Лабораторная работа №1

Сортировка массивов

Цель работы

Знакомство с типом данных “массив”, получение навыков ввода, обработки массивов, создание диалогового приложения, осуществляющего сортировку массивов различными методами.

Введение

Массив – упорядоченный набор однотипных переменных , объединенных одним именем. В качестве типа элементов массива можно использовать все числовые, символьный, строковый и логический типы. Каждый элемент массива имеет свой номер (индекс). Для индексов массивов подходит любой порядковый тип, то есть такой, который в памяти машины представляется целым числом. Каждый элемент является переменной, т.е. обладает своим именем и значением.

Массив относится к так называемым структурированным данным, то есть таких, что имеют фиксированную внутреннюю структуру (организацию).

При обращении к отдельному элементу массива необходимо указать его индекс (местонахождение в массиве):

```
A[7]   i:=7;    A[i]
```

Здесь i – индекс элемента массива

Объявление массива

Массивы, как и другие переменные, должны быть объявлены в разделе var

```
var
Mas: array [1..15] Of real;
Work: array [(Mon, Tue, Wed)] Of integer;
B: array ['A'..'Z'] Of boolean;
C: array [1..3, 1..5] Of real;
```

Ввод массива

Чтобы заполнить массив данными существуен несколько способов:

- непосрерственное присваивание значений элементам;
- генерация и присваивание значений с помощью функции random;
- ввод значений элементов с клавиатуры;

1) Ввод элементов одномерного массива с клавиатуры:

```
var
A : array[1..20] of real;
begin
writeln('Введите элементы массива:');
for i:=1 to n do readln(A[i]);
...
```

2) Заполнение массива случайными числами.

В этом случае необходимо перезапустить генератор случайных чисел. Затем в цикле(например, в цикле с параметром, где в качестве параметра выступает индекс массива) сгенерровать значения для всех элементов.

```
randomize;
for i:=1 to n do a[i]:=random(100);
...
```

Сортировка массивов

Сортировка массивов – это упорядочение их элементов

Метод пузырька (Bubble Sort)

Суть метода заключается в многократном проходе по списку. На каждом шаге последовательно сравниваются пары соседних элементов, и если порядок в такой паре неверный, то элементы в паре меняются местами. При проходе алгоритма, элемент, стоящий не на своей позиции, «всплывает» до нужной позиции как пузырёк, откуда и название алгоритма.

Например, чтобы отсортировать массив размером N по возрастанию элементов, необходимо выполнить:

```
для j от 1 до N-1 выполнять
  для i от 1 до N-j выполнять
      если M[i] > M[i+1]  то
          обмен M[i] и M[i+1];
```

Обмен между значениями элементов можно осуществить при помощи

вспомогательной переменной, например:

```
Tmp := M[i];
M[i] := M[i+1];
M[i+1] := Tmp;
```

Сортировка со вставками (Insertion Sort**)**

Сортируемый массив просматривается в порядке возрастания номеров и каждый элемент вставляется в уже просмотренную часть массива так, чтобы сохранить порядок.

```
для i от 2 до N do
  нц
    Tmp:=M[i];
    j:=i-1;
    пока (j>0) и (M[j]>Tmp) do
      нц
        M[j+1]:=M[j];
        j:=j-1;
      кц
    M[j+1]:=Tmp;
  кц
```

Задание на лаюораторную работу

Реализовать пользовательское приложение со следующим функционалом:

- программа должна производить сортировку массива двумя различными способами;
- начальный массив формируется из случайных чисел;
- на экран выводятся начальный и конечный массив;
- выход осуществляется при выборе соотвествующего пункта меню.

Порядок выполнения работы

Объявить массив из 100 элементов а разделе описания переменных.

В основном блоке программы заполнить массив случайными числами в интервале [0,99]

Вывести на экран заполненный массив, расположив элементы в виде таблицы 10x10

Вывести меню из трех пунктов:

1) сортировка методом пузырька;

2) сортировка методом вставок;

3) выход.

