Элементы ЯПВУ

C++

Потоковый (неформатный) ввод / вывод

Одной из важнейших компонент языка программирования C++ является потоковый ввод-вывод.

Поток – в языке C++ понятие относящееся к любому переносу данных от источника к приемнику. Потоковые операции рассматривают все передаваемые данные как поток байтов, без какой-либо структуры.

Входной поток – байты из потока поступают (*извлекаются* >>) в переменные.

cin – объект, извлекающий байты из входного потока и помещающий их в указанные переменные (входной поток по умолчанию связан с буфером клавиатуры).

Выходной поток – байты в поток поступают (*помещаются, включаются* <<) из переменных.

cout – объект, помещающий байты из указанных переменных в выходной поток (выходной поток по умолчанию связан с экраном дисплея).

Система ввода-вывода C++, как объектно-ориентированного языка программирования (ООП), основана не на библиотеке функций, а на библиотеке классов. Одним из базовых принципов ООП является предположение о том, что объекты "знают", что нужно делать при появлении обращения (сообщения) определенного типа, т.е. для каждого типа адресованного ему обращения объект имеет соответствующий механизм обработки.

Объект cout, представляющий выходной поток, выбирает соответствующую процедуру обработки и выводит значение в соответствующем виде. Объект cout не может перепутать и вывести, например, целое число в формате с плавающей точкой.

Для использования cin и cout надо подключать библиотеку <iostream.h> и файлам исходного текста давать расширение .cpp

Элементы ЯПВУ

C++

Потоковый (неформатный) ввод / вывод

cout << элемент_1 << элемент_2 <<…<< элемент_n;

где - cout - **вывод данных** (на экран),
- элемент_n может быть - переменная;
- числовая константа;
- выражение (в круглых скобках);
- строковая константа (в двойных кавычках, в том числе \n – перевод строки);
- ключевое слово endl - перевод строки.

Пример:
```
cout << RL << " – это флаг истинности правила. \n";
cout << " S = " << pow (a+b, 1.0/3);  // Кубический корень из a+b
```

cin >> элемент_1>> элемент_2 >>…>> элемент_n;

где - cin - **ввод данных** (с клавиатуры),
- элемент_n - переменная (но не выражение и не константа).

Пример:
```
int S;  char kl;
cin >>  S >>  kl;   // тип вводимых данных определяется автоматически
```

в C++ сохраняется возможность использовать printf и scanf

Рекурсия и рекуррентность

Рекурсия – от латинского слова "recursio" – возвращение.

Если программа обращается сама к себе как к подпрограмме непосредственно или через цепочку подпрограмм, то это называется рекурсией.

Если подпрограмма p содержит явное обращение к самой себе, то она называется явно рекурсивной. Если подпрограмма p содержит обращение к некоторой подпрограмме q, которая в свою очередь содержит прямое или косвенное обращение к p, то p - называется косвенно рекурсивной.

Рекурсивная программа должна обязательно иметь так называемое терминальное условие, то есть условие при котором программа прекращает рекурсивный процесс, иначе она никогда не остановится.

Рекуррентность – это способ вычисления функции.

Рекуррентный алгоритм задает способ вычисления членов последовательности описывающей функцию при помощи рекуррентных формул. Следующий член последовательности вычисляют как функцию от предыдущего:

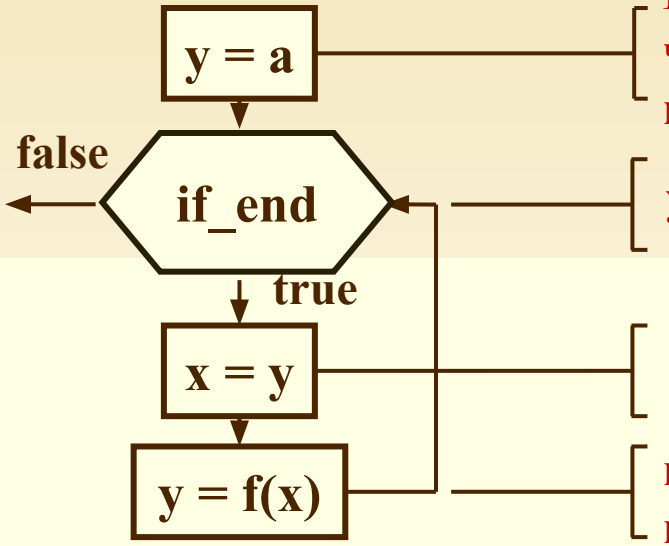
$$x_k = f(x_{k-1}), \text{ где } x_0 = a.$$

