УНИВЕРСИТЕТ СИНЕРГИЯ | БОЛЬШЕ ЧЕМ ОБРАЗОВАНИЕ

Основа алгоритмизации и программирования «Основы структурного программирования»

Лекция

Кафедра цифровой экономики

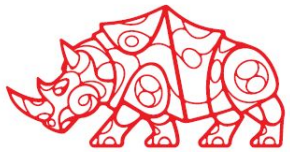
Гринева Е.С., преподаватель

2 ноября 2022 г.

УНИВЕРСИТЕТ СИНЕРГИЯ

Цель занятия:

- **Изучить:**
- Основные управляющие структуры.
- Основные структуры данных.
- Методология программирования "сверху-вниз".
- Проектирование модулей. Модуль RAT. Оформление модуля.



Структурное программирование - это технология проектирования программ, базирующаяся на строгом определении средств языка и методов их использования. К средствам языка относятся стандартные типы данных и операторы управления вычислениями.

1. Основные управляющие структуры

а) Алгоритм Евклида:

```
Program Evclid;
    Var a, b, u, v, w: Integer;
  Begin
    Read(a,b);
    u := a; v := b;
    While u <> v do begin
      w := u - v;
      If w > 0 then u := w else v := -w;
    end;
   Write(u)
  end.
```

б) Приближенное решение уравнения методом деления пополам:

```
        Program EquationSol;
     Const Eps = 1e-4;
       Var a, b, u, v, w: Real;
      Begin
       Read(a, b);
       u := a; v := b;
       While u - v >= Eps do begin
         w := (u + v)/2;
         If f(u)*f(w) > 0  then u := w  else v := w;
      end;
Write(u)
End.
```

в) Разделение массива на два по барьерному элементу 0:

```
Program Partition ;
  Const n = 16;
    Var f, g, h : array [1..n] of Integer;
        x, b: Integer;
        i, j, k: Integer;
  Begin
    { ввод массива f }
    i := 1; j := 1; k := 1;
    While f[i] <> 0 do begin
      x := f[i];
     If x > 0
       then begin
         g[j] := x; j := j + 1; i := i + 1 end
       else begin h[k] := x; k := k + 1; i := i + 1 end;
    end;
    Write(j, k)
  End.
```

Программа Evclid работает с целыми числами. Предметная область программы EquationSol - область вещественных чисел. Третья программа - Partition - обрабатывает массивы, причем тип компонент массива не имеет существенного значения. Несмотря различие в предметных областях, между этими программами существует сходство, играющее большую роль для понимания сути программирования.

В самом деле, раздел операторов каждой из этих программ может быть описан так:

Begin

< процедура ввода >;

< последовательность простых операторов >;

while <условие> do begin

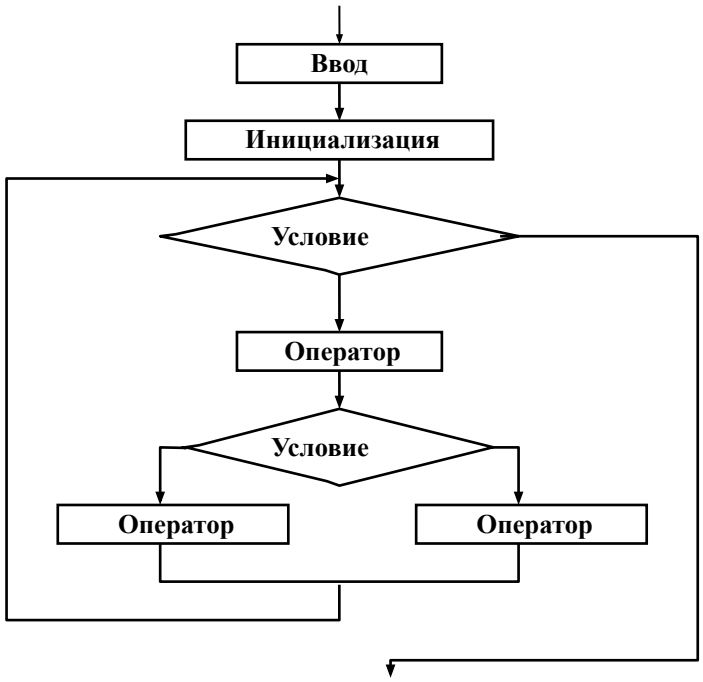
<оператор>;

if <условие> then < оператор > else < оператор > ;