Сделать обработку всех пунктов меню, воспользовавшись оператором case или оператором if и командами языка Pascal.

Перед сортировкой предварительно скопировать массив в другой (для того, чтобы начальный массив сохранился после сортировки). Отсортировать массив по возрастанию элементов и вывести результат на экран в виде таблицы (наименьший элемент должен оказаться в левом верхнем углу экрана, наибольший – в правом нижнем).

После вывода отсортированного массива снова вывести меню на экран. Процесс повторять до тех пор, пока не будет выбран пункт меню “выход”

Результаты оформить в виде отчета, где привести:

- текст программы на языке Pascal;
- описание используемых в программе типов данных, операторов, функций и процедур;
- скриншоты результатов выполнения программы.

Лабораторная работа №2

Использование подпрограмм в Pascal

Цель работы

Знакомство с понятиями "процедура" и "функция" в языке Pascal, изучение их сходств и различий. Получение навыков в модульном программировании.

Введение

Существуют случаи, когда в программе приходится выполнять одни и те же вычисления, но при различных исходных данных. повторяющиеся вычисления выделяют в самостоятельную часть программы, которая может быть использована многократно по мере необходимости. Такая автономная часть программы, реализующая определенный алгоритм, оформленная в виде отдельной синтаксической конструкции, снабжённая именем и допускающая обращение к ней из различных частей общей программы, называется подпрограммой. Подпрограмма – это последовательность операторов, которые определены и записаны только в одном месте программы, однако их можно вызвать для выполнения из одной или нескольких точек программы. Каждая подпрограмма определяется уникальным именем.

Для использования подалгоритма в качестве подпрограммы его необходимо описать в разделе описания подрограмм. Для вызова в основной части программы необходимо вызвать подпрограмму с нужными параметрами Это приведет к выполнению входящих в подпрограмму операторов, работающих с указанными параметрами. После выполнения подпрограммы работа продолжается с той команды, следует за вызовом подпрограммы.

В Pascal существует два вида подпрограмм – **процедуры** и **функции**.

Процедура – это независимая именованная часть программы, которую можно вызвать по имени для выполнения определённой в ней последовательности

действий. Процедуры служат для задания совокупности действий, направленных на изменение внешней по отношению к ним программной обстановки.

Функция отличается от процедуры тем, что возвращает результат указанного при её описании типа. Вызов функции может осуществляться из выражения, где имя функции используется в качестве операнда. Функции являются частным случаем процедур, и обязательно возвращают в точку вызова результат как значение имени этой функции. При использовании функций необходимо учитывать совместимость типов в выражениях.

Для обмена информацией между процедурами и функциями и другими блоками программы существует механизм **входных** и **выходных параметров**. Входными параметрами называют величины, передающиеся из вызывающего блока в подпрограмму (исходные данные для подпрограммы), а выходными – передающиеся из подрограммы в вызывающий блок (результаты работы подпрограммы).

Одна и та же подпрограмма может вызываться неоднократно, выполняя одни и те же действия с разными наборами входных данных. Параметры, использующиеся при записи текста подпрограммы в разделе описаний, называют **формальными**, а те, которые используются при ее вызове – **фактическими**.

Описание и вызов процедур и функций

Описание подпрограммы производится в разделе описаний основной программы. Любая процедура оформляется аналогично программе, может содержать заголовок, разделы описаний и операторов. Синтаксис заголовка процедуры:

Формат описания процедуры имеет вид:

```
procedure имя процедуры (формальные параметры);
раздел описаний процедуры
begin
    исполняемая часть процедуры
end;
```

Формат описания функции:

```
function имя функции (формальные параметры):тип результата;
раздел описаний функции
begin
    исполняемая часть функции
end;
```

Формальные параметры в заголовке процедур и функций записываются в виде:

```
var имя праметра: имя типа
```

и отделяются друг от друга точкой с запятой. Ключевое слово var может отсутствовать Если параметры однотипны, то их имена можно перечислять через запятую, указывая общее для них имя типа. При описании параметров можно использовать только стандартные имена типов, либо имена типов, определенные с помощью команды type. Список формальных параметров может отсутствовать, при этом символ " ; " ставится сразу за именем процедуры и данные из места вызова процедуры в её тело не передаются.

