МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Омский государственный технический университет»

# ПРОГРАММИРОВАНИЕ C/C++. ПРОЕКТИРОВАНИЕ АЛГОРИТМОВ И ПРОГРАММ

*Учебное текстовое электронное издание*
*локального распространения*

Омск
Издательство ОмГТУ
2017

Сведения об издании: 1, 2

Составитель *О. П. Шафеева*, канд. техн. наук, доцент кафедры ИВТ

**Программирование С/С++. Проектирование алгоритмов и программ** : метод. указания / Минобрнауки России, ОмГТУ ; [сост. О. П. Шафеева]. – Омск : Изд-во ОмГТУ, 2017.

В методических указаниях к практическим занятиям и самостоятельной работе студентов рассмотрены правила разработки алгоритмов и программ, элементы языка программирования С/С++. Приведены примеры проектирования программ и задания для самостоятельного выполнения.

Для студентов, обучающихся по направлениям 09.03.01 «Информатика и вычислительная техника», 09.03.02 «Информационные системы и технологии», 09.03.03 «Прикладная информатика», 09.03.04 «Программная инженерия», 02.03.02 «Фундаментальная информатика и информационные технологии», 02.03.03 «Математическое обеспечение и администрирование информационных систем», 27.03.03 «Системный анализ и управление».

*Рекомендовано редакционно-издательским советом*
*Омского государственного технического университета*

1 электронный оптический диск

Оригинал-макет издания выполнен в Microsoft Office Word 2007/2010
с использованием возможностей Adobe Acrobat Reader.

**Минимальные системные требования:**

- процессор Intel Pentium 1,3 ГГц и выше;
- оперативная память 256 Мб и более;
- свободное место на жестком диске 260 Мб и более;
- операционная система Microsoft Windows XP/Vista/7/10;
- разрешение экрана 1024×768 и выше;
- акустическая система не требуется;
- дополнительные программные средства Adobe Acrobat Reader 5.0 и выше.

Редактор *А. Ю. Леонтьева*
Компьютерная верстка *О. Г. Белименко*

Сводный темплан 2017 г.
Подписано к использованию 07.07.17.
Объем 0,49 Мб.

---

Издательство ОмГТУ.
644050, г. Омск, пр. Мира, 11; т. 23-02-12
Эл. почта: info@omgtu.ru

# ПРЕДИСЛОВИЕ

Во время практических занятий по дисциплине «Программирование» вместе со студентами анализируется процесс разработки алгоритмов и программ для задач различной сложности. Этот процесс включает этапы физического и математического анализа, алгоритмизации, непосредственно написания программы, ее отладку и тестирование, при необходимости этапы оптимизации и документирования.

Методические указания содержат 9 тем, их очередность определяет последовательность освоения материала. Принцип расположения – от простого к сложному.

Первые темы посвящены изучению отдельных операторов языка C/C++, правил работы с массивами, подпрограммами, последние – разработке сложных алгоритмов восходящим и нисходящим методами. В каждой теме при анализе операторов языка или других элементов приводится их графическое изображение на схеме алгоритма, синтаксические правила записи, а также примеры спроектированных и отлаженных программ.

В проектировании и отладке ряда программ участвовал студент группы ИВТ-151 Руслан Маратович Касымов.

*Автор*

# 1. ПРОЕКТИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ И ПРОГРАММ

*Пример 1.* Спроектировать алгоритм, начертить схему алгоритма (СА) и подготовить программу для вычисления $y = \sin x + \cos x$ при $x = 0{,}5$.

*Физический и математический анализ задачи.* Необходимо вычислить значение переменной *y* по заданной формуле с использованием математических функций.

*Проектирование алгоритма.* В алгоритме необходимо предусмотреть задание переменной *x*, вычисление *y* и вывод результата.

Представим алгоритм для этой задачи в виде схемы, которая выполнена по ГОСТ 19.701–90 [1] (часть условных обозначений блоков показана в прил. А). Для реализации задачи разработана СА, которая приведена на рис. 1.

*Разработка программы.* Для программирования понадобиться оператор присваивания и функция форматированного вывода на экран, исходное значение *x* задается при объявлении переменной.

Оператор «выражение», содержащий операцию присваивания (прил. Б), имеет вид

**<идентификатор> = <выражение>;**

Функция форматированного вывода записывается в следующем формате:

**printf** («управляющая строка», список вывода);

Кроме того, необходимо подключить библиотеки.

*Текст программы:*

```
#include <conio.h> // для функции getch()
#include  <math.h> // для математических функций
#include  <stdio.h> // для функций ввода-вывода
void  main()
{
  float  y, x = 0.5; // плавающий тип (прил. В)
  y = sin(x) + cos(x);   /* Оператор выполняет присваивание  y вычисляемого значения */
  printf(«y = sin(x) + cos(x) = %f\n», y);  // Вывод  y
  getch();  /* задержка информации на экране до  нажатия любой клавиши  */
}
```

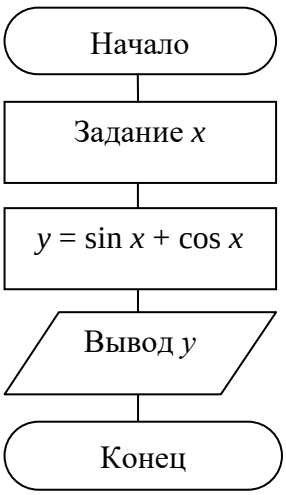



*Рис. 1*

Далее выполняется компиляция программы, отладка и тестирование.

*Пример 2*. Разработать алгоритм и программу для нахождения площади прямоугольного треугольника по вводимым сторонам *A* и *B*.

Для ввода данных применяется следующая функция:

**scanf** («управляющая строка», список адресов вводимых переменных);

*Анализ задачи*. Если катеты треугольника имеют размеры соответственно *A* и *B*, то его площадь вычисляется как *S* = *A*·*B*/2.

Спроектированная СА приведена на рис. 2.

*Текст программы:*

```
#include  <stdio.h>
#include <conio.h> // для функции getch();
#include <locale>  // для функции setlocale
void main()
{
setlocale(LC_ALL, «RUS»); //для вывода русских букв
int  A, B;   //объявление переменных целого типа
float  S;   //объявление переменной плавающего типа
printf(«Введите A и B»);
scanf («%d %d», &A, &B);     // ввод A и B
S = (float)A * B/2; // преобразование к типу «float»
printf («S=%6.2f для A=%d и B=%d \n», S, A, B);
getch(); // задержка до нажатия любой клавиши
}
```

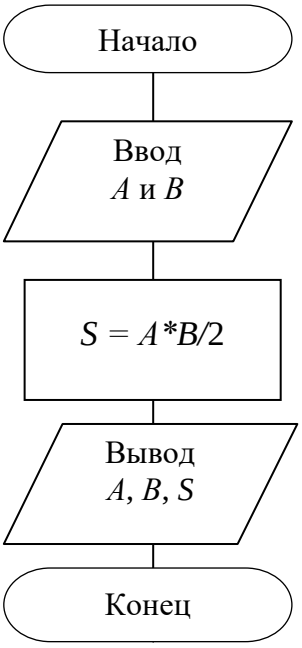



*Рис. 2*

В функции вывода запись «*S*=%6.2f» означает, что значение *S* нужно выводить как вещественное число в естественном виде, причем общее поле вывода – 6 знаков, после десятичной точки – 2 знака.

## Задачи по теме

Разработать алгоритмы и подготовить программы для задач.

1. Вычислить $z = e^{\sin x} - a \cdot \sin x$.

2. Вычислить $y = \dfrac{2ab - S^2}{7a^{-1} - 3{,}2}$, где $S$ – переменная, $a$ = 0,384, $b$ = –0,293.

## Задания для самостоятельного выполнения

1. Определить площадь треугольника и его высоты по вводимым длинам сторон *A, B, C*.

Математический анализ предполагает вывод или использование известных математических формул для вычисления площади $S = \sqrt{p(p-a)(p-b)(p-c)}$, где $p = \frac{a+b+c}{2}$ – полупериметр, $h_a = \frac{2S}{a}$ – высота стороны $a$; $h_b = \frac{2S}{b}$ – высота стороны $b$; $h_c = \frac{2S}{c}$ – высота стороны $c$.

2. Ввести $K$ ($K > 4000$) – число секунд. Напечатать правильное время, вычисляя часы, минуты и секунды (например, 3 часа 10 минут 5 секунд).

## 2. РАЗРАБОТКА РАЗВЕТВЛЕННЫХ АЛГОРИТМОВ

*Пример 1.* Спроектировать алгоритм, начертить СА, написать и отладить программу для определения попадания точки в квадрат со стороной $a$ и центром в начале координат.

*Физический анализ задачи.* Представим квадрат на плоскости (рис. 3).

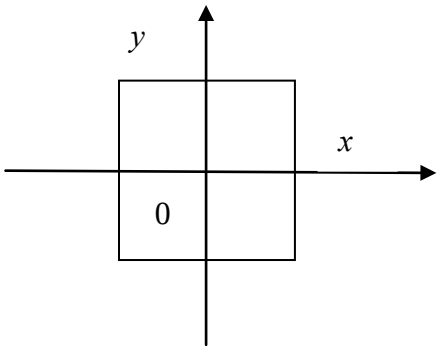


*Рис. 3*

*Математический анализ.* Точка с координатами ($x$, $y$) попадет в квадрат со стороной $a$, если $|x| < a/2$ и $|y| < a/2$.

*Алгоритмизация.* Словесно-формульная запись:

1. Ввести исходные данные: размер стороны квадрата $a$, координаты точки ($x$, $y$).

2. Проверить значения каждой из координат: оказались ли они по модулю меньше половины стороны квадрата? Если да, напечатать «Точка в квадрате», если нет – «Точка не попадает в квадрат».

*Проектирование схемы алгоритма.* СА состоит из блоков, условные обозначения для которых приведены в прил. А. Для рассматриваемой задачи такая схема изображена на рис. 4.

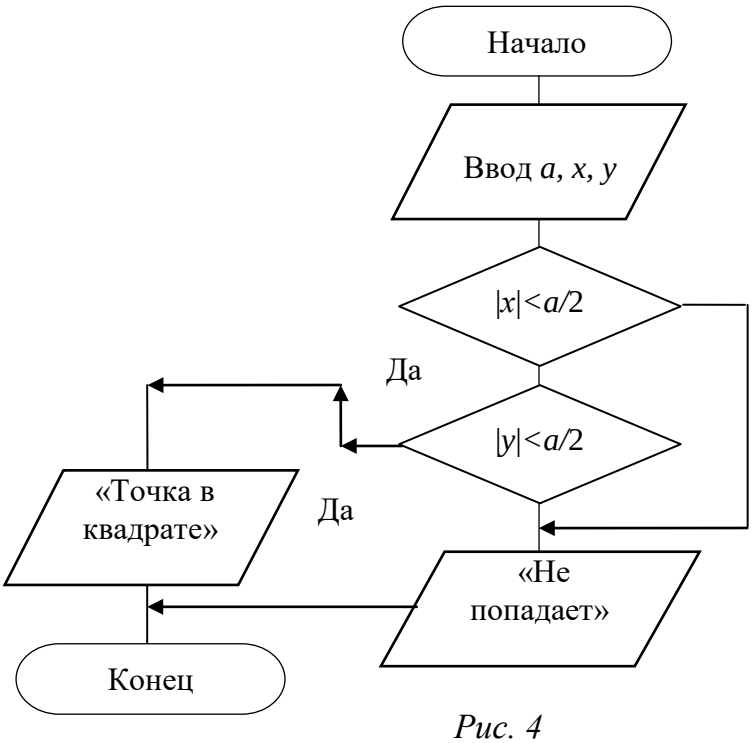


*Рис. 4*

*Разработка программы.* Для реализации разветвления на две ветви на языке С/С++ применяется условный оператор (прил. Г).

