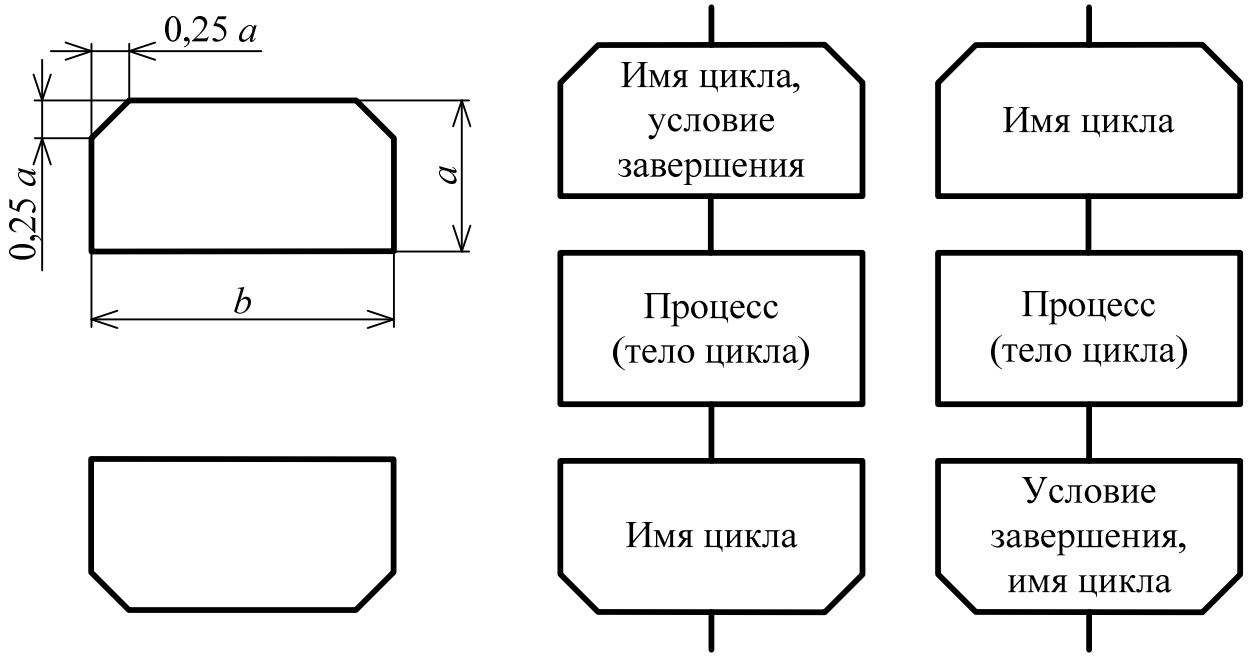


Рисунок А.3 — Граница цикла (а) и примеры использования символа (б)

А.2.3. Символы линий

При изображении схем алгоритмов программ применяют символы «линия», который является основным, и «пунктирная линия», являющийся специфическим (рисунок 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. рисунок А.6).

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

А.2.4. Специальные символы

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

Изображенный на рисунок 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» (рисунок 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» (рисунок 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 (рисунок 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 (рисунок А.5,г). Применяется главным образом в схемах, изображающих общие решения с неизвестным числом повторений. На рисунке А.7 изображен пример использования символа «пропуск».

