

Информатика (основы программирования)

Лекция 3.

Алгоритмы обработки чисел различных форматов

Автор: Бабалова И.Ф.

Доцент, каф.12

Ряд Фибоначчи

- Рассмотрим вектор F и матрицу V :
- $F = \begin{vmatrix} 1 \\ 1 \end{vmatrix}$ $V = \begin{vmatrix} 0 & 1 \\ 1 & 1 \end{vmatrix} \rightarrow$ Умножив матрицу на вектор, получим следующее число Фибоначчи.
- Если вектор F равен $(1; 1)$, то умножив матрицу V на вектор F , получим новый вектор
- $\begin{vmatrix} 0 & 1 \\ 1 & 1 \end{vmatrix} \cdot \begin{vmatrix} 1 \\ 1 \end{vmatrix} = \begin{vmatrix} 0 \cdot 1 + 1 \cdot 1 \\ 1 \cdot 1 + 1 \cdot 1 \end{vmatrix} = \begin{vmatrix} 1 \\ 2 \end{vmatrix}$
- $\begin{vmatrix} 0 & 1 \\ 1 & 1 \end{vmatrix} \cdot \begin{vmatrix} 1 \\ 2 \end{vmatrix} = \begin{vmatrix} 0 \cdot 1 + 1 \cdot 2 \\ 1 \cdot 1 + 1 \cdot 2 \end{vmatrix} = \begin{vmatrix} 2 \\ 3 \end{vmatrix}$
- Таким образом, делаем вывод, что не надо выполнять n сложений для вычисления n -ого числа Фибоначчи, а можно возвести матрицу V в n -ю степень. Это чисто математическое исследование существенно уменьшает количество вычислений, и сложность алгоритма уменьшается практически в n раз.

Алгоритмизация вычисления рядов

■ Теорема:

- В Для сходящегося ряда к сумме S точность вычислений не будет превышать ε при выполнении условия:

$$|S - S_n| \leq \varepsilon, \quad \text{если} \rightarrow |S_n - S_{n-1}| \leq \varepsilon$$

Имеем $S_1, S_2, \dots, S_{n-1}, S_n$ - приближения к результату

Метод вычисления рядов и многочленов - метод итераций

Этот метод ещё носит название – **метод последовательных приближений**

Последовательность разработки алгоритма для получения результата с заданной точностью при разложении функции в числовой ряд

$$\pi = 3 + 4 \cdot \left(\frac{1}{2 \cdot 3 \cdot 4} - \frac{1}{4 \cdot 5 \cdot 6} + \frac{1}{6 \cdot 7 \cdot 8} - \dots \right)$$

1. Анализ формулы для вычисления числа π Функция в виде числового ряда, каждый член которого отличается от предыдущего члена знаком и делителем. Ряд знакопеременный
2. Записать инвариант цикла, обеспечивающий решение задачи для произвольного числа слагаемых: $S = 0$, $I = 0$, $j = 2$, где S – начальная сумма, i – номер приближения, j – начальное значение делителей.
3. Формула для вычисления каждого слагаемого S_i

$$\pi = (-1)^n \cdot \left(\frac{1}{j \cdot (j+1) \cdot (j+2)} \right) \text{ или } (-1)^i / j / (j+1) / (j+2)$$

4. Определение условия окончания решения: $\text{Abs}(S-S1) \leq \varepsilon$, где ε – заданная погрешность решения задачи

