

Министерство образования и науки РФ
Государственное образовательное учреждение
высшего профессионального образования

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

ПОДПРОГРАММЫ В С

Лабораторная работа № 9
по курсу «Программирование на ЯВУ»

Вариант № 4

Выполнил: студент группы 220601

_____ Белым А.А.
(подпись)

Проверил: к. ф.-м. н., доцент

_____ Сулимова В.В.
(подпись)

Тула 2011

Цель работы

Цель работы заключается в том, чтобы научиться использовать функции на языке программирования С. Также необходимо написать программу, демонстрирующие полученные знания.

Задание

1. Сформировать массив M , элементы которого $m_i = C_n^i$, где

$$C_n^i = \frac{n!}{i!(n-i)!}, n=20, i=1, 2, \dots, 5$$

2. Описать рекурсивную функцию $ROOT(F, A, B, EPS)$, которая методом деления отрезка пополам находит с точностью $EPS > 0$ корень уравнения $F(X)=0$ на отрезке $[A, B]$ (считать, что $A < B$, $F(A) < F(B)$ и $F(X)$ - непрерывная и монотонная функция на отрезке $[A, B]$).

1. ЗАДАЧА ФОРМИРОВАНИЯ МАССИВА СОЧЕТАНИЙ

1.1. Математическая формулировка

Формулу сочетания из n по k ($C_n^k = \frac{n!}{k!(n-k)!}$) можно представить в виде, возможно, более оптимальном для расчетов:

$$C_n^k = \frac{n!}{k!(n-k)!} = [m = \max\{k, n-k\}] = \frac{m! \cdot \prod_{i=m+1}^n i}{m!(n-m)!} = \frac{\prod_{i=m+1}^n i}{(n-m)!} = \frac{\prod_{i=m+1}^n i}{\prod_{i=1}^{n-m} i}$$

1.2. Схема алгоритма

На рисунке 1.1 представлена схема получения параметров расчета, заполнения массива сочетаний и его вывода.

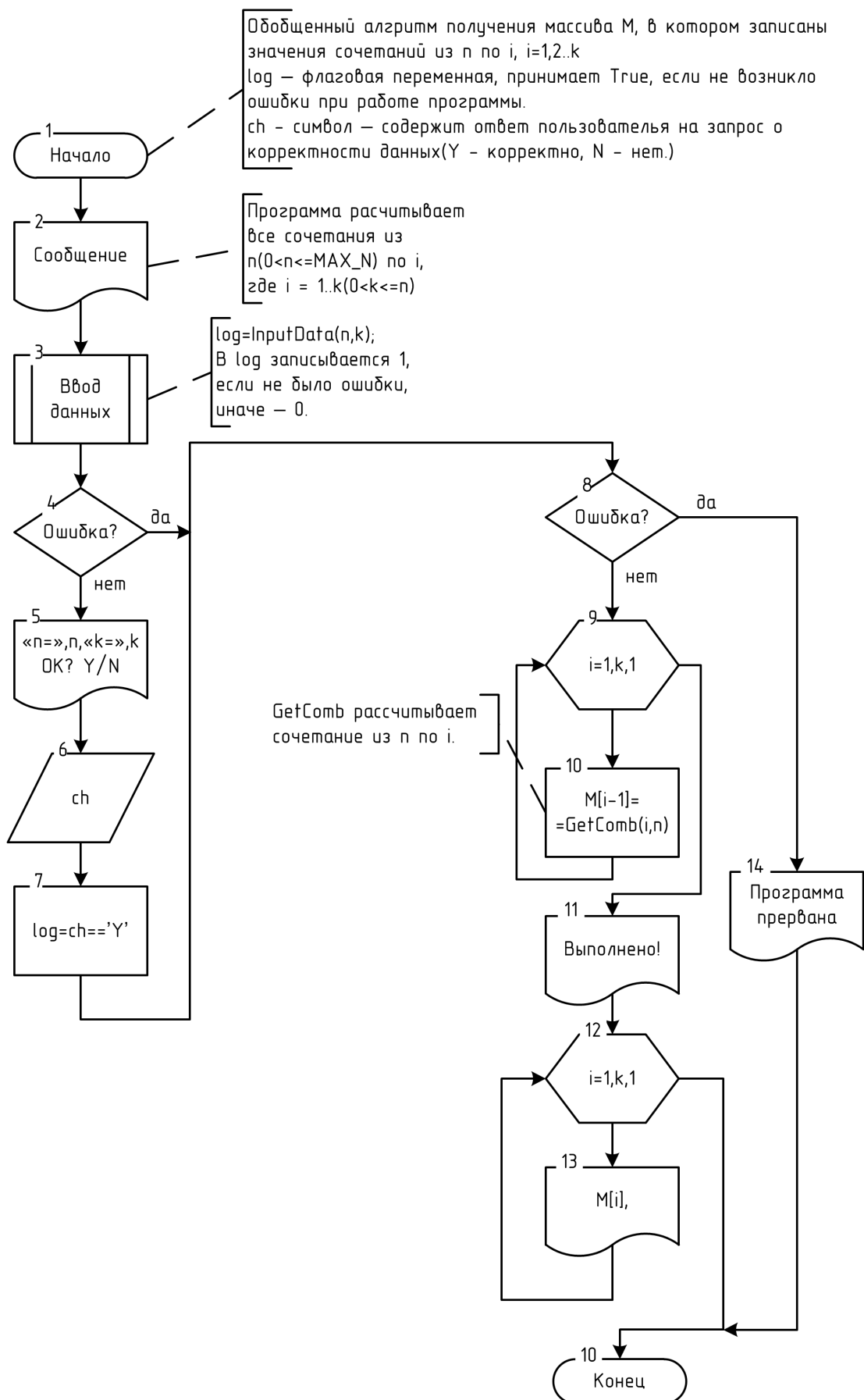


Рисунок 1.1 — Схема обобщенного алгоритма заполнения массива сочетаний

На рисунке 1.2 представлена схема ввода данных для расчета сочетаний.

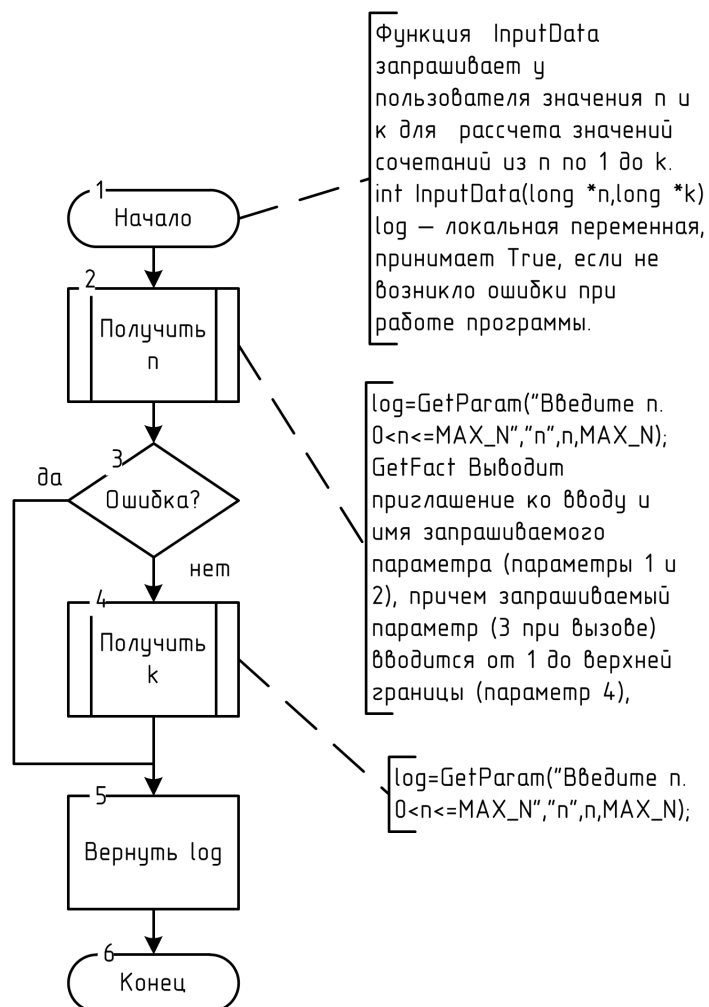


Рисунок 1.2 — Схема алгоритма получения данных для расчета сочетаний

На рисунке 1.3 представлена схема получения целого числа из промежутка от 1 до некоторого значения.