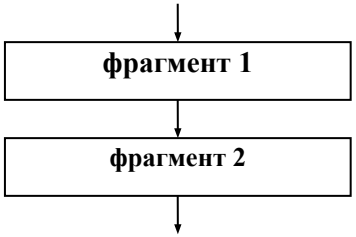
end;

< процедура вывода >

End.

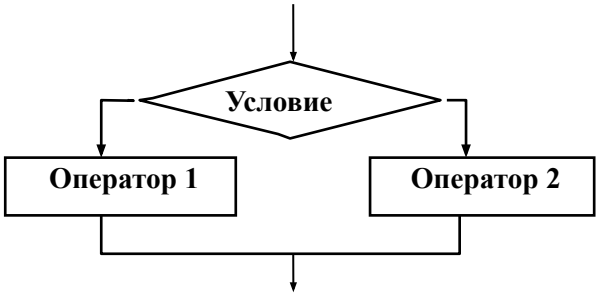


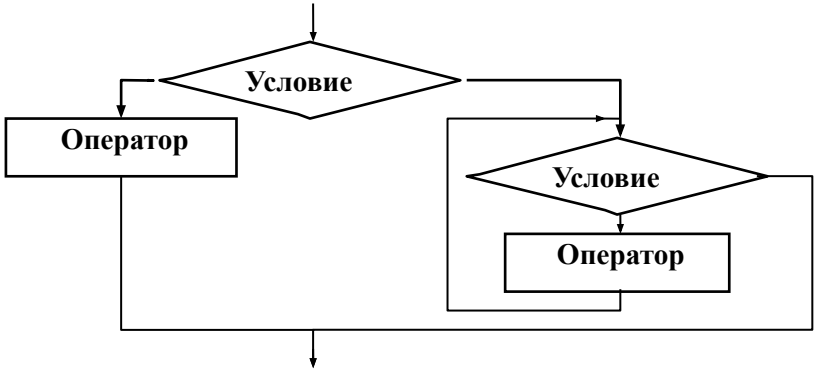
последовательное выполнение



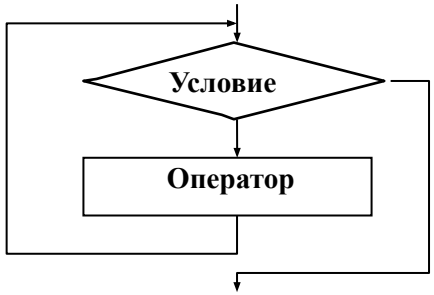
Программирование осуществляется комбинированием основных управляющих структур. Например, комбинирование ветвления и повторения приводит к блок-схеме

ветвление





повторение



Управление в любом алгоритме может быть реализовано в виде комбинации основных управляющих структур.

В реальных языках структурного программирования применяют и другие управляющие структуры, каждая из которых может быть отнесена к одному из трех основных типов. Например, в языке Pascal ветвления - условный оператор, короткий условный оператор, оператор варианта; повторения - оператор цикла с параметром, оператор цикла с предусловием, оператор цикла с постусловием.

Отметим, что оператор перехода Goto не включен ни в список основных, ни дополнительных управляющих операторов.. Бесконтрольное применение этого оператора приводит к тому, что программа теряет свойства, указанные выше. Поэтому структурное программирование часто называют программированием без Goto

Обратим внимание на совокупности данных, с которыми работают эти программы. Данные определены в разделах типов, переменных и констант:

```
Program Evclid;
    Var a, b, u, v, w: Integer;
Program EquationSol;
    Const Eps = 1e-4;
    Var a, b, u, v, w: Real;
Program Partition ;
    Const n = 16;
    Var f, g, h : array [1..n] of
Integer;
    x, b: Integer;
    i, j, k: Integer;
```

Теория структур данных подобна теории структур управления. Именно, существуют основные (стандартные) структуры, каждая из которых определяет один из способов объединения данных в структуру. Данные простых типов при этом играют роль кирпичиков, из которых строится все здание.

