

Глава 2 Основные операторы языка

2.1 Элементарный ввод вывод

2.1.1 Форматный ввод /вывод

Ввод:

```
int scanf(<Форматная строка>,  
          <Список адресов переменных>);  
// возвращает количество значений или EOF(-1)
```

Вывод:

```
int printf(<Форматная строка>, <Список выражений>);
```

где < Форматная строка> - строка, которая помимо символов содержит спецификации формата вида:

%[-] [<Целое 1>] [.<Целое 2>] <Формат>

«-» - выравнивание по левой границе,

<Целое 1> - ширина поля вывода;

<Целое 2> - количество цифр дробной части числа;

<Формат > - определяется специальной литерой

Спецификации формата

d - целое десятичное число;

u - целое десятичное число без знака;

o - целое число в восьмеричной системе счисления;

x - целое число в шестнадцатеричной системе счисления (% 4x - без гашения незначащих нулей);

f - вещественное число;

e - вещественное число в экспоненциальной форме;

c - СИМВОЛ;

p - указатель (адрес);

s - символьная строка.

Кроме этого, форматная строка может содержать:

\n - переход на следующую строку;

\n hhh - вставка символа с кодом ASCII hhh (код задается в шестнадцатеричной системе счисления);

%% - печать знака %.

Примеры форматного ввода/вывода

а) `i=26;`

```
printf ("% -6dUUU%%U %oU %x\n", i, i, i);
```



26UUUUUUUU%U32U1A ↵

б) `scanf ("%d %d", &a, &b);`

Вводимые значения: 1) 24 28 2) 24 ↵
28

в) `scanf ("%d,%d", &a, &b);`

Вводимые значения: 24,28

г) `scanf ("%s", name);`

Вводимые значения: Иванов Иван

Результат ввода: name="Иванов"

Ввод/вывод строк

Ввод:

```
char* gets (<Строковая переменная>) ;  
// возвращает копию строки или NULL
```

Вывод:

```
int puts (<Строковая константа или переменная>) ;
```

Примеры:

а) `puts ("Это строка") ;`

Результат: **Это строка**↵

б) `gets (st) ;`

Вводимые значения: **Иванов Иван**↵

Результат: **st = "Иванов Иван"**

Ввод/вывод символов

Ввод

`int getchar() ;` // возвращает символ или **EOF**

Вывод

`int putchar(<Символьная переменная или константа>);`

Примеры:

а) `ch=getchar() ;`

б) `putchar(' t') ;`

Программа определения корней кв. уравнения

// Ex2_1

```
#include "stdafx.h"
#include <stdio.h>
#include <math.h>
int main(int argc, char* argv[])
{ float A,B,C,E,D,X1,X2;
  puts("Input A,B,C");
  scanf("%f %f %f",&A,&B,&C);
  printf("A=%5.2f B=%5.2f C=%5.2f \n",A,B,C);      ;
  E=2*A;
  D=sqrt(B*B-4*A*C);
  X1=(-B+D)/E;
  X2=(-B-D)/E;
  printf("X1= %7.3f X2=%7.3f \n",X1,X2);
  return 0;
}
```

2.2 Блок операторов

{ <Оператор>;... <Оператор>;}

Пример:

```
{  
    f=a+b;  
    a+=10;  
}
```

2.3 Управляющие конструкции

Управляющими называются операторы, способные изменять естественный ход линейного процесса.

2.3 Оператор условной передачи управления

if (<Выражение>) <Оператор;> [else <Оператор;>]

Оператор – любой оператор C++, в том числе другой оператор условной передачи управления, а также **блок операторов**.

Выражение – любое выражение, соответствующее правилам C++
если значение выражения не равно нулю, то выполняется оператор, следующий за выражением;

если значение выражения равно нулю, то либо выполняется оператор альтернативной ветви, либо управление передается следующему за IF оператору.

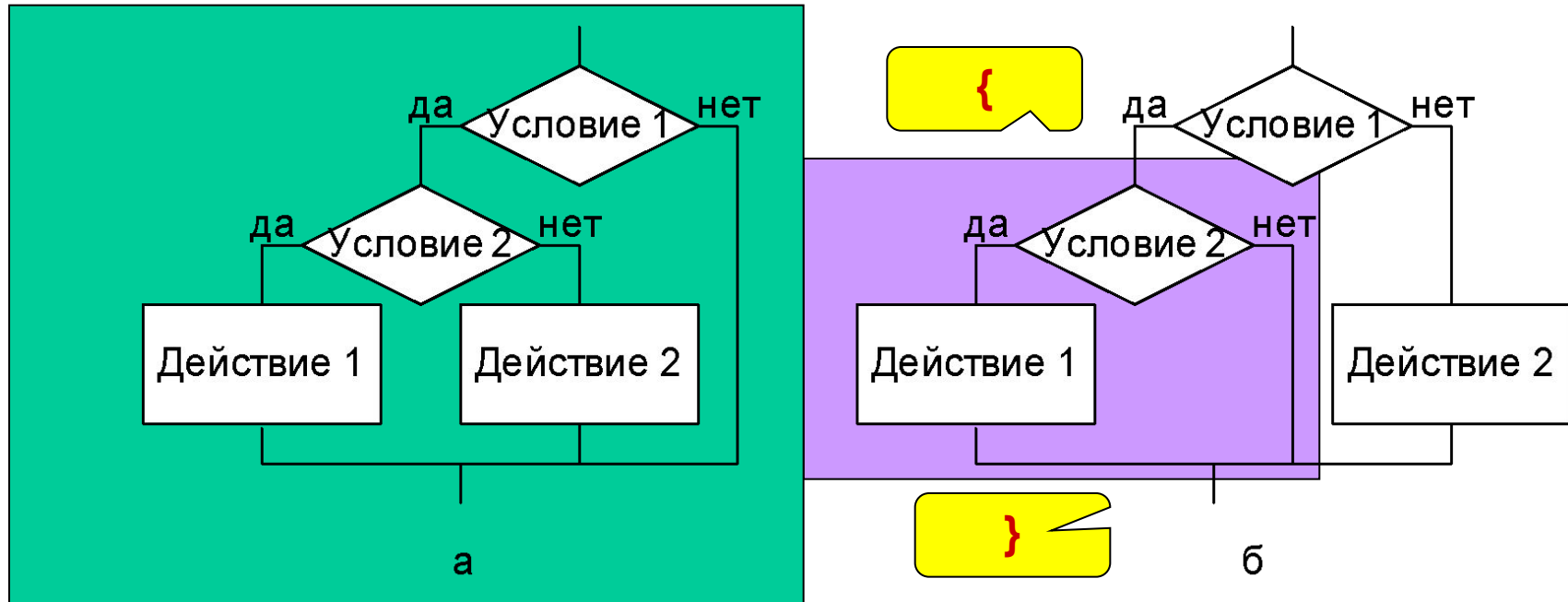
Оператор условной передачи управления(2)

Правило

вложения
`if <Условие1> then`

`if <Условие2> then <Действие1>`

`else <Действие 2>`



Ветвь `else` относится к ближайшему `if`.

Для реализации варианта `б` используют блок операторов `{...}`:

```
if <Условие1>
```

```
    {if <Условие2> <Действие1> }
```

```
else <Действие 2>
```

Оператор условной передачи управления (3)

Примеры:

а) `if (!b)`

```
    puts("с - не определено") ; // если b=0, то – ошибка,  
    else {c=a/b; printf("с=%d\n", c) ;} // иначе - выводится с.
```

