Метод пошаговой детализации разработки алгоритмов. Особенности использования массивов в качестве параметров

Лекция 10

Метод пошаговой детализации разработки алгоритмов

При использовании метода пошаговой детализации первоначально продумывается и фиксируется множество данных и результатов алгоритма без детальной проработки отдельных частей.

Задачу разбивают на автономные части, каждая из которых существенно проще исходной. Может оказаться необходимым повторять процесс детализации многократно, но это определяется только сложностью решаемых задач. Конечным уровнем детализации алгоритма можно считать такой, при котором в алгоритме нет действий более крупных, чем: обращение к готовому алгоритму; вычисление арифметического выражения и присваивание значения переменной; сравнение арифметических выражений (или переменных); ввод (вывод) данных и т.п.

Использование подпрограмм позволяет вести проектирование и разработку приложения сверху вниз — такой подход называется нисходящим проектированием.

Сначала выделяется несколько подпрограмм, решающих самые глобальные задачи (например, инициализация данных, главная часть и завершение), потом каждый из этих модулей детализируется на более низком уровне, разбиваясь в свою очередь на небольшое число других подпрограмм, и так происходит до тех пор, пока вся задача не окажется реализованной.

Такой подход удобен тем, что позволяет человеку постоянно мыслить на предметном уровне, не опускаясь до конкретных операторов и переменных. Кроме того, появляется возможность откладывать реализацию некоторых подпрограмм, пока не будут закончены другие части. Немаловажно, что небольшие подпрограммы проще отлаживать, что повышает надежность всей программы.

*Массивы как параметры подпрограмм

При использовании массивов в качестве параметров подпрограмм есть одна особенность — тип массива должен быть обязательно описан заранее в разделе типов, причем типы фактического и формального параметров должны быть одинаковыми.

Чтобы снять ограничения размерности для параметров — одномерных массивов, можно использовать открытые массивы.

Открытый массив — это конструкция описания типа массива без указания типа индексов, например:

array of real; **или** **array of integer;**

Такое определение массива в Turbo Pascal возможно только при описании формальных параметров подпрограммы.

Применяя открытые массивы, *следует знать:*

- индексы параметров, описанных как открытые массивы, всегда начинаются с нуля (а не с единицы);
- размерность открытых массивов (количество индексов) всегда равна 1 — за этим следит компилятор.

Реальное количество элементов массива можно передать через дополнительный параметр.

*Текстовые файлы

Эффективным способом хранения и обработки большого количества однотипных переменных в программе являются массивы. Однако у них есть один недостаток: массивы не пригодны для долговременного хранения информации.

Файл — это набор данных, хранящихся во внешней памяти компьютера (на жестком диске, компакт-диске и т. п.) под заданным именем.

Текстовый файл — это последовательность символов, сгруппированных в строки, заканчивающиеся специальным символом **eoln.**

Текстовый файл можно подготовить и прочитать при помощи обычного текстового редактора, который не использует операции форматирования текста.

Объявление текстового файла в программе осуществляется заданием файловой переменной типа text в виде:

var ФайловаяПеременная : text;

С файлами текстового типа можно работать только путем *последовательного продвижения* по файлу, прочитывая или записывая одну строку за другой, *начиная с первой*.

Процедуры и функции для работы с текстовыми файлами

Процедура **assign (Файловаяпеременная, ИмяФайла)** – предшествует другим процедурам, т. к. ставит в соответствие физическому файлу (**ИмяФайла**) на внешнем устройстве логический файл – файловую переменную (связывает их). Процедуру **assign** недопустимо использовать для уже открытого файла.

ИмяФайла при необходимости должно содержать путь доступа к этому файлу. При этом **ИмяФайла** – строковая величина, т.е. должна быть заключена в апострофы.

После вызова **assign** связь файловой переменной с физическим файлом существует до тех пор, пока не будет выполнен другой **assign** для данной файловой переменной.