Структура *условного оператора* и обозначение на СА (рис. 5) имеет вид:

**if** (<**выражение**>)
 {
 <**оператор 1**>;
 }
 **else**
 {
 <**оператор 2**>;
 }

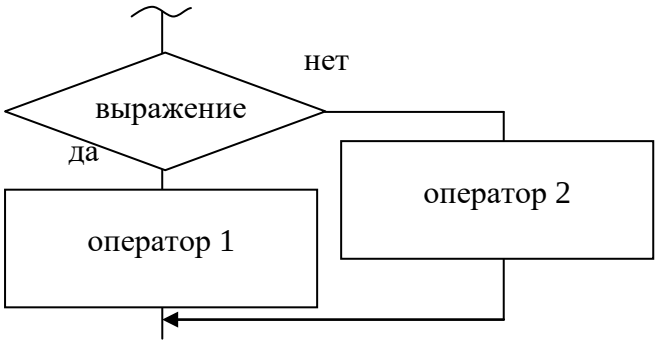



*Рис. 5*

где <**выражение**> – это условие или произвольное выражение, которое может принимать два значения: истина (ненулевое) и ложь (нуль), <**оператор**> – любой оператор языка, в том числе и составной (если в любой из частей записывается простой оператор, то фигурные скобки можно опустить). На СА такой оператор изображается ромбом (рис. 5).

*Текст программы* (для спроектированной на рис. 4 схемы):

```
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <locale>   // для функции setlocale
void main()
{
setlocale(LC_ALL, «RUS»);
 //для русских букв
float  a, x, y;   /* объявление переменных плавающего типа */
printf(«Введите a, x, y:»);
 // или printf_s
scanf(«%f %f %f»,&a, &x, &y);
 // ввод a, x, y; можно scanf_s
if (fabs(x < a/2) && fabs(y < a/2))
   printf («Точка в квадрате \n»);
else printf(«Точка не попадает в квадрат \n»);
getch(); // или _getch();
}
```

Особенностью разработки программы является объединение двух условий в одном условном операторе.

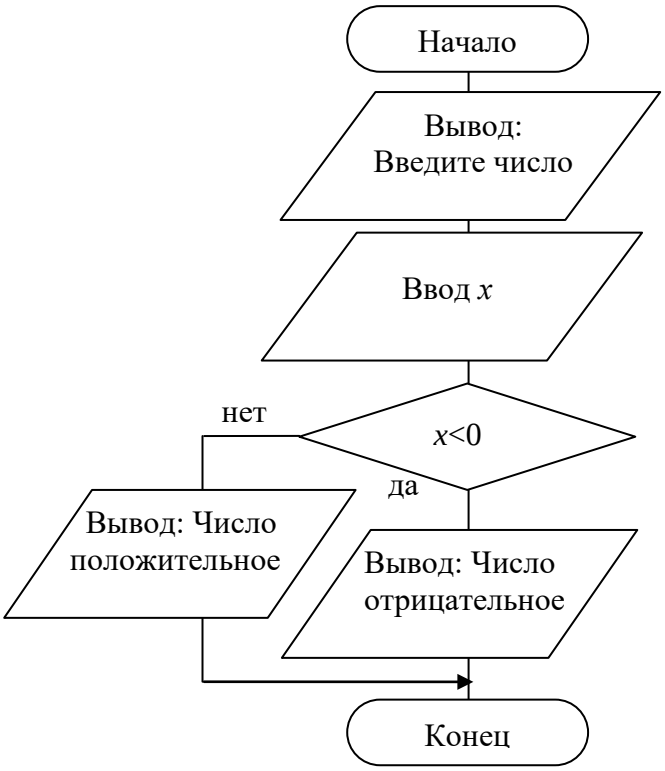



*Рис. 6*

*Отладка и тестирование.* На этом этапе программа компилируется, при необходимости исправляются синтаксические ошибки, которые «подсказывает» компилятор языка. Затем программа тестируется на заранее рассчитанных примерах для всех ветвей алгоритма [4]: полученные результаты сравниваются с эталонными ответами. Если тест полный, то делается вывод о том, что программа отлажена.

*Пример 2.* Определить знак введенного числа.

Разработаем СА (рис. 6) и подготовим *текст программы*:

```
#include <stdio.h>
#include <conio.h>
#include <locale>
void main()
{
setlocale(LC_ALL, «RUS»);
int x;
printf(«Введите  число \n»);
scanf(«%d», &x); // Ввод x
if (x<0) printf(«Число отрицательное \n»);
else printf(«Число положительное \n»);
  getch();
}
```

## Задачи по теме

Разработать алгоритмы и написать программы для задач.

1. Вычислить $Z = \begin{cases} 0{,}4 / x, & \text{если} \quad x < 0, \\ 2x, & \text{если} \quad 0 \le x \le 3, \\ \sqrt{x}, & \text{если} \quad x > 3. \end{cases}$

2. Найти действительные корни уравнения $ax^2 + bx + c = 0$ по заданным коэффициентам *a, b, c*.

На этапе математического анализа определим формулы:

а) действительные корни: $x_{1,2} = \dfrac{-b \pm \sqrt{D}}{2a}$, где $D = b^2 - 4ac$ – дискриминант;

б) мнимые корни: $x_{1,2} = \alpha \pm i\beta$; $\alpha = -\dfrac{b}{2a}$; $\beta = \dfrac{\sqrt{|D|}}{2a}$.

*Многовыходные разветвления. Оператор switch.* Оператор выбора **switch** позволяет в зависимости от значения какой-либо переменной или выражения (ключа выбора) выполнить те или иные операторы, помеченные соответствующими константами.

СА для оператора-переключателя (селектора) изображена на рис. 7 (где <выражение> – выражение (переменная) любого порядкового типа; <константа> – константа того же типа, что и <выражение>; <оператор> – произвольный оператор C/C++).

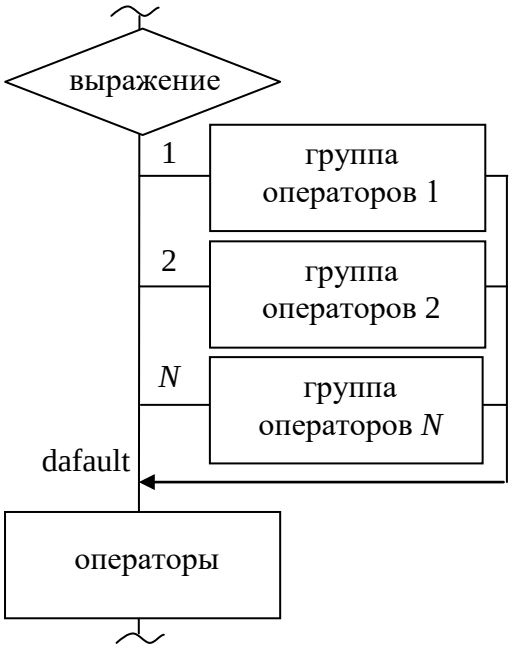



*Рис. 7*

Структура оператора:

**switch** (<выражение>)
**{**
**case**<константа 1>:
{<группа операторов 1>; break;}
**case**<константа 2>:
{<группа операторов 2>; break;}
. . .
**case**<константа N>:
{<группа операторов N>; break;}
**default:** {<операторы>;}
**}**

*Пример 3.* Напечатать введенную цифру (1–5) на английском языке. Для реализации использовать оператор-переключатель.

СА приведена на рис. 8.

*Текст программы:*

```
#include <stdio.h>
#include <conio.h>
#include <locale>
void main()
{
setlocale(LC_ALL, «RUS»);
```

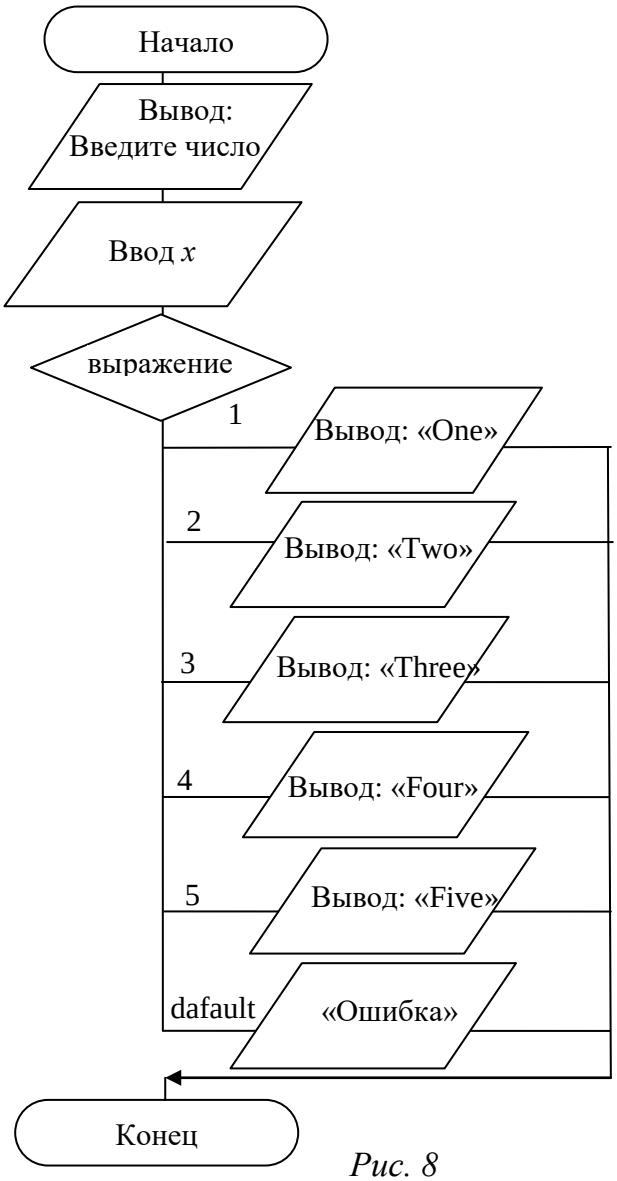



*Рис. 8*

```
int x; //Объявление переменной x
     printf(«Введите   цифру от 1 до 5 \n»);
scanf(«%d», &x);  // Ввод x
switch (x)
  {
  case 1: { printf(«One \n»); break; }
  case 2: { printf(«Two \n»); break; }
  case 3: { printf(«Three \n»); break; }
  case 4: { printf(«Four \n»); break; }
  case 5: { printf(«Five \n»); break; }
  default:{ printf(«Ошибка \n»); }
  }
getch();
}
```

## Задачи по теме

Разработать алгоритмы и написать программы для задач.

1. Ввести номер месяца. Определить, к какому времени года он относится.

2. Цифру, введенную с клавиатуры, напечатать прописью.

3. Десятичное число (меньше 17), введенное с клавиатуры, преобразовать в шестнадцатеричную цифру.

Этапы проектирования и дополнительные примеры приведены в [2, 3, 5, 7, 8, 9].

## Задания для самостоятельного выполнения

1. Определить, попадет ли точка с координатами ($x$, $y$) в круг радиуса $R$ с центром в начале координат (уравнение окружности $x^2 + y^2 = R^2$).

2. Вычислить площадь треугольника с проверкой условий его существования (сумма двух сторон должна быть больше третьей: $A + B > C$, $A + C > B$, $B + C > A$).

3. Напечатать номер четверти координатной плоскости, в которую попадает точка с вводимыми координатами ($X$, $Y$).

4. В зависимости от введенного значения $N$ ($N < 50$) сформировать сообщение «В моем доме … этажей». В нужных случаях слово «этажей» заменить на слова «этаж» или «этажа» (для чисел $N > 20$ предварительно выделить (операция «%») младшую цифру).

# 3. ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ

## 3.1. Циклические алгоритмы с известным числом повторений

Оператор *цикла с заданным числом повторений* (*с параметром*) имеет вид:

**for (<выражение1> ; < выражение2> ; <выражение3>) {<оператор>;}**

где **<выражение1>** – это выражение инициализации цикла (может содержать несколько операций, разделенных запятыми, это действия, которые выполняются до начала цикла); **<выражение2>** – выражение или условие, при истинности (ненулевом значении) которого происходит повторение выполнения оператора цикла; **<выражение3>** – изменение параметра цикла (не обязательно целое) (действия, которые выполняются в конце каждой итерации).