Вызов процедуры производится оператором, имеющим следующий формат:

```
имя  процедуры(список фактических параметров);
```

В списке фактических параметров параметры перечисляются через запятую в соответсвующем порядке.

Выполнение оператора вызова процедуры состоит в том, что все формальные параметры заменяются соответствующими фактическими. После этого создается динамический экземпляр процедуры, который и выполняется. После выполнения процедуры происходит передача управления в основную программу, т.е. начинает выполняться оператор, следующий за оператором вызова процедуры

Вызов функции в Pascal может производиться аналогичным способом, кроме того имеется возможность осуществить вызов внутри какого-либо выражения. В частности имя функции может стоять в правой части оператора присваивания, в разделе условий оператора if и т.д.

Для передачи в вызывающий блок выходного значения функции в исполняемой части функции для возврата какого-либо значения необходимо поместить следующую команду:

имя функции := *результат*;

При вызове процедур и функций необходимо соблюдать следущие правила:

- количество фактических параметров должно совпадать с количеством формальных;
- соответствующие фактические и формальные параметры должны совпадать по порядку следования и по типу.

Следует отметить, что имена формальных и фактических параметров могут совпадать. Это не приводит к проблемам, так как соответствующие им переменные все равно будут различны из-за того, что хранятся в разных областях памяти. Кроме того, все формальные параметры являются временными переменными – они создаются в момент вызова подпрограммы и уничтожаются в момент выхода из нее.

В отличие от констант и переменных, объявление подпрограммы может быть оторвано от ее описания. В этом случае после объявления нужно указать ключевое слово forward:

```
function <имя_функции> [(<параметры>)]:<тип_результата>;
forward;
procedure <имя_процедуры> [(<список_параметров>)]; forward;
```

Если объявление подпрограммы было оторвано от ее описания, то описание начинается дополнительной строкой с указанием только имени подпрограммы:

```
function <имя_подпрограммы>;
```

или

```
procedure <имя_подпрограммы>;
```

Описания двух различных подпрограмм не могут пересекаться: каждый блок должен быть логически законченным. Однако внутри любой подпрограммы (она

ведь тоже является программой, помните?) могут быть описаны другие процедуры или функции – **вложенные**. На них распространяются все те же правила объявления и описания подпрограмм.

Рассмотрим использование процедуры на примере программы поиска максимума из двух целых чисел.

```
var x,y,m,n: integer;

procedure MaxNumber(a,b: integer; var max: integer);
begin
   if a>b then max:=a else max:=b;
end;

begin
   write('Введите x,y ');
   readln(x,y);
   MaxNumber(x,y,m);
   MaxNumber(2,x+y,n);
   writeln('m=',m,'n=',n);
end.
```

Аналогичную задачу,можно решить с помощью функции:

```
var x,y,m,n: integer;

function MaxNumber(a,b: integer): integer;
   var max: integer;
begin
   if a>b then max:=a else max:=b;
   MaxNumber := max;
end;

begin
   write('Введите x,y ');
   readln(x,y);
   m := MaxNumber(x,y);
   n := MaxNumber(2,x+y);
   writeln('m=',m,'n=',n);
end.
```

Передача параметров

В стандарте языка Паскаль передача параметров может производиться двумя

способами – по значению и по ссылке. Параметры, передаваемые по значению, называют **параметрами-значениями**, передаваемые по ссылке – **параметрами-переменными**. Последние отличаются тем, что в заголовке процедуры (функции) перед ними ставится служебное слово var.