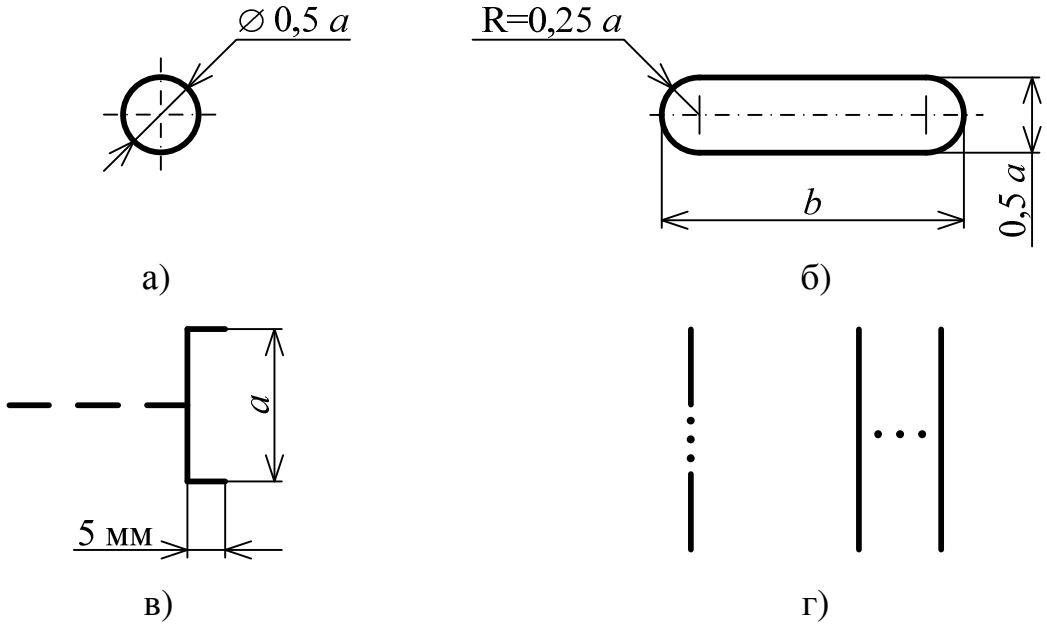


Рисунок А.5 — Соединитель (а), терминатор (б),
комментарий (в) и пропуск (г)

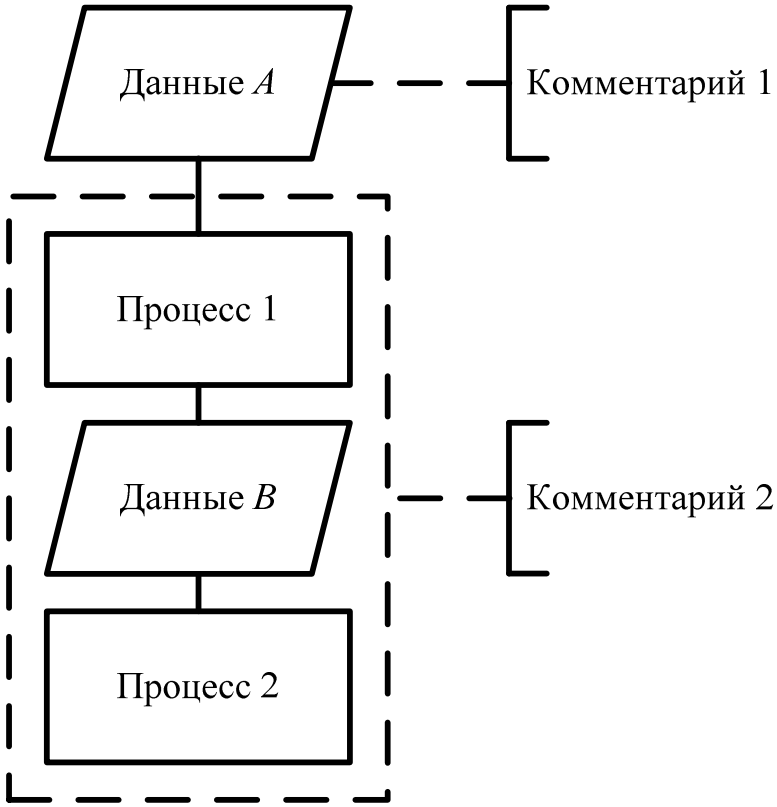


Рисунок А.6 — Пример использования комментария

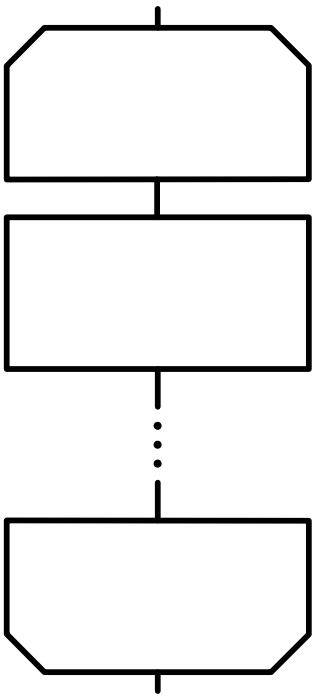


Рисунок А.7 — Пример использования пропуска (троеточия)

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.