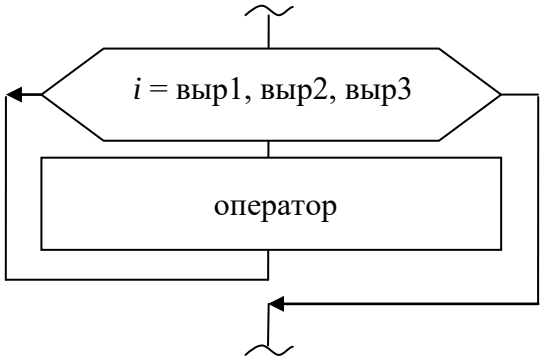



*Рис. 9*

Такой оператор цикла изображен на рис. 9.

*Пример 1*. Спроектировать алгоритм, начертить СА, написать и отладить программу для вычисления факториала для числа 10. Определение факториала сводится к вычислению произведения чисел натурального ряда от 1 до 10.

Разработанная СА изображена на рис. 10.

*Текст программы:*

```
#include <stdio.h>
#include <conio.h>
void  main()
{
long factorial = 1; /* Объявление  переменной (типом long int – длинное целое со знаком */
for (int i = 1; i<=10; i++)
  /*Вычисление  факториала */
  {
        factorial = factorial * i;   /*  При i = 11 итерация выполняться не будет */
  }
  printf(«10! = %ld \n», factorial); /* Вывод полученного ответа типа long int */
```

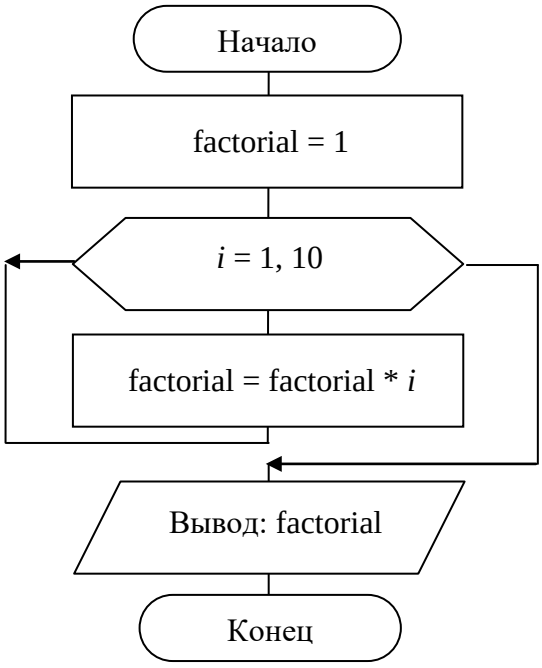



*Рис. 10*

```
  getch(); // Задержка информации на экране
}
```

*Пример 2*. Вычислить сумму квадратов чисел от 1 до $N$. $S = 1 + 2^2 + 3^2 + + 4^2 + \ldots + 2^N$.

В каждом цикле необходимо к сумме добавлять очередной элемент $r = 2^N$. Количество циклов зависит от числа слагаемых. Начальное значение суммы равно нулю. Для ввода данных и вывода результата используем потоковые операции (cin и cout соответственно) из библиотеки <iostream>, комментариями оставлен форматированный ввод-вывод.

*Текст программы* (для спроектированной СА на рис. 11):

```
#include «stdafx.h»
#include<locale>
#include<iostream>
//  #include<stdio.h>
#include <conio.h>
usingnamespace std; // пространство имен
void main ()
{
setlocale(LC_ALL, «RUS»);
 float s = 0, r;
 int i, N;
// printf(«\n Введите N»);
// scanf(«%d»,&N);
cout << «\n Введите N»;
  cin >> N;  //  Вывод N
 for ( i=1; i <= N; i++ )
  {
    r =  i*i;
    s+ = r; // s = s + r
  }
cout <<«\n Сумма = » << s; //  printf («Сумма = %6.2f\n»,s);
_getch();  //  getch();
}
```

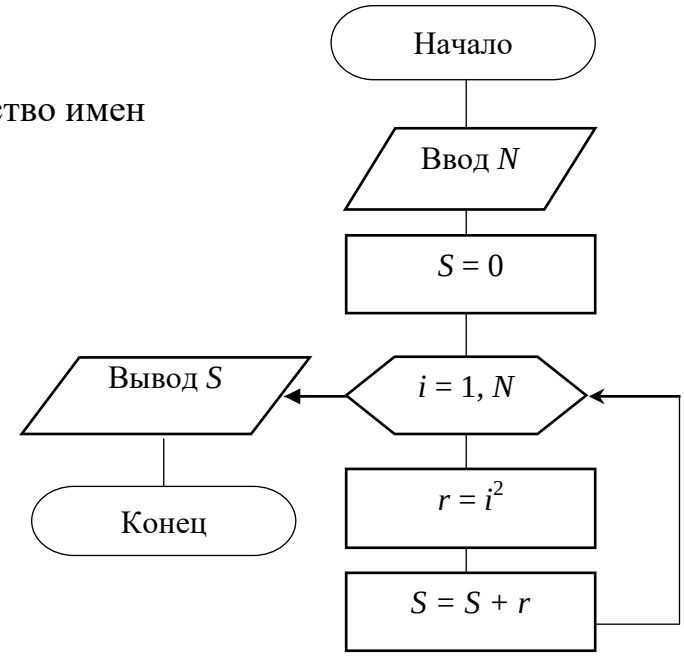



*Рис. 11*

### Задачи по теме

Разработать алгоритмы и написать программы для задач.

1. Вычислить $P = \prod_{i=1}^{20}(e^x + \frac{x^i}{2i})$.

2. Вычислить $S = \sum_{i=1}^{10}(e^x)/i!$

3. Вычислить *n* значений функции *Y* = *Cx* + *D*; (*n*, *C*, *D* ввести) для *n* точек *x*, равномерно распределенных на интервале [*A*, *B*]; *A*=2, *B*=4.

Для вычисления шага (число вещественное, с плавающей точкой) можно использовать формулу $\Delta x = \frac{|B-A|}{n-1}$.

## 3.2. Циклические алгоритмы, управляемые условием

Оператор *цикла с предусловием* (*с предварительной проверкой условия*) имеет структуру:

**while (<выражение>)**
**{**
<оператор;>
**}**

где **<выражение>** – условие выполнения цикла, **<оператор>** – это произвольный оператор C/C++, в том числе и составной.

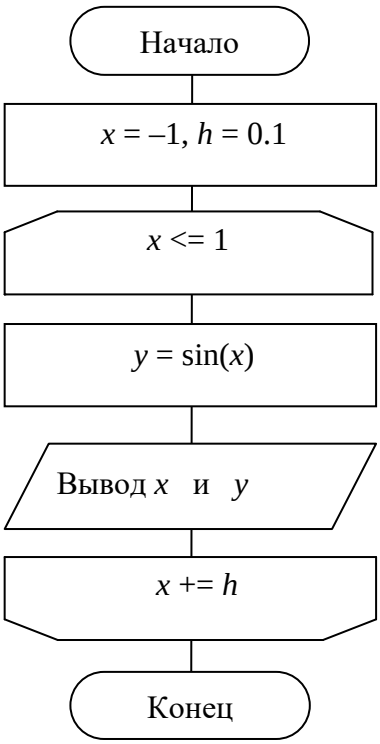


*Рис. 12*

*Пример 3.* Протабулировать функцию для вычисления синуса от –1 до 1 с шагом 0,1.

Разработанная СА изображена на рис. 12.

*Текст программы:*

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void  main()
{
float  y, x = -1, h = 0.1; // Объявление  переменных
while (x < 1.01)
  {
        y = sin(x);
        printf(«for x = %5.2f\t», x); // Вывод  x
        printf(«y = %9.4f \n», y); // Вывод ответа  y
```

```
        x += h;
  }
getch(); // Задержка данных на экране
}
```

Структура *оператора цикла с постусловием*:

```
do
{
<оператор>
}
while (<условие выполнения цикла>);
```

*Пример 4.* Присвоить переменной случайное значение, меньшее 10, и вычислить произведение сгенерированных чисел. Цикл завершить при генерации *x*, равного единице.

Разработанная СА изображена на рис. 13.

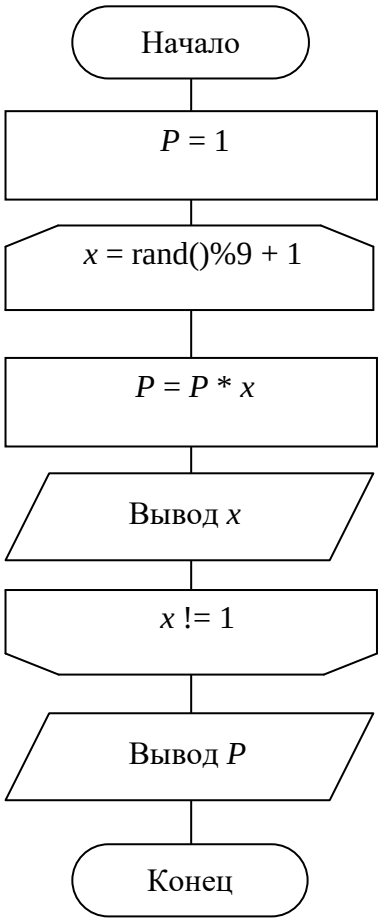



*Рис. 13*

*Текст программы:*

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
void main()
{
int  x;
long P = 1; // P – длинное целое
do
  {
    x = rand() % 9 + 1; / *Присваивается от 1 до  9
    включительно */
    P *= (long) x; // P = P * (long) x;
    printf(«x = %d», x);
   }
while (x != 1); // Цикл завершится, когда x = 1
printf(«P = %ld», P);
getch();
}
```

/ (long)x – операция преобразования x к длинному целому типу

## Задачи по теме

Реализовать через цикл с параметром, предусловием и постусловием.

1. Вычислить $y = \sin^3 x + \cos^3 x$ при $0 \le x \le 2\pi$, где $\Delta x = \frac{\pi}{16}$.

2. Вычислить $y = \begin{cases} a \cdot x \cdot e^{\sqrt{x}} & \text{при } 0{,}2 \le x \le 2, \\ -x \cdot a + 4 & \text{при } 2 < x \le 3 \end{cases}$

с шагом $\Delta x = 0{,}2$ для а) $a$ = 2,5; б) $a$ = 2,5; 2,7; 2,9; 3,1.

3. Вычислить $y = \begin{cases} \lg x & \text{при } 0 \le x \le \frac{\pi}{4}, \\ \sin x & \text{при } \frac{\pi}{4} < x \le \frac{\pi}{2}, \\ \cos x + e^x & \text{при } \frac{\pi}{2} < x < \frac{3\pi}{2} \end{cases}$

с шагом $\Delta x = \frac{\pi}{16}$.

## Задания для самостоятельного выполнения

1. Определить сумму чисел непустой последовательности, за которой следует нуль.

2. Определить площади треугольников с проверкой условий их существования и счетом несуществующих треугольников по введенным длинам сторон *A, B, C*:

а) для *n* троек сторон *A, B, C*;

б) пока не будет получено количество существующих треугольников *k*, при этом подсчитать количество введенных троек.

3. Разработать СА для решения *N* квадратных уравнений (в цикле вводить коэффициенты *A, B, C*), при этом подсчитать количество уравнений с равными и количество уравнений с разными корнями.

4. Разработать СА и программу подсчета точек, попадающих в круг радиуса *R*, из *n* введенных.

5. Начертить СА и написать программу подсчета точек из *k* введенных, попадающих в каждую из четвертей.

## 4. ИТЕРАЦИОННЫЕ АЛГОРИТМЫ

Итерационные алгоритмы содержат итерационные циклы, связанные со значением монотонно изменяющейся величины.