При первом способе (передача по значению) значения фактических параметров копируются в соответствующие формальные параметры. При изменении этих значений в ходе выполнения процедуры (функции) исходные данные (фактические параметры) измениться не могут. Поэтому таким способом передают данные только из вызывающего блока в подпрограмму (т.е. входные параметры). При этом в качестве фактических параметров можно использовать и константы, и переменные, и выражения.

При втором способе (передача по ссылке) все изменения, происходящие в теле процедуры (функции) с формальными параметрами, приводят к немедленным аналогичным изменениям соответствующих им фактических параметров. Изменения происходят с переменными вызывающего блока, поэтому по ссылке передаются выходные параметры. При вызове соответствующие им фактические параметры могут быть только переменными.

Выбор способа передачи параметров при создании процедуры (функции) : входные параметры нужно передавать по значению, а выходные – по ссылке. Практически это сводится к расстановке в заголовке процедуры (функции) описателя var при всех параметрах, которые обозначают результат работы подпрограммы. Однако, в связи с тем, что функция возвращает только один результат, в ее заголовке использовать параметры-переменные не рекомендуется.

Локальные и глобальные идентификаторы

Использование процедур и функций в Паскале тесно связано с некоторыми особенностями работы с идентификаторами (именами) в программе. В часности, не все имена всегда доступны для использования. Доступ к идентификатору в

конкретный момент времени определяется тем, в каком блоке он описан.

Имена, описанные в заголовке или разделе описаний процедуры или функции называют **локальными** для этого блока. Имена, описанные в блоке, соответствующем всей программе, называют **глобальными**. Следует помнить, что формальные параметры прцедур и функций всегда являются локальными переменными для соответствующих блоков.

Основные правила работы с глобальными и локальными именами можно сформулировать так:

- Локальные имена доступны (считаются известными, "видимыми") только внутри того блока, где они описаны. Сам этот блок, и все другие, вложенные в него, называют **областью видимости** для этих локальных имен.
- Имена, описанные в одном блоке, могут совпадать с именами из других, как содержащих данный блок, так и вложенных в него. Это объясняется тем, что переменные, описанные в разных блоках (даже если они имеют одинаковые имена), хранятся в разных областях оперативной памяти.

Глобальные имена хранятся в области памяти, называемой **сегментом данных** (статическим сегментом) программы. Они создаются на этапе компиляции и действительны на все время работы программы.

В отличие от них, локальные переменные хранятся в специальной области памяти, которая называется **стек**. Они являются временными, так как создаются в момент входа в подпрограмму и уничтожаются при выходе из нее.

Имя, описанное в блоке, "закрывает" совпадающие с ним имена из блоков, содержащие данный. Это означает, что если в двух блоках, один из которых содержится внутри другого, есть переменные с одинаковыми именами, то после входа во вложенный блок работа будет идти с локальной для данного блока переменной. Пременная с тем же имнем, описанная в объемлющем блоке, становится временно недоступной и это продолжается до момента выхода из вложенного блока.

Рекомендуется все имена, которые имеют в подпрограммах чисто внутреннее, вспомогательное назначение, делать локальными. Это предохраняет от изменений глобальные объекты с такими же именами.

Задание на лабораторную работу

Создать программу, в которой производится вычисление заранее заданной преподавателем функции от заранее заданного набора аргументов и осуществляется вывод результатов.

Расчет функции должен быть произведен с помощью механизма функций языка Pascal. Вывод результатов должен производиться на консоль пользователя с помощью специально написанной процедуры.

