Решение задач с использованием циклов

Рассмотрим использование циклических операторов на конкретных примерах.

ЗАДАЧА 1. Найти наибольший общий делитель (НОД) натуральных чисел *A* и *B*.

Исходные данные: *A* и *B*.

Результаты работы программы: *A* – НОД.

Для решения поставленной задачи воспользуемся алгоритмом Евклида: будем уменьшать каждый раз большее из чисел на величину меньшего до тех пор, пока оба значения не станут равными, так, как показано в таблице 1.

Таблица. Поиск НОД для чисел A=25 и B=15.

Исходные данные	**Первый шаг**	**Второй шаг**	**Третий шаг**	**НОД(A,B)=5**
A=25	A=10	A=10	A=5	
B=15	B=15	B=5	B=5	

В блок–схеме, представленной на рис. 1, для решения поставленной задачи используется *цикл с предусловием*, то есть тело цикла повторяется до тех пор, пока *A* не равно *B*.

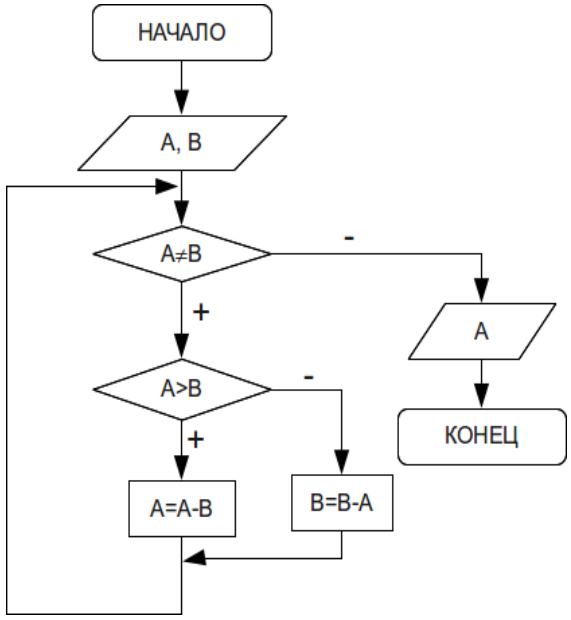

Рис. 1. Поиск наибольшего общего делителя двух чисел

Следовательно, при создании программы воспользуемся циклом `while`:

```
#include <stdio.h>
int main()
{
    unsigned int a,b;
    printf("A="); scanf("%d",&a);
    printf("B="); scanf("%d",&b);
    //Если числа не равны, выполнять тело цикла
    while (a!=b)
    //Если число А больше,
    //чем В, то уменьшить его значение на В,
        if (a>b) a-=b;
    //иначе уменьшить значение числа В на А
        else b-=a;
    printf("НОД=%d\n",a);
return 0;
}
```

Результат работы программы не изменится, если для ее решения воспользоваться циклом с постусловием `do…while`:

```
#include <stdio.h>
int main()
{
    unsigned int a,b;
    printf("A="); scanf("%d",&a);
    printf("B="); scanf("%d",&b);
    //Если числа не равны, выполнять тело цикла
    do
    //Если число А больше,
    //чем В, то уменьшить его значение на В,
        if (a>b) a-=b;
    //иначе уменьшить значение числа В на А
        else b-=a;
    while (a!=b);
    printf("НОД=%d\n",a);
return 0;
}
```

Предлагаем версию программу с оператором `for`.

```
#include <stdio.h>
int main()
{
    unsigned int a,b;
    printf("A=");scanf("%d",&a);
    printf("B=");scanf("%d",&b);
    for(;a!=b;)
        if (a>b) a-=b;
        else b-=a;
        printf("НОД=%d\n",a);
```

```
        return 0;
}
```

ЗАДАЧА 2. Вычислить факториал числа N ($N!=1\cdot2\cdot3\cdot\ldots\cdot N$).

Исходные данные: `N` – целое число, факториал которого необходимо вычислить.

Результаты работы программы: `factorial` – значение факториала числа `N`, произведение чисел от 1 до `N`, целое число.

Промежуточные переменные: `i` – параметр цикла, целочисленная переменная, последовательно принимающая значения 2, 3, 4 до `N`.

Блок-схема приведена на рис. 2.

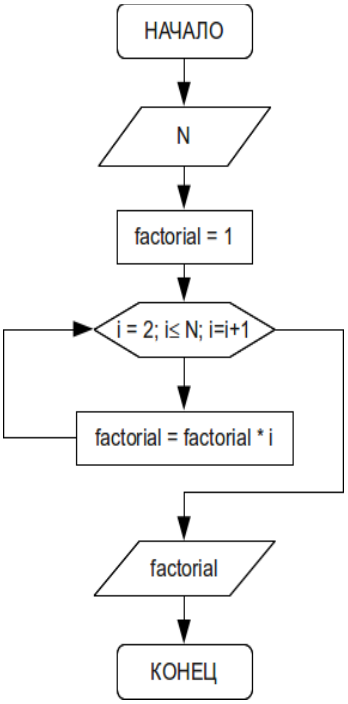

Рис.2 Алгоритм вычисления факториала

Вводится число `N`. Переменной `factorial`, предназначенной для хранения значения произведения последовательности чисел, присваивается начальное значение, равное единице. Затем организуется цикл, параметром которого выступает переменная `i`. Если значение параметра цикла не превышает `N`, то выполняется оператор тела цикла, в котором из участка памяти с именем `factorial` считывается предыдущее значение произведения, умножается на текущее значение параметра цикла, а результат снова помещается в участок памяти с именем `factorial`. Когда параметр `i` превысит `N`, цикл заканчивается, и на экран выводится значение переменой `factorial`, которая была вычислена в теле цикла.

Обратите внимание, как в программы записан *оператор цикла*. Здесь операторы ввода и операторы присваивания стартовых значений записаны

как начальные присваивания цикла for, а оператор накапливания произведения и оператор модификации параметра цикла представляют собой приращение:

```
#include <stdio.h>
int main()
{
    unsigned long long int factorial;
    unsigned int N, i;
for(printf("N="),scanf("%d",&N),factorial=1,i=2;i<=N;
factorial*=i,i++);
printf("факториал=%lld\n",factorial);
return 0;
}
```

Обратите внимание на вывод числа типа long long int (%lld). Можно было использовать и %lli.