б) `if ((c=a+b) != 5) c+=b;`
`else c=a;`

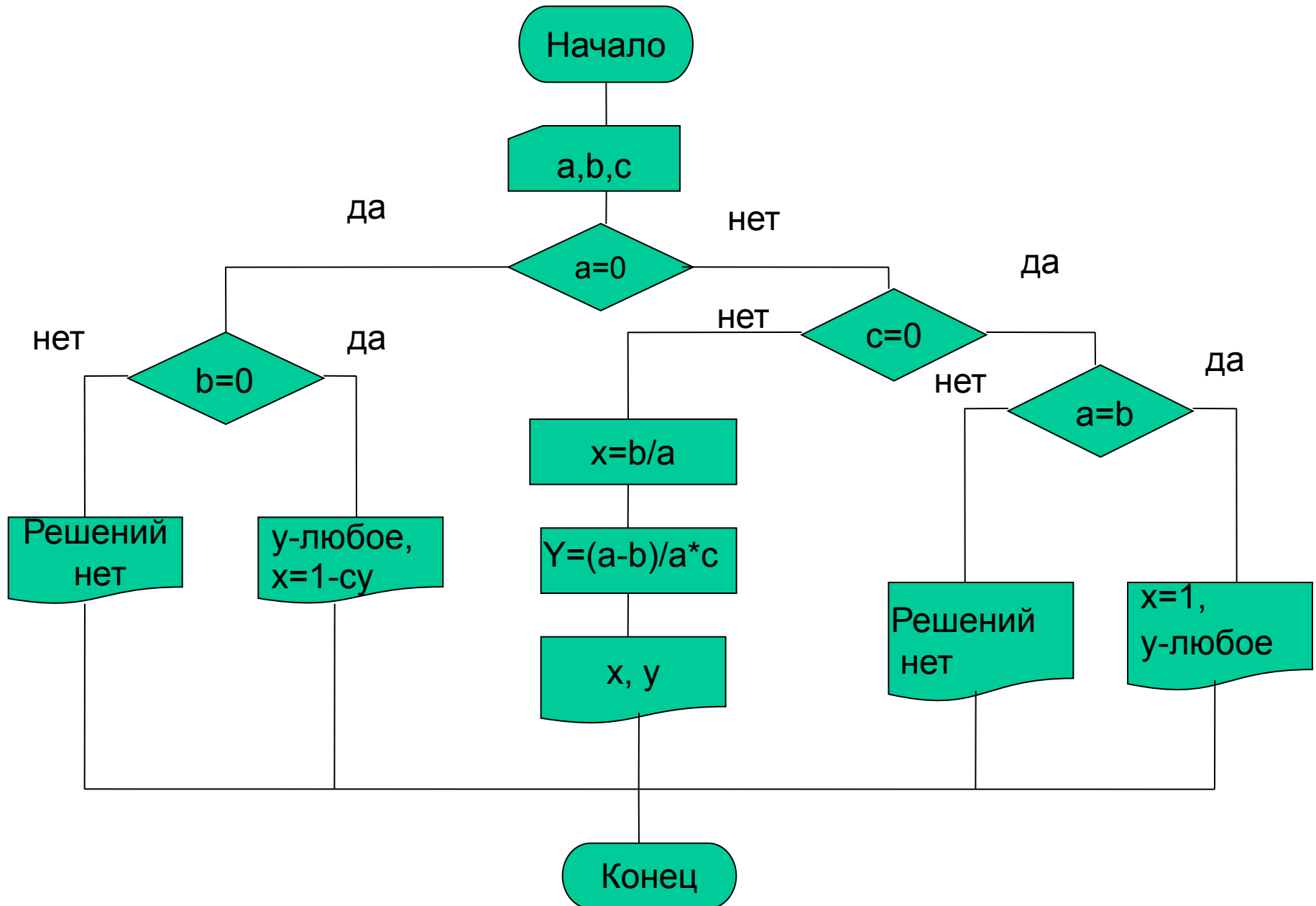
в) `if ((ch=getchar()) == 'q')` // если в ch введено q,
`puts ("Программа завершена.");` // то ...
`else puts ("Продолжаем работу...");` // иначе ...

г) `ch='a' ;`
`if ((oldch=ch, ch='b') == 'a') puts ("Это символ 'a'\n") ;`
`else puts ("Это символ 'b'\n") ;`

Задача: решить систему уравнений

$$\begin{cases} ax=b \\ x+cy=1 \end{cases}$$

Схема алгоритма решения системы уравнений



Программа решения системы уравнений

```
// Ex2_2
```

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

```
float y,x,a,b,c;
```

```
int main(int argc, char* argv[])
```

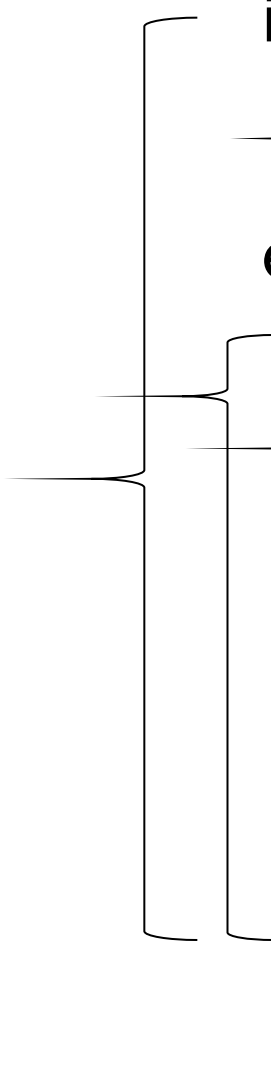
```
{ puts("Input a,b,c");  
  scanf("%f %f %f",&a,&b,&c);  
  printf("a=%5.2f  b=%5.2f  c=%5.2f\n",a,b,c);
```

Подключение
библиотек

Описание
переменных

Ввод и печать
исходных данных

Программа решения системы уравнений(2)



```
if (a==0)
{
    if (b==0) puts("Solution is epsent");
    else printf("y - luboe x=1-c*y");
}
else
{
    if (c==0)
    {
        if (a=b) puts("Solution is epsent");
        else puts("x=1, y- luboe");
    }
    else
    {
        x=b/a;
        y=(a-b)/a/c;
        printf("x= %7.3f  y=%7.3f\n",x,y);
    }
}
return 0;
}
```

2.2 Оператор выбора

Если количество альтернатив велико, то можно использовать оператор выбора.

Оператор реализует конструкцию выбора.

```
switch (<выражение>)  
{  
    case <элемент>: <операторы;>  
    case <элемент>: <операторы;>  
    ...  
    [ default : <операторы;> ]  
}
```

Где:

<выражение> –переключающее выражение . Должно быть целочисленного типа или его начение приводится к целочисленному.

<элемент> - константное выражение, приводимое к переключающему. Любой из операторов может быть помечен несколькими метками типа **case** <элемент>:

Результат выражения сравнивается с заданными значениями и, в случае равенства, выполняются соответствующие операторы, которых может быть 0 или более.

Затем выполняются операторы всех последующих альтернатив, если не встретится **break**.

Оператор выбора (2)

Пример:

```
switch (n_day)  
{ case 1:  
  case 2:  
  case 3:  
  case 4:  
  case 5: puts("Go work!"); break;  
  case 6: printf("%s", "Clean the yard and");  
  case 7: puts("relax!");  
}
```

Разработать программу, вычисляющую значения нескольких функций.

Функция выбирается пользователем, который вводит ее код. (Ex2_3).

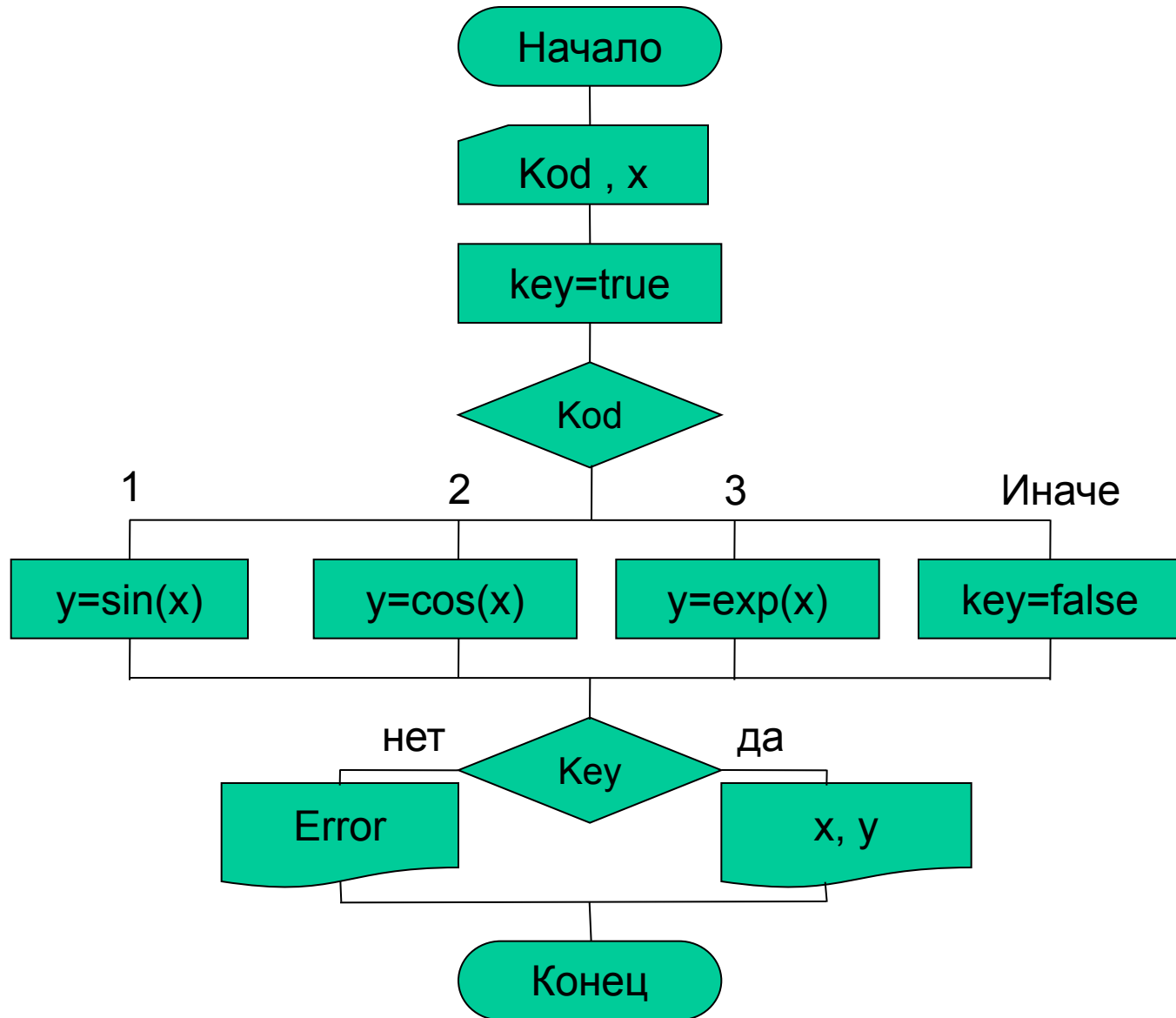
Input cod:

1 – $y = \sin x$

2 – $y = \cos x$

3 – $y = \exp x$

Схема алгоритма



Программа вычисления функции

// Ex2_3

#include "stdafx.h"

#include <stdio.h>

#include <math.h>

int main(int argc, char* argv[])

{ int kod,key;

float x,y;

puts("input x");

scanf("%f",&x)

printf("x=6.3f",x);

puts("input kod");

puts("1 - y=sin(x)");

puts("2 - y=cos(x)");

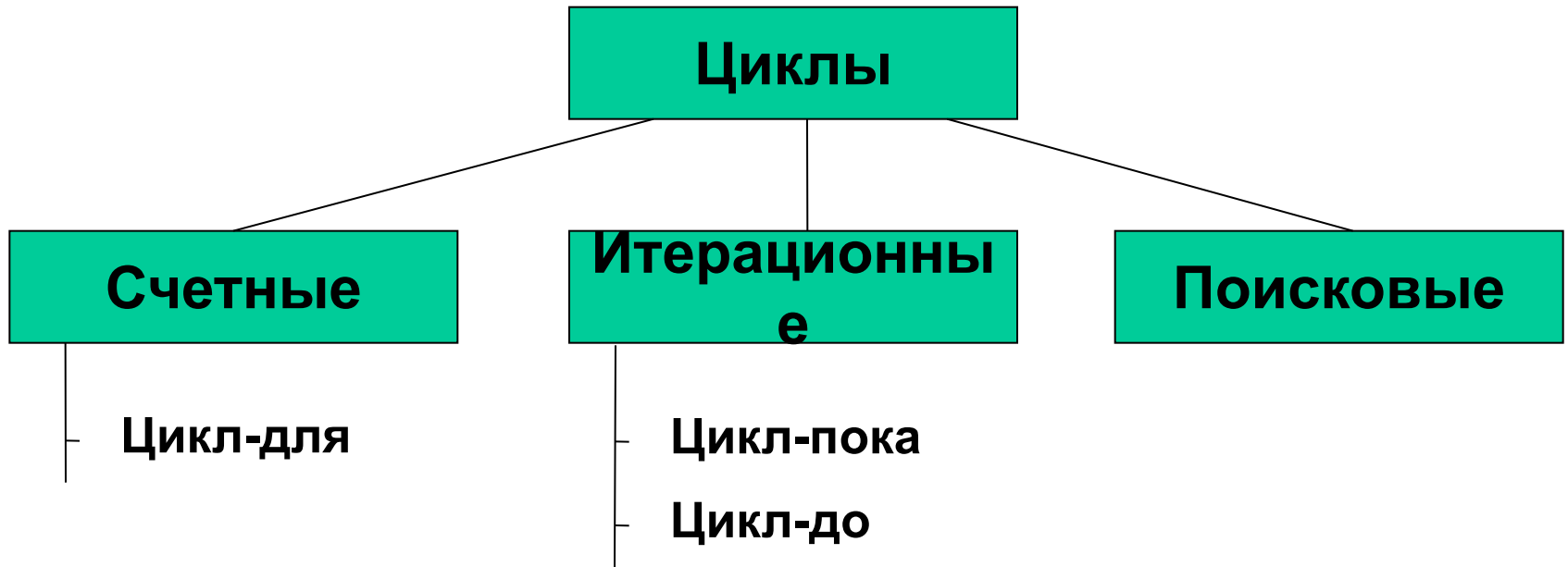
puts("3 - y=exp(x)");

scanf("%d",&kod);

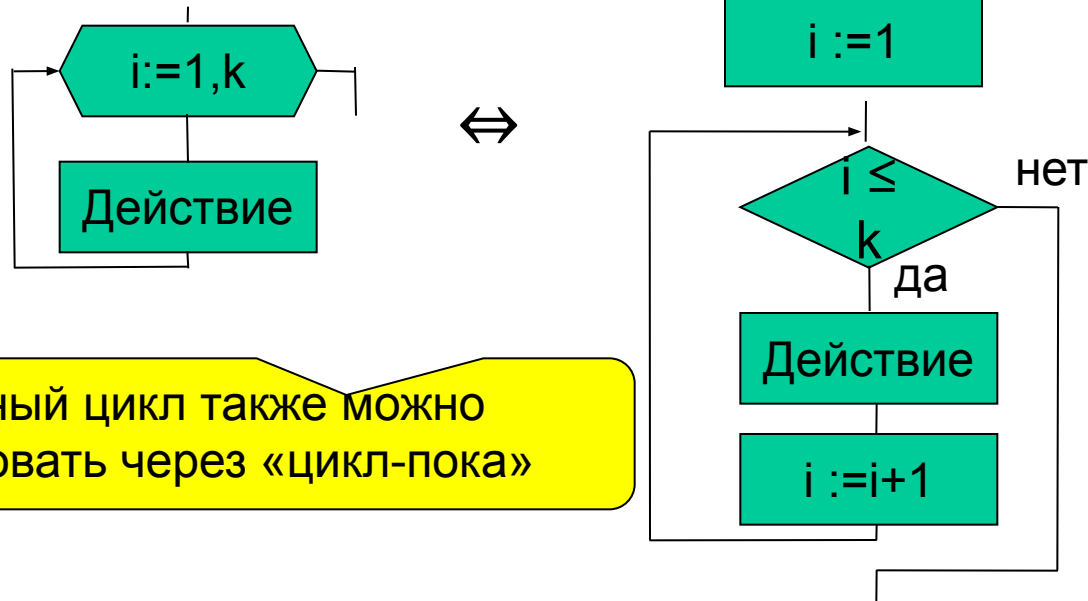
Программа вычисления функции (2)

```
key=1;  
switch(kod)  
{  
    case 1: y=sin(x); break;  
    case 2: y=cos(x);break;  
    case 3: y=exp(x); break;  
    default: key=0;  
  
    }  
if (key) printf("x= %5.2f  y=%8.6f\n",x,y);  
else puts("Error");  
    return 0;  
}
```

2.5 Операторы организации циклов



1. Оператор счетного цикла for



Счетный цикл также можно реализовать через «цикл-пока»

for (<Выражение1>;<Выражение2>;<Выражение3>)<Оператор>;

Эквивалентно:

```
<Выражение1>  
while (<Выражение2>)  
{<Оператор>;  
  <Выражение3>;  
}
```

Оператор счетного цикла **for** (2)

Выражение1 – инициализирующее выражение; представляет собой последовательность описаний, определений и выражений, разделенных запятыми. Выполняется только один раз в начале цикла и задает начальные значения переменным цикла. Может отсутствовать, при этом точка с запятой остается.

Выражение2 –выражение условия; определяет предельное значение параметра цикла. Может отсутствовать, при этом точка с запятой остается.