Варианты функций:

1.$\cos(3x-\pi x/2)^5$; аргументы: x изменяется от 0 до 2π с шагом $\pi/10$

2. $\cos(2x/\pi)*\sin(3x-2\pi)$; аргументы: x изменяется от $\pi/2$ до $3\pi/2$ с шагом $\pi/20$

3. $\lg x* (2x^2-1)$; аргументы: x изменяется от 1 до 20 с шагом 1

Процедура должна осуществлять вывод аргументов и соответствующих им значений функций в следующем формате:
x=<значение x, два знака после запятой> f(x)=<значение функции, четыре знака после запятой>

Результаты работы оформить в виде отчета, где привести:

- текст программы на языке Pascal;
- описание используемых в программе типов данных, операторов, функций и процедур;
- скриншоты результатов выполнения программы.

Лабораторная работа №3

Файловый ввод вывод в программах на языке Pascal

Цель работы

Изучение основных функций работы с файлами в Pascal, организация ввода и вывода структурированных данных из файлов.

Введение

Записи

Тип “запись” является структурированным типом данных, то есть таким, переменные которого составлены из нескольких частей. В Pascal существует возможность объединить в одну переменную данные разных типов (тогда как в массиве все элементы имеют одинаковый тип).

Пусть в переменной требуется хранить сведения о некотором человеке: ФИО, пол, адрес, телефон. Тогда для хранения этих данных будет удобен такой тип:

```
type tPerson = record
    Name,Surname,SecondName: string[30];
    Male: boolean;
    Address: string[50];
    Phone: string[11];
end;
```

Объявление переменной типа запись выполняется стандартно, в разделе `var`.

Части записи (в нашем случае: Name, Surname, SecondName, Male, Address, Phone) называются полями. Обращение к полю записи в программе производится с помощью знака '.' (точка). Пример обращения к полям:

```
var emp: tPerson;
...
begin
...
```

```
emp.Surname:='Иванов';
emp.Name:='Иван';
emp.SecondName:='Иванович';
...
```

В случаях, когда приходится много раз обращаться к полям одной и той же записи, можно воспользоваться ключевым оператором with, который упрощает ссылку к структурированным переменным:

```
with <имя_записи> do <оператор>;
```

Пример:

```
with emp do
begin
     Surname:=' Иванов';
     Name:='Иван';
     SecondName:='Иванович';
     ...
end;
```

Записи можно включать в состав более сложных переменных, например массивов и других записей:

```
type tStaff = array [1..30] of tPerson;
```

Работа с файлами

При работе с файлами существует определенный порядок действий, которого необходимо придерживаться. Вот все эти действия:

1. Создание (описание) файловой переменной;
2. Связывание этой переменной с конкретным файлом на диске или с устройством ввода-вывода (экран, клавиатура, принтер и т.п.);
3. Открытие файла для записи либо чтения;
4. Действия с файлом: чтение либо запись;
5. Закрытие файла.

Типы файловых переменных:

1. **text** – текстовый файл. Из переменной такого типа мы сможем читать строки и символы.
2. **file of _любой_тип_** – так называемые "типизированные" файлы, то есть файлы, имеющие тип. Этот тип определяет, какого рода информация содержится в файле и задается в параметре_любой_тип_. Например:

```
F: file of integer;
```

Файл F содержит числа типа integer; Соответсвенно, читать из такого файла можно только переменные типа integer, ровно как и писать.

```
type
A = record
    I,J: Integer;
    S: String[20];
end;
var
  F: File of A;
```

3. **file** – нетипизированный файл:

```
F: File;
```

Чтение и запись в типизированные файлы отличается от работы с файлами других типов. Эти действия производятся путем указания количества байт, которые нужно прочитать, а также указанием области памяти, в которую нужно прочитать эти данные.

Связывание переменной с файлом

Выполняется одинаково для всех типов файлов:

```
assign(<переменная_файлового_типа>, '<путь к файлу>');
```

В качестве параметров задаются переменная любого файлового типа и строка – путь к файлу:

```
var
F1: file of integer;
begin
```

```
Assign(F1, 'int.txt');
```

Открытие файла

При открытии файла необходимо учитывать, зачем открывается файл – для записи или чтения. Более того, в зависимости от типа файла процедуры выполняют различные действия.