Структуры данных в языке Pascal, отражают идеологию структурного программирования. Простые данные: целые числа, вещественные числа, символы, логические значения, данные - имена. Способы структурирования данных: массивы, записи, файлы, множества, ссылки.

определения типов

```
Word = Array [1..20] of Char;
Man = Record
        Name: Word;
        Age: Integer
      end;
Notebook = array [1..n] of Man;
```

означает массив из записей, первое поле которых - массив из двадцати символов, а второе - целое число.

Применение в программировании концепции структур данных и управления требует специфической методологии процесса разработки программ. Этот подход называют программированием "сверху-вниз". Суть метода - в следующем:

1. Процесс разработки программы состоит из последовательности шагов, на каждом из которых программист

* уточняет структуру программы;
* уточняет структуру данных.

2. Уточнение структуры управления заключается в определении того оператора управления, который следует использовать для реализации алгоритма и тех элементов оператора, которые участвуют в управлении (условий, границ циклов и т.п.)

3. Уточнение структуры данных состоит в определении и описании данных, обрабатываемых выбранным оператором.

4. Определение структуры данных программы начинается с описания входа-выхода, т.е. с определения структуры исходных и выходных данных.

5. При работе пользуются принципом минимальной необходимости: уточнение осуществляют лишь с той степенью точности, которая необходима на данном шаге.

6. При разработке нескольких фрагментов программы предпочтительнее уточнить каждый из них на один шаг, а не один - полностью (особенно тогда, когда это уточнение связано с принципиальными решениями).

7. Основными структурными единицами программы являются процедуры и функции. Каждую относительно самостоятельную подзадачу оформляют как подпрограмму (процедуру или функцию).

Модуль RAT содержит все средства, обеспечивающие применение в программах рациональных чисел. Сюда входят:

* описание типа Rational, констант типа Rational;
* процедуры ввода-вывода рациональных чисел;
* функции преобразования числовых типов и типа Rational;
* функции арифметических операций и сравнений рациональных чисел;
* средства, используемые для реализации вышеуказанных процедур (внутренние средства модуля).

Проектирование начнем с определения понятия рационального числа. Рациональные числа - это дроби вида Num/Den, где Num - числитель, а Den - знаменатель. Для обеспечения корректности и единственности представления рационального числа в виде пары <Num, Den> (дроби Num/Den) потребуем выполнения следующих ограничений:

Den > 0, НОД(Num, Den) = 1, Если Num = 0, то Den = 1

Такое представление мы будем называть канонической формой. Действия над дробями естественно и удобно реализовать в виде функций. Но функции языка Pascal возвращают скалярные значения. уточним тип Rational как ссылку на запись:

```
Type
   Rational = ^RatValue;
   RatValue = Record
   Num, {числитель }
   Den: LongInt {знаменатель}
End;{Тип LongInt - один из целочисленных типов в TP-6. Множество значений определено константой MaxLongInt = 2^31 - 1.}
```

Ниже описаны некоторые (далеко не все) средства RAT, необходимые для работы с рациональными числами. Ключевой функцией модуля является функция CanRat, приводящая дробь к канонической форме. В свою очередь, CanRat использует целочисленную функцию GCD, вычисляющую наибольший общий делитель (НОД) двух натуральных чисел.

A / B = A div НОД(A, B) / B div НОД(A, B)

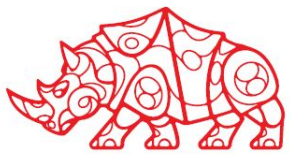
```
{-----------НОД двух натуральных чисел-----------------------}
    Function GCD(u, v: LongInt): LongInt;
       Begin
        While (u <> 0) and (v <> 0) do
          If u > v
            then u := u mod v else v := v mod u;
            If u = 0 then GCD := v else GCD := u
         End;
```

```
{-----------канонизация рационального числа-------------------}
    Function CanRat(X: Rational): Rational;
      Var u, v, w: LongInt;
    Begin
      u := X^.Num; v := X^.Den;
      If  v = 0
          then CanRat := Nil {дробь с нулевым знаменателем}
          else begin
             If v < 0 {знаменатель > 0}
              then begin u := -u; v := -v end;
              w := GCD(Abs(u), v);
        If  w <> 1                          then {сокращение числителя и знаменателя}
                begin u := u div w; v := v div w end;
                New(R);
                Z^.Num := u;
                Z^.Den := v ;
                CanRat := Z;
          end
    End;
```