*Пример*. Разработать СА и программу для вычисления суммы элементов вида $S = 1 + 1/2 + 1/4 + 1/8 + ...$ с точностью до $e = 0{,}0001$.

Особенностью алгоритма является добавление значения $1/(2 \cdot i)$, пока добавляемое значение не меньше заданной точности $e = 0{,}0001$. Спроектированная СА изображена на рис. 14.

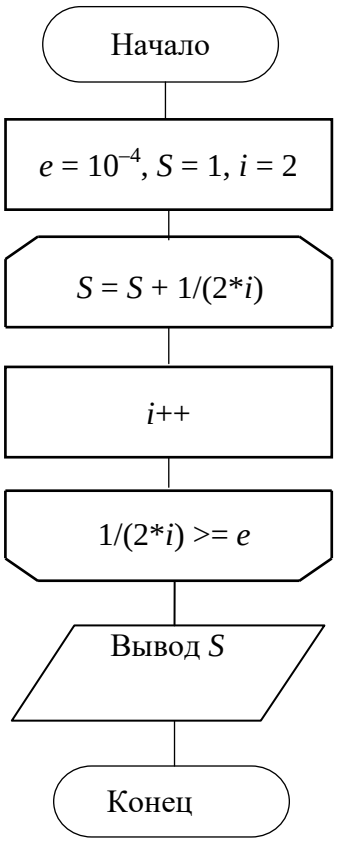



*Рис. 14*

*Текст программы:*

```
#include <stdio.h>
#include <conio.h>
void main ()
{
float  e = 0.0001, S=1;
int i = 2; // счетчик со второго элемента
do
 {
 S += 1.0/(2*i); // S = S + 1.0/(2*i);
 i++ ;  //  i = i +1;
 }
while (1.0/(2*i)  > e); // сравнение с е сомножителей
printf(«Сумма  S=%9.5f\n», S);  // вывод суммы
getch();
}
```

### Задачи по теме

Разработать алгоритмы и написать программы для задач.

1. Вычислить сумму вида $S = \sum_{i=1}^{\infty}(1 - x^{-i}) \cdot x^{-i}$. Вычисление закончить, если $n$-й добавленный элемент ряда окажется меньше заданной точности: $(1 - x^{-i}) \cdot x^{-i} < \varepsilon$.

2. Для произвольного значения $a$ вычислить приближенное значение корня $x = \sqrt{a}$ по итерационной формуле $x_i = \frac{1}{2}(x_{i-1} + \frac{a}{x_{i-1}})$. В качестве начального приближения взять $a$. Критерий окончания вычислений $|x_i - x_{i-1}| \le 10^{-5}$.

### Задания для самостоятельного выполнения

1. Разработать СА перевода целого десятичного числа в двоичную систему счисления (СС).

2. Разработать СА перевода дробной части десятичного числа в двоичную СС.

3. Разработать СА для вычисления $S = \sum_{i=1}^{\infty} \frac{x^2}{i!}$.

## 5. АЛГОРИТМЫ СО СЛОЖНЫМИ ЦИКЛАМИ

Сложные циклы содержат вложенные циклы (или цикл в цикле).

*Пример 1*. Вывести номер итерации внутреннего и внешнего цикла.

При проектировании СА и программировании используются 2 параметра целого типа (по числу циклов). СА для данной задачи изображена на рис. 15.

*Текст программы:*

```
#include <stdio.h>
#include <locale>
void  main()
{
setlocale(LC_ALL, «RUS»);
for (int i = 0; i < 10; i++) // Внешний  цикл
  {
  printf(«Итерация %d внешнего
  цикла \n», i);
  for  (int  j  =  0;  j  <  5;  j++)  //  Внутренний
  цикл
  printf(«Итерация %d внутреннего
  цикла \n», j);
  }
system(«pause»);
//Задержка диалогового окна
}
```

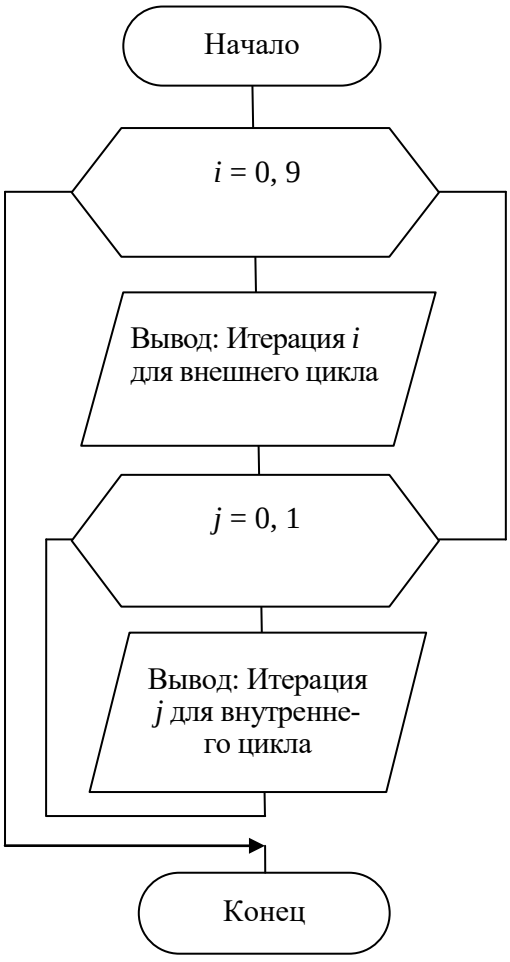



*Рис. 15*

*Пример 2*. Для трех значений $a$ = 0,1, 0,2, 0,3 протабулировать функции при изменении аргумента $x$ на интервале [0, 2π] с шагом, равным $a$, причем $y = a \cdot \sin(x/4)$ на интервале [0, π], $y = a \cdot \cos(x/4)$ на интервале [π, 2π].

При реализации данной задачи применены внутренний и внешний циклы с предусловием. СА изображена на рис. 16.

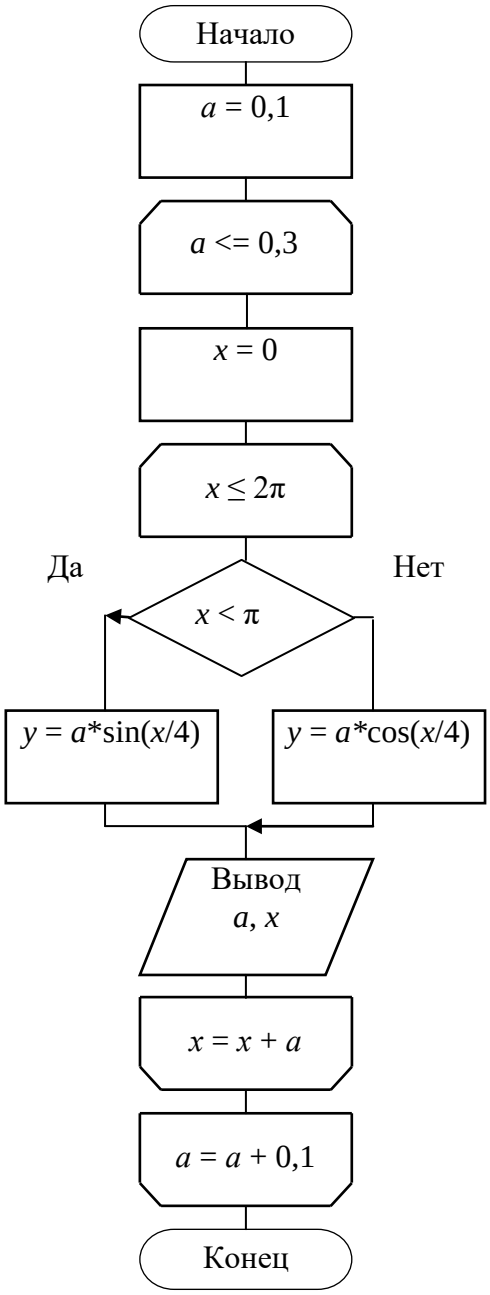



*Рис. 16*

Текст программы:

```
#include<conio.h>
#include <stdio.h>
#include <math.h>
# define π 3.1415
void main ()
{
float a, x, y;
printf(«-------------------------------\n»);
printf(«    a|        x|        y |\n»);
printf(«-------------------------------\n»);
a = 0.1;
while (a<=0.31)
{  printf(«%5.2f\n»,a);
   x = 0;
   while (x <= 2*π+0.01)
      {
          if (x <π)  y = a*sin(x/4);
          else y=a*cos(x/4);
          printf («    %12.2f| %10.2f\n»,x,y);
          x = x + a; // x += a;
      }
   printf(«-------------------------------\n»);
   a = a + 0.1;      // a+=0.1;
}
getch();
}
```

## Задачи по теме

Разработать СА и программы для решения задач.

1. Вычислить $z = \dfrac{(x^2 + yx + y^2)^{xy}}{\sin^2(x) + 0{,}5}$ при $0 \le x \le 2$, $\Delta x = 0{,}5$; при $3 \le y \le 4$, $\Delta y = 0{,}2$.

2. Вычислить $u = e^{x/2} + e^{z/y}$ при $0{,}1 \le x \le 1{,}1$, $\Delta x = 0{,}25$;
при $0{,}5 \le y \le 1$, $\Delta y = 0{,}5$; при $0 < z < 1{,}5$, $\Delta z = 0{,}1$.

### Задания для самостоятельного выполнения

1. Вычислить $S = \sum_{i=1}^{\infty} \frac{(x \cdot y + 4)}{i!}$.

2. Вычислить каждый элемент и построить таблицу Пифагора.

## 6. ОБРАБОТКА ЭЛЕМЕНТОВ МАССИВОВ

*Массив* – набор элементов, способных хранить данные одинакового типа. Прежде чем начать работу с массивом, необходимо объявить его, т. е. указать тип хранимых данных, имя массива и его размер (каждый индекс записывается в квадратных скобках). Размером массива называется количество элементов в нем.

### 6.1. Одномерные массивы

*Пример.* Инициализировать элементы массива и вывести на экран сам массив и сумму его элементов.

СА изображена на рис. 17.

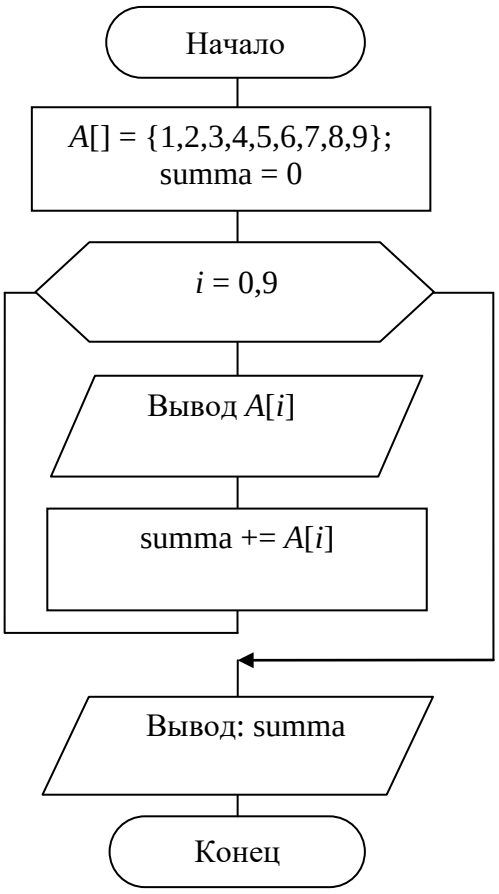



*Рис. 17*

*Текст программы*:

```
#include <conio.h>
#include <stdio.h>
void main()
{
int A[] = { 1,2,3,4,5,6,7,8,9,10 }; /* Объявление и инициализация  массива */
int summa = 0;
for (int i = 0; i < 10; i++)
  {
printf(«%d», A[i]);  //вывод массива на экран
summa += A[i];
  }
printf(«%d», summa); //вывод массива на экран
system(«pause»); // Задержка экрана
}
```

### Задачи по теме

Разработать СА для решения задач.