Возможен более сложный случай, когда очередной член последовательности зависит от 2-х предыдущих:

$$x_k = f(x_{k-1}, x_{k-2}), \text{ где } x_0 = a,\ x_1 = b.$$

Рекуррентность

При программировании рекуррентного алгоритма организуют цикл, вычисляющий значение очередного члена последовательности $x_k = f(x_{k-1})$:



Инициализация – вычисление значения первого члена последовательности перед запуском цикла и присваивание его переменной y

условие прекращения цикла

смещение очередного члена последовательности

вычисление очередного члена последовательности

Примеры рекуррентных соотношений – это факториал, числа Фибоначчи, алгоритм Евклида и др.

Программы вычисления этих соотношений могут быть реализованы как рекурсивным, так и итерационным способом (в цикле).

Пример рекуррентного алгоритма: вычисление значения очередного члена последовательности ряда натуральных чисел:

$$X_n = X_{n-1} + 1$$

C \ C++

Рекуррентность

Практическое занятие

Вычислить число π с заданной пользователем точностью:

Для вычисления используем факт, что значение частичной суммы ряда $1 - {}^1/_3 + {}^1/_5 - {}^1/_7 + {}^1/_9 - {}^1/_{11}\ldots$ *при суммировании достаточно большого количества членов приближается к* $\pi / 4$.

На экране должно быть: **Задайте точность вычисления ПИ -> 0.001**

Вычисление ПИ с точностью 0.001000:
Значение числа ПИ с точностью 0.001000 равно 3.143589.
Просуммировано 502 член(а)(ов) ряда.

```
#include <stdio.h>
#include <clocale>
void main()
{   setlocale(LC_CTYPE, "rus");
float p, t, el;     // значение ПИ, точность, значение члена ряда
int   n;            // номер члена ряда
 p = 0;
 n = 1;
 el = 1;     // начальное значение члена ряда
printf ("\nЗадайте точность вычисления Пи -> ");
scanf ("%f", &t);
printf("Вычисление Пи с точностью %f\n", t);
  while (el >= t)
    {
     el = (float) 1 / (2*n-1);
     if ((n % 2) == 0)
       p -= el;
     else
       p += el;
     n++;
    }
    p = p*4;
printf("\nЗначение ПИ с точностью %f равно %f.\n", t, p);
printf ("\nПросуммировано %i члена(ов) ряда.\n", n);
}
```

Теория чисел

Теория чисел, или **высшая арифметика** – раздел чистой математики, изучающий свойства натуральных и целых чисел.

Число – одно из основных понятий математики, позволяющее выразить результаты счета или измерения.

Для обозначения чисел существуют условные знаки – ЦИФРЫ.

Арабских цифр 10 – это 0,1,2,3,4,5,6,7,8,9.

А чисел - бесконечно много.

Способ выражать числа знаками (цифрами) называется счислением, нумерацией.

Натуральные числа – 1, 2, 3, 4, 5… и так до бесконечности. Это единица или её сумма с любым другим натуральным числом.

Целые числа – математический объект, представляющий собой множество, получающееся из натуральных чисел N добавлением к ним нуля и отрицательных чисел. Целые числа, упорядоченные по возрастанию образуют бесконечный в обе стороны ряд: ...,−3,−2,−1,0,1,2,3,...

Элементарная теория чисел изучает целые числа без использования методов других разделов математики. Делимость целых чисел, алгоритм Евклида для вычисления НОД и НОК, разложение числа на простые множители, построение магических квадратов, совершенные числа, числа Фибоначчи, малая теорема Ферма, теорема Эйлера, задача о четырёх кубах относятся к этому разделу.

Аналитическая теория чисел для вывода и доказательства утверждений о числах и числовых функциях использует аппарат математического анализа.