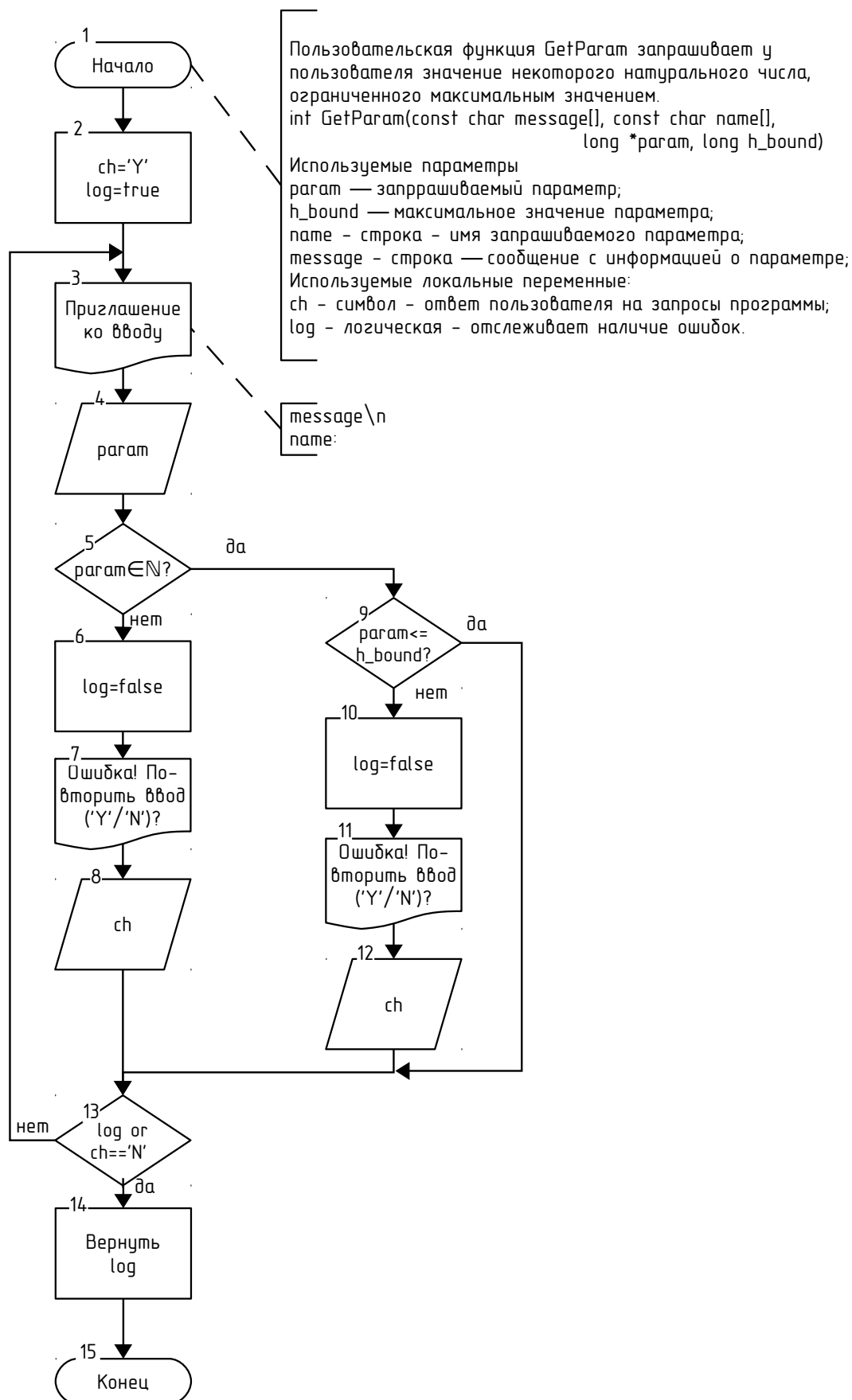


Рисунок 1.3 — Схема алгоритма получения натурального числа из промежутка от 1 до некоторого значения

На рисунке 1.4 представлена схема расчета сочетания из n по k .

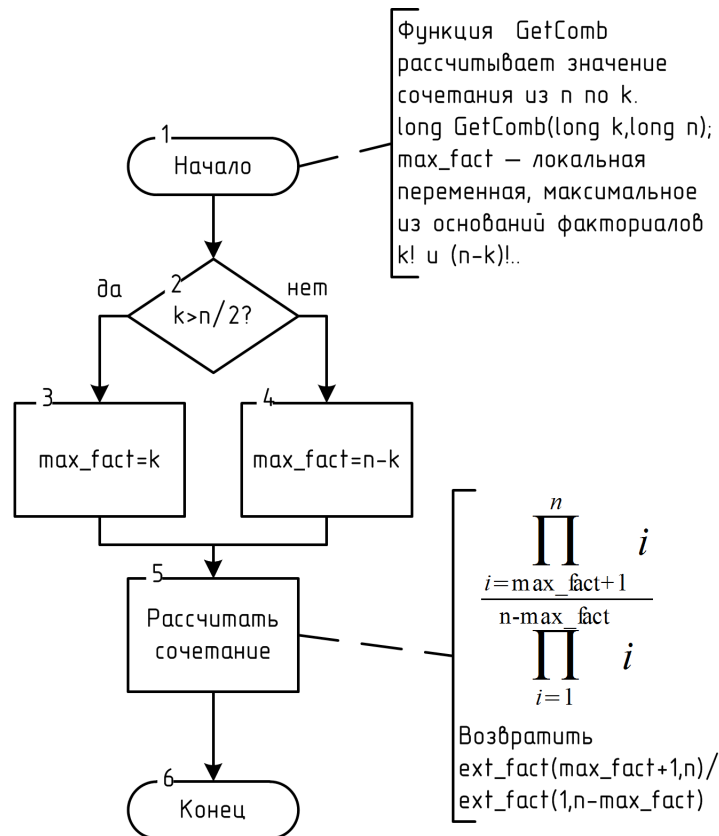


Рисунок 1.4 — Схема алгоритма расчета сочетания из n по k

На рисунке 1.5 представлена схема расчета выражения $\prod_{i=\text{beg}}^{\text{end}} i$.

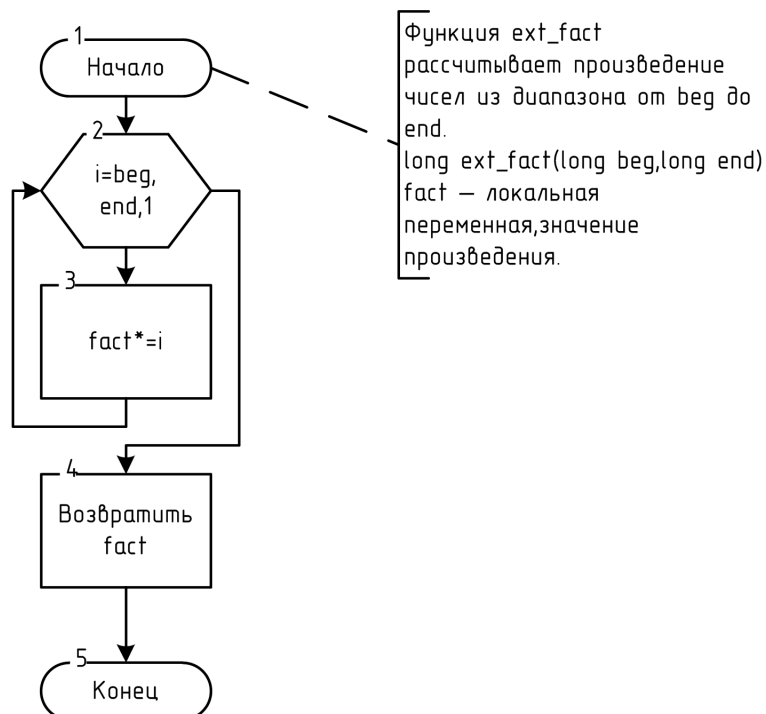


Рисунок 1.5 — Схема алгоритма расчета произведения $\prod_{i=\text{beg}}^{\text{end}} i$

1.3. Инструкция пользователю

Программа предназначена для вычисления ряда сочетаний C_n^i , где

$$C_n^i = \frac{n!}{i!(n-i)!}, \quad i=1,2..k. \text{ Числа } k \text{ и } n \text{ задаются с клавиатуры.}$$

Для работы передайте программе 2 натуральных числа n и k . Максимальное значение этих чисел подсказывается программой при вводе, минимальное значение равно 1. Программа либерально относится к ошибкам при вводе, позволяя исправить ошибку или отказаться от ввода. В последнем случае программа завершает работу с выводом соответствующего сообщения. После ввода программа предлагает проверить значения и отказаться от них, либо начать расчет.

После этого программа рассчитает и выведет k строк, где будут указаны конкретное i и C_n^i . Для завершения работы после вывода результатов нажмите <Enter>.

1.4. Инструкция программисту

Для решения задачи формирования массива сочетаний C_n^i были предприняты следующие действия.

Объявлен макрос MAX_N, который содержит максимальное значение n.

Подключены заголовочные файлы <ctype.h> - для функции toupper() и <stdio.h> для функций ввода-вывода.

Объявлены следующие структуры данных:

Таблица 1.1 - Структуры данных, используемые в основной части программы формирования массива сочетаний

| имя | тип | предназначение |
|-----|-------------|--|
| M | long[MAX_N] | Массив сочетаний. |
| n,k | long | Параметры расчета сочетаний: из n по i=1,2..k. (i – переменная-счетчик). |
| i | long | Переменная-счетчик. |
| log | int | Указывает на отсутствие ошибок (значение true) ввода данных или желания пользователя прервать программу. |
| ch | char | Ответ пользователя на запрос о корректности данных. |

Также программа была разбита на следующие подпрограммы:

1. Функция GetParam – используется для ввода натуральных чисел от 0 до некоторого значения; необходимы функции scanf, printf, clearline.

Заголовок функции:

```
int GetParam(const char message[],const char name[],long *param,long h_bound);
```

Функция запрашивает некоторый параметр param следующим образом:

Сначала выводится приглашающее ко вводу сообщение message, а также имя параметра name, и считывает с помощью функции scanf по формату "%ld". Если параметр считался, и буфер при очистке функцией clearline() оказался пуст, то проверяется, лежит ли значение в промежутке [1..h_bound]. Если да, то функция возвращает 1, во всех остальных случаях возвращается 0.

Параметры функции представлены в таблице 2, локальные переменные — в таблице 3.