1. Вычислить $s = \sum_{i=1}^{n} a_i + \sum_{j=1}^{m} b_j$.

2. Вычислить $z = \sum_{i=1}^{n} a_i \cdot b_i$.

3. Разработать СА и программу для нахождения среднего геометрического значения для положительных элементов и среднего арифметического – для отрицательных элементов в массиве из 20 чисел.

## 6.2. Двумерные и многомерные массивы

Для работы с двумерными массивами необходимо использовать вложенные циклы (цикл в цикле) (см. рис. 15).

### Задачи по теме

Разработать СА для решения задач.

1. Вычислить суммы положительных и отрицательных элементов в массиве $Z$ (10·8·4) и подсчитать их количество. Найти номер минимального элемента в массиве.

2. Вычислить суммы положительных элементов с четными номерами строк матрицы $Y$(10·20).

### Задания для самостоятельного выполнения

Разработать СА и программы.

1. Найти элемент, наиболее близкий к среднему арифметическому значению, в массиве из 10 элементов.

2. Ввести массив $A$(10), подсчитать сколько в нем четных, сколько нечетных элементов.

3. Найти отдельно количество чисел с четными номерами и четными значениями и нечетными номерами и нечетными значениями в массиве $Y$(10·20).

4. Найти максимальный элемент (его номер и значение) в массиве $A$ из 100 элементов.

5. Упорядочить элементы одномерного массива $A$(100) по возрастанию.

*Для упорядочения массива по возрастанию* (*по неубыванию*) можно использовать один из следующих алгоритмов:

а) сортировка выбором.

Отыскивается максимальный элемент и переносится в конец массива. Затем этот метод применяется ко всем элементам, кроме последнего (он уже находиться на своем месте), и т. д.

б) сортировка обменом (метод пузырька).

Последовательно сравниваются пары соседних элементов $x_k$ и $x_{k+1}$ (где $k$ = 1, 2,…, n–1), и если $x_k > x_{k+1}$, то элементы переставляются, тем самым наибольший элемент окажется на своем месте в конце массива. Затем этот метод повторяется для всех элементов массива, кроме последнего, и т. д.

## 7. ПОДПРОГРАММЫ (ФУНКЦИИ)

В подпрограммы (отдельные функции) рекомендуется выделять фрагменты, которые используются на разных участках программы несколько раз, возможно для различных данных.

*Пример.* Разработать схемы алгоритмов с подпрограммами для вычисления $Z = k!+(1+2k)!–5!$

Поскольку в выражении трижды вычисляется значение факториала для разных значений параметров, выделим его определение в функцию с параметром. При каждом вызове такой функции из главной программы в качестве фактического аргумента необходимо передавать нужный параметр. Спроектируем СА для главной функции, в которой предусмотрим ввод параметра $k$ и вычисление выражения со встроенными вызовами подпрограммы (рис. 18), а также для функции (подпрограммы) вычисления факториала (рис. 19).

### Задачи по теме

Разработать СА и программы для решения задач.

1. Вычислить $W = \dfrac{(\max(X,Y,Z) - \min(X,Y,Z))}{(\max(A,D,E) + \min(A,D,E))}$.

2. Вычислить $S = \dfrac{\max(A_i) - \min(A_i)}{\max(B_j) + \min(B_j)}$.

3. Вычислить $Z = \dfrac{(\sum_{i=1}^{k} A_i)}{k!} + \dfrac{\sum_{j=1}^{n} B_j}{n!}$, где $A_i$, $B_j$ – элементы массивов.

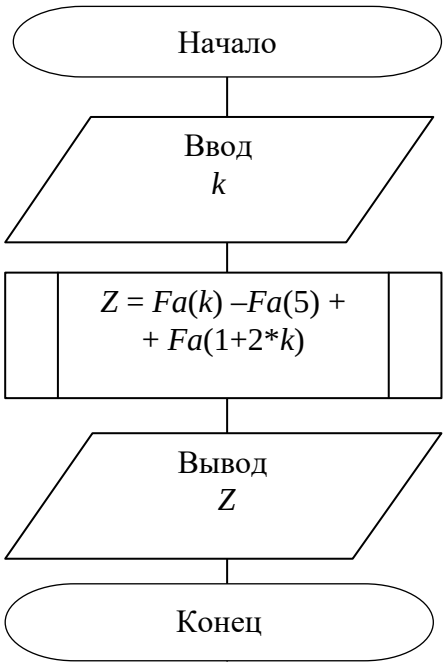



*Рис. 18*

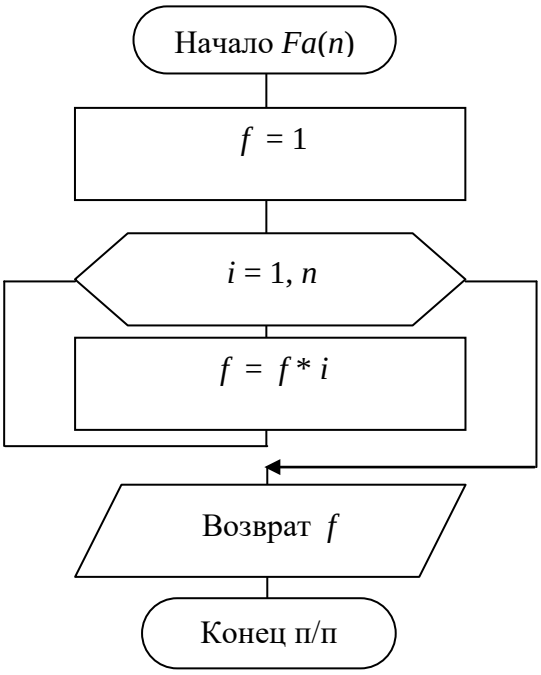



*Рис. 19*

### Задания для самостоятельного выполнения

1. Оптимизировать алгоритм для задачи 2, объединяя вычисление суммы и факториала. Отладить программу.

2. Разработать алгоритм и программу для задачи 3, при этом объединить нахождение «max» и «min» в одной подпрограмме.

3. Разработать схемы алгоритмов перевода смешанных чисел из десятичной СС в другую (двоичную, восьмеричную) (использовать подпрограмму для перевода каждой части).

4. Разработать и отладить программы для моделирования сложения чисел в двоичной СС.

## 8. ВОСХОДЯЩИЙ МЕТОД ПРОЕКТИРОВАНИЯ И ТЕСТИРОВАНИЯ АЛГОРИТМОВ И ПРОГРАММ

Восходящее программирование предполагает проектирование программы «снизу вверх» [5–7]. Сначала отдельно проектируются и отлаживаются программы для каждой простой подзадачи, а затем разработанные модули последовательно (поочередным подключением) объединяются в единую программу.

*Пример*. Разработать универсальную СА перевода из любой СС в любую (например, в шестнадцатеричную, восьмеричную, двоичную, десятичную).

Для проектирования алгоритма выделим подзадачи:

а) перевод целого десятичного числа в СС с основанием $q$ (делением на ее основание $q$);

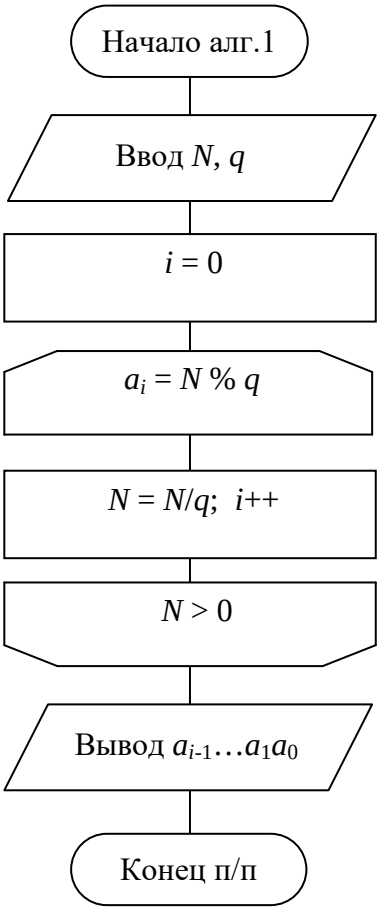



*Рис. 20*

б) перевод десятичной дроби в СС с основанием $q$ (умножением на основание СС);

в) перевод смешанного числа в десятичную СС (по весовым коэффициентам);

г) перевод группировкой разрядов для СС с основанием, кратным степени 2.

Разработаем алгоритмы для каждой подзадачи, а затем объединим в единый универсальный [5].

*Алгоритм 1.* Для перевода целого десятичного числа в СС с основанием $q$ необходимо делить исходное число $N$, а затем полученные частные на основание новой СС ($q$) до тех пор, пока целая часть не станет равна нулю; полученные остатки $a_i$ ($i$ = 0, 1, 2, …, $n$), записанные начиная с последнего, $a_i$ – это цифры числа в новой СС: $a_n a_{n-1} ... a_2 a_1 a_0$.

Спроектируем СА для перевода целого десятичного числа в СС с основанием $q$ с учетом того, что разряды новой СС можно вычислить за $n$ + 1 цикл. В цикле необходимо использовать операции деления и вычисления остатка от деления (%) (рис. 20). Перед выводом цифр числа необходимо сделать проверку: если СС – шестнадцатеричная, то цифры от 0 до 9 выводятся в естественном виде, если получено 10, то печатается буква A, 11 – B, 12 – C, 13 – D, 14 – E, 15 – F.

*Алгоритм 2.* Дробную часть исходного числа (затем дробные части произведений) необходимо последовательно умножать на основание новой СС $q$ до тех пор, пока дробная часть не станет равна нулю или не будет получено заданное количество цифр ($t$ – точность). Целые части полученных произведений, начиная с первой, определят цифры дробной части в СС с основанием $q$, все действия выполняются по правилам исходной СС.

Реализуем указанное правило для перевода дробной части числа $D$ из десятичной СС в СС с основанием $q$ в виде СА (рис. 21).

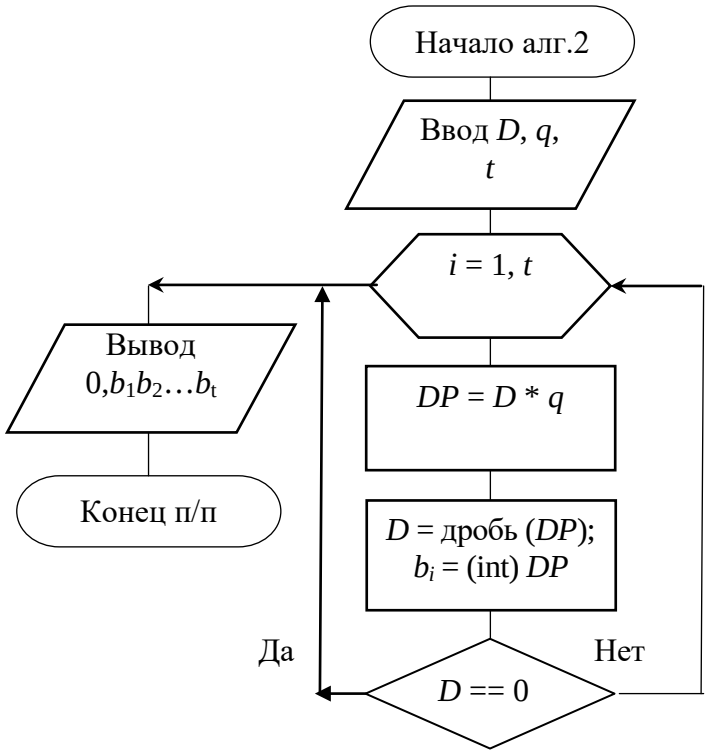



*Рис. 21*

На рисунке дополнительно введены следующие обозначения: *DP* – произведение дробной части и *q, i =1, 2, …, t* – номер вычисляемого бита, $b_i$ – цифры числа в новой СС, операция (int) *DP* выделяет целую часть из *DP*.