Выражение3 – список выражений, которые выполняются на каждой итерации цикла после тела цикла, но до следующей проверки условия. Обычно определяют изменение параметра цикла. Может отсутствовать

Оператор – тело цикла. Может быть любым оператором C++, блоком операторов (тело цикла содержит более одного простого оператора) или может отсутствовать.

1. **for(int i=0,float s=0;i<n;i++)s+=i;**

2. **int i=0;float s=0;**
for(;i<n;s+=i++);

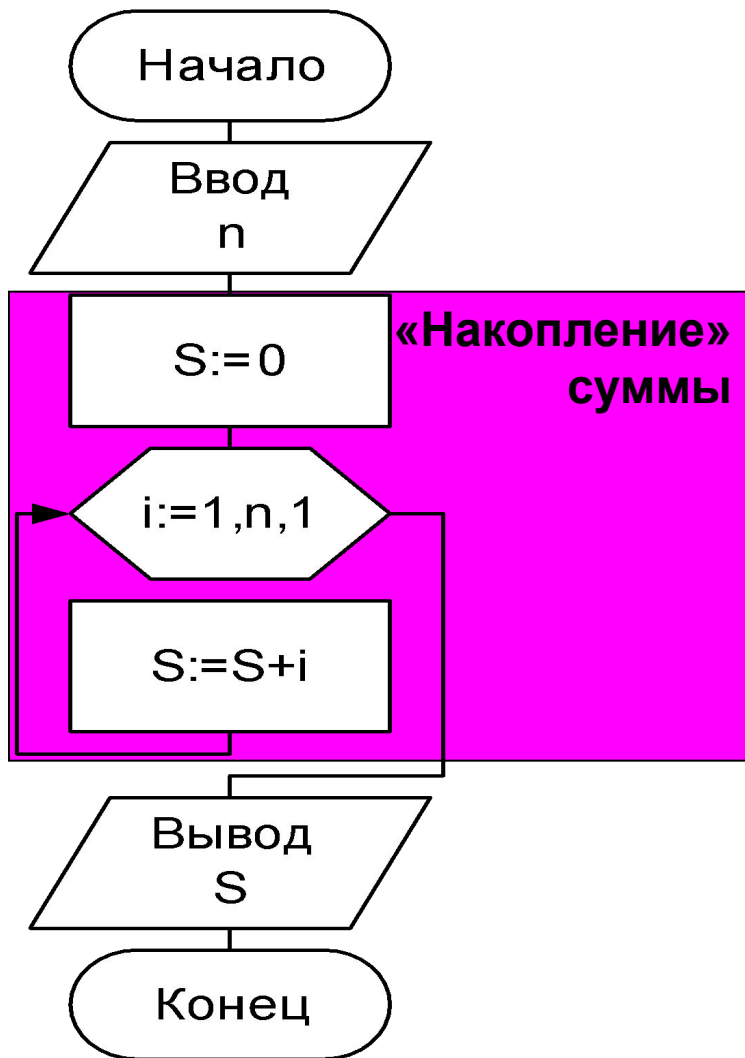
3. **for(;i<n;)s+=i++;**

4. **int l;float s; s=0;**
for(i=n;i>0;i--) s=s+i;

5. **for(;;);**

Суммирование натуральных чисел

Найти сумму N натуральных чисел.(Ex2_for)



```
#include "stdafx.h"
```

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

```
int main(int argc, char* argv[])
```

```
{ int i,n,s;
```

```
puts("Input n");
```

```
scanf("%d",&n);
```

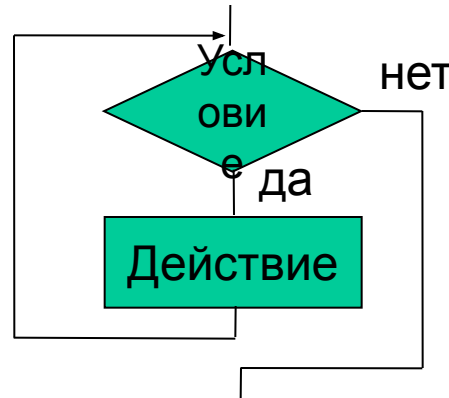
```
for (i=1,s=0;i<=n;i++) s+=i;
```

```
printf("Sum=%5d n=%4d\n",s,n);
```

```
return 0;
```

```
}
```

Цикл-пока



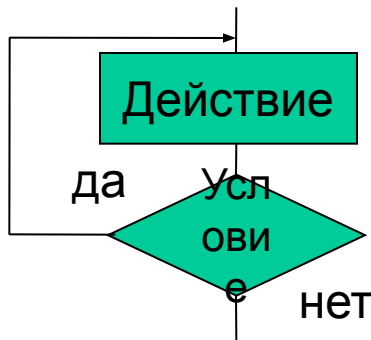
while (<Выражение>) <Оператор>;

Где:

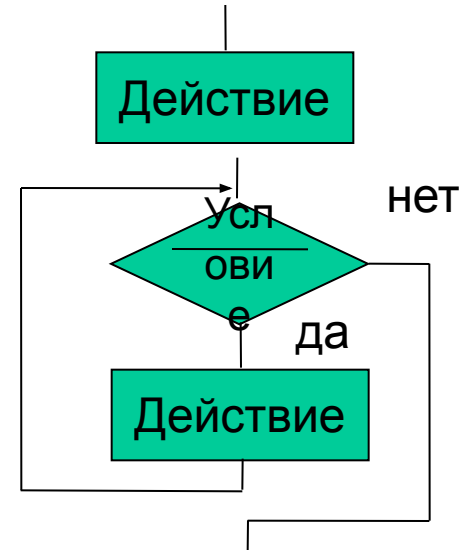
Выражение - совокупность выражений, разделенных запятой, определяющая условия выполнения цикла. Результат такого составного выражения – значение последнего выражения. Цикл выполняется до тех пор, пока результат выражения отличен от нуля.

Оператор – любой оператор C++, в том числе блок операторов.

Цикл-до



⇔



«Цикл-до» можно реализовать
через «цикл-пока»

do <Оператор > while (<Выражение>) ;

Цикл выполняется до тех пор, пока результат выражения отличен от нуля.

Пример. Игнорировать ввод значения, выходящего за пределы заданного интервала.

```
do {  
    printf("Введите значение от %d до %d : ", low, high);  
    scanf(" %d ", &a);  
} while (a < low || a > high);
```


Вложенные циклы

Вложенными циклическими процессами называются такие процессы, при которых внутри одного циклического процесса, происходит другой.

Каждый из процессов может реализоваться различными операторами цикла.

Внешний цикл может быть счетным, а внутренний – итерационным и наоборот.

На количество вложенных циклов компилятор C++ не накладывает никаких ограничений. Оно определяется логикой программы и желанием программиста.

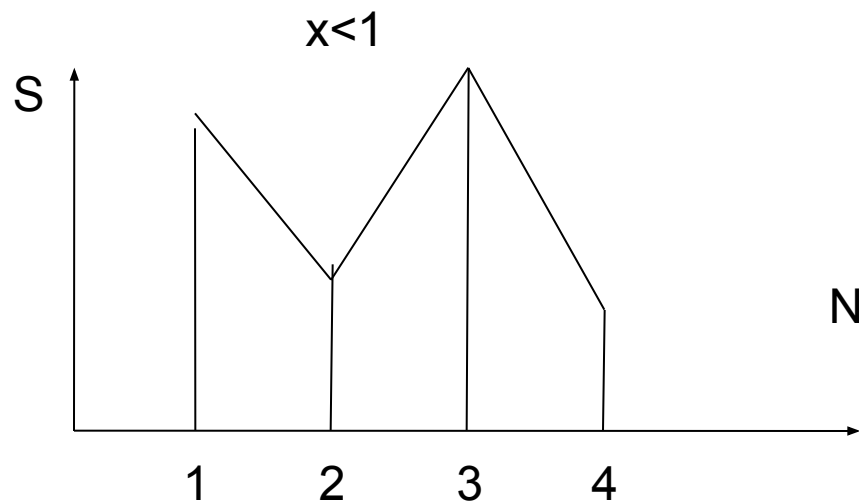
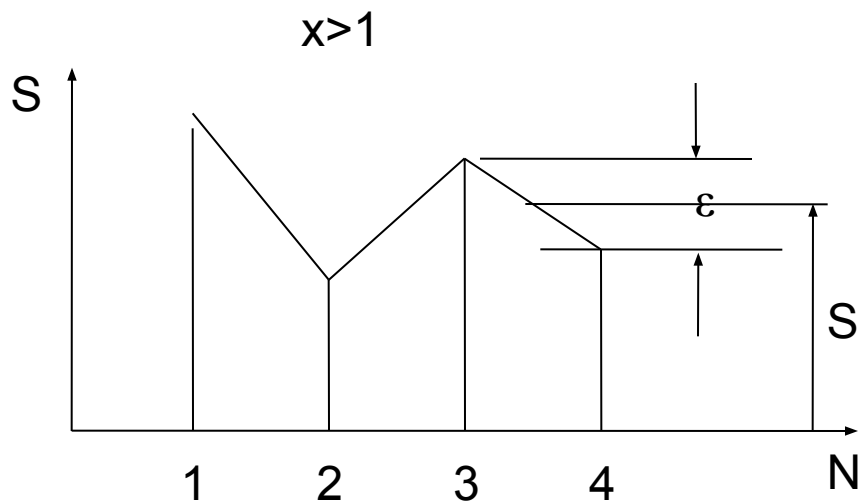
При программировании циклов необходимо соблюдать правило строгой вложенности – начала и концы циклов не должны перекрещиваться, а каждый вложенный цикл иметь начало и конец внутри внешнего цикла.

