Урок в 9-м классе по теме: "Процедуры"

Цели:

- Познакомить с понятием *подпрограмма*.
- Формировать умение составлять *процедуры*.
- Развивать алгоритмическое мышление.

ХОД УРОКА

I. Оргмомент.

II. Разминка (проводится обязательно в любом классе, если урок первый, после обеденного перерыва или после физкультуры).

Загадки

1. В греческом их — 24, в латинском — 26. Сколько их в русском?

Ответ: 33 (количество букв в алфивите).

2. Что находится в середине капусты?

Ответы: а) кочерыжка, б) буква «у».

3. С буквой «в» лечу и с буквой «г» лечу. Кто это?

Ответ: грач, врач.

Анаграммы. Задание 1

1. Бискей
2. Салькап
3. Транфор
4. Галло
5. Массбелер
6. Горлоп

Ответы

1. Бейсик
2. Паскаль
3. Фортран
4. Алгол
5. Ассемблер
6. Пролог

Анаграммы. Задание 2

1. Триголам
2. Ротарепо
3. Темка
4. Волусие
5. Граммпора
6. Клиц
7. Влетвение
8. Талодка

Ответы

1. Алгоритм
2. Оператор
3. Метка
4. Условие
5. Программа
6. Цикл
7. Ветвление
8. Отладка

III. Сообщение темы и целей.

План занятия

- понятие подпрограммы
- структура программы;
- о взаимодействии программы и процедуры;
- стандартные определения;
- экспериментальная работа с программами:
  - формирования значений одномерного массива с помощью датчика случайных чисел;
  - поиска элементов, принадлежащих двум массивам одновременно;
  - перестановки частей массива;
- выполнение самостоятельной работы

В практике программирования достаточно часто встречаются алгоритмы, в которых повторяются фрагменты, одинаковые по выполняемым действиям и различающиеся только значениями обрабатываемых данных. Например, при расчете траектории взлета космического аппарата необходимо при наборе высоты на каждом километре пересчитывать плотность воздушных слоев атмосферы.

Такие повторяющиеся фрагменты могут использоваться в разных программах. Например, вывод результатов работы программы на экран.

При составлении программы по такому алгоритму приходится задавать одну и ту же группу операторов соответственно для каждого из повторяющихся фрагментов. Для более эффективного программирования подобных алгоритмов в языках вводится понятие подпрограммы.

> Повторяющаяся группа операторов оформляется в виде ***подпрограммы*** — самостоятельной программной единицы со своими входными и выходными данными, которая записывается однократно, а в соответствующих местах программы обеспечиваются лишь *обращения* к ней (ссылки) и *передача данных.*

Использование подпрограмм позволяет улучшить структуру программы, делает ее более компактной и наглядной, облегчает процесс проектирования, разработки и отладки программы. Большинство современных программных средств представляют собой огромные тексты в тысячи, десятки тысяч и даже сотни тысяч команд. Разобраться в такой программе исключительно сложно, но и написать такую программу не менее трудно. Чем больше объем информации, тем труднее держать его в голове. Эта сложная проблема решается очень просто. Если вы не можете обработать задачу целиком, разбейте ее на части решите по частям и потом сложите полученные небольшие программки вместе. В языке

Паскаль подпрограммы реализуются в виде процедур и функций, которые вводятся в программу с помощью своего описания.

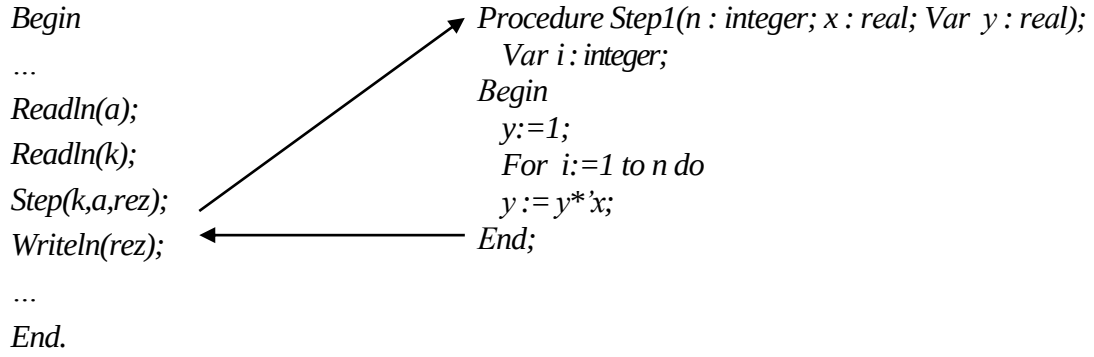
Структура программы. Программы на языке Паскаль состоят из заголовка программы, раздела описаний и тела программы. Раздел описаний может включать в себя следующие подразделы: меток, констант, типов, переменных, процедур и функций. Последовательность подразделов в структуре программы произвольная, но, естественно, если вводится переменная нового типа, заданного в *Type*, то подраздел *Type* предшествует подразделу *Var*. Принцип языка программирования «то, что используется, должно быть описано» сохраняется и для раздела описаний.