Таблица 1.2 - Параметры функции получения значения целого числа

| ИМЯ | ТИП | предназначение |
|---------|--------|----------------------------------|
| message | char[] | Приглашение ко вводу параметра. |
| name | char[] | Имя запрашиваемого параметра. |
| param | long* | Запрашиваемый параметр. |
| h_bound | long | Максимальное значение параметра. |

Таблица 1.3 - Локальные переменные функции получения значения целого числа

| ИМЯ | ТИП | предназначение |
|----------|-----|--|
| log | int | Указывает на отсутствие ошибок (значение true) ввода данных. |
| req_rslt | int | Указывает на желание пользователя повторить ввод данных. |

2. Функция InputData — запрашивает параметры n и k; необходимы функция GetParam и константа MAX_N.

Заголовок функции:

```
int InputData(long *k, long *n);
```

Функция запрашивает значения чисел n и k следующим образом:

Сначала производится попытка получения значения n с помощью функции GetParam, в качестве верхней границы n берется значение MAX_N. Если операция прошла успешно, то запрашивается значение k, и в качестве верхней границы n берется значение n. Если все операции проведены успешно, функция возвращает 1, иначе возвращается 0.

Параметры функции представлены в таблице 1.4, локальные переменные — в таблице 1.5.

Таблица 1.4 - Параметры функции получения параметров расчета

| ИМЯ | ТИП | предназначение |
|------|-------|--|
| n, k | long* | Параметры расчета сочетания из n по k. |

Таблица 1.5 - Локальные переменные функции получения параметров расчета

| ИМЯ | ТИП | предназначение |
|-----|-----|--|
| log | int | Указывает на отсутствие ошибок (значение true) |

| | | |
|-----|----------|---|
| | | ввода данных. |
| msg | char[50] | Временная переменная для хранения сообщений о параметрах. |

3. Функция GetComb – рассчитывает сочетание из n по k; необходима функция ext_fact.

Заголовок функции:

long GetComb(long k, long n);

Функция рассчитывает сочетание C_n^k по формуле $C_n^k = \frac{\prod_{i=m+1}^n i}{\prod_{i=1}^{n-m} i}$, где m —

максимальное из k и n-k - выбирается с помощью условного оператора, а

выражения $\prod_{i=m+1}^n i$ и $\prod_{i=1}^{n-m} i$ считаются с помощью функции ext_fact.

Параметры функции представлены в таблице 1.6, локальные переменные — в таблице 1.7.

Таблица 1.6 - Параметры функции получения сочетания

| ИМЯ | ТИП | предназначение |
|-----|------|--|
| n,k | long | Параметры расчета сочетания из n по k. |

Таблица 1.7 - Локальные переменные функции получения сочетания

| ИМЯ | ТИП | предназначение |
|----------|------|--------------------------|
| max_fact | long | Максимальное из k и n-k. |

4. Функция ext_fact – рассчитывает произведение $\prod_{i=beg}^{end} i$.

Заголовок функции:

long ext_fact(long beg, long end);

Функция с помощью цикла for рассчитывает выражение $\prod_{i=beg}^{end} i$. Возвращает вычисленное значение.

Параметры функции представлены в таблице 1.8, локальные переменные — в таблице 1.9.

Таблица 1.8 - Параметры функции получения произведения

| имя | тип | предназначение |
|-----|------|--|
| n,k | long | Параметры расчета сочетания из n по k. |

Таблица 1.9 - Локальные переменные функции получения произведения

| имя | тип | предназначение |
|------|------|-------------------------|
| i | long | Переменная-счетчик. |
| fact | long | Результат произведения. |

5. Функция `clearline` используется для очистки строки файла.

Заголовок функции:

```
int clearline(FILE *f);
```

Функция считывает до конца файла или строки файла `f` символы в цикле `while` и возвращает их количество.

Параметры функции представлены в таблице 1.8, локальные переменные — в таблице 1.9.

Таблица 1.8 - Параметры функции получения сочетания

| имя | тип | предназначение |
|-----|-------|-----------------------------------|
| f | FILE* | Файл, в котором очищается строка. |

Таблица 1.9 - Локальные переменные функции получения сочетания

| имя | тип | предназначение |
|-------|------|-----------------------------|
| count | long | Счетчик считанных символов. |

6. Функция `GetAskResult` используется для получения ответов пользователя на запросы программы.

Заголовок функции:

```
int GetAskResult();
```

Функция запрашивает у пользователя символы `'y','Y','n'` или `'N'` до тех пор, пока он не введет какой-либо из них. Соответственно возвращается либо символ `'Y'`, либо `'N'`.

Локальные переменные функции представлены в таблице 1.10.

Таблица 1.10 - Локальные переменные функции получения сочетания

| имя | тип | предназначение |
|------------|------------|-----------------------|
| answer | char | Ответ пользователя. |

1.5. Текст программы

Ниже представлен текст программы на языке ANSI C, компилируемый Borland C 3.1 и GCC 4.5.2 , вычисляющей массив сочетаний.

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

/***** МАКРОСЫ-КОНСТАНТЫ *****/

#define MAX_N 20L //число элементов в множестве для перестановок

/***** ПРОТОТИПЫ ФУНКЦИЙ *****/
int InputData(long *k, long *n);
long GetComb(long k, long n);
int clearline(FILE *f);
long ext_fact(long beg, long end);
int GetAskResult();
int GetParam(const char message[], const char name[], long *param, long h_bound);
/***** MAIN PROGRAM *****/
int main() {

    long i, M[MAX_N];
    int log; long n, k; char ch;

    printf("Программа рассчитывает все сочетания из n(0<n<=%ld) по i, \n" \
           "где i = 1..k(0<k<=n) \n", MAX_N);
    log=InputData(&n, &k);
    if(log) {
        printf("Вы готовы? Y/N \n");
        ch=GetAskResult();
        log=ch=='Y';
    }
    if(log) {
        for(i=1; i<=k; i++)
            M[i-1]=GetComb(i, n);

        printf("Значения сочетаний из %ld по...\n", n);
        for(i=0; i<=k-1; i++)
            printf("%ld: %ld\n", i+1, M[i]);
    }
    else
        printf("Работа программы прервана!\n");
    printf("Нажмите <Enter>...\n");
}
```

```

        clearline(stdin);
        return 0;
    }
/***** END MAIN PROGRAM *****/
int InputData(long *n,long *k) {
    int log; char msg[50];
    sprintf(msg,"Введите n. 0<n<=%ld\n",MAX_N);
    log=GetParam(msg,"n",n,MAX_N);
    if (log){
        sprintf(msg,"Введите k. 0<k<=%ld\n",*n);
        log=GetParam(msg,"k",k,*n);
    }
    return log;
}

int GetParam(const char message[],const char name[],long *param,long h_bound) {
    int log,req_rslt;
    do{
        req_rslt=0;
        printf("%s",message);
        printf("%s: ",name);
        log=scanf("%ld",param);
        log=!clearline(stdin)&&log;
        if (!log){ //введено не вещественное число?
            printf("Введено не число!\n"
                "Повторить ввод? Y/N\n");
            req_rslt=GetAskResult()=='Y';
        }
        if (log&&(*param<1||*param>h_bound)){
            log=0;
            printf("Ошибка!%s должен быть больше 0 и меньше %ld!\n",name,h_bound);
            printf("Повторить ввод? Y/N\n");
            req_rslt=GetAskResult()=='Y';
        }
    } while(!log&&req_rslt);
    return log;
}

int GetAskResult() {
    char answer;
    answer=getchar();
    clearline(stdin);
    while (toupper(answer)!='Y'
        &&toupper(answer)!='N'){
        printf("Неправильный ответ! Допустимо:\n\"

```

```

        "Y - да; N - нет.\n");

        answer=getchar();
        clearline(stdin);
    }

    return toupper(answer);
}

long GetComb(long k,long n){
    long max_fact;
    if(k>n/2)
        max_fact=k;
    else
        max_fact=n-k;
    return ext_fact(max_fact+1,n)/ext_fact(1,n-max_fact);
}

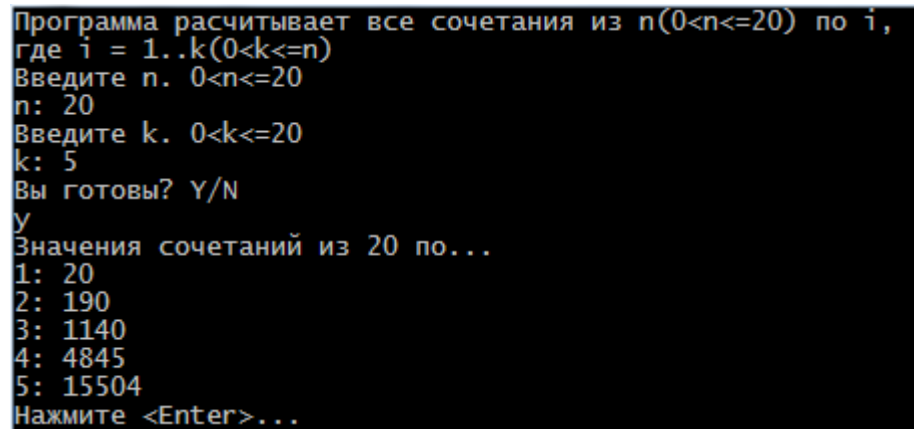
long ext_fact(long beg,long end){
    long fact=1,i;
    for (i=beg;i<=end;i++)
        fact*=i;
    return fact;
}

int clearline(FILE *f){
    int count=0;
    while (!feof(f)&&(getc(f)!='\n')) count++;
    return count;
}

```

1.6. Тестовый пример

Ниже на рисунке 1.6 представлен результат работы программы для вычисления сочетаний с параметрами $n=20$ и $k=5$.