Алгебраическая теория чисел рассматривает в качестве алгебраических чисел корни многочленов с рациональными коэффициентами.

Целочисленные алгоритмы

Основные понятия и утверждения целочисленной арифметики

Определение 1. Целое число **a** делится на целое число **b** (**b ≠0**), если существует такое целое число **k**, что **a=k · b**.
В таком случае **b** называют ***делителем*** числа **a**; **a** называют ***кратным*** числа **b**.

Утверждение 1. Если числа **a** и **b** делятся на **c**, то и их сумма **(a+b)**, и их разность **(a-b)** делятся на **c**.

Утверждение 2. Если **a** делится на **c**, а **b** делится на **d**, то их произведение **a * b** делится на **c * d**.

Определение 2. Пусть **a** и **b** – положительные целые числа, **c** - общий делитель чисел **a** и **b**, если **a** делится на **c** и **b** делится на **c**. Среди общих делителей чисел **a** и **b** (не равных одновременно нулю) есть наибольший общий делитель, обозначаемый **НОД(a, b)**.

Утверждение 3. Если **a** делится на **b**, то **НОД(a, b) = b**.

Утверждение 4. **НОД(a, a) = a**.

Утверждение 5. Если **a > b**, то **НОД(a, b) = НОД(a - b, b)**.

Определение 3. Если **НОД(a, b) = 1**, то числа **a** и **b** называются взаимно простыми.

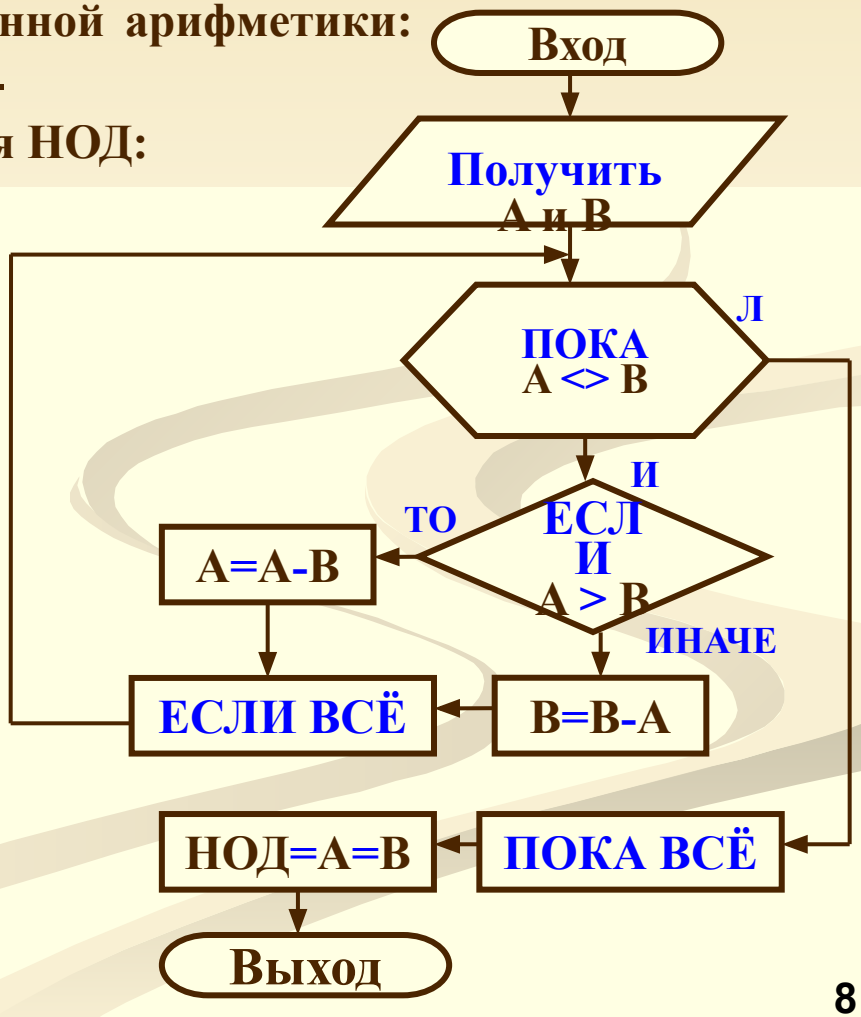
Алгоритм Евклида (1)

Алгоритм Евклида – это метод нахождения наибольшего общего делителя (НОД) двух чисел.