Вход внутрь цикла по `goto` возможен только через его начало.

Суммирование ряда

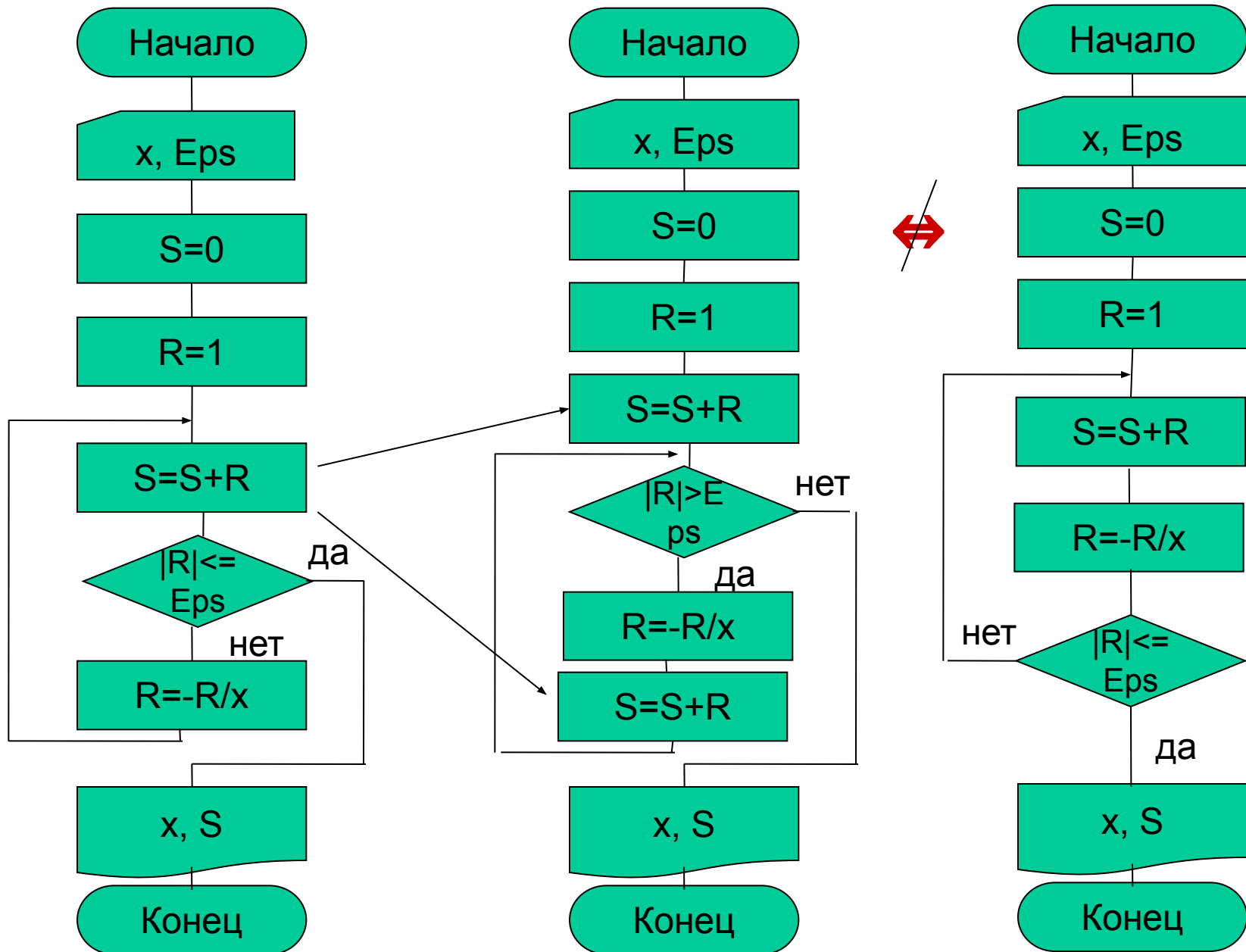
Определить сумму ряда

$S = 1 - 1/x + 1/x^2 - 1/x^3 + \dots$ с заданной точностью ϵ .

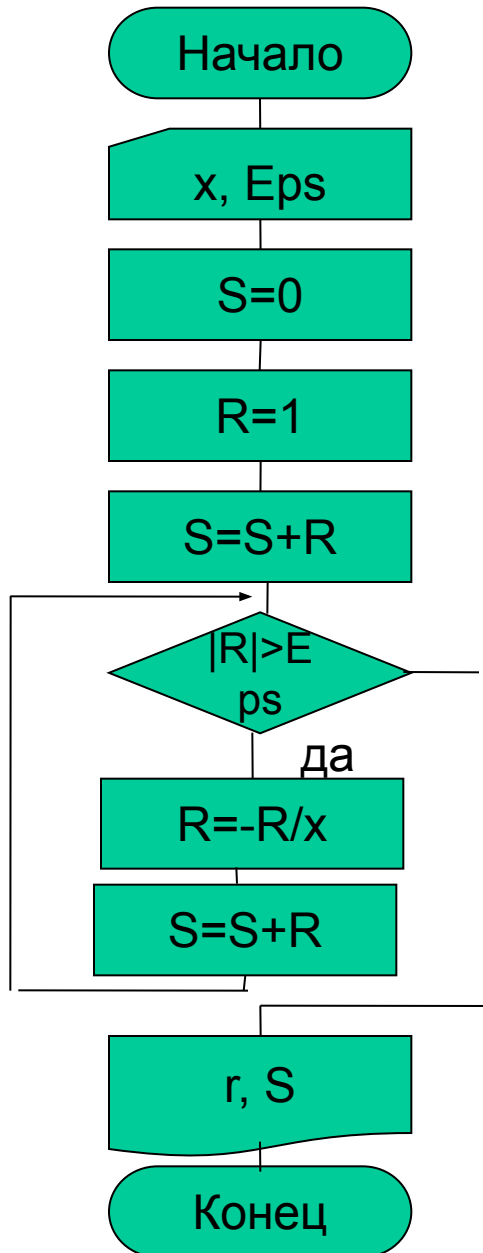


$$R_n = -R_{n-1}/x$$

Приведение алгоритма к структурному



Вариант а (Ex2_4)



```
#include "stdafx.h"
```

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

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

```
void main(int argc, char* argv[])
```

```
{
```

```
    float s, r,x,eps;
```

```
    puts("Input x, eps:");
```

```
    scanf("%f %f", &x, &eps);
```

```
    s=0;
```

```
    r=1; s+=r;
```

```
    while (fabs(r)>eps)
```

```
    {r=-r/x;
```