Процедура **reset (Файловаяпеременная)** — открывает *существующий* файл на *чтение* и ставит указатель на начало *первого* элемента файла.

Если при чтении файла возникнет необходимость вернуть указатель в его начало, достаточно будет просто применить процедуру **reset** к этому файлу еще раз.

Процедура **rewrite (Файловаяпеременная)** — создает **и** открывает *новый (выходной)* файл для последующей *записи* данных.

Обратите внимание — использование **rewrite** требует особой аккуратности. Если физический файл с указанным именем уже существует, то он *удаляется*, и на его месте создается новый *пустой* файл *с тем же именем*.

Процедура **close (ФайловаяПеременная)** закрывает файл после того, как завершена его обработка. В противном случае может произойти *потеря данных*. При закрытии физический файл обновляется и его автоматически завершает *символ конца файла*.

Процедура **append (ФайловаяПеременная)** открывает существующий файл для *дозаписи*. Указатель ставится не в начало, а в *конец файла*, куда и будут дописываться новые компоненты.

Процедура append применима *только* к текстовым файлам. Если файл ранее уже был открыт с помощью reset или rewrite, использование append приведет к закрытию этого файла и к повторному открытию, но уже для добавления элементов.

Процедуры чтения:

read(ФайловаяПеременная, x1, x2,...,xN);

readln(ФайловаяПеременная);

readln(ФайловаяПеременная, x1, x2,...,xN);

где **x1, x2,... ,xN** – список *ввода*, содержащий имена переменных, значения которых процедура **read** считывает из текстового файла, *начиная чтение с элемента, на который установлен текущий указатель*.

При выборе между **read** и **readln** для выполнения чтения из текстового файла необходимо помнить, что после считывания последней переменной списка ввода процедура **readln** пропускает оставшуюся часть строки до **eoln** и обязательно переходит к следующей строке текстового файла, даже если из текущей строки прочитаны еще не все символы. Следующее обращение к очередной процедуре **read** или **readln** начнется с первого символа новой строки. В отличие от **readln,** процедура **read** не делает после считывания пропуск до следующей строки. Поэтому она непригодна для чтения последовательности строк, поскольку не дает возможности продвинуться дальше первой строки.

Процедуры записи:

write(ФайловаяПеременная, y1, y2,...,yN);

writeln(ФайловаяПеременная);

writeln(ФайловаяПеременная, y1, y2,...,yN);

где **y1, y2,... ,yN** — *список вывода*, содержащий выводимые выражения, значения которых должны быть записаны в файл, *начиная с позиции текущего указателя*. Возможно использование формата вывода. Файл должен быть открыт для вывода.

Процедура **writeln** выполняет те же действия, что и **write**, и дополнительно вставляет признак конца строки **eoln**.

Функция **eoln(Файловаяпеременная)**, принимающая значение **true**, если указатель текущей позиции находится на маркере конца строки, иначе — **false**.

Функция **eof (ФайловаяПеременная)** — выполняет проверку, не достигнут ли конец файла (End Of File) при чтении из него данных. Функция возвращает true, если при чтении обнаружен конец файла.

Пример. Написать программу для нахождения суммы элементов каждой строки матрицы. Ввод и вывод данных в текстовые файлы.

```
type
  mas2=array [1..20,1..20] of integer;   mas1=array [1..20] of integer;
var
  result,isxdan:text;   a:mas2; b:mas1;   n,m,i,j:byte;
begin
  assign(isxdan,'isxdan.txt');
  reset(isxdan);
  readln(isxdan,n,m);
for i:=1 to n do
    begin
    for j:=1 to m do
      read(isxdan, a[i,j]);
      readln(isxdan);
    end;
```

```
close(isxdan);
for i:=1 to n do
      begin
        b[i]:=0;
        for j:=1 to m do
          b[i]:=b[i]+a[i,j];
      end;
    assign(result,'result.txt');
    rewrite(result);
    for i:=1 to n do
      writeln (result,' Сумма элементов',i,'-й строки равна ', b[i]);
    close(result);
  end.
```

Пример. В одномерном массиве из натуральных чисел заменить все простые числа суммой их цифр.