ЗАДАЧА 3. Вычислить сумму натуральных чётных чисел, не превышающих *N*.

Исходные данные: N – целое число.

Результаты работы программы: S – сумма четных чисел.

Промежуточные переменные: i – целочисленный параметр цикла, который принимает чётные значения 2, 4, 6, 8 … .

При сложении нескольких чисел необходимо накапливать результат в определённом участке памяти (S), каждый раз считывая из этого участка (S) предыдущее значение суммы (S) и прибавляя к нему слагаемое i. Для выполнения первого оператора накапливания суммы из участка памяти необходимо взять такое число, которое не влияло бы на результат сложения. Перед началом цикла переменной, предназначенной для накапливания сумы, необходимо присвоить значение нуль. Блок-схема решения этой задачи представлена на рис. 3.

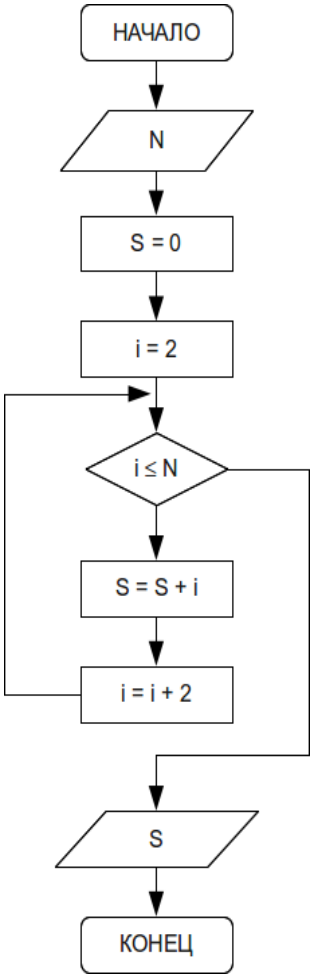

Рис. 3 Алгоритм вычисления суммы четных, натуральных чисел

Решим задачу двумя способами: с применением циклов `while` и `for`:

```
//Решение задачи с помощью цикла while
#include <stdio.h>
int main()
{
    unsigned int N,i,S;
    printf("N="); scanf("%d",&N);
    S=0;
    i=2;
    while (i<=N)
    {
        S+=i;
        i+=2;
    }
    printf("S=%d\n",S);
return 0;
}
```

```
//---------------------------------
//Решение задачи с помощью цикла for
#include <stdio.h>
int main()
{
    unsigned int N,i,S;
    for(printf("N="),
scanf("%d",&N),S=0,i=2;i<=N;S+=i,i+=2);
    printf("S=%d\n",S);
return 0;}
```

ЗАДАЧА 4. Дано натуральное число *N*. Определить *K* – количество делителей этого числа, меньших самого числа (Например, для *N*=12 делители 1, 2, 3, 4, 6. Количество *K*=5).

Исходные данные: `N` – целое число.

Результаты работы программы: целое число `K` – количество делителей `N`.

Промежуточные переменные: `i` – параметр цикла, возможные делители числа `N`.

В блок-схеме, изображенной на рис. 4, реализован следующий алгоритм: в переменную `K`, предназначенную для подсчета количества делителей заданного числа, помещается значение, которое не влияло бы на результат, т.е. нуль. Далее организовывается цикл, в котором изменяющийся параметр `i` выполняет роль возможных делителей числа `N`. Если заданное число (`N`) делится нацело на параметр цикла (`i`), это означает, что `i` является делителем `N`, и значение переменной `K` следует увеличить на единицу. Цикл необходимо повторить `N/2` раз.

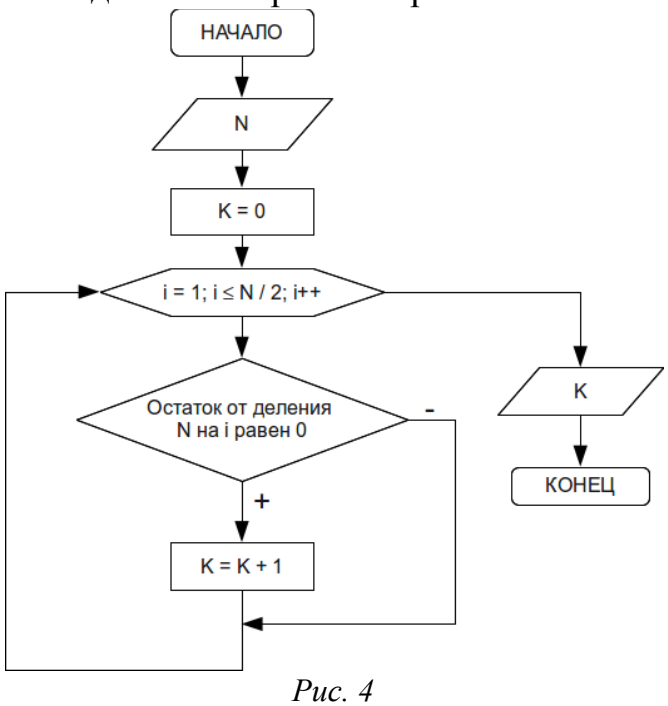

Рис. 4

Текст программы на С:

```
#include <stdio.h>
int main()
{
    unsigned int N,i,K;
    printf("N="); scanf("%d",&N);
    for (K=0,i=1;i<=N/2;i++) if (N%i==0) K++;
    printf("K=%d\n",K);
return 0;
}
```

ЗАДАЧА 5. Дано натуральное число *N*. Определить, является ли оно простым. Натуральное число *N* называется простым, если оно делится без остатка только на единицу и на само себя. Число 13 – простое, так как делится только на 1 и 13, а число 12 таковым не является, так как делится на 1, 2, 3, 4, 6 и 12.