*Алгоритм 3.* Смешанное число *A* в СС с основанием *q* (кроме десятичной) представим в поразрядном виде

$$a_n a_{n-1} ... a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3} ... a_{-t},$$

где $a_i$ ($i = -t, \dots, 0, 1, 2, \dots, n$) – цифры числа, *n* – номер старшего разряда целой части числа, *t* – номер младшего бита дробной части числа.

Для перевода смешанного числа в десятичную СС необходимо просуммировать произведения каждого разряда с его весом:

$$x_{10} = \sum_{i=-t}^{n} a_i \cdot q^i .$$

Для нахождения такой суммы при разработке алгоритма применен цикл со счетчиком по переменной номера разряда $i = -t, \dots, n$, изменяющейся с шагом, равным единице, где *t* – точность или число разрядов в дробной части ($K_{др}$) числа *A*, *n* – номер старшего бита в целой части. Для перевода смешанного числа в десятичную СС необходимо в числе *A* подсчитать количество битов в целой ($K_ц$) и отдельно дробной ($K_{др}$) частях. Поскольку вес каждого следующего бита в *q* раз больше, чем у предшествующего, то в каждом цикле возведение в степень выполнять неэффективно. Достаточно ввести переменную для веса *v*, значение которой определяется до цикла $v = q^{-t}$. Затем в каждом цикле вес увеличивается в *q* раз, $v = v \cdot q$. Сформированная сумма произведения каждого разряда и его веса за ($K_ц + t$) итераций цикла в переменной *x*10 определит искомое число в десятичной СС. Спроектированная СС изображена на рис. 22.

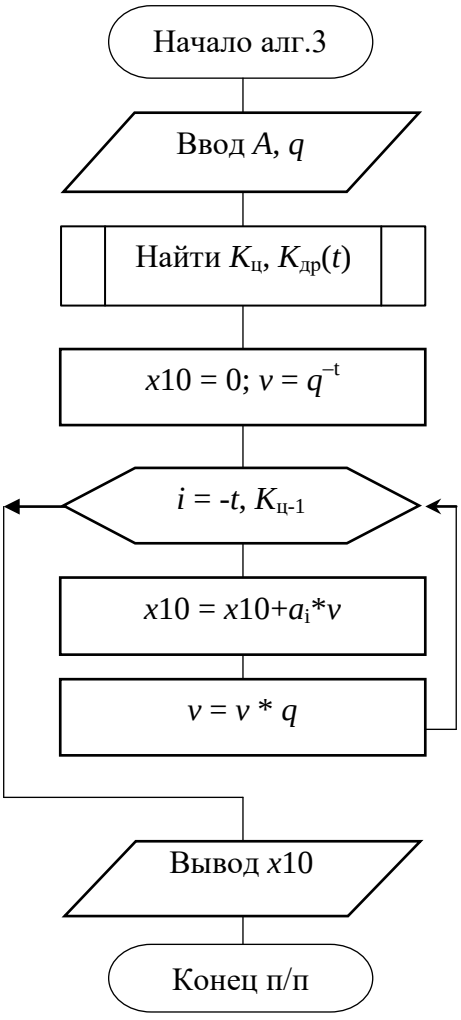



*Рис. 22*

Далее выполним разработку схемы универсального алгоритма перевода чисел методом «снизу вверх» (от подзадач к общему алгоритму [5]).

Приведенные три алгоритма (см. рис. 20–22) позволяют спроектировать универсальный алгоритм перевода числа из любой СС в любую другую, причем если исходная и результирующая СС не десятичные, то десятичная СС используется в качестве промежуточной. Схема универсального алгоритма представ-

лена на рис. 23. Для нее введены дополнительно следующие обозначения: $X$ − исходное число; $X_{ц}$ – целая часть числа $X$; $X_{др}$ – дробная часть числа $X$; $q$ − основание исходной СС; $p$ − основание новой СС, в которую осуществляем перевод.

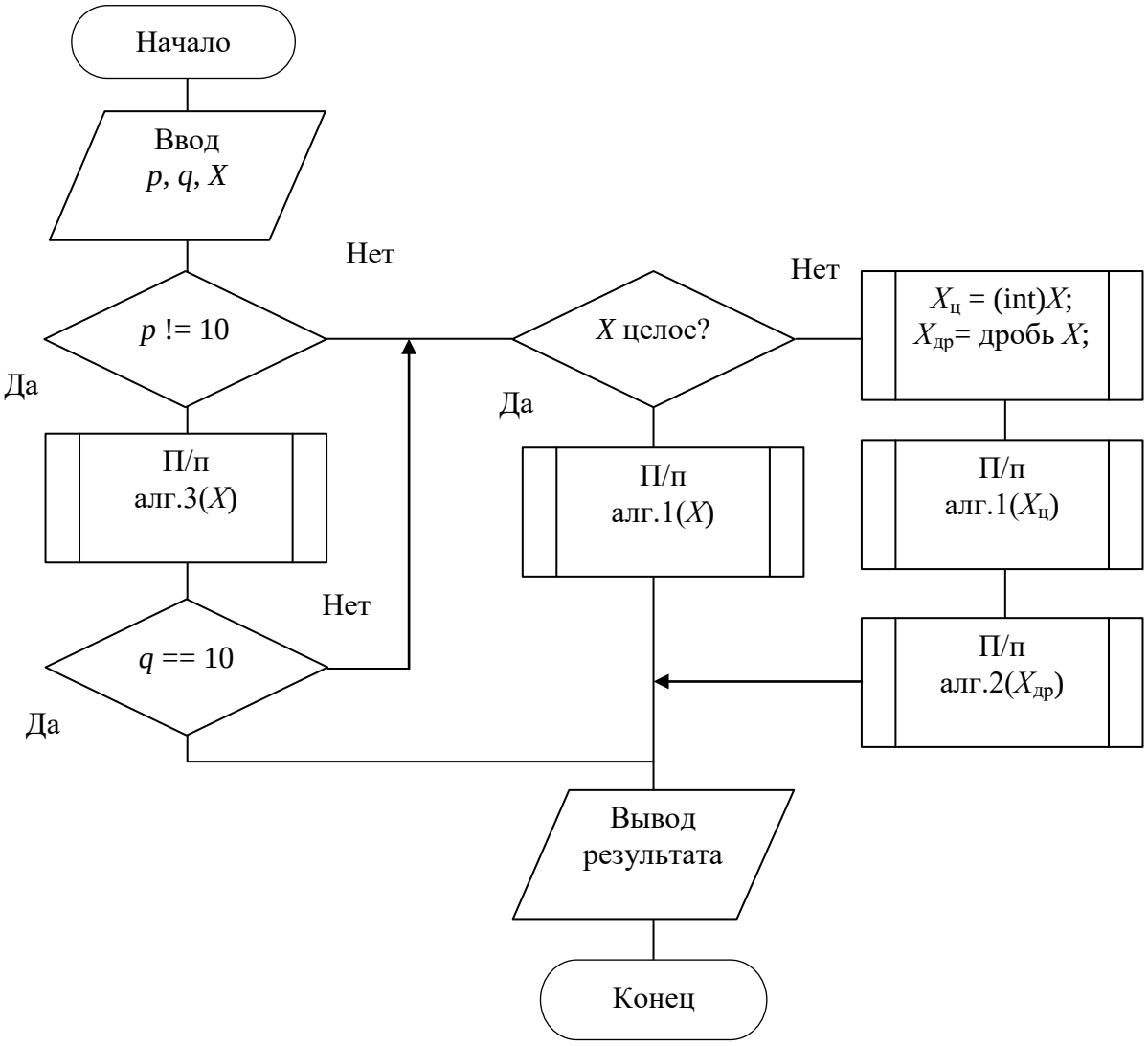



*Рис. 23*

Алгоритмы для отдельных подзадач оформлены подпрограммами, каждая из которых вызывается с передачей нужного параметра (числа $X$, его части $X_{ц}$ или $X_{др}$ либо битов числа) в зависимости от получаемых значений условий.

*Алгоритм 4.* Перевод чисел между СС, кратными степени числа 2, можно выполнить также группировкой двоичных разрядов триадами (вправо и влево от десятичной точки) для перевода в восьмеричную систему и тетрадами для перевода в шестнадцатеричную СС, а затем записать триады или тетрады соответствующими восьмеричными или шестнадцатеричными цифрами (табл. 1).

*Таблица 1*

| Десятичная СС | Двоичная СС | Восьмеричная СС | Шестнадцатеричная СС |
|---|---|---|---|
| 0 | 0000 | 0 | 0 |
| 1 | 0001 | 1 | 1 |
| 2 | 0010 | 2 | 2 |
| 3 | 0011 | 3 | 3 |
| 4 | 0100 | 4 | 4 |
| 5 | 0101 | 5 | 5 |
| 6 | 0110 | 6 | 6 |
| 7 | 0111 | 7 | 7 |
| 8 | 1000 | 10 | 8 |
| 9 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

В таблице приведены эквивалентные способы записи для каждого из чисел от нуля до пятнадцати для десятичной, двоичной, восьмеричной и шестнадцатеричной СС. Например, для записи шестнадцатеричной цифры в двоичном виде понадобится одна тетрада (четыре двоичных разряда), для записи восьмеричного числа (>7) – две триады битов.

### Задания для самостоятельного выполнения

Отладить программы для перевода смешанных чисел из десятичной СС в двоичную, восьмеричную, шестнадцатеричную (использовать подпрограммы для перевода каждой части), а также для перевода в десятичную СС чисел из двоичной и восьмеричной.

## 9. НИСХОДЯЩИЙ МЕТОД РАЗРАБОТКИ ПРОГРАММ

Нисходящий метод предполагает проектирование программы «сверху вниз»: сначала проектируется укрупненный алгоритм, затем на каждом шаге детализации для каждой задачи разрабатывается более подробный алгоритм в терминах выделенных в ней подзадач [5, 8].

*Пример*. Разработать модель функционирования арифметического устройства компьютера, выполняющего сложение, вычитание, умножение, деление над числами с плавающей точкой. Числа вводить и выводить в прямом коде. Для выполнения операций использовать любой код (прямой, дополнительный, обратный).

Числа с плавающей точкой представляются в виде нормализованной мантиссы и порядка. Для исходных чисел *A* и *B* обозначим мантиссы и порядки: *mA, pA* и *mB, pB,* для результата *C* – это *mC, pC*. На начальном этапе проектируем укрупненную СА в терминах крупных блоков (рис. 24): «сложение», «вычитание», «умножение», «деление». Параметром *Op* задается символ требуемой операции.