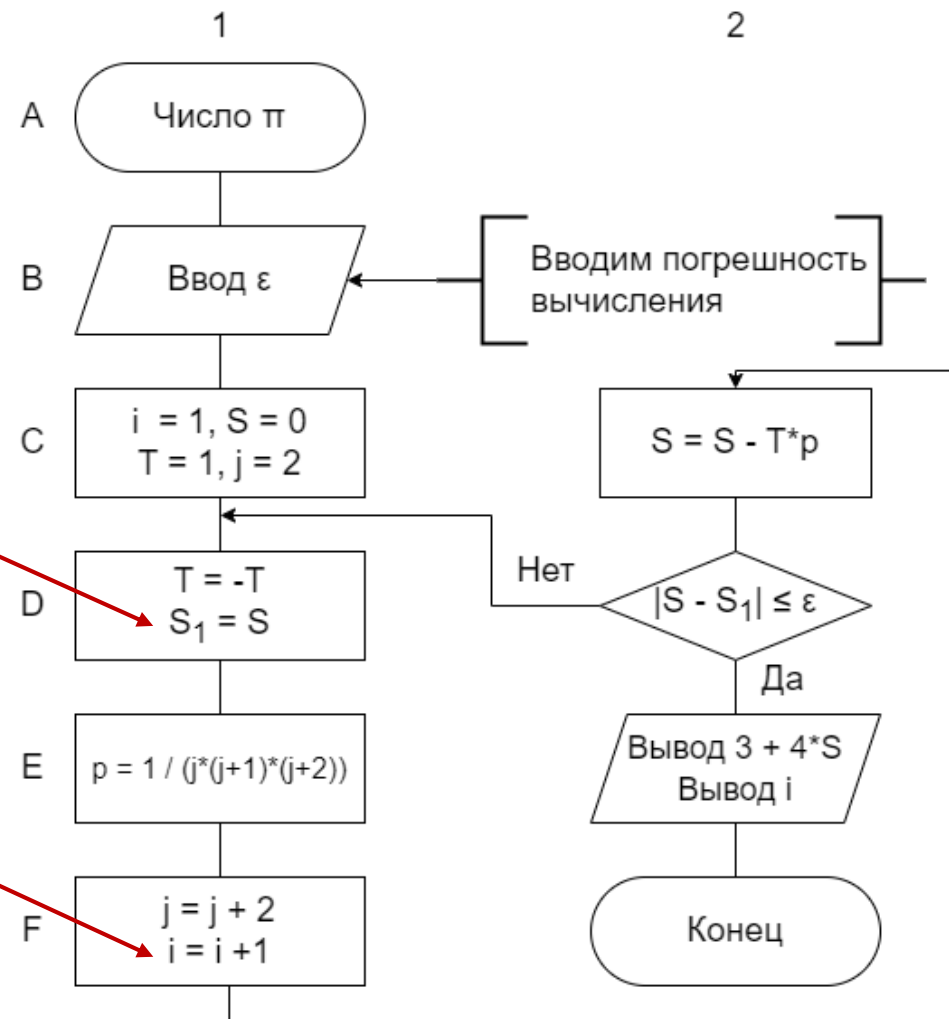
Алгоритм вычисления числа π с помощью ряда

$$\pi = 3 + 4 \cdot \left(\frac{1}{2 \cdot 3 \cdot 4} - \frac{1}{4 \cdot 5 \cdot 6} + \frac{1}{6 \cdot 7 \cdot 8} - \dots \right)$$

Сохранение предыдущей суммы

Подсчет количества итераций

Решение должно быть выполнено
с заданной точностью

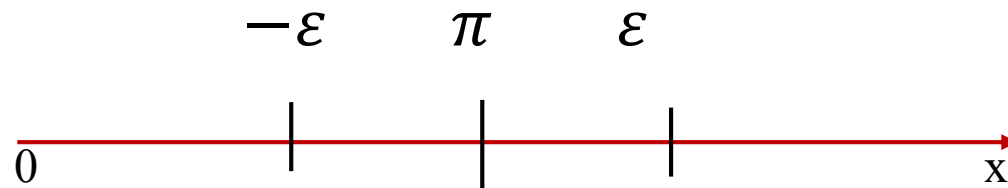


Алгоритмизация вычисления рядов

- Запись ответа с семью верными цифрами:

- $\pi = 3,141593 \pm 0.5 \cdot 10^{-6}$ - точное значение

- Числовая ось



Самое точное значение числа π :

3,1415926535897932384626433832795

Проверьте результат.

- Известный на сегодня результат вычисления числа π с 13,3 триллионов знаков после запятой.