Первая модификация алгоритма Евклида

Основываясь на утверждение 5 целочисленной арифметики: если a > b, то НОД(a, b) = НОД(a - b, b),- составим простейший алгоритм нахождения НОД:

1. задать два числа;
2. если числа равны, то взять любое из них в качестве ответа и остановиться;
3. в противном случае продолжить выполнение алгоритма;
4. определить большее из чисел;
5. заменить большее из чисел разностью большего и меньшего из чисел;
6. повторить алгоритм с шага 2.

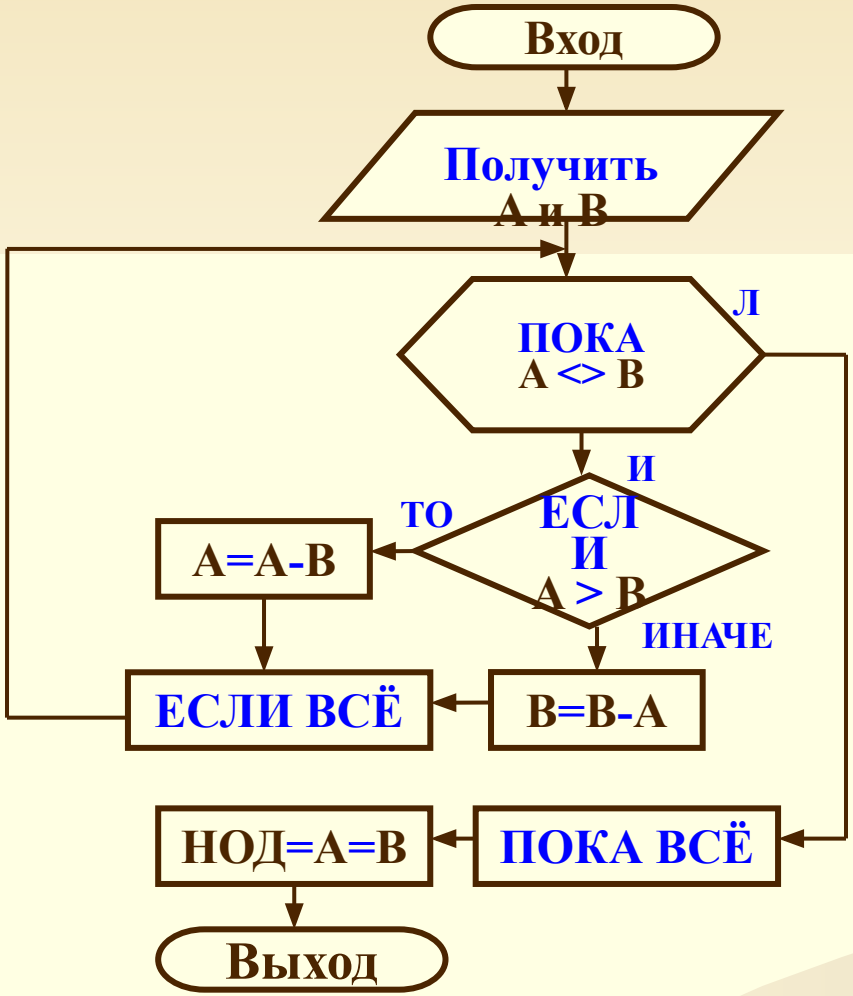


C/C++

Алгоритм Евклида

Первая модификация алгоритма Евклида

Написать на C/C++ программу определения НОД




```
#include <clocale>
#include <stdio.h>
#include <iostream>
using namespace std;
int main()
{  setlocale(LC_CTYPE, "rus");
int   a, ai, b, bi;  // a и b – числа, для поиска НОД
  int   nod;  // nod – наибольший общий делитель
 cout<<"Введите 2-а целых положительных
          числа для определения НОД");
 cout << "a=";  cin >> a;    a = ai;
 cout << "b= ";  cin >> b;   b = bi;
while (ai != bi)
   {
       if ((ai > ib)
          ai -= bi;
        else
          bi -= ai;
   }
   nod = ai;
cout << "\nЗначение НОД(<<a<<","<<b<<")
            равно " << nod;
return 0;
}
```

Алгоритм Евклида (2)

Вторая модификация алгоритма Евклида

Она основана на утверждении 6 целочисленной арифметики: существует такое целое q, что $a = b * q + r$, где остаток от деления r – целое число, удовлетворяющее неравенству $0 \le r < b$, при этом,

$$НОД(a, b) = НОД(r, b).$$

Следовательно, НОД двух чисел – это последний не равный нулю остаток от деления большего числа на меньшее.