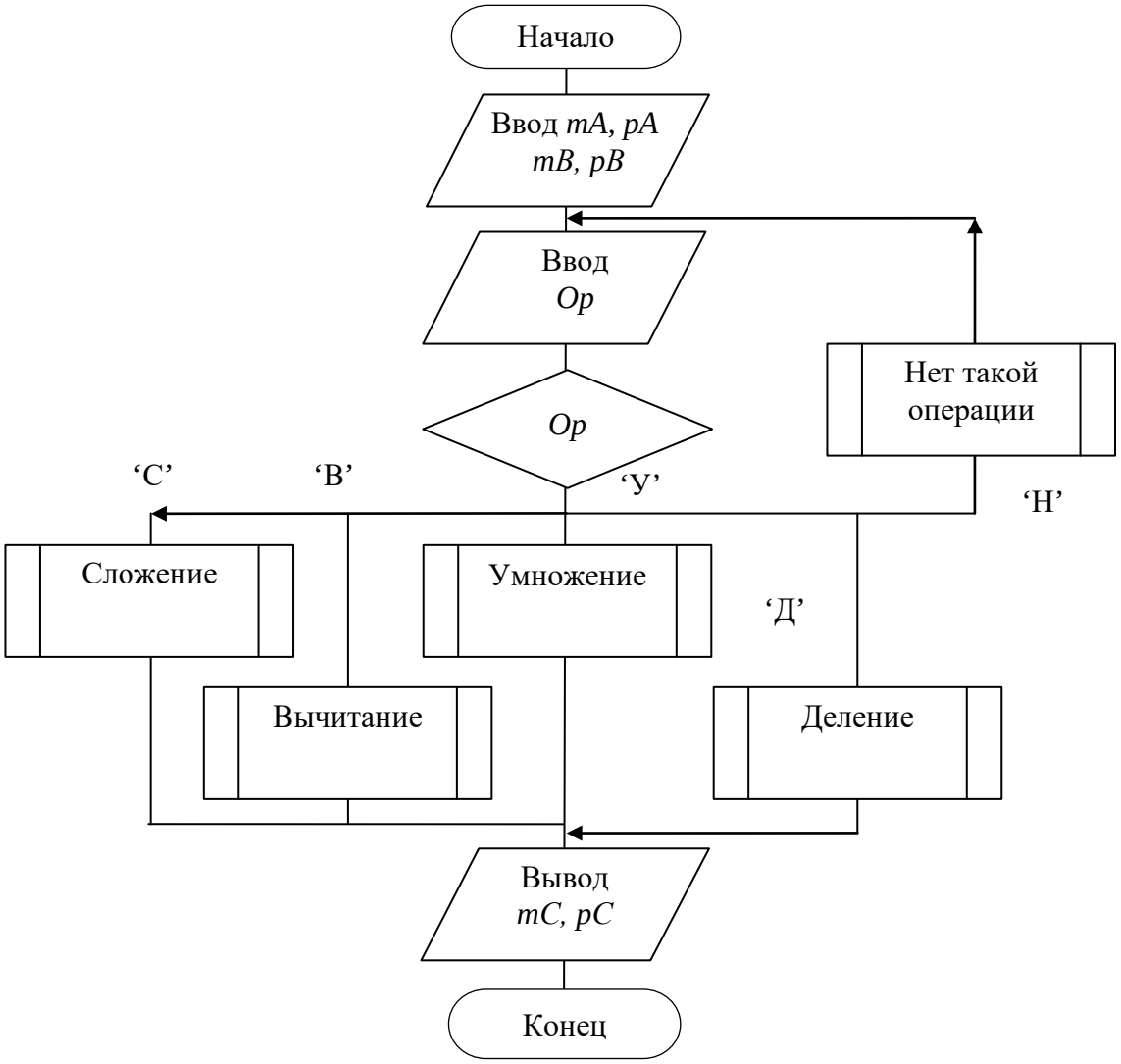



*Рис. 24*

Далее выполним детализацию каждого блока для операций в несколько этапов. При необходимости будем использовать для мантисс дополнительный

модифицированный (с двумя знаковыми разрядами) код со следующей нумерацией разрядов: *Зн1 Зн2 1 2 3 … k*, где *k* – номер младшего бита, вес которого равен $2^k$.

*Сложение.* При сложении необходимо выровнять порядки, затем сложить мантиссы, при необходимости мантиссу нормализовать. Если хотя бы одна из мантисс отрицательная, выполняется перевод в дополнительный модифицированный код. СА на первом этапе детализации изображена на рис. 25.

Далее выполняется второй этап детализации (рис. 26), на котором раскрываются процессы выравнивания порядков и нормализации мантисс вправо (при переполнении) и влево.

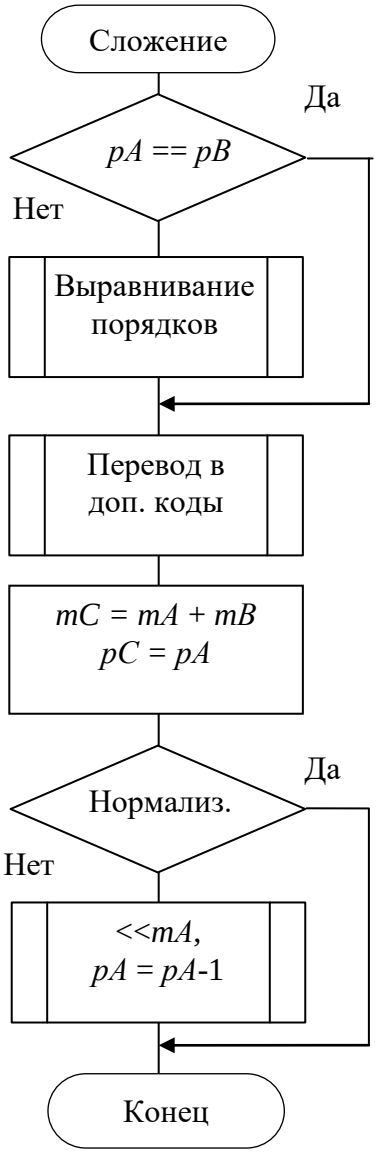



*Рис. 25*

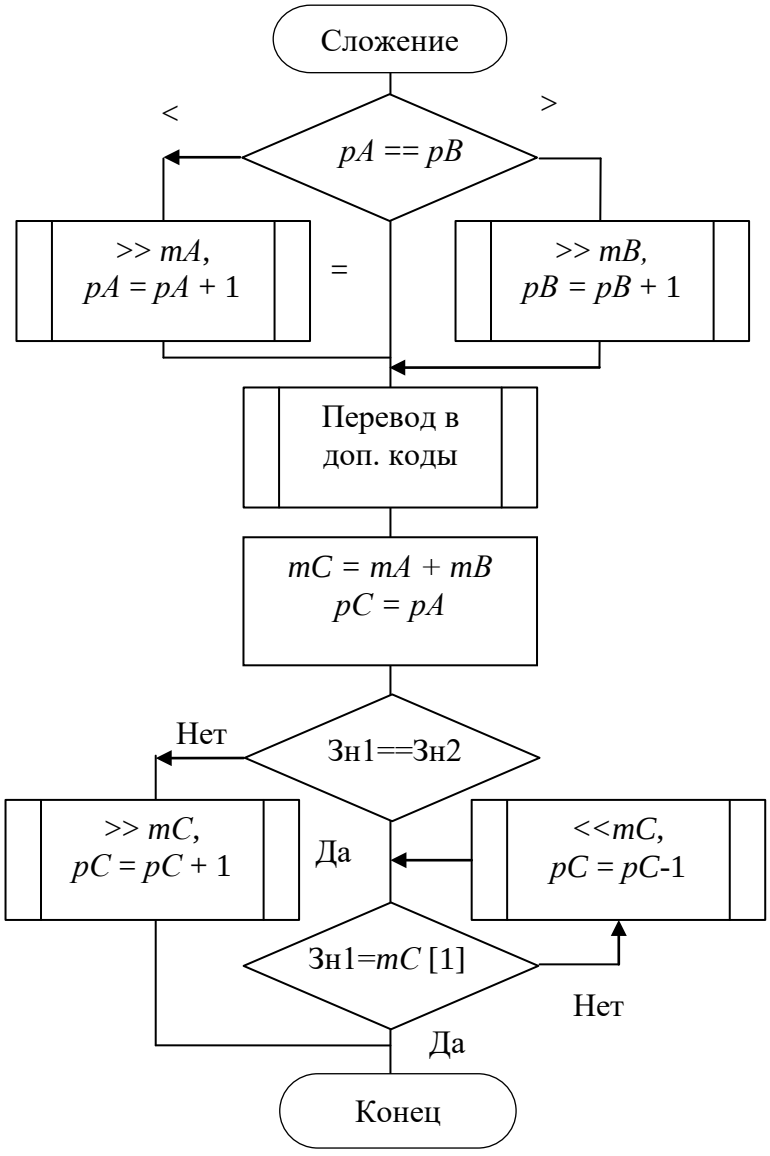



*Рис. 26*

В зависимости от знаков операндов определяется наличие переполнения и выполняются необходимые действия (если знаковые биты Зн1 и Зн2 мантиссы результата *C* разные, производится сдвиг разрядов мантиссы вправо на один

бит, а порядок корректируется увеличением на единицу). При появлении незначащих нулей в мантиссе (денормализация вправо) производится сдвиг влево (на схеме обозначено как <<) мантиссы результата до первой значащей единицы, а порядок уменьшается на число незначащих нулей.

*Вычитание* сводится к сложению с изменением знака вычитаемого.

*Умножение.* При умножении знак результата определяется операцией «исключающее ИЛИ», порядки складываются, мантиссы перемножаются по специальному алгоритму. При необходимости выполняется нормализация результата вправо, но не более чем на один бит. Переполнения произойти не может, если исходные мантиссы нормализованные. Схема реализованного алгоритма для умножения чисел с плавающей точкой приведена на рис. 27.

*Деление.* При делении знак результата определяется операцией «исключающее ИЛИ», порядки вычитаются, мантиссы делятся по специальному алгоритму. При необходимости результат нормализуется. Чтобы не произошло переполнения, предварительно сравниваются мантиссы исходных чисел: если $mA > mB$, денормализуется $mA$ (сдвигом на один бит вправо, порядок увеличивается на 1).

Схема реализованного для деления алгоритма приведена на рис. 28.

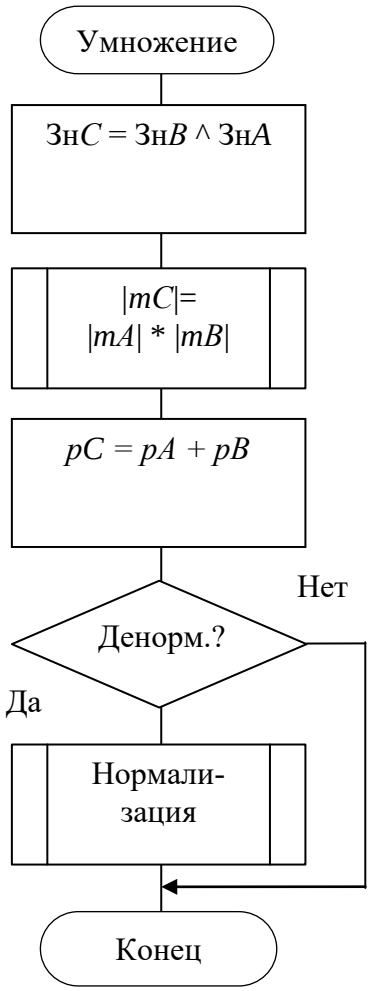



*Рис. 27*

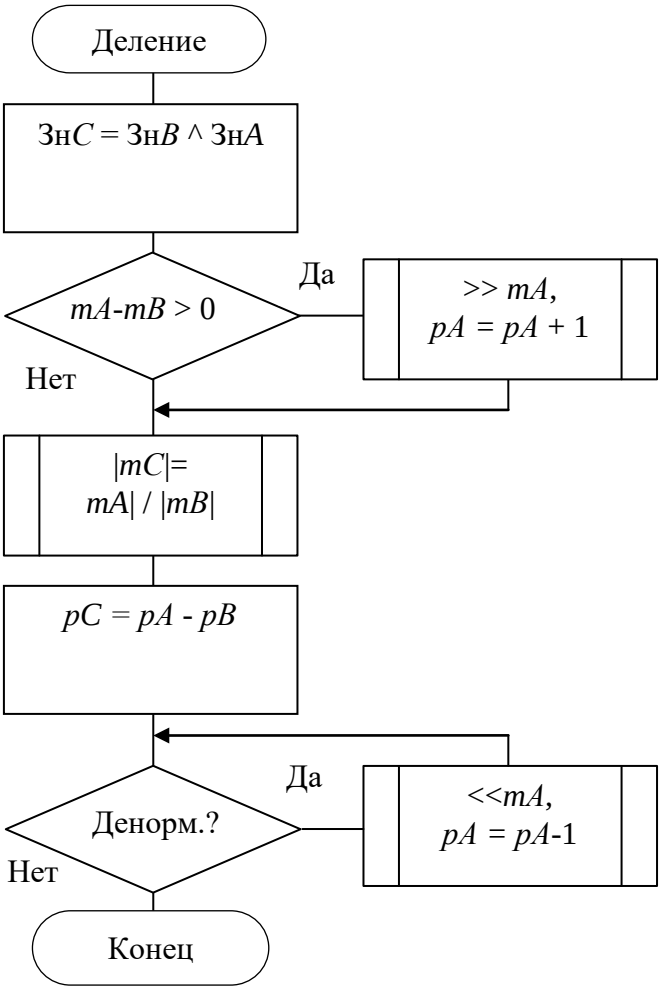



*Рис. 28*

Далее могут следовать другие этапы детализации алгоритмов (у каждого программиста их количество разное). Детализация производится до тех пор, пока не появится возможность запрограммировать алгоритм.