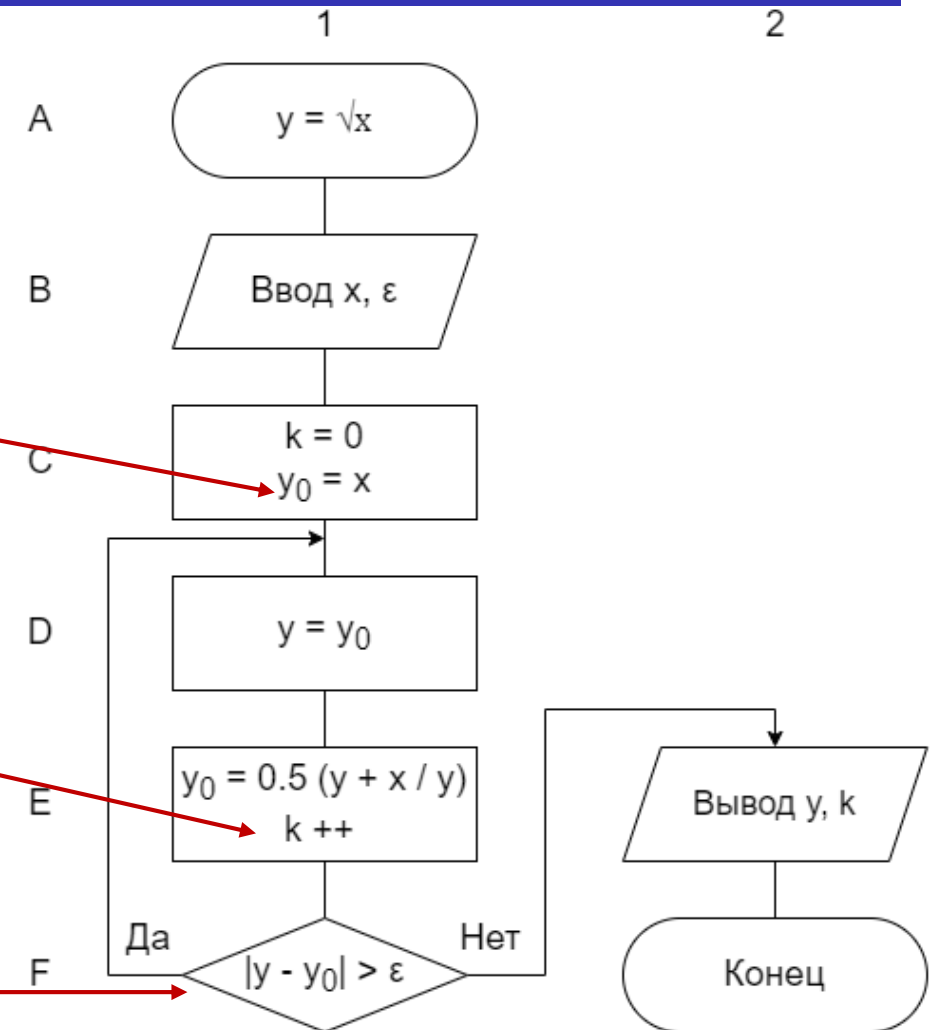
Итерационный алгоритм вычисления коря квадратного из числа x

$$y = \sqrt{x} \quad y_{i+1} = \frac{1}{2} \left(y_i + \frac{x}{y_i} \right)$$

Начальное значение корня из x равно x

Считаем количество итераций для достижения заданной точности

Цикл do ... while



Программа вычисления корня квадратного числа x

```
#include <stdio.h>
#include <math.h>
int main () {
    double y, x, eps,
    double y0 = 0;
    int k;
    printf("Input number x and eps\n");
    scanf("%lf",&x);
    scanf("%lf",&eps);
    y0 = x;
    k = 0;
    do {
        y = y0;
        k++;
        y0 = 0.5 * (y + x / y);
    } while(fabs(y - y0) > eps);
    printf("Result = %.15lf, count = %d\n", y, k);
    return 0;
}
```

Оценка точности результата вычислений:

$\Delta = 0.5 \cdot 10^{m-n+1}$, где m – это наивысшая степень основания системы счисления в записи числа, а n – количество значащих цифр в рассматриваемом числе.

Результаты для разных погрешностей

Input number x and eps

2 0.01

Result = 1.416666666666667, count = 3

Input number x and eps

2 0.00001

Result = 1.414215686274510, count = 4

Input number x and eps

2 0.00000001

Result = 1.414213562374690, count = 5

Запись результата:

$$n = 5 \quad \Delta = 0.5 \cdot 10^{0-5+1} = 0.5 \cdot 10^{-4}$$

$$\text{Result} = 1.4142 \quad 0.1 \cdot 10^{-4} < 0.5 \cdot 10^{-4}$$

Производные структуры

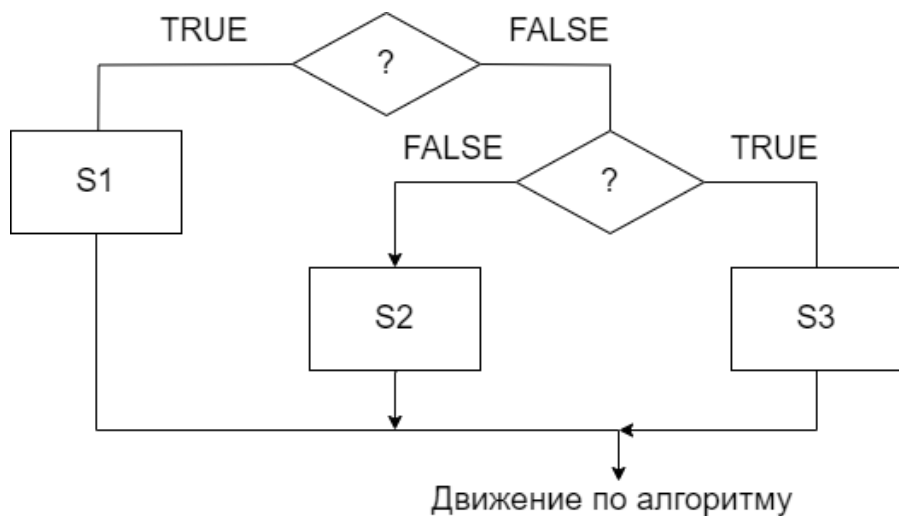
Производные структуры, соответствующие основным алгоритмическим действиям:

1. Ветвление – проверка условия и выбор направления движения по алгоритму
2. Цикл до получения истинного значения условия
3. Счётный цикл с известным (заданным) числом его повторений

Примеры реализации производных структур

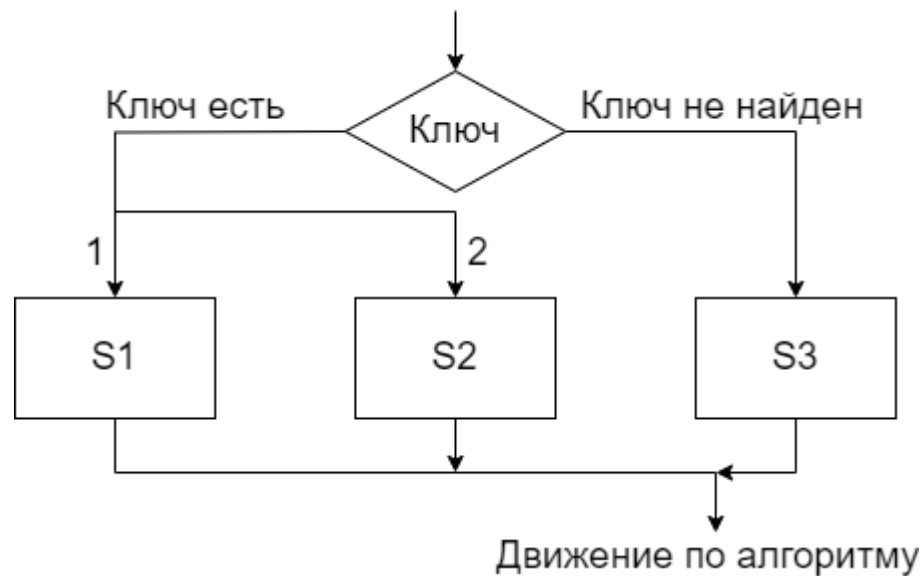
Ветвление

```
if <условие>  
{S1}  
...  
else  
{S3}
```



Выбор

```
switch <условие>  
case 1: S1; break;  
case 2: S2; break;  
default: S3;
```



Пример использования оператора case

```
char sign;
int x, y, z;
switch(sign) {
    case '+':
        x = y + z; break;
    case '-':
        x = y - z; break;
    case '*':
        x = y * z; break;
    case '/':
        x = y / z; break;
    default: ; // unknown sign
}
```

- Использование **оператора break** позволяет в необходимый момент прервать последовательность выполняемых операторов в теле оператора **switch**, путем передачи управления оператору, следующему за **switch**.

Циклы

Цикл «Пока»

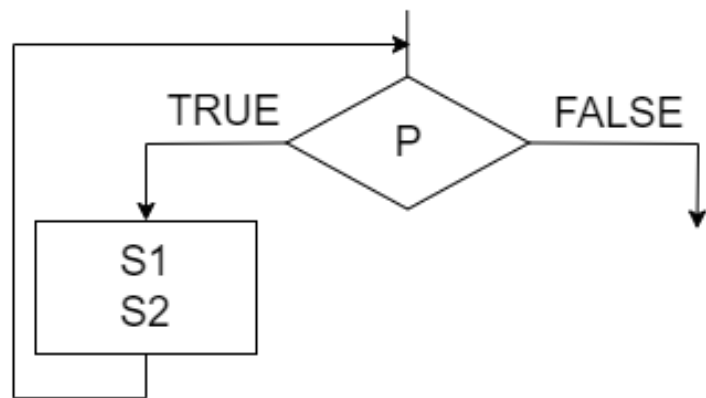
while P – выполняется одно действие

```
while (P) {
```

```
    S1; // Операторы работают в цикле
```

```
    S2; // любое число раз
```

```
}
```