1. `reset(<любая_файловая_переменная>);`

Открывает файл на чтение. В качестве параметра – файловая переменная любого типа. В случае с текстовым файлом, он открывается только на чтение. В случае с типизированным и нетипизированным файлом – открывается на чтение и запись.

2. `append(T: Text);`

Открывает текстовый файл (только текстовый!) на запись. Reset при задании параметра типа Text не позволит писать в него данные, открыв файл лишь для чтения. Для записи в текстовый файл нужно использовать append. Если чтение – reset.

Если файл открыт на чтение, не нужно закрывать его и открывать снова на запись. В этом случае файл закрывается сам и открывается заново. При записи данных в файл при открытии его с помощью этой процедуры они записываются в конец файла.

3. `rewrite(F);`

Создает новый файл либо перезаписывает существующий. Необходимо быть осторожным при использовании этой процедуры, т.к. файл, открытый таким образом будет полностью перезаписан.

Закрытие файла

Закрытие файла производиться с помощью процедуры **Close(F)**, где F – это переменная файлового типа. Эта процедура одна для всех типов файлов.

Запись и чтение файлов

Текстовые и типизированные файлы

Чтение файлов.

Чтение файлов производится с помощью процедур `read` и `readln`. Они используются также, как и при чтении информации с клавитуры. Отличие лишь в том, что перед переменной, в которую помещается считанное значение, указывается переменная файлового типа (дескриптор файла):

```
read(F, C);
```

где `F` – дескриптор файла, `C` – переменная (char, string – для текстовых, любого типа – для типизированных файлов).

Запись в файлы.

Запись в файлы производиться точно так же, как и запись на экран с помощью процедур `write` и `writeln`. Как и в случае с чтением, перед записываемой в файл переменной указывается тескриптор файла:

```
write(F, S);
```

где `F` – дескриптор, `S` – переменная.

При этом переменная должна соответствовать типу файла.

```
program cat;
var
   f: text;
   c: char;
begin
   assign(f,'prog.pas');
   reset(f);
   while not eof(f) do
   begin
       while not eoln(f) do
       begin
           read(f,c);
           write(c);
       end;
```

```
        readln(f);
        writeln;
    end;
    close(f);
end.
```

В программе бывает необходимо определить, дошёл ли указатель файла до конца строки или до конца файла. В этом случае полезно использовать такие функции:

```
eoln(TxtFile: text): boolean;
eof(TxtFile: text): boolean;
```

Первая принимает значение true (истина), если указатель стоит на конце строки, вторая – то же самое для конца файла.

Нетипизированные файлы

Суть таких файлов заключается в следующем: имея файл без определенного типа, мы можем читать из него любые данные, будь то строки, символы или записи.

Читая данные из файла без типа мы получаем блоки информации, которые составляют обычный набор байт. Указывая переменную, в которую эти байты надо поместить, мы как бы "на ходу преобразуем" эти данные к нужному типу.

Чтение из файлов без типа.

Сама процедура связывания файловой переменной с внешним файлом и его открытие ничем чем отличаются от обычного порядка действий. Только переменная в данном случае должна иметь тип File; , то есть быть файлом без типа.

Чтение происводиться с помощью процедуры `blockread`:

```
blockread(F: file, Buf: var, size: word, result: word)
```

`F: file;` – переменная типа `file`; Именно из этой переменной и происходит чтение данных., `Buf: var;` – переменная любого типа. В эту переменную

помещаются прочитанные данные., size: word; – количество считываемых байт, result: word; – в эту переменную помещается реальное количество байт, которые были прочитаны.

Процедура работает следующим образом: из файла F считывается Size записей, которые помещаются в память, начиная с первого байта переменной Buf. После выполнения процедуры реальное количество прочитанных байт помещается в переменную Result. Здесь надо сказать, что эта переменная совсем не обязательно должна присутствовать в качестве параметра, то есть ее попросту можно опустить. Однако иногда она довольно полезна и даже необходима – например, если чтение было окончено до того, как было прочитано требуемое количество байт (достигнут конец файла), мы можем это отследить через переменную Result. Если же в этом случае (чтение данных после конца файла) переменная Result не будет указана, то образуется ошибка времени выполнения N100 "Disk read error" (Runtime error 100).