{ВВОД-ВЫВОД}

```
Procedure RatInp(var X: Rational);
{Ввод}
    Var Ch: Char;
  Begin
      New(X);
                Write('ввод  числителя  ');
Read(X^.Num);
          Ch := ReadKey;
          If Ch = '/'
           then begin
                  Write('ввод знаменателя ');
Read(X^.Den);
             X := CanRat(X)
           end
           else X^.Den := 1
        End;
```

```
Procedure RatPrint(X: Rational);
    {Вывод}
  Begin
      If  X = Nil
   then Write('Деление на ноль')
   else begin
     Write(X^.Num);
     If X^.Den <> 1
            then  begin  Write('/');
Write(X^.Den) end;
    end;
     Writeln
  End;
```

{АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ}

```
Function AddRat(X, Y: Rational): Rational;          {Сложение}
  Begin
    New(Z);
      Z^.Num := X^.Num*Y^.Den + X^.Den*Y^.Num;
    Z^.Den := X^.Den*Y^.Den;
    AddRat := CanRat(Z)
  End;
Function SubRat(X, Y: Rational): Rational;          {Вычитание}
  Begin
    New(Z);
      Z^.Num := X^.Num*Y^.Den - X^.Den*Y^.Num;
    Z^.Den := X^.Den*Y^.Den;
    SubRat := CanRat(Z)
  End;
```

```
Function MultRat(X, Y: Rational): Rational;         {Умножение}
  Begin
    New(Z);
    Z^.Num := X^.Num*Y^.Num;
    Z^.Den := X^.Den*Y^.Den;
    MultRat := CanRat(Z)
  End;
Function DivRat(X, Y: Rational): Rational; {Деление}
  Begin
    If Y^.Num = 0
     then DivRat := Nil
     else begin
       New(Z);
       Z^.Num := X^.Num*Y^.Den;
       Z^.Den := X^.Den*Y^.Num;
       DivRat := CanRat(Z)
     end
  End;
```

{ПРЕОБРАЗОВАНИЯ ТИПОВ}

```
Function RatReal(X:Rational): Real;
{Рациональное в вещественное}
  Begin
    RatReal := X^.Num/X^.Den
  End;

Function IntRat(X: LongInt): Rational; {Целое в рациональное}
  Begin
    New(Z);
    Z^.Num := X; Z^.Den := 1;
    IntRat := Z
  End;
```

Заголовок модуля имеет вид:

Unit < Имя модуля >.

Имя модуля должно совпадать с именем файла, содержащего этот модуль.

Интерфейсная часть имеет вид:

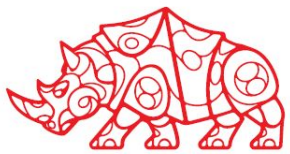
Interface

<

* Описания констант, меток, типов и переменных, доступных для для использования внешними программами;
* Заголовки процедур и функций, доступных для использования внешними программами;
* Uses-директивы с именами модулей, доступных для использования внешними программами.

>

Все доступные извне средства модуля должны быть описаны в интерфейсе этого модуля.



Реализационная часть имеет вид:

Implementation

<

* Описания всех средств модуля, скрытых от внешних программ;
* Uses-директивы с именами модулей, используемых в этом модуле и скрытых от внешних программ;
* Полные описания всех процедур и функций модуля.

>

Инициализационная часть - раздел операторов модуля. Он выполняется перед выполнением внешней программы, использующей модуль. Во внешнюю программу модуль включается директивою Uses.

Домашние задание (Задание на самоподготовку)

1 выполнить задания несделанные во время практики