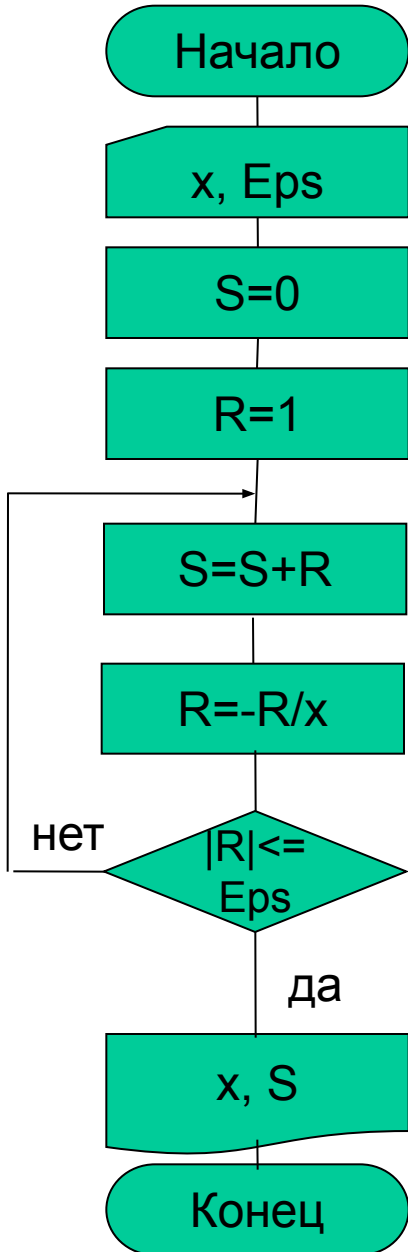
```
      s+=r;
```

```
    }
```

```
    printf(" Result= %10.7f  r=%10.8\n",  
          s,r);
```

```
}
```

Вариант 6 (Ex2_5)



```
#include "stdafx.h"
```

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

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

```
void main(int argc, char* argv[])
```

```
{ float s, r,x,eps;
```

```
    puts("Input x, eps:");
```

```
    scanf("%f %f", &x, &eps);
```

```
    s=0;  r=1;
```

```
    do
```

```
        { s+=r;
```

```
          r=-r/x;
```

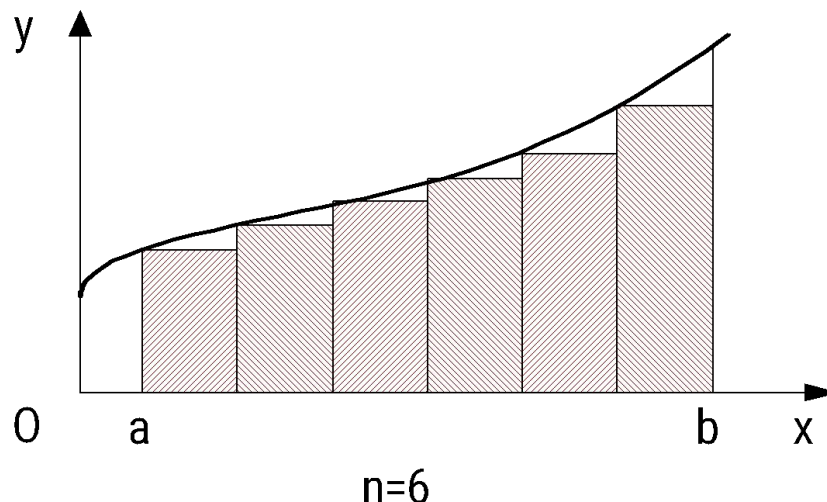
```
        } while (fabs(r)>eps);
```

```
    printf("Result= %10.7f r=%10.8f.\n", s,r);
```

```
}
```

Решение задач вычислительной математики

Задача. Вычислить определенный интеграл функции $f(x)$ на интервале $[a, b]$ методом прямоугольников с точностью δ .



Итак

$$S = f(x_1) \times d + f(x_2) \times d + f(x_3) \times d + \dots + f(x_n) \times d = d \times \sum_{i=1}^n f(x_i), \text{ где } d = (b-a)/n.$$

Увеличивая n , получаем приближения площади: $S_1, S_2, S_3 \dots$

Останавливаемся, когда $|S_k - S_{k+1}| < \delta$

Неформальное описание алгоритма

Алгоритм:

Шаг 1. Ввести a , b , δ .

Шаг 2. Задать число прямоугольников $n:=10$.

Шаг 3. Определить шаг $d:=(b-a)/n$.

Шаг 4. Определить площадь фигуры $S1$.

Шаг 5. Увеличить число прямоугольников вдвое $n:=n*2$.

Шаг 6. Уменьшить шаг вдвое $d:=d/2$.

Шаг 7. Определить площадь фигуры $S2$.

Шаг 8. Если Разность площадей меньше δ , то перейти к шагу 11

Шаг 9. Запомнить новое значение площади $S1:=S2$.

Шаг 10. Перейти к шагу 5.

Шаг 11. Вывести $S1$.

Конец.

Схема алгоритма (неструктурная и неэффективная)