Входные данные: `N` – целое число.

Результаты работы программы: сообщение.

Промежуточные переменные: `i` – параметр цикла, возможные делители числа `N`.

Необходимо проверить есть ли делители числа `N` в диапазоне от 2 до `N/2` (рис. 5).

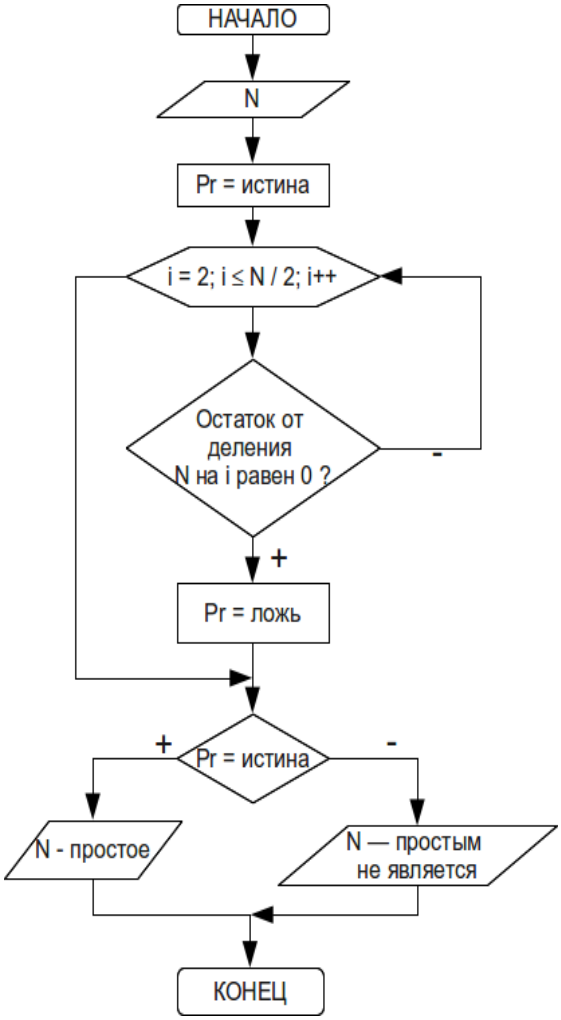

Рис. 5 Алгоритм определения простого числа

Если делителей – нет, `N` – простое число, иначе оно таковым не является. Обратите внимание на то, что в алгоритме предусмотрено два выхода из цикла. Первый – естественный, при исчерпании всех значений параметра, а второй - досрочный. Нет смысла продолжать цикл, если будет найден хотя бы один делитель из указанной области изменения параметра.

При составлении программы на языке С досрочный выход из цикла удобно выполнять при помощи оператора `break`:

```
#include <stdio.h>
int main()
{   unsigned int N,i;  int Pr;
    printf("N="); scanf("%d",&N);
    Pr=1;     //Предположим, что число простое
    for (i=2;i<=N/2;i++)
    if (N%i==0)
//Если найдется хотя бы один делитель, то
    {
        Pr=0;  //число простым не является и
```

```
            break;      //досрочный выход из цикла
        }
        if (Pr)
    //Проверка значения логического параметра и
    //вывод на печать соответствующего сообщения
            printf("%d - простое число \n",N);
        else
            printf("%d - не является простым \n",N);
    return 0;
    }
```

Можно ли переписать основной цикл в программе таким образом.

```
    for (i=2;i<=sqrt(N);i++)
    if (N%i==0) {Pr=0;break;}
```

Ответ обоснуйте.

ЗАДАЧА 6. Дано натуральное число N. Определить количество цифр в числе.

Исходные данные: `N` – целое число.

Результаты работы программы: `kol` – количество цифр в числе.

Промежуточные данные: `M` – переменная для временного хранения значения `N`[1].

Для того, чтобы подсчитать количество цифр в числе, необходимо определить, сколько раз заданное число можно разделить на десять нацело. Например, пусть `N=12345`, тогда количество цифр `kol = 5`. Результаты вычислений сведены в таблицу 2.

Таблица 2. Определение количества цифр числа

kol	N
1	12345
2	12345 / 10 = 1234
3	1234 / 10 = 123
4	123 / 10 = 12
5	12 / 10 = 1
	1 / 10 = 0

Алгоритм определения количества цифр в числе представлен на рис. 6.

1 При решении задачи (см. алгоритм на рис. 4.29) исходное число изменятся, поэтому, чтобы его, не потерять, копируем исходное число `N` в переменную `M`, и делить будем уже `M`.

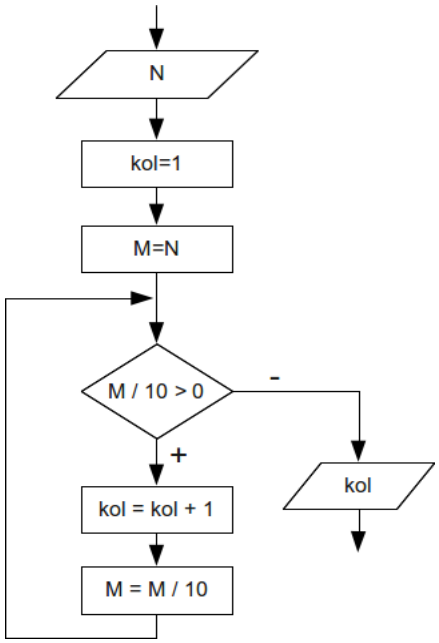

Рис. 6 Алгоритм определения количества цифр в числе

```
#include <stdio.h>
int main()
{
    unsigned long int N, M;
    unsigned int kol;
    printf("N="); scanf("%ld",&N);
    for (M=N, kol=1; M/10>0; kol++,M/=10);
    printf("kol=%d\n",kol);
return 0;
}
```

ЗАДАЧА 7. Дано натуральное число *N*. Определить содержит ли это число нули и в каких разрядах они расположены (например, число 1101111011 содержит ноль в третьем и восьмом разрядах, а число 120405 - во втором и четвертом).