```
Программа рассчитывает все сочетания из n(0<n<=20) по i,  
где i = 1..k(0<k<=n)  
Введите n. 0<n<=20  
n: 20  
Введите k. 0<k<=20  
k: 5  
Вы готовы? Y/N  
y  
Значения сочетаний из 20 по...  
1: 20  
2: 190  
3: 1140  
4: 4845  
5: 15504  
Нажмите <Enter>...
```

Рисунок 1.6 — Результат работы программы для $n=20$, $k=5$

2. ЗАДАЧА ПОИСКА КОРНЯ ФУНКЦИИ С ПОМОЩЬЮ РЕКУРСИИ

2.1. Схема алгоритма

На рисунке 2.1 представлена схема алгоритма ввода данных, расчета корня функции и его вывода.

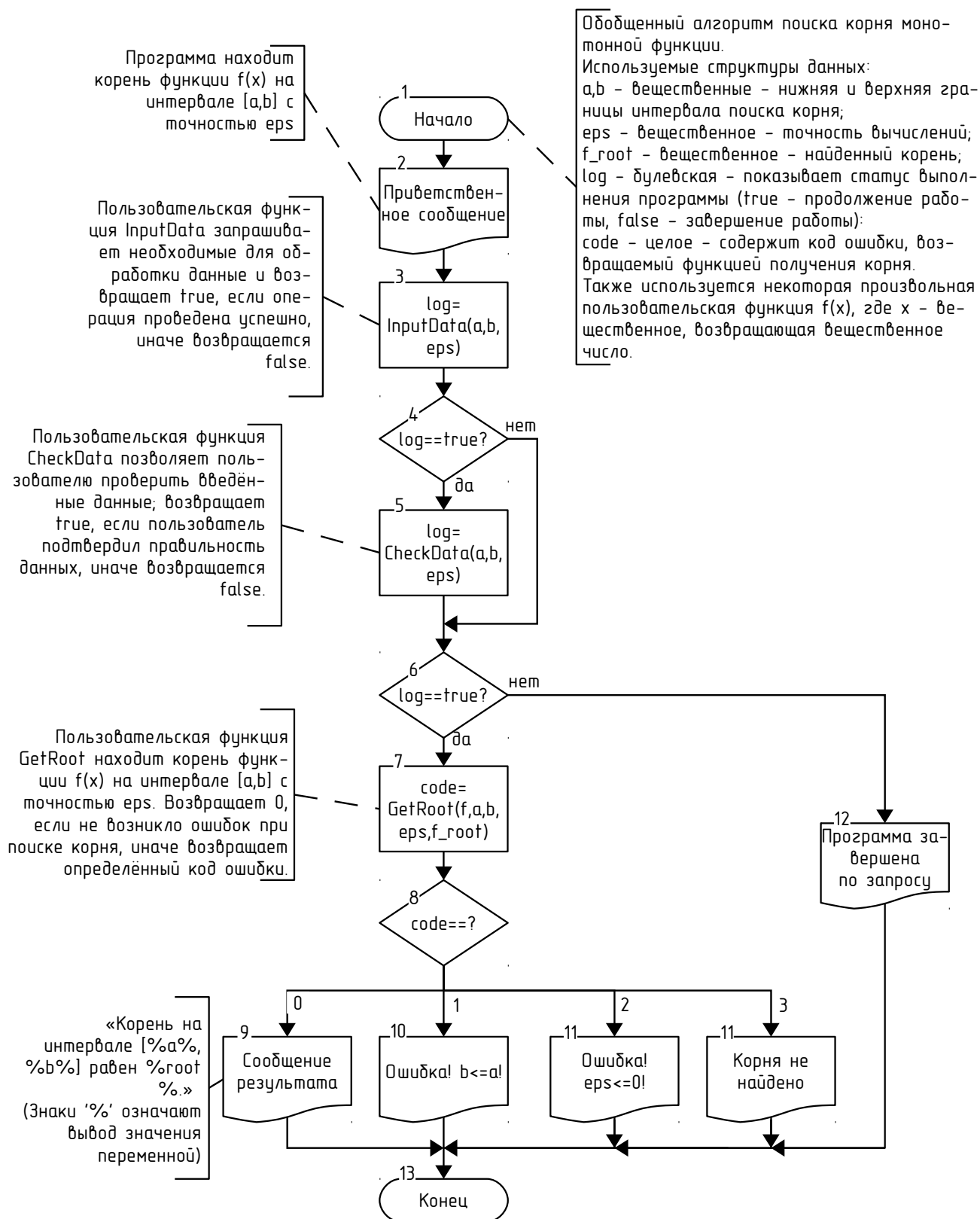


Рисунок 2.1 — Блок-схема обобщенного алгоритма поиска корня функции

На рисунках 2.2 и 2.3 представлена схема алгоритма ввода параметров расчета корня функции.

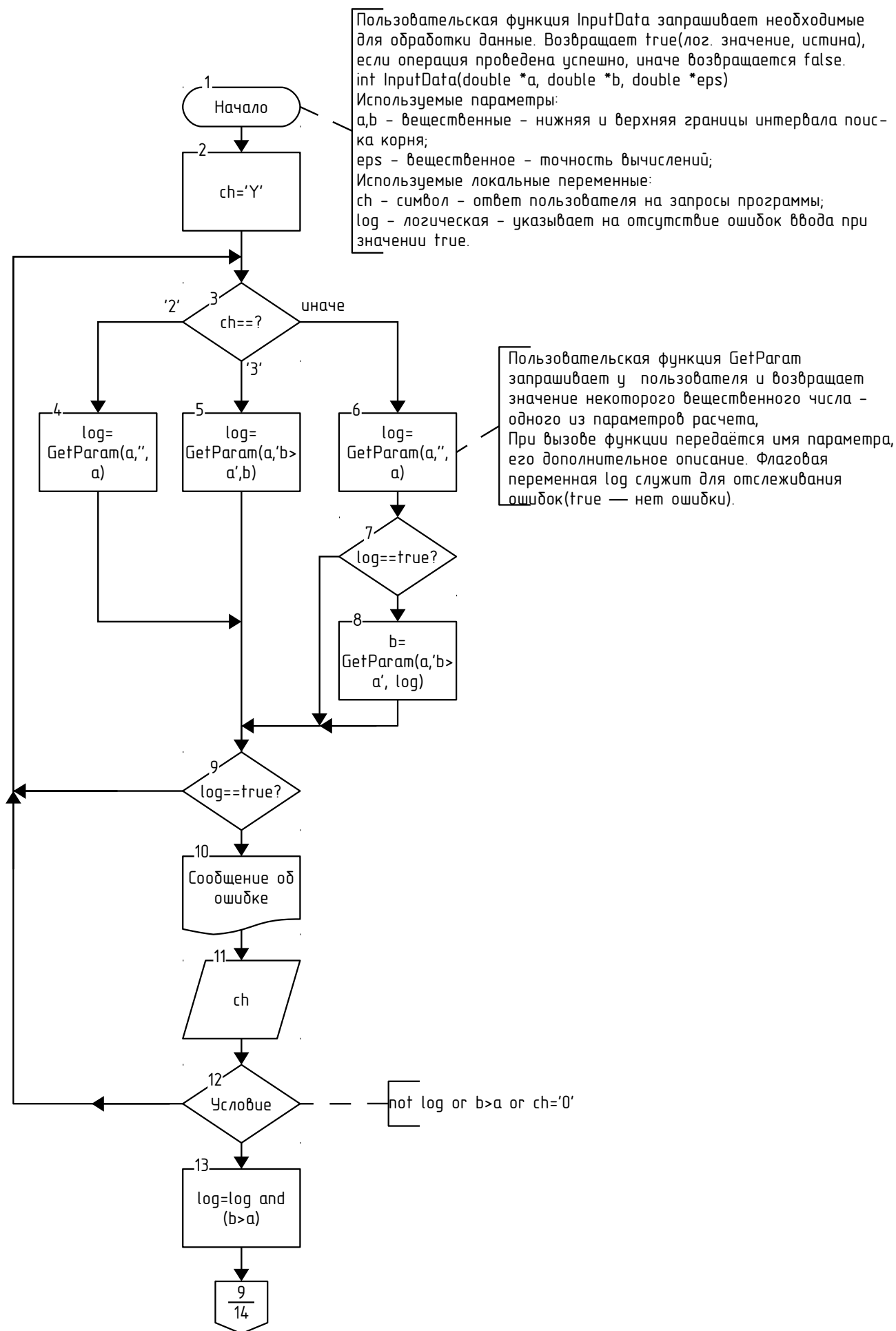


Рисунок 2.2 — Блок-схема алгоритма ввода параметров расчета корня функции(начало)

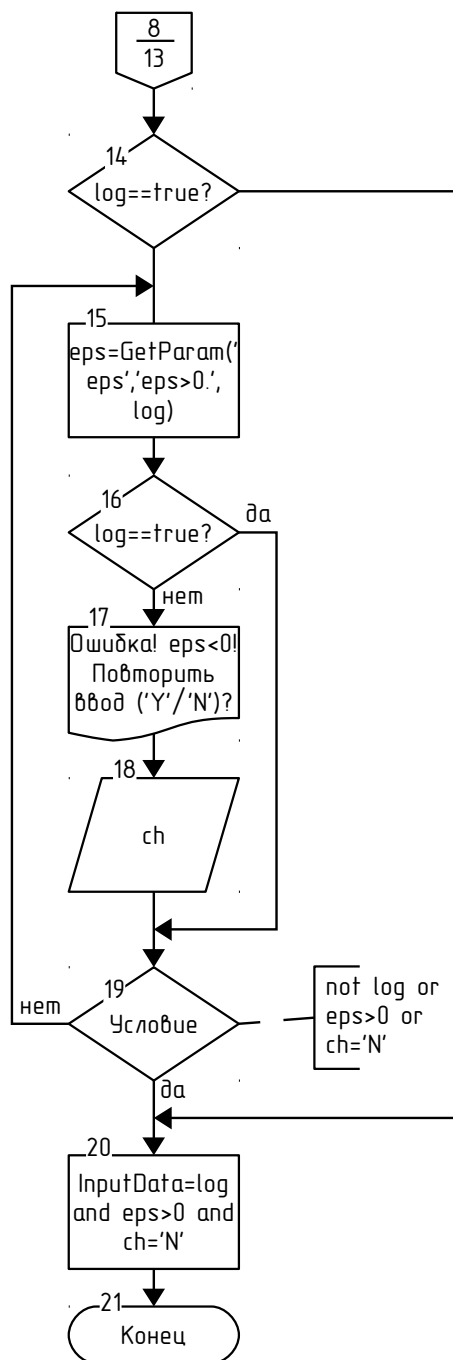


Рисунок 2.3 — Блок-схема алгоритма ввода параметров расчета корня функции(продолжение)

На рисунке 2.4 представлена блок-схема алгоритма запроса одного из параметров расчета.

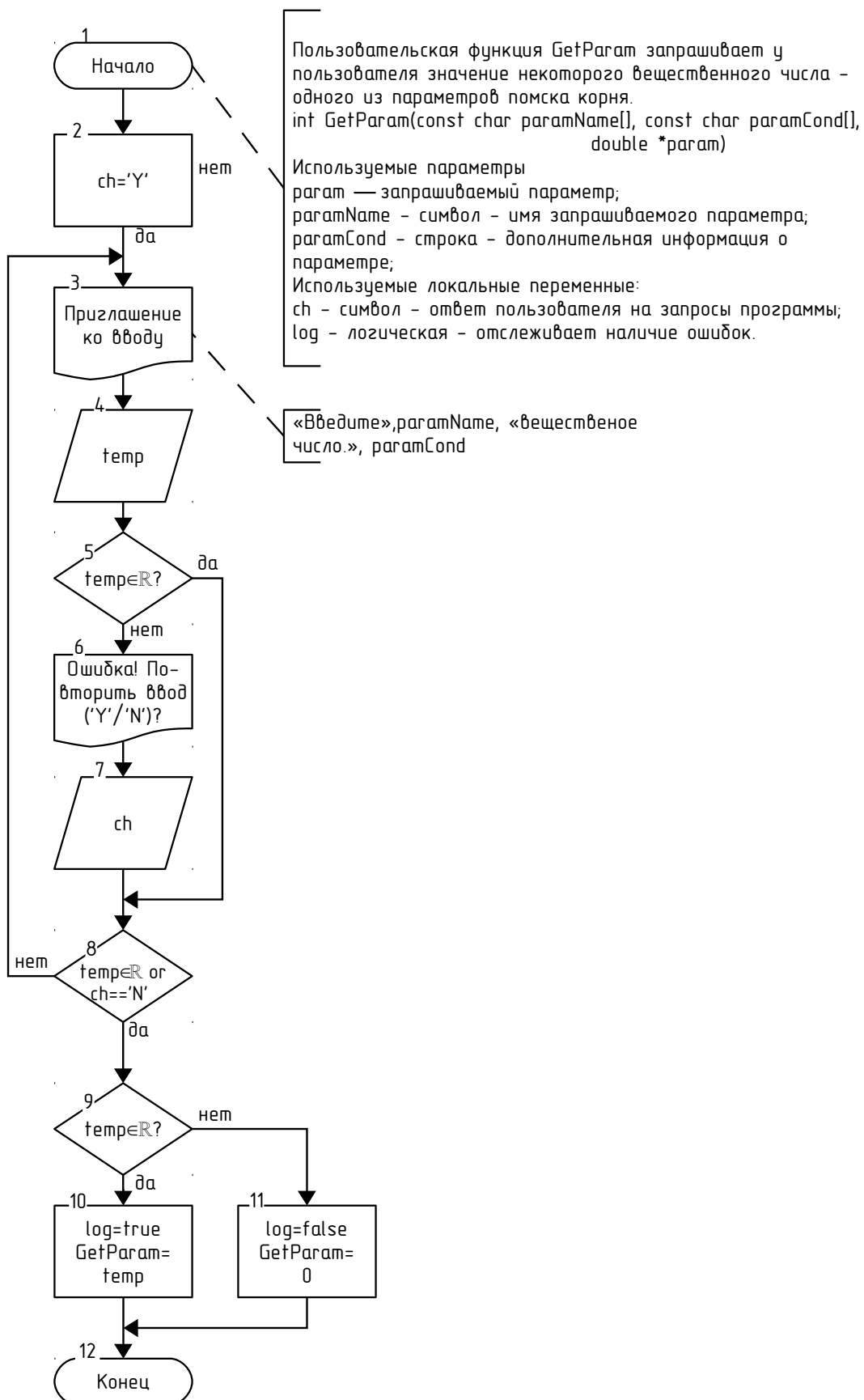


Рисунок 2.4 — Блок-схема алгоритма ввода произвольного параметра

На рисунке 2.5 представлена схема алгоритма организации проверки пользователем введенных данных.

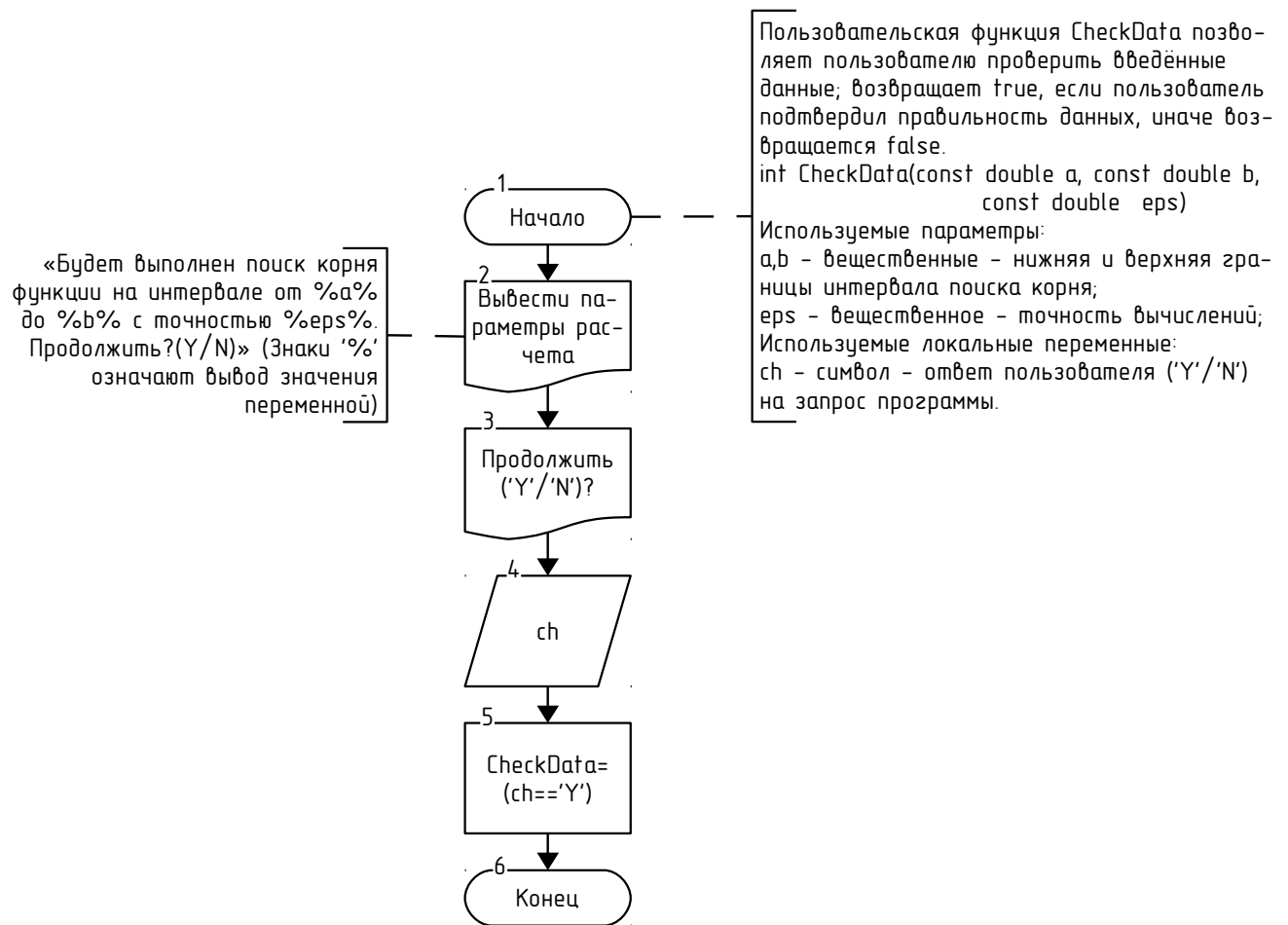


Рисунок 2.5 — Блок-схема алгоритма организации проверки пользователем введенных данных

На рисунке 2.6 представлена схема алгоритма проверки параметров для расчета корня функции и получения корня.

Пользовательская функция GetRoot проверяет переданные данные и находит корень функции $f(x)$ на интервале $[a,b]$ с точностью eps . Возвращает 0, если не возникло ошибок при поиске корня, иначе возвращает 1, если верхняя граница интервала поиска (b) меньше нижней (a), 2 – если точность вычислений (eps) равна 0, 3 – если на заданном участке нет корня.