До этого момента времени мы использовали из раздела описаний только описание констант, переменных и типов. На этом занятии мы начнем изучать процедуры. Структура процедуры повторяет структуру программы.

```
Program <имя программы>;
    Label <метки>;
    Const <описание констант>;
    Type <описание типов данных>;
    Var <описание переменных>;
    <процедуры и функции>;
     Begin
    <основное тело программы>;
End.
```

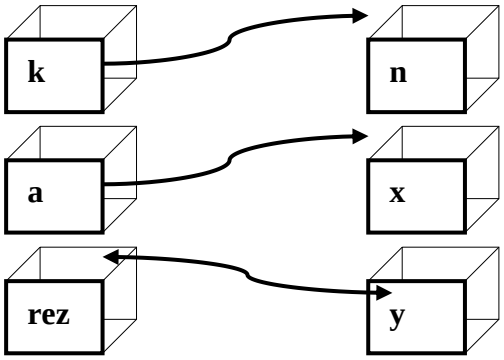
```
Procedure <имя процедуры>(<параметры>);
    Label < метки >;
    Const < описание констант >;
    Type < описание типов данных >;
    Var < описание переменных >;
    < процедуры и функции >;
     Begin
    < основное тело процедуры >;
End;
```

О взаимодействии программы и процедуры. Чтобы объяснение было наглядным, обратимся к примеру. Слева приведен фрагмент текста основной программы, справа — процедура вычисления натуральной степени действительного числа.



Что же мы видим? Процедура вызывается указанием ее имени, которое записывается после зарезервированного слова *Procedure*, используемого для обозначения процедуры.

Линейный ход выполнения основной программы становится нелинейным — управление вычислительным процессом передается на участок программного кода, занимаемый процедурой. После выполнения процедуры осуществляется возврат на оператор основной программы, следующий за вызовом процедуры (*End;* в процедуре очень содержательный оператор, сверхсодержательный — это возврат управления в логику, из которой вызывается процедура). Мы не рассматриваем вопросы о том, как это выполняется. Наше объяснение идет на уровне констатации фактов и только. Итак, взаимодействие программы и процедуры по управлению мы обозначили. Перейдем к взаимодействию по данным. В программе определены переменные k, a, rez. В процедуре — n, x, y её параметры, но они являются переменными процедуры. Причем n, x описаны без идентификатора *Var* , а rez — с идентификатором *Var* .Поясним разницу рисунком.



При вызове процедуры *Step(k,a,rez*) из основной программы значение переменной *k* присваивается переменной *n* процедуры, а значение переменной *a* становится значением переменной *x* и все. Стрелочка на рисунке в одну сторону. С переменными *y* и *rez* немного иначе. Стрелочка на рисунке и в ту, и в другую стороны. Образно говоря, процедура *Step1* знает о том, где в памяти компьютера находится переменная *rez*,и она знает, что при изменении значения переменной *y* необходимо произвести соответствующее изменение значения переменной *rez*. Это обратная связь по данным от процедуры к основной программе. На такой тип связи указывает идентификатор *Var* в описании параметров процедуры.

Стандартные определения. В любой программе все переменные делятся на *глобальные* и *локальные*. В нашем фрагменте программы переменные *k, a, rez* — глобальные, а *n, x, y* — локальные. *Глобальные переменные* — это переменные из раздела описаний основной части программы, а *локальные* — из раздела описаний процедур и функций. Но дело не только в месте описания переменных. Локальные переменные существуют только в течение времени работы процедуры, определяются (создаются) при ее вызове и исчезают после завершения работы процедуры (после выполнения «сверхсодержательного» оператора *End;)*. Таким образом, если бы в основной программе было три фрагмента с вызовом процедуры *Step1* (желательно с

различными параметрами), то для программного кода процедуры *Step1* три раза выделялось бы место в оперативной памяти для переменных *n, x, y*, и, соответственно, три раза освобождалось.