Алгоритм имеет вид:

1. задать два числа;
2. проверить в цикле условия A=0 и A=B;
3. если равно, то НОД=B;
4. иначе, определить большее из чисел;
5. если A>B, то заменить A остатком от деления A на B;
6. если A<B, то взаимно заменить значения A и B;
7. выполнить шаг 5;
8. повторить алгоритм с шага 2.

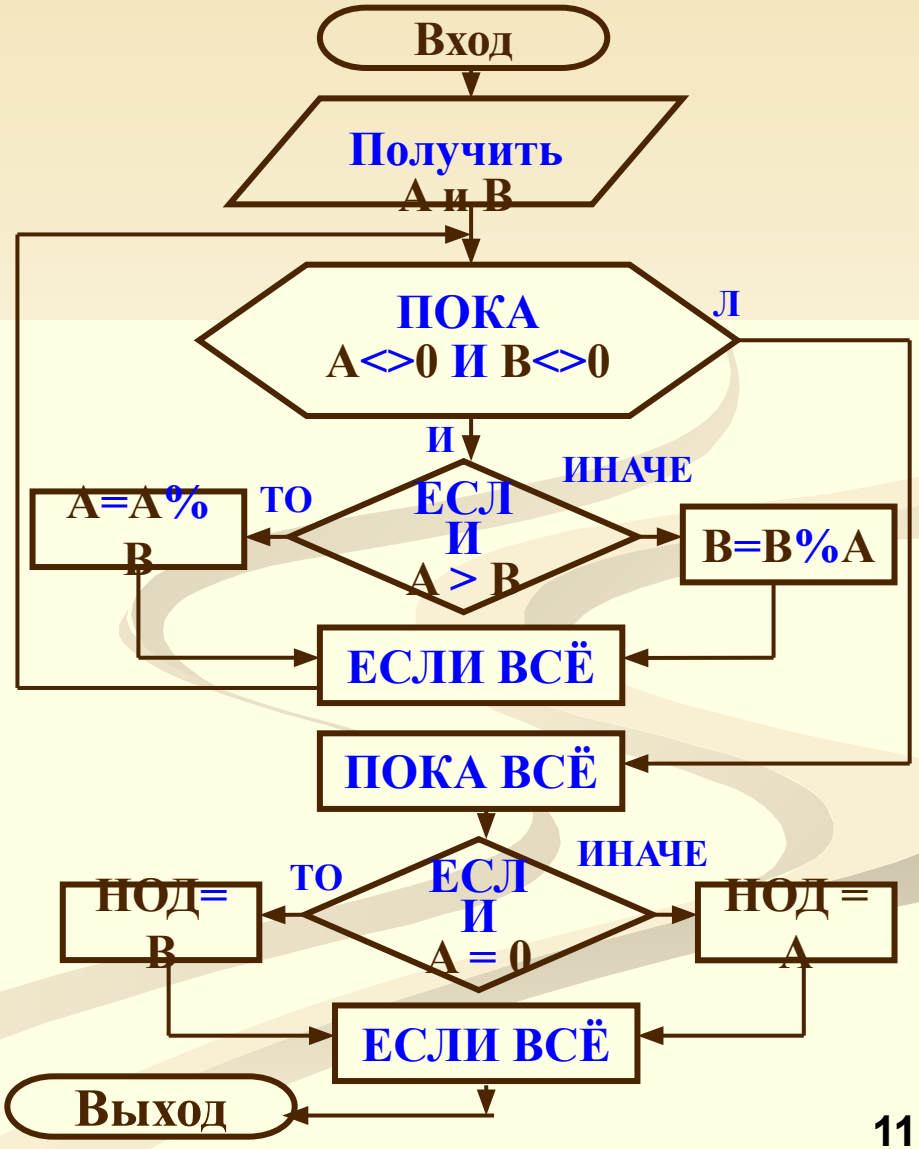
Алгоритм Евклида (2)