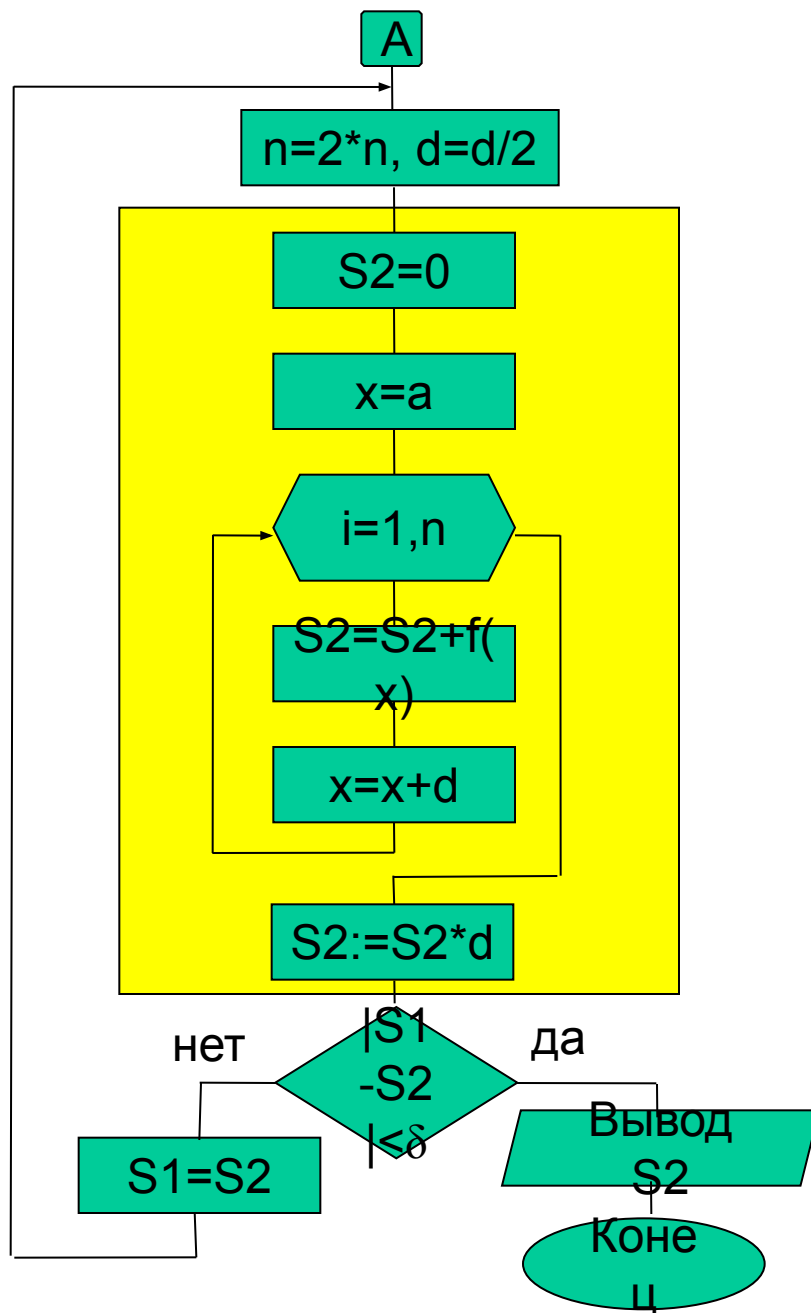
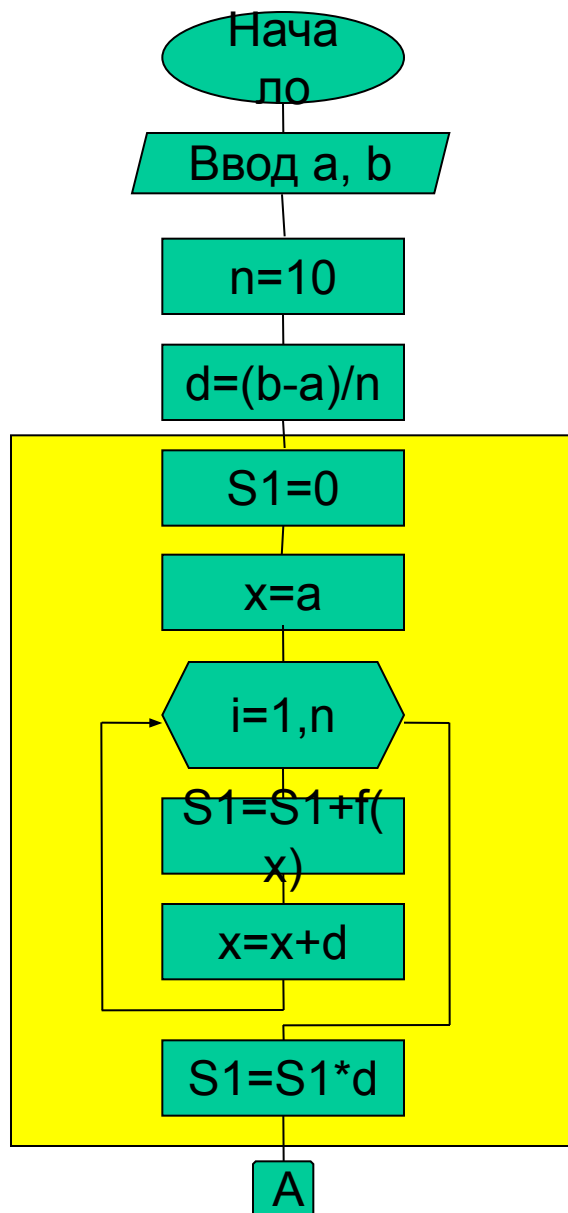
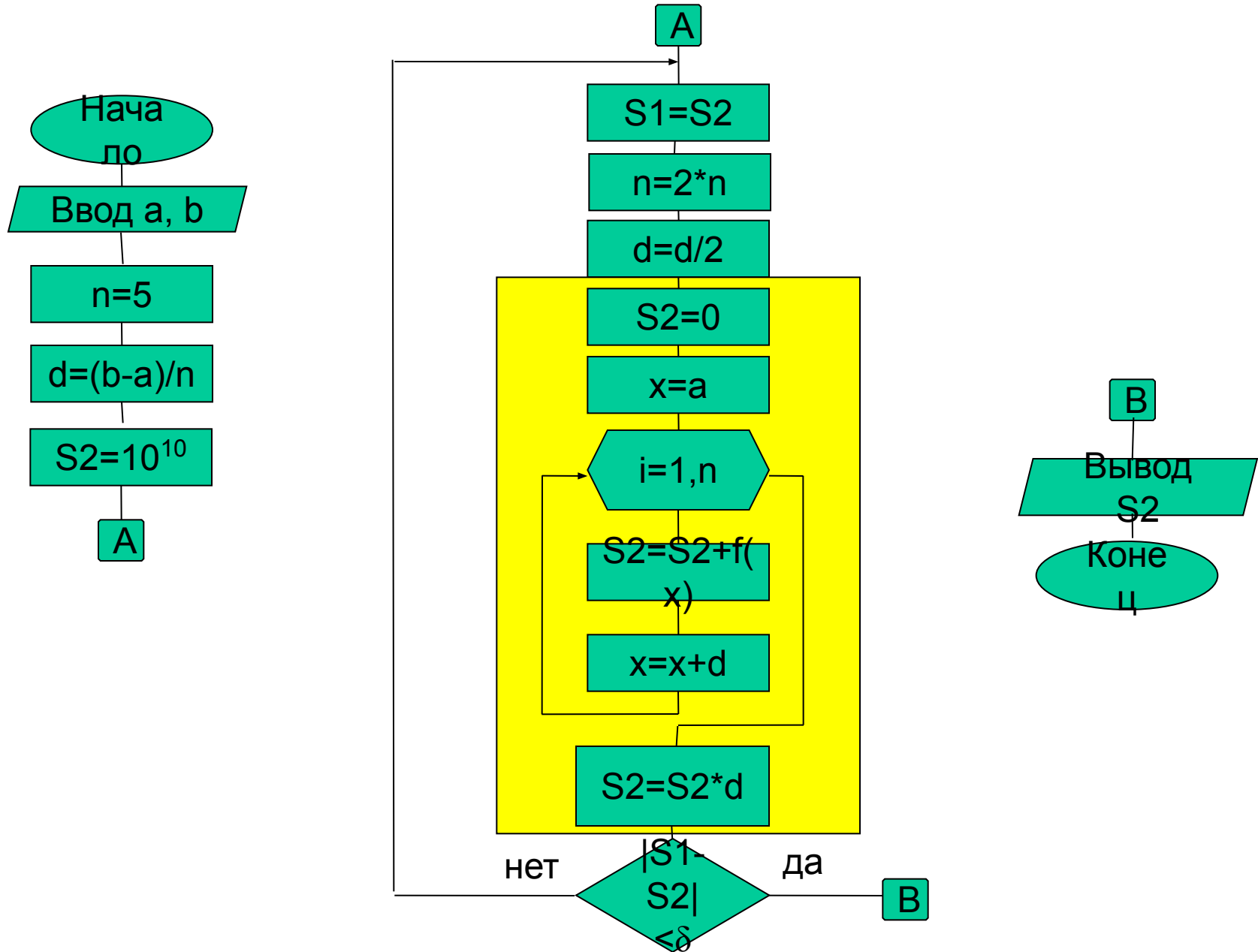


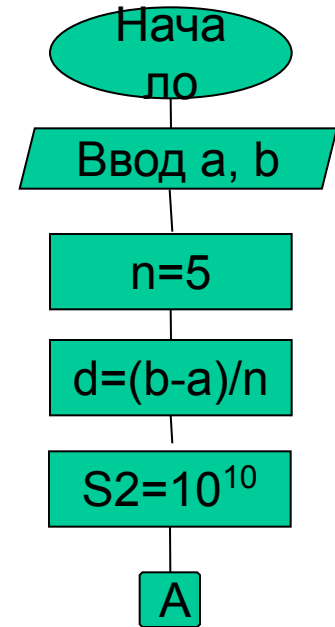
Схема структурированная и сокращенная



Программа

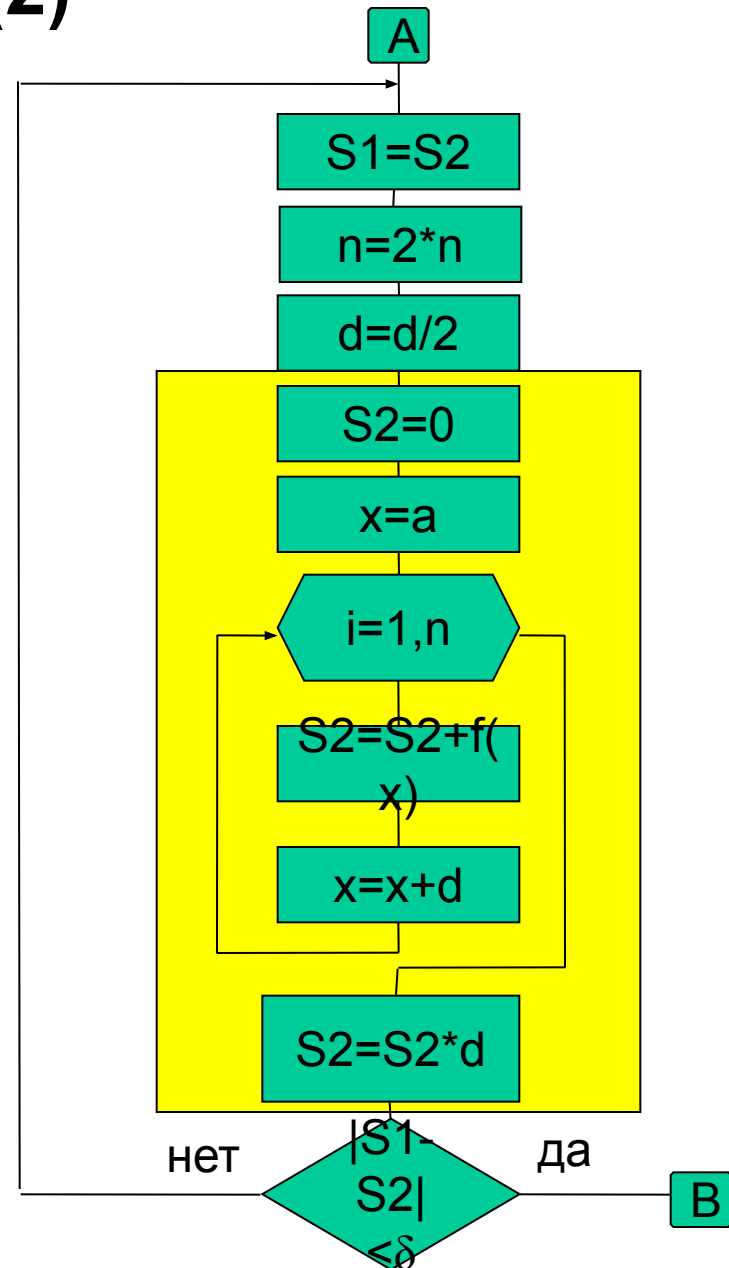
```
// Ex2_6.cpp
#include "stdafx.h"
#include <stdio.h>
#include <math.h>

int main(int argc, char* argv[])
{int i,n;
float s1,s2,x,a,b,eps,d;
puts("input a,b,eps");
scanf("%f %f %f",&a,&b,&eps);
n=5;
d=(b-a)/n;
s2=1.0e+10;
```