Запись в файлы без типа.

```
BlockWrite(F: File, Buf: Var, Size: Word, Result: Word)
```

F: File; – переменная типа File; Buf: Var; – переменная любого типа. Начиная с этой переменной, данные будут записываться в файл. Size: Word; – количество записываемого блока данных в байтах. Result: Word; – в эту переменную помещается реальное количество байт, которые были записаны.

```
var
   Fin, fout : file;
   NumRead,NumWritten : word;
   Buf : Array[1..2048] of byte;
   Total : longint;

begin
   Assign (Fin, Paramstr(1));
   Assign (Fout,Paramstr(2));
   Reset (Fin,1);
   Rewrite (Fout,1);
```

```
    Total:=0;
    Repeat
        BlockRead (Fin,buf,Sizeof(buf),NumRead);
        BlockWrite (Fout,Buf,NumRead,NumWritten);
        inc(Total,NumWritten);
    Until (NumRead=0) or (NumWritten<>NumRead);
    Write('Copied ',Total,' bytes from file ',paramstr(1));
    Writeln (' to file ',paramstr(2));
    close(fin);
    close(fout);
end.
```

Задание на лабораторную работу

Написать программу, осуществляющую ввод структурированных данных в файл и вывод данных из этого файла на экран (сделать хранилище данных в файле).

Варианты задания

1) Карточка студента
2) Библиотека
3) Фонотека
4) Телефонный справочник
5) Отдел кадров
6) Страны мира

Порядок выполнения работы

Создать новый тип данных «запись» согласно полученному варианту.

Вывести приглашение пользователю на ввод данных об объекте. Заполнить поля записи введенными пользователями данными. Записать в файл, открыв его в соответствующем режиме. Процесс записи оформить в виде цикла, на каждом шаге которого вводятся данные об очередном объекте и выводится приглашение “Ввести данные о новом объекте? (Yes/No)”. Выход из цикла – ввод пользователем “n”(No) после ввода данных об очередном объекте.

Считать все записи из файла и вывести их на экран.

Лабораторная работа №4

Алгоритмы на списках

Цель работы

Знакомство со ссылочными реализациями структур данных. Создание приложения, осуществляющего работу со списками.

Введение

Списки

Классический пример структуры данных последовательного доступа, в которой можно удалять и добавлять элементы внутри структуры, — это линейный список.

Список – это конечное множество динамических элементов, размещающихся в разных областях памяти и объединенных в логически упорядоченную последовательность с помощью специальных указателей (адресов связи).

Список – структура данных, в которой каждый элемент имеет информационное поле (поля) и ссылку (ссылки), то есть адрес (адреса), на другой элемент (элементы) списка. Список - это так называемая линейная структура данных, с помощью которой задаются одномерные отношения.

Каждый элемент списка содержит информационную и ссылочную части. Порядок расположения информационных и ссылочных полей в элементе при его описании - по выбору программиста, то есть фактически произволен. Информационная часть в общем случае может быть неоднородной, то есть содержать поля с информацией различных типов. Ссылки однотипны, но число их может быть различным в зависимости от типа списка. В связи с этим для описания элемента списка подходит только тип «запись», так как только этот тип данных может иметь разнотипные поля. Например, для однонаправленного списка элемент

должен содержать как минимум два поля: одно поле типа «указатель», другое - для хранения данных пользователя. Для двунаправленного – три поля, два из которых должны быть типа «указатель».