Исходные данные: `N` – целое число.

Результаты работы программы: `pos` – позиция цифры в числе.

Промежуточные данные: `i` – параметр цикла, `M` – переменная для временного хранения значения `N`.

В связи с тем, что разряды в числе выделяются начиная с последнего, то для определения номера разряда в числе, необходимо знать количество цифр в числе[2]. Таким образом, на первом этапе решения задачи необходимо определить `kol` – количество цифр в числе. Затем нужно выделять из числа цифры, если очередная цифра равна нулю, вывести на экран номер разряда, который занимает эта цифра. Процесс определения текущей цифры числа `N=120405` представлен в таблице 3.

Таблица 3. Определение текущей цифры числа

2 Алгоритм нахождения количества цифр в числе был рассмотрен в предыдуще задаче.

i	Число M	Цифра	Номер позиции
1	120405	120405 % 10 = 5	6
2	12040 / 10 = 1204	12040 % 10 = 0	5
3	1204 / 10 = 120	1204 % 10 = 4	4
4	120 / 10 = 12	120 % 10 = 0	3
5	12 / 10 = 1	12 % 10 = 2	2
6	1 / 10 = 0	1 % 10 = 1	1

Программный код к задаче 4.18.

```
#include <stdio.h>
int main()
{
    unsigned long int N,M; int kol, i;
    printf("N="); scanf("%ld",&N);
    //В переменной kol формируем количество позиций в
    // числе
    for (kol=1,M=N;M/10>0; kol++,M/=10);
    //Разряды числа (цифры) получаем в порядке справа
    // налево.
    //Переменная i хранит номер разряда. i изменяется
// от kol до 1.
    for (M=N,i=kol;i>0;M/=10,i--)
        if (M%10==0) printf("Позиция = %d\n",i);
return 0;
}
```

ЗАДАЧА 8. Дано натуральное число *N*. Получить новое число, записав цифры числа *N* в обратном порядке. Например, 12345 – 54321.

Исходные данные: `N` – целое число.

Результаты работы программы: `S` – целое число, полученное из цифр числа `N` , записанных в обратном порядке.

Промежуточные данные: `i` – параметр цикла, `M` – переменная для временного хранения значения `N`, `kol` – количество разрядов в заданном числе, $R= 10^{kol}$ – старший разряд заданного числа.

Рассмотрим пример. Пусть `N=31457`, тогда

$$S=7\cdot10^4+5\cdot10^3+4\cdot10^2+1\cdot10^1+3\cdot10^0=75413$$

Значит, для решения поставленной задачи, нужно знать количество разрядов (`kol`) в заданном числе и и вес его старшего разряда $R=10^{kol-1}$. Новое число `S` формируют как сумму произведений последней цифры заданного числа на её вес `S+=M %10*R`. Цикл выполняют `kol` раз, при каждой итерации уменьшая само число и вес его разряд R в десять раз.

```
#include <stdio.h>
int main()
{unsigned long int N,M,R,S; int kol, i;
printf("N="); scanf("%li",&N);
//В цикле вычисляет количество разрядов в числе
//(переменная kol), параллельно с этим накапливается вес
```

```
// старшего разряда (переменная R).
for (R=1,kol=1,M=N;M/10>0; kol++,R*=10,M/=10);
//В цикле формируется переменная S -  число в обратном
// порядке.
for(S=0,M=N,i=1;i<=kol;S+=M%10*R,M/=10,R/=10,i++);
printf("S=%li\n",S);
return 0;
}
```

ЗАДАЧА 9. Проверить является ли заданное число N палиндромом[3]. Например, числа 404, 1221 — палиндромы.

Исходные данные: N – целое число.

Результаты работы программы: сообщение.

Промежуточные данные: i – параметр цикла, M – переменная для временного хранения значения N, kol – количество разрядов в заданном числе, R= 10^{kol} – старший разряд заданного числа, S – целое число, полученное из цифр числа N , записанных в обратном порядке.

Можно предложить следующий алгоритм решения задачи. Записать цифры заданного числа N в обратном порядке (задача 8), получится новое число S. Сравнить полученное число S с исходным N. Если числа равны, то заданное число является палиндромом.

Текст программы на языке С:

```
#include <stdio.h>
int main()
{unsigned long int N,M,R,S;
int kol, i;
printf("N="); scanf("%ld",&N);
//В цикле вычисляет количество разрядов в числе
//(переменная kol), параллельно с этим накапливается вес
// старшего разряда (переменная R).
for (R=1,kol=1,M=N;M/10>0; kol++,R*=10,M/=10);
//В цикле формируется переменная S -  число в обратном
// порядке.
for(S=0,M=N,i=1;i<=kol;S+=M%10*R,M/=10,R/=10,i++);
//Сравнение исходного числа с перевёрнутым.
if (N==S) printf("Число - палинром\n");
else printf("Число не является палиндромом\n");
return 0;
}
```

Предлагается ещё один алгоритм проверки является ли число палиндромом. Сравниваем левую и правую цифры числа, если встречаем несовпадение – исходное число не является палиндромом.

Код программы с минимальными комментариями приведён ниже. Читателю предлагается самостоятельно изучить программу.

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

3 Палиндром — это число, слово или фраза одинаково читающееся в обоих направлениях, или, другими словами, любой симметричный относительно своей середины набор символов.

```
int main()
{unsigned long int N,M,R;
int kol, i, pr, lev, prav;
printf("N="); scanf("%ld",&N);
for (R=1,kol=1,M=N;M/10>0; kol++,R*=10,M/=10);
//Переменная pr отвечает за совпадение левой и правой
цифры числа
for(M=N,pr=i=1;i<=kol;i+=2)
{
//В переменной prav хранится текущая правая цифра
числа.
    prav=M%10;
//В переменной lev хранится текущая правая цифра
числа.
    lev=M/R;
    if (lev!=prav) {pr=0;break;}
//Отрезаем левую цифру числа
    M-=(M/R)*R;
//Отрезаем правую цифру числа
    M/=10;
//Вес левого разряда уменьшаем в 100 раз
    R/=100;
}
if (pr) printf("Число - палинром\n");
else printf("Число не является палиндромом\n");
return 0;
}
```

```
Ещё одна версия программы
#include <stdio.h>
int main()
{unsigned long int N,K,S;
printf("N="); scanf("%ld",&N);
for(K=N,S=0;N!=0;N/=10)
    S=S*10+N%10;
if (K==S) printf("Число - палинром\n");
else printf("Число не является палиндромом\n");
return 0;
}
```

ЗАДАЧА 10. Вводится последовательность из N чисел. Найти сумму чисел.