Параметры. При описании процедуры задается список формальных параметров. Каждый параметр, описанный в этом списке, является локальным по отношению к описываемой процедуре, т.е. к нему можно обращаться только в пределах данной процедуры, но не из основной программы. (В нашем примере *n, x, y* — формальные параметры).

Фактические параметры — это параметры, которые передаются процедуре при обращении к ней (*k, a, rez* — фактические параметры). Число и тип формальных и фактических параметров должны совпадать с точностью до их следования.

Параметры значения. Другими словами, передача параметров по значению. Копия фактического параметра становится значением соответствующего формального параметра. Внутри процедуры можно производить любые действия с данным формальным параметром (допустимые для его типа), но эти изменения никак не отражаются на значении фактического параметра, то есть каким он был до вызова процедуры, то таким же и останется после завершения ее работы (*n, x* — параметры-значения).

Параметры-переменные. Другими словами, передача параметров по ссылке. Это те формальные параметры, перед которыми стоит идентификатор *Var*. Передается адрес фактического параметра (обязательно переменной), после этого формальный параметр становится его синонимом. Любые операции над формальным параметром выполняются непосредственно над фактическим параметром.

Договоримся о том, что мы будем стараться использовать процедуры, может быть в ущерб эффективности программ, например, с точки зрения количества переменных. Каждая процедура должна иметь одну точку входа и одну точку выхода, использование глобальных переменных в процедуре должно быть минимально, взаимодействие вызывающей логики с процедурой должно осуществляться (по возможности) только через параметры. Для чего мы это оговариваем? Мы только осваиваем структурную технологию разработки программ, поэтому на каждом этапе текущую задачу будем разбивать на ряд подзадач, определяя тем самым некоторое количество отдельных подпрограмм.

Экспериментальный раздел работы

1. Задание значений элементам одномерного массива (с помощью генератора случайных чисел) и вывод результата на экран мы уже делали. Оформим эти действия как процедуры.

```
Program ex1;
Const n=10;
      l=-10;
      r=21;
Type my =array [1 . . n] of integer;
Var a: my;

Procedure Init(t, v, w: integer; Var x: my);
Var i: integer;
Begin
  Randomize;
    For i:=1 to t do x[i]:=v + integer(random(w));
End;

Procedure Print(t: integer;  x: my);
Var i: integer;
Begin
  For i:=1 to t doWrite(x[i]:5);
  Writeln;
End;
Begin
Init(n,l,r,a);                          {Формирование элементов массива}
Print(n,a);                             {Вывод элементов массива}
End.
```

Кажется, что мы проиграли. Было гораздо меньше строк программного кода. Но надо немного потерпеть. Начиная с этого момента, будем стараться создавать программы так, чтобы основная программа состояла из вызовов процедур и функций. Все написанные вами процедуры (только процедуры, без основной программы), а потом и функции, будем сохранять в отдельном файле, который в дальнейшем оформим как личный библиотечный модуль.

2. Даны два одномерных массива из целых чисел разной размерности. Найти среднее арифметическое элементов каждого массива и их сумму.

   Процедуру нахождения среднего арифметического элементов массива кто-то из учеников делает у доски, остальные на рабочих местах. После сверки решения переходим к самостоятельной работе, начиная с текущей задачи для трех массивов.

Задания для самостоятельной работы

1. Решить предыдущую задачу для трех массивов.
2. Даны три одномерных массива из целых чисел одинаковой размерности. Сформировать четвертый массив, каждый элемент которого равен максимальному из соответствующих элементов первых трех массивов.

Домашнее задание

Даны два одномерных массива одинаковой размерности. Поменять местами первые элементы массивов и выполнить перестановку элементов массивов в обратном порядке.

Использованная литература

1. *Е.В. Андреева*. Методика обучения основам программирования на уроках информатики. Лекция 8. «Первое сентября. Информатика», 2005
2. *С. Окулов.* Основы программирования. М.: — Лаборатория Базовых Знаний, 2002
3. *В. Потопахин*. Turbo Pascal. Освой на примерах. СПб.: — БХВ-Петербург, 2005
4. *В.С.Новиков, Н.И.Парфилова, А.Н.Пылькин*. Алгоритмизация и программирование на Турбо Паскале. М.: — Горячая линия - Телеком, 2005
5. *Д.М. Златопольский*. Сборник заданий для внеклассной работы. М., 2005