```
double GetRoot( double (*f)(double x), const double a,
               const double b, const double eps,
               double *f_root)
```

Используемые параметры:

a, b – вещественные – нижняя и верхняя границы интервала поиска корня;
 eps – вещественное – точность вычислений;
 f_root – вещественное – найденный корень;
как параметр также передается некоторая произвольная пользовательская функция $f(x)$, где x – вещественное, возвращающая вещественное число.

Пользовательская функция Root рассчитывает корень функции $f(x)$ на интервале $[a,b]$ с точностью eps . Возвращает искомый корень

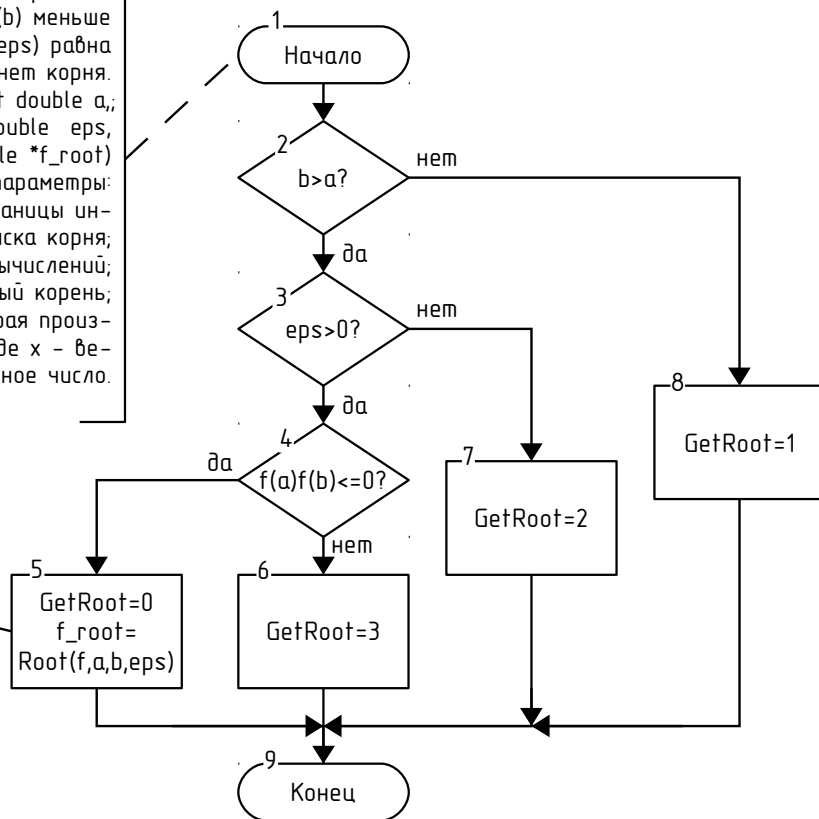


Рисунок 2.6 — Блок-схема алгоритма оценки данных и получения корня функции

На рисунке 2.7 представлена схема алгоритма расчета корня функции методом половинного деления.

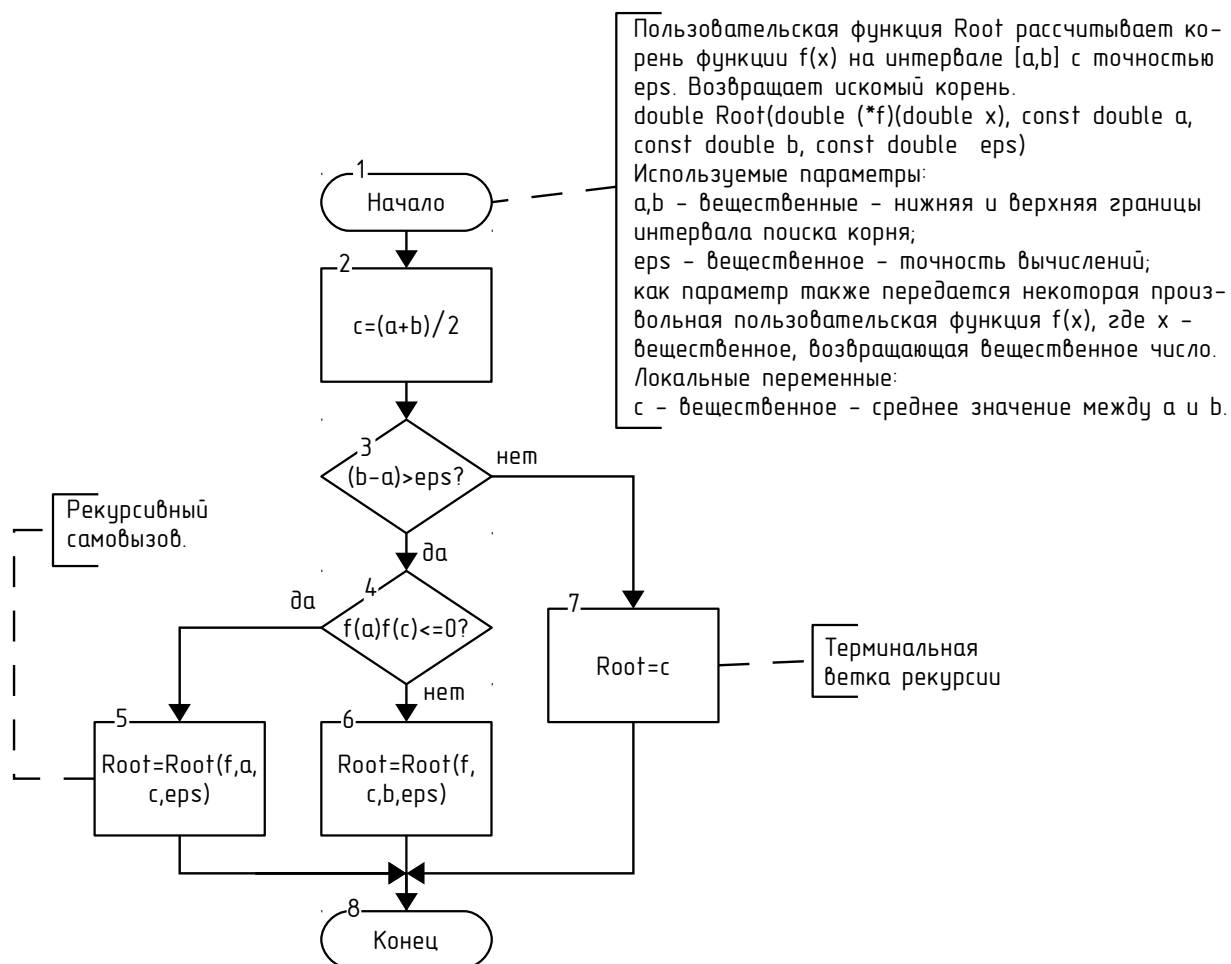


Рисунок 2.7 — Схема алгоритма расчета корня функции методом половинного деления

2.2. Инструкция пользователю

Данная программа позволяет найти корень монотонной функции на заданном промежутке.

Для работы программы необходимо ввести с клавиатуры сначала нижнюю границу отрезка поиска корня, затем верхнюю. Эти параметры - вещественные числа, причём верхняя граница обязательно строго больше нижней, в противном случае имеется возможность ввести заново верхнюю границу, или нижнюю границу, или обе сразу заново. Также есть возможность отказаться от повторного ввода и завершить программу. Затем вводится требуемая точность вычислений - вещественное число, большее нуля. После этого предоставляется возможность проверить введённые данные; если они корректны, для продолжения работы нужно ввести 'у', ввод любого другого значения приведёт к завершению работы программы с соответствующим сообщением.

В качестве результатов работы программа выведет искомый корень функции; иначе если функция не имеет корня на данном участке или она на нём не монотонна, будет выведено сообщение об отсутствии корня функции на данном промежутке.

2.3. Инструкция программисту

При создании программы вычисления корня монотонной функции на заданном отрезке были предприняты следующие действия.

Подключены заголовочные файлы `<stdio.h>` - для функций ввода-вывода, `<ctype.h>` для функции `toupper()` и `<math.h>` для функции `sin(x)`.

Были введены структуры данных, описание которых представлено в таблице 2.1.

Таблица 2.1 - Структуры данных, используемые в в основной части программы оценки элементов массива

| имя | тип | предназначение |
|------|--------|--|
| a,b | double | Границы отрезка поиска корня функции |
| eps | double | Точность вычислений. |
| log | int | Указывает на отсутствие ошибок (значение true) ввода данных или желания пользователя прервать программу. |
| code | int | Код ошибки вычисления корня функции; код 0 - нет ошибки |

Кроме того, в процессе создания вышеуказанной программы были определены следующие подпрограммы:

1. Анализируемая функция f - любая функция, по определенная как `double (*function)(double x)`.

`double f(double x);`

В данной программе функция возвращает синус от x . В таблице 2.2 приведено описание её параметра.

Таблица 2.2 - Параметры-константы функции, для которой ищется корень

| имя | тип | предназначение |
|-----|--------|--|
| x | double | Аргумент функции <code>sin(x)</code> . |

2. Функция `GetParam` запрашивает у пользователя произвольный параметр расчета - вещественное число. Возвращает запрошенный параметр; кроме того, в случае ошибки пользователя уведомляет об этом основную программу с помощью переменной `log`.

`function GetParam(const paramName, paramCond:string;var log:boolean):real;`

В теле функции в цикле с постусловием происходит запрос у пользователя вещественного числа, также выводится дополнительная информация о параметре. Запрошенное значение сначала считывается в буфер, потом производится попытка его извлечения (функцией `val`). Если попытка была успешной, функция завершает работу и передает указанное значение и значение `log`, равное `true`. Если нет, то производится запрос пользователя о продолжении работы. Если пользователь ответил 'N', функция завершает работу и возвращает значение `log`, равное `false`. Иначе операция повторяется до достижения двух вышеуказанных условий.

Используемые функцией параметры-константы приведены в таблице 2.3; параметры-переменные - в таблице 2.4, локальные переменные - в таблице 2.5.

Таблица 2.3 - Параметры-константы функции ввода произвольного параметра расчета корня

| имя | тип | предназначение |
|------------------------|---------------------|--|
| <code>paramName</code> | <code>char[]</code> | Имя запрашиваемого параметра. |
| <code>paramCond</code> | <code>chat[]</code> | Дополнительная информация о запрашиваемом параметре. |

Таблица 2.4 - Параметры-переменные функции ввода произвольного параметра расчета корня

| имя | тип | предназначение |
|--------------------|----------------------|-------------------------|
| <code>param</code> | <code>*double</code> | Запрашиваемый параметр. |

Таблица 2.5 - Локальные переменные функции ввода произвольного параметра расчета корня

| имя | тип | предназначение |
|------------------|----------------------|---|
| <code>ch</code> | <code>char</code> | Содержит ответ пользователя на запросы программы о повторении ввода данных. |
| <code>log</code> | <code>boolean</code> | Указывает на желание пользователя продолжить работу программы(при значении <code>true</code> , <code>false</code> - завершение программы) |

3. Функция `InputData` вводит параметры расчёта - границы отрезка поиска корня и точность вычислений; возвращает `true`, если не было ошибки ввода, иначе возвращается `false`. Функция `InputData` использует функцию `GetParam`, поэтому она должна быть глобальной по отношению к `InputData`.