Исходные данные: целое число `N`, `N` значений элементов последовательности.

Результаты работы программы: `S` – сумма чисел.

Промежуточные данные: `i` – параметр цикла.

Для ввода последовательности чисел нам понадобятся следующие переменные: N – количество элементов в последовательности чисел; X – текущее значение элемента последовательности; i – переменная цикла, определяющая номер элемента последовательности.

Вначале определяем N – количество элементов последовательности. Затем на каждом шаге цикла вводится элемент последовательности X и происходит его обработка.

В нашей задаче необходимо найти сумму введённых чисел. Алгоритм нахождения суммы стандартен: до цикла значение суммы равно 0 (S=0), на каждом шаге цикла – вводим очередное значение элемента последовательности X и увеличиваем значение суммы на X (S+=X).

```
#include <stdio.h>
int main(int argc, char **argv)
{
    int i,N;
    float S,X;
    printf("N=");scanf("%d",&N);
    S=0;
    for(i=1;i<=N;i++)
    {
        printf("X=");scanf("%f",&X);
        S+=X;
    }
    printf("S=%8.3f\n",S);
    return 0;
}
```

ЗАДАЧА 11. Вводится последовательность целых чисел. 0 – конец последовательности. Найти произведение чисел.

Исходные данные:X – текущее значение элемента последовательности.

Результаты работы программы: P – произведение чисел.

В этой задаче принципиальным отличием является организация ввода последовательности чисел. Заранее неизвестно сколько будет введено чисел, после ввода очередного числа необходимо проверять не введено ли число 0. Если введён ноль, то программа заканчивается, иначе идёт обработка очередного числа.

Цикл осуществляющий ввод данных представлен ниже.

```
printf("X=");scanf("%f",&X);
for(;X!=0;)
{
//Здесь должна быть обработка числа X
printf("X=");scanf("%f",&X);
}
```

В нашей задаче необходимо найти произведение введённых чисел.

Алгоритм нахождения произведения стандартен: до цикла значение произведения равно 1 (P=1), на каждом шаге цикла – вводим очередное значение элемента последовательности X и умножаем значение произведения на X (P*=X). Код программы решения задачи 11 приведён ниже.

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int P,X;
    printf("X=");scanf("%d",&X);
    for(P=1;X!=0;)
    {
        P*=X;
        printf("X=");scanf("%d",&X);

    }
    printf("P=%d\n",P);
    return 0;
}
```

ЗАДАЧА 12. Поступает последовательность из N вещественных чисел. Определить наибольший элемент последовательности.

Исходные данные: N – целое число; X – вещественное число, определяет текущий элемент последовательности.

Результаты работы программы: Max – вещественное число, элемент последовательности с наибольшим значением.

Промежуточные данные: i – параметр цикла, номер вводимого элемента последовательности.

Алгоритм поиска наибольшего элемента в последовательности следующий (рис. 7). Вводится N – количество элементов последовательности и X – первый элемент последовательности. В памяти компьютера отводится ячейка, например с именем Max, в которой будет храниться наибольший элемент последовательности – максимум. Далее предполагаем, что первый элемент последовательности наибольший и записываем его в Max. Затем вводится второй элемент последовательности и сравниваем его с предполагаемым максимумом. Если окажется, что второй элемент больше, его записывают в ячейку Max. В противном случае никаких действий не предпринимаем. Потом переходим к вводу следующего элемента последовательности (X), и алгоритм повторяется с начала. В результате в ячейке Max сохраниться элемент последовательности с наибольшим значением[4].

4 Для поиска наименьшего элемента последовательности (минимума), предполагают, что первый элемент – наименьший, записывают его в ячейку min, а затем среди элементов последовательности ищут число, значение которого будет меньше чем предполагаемый минимум

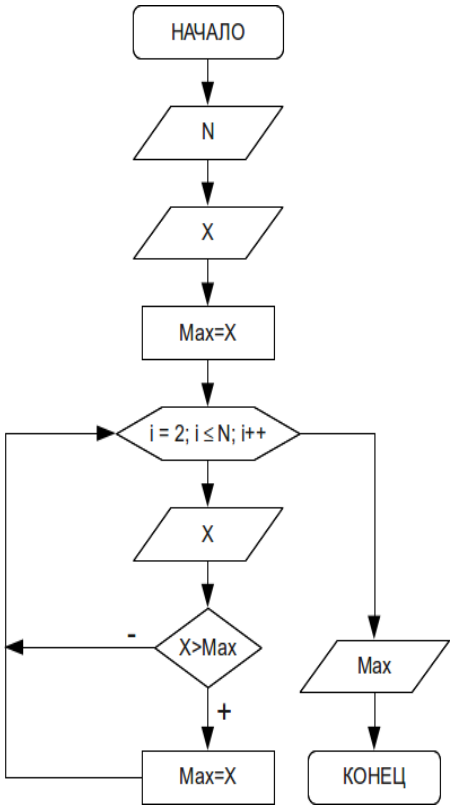

Рис. 7 Алгоритм поиска наибольшего числа в последовательности

Текст программы на С:

```
#include <stdio.h>
int main()
{
    unsigned int i,N;
    float X,Max;
    printf("N="); scanf("%d",&N);
    printf("X=");scanf("%f",&X);//Ввод первого
элемента
//последовательности. Параметр цикла принимает
стартовое
// значение i=2, т.к. первый элемент уже введен
// предположим, что он максимальный, Max=X.
    for (i=2,Max=X;i<=N;i++)
    {
//Ввод следующих элементов последовательности.
       printf("X=");scanf("%f",&X);
//Если найдется элемент превышающий максимум,
//записать его в ячейку Max, теперь он предполагаемый
//максимум.
       if (X>Max) Max=X;   }
//Вывод наибольшего элемента последовательности.
    printf("Max=%8.3f\n",Max);
```

```
return 0;}
```

ЗАДАЧА 13. Вводится последовательность целых чисел, 0 – конец последовательности. Найти наименьшее число среди положительных, если таких значений несколько[5], определить, сколько их.

Исходные данные: `N` – текущий текущий элемент последовательности.

Результаты работы программы: `Min` – минимальный положительный элемент последовательности, `K` – количество значений равных минимуму.

Блок-схема решения задачи приведена на рис. 8.

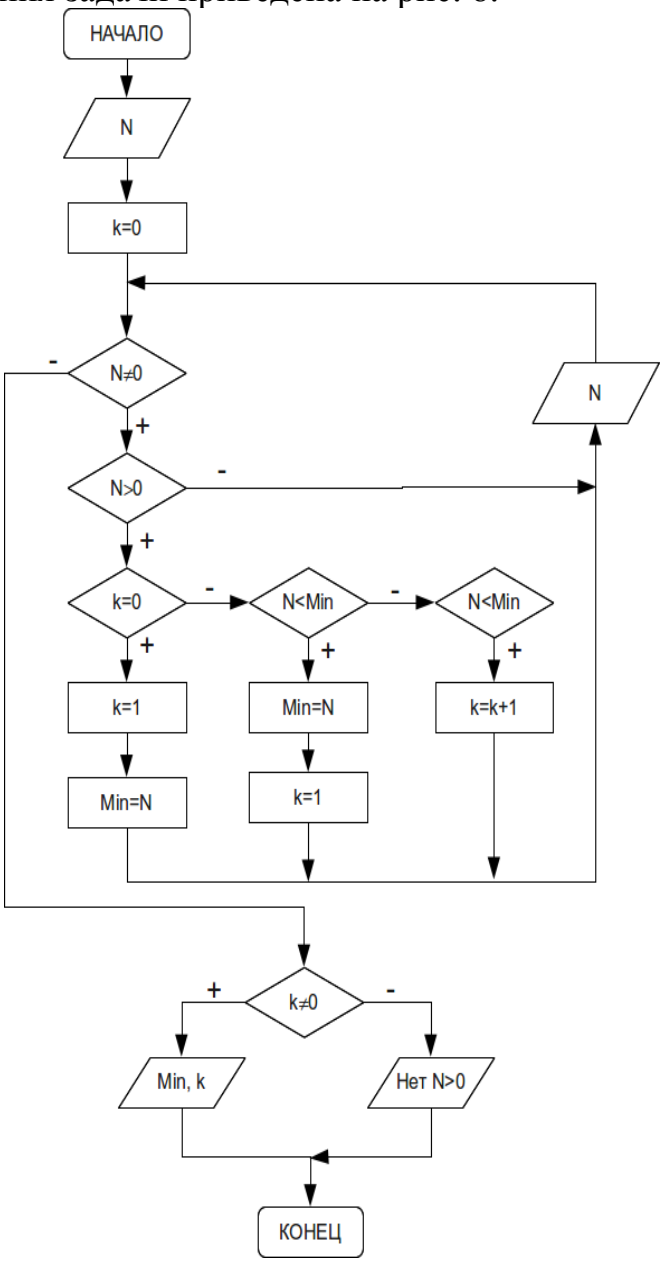

Рис. 8 Алгоритм поиска минимального положительного числа в последовательности

5 Предположим вводится последовательность чисел 11, -3, 5, 12, -7, 5, 8,-9, 7, -6, 10, 5,0. Наименьшим положительным числом является 5. Таких минимумов в последовательности 3.

Далее приведен текст подпрограммы с подробными комментариями[6].

```
#include <stdio.h>
int main()
{
    int N,Min,K;
//Предположим, что в последовательности нет
положительных
//чисел, K=0. Переменная K содержит количество
значений
//равных минимуму. Вводим число N и если оно не равно
нулю
printf("N=");scanf("%d",&N);
for (K=0;N!=0;)
{
//проверяем является ли оно положительным.
       if (N>0){
//если K=0, поступил 1-й положительный элемент,
// который и объявляем минимальным.
         if (K==0) {K=1;Min=N;}
//если текущее положительное число элемент не первое
// сравниваем его с предполагаемым минимумом,
//если текущее число меньше, записываем его в Min
//и сбрасываем счетчик K в единицу
         else if (N<Min) {Min=N;K=1;}
//если положительный элемент равен минимуму,
// увеличиваем значение счетчика K.
         else if (N==Min) K++;
       }
       printf("N=");scanf("%d",&N);
}
//Конец цикла
//Если значение счетчика не равно нулю,
// выводим значение минимального элемента
//и количество таких элементов.
    if (K!=0) printf("Min=%d\tK=%d\n",Min,K);
//в противном случае сообщаем об отсутствии
// положительных элеменнтов.
else printf("Нет положительных элементов \n");
return 0;
}
```

ЗАДАЧА 14. Определить сколько раз последовательность из *N* произвольных чисел меняет знак.

Исходные данные: `N` – количество элементов последовательности; `B` определяет текущий элемент последовательности.

Результаты работы программы: k – количество смен знака в последо-

6 Алгоритм поиска максимального (минимального) элементов последовательности подробно описан в задаче 4.23

вательности.

Промежуточные данные: `i` – параметр цикла, номер вводимого элемента последовательности, `A` – значение предыдущего элемента последовательности.

Чтобы решить задачу нужно попарно перемножать элементы последовательности. Если результат произведения пары чисел – отрицательное число, значит, эти числа имеют разные знаки.

Пусть в переменной `B` хранится текущий элемент последовательности, в `A` – предыдущий. Введём первое число `A` (до цикла) и второе `B` (в цикле). Если их произведение отрицательно, то увеличиваем количество смен знака на 1 (`k++`). После чего сохраняем значение `B` в переменную `A` и повторяем цикл (рис. 9).

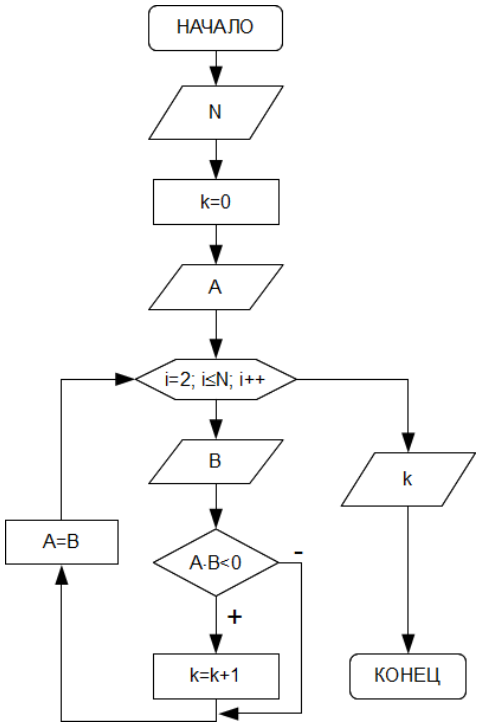

Рис. 9. Алгоритм решения задачи 14.

Предлагается самостоятельно разобраться с текстом программы на С:

```
#include <stdio.h>
int main()
{
    float A,B; int i,K,N;
    printf("N=");scanf("%d",&N);
    printf("A=");scanf("%f",&A);
    for (K=0,i=2;i<=N;i++)
    {
        printf("B=");scanf("%f",&B);
        if (A*B<0) K++;
        A=B;
```

```
    }
    printf("K=%d\n",K);
return 0;
}
```

ЗАДАЧА 15. Поступает последовательность из N целых чисел. Определить количество простых чисел в последовательности.

Исходные данные: `N` – количество элементов последовательности; `X` (целое число) определяет текущий элемент последовательности.

Результаты работы программы: `Pr` определяет является ли число простым, `k` – количество простых чисел.

Промежуточные данные: `i` – параметр цикла, номер вводимого элемента последовательности, `j` – возможные значения делителей числа `X`.

Для решения задачи было организовано два цикла. Первый цикл обеспечивает ввод элементов последовательности. Второй цикл, находится внутри первого и определяет, является ли поступившее число простым.

```
#include <stdio.h>
int main()
{
    unsigned long int X;
    unsigned int N;
    int i,k,j,Pr;
//Вводим количество чисел последовательности.
    printf("N=");scanf("%d",&N);
    for (k=0,i=1;i<=N;i++)
    {
//Вводим очередное число X последовательности.
       printf("X=");scanf("%ld",&X);
//Внутренний цикл для проверки, является ли число
число X
// простым.
       for (Pr=1,j=2;j<=X/2;j++)
//Если находится нетривиальный делитель,
             if (X%j==0)
             {
//то признак простоты числа, сбрасываем в 0, и
аварийно
// покидаем внутренний цикл.
                  Pr=0;
                  break;
             }
//Если число простое, счётчик количества простых чисел
//увеличиваем на 1.
       if (Pr) k++;
    }
//Вывод результатов.
```

```
    if (k==0)
//Сообщение об отсутствии простых чисел.
       printf("Нет простых чисел\n");
//Вывод количества простых чисел.
    else printf("Количество простых чисел равно %d\
n",k);
return 0;
}
```

ЗАДАЧА 16. Поступает последовательность из N вещественных чисел. Определить является ли она возрастающей.

Исходные данные: `N` (целое число) – количество элементов последовательности; `B` (вещественное число) определяет текущий элемент последовательности.

Результаты работы программы: `Pr` (целое число) определяет является ли последовательность возрастающей.

Промежуточные данные: `i` – параметр цикла, номер вводимого элемента последовательности, `A` – значение предыдущего элемента последовательности.

В возрастающей последовательности каждое число больше предыдущего. Если встретится число, которое окажется меньше или равно предыдущего, то последовательность окажется невозрастающей.

Алгоритм решения задачи следующий.

Вводится N – количество чисел в последовательности. Предполагаем, что последовательность возрастающая (Pr=1). Далее организуется цикл, который повторяется N раз (переменная цикла i меняется от 1 до N с шагом 1). На каждой итерации выполняются следующие действия.

1. В переменную B вводится очередной элемент последовательности.
2. На всех итераций (кроме первой) сравниваем текущее значение последовательности (B) с предыдущим (A). Если $B \leq A$, то наша последовательность невозрастающая (в переменную Pr записываем 0).
3. В переменную A (предыдущее значение элемента последовательности) сохраняем значение B (текущее значение элемента последовательности).

После выхода из цикла анализируем значение Pr. Если Pr=1, последовательность возрастающая; иначе последовательность не является возрастающей.

Текст программы с комментариями приведён ниже.

```
#include <stdio.h>
int main()
{
    int i,N,Pr;
    float B,A;
//Ввод N – количества элементов последовательности.
    printf("N=");scanf("%d",&N);
```

