Методы вычислений

---

1. Структурное программирование на АЯ Pascal

Управляющие конструкции АЯ Pascal структурированные и это создаёт прекрасные предпосылки для использования метода пошаговой детализации. Для выражения абстрактных понятий в АЯ Pascal существует специальный тип управляющих конструкций - процедуры. Их изучение требует времени, а пока воспользуемся так называемым псевдокодом, который позволяет использовать уже известные нам управляющие конструкции с описанием вложенных в них операций на естественном языке. Рассмотрим пример применения псевдокода.

Задача. Определить среднее по значению из трёх различных чисел входа.

Практически сразу мы можем написать такую абстрактную программу:

```
program Middle;
var a,b,c,r: Integer;
begin
ReadLn(a,b,c);
{1. r := Среднее по значению из переменных
    a, b и c с различными значениями.};
WriteLn(r);
end.
```

Далее детализируем блок 1:

```
if a > b then
    {1.1. r := Среднее по значению из переменных a, b и c
          с различными значениями. Причём a > b.}
else
    {1.2. r := Среднее по значению из переменных b, a и c.
     Причём b > a.}
```

Заметим, что блоки 1.1 и 1.2 с с точностью до обозначений переменных одинаковы, что позволяет нам в последующем сосредоточиться на детализации одного блока. Пусть это будет 1.1.

Так как известно, что a > b, то возможны следующие варианты соотношения значений переменных:
1) c > a > b, 2) a > c > b, 3) a > b > c.
Анализ этих соотношений приводит к следующей детализации блока 1.1:

```
if c > a then
    r := a
else
```

```
    {1.1.1. r := max(b,c)}
```

Детализация блока 1.1.1 очевидна. Далее выполняем необходимые подстановки и собираем блок 1.1:

```
if c > a then
    r := a
else
    if b > c then r := b else r := c
```

Блок 1.2 получаем из блока 1.1 перестановкой местами имён переменных a и b. Далее собираем блок 1:

```
if a > b then
    if c > a then
        r := a
    else
        if b > c then r := b else r := c
else
    if c > b then
        r := b
    else
        if a > c then r := a else r := c
```

Теперь остаётся поставить этот блок на своё место и мы получим почти готовую программу. Останется только добавить некоторые технические детали улучшающие наглядность ввода и вывода данных.

2. Табуляция функций

Пусть некоторая функция y = f(x) должна быть протабулирована на отрезке (a,b) с шагом h = (b - a) / n, где n - количество интервалов табуляции. Эта задача может быть решена с помощью абстрактной программы Tabula:

```
program Tabula;
var a,b,x,y,h: Real;
    n,i:     Integer;
begin
ReadLn(a,b,n);
h:=(b-a)/n;
i:=0; x:=a;
while i<=n do
    begin
    {1. y:=f(x)};
    WriteLn(x,y);
    i:=i+1; x:=x+h;
    end;
end.
```

Остаётся вместо блока 1 подставить фрагмент программы вычисления значения реальной функции. Пример детализации блока 1 для вычисления значения функции y = Sin(x)/x:

```
if Abs(x)<1e-5 then
    y:=1
else
    y:=Sin(x)/x
```

Здесь условие Abs(x) < 1e-5 определяет близость переменной x к нулю и введено для исключения возможности появления ошибки "Деление на нуль". В рабочей программе, естественно, надо предусмотреть необходимые меры по наглядному представлению данных на её входе и выходе. Окончательный вариант программы Tabula:

```
program Tabula;
const eps = 1e-5;
var a,b,x,y,h: Real;
    n,i:     Integer;
begin
Write('Введите a,b и n: ');
ReadLn(a,b,n);
h:=(b-a)/n;
i:=0; x:=a;
WriteLn('x':8,'y':8);
WriteLn('----------------');
while i<=n do
    begin
    if Abs(x)<eps then
        y:=1
    else
        y:=Sin(x)/x;
    WriteLn(x:8:3,y:8:3);
    i:=i+1; x:=x+h;
    end;
end.
```

3. Приближённое интегрирование функций

Определённый интеграл от функции y = f(x) на интервале интегрирования [a,b] можно приближенно найти методом прямоугольников по формуле:
S = ((y[1]+y[2] + ... + y[n]) * h, где y[i] = f(x[i]), i = 1..n и h = (b - a) / n. Здесь n - число равных отрезков, на которые делится интервал интегрирования [a,b], причём x[1] = a, x[i] = x[i - 1] + h для i = 2..n.