Составим основную программу

```
type mas=array[1..100] of integer;
var a:mas; n,i:integer; f,f1:text;
begin
vvod(a,n);
preobr(a,n);
vivod(a,n);
end.
```

Затем будем последовательно детализировать алгоритмы всех входящих в нее процедур. Начнем с процедуры ввода:

```
procedure vvod(var a:mas;var n:integer);
var i:integer;
begin
assign(f,'vhod.txt');
reset(f);
readln(f,n);
for i:=1 to n do
  read(f,a[i]);
close(f);
end;
```

Затем вывод:

```
procedure vivod(a:mas;n:integer);
begin
assign(f1, 'vihod.txt');
rewrite(f1);
for i:=1 to n do
  write(f1,a[i],' ');
close(f1);
end;
```

Затем процедура преобразования массива:

```
procedure preobr(var a:mas;n:integer);
var i:integer;
begin
for i:=1 to n do
 if prost(a[i]) then sum(a[i]);
end;
```

Теперь детализируем функцию проверки, является ли число простым:

```
function prost(b:integer):boolean;
var i,k:integer;
begin
k:=0;
for i:=1 to b do
  if b mod i=0 then k:=k+1;
if k=2 then prost:=true
       else prost:=false;
end;
```

И затем процедуру нахождения суммы цифр числа и замены числа суммой цифр:

```
procedure sum(var d:integer);
var s:integer;
begin
s:=0;
while d>0 do
  begin
    s:=s+d mod 10;
    d:=d div 10;
  end;
d:=s;
end;
```

Таким образом, можно записать программу, обращая внимание на то, что любая процедура должна быть описана до своего вызова:

```
type mas=array[1..100] of integer;
var a:mas; n,i:integer; f,f1:text;
procedure vvod(var a:mas;var n:integer);
var i:integer;
begin
assign(f,'vhod.txt');
reset(f);
readln(f,n);
for i:=1 to n do
  read(f,a[i]);
close(f);
end;
procedure vivod(a:mas;n:integer);
Begin
assign(f1, 'vihod.txt');
rewrite(f1);
```

```
for i:=1 to n do
  write(f1,a[i],' ');
close(f1);
end;
function prost(b:integer):boolean;
var i,k:integer;
begin
k:=0;
for i:=1 to b do
  if b mod i=0 then k:=k+1;
if k=2 then prost:=true
       else prost:=false;
end;
procedure sum(var d:integer);
var s:integer;
begin
```

```
s:=0;
while d>0 do
  begin
    s:=s+d mod 10;
    d:=d div 10;
  end;
d:=s;
end;
procedure preobr(var a:mas;n:integer);
var i:integer;
begin
for i:=1 to n do
 if prost(a[i]) then sum(a[i]);
end;
```

```
begin
vvod(a,n);
preobr(a,n);
vivod(a,n);
end.
```

Домашнее задание

1. Составить опорный конспект лекции по теме «Метод пошаговой детализации разработки алгоритмов. Особенности использования массивов в качестве параметров» на основе презентации.

2. **Программирование на языке Pascal. Рапаков Г. Г., Ржеуцкая С. Ю.** СПб.: БХВ-Петербург, 2004, стр. стр. 247-249, 192-200.

Составить программы с использованием процедур и функций, ввод-вывод в текстовые файлы:

- В исходном файле размерность двумерного массива и его элементы. Найти количество четных элементов в каждом столбце. Результат вывести в текстовый файл.
- В исходном файле размерность одномерного массива и его целые элементы. Заменить все положительные элементы массива их факториалами. Результат вывести в текстовый файл.