```
//Цикл, который повторяется N раз.
//Предполагаем, что последовательность возрастающая.
    for(Pr=i=1;i<=N;i++)
    {
//Ввод очередного элемента последовательности
       printf("B=");scanf("%f",&B);
//Начиная со второго элемента
       if (i!=1)
//Сравниваем текущий элемент (B) с предыдущим (A).
//Если текущий элемент меньше или равен предыдущего,
то
//последовательность не является возрастающей (Pr=0).
       if (B<=A) Pr=0;
//В переменную A сохраняем текущее значение элемента
// последовательности (B).
       A=B;
    }
//Проверка является ли последовательность возрастающей
// (по значению переменной Pr)
    if (Pr) printf("Последовательность возрастающая\
n");
    else
printf("Последовательность не является возрастающей\
n");
    return 0;
}
```

ЗАДАЧА 17. Задано число N в десятеричной системе. Перевести число в восьмеричную систему счисления.

Исходные данные: N – исходное целое число.

Результаты работы программы: S – число в восьмеричной системе.

Промежуточные данные: T – очередная цифра числа, R – вес цифры числа.

Алгоритм перевода следующий в восьмеричную систему следующий.

1. Определить очередную цифру числа (T) в восьмеричной системе счисления, вычислив остаток от деления числа N на 8.
2. Уменьшить число N в 8 раз.
3. Если N>0, то повторяем пункты 1-3. Иначе мы определили все разряды числа.

Чтобы реализовать в виде программы алгоритм, представленный пунктами 1-3, необходимо собрать число из получаемых в п.1 цифр. Для этого надо просуммировать все цифры числа, умноженные на их вес. Вес первой (младшей) числа равен 1 (10^0), второй – 10 (10^1), третьей – 100 (10^2) и т д.

На рис. 10 Приведен пример перевода числа 256 заданного в десятич-

ной системе счисления в восьмеричную. В результате получим $256_{(10)}=400_{(8)}$.

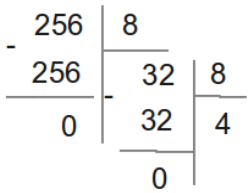

Рис. 4 Пример перевода числа в новую систему счисления

Формальный алгоритм можно записать так.

1. Ввод числа `N`. Копируем значение `N` в переменную `M`. Определяем вес младшего разряда `R=1`. Число в восьмеричной системе системе будет представлять из себя сумму цифр умноженных на их вес (степени числа 10). Поэтому в переменную `S` запишем число `0`.
2. Пока M>0, повторяем пункты 3-4.
3. Определяем очередную цифру числа `T=M%8`. **Накапливаем число S в восьмеричной системе счисления** `S+=T*R`. **Увеличиваем вес следующего разряда числа** `R*=10`.
4. Уменьшить число M в 8 раз.
5. Вывод S. В S хранится число в восьмеричной системе счисления.

Рассмотрим текст программы с комментариями.

```
#include <stdio.h>
int main(int argc, char **argv)
{
    long int N,S,R,M;
    int T;
//Ввод исходного числа N.
    printf("N=");scanf("%ld",&N);
//Сохраняем копию исходного числа в переменной M.
    M=N;
//В переменной R храним вес очередного разряда числа.
//Первоначальное значение R равно 1.
//S=0, в переменной S будет собираться число в
// восьмеричной системе счисления.
    for(R=1,S=0;M>0;M/=8)
    {
//Определяем T – очередную цифру числа.
        T=M%8;
//К переменной S добавляем цифру, умноженную на вес.
        S+=T*R;
//Вес следующей цифры увеличиваем в 10 раз.
        R*=10;
    }
//Вывод числа в восьмеричной системе счисления.
```

```
    printf("S=%ld\n",S);
    return 0;
}
```

Программа и алгоритм мало изменятся при переводе из десятеричной в р-чную $2 \le p \le 9$ систему счисления. Необходимо в коде просто заменить цифру 8 заменить на основание другой системы счисления. Текст программы перевода числа из десятеричной в р-чную систему счисления приведён ниже без комментариев.

```
#include <stdio.h>
int main(int argc, char **argv)
{
    long int N,S,R,M;
    int T,p;
    printf("N=");scanf("%ld",&N);
    printf("p=");scanf("%d",&p);
    M=N;
    for(R=1,S=0;M>0;M/=p)
    {
        T=M%p;
        S+=T*R;
        R*=10;
    }
    printf("S=%ld\n",S);
    return 0;
}
```

ЗАДАЧА 18. Задано число N в восьмеричной системе счисления. Перевести число в десятеричную систему счисления.

Исходные данные: `N` – целое в восьмеричной системе сисчисления.

Результаты работы программы: `S` – число в десятеричной системе.

Промежуточные данные: T – очередная цифра числа, R – вес цифры числа.

Алгоритм перевода из восьмеричной в десятеричную.

1. Ввод числа `N`. Копируем значение `N` в переменную `M`. Определяем вес младшего разряда `R=1`. Число в десятеричной системе системе будет представлять из себя сумму цифр умноженных на их вес (степени числа 8). В переменную `S` запишем число `0`.
2. Пока M>0, повторяем пункты 3-4.
3. Определяем очередную цифру числа `T=M%10`. **Накапливаем число S в восьмеричной системе счисления** `S+=T*R`. **Увеличиваем вес следующего разряда числа** `R*=8`.
4. Уменьшить число M в 10 раз.
5. Вывод S. В S хранится число в десятеричной системе счисления.

Код программы приведён ниже.

```
#include <stdio.h>
int main(int argc, char **argv)
{
    long int N,S,R,M;
    int T;
//Ввод исходного числа N.
    printf("N=");scanf("%ld",&N);
//Сохраняем копию исходного числа в переменной M.
    M=N;
//В переменной R храним вес очередного разряда числа.
//Первоначальное значение R равно 1.
//S=0, в переменной S будет собираться число в
// десятеричной системе счисления.
    for(R=1,S=0;M>0;M/=10)
    {
//Определяем T – очередную цифру числа.
        T=M%10;
//К переменной S добавляем цифру, умноженную на вес.
        S+=T*R;
//Вес следующей цифры увеличиваем в 8 раз.
        R*=8;
    }
//Вывод числа в десятеричной системе счисления.
    printf("S=%ld\n",S);
    return 0;
}
```

Ниже приведён текст программы перевода числа из p-чной ($2 \le p \le 9$) системы счисления в десятеричную без комментариев.

```
#include <stdio.h>
int main(int argc, char **argv)
{
    long int N,S,R,M;
    int T,p;
    printf("N=");scanf("%ld",&N);
    printf("p=");scanf("%d",&p);
    M=N;
    for(R=1,S=0;M>0;M/=10)
    {
        T=M%10;
        S+=T*R;
        R*=p;
    }
    printf("S=%ld\n",S);
    return 0;
}
```