Цикл «До»

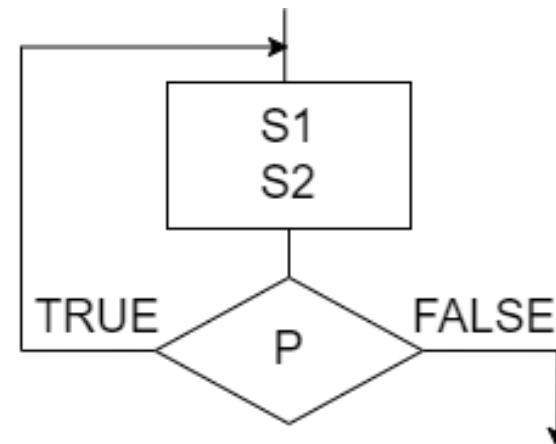
```
do {
```

```
    S1;
```

```
    S2;
```

```
} while (P);
```

// Особенность – сначала выполняется действие, а потом выполняется проверка цикла (пока истина)



Циклы (2)

Цикл с параметром

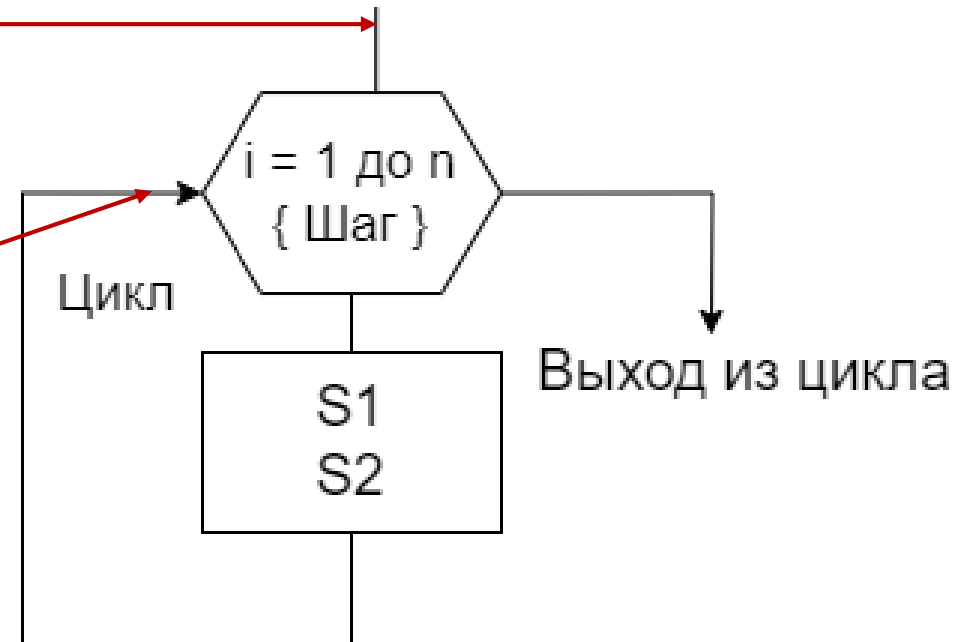
```
for (<выражение>; <условие>; <выражение>)  
{  
    S1;  
    S2;  
    ...  
}
```

Вход в начало цикла

Вход для продолжения цикла

Этот оператор при известном количестве повторений цикла является дополнением операторов while и do-while.

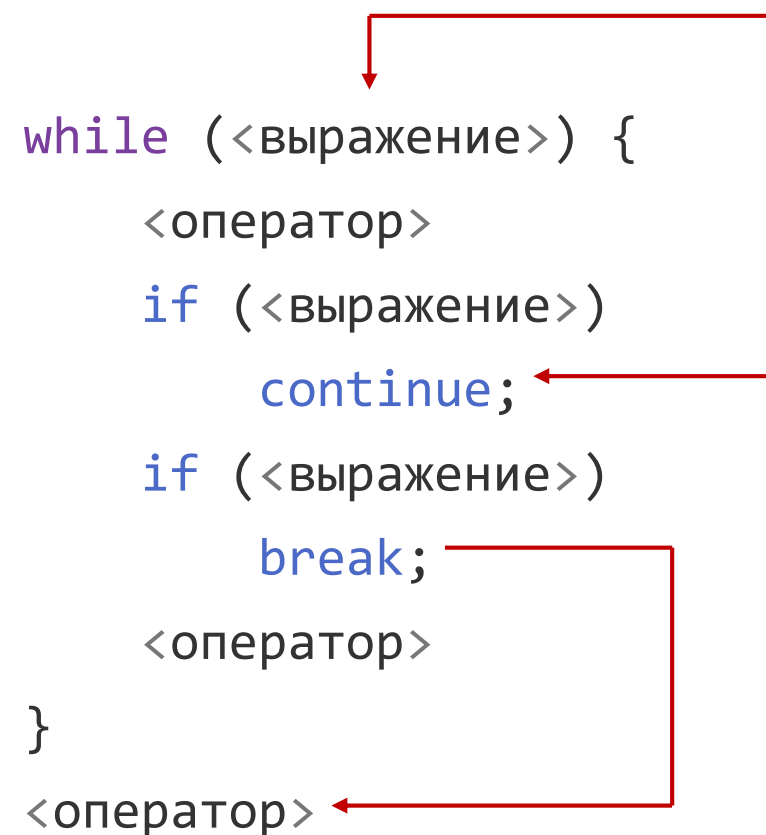
Войти в цикл можно только через его начало