C \ C++

Вторая модификация алгоритма Евклида

Написать на С программу определения НОД

```
#include <clocale>
#include <stdio.h>
#include <iostream>
using namespace std;
int main()
{
 setlocale(LC_CTYPE, "rus");
  int   x, xi, y, yi;  // x и y – числа, для поиска НОД
  int   nod;  // nod – наибольший общий делитель
 cout<<"Введите 2-а целых положительных числа
для определения НОД\n";
 cout << "x=";   cin >> x;    xi = x;
 cout << "y= ";  cin >> y;    yi = y;
while ((xi != 0) && (yi != 0)) /*до тех пор, пока
одно из чисел не станет равно нулю*/
    {  if (xi > yi)
                xi %= yi;
         else  yi %= xi;
        if (xi==0)
                nod= yi;
          else nod=xi;    }
cout << "\nЗначение НОД("<<x<<","<<y<<") =
" << nod;
return 0;
}
```



Функции генерации случайных последовательностей

C \ C++

Для моделирования ситуаций, когда требуется, чтобы результат работы программы был случайным в определенных пределах, во многих языках программирования присутствуют встроенные функции, код которых выдает случайные числа. На самом деле числа не совсем случайные, а псевдослучайные, т.к любая программа представляет собой детерминированный алгоритм и с его помощью реализовать случайность невозможно. Алгоритмы реализации псевдослучайных чисел оцениваются по длине последовательности, после которой последовательность начинает повторяться.

Функция стандартной библиотеки языка C (stdlib.h) – rand() генерирует псевдослучайное число на интервале значений от 0 до RAND_MAX (константа, обычно 32767).
Чтобы получить случайные числа от 0 до 9 – используется получение остатка от деления
rand() % 10. Если нам нужны числа от 1 (а не от 0) до 9, то можно прибавить единицу -- rand() % 9 + 1, т.е. генерируется случайное число от 0 до 8, и после прибавления 1 оно превращается в случайное число от 1 до 9.
Для получения при каждом вызове rand() различных случайных последовательностей надо сначала вызвать функцию srand(), которая в качестве аргумента просит какое-то число. И по этому числу уже будет генерироваться случайное число функцией rand():

```
srand(time(NULL)); // time() в биб-ке <time.h>
chislo = rand();
```

Функции генерации случайных последовательностей

C \ C++

Пример:

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
   int main()
{
    int  rand_chislo;
     srand(time(NULL));
for (int i = 0; i <5; i++ )
{
  rand_chislo = 2 + rand() %7;
  printf (" rand_chislo =%d ", rand_chislo);
}
return 0;
}
```

Будем использовать **rand** для заполнения массивов.

Рекуррентные алгоритмы

Суммирование последовательностей

Формула для вычисления суммы n членов последовательности $a_1,a_2,...,a_n$ имеет вид: $S_n = S_{n-1} + a_n$, где $S_1 = a_1$.

Произведение членов последовательностей

Формула для вычисления произведения n членов последовательности $a_1,a_2,...,a_n$ имеет вид: $P_n = P_{n-1} * a_n$, где $P_0 = a_0$.

Вычисление корня квадратного

Рекуррентная формула вычисления (формула Ньютона) выглядит так:

$$X_{k+1} = ½ (X_k + a / X_k), \text{ где } X_0=a.$$

Цикл вычисления завершаем, когда разность между двумя соседними членами последовательности становится меньше заданной константы (определяющей точность вычисления)

Числа Фибоначчи

Это последовательность, в которой каждый следующий член равен сумме двух предыдущих, при этом первые два члена равны 1.
Получаем ряд 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ... , который описывается рекуррентным соотношением

$$F_n = F_{n-1} + F_{n-2}, \text{ при } F_1=F_2=1.$$

Рекурсия

Рекурсия – от латинского слова "recursio" – возвращение.

У термина Рекурсивная функция два значения :

1) Рекурсивная функция – числовая функция числового аргумента, которая в своём определении содержит себя же.

2) Рекурсивная функция – – функция, принадлежащая одному из следующих классов: примитивно рекурсивные функции, общерекурсивные функции, частично рекурсивные функции (в теории вычислимости – алгоритм, допустимые исходные данные которого представляют собой системы натуральных чисел, а возможные результаты применения являются натуральными числами).

В программировании используется термин Рекурсивная функция в первом значении.

Если программа обращается сама к себе как к подпрограмме непосредственно или через цепочку подпрограмм, то это называется рекурсией.

Если подпрограмма p содержит явное обращение к самой себе, то она называется явно рекурсивной. Если подпрограмма p содержит обращение к некоторой подпрограмме q, которая в свою очередь содержит прямое или косвенное обращение к p, то p - называется косвенно рекурсивной.

Рекурсивная программа должна обязательно иметь так называемое терминальное условие, то есть условие при котором программа прекращает рекурсивный процесс, иначе она никогда не остановится.

Рекурсивные подпрограммы

Практическое занятие

Рекурсия – это такой способ организации вспомогательного алгоритма (подпрограммы), при котором эта подпрограмма (процедура или функция) в ходе выполнения ее операторов обращается **сама к себе**.

Рекурсивным называется любой объект, который частично определяется через себя.

Например, приведенное ниже определение двоичного кода является рекурсивным:

<двоичный код> ::= <двоичная цифра> | <двоичный код><двоичная цифра>

<двоичная цифра> ::= 0 | 1

Здесь для описания понятия были использованы, так называемые, металингвистический формулы Бэкуса-Наура (язык БНФ); знак "::=" обозначает "по определению есть", знак "|" – "или".

В рекурсивном определении обязательно должно присутствовать ограничение, граничное условие, при выходе на которое дальнейшая инициация рекурсивных обращений прекращается.

Рекурсия

Пример рекурсивного алгоритма –
вычисление суммы K членов ряда арифметической прогрессии:

```
/* Вычисление суммы  K  членов ряда арифметической прогрессии
 K - количество суммируемых членов ряда,
 N-шаг прогрессии,
 FS - значение первого члена ряда  */
   int SUMpr(int K, int FS, int N)        /* рекурсивная функция вычисления суммы
                                             членов ряда */
    {
     if(K==1) return FS;
     return FS+(K-1)*N+SUMpr(K-1,FS,N);    // рекурсивное выражение
    }
int main()
{
  int  n,arg,ras;
    cout<<"Введите количество суммируемых членов ряда n = ";
    cin>>n;
    cout<<"Введите значение первого члена ряда arg = ";
    scanf ("%d", arg);
    printf ("Введите шаг прогрессии ras = ");
    scanf ("%d", ras);
    cout<<"\nСумма членов ряда = "<< SUMpr(n,arg,ras);
}
```

Рекурсивные подпрограммы

C \ C++

Практическое занятие

Пример 1. Определение факториала.

С одной стороны, факториал определяется так: *n*!=1*2*3*...n*.**

С другой стороны,

$$n! = \begin{cases} 1, & \text{если } n \le 1, \\ (n-1)! \times n, & \text{если } n > 1. \end{cases}$$

Граничным условием в данном случае является *n*<=1.

```
/* Функция  Факториал */
double  Factorial (int  N)
 {
    double  F;
   if (N<=1) F=1.0;
      else F=Factorial(N-1)*N;
   return F;
 }
```

Рекурсивные подпрограммы

C \ C++

Практическое занятие

Пример 2. Количество цифр K в заданном натуральном числе n

Определим функцию *K(n)*:

$$K(n) = \begin{cases} 1, & \text{если } n < 10, \\ K(n/10) + 1, & \text{если } n >= 10. \end{cases}$$