## Задания для самостоятельного выполнения

1. Для трех одномерных массивов выполнить сдвиг элементов вправо (влево) на 1 разряд (с подпрограммой).

2. Пары одномерных массивов, состоящих из нулей и единиц, сложить по правилам двоичной арифметики (с подпрограммой).

3. Разработать программу-модель умножения чисел с фиксированной точкой по правилам двоичной арифметики (с младших разрядов).

4. Разработать программу-модель сложения чисел с плавающей точкой по правилам двоичной арифметики.

5. Выполнить перевод в дополнительный код (инвертировать биты и добавить единицу в младший разряд) числа, представленного массивом.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. ГОСТ 19.701–90. ЕСПД. Схемы алгоритмов, программ, данных и систем. – Введ. 1992–01–01. – М. : Изд-во стандартов, 1991. – 26 с.

2. Иванова, Г. С. Технология программирования [Электронный ресурс] : учеб. для вузов по направлению «Информатика и вычислительная техника» / Г. С. Иванова. – М. : КНОРУС, 2012. – 1 электрон. опт. диск (CD-ROM).

3. Крылов, Е. В. Техника разработки программ. В 2 кн. Кн. 2. Технология, надежность и качество программного обеспечения : учебник / Е. В. Крылов, В. А. Острейковский, Н. Г. Типикин. – М. : Высшая школа, 2008. – 469 с.

4. Липаев, В. В. Тестирование компонентов и комплексов программ : учебник / В. В. Липаев. – М. : Синтег, 2010. – 392 с.

5. Восходящее и нисходящее программирование [Электронный ресурс] : метод. указания / Минобрнауки России, ОмГТУ ; [сост. О. П. Шафеева]. – Электрон. текст. дан. (624 Кб). – Омск : Изд-во ОмГТУ, 2015. – 1 электрон. опт. диск.

6. Потапов, В. И. Компьютерная арифметика и алгоритмическое моделирование арифметических операций [Электронный ресурс] : учеб. пособие / В. И. Потапов, О. П. Шафеева ; ОмГТУ. – Электрон. текст. дан. (936 Кб). – Омск : Изд-во ОмГТУ, 2014. – 1 электрон. опт. диск.

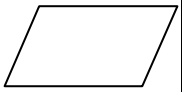
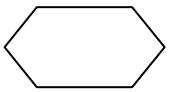
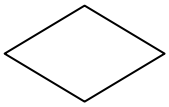
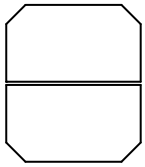
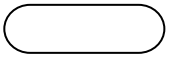
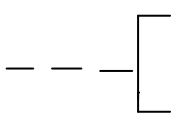
7. Павловская, Т. А. C/C++. Программирование на языке высокого уровня / Т. А. Павловская. – СПб. : Питер, 2012. – 544 с.

8. Программирование на языке Си : метод. указания / ОмГТУ ; [сост. О. П. Шафеева, Ю. Г. Каворина, Г. С. Шукурова]. – Омск : Изд-во ОмГТУ, 2008. – 72 с.

9. Системы программирования : метод. указания / ОмГТУ ; [сост. О. П. Шафеева, И. А. Волчкова, С. А. Гончаров]. – Омск : Изд-во ОмГТУ, 2012. – 32 с.

10. Шафеева, О. П. Технологии программирования. C++ : учеб. пособие / О. П. Шафеева ; ОмГТУ. – Омск : Изд-во ОмГТУ, 2007. – 80 с.

## ПРИЛОЖЕНИЕ А
## ОБОЗНАЧЕНИЯ ГРАФИЧЕСКИЕ В СХЕМАХ АЛГОРИТМОВ

| Символ | Наименование | Назначение |
|---|---|---|
|  | Данные | Определяет ввод или вывод на внешнее устройство или любой носитель данных |
|  | Процесс | Отражает обработку данных: выполнение отдельной операции или группы операций |
|  | Предопределенный процесс | Отображает предопределённый процесс, состоящий из одной или нескольких операций программы, которые определены в другом месте (подпрограмме, модуле) |
|  | Подготовка | Отражает инициализацию и модификацию параметра для управления циклом со счётчиком |
|  | Решение | Описывает проверку условия и выполняет переключение по одному из условий. Имеет один вход и два или более альтернативных выходов, один из которых активизируется после вычисления условия внутри символа |
|  | Границы цикла | Состоит из двух частей: начала и конца цикла. Обе части имеют один и тот же идентификатор. Изменение значения идентификатора, условия для выполнения или завершения помещаются внутри символов в начале или в конце цикла |
|  | Соединитель | Используется для обрыва линии и продолжения её в другом месте. Должен содержать уникальное обозначение |
|  | Терминатор | Определяет начало и конец структурной СА программы или подпрограммы |
|  | Основная линия | Отображает последовательность выполнения действий в алгоритме |
|  | Комментарий | Используется для добавления пояснительных записей. Связывается с символом или группой символов, обведённых пунктиром |

# ПРИЛОЖЕНИЕ Б
# ОПЕРАЦИИ ЯЗЫКА С/С++

| Вес | Знак | Наименование операции | Тип операции | Порядок |
|---|---|---|---|---|
| 1 | ( )<br>[]<br>.<br><br>-> | вызов функции<br>выделение элемента массива<br>выделение элемента структуры или объединения<br>выделение элемента структуры или объединения (объединения), адресуемой(го) указателем | выражение | Слева направо (→) |
| 2 | !<br>~<br>-<br>++<br>--<br>&<br>*<br>(тип)<br>sizeof | логическое отрицание<br>побитовое отрицание<br>изменение знака (унарный минус)<br>увеличение на 1 (инкремент)<br>уменьшение на 1 (декремент)<br>определение адреса<br>обращение по адресу<br>преобразование типа<br>определение размера в байтах | унарные | Справа налево (←) |
| 3 | *<br>/<br>% | умножение<br>деление<br>остаток от деления | бинарные арифметические | → |
| 4 | +<br>- | сложение<br>вычитание | бинарные арифметические | → |
| 5 | <<<br>>> | сдвиг влево<br>сдвиг вправо | сдвиг | → |
| 6 | <<br><=<br>><br>>= | меньше чем<br>меньше или равно<br>больше чем<br>больше или равно | отношения | → |
| 7 | ==<br>!= | равно<br>неравно | отношения | → |
| 8 | & | поразрядная операция «И» | побитовая | → |
| 9 | ^ | поразрядная операция «исключающее ИЛИ» | побитовая | → |
| 10 | \| | поразрядная операция «ИЛИ» | побитовая | → |
| 11 | && | логическая операция «И» | логическая | → |
| 12 | \|\| | логическая операция «ИЛИ» | логическая | → |

| Вес | Знак | Наименование операции | Тип операции | Порядок |
|---|---|---|---|---|
| 13 | ? : | условная операция | тернарная | ← |
| 14 | = | простое присваивание | присваивания | ← |
| | | составные присваивания: | | |
| | += | сложение с присваиванием | бинарные | |
| | -= | вычитание с присваиванием | | |
| | *= | умножение с присваиванием | | |
| | /= | деление с присваиванием | | |
| | %= | остаток от деления с присваиванием | | |
| | <<= | сдвиг двоичного числа влево с присваиванием | (для двоичных операндов) | |
| | >>= | сдвиг двоичного кода вправо с присваиванием | | |
| | &= | побитовая операция «И» с присваиванием | | |
| | \|= | поразрядная операция «ИЛИ» с присваиванием | | |
| | ^= | исключающее «ИЛИ» с присваиванием | | |
| 15 | , | операция «запятая» (соединения) | тернарная | → |

## ПРИЛОЖЕНИЕ В
## ТИПЫ ДАННЫХ

| № | Обозначение | Размер (байт) | Диапазон | Тип данных |
|---|---|---|---|---|
| 1 | char, signed char | 1 | -128...127 | Символьный со знаком |
| 2 | unsigned char | 1 | 0...255 | Символный без знака |
| 3 | short, short int, signed short, signed short int | 2 | -32768... 32767 | Короткое целое со знаком |
| 4 | unsigned short, unsigned short int | 2 | 0...65535 | Короткое целое без знака |
| 5 | int, signed, signed int | 1, 2, 4 | Зависит от реализации | Целое |
| 6 | unsigned, unsigned int | 1, 2, 4 | Зависит от реализации | Целое без знака |
| 7 | long, signed long, long int | 4 | -2147483648 …2147483647 | Длинное целое со знаком |
| 8 | unsigned long | 4 | 0...4294967295 | Длинное целое без знака |
| 9 | float | 4 | -3.4e-38... 3.14e+38 | Вещественное число с плавающей точкой |
| 10 | double | 8 | -1.7e-308… 1.7e308 | Вещественное число удвоенной точности |
| 11 | long double | 10 | -3.4e-4932... 4e4932 | Длинное вещественное число удвоенной точности |

## ПРИЛОЖЕНИЕ Г
## ОПЕРАТОРЫ ЯЗЫКА С/С++

1. Оператор «выражение» имеет три формы:
**<идентификатор> = <выражение>; //** выполняет простое присваивание
**<идентификатор> <знак>= <выражение>;** // составное присваивание
**<идентификатор> = <идентификатор> = <выражение>;**
Он реализует многоступенчатое присваивание справа налево.

*Пример*: z = x; w += 2; /* w = w+2*/ s = t = 1;

2. Условный оператор записывается следующим способом:
**if (<выражение>) <оператор1;>**
**else (<оператор2>);**

*Пример*: а) **if** (k != 0) k = k+1; б) **if** (k) k++;
**else** k = k-1; **else** k--;

3. Оператор цикла со счетчиком:
**for (<выражение1>;<условие выполнения>;<выражение2>)**
**<оператор>;**
где <выражение1> – выражение инициализации параметра цикла;
<выражение2> – изменение параметра цикла.

*Пример*: **for (**i=0; i<100; i++**)** sum += x[i]**;** // sum=sum+x[i];

4. Оператор цикла с предусловием:
**while ( <выражение> ) <оператор>;**

*Пример*: while (i<100) p++;

5. Оператор цикла с постусловием:
**do {<операторы>} while ( <условие выполнения> );**

*Пример*:
```
do
   {
        n *= i;
        i++;
    }
while (i<= 100);
```

6. Оператор безусловного перехода:

**goto** <идентификатор-метка>**;**

7. Оператор возврата из функции:

**return** <выражение>**;**

8. Оператор-переключатель:

```
switch ( <выражение> )
   {
   case<константа1> : <группа операторов1>;
   case<константа2> : <группа операторов2>;
                  ...
   case<константаN> : <группа операторовN>;
   default : <операторы>;
   };
```

Метки определяют точки входа в тело оператора. Чтобы выполнить только одно действие, необходимо предусмотреть выход из данного оператора в нужном месте оператором **break**.

*Пример:*

```
switch (operand)
            {
            case 1: { x *= y; break; }
            case 2: { x /= y; break; }
            case 3: { x += y; break; }
            case 4: { x -= y; break; }
            case 5: { x++; }
            case 6: { x++; break; }
            case 7:
            case 8:
            case 9: { printf(«Not done\n»); break; }
            default: { printf(«Bug!\n»);
              exit(1);  }
            }
```

9. Оператор **break;** организует досрочный выход из операторов while, do, for или switch.

10. Оператор **continue;** выполняет переход на следующую итерацию и выполняет тело цикла  while, do или for с первого  оператора.

*Пример:*

```
for (i = 0; i < 20; i++)
{
    if (array[i] == 0) continue;
    array[i] = 1/array[i];
}
```