```
function InputData(var a,b,eps:real):boolean;
```

В цикле с постусловием происходит запрос параметров следующим образом:

Перед входом в цикл значение переменной `ch` инициализируется как 'Y'. Уже в цикле происходит ветвление с помощью оператора `case` по переменной `ch`: если `ch` равно 'A', то происходит запрос параметра `a`, если 'B', то запрос параметра `b`, иначе запрашиваются оба значения. Если `b` оказывается больше `a`, пользователю предлагается закончить работу (ответом 'N') или ввести заново значения указанным выше образом. Затем, если не возникло ошибок при вводе, запрашивается значение `eps` до тех пор, пока оно не будет больше 0 или пользователь не откажется от ввода. Вся процедура повторяется, пока все параметры не будут введены правильно, и тогда будет возвращено значение `true`, или пользователь не откажется от продолжения работы (и тогда возвращается `false`).

Используемые функцией параметры-переменные приведены в таблице 2.6; локальные переменные - в таблице 2.7.

Таблица 2.6 - Параметры-переменные функции ввода данных для расчета корня

| имя | тип | предназначение |
|------------------|---------------------|--------------------------------------|
| <code>a,b</code> | <code>double</code> | Границы отрезка поиска корня функции |
| <code>eps</code> | <code>double</code> | Точность вычислений. |

Таблица 2.7 - Локальные переменные функции ввода данных для расчета корня

| имя | тип | предназначение |
|------------------|-------------------|---|
| <code>ch</code> | <code>char</code> | Содержит ответ пользователя на запрос программы о правильности данных. |
| <code>log</code> | <code>int</code> | Указывает на желание пользователя продолжить работу программы(при значении <code>true</code> , <code>false</code> - завершение программы) |

4. Функция `CheckData` позволяет организовать проверку пользователем введенных им ранее значений. Возвращает `true`, если пользователь подтвердил правильность данных.