```
/* Функция «Количество цифр целого числа»
*/
int K (int N)
 {
    int Kol;
   if (N <10 )
       Kol =  1;
   Kol = K ((int) N/10)+1;
  return Kol;
 }
```

Рекуррентные алгоритмы
Вычисление корня квадратного

Пример 3. Рекуррентная формула вычисления квадратного корня (формула Ньютона):

$$X_{k+1} = \frac{1}{2}\left(X_k + a / X_k\right),$$

где **a** – число, X_0 – начальное приближение результата (например, можно $X_0=a$).

Цикл вычисления завершаем, когда разность между двумя соседними членами последовательности становится меньше заданной константы (определяющей точность вычисления)

Вычисление циклом

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
void main()     {  clrscr();
  float A;          // число из которого извлекается корень
  float X;          // член ряда -  значение корня квадратного
  float Xprev;  // предыдущий член ряда, приближенный результат
  int   i=1;        // количество итераций
  float t;          // заданная точность
do   {  cout<<"Введите число A > 0 \n";  cin >> A;
   cout<<"Введите точность t > 0 \n";   cin >> t;  }
while ((A<=0) && (t<=0));
    X = A/2;
  do  {  Xprev=X;  X=(Xprev+A/Xprev)/2;  i++;  }
  while (abs(Xprev-X)>t);
 cout << "\nЗначение квадратного корня числа "
<<A << " = " << X <<"\n";
cout << "Количество итераций - " << i;
getch();
}
```

Вычисление рекурсией

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
   float  sqrtnewton (float N, float prev, float pr)
     /* рекурсивная функция вычисления квадратного корня*/
     {  if(abs(N/prev-prev) <  pr)
          return (N/prev+prev)/2;
          return sqrtnewton (N, (N/prev+prev)/2, pr);
     }
void main()
 {  clrscr();
  float A;          // число из которого извлекается корень
  float X;          // член ряда -  значение корня квадратного
  float Xprev;  // предыдущий член ряда, приближенный результат
  float  t;         // заданная точность
do   {  cout<<"Введите число A > 0 \n";  cin >> A;
   cout<<"Введите точность t > 0 \n";   cin >> t;  }
while ((A<=0) && (t<=0));    Xprev = A/2;
   X =  sqrtnewton( A, Xprev, t);
 cout << "Значение квадратного корня числа  "
<< A << " = " << X <<"\n";    getch();
 }
```

Рекурсивные подпрограммы

C \ C++

Практическое занятие

Пример 4. Вычислить сумму элементов линейного массива.

При решении задачи используем следующее рассуждение: сумма равна нулю, если количество элементов равно нулю, и сумме всех предыдущих элементов плюс последний, если количество элементов не равно нулю.

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

 int summa (int N,  int a[100]);
   int i,n, a[100] ;
void  main()     // Основная программа
{
 printf ("\nК-во элементов массива = ");
 scanf("%d",&n);
 printf("\nВ массив введено %d чисел:\n", n);
 srand(time(NULL));
for (i=0; i<n; i++)
   { a[i] =-10+rand()%21;
     printf("%d", a[i]);
    }
  printf ("Сумма: %d", summa (n-1, a)) ;
}
//  Рекурсивная  функция
int summa(int N, int a[100])
  {
     if (N==0) return 0;
          else return a[N]+summa (N-1, a);
  }
```

Рекурсивные подпрограммы

Практическое занятие

C

Пример 5. Является ли заданная строка палиндромом.

Идея решения заключается в просмотре строки одновременно слева направо и справа налево и сравнении соответствующих символов. Если в какой-то момент символы не совпадают, делается вывод о том, что строка не является палиндромом, если же удается достичь середины строки и при этом все соответствующие символы совпали, то строка является палиндромом. Граничное условие – строка является палиндромом, если она пустая или состоит из одного символа.

```
#include <stdio.h>
#include <conio.h>
#include <strinq.h>
   char s[100];
   int pal(char  s [100]);
 void  main ()    // Основная программа
{  clrscr() ;
   printf("\nВведите строку: ");  gets(s);
      if (pal(s))
         printf("Строка – палиндромом");
       else printf("Строка  – не палиндром");
 int pal(char  s[100])    //  Рекурсивная функция
{ int L;  char  s1[100];
  if (strlen(s)<=1)     return 1;
    else {   L=s[0]==s [strlen (s)-1 ];
             strncpy(sl,  s + 1, 'strlen(s)-2) ;
             s1 [strlen (s)-2] = '\0';
             return   L && pal(s1);   }
}
```