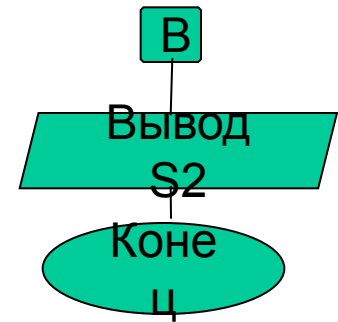
Программа (2)

```
do
{ s1=s2;
  s2=0;n=n*2;
  d=d/2;
  x=a;
  for(i=1;i<=n;i++)
  { s2=s2+x*x-1;
    x=x+d;
  }
  s2=s2*d;
} while(fabs(s2-s1)>eps);
```



Программа(3)

```
printf("l= %10.7f  n= %6d\n",s2,n);  
return 0;  
}
```



2.6 Неструктурные операторы передачи управления

1. Оператор безусловного перехода goto

goto <Метка перехода>;

Пример:

```
again: x=y+a;
```

```
...
```

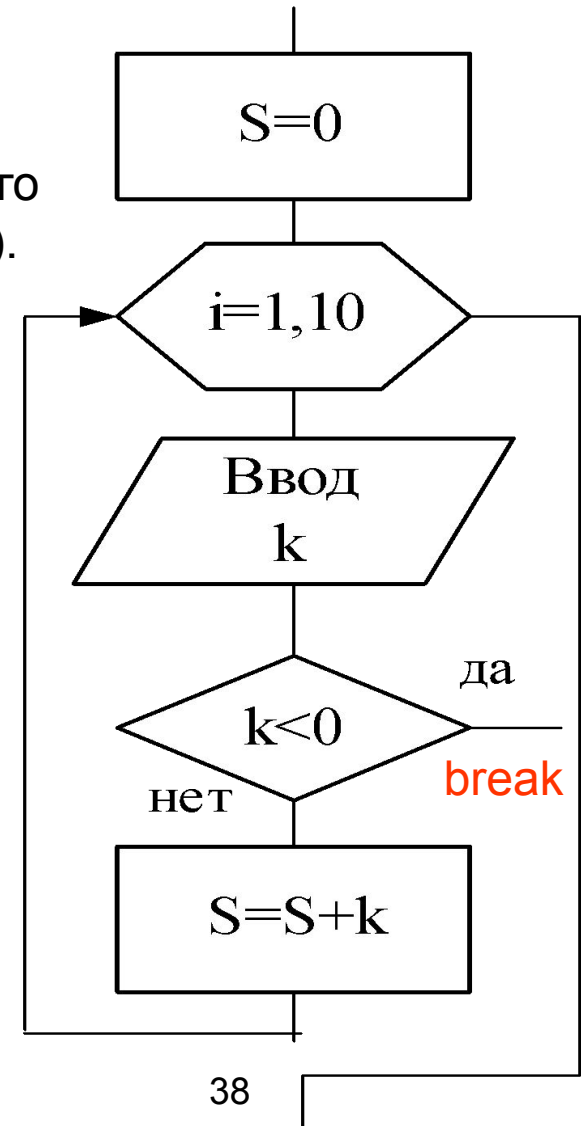
```
goto again;
```

2. Оператор досрочного завершения break

break;

Пример. Суммирование до 10 чисел вводимой последовательности. При вводе отрицательного числа работа программы завершается (**Ex2_7**).

```
#include "stdafx.h"
#include <stdio.h>
void main()
{ int s=0, i, k;
  puts("Input up to 10 numbers.");
  for (i=1; i<11; i++)
  { scanf("%d",&k);
    if (k<0) break;
    s+=k;
  }
  printf("Result = %d.\n",s);
}
```

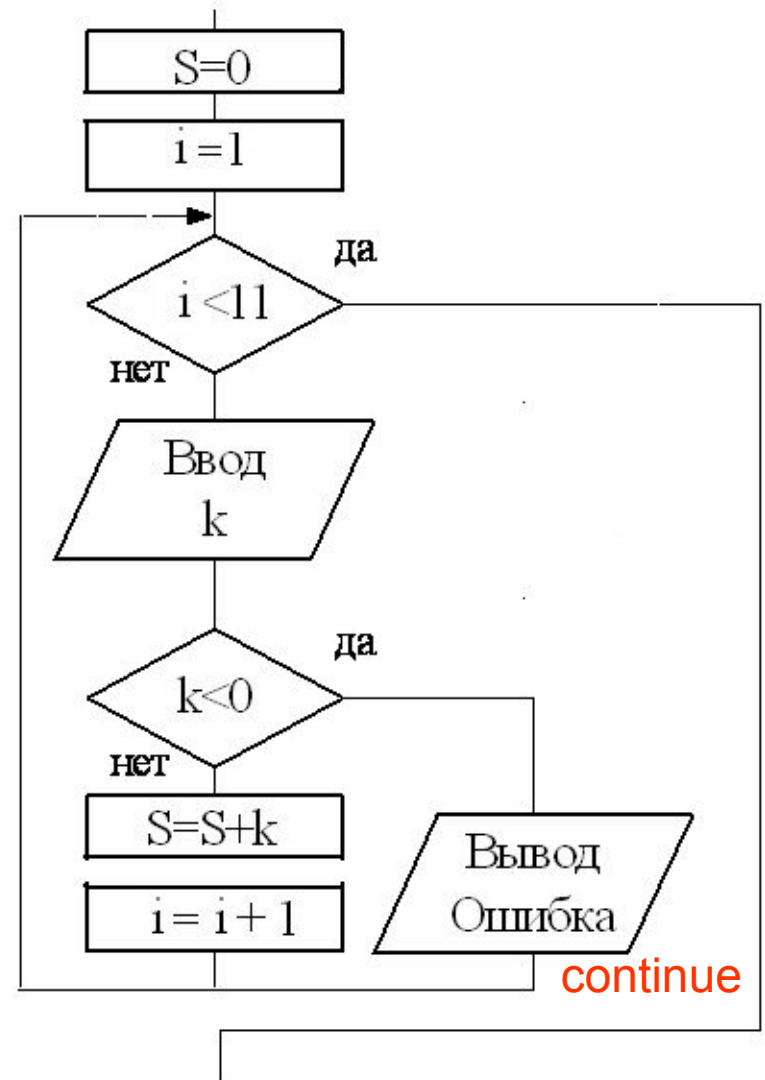


3. Оператор продолжения continue

continue;

Пример 5. Программа суммирует 10
целых положительных чисел (**Ex2_8**).

```
#include "stdafx.h"
#include <stdio.h>
void main()
{ int s=0,i=1,k;
  puts("Input 10 numbers.");
  while(i<11)
  { scanf("%d",&k);
    if (k<0) { puts("Error.")
              continue;
            }
    s+=k; i++;
  }
  printf("Result = %d.\n",s);
}
```



Пример 6. Вывод таблицы кодов (Ex2_9)

```
#include "stdafx.h"
#include <stdio.h>
int main(int argc, char* argv[ ])
{
    int i,i1,in,col;
    puts("Input first and last values");
    scanf("%d %d",&i1,&in);
    puts("Input colon number");
    scanf("%d",&col);
    for(i=i1;i<=in;i++)
        if (i<in)
            printf("%c-%3d;%c",i,i,((i-i1+1)%col!=0)?' ':'\n');
        else printf("%c - %3d.",i,i);
    return 0;
}
```

-32; !-33; "-34; #-35;
\$-36; %-37; &-38; ' -39.