```
function CheckData(const a,b,eps:real):boolean;
```

Функция выводит значения параметров и спрашивает пользователя о продолжении работы. Если он отвечает 'Y', то функция возвращает `true`, иначе возвращается `false`.

Используемые функцией параметры-константы приведены в таблице 2.8; локальные переменные - в таблице 2.9.

Таблица 2.8 - Параметры-константы функции проверки пользователем введенных значений

| имя | тип | предназначение |
|-----|--------|--------------------------------------|
| a,b | double | Границы отрезка поиска корня функции |
| eps | double | Точность вычислений. |

Таблица 2.9 - Локальные переменные функции проверки пользователем введенных значений

| имя | тип | предназначение |
|-----|------|--|
| ch | char | Содержит ответ пользователя на запрос программы о правильности данных. |

5. Функция GetRoot проверяет переданные данные и получает корень функции f с помощью функции Root (определена локально по отношению к GetRoot). Возвращает отпределённый код ошибки:

0 - ошибки нет;

1 - верхняя граница отрезка поиска корня меньше нижней;

2 - точность вычислений меньше или равна 0;

3 - корня на заданном участке не найдено.

`function GetRoot(const f:TestFunc; const a,b,eps:real;var f_root:real):byte;`

Функция проверяет вышеуказанные условия, и если они выполняются, возвращает соответственный код ошибки. Если возвращается код 0, то через переменную `f_root` возвращается найденный корень.

Используемые функцией параметры-константы приведены в таблице 2.10, параметры-переменные - в таблице 2.11.

Таблица 2.10 - Параметры-константы функции проверки данных и получения корня

| имя | тип | предназначение |
|-----|-------------------------------------|--------------------------------------|
| a,b | double | Границы отрезка поиска корня функции |
| eps | double | Точность вычислений. |
| f | double (*function)(double x) | Анализируемая функция |

Таблица 2.11 - Параметры-переменные функции проверки данных и получения корня

| имя | тип | предназначение |
|--------|--------|------------------------------|
| f_root | double | Корень анализируемой функции |

5.1 Функция Root рассчитывает корень функции f. Возвращает искомый корень. Является рекурсивной; поэтому проверки передаваемых значений в целях повышения производительности в ней не производятся, и поэтому Root определена локальной функцией по отношению к GetRoot.

```
function Root(const f:TestFunc; const a,b,eps:real):real;
```

Сначала находится среднее значение между a и b и записывает его в c. Если разница между a и b меньше eps, то возвращается c; иначе если $f(a)*f(c) \leq 0$, то возвращается значение от рекурсивного вызова Root(f,a,c,eps), иначе возвращается значение от вызова Root(f,c,b,eps).

Используемые функцией параметры-константы приведены в таблице 2.12; локальные переменные - в таблице 2.13.

Таблица 2.12 - Параметры-константы функции расчета корня

| имя | тип | предназначение |
|-----|-------------------------------------|--------------------------------------|
| a,b | double | Границы отрезка поиска корня функции |
| eps | double | Точность вычислений. |
| f | double (*function)(double x) | Указатель на анализируемую функцию |

Таблица 2.13 - Локальные переменные функции расчета корня

| имя | тип | предназначение |
|-----|--------|-------------------------------|
| c | double | Среднее значение между a и b. |

6. Функция clearline используется для очистки строки файла.

Заголовок функции:

```
int clearline(FILE *f);
```

Функция считывает до конца файла или строки файла f символы в цикле while и возвращает их количество.

Параметры функции представлены в таблице 2.14, локальные переменные — в таблице 2.15.

Таблица 2.14 - Параметры функции получения сочетания

| имя | тип | предназначение |
|-----|-------|-----------------------------------|
| f | FILE* | Файл, в котором очищается строка. |

Таблица 2.15 - Локальный переменные функции получения сочетания

| имя | тип | предназначение |
|-------|------|-----------------------------|
| count | long | Счетчик считанных символов. |

7. Функция GetReqResult используется для получения ответов пользователя на запросы программы.

Заголовок функции:

```
int GetAskResult();
```

Функция запрашивает у пользователя символы 'y','Y','n' или 'N' до тех пор, пока он не введет какой-либо из них. Соответственно возвращается либо символ 'Y', либо 'N'.

Локальные переменные функции представлены в таблице 1.10.

Таблица 2.16 - Локальный переменные функции получения сочетания

| имя | тип | предназначение |
|--------|------|---------------------|
| answer | char | Ответ пользователя. |

8. Функция GetOption позволяет выбирать пользователю из нескольких (до 10) вариантов ответа.

Заголовок функции:

```
int GetOption(char a,char b);
```

Функция запрашивает у пользователя символы из промежутка от a до b до тех пор, пока он не введет какой-либо из них. Соответственно возвращается значение введенного символа.

Таблица 2.17 - Параметры функции получения сочетания

| имя | тип | предназначение |
|-----|------|--|
| a,b | char | Различные варианты представлены цифрами от a до b. |

Таблица 2.18 - Локальный переменные функции получения сочетания

| имя | тип | предназначение |
|------------|------------|--|
| ch | char | Ответ пользователя. |
| ok | int | Флаг — равен 1, если ответ пользователя допустим, иначе равен 0. |

2.4. Текст программы

Ниже представлен текст программы, написанной на языке Borland C 3.1, которая находит корень функции $\sin(x)$ на заданном промежутке и с заданной ТОЧНОСТЬЮ.

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

#define FUNC_NAME "sin(x)"
double f(double x){
    return sin(x);
}

int clearline(FILE *f){
    int count=0;
    while (!feof(f)&&(getc(f)!='\n')) count++;
    return count;
}

int GetReqResult(){
    char answer;
    answer=getchar();
    clearline(stdin);
    while (toupper(answer)!='Y'
           &&toupper(answer)!='N'){
        printf("Неправильный ответ! Допустимо:\n\"
               \"Y - да; N - нет.\n");
        answer=getchar();
        clearline(stdin);
    }

    return toupper(answer);
}

/***** ФУНКЦИЯ ВВОДА ВЕЩЕСТВЕННОГО ПАРАМЕТРА *****/
int GetParam(const char paramName[], //paramName - имя запрашиваемого параметра;
             const char paramCond[], //paramCond - дополнительная информация о
параметре;
             double *param
             ){
    int log; char ch;
    do {
        printf("Введите %s - вещественное число. %s\n",paramName,paramCond);
```

```

printf("%s: ",paramName);
log=scanf("%lf",param);
log=!clearline(stdin)&&log;
if (!log){ //введено не вещественное число?
    printf("Введено неправильное значение!\n");
    printf("Хотите повторить ввод? Y/N\n" );
    ch=GetReqResult();
}
} while (!log&&toupper(ch)=='Y'); //пока пользователь не отказался или число
некорректное
return log;
}

/***** ФУНКЦИЯ ВВОДА ДАННЫХ *****/
int InputData(double *a, //a,b - границы отрезка поиска корня;
             double *b,
             double *eps //eps - точность вычислений;
             ){
    char ch='Y'; int log=1; //инициализация так и только так! см. ниже, почему
    do{
        switch(toupper(ch)){
            case 'A':log=GetParam("a","",a);
            break;
            case 'B':log=GetParam("b","b>a.",b);
            break;
            default:
                log=GetParam("a","",a);
                if (log)
                    log=GetParam("b","b>a.",b);
            break;
        }
        if (log&&!(b>a)){
            printf("Ошибка! b<=a!\n");
            printf("Повторить ввод?\n");
            printf("Y - ввести a и b;\n");
            printf("A - ввести только a;\n");
            printf("B - ввести только b;\n");
            printf("N - отмена.\n");
            scanf("%c",&ch);
            clearline(stdin);
        }
    }while (((!log)||(*b<=a))&&(toupper(ch)!='N'));
    log=log&&(*b>a);
    if (log)

```

```

do{
    log=GetParam("eps","eps>0.",eps);
    if (log&&!( *eps>0)){
        printf("Ошибка! eps<=0!\n");
        printf("Повторить ввод?(Y/N)\n");
        ch=GetReqResult();
    }
    }while (((!log)|| (eps<0))&&(toupper(ch)=='Y'));
log=log&&(*eps>0);
return log&&(ch!='N');
}

/***** ФУНКЦИЯ ПРОВЕРКИ ДАННЫХ *****/
int CheckData(const double a, //a,b - границы отрезка поиска корня;
              const double b,
              const double eps //eps - точность вычислений;
              ){
    printf("Будет выполнен поиск корня\n"\
           "на интервале от %1.4lf до %1.4lf с точностью %1.8lf\n"\
           "Продолжить? (Y/N)\n",a,b,eps);
    return GetReqResult()=='Y';
}

/***** ФУНКЦИЯ РАСЧЕТА КОРНЯ *****/
double Root(double (*f)(double x), //f - анализируемая функция;
            const double a, //a,b - границы отрезка поиска корня;
            const double b,
            const double eps //eps - точность вычислений;
            ){
    double c;
    c=(a+b)/2;
    if (b-a>eps)
        if ((*f)(c)*(*f)(a)<=0)
            return Root(f,a,c,eps);
        else
            return Root(f,c,b,eps);
    else
        return c;
}

/***** ФУНКЦИЯ ПОЛУЧЕНИЯ КОРНЯ *****/
double GetRoot( double (*f)(double x), //f - анализируемая функция;
               const double a, //a,b - границы отрезка поиска корня;

```

```

        const double b,
        const double eps, //eps - точность вычислений;
        double *f_root //f_root - найденный корень
    ){

    if (!(b>a))
        return 1;
    else {
        if (!(eps>0))
            return 2;
        else {
            if (f(a)*f(b)>0)
                return 3;
            else {
                *f_root=Root(f,a,b,eps);
                return 0;
            }
        }
    }
}

/***** ОСНОВНАЯ ПРОГРАММА *****/

int main(){
    //a,b - границы отрезка поиска корня;
    //eps - точность вычислений;
    //log - переменная состояния;
    //code - код ошибки поиска корня}

    double a,b,eps,f_root;
    int log; int code;

    printf("Программа находит корень функции %s на отрезке [a,b] с точностью
eps.\n",FUNC_NAME);
    log=InputData(&a,&b,&eps);
    if (log)
        log=CheckData(a,b,eps);
    if (log) {
        code=GetRoot(&f,a,b,eps,&f_root);
        switch (code){
            case 0:
                printf("Корень на промежутке [%1.4lf,%1.4lf] равен
%1.8lf.\n",a,b,f_root);
                break;

```



```

        case 1:
            printf("Верхняя граница интервала поиска корня (b) меньше нижней
(a)!\n");
            break;
        case 2:
            printf("Точность вычислений (eps) меньше/равна 0!\n");
            break;
        case 3:
            printf("Корня на промежутке [%1.4lf,%1.4lf] не найдено.\n",a,b);
            break;
    }
}

else
    printf("Работа программы прервана!\n");
    printf("Нажмите <Enter>...\n");
    clearline(stdin);
    return 0;
}

```

2.5. Тестовый пример

На рисунке 2.8 представлен пример работы программы нахождения корня функции для функции $y=\sin(x)$ на отрезке $[3,4]$ с точностью 0.00001.

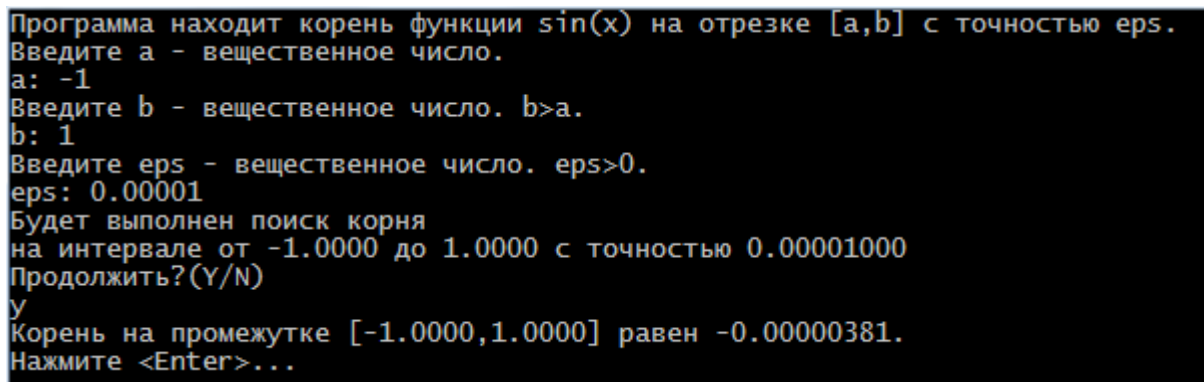
A screenshot of a text-based program interface. The text is as follows:
Программа находит корень функции $\sin(x)$ на отрезке $[a,b]$ с точностью ϵ .
Введите a - вещественное число.
a: -1
Введите b - вещественное число. $b>a$.
b: 1
Введите ϵ - вещественное число. $\epsilon>0$.
 ϵ : 0.00001
Будет выполнен поиск корня
на интервале от -1.0000 до 1.0000 с точностью 0.00001000
Продолжить?(Y/N)
Y
Корень на промежутке $[-1.0000,1.0000]$ равен -0.00000381.
Нажмите <Enter>...
The text is displayed in a monospaced font on a dark background with light-colored text.

Рисунок 2.8— Результат работы программы поиска корня функции

На рисунке 2.9 приведен результат работы программы при ошибках пользователя для функции $\sin(x)$.

```

Программа находит корень функции  $\sin(x)$  на отрезке  $[a,b]$  с точностью  $\text{eps}$ .
Введите a - вещественное число.
a: 3
Введите b - вещественное число.  $b > a$ .
b: 3
Ошибка!  $b \leq a$ !
Повторить ввод?
1 - ввести a и b;
2 - ввести только a;
3 - ввести только b;
0 - отмена.
Введите цифру от 0 до 3.
3
Введите b - вещественное число.  $b > a$ .
b: 2
Ошибка!  $b \leq a$ !
Повторить ввод?
1 - ввести a и b;
2 - ввести только a;
3 - ввести только b;
0 - отмена.
Введите цифру от 0 до 3.
2
Введите a - вещественное число.
a: 6
Ошибка!  $b \leq a$ !
Повторить ввод?
1 - ввести a и b;
2 - ввести только a;
3 - ввести только b;
0 - отмена.
Введите цифру от 0 до 3.
1
Введите a - вещественное число.
a: 3
Введите b - вещественное число.  $b > a$ .
b: 4
Введите  $\text{eps}$  - вещественное число.  $\text{eps} > 0$ .
eps: 0.0000001
Будет выполнен поиск корня
на интервале от 3.0000 до 4.0000 с точностью 0.00000010
Продолжить?(Y/N)
y
Корень на промежутке  $[3.0000, 4.0000]$  равен 3.14159265.
Нажмите <Enter>...

```

Рисунок 2.9— Результат работы программы при ошибочных значениях

Вывод

В ходе выполнения данной лабораторной работы я научился использовать подпрограммы-функции при написании программ, а также изучил особый алгоритмический приём, при меняющийся при использовании функций - рекурсию. Рекурсия представляет собой мощный инструмент, который позволяет изящно и компактно описать очень сложные для итеративной реализации алгоритмы. Однако минусами рекурсии является повышенный расход памяти и (иногда) меньшая понятность и наглядность. Поэтому если имеется очевидная реализация с помощью циклов, то лучше предпочесть последнее.