Описать элемент однонаправленного списка (см. рис 1) можно следующим образом:

```
type
     el=^zap;
     zap=record
     inf1 : integer;  { первое информационное поле  }
     inf2 : string;   { второе информационное поле  }
     next : el;    {ссылочное поле  }
     end;
```

Из этого описания видно, что имеет место рекурсивная ссылка: для описания типа **point** используется тип **zap**, а при описании типа **zap** используется тип **point**. По соглашениям Паскаля в этом случае сначала описывается тип «указатель», а затем уже тип связанной с ним переменной. Правила Паскаля только при описании ссылок допускают использование идентификатора (**zap**) до его описания. Во всех остальных случаях, прежде чем упомянуть идентификатор, необходимо его определить.

```
var
     first,   { указатель на первый элемент списка }
     p, q , t : el;   { рабочие указатели, с помощью которых будет
выполняться работа с элементами списка }
```

Формирование пустого списка.

```
procedure Create_Empty_List ( var first : el);
begin
     first = nil;
End;
```

Формирование очередного элемента списка.

```
procedure Create_New_Elem(var p: el);
begin
     New (p);
  Writeln ('введите значение первого информационного поля: ');
```

```
  Readln ( p^.inf1 );
  Writeln ('введите значение второго информационного поля: ');
  Readln ( p^.inf2 );
  p^.next := nil; {все поля элемента должны быть инициализированы}
end;
```

Подсчет числа элементов списка.

```
function Count_el(First:el):integer;
var
  K : integer;
  q : el;
begin
  If First = Nil then
  k:=0 { список пуст }
  Else
  begin {список существует}
    k:=1; {в списке есть хотя бы один элемент}
    q:=First; {перебор элементов списка начинается с  первого}
    while q^.Next <> Nil do
    begin
      k:=k+1;
      q:=q^.Next; {переход к следующему элементу списка}
    end;
  end;
  Count_el:=k;
end;
```

Включение элемента в конец списка.

```
procedure Ins_end_list(P : el; Var First : el);
begin
  If First = Nil Then
  First:=p
  Else
  Begin
    q:=First; {цикл поиска адреса последнего элемента}
    While q^.Next <> Nil do
      q:=q^.Next;
    q^.Next:=p;{ссылка с бывшего последнего на включаемый элемент}
    P^.Next:=Nil; {не обязательно}
  End;
end;
```

Включение в середину (после i-ого элемента).

```
procedure Ins_after_I ( first : el; p : el; i : integer);
var
```

```
  t, q : el;
  K ,n : integer;
begin
  n := count_el(first); {определение числа элементов списка}
  if (i < 1 )  or ( i > n )then
  begin
    writeln ('i задано некорректно');
    exit;
  end
  else
  begin
    if i = 1 then
    begin
      t := first;{адрес 1 элемента}
      q := t^.next; {адрес 2 элемента}
      t^.next := p;
      p^.next := q;
    end
    ELSe
    if i = n then
    begin { см. случай вставки после последнего элемента}
       .  .  .
    end
    else {вставка в «середину» списка}
    begin
      t := first;
      k := 1;
      while ( k < i ) do
      begin {поиск адреса i-го элемента}
        k := k + 1;
        t := t^.next;
      end;
      q := t^.next;
      {найдены адреса i-го (t) и i+1 -го (q) элементов }
       t^.next := p;
      p^.next := q;
      {элемент с адресом p вставлен}
    end;
  end;
end;
```

Удаление элемента из середины списка (i-ого элемента).

```
Procedure Del_I_elem ( first : el; i : integer);
Var
  t, q, r : el;
  K ,n : integer;
```

```
Begin
   n := count_el(first); {определение числа элементов списка}
  if (i < 1 )  or ( i > n ) then
  begin
    writeln ('i задано некорректно');
    exit;
  end
  else
  begin {нужно добавить подтверждение удаления }
    if i = 1 then
    begin {удаляется 1 элемент}
      t := first;
      first:= first^.next;
      dispose ( t);
    end
    else
    if i = n then
    begin { см. случай удаления последнего