Выбранные размеры a и b для всех элементов одной схемы одинаковы.

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 (рисунок 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 (рисунок 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 (рисунок А.10).

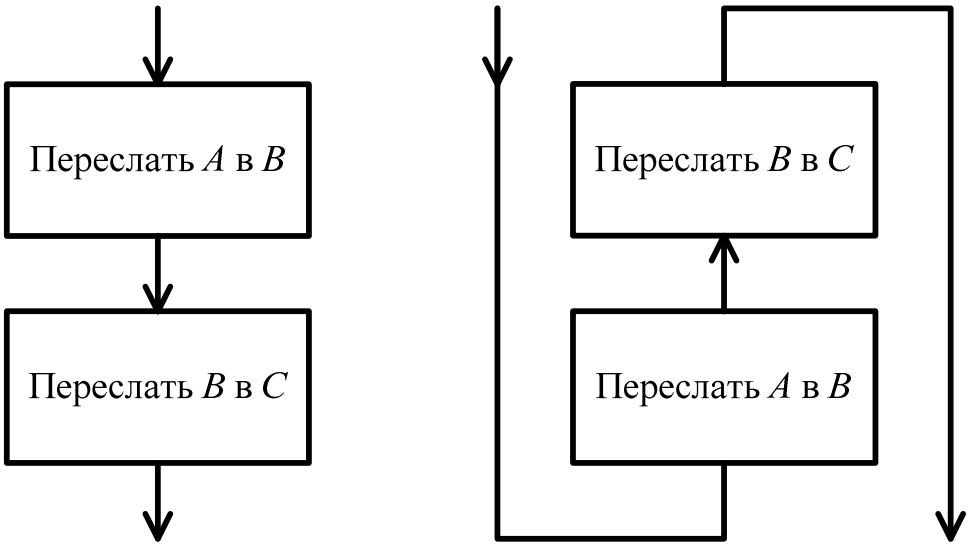


Рисунок А.8 — Запись текста внутри символов

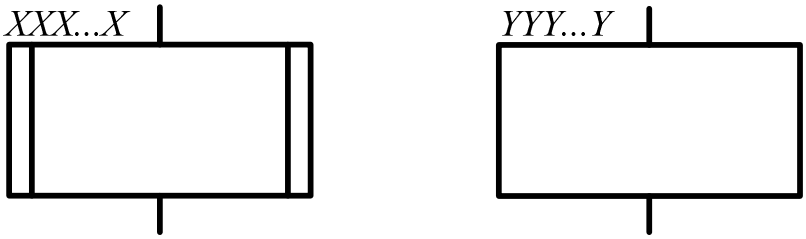


Рисунок А.9 — Расположение идентификаторов

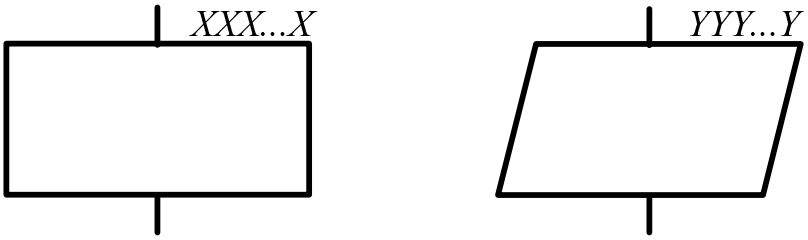
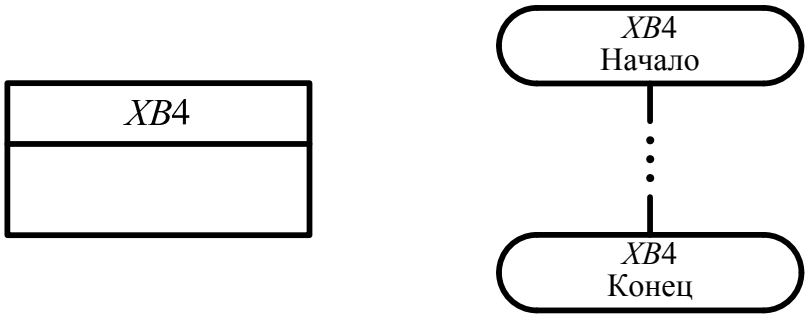


Рисунок А.10 — Расположение описаний символов



а) б)

Рисунок 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. рисунок 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. рисунок А.11,б).