За основу можно взять следующую абстрактную программу:

```
program Integral;
var a,b,x,y,h,S: Real;
    n,i:     Integer;
begin
ReadLn(a,b,n);
h:=(b-a)/n;
i:=1; x:=a; S:=0;
while i<=n do
    begin
    {1. y:=f(x)};
    S:=S+y;
    i:=i+1; x:=x+h;
    end;
S:=S*h;
WriteLn(S);
end.
```

4. Рекуррентные вычисления

Рекуррентные вычисления основаны на многократном повторном использовании одной и той же формулы. Такие вычисления называют также итерационными. Для их реализации используют циклические программы. Известна итерационная формула для вычисления квадратного корня из x: y[0] = x, y[i+1] = (y[i] + x / y[i]) / 2 для i > 0. Вычисления можно прекратить, когда будет достигнута необходимая точность: Abs(y[i+1] - y[i]) <= Eps. Следующая программа решает поставленную задачу.

```
program MySqrt;
const Eps =1e-5;
var x:  Real;
    y:  Real; {Текущее значение}
    ys: Real; {Следующее значение}
begin
Write('Entry x: ');
ReadLn(x);
ys:=x;
repeat
    y:=ys;
    WriteLn('y = ', y);
    ys:=(y + x/y)/2;
until Abs(ys-y)<Eps;
WriteLn('MySqrt(', x,') = ', ys);
WriteLn('  Sqrt(', x,') = ', Sqrt(x));
end.
```

5. Суммирование степенных рядов

Рассмотрим на примере разложения функции y = Exp(x) в степенной ряд Тейлора:
y = 1 + x + x^2/2! + x^3/3! + ... + x^i/i! + ...
Обозначим через a[i] i-ый член ряда. Для степенных рядов a[i] обычно легко выражается через a[i-1] по общей формуле a[i] = K*a[i-1]. Определим K для нашего примера. K = a[i] / a[i-1] = (x^i / i!) / ((x^(i-1) / (i-1)!) = x / i . Отсюда следует, что a[i] = x * a[i-1] / i, то есть мы получили итерационную формулу для вычисления текущего члена ряда. В общем случае мы приходим к следующей абстрактной программе вычисления суммы степенного ряда:

```
program SumRow;
var s,a,x: Real;
    i:     Integer;
begin
ReadLn(x);
i:=1;
{1. a:=A1};
{2. s:=S0};
while i < 20 do
    begin
    i:=i+1;
    {3. a:=K*a};
    s:=s+a;
    end;
WriteLn(s);
end.
```

В этой программе A1 - это значение первого члена ряда, для которого справедливо a[2] = K*a[1]; S0 - сумма начальных членов ряда по A1 включительно. Для нашего примера абстрактные операторы программы SumRow будут иметь такую детализацию: 1.: a := x ; 2.: s := 1 + x ; 3.: a := a*x / i .

6. Цепные дроби

В качестве примера приведём представление функции y = tg(x) в виде цепной дроби:
y = x / (1 - x^2 / (3 - x^2 / (5 - x^2 / (7 - x^2 / (9 - ...))))) .
Составим программу MyTan для приближённого вычисления тангенса на основе этой цепной дроби. Если представить i-ый знаменатель дроби как d[i] - v[i], то соответственно (i-1)-ый : d[i-1] - v[i-1] . Очевидно, что d[i-1] = d[i] - 2 , а v[i-1] = x^2 / d[i] - v[i] . Полагая d[5] = 9 и v[5] = 0 последовательно получим
d[4] = 7 и v[4] = x^2 / 9,
d[3] = 5 и v[3] = x^2 / (7 - x^2 / 9),
d[2] = 3 и v[2] = x^2 / (5 - x^2 / (7 - x^2 / 9)),
d[1] = 1 и v[1] = x^2 / (3 - x^2 / (5 - x^2 / (7 - x^2 / 9))).
Таким образом мы видим, что рекуррентные соотношения для d и v работают правильно. Очевидно, что окончательный результат можно определить по формуле x / (d[1] - v[1]) . Воплотим теперь этот подход в программе MyTan:

```
program MyTan;
var x,x2,xg,y,v: Real;
    d: Integer;
begin
Write('Entry angle[grad]: ');
ReadLn(xg);
x:=xg*Pi/180;
x2:=sqr(x);
d:=9;
v:=0;
repeat
    v:=x2/(d-v);
    d:=d-2;
until d = 1;
y := x/(d-v);
WriteLn('MyTan(', x, ') = ', y);
WriteLn('  Tan(', x, ') = ', Sin(x)/Cos(x));
end.
```

При желании увеличить точность вычислений по этой программе достаточно увеличить начальное значение для d. Но при этом, очевидно, увеличится и время вычислений.