Пример использования циклов

```
int main() {  
    int a, b;  
    for (a=1, b=0; a < 100; b+=a, a++) {  
        if (b % 2)  
            continue;  
        ... /* обработка четных сумм */  
    }  
    return 0;  
}
```

Когда сумма чисел от 1 до a становится нечетной, оператор `continue` передает управление на очередную итерацию цикла `for`, не выполняя операторы обработки четных сумм. Оператор `continue`, как и оператор `break`, прерывает самый внутренний из объемлющих его циклов.



```
while (<выражение>) {  
    <оператор>  
    if (<выражение>)  
        continue;  
    if (<выражение>)  
        break;  
    <оператор>  
}
```

Представление функции полиномом

- Полиномом называется функция вида

$$P(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + a_3 \cdot x^3 + \dots + a_n \cdot x^n$$

- Эффективность решения задачи о вычислении значения полинома проверяется двумя вариантами алгоритмизации решения задачи:

1. $P(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + a_3 \cdot x^3 + \dots + a_n \cdot x^n$

2. Схема Горнера

$$P(x) = (\dots (0 + a_n) \cdot x + a_{n-1}) \cdot x + \dots + a_1) \cdot x + a_0$$

- За счет исключения операции возведения в степень существенно увеличивается точность решения задачи и задача решается за меньшее число итераций.

Алгоритмы этих методов на следующем слайде

Алгоритм вычисления многочлена в естественной форме его записи

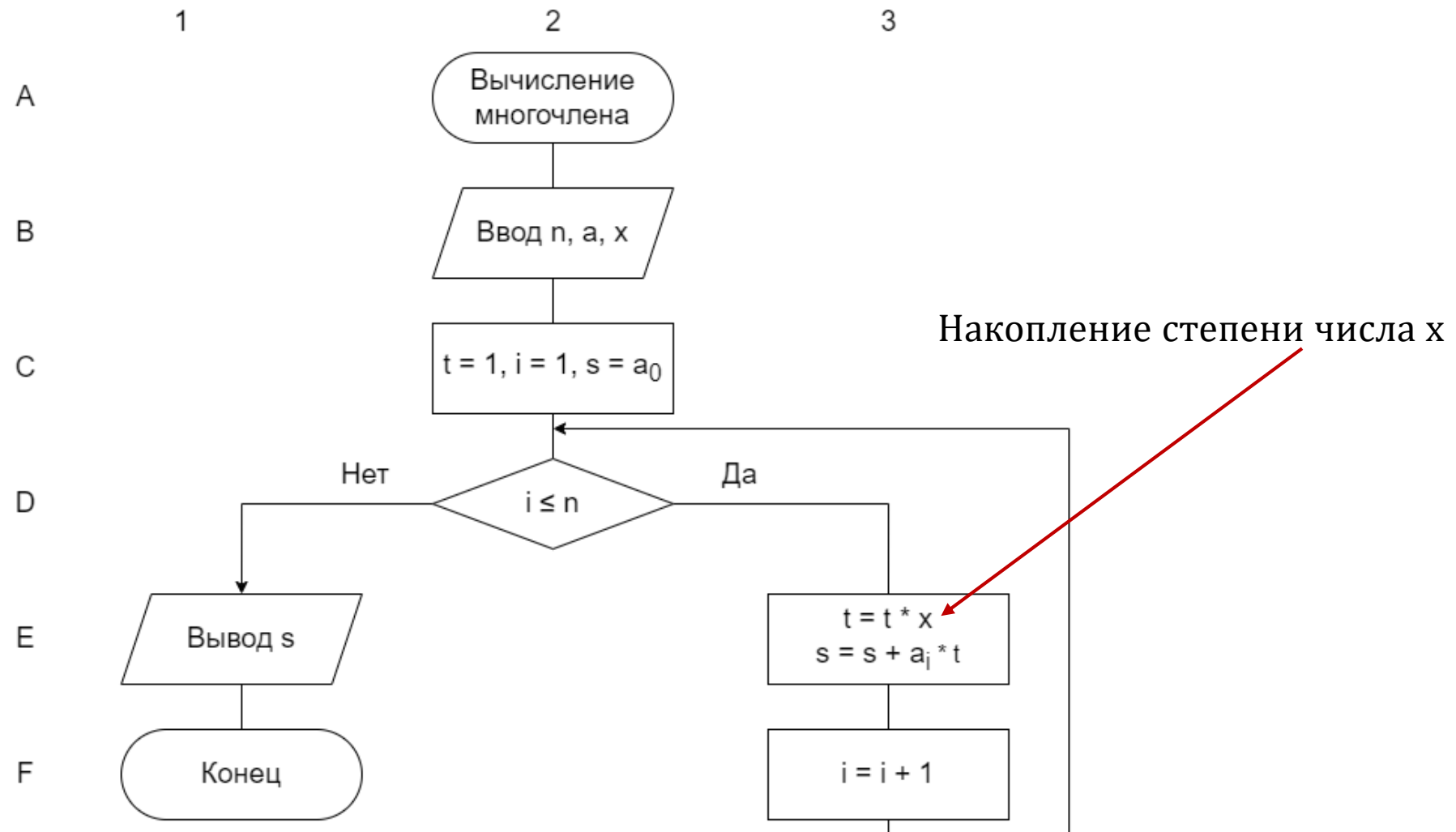
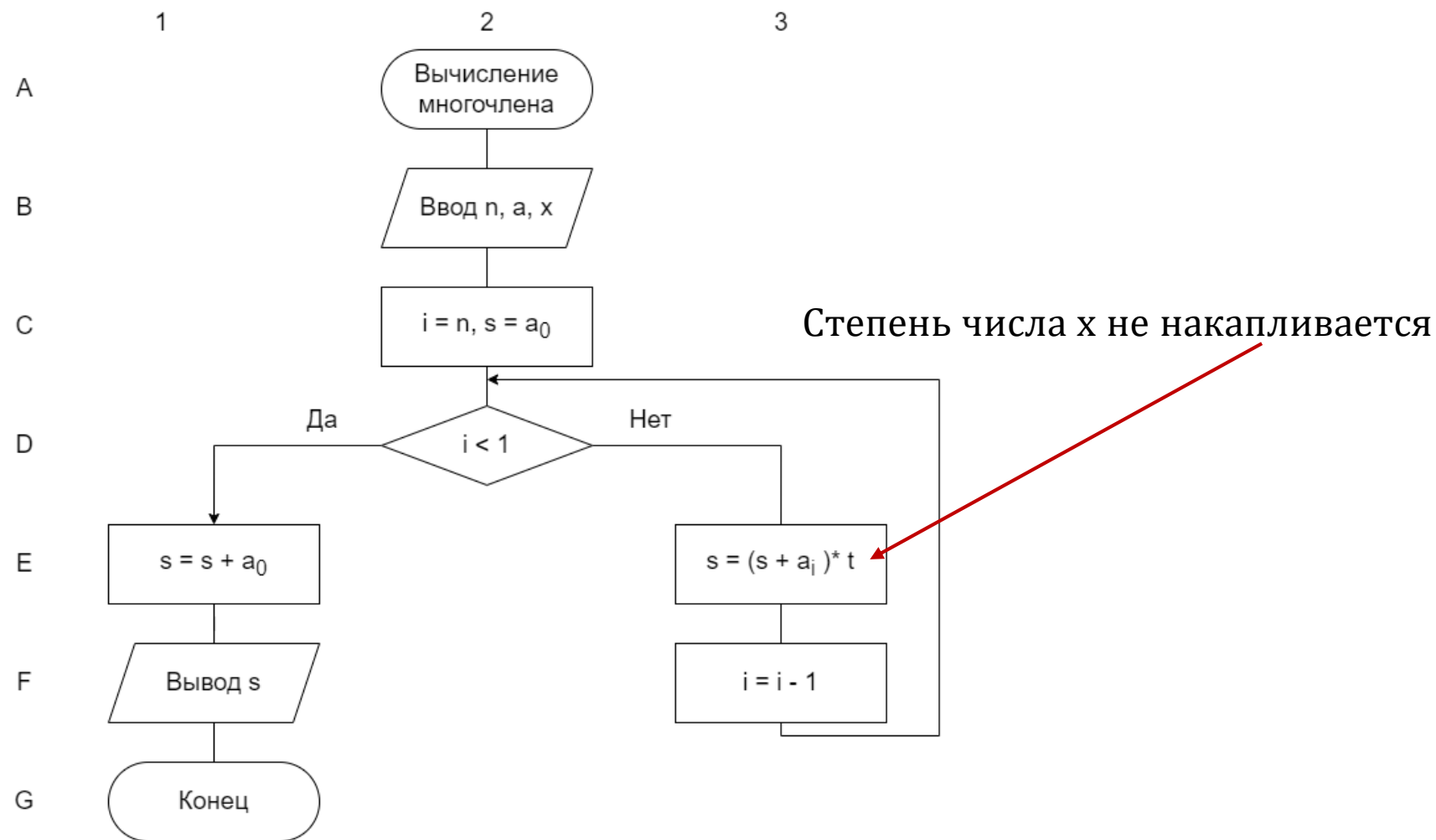


Схема Горнера для вычисления многочлена

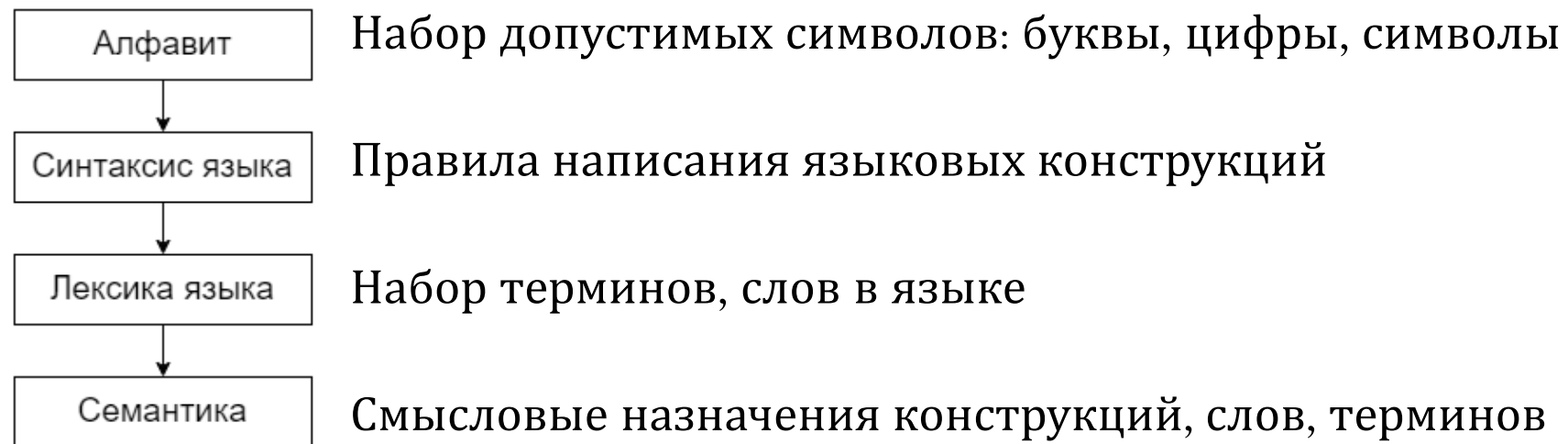


Основные правила алгоритмизации при вычислениях по формулам

- Накопление суммы начинается с 0
- Накопление произведения начинается с 1
- Возведение в степень числа -1 обеспечивается только умножением или вычитанием
- Многократно повторяющиеся действия оформляются циклом или рекурсивно
- Ветвление в алгоритмах возможно только в двух направлениях – направо или налево

Определение языка программирования

- Язык для написания программ по разработанным алгоритмам называется языком программирования.
- Каждый язык программирования определяется средствами для изображения программ.



- В настоящее время разработана система критериев для сравнения языков программирования при обосновании выбора языка для записи решения задачи

Сравнение языков программирования

- Целостность языка.
 - Избыточность конструкций языка
 - Выразительность
 - Многословие и безопасность
-
- Степень типизации данных
 - Контроль типов данных
 - Неявные преобразования типов данных
 - Возможность введения новых типов данных
-
- Операции
 - Управление последовательностью действий
 - Раздельная трансляция
- 
- Синтаксис языка
- Описание данных
- Набор операций

Спасибо за внимание