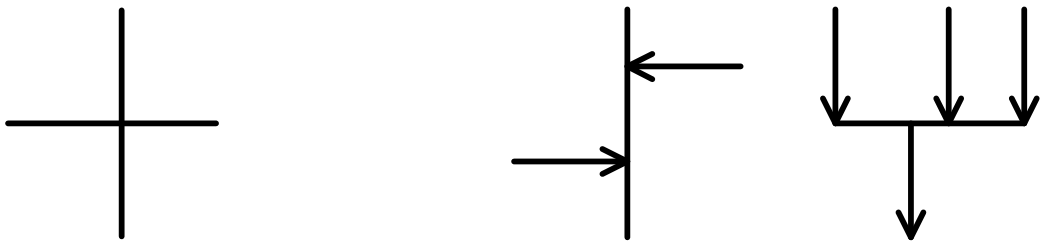
А.4.2. Правила выполнения соединений

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 (рисунок 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 (рисунок А.12,б).



а) б)

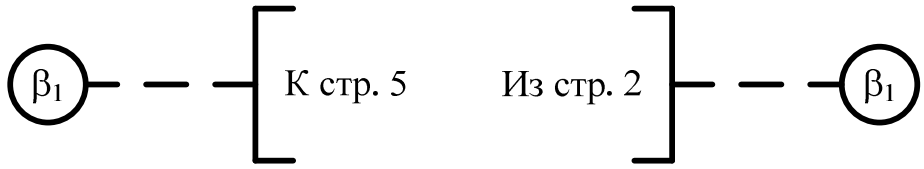
Рисунок А.12 — Примеры пересечения линий (а) и объединения линий (б)

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 (рисунок А.13).



а) б)

Рисунок 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 рисунке А.14.

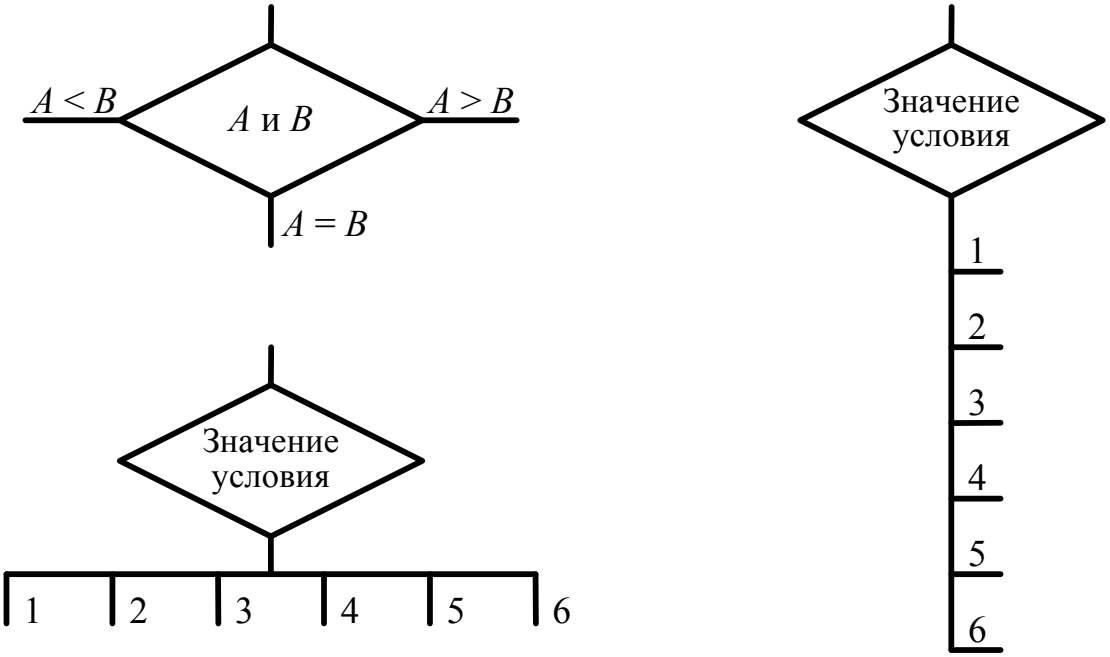


Рисунок А.14 — Варианты применения нескольких выходов из символа

А.6. Примеры изображения схем алгоритмов

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

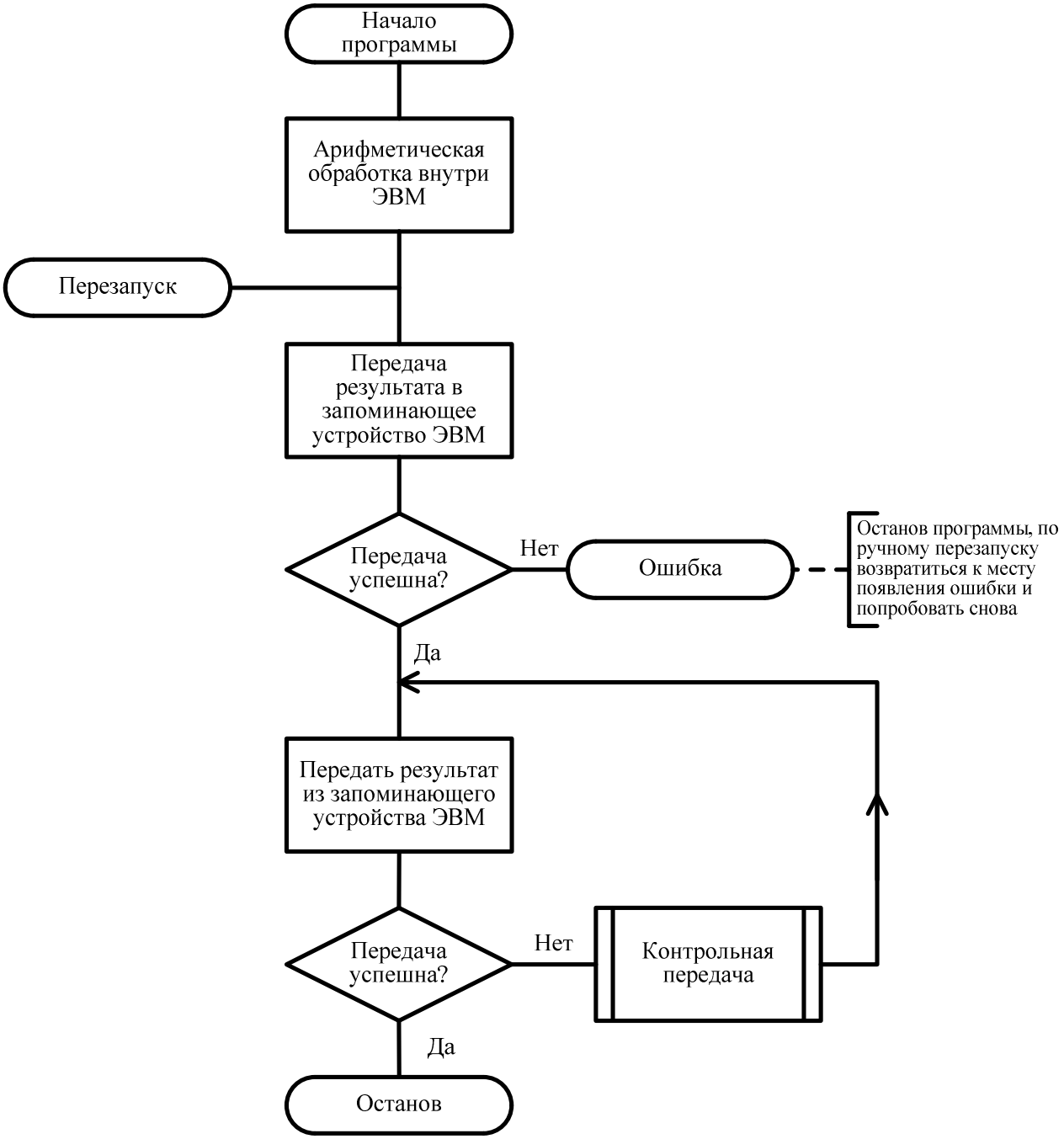


Рисунок А.15 — Первый пример изображения схемы программы [3]

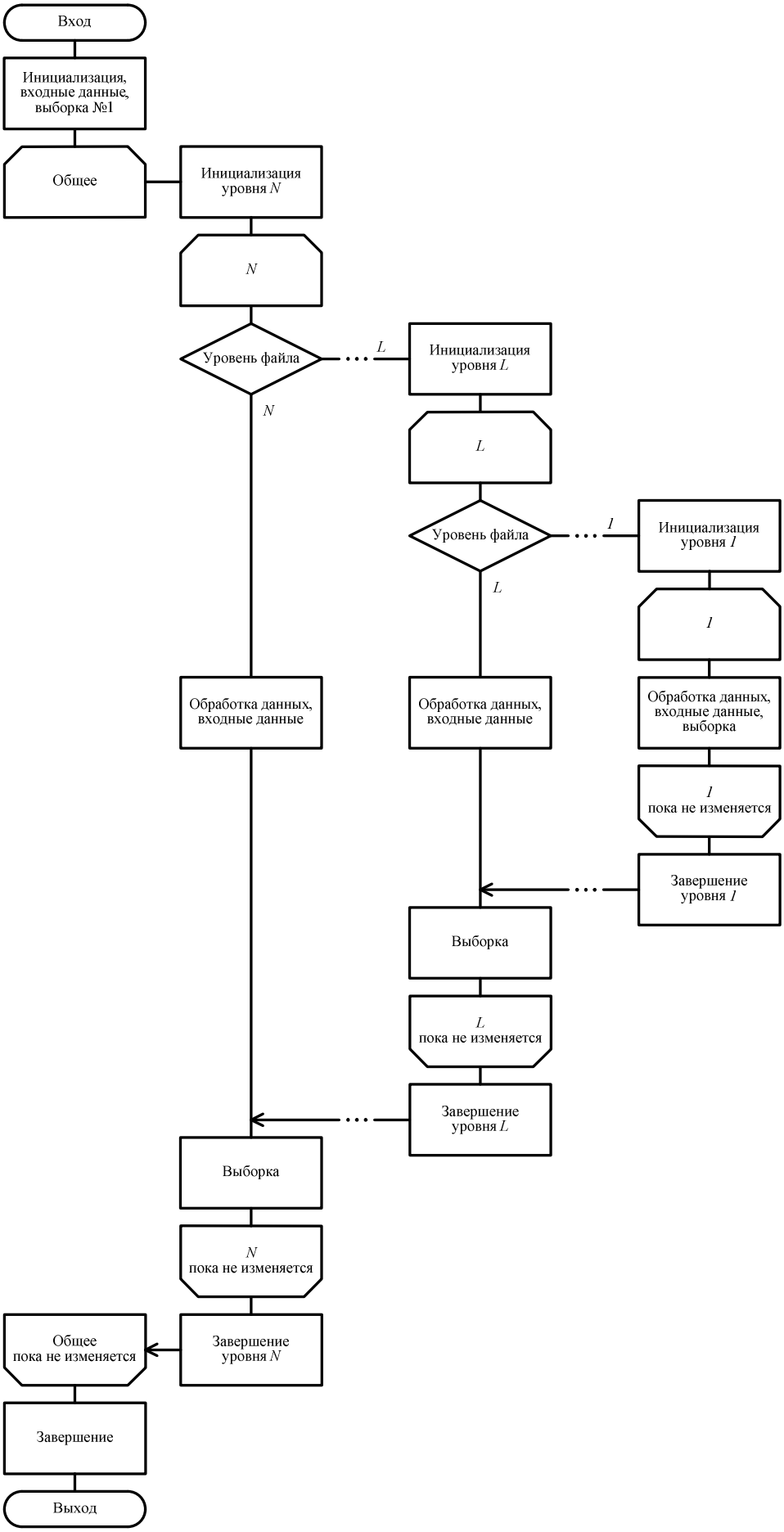
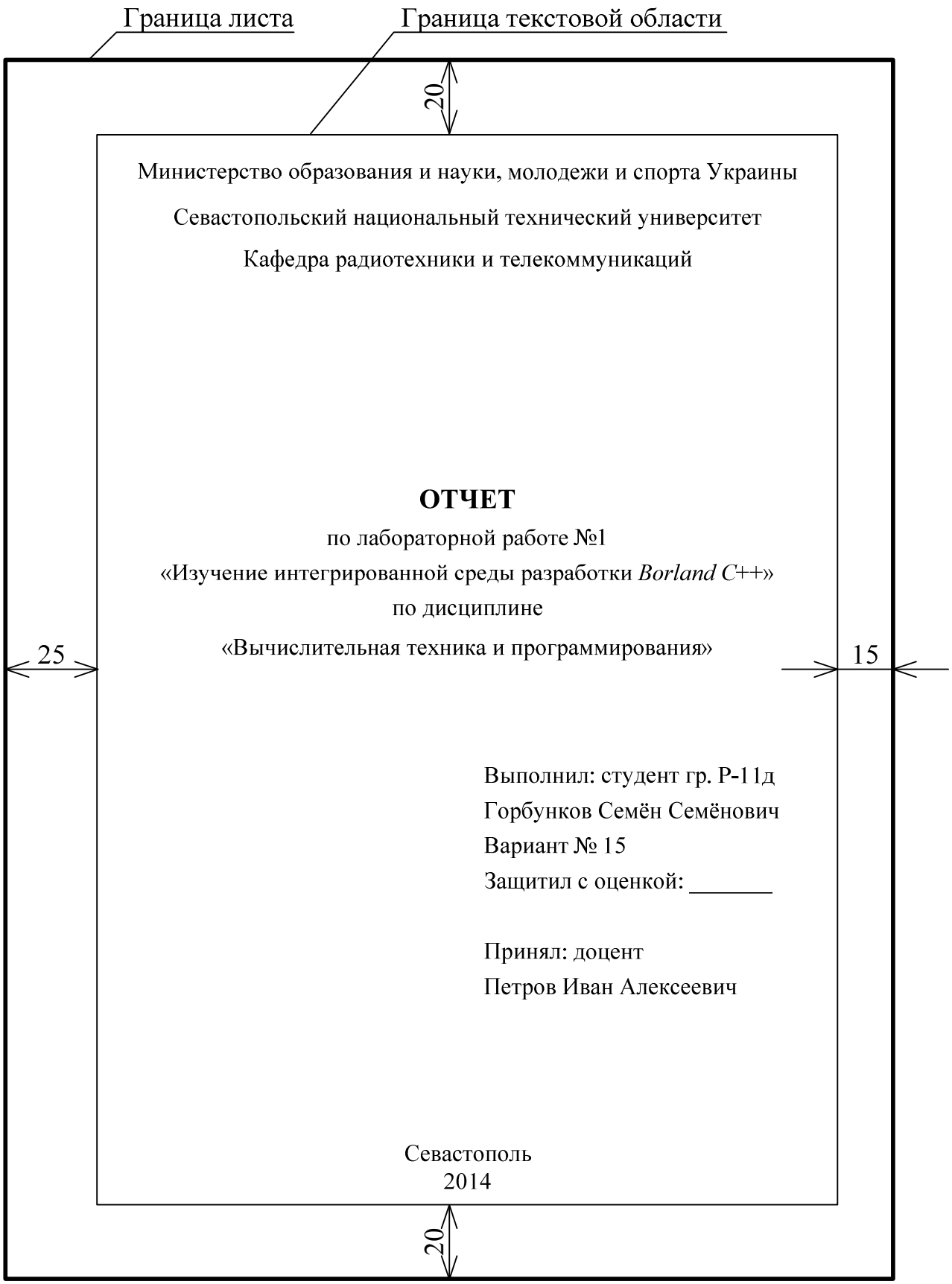


Рисунок А.16 — Второй пример изображения схемы программы [3]

ПРИЛОЖЕНИЕ Б

ПРИМЕР ОФОРМЛЕНИЯ ТИТУЛЬНОГО ЛИСТА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ



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

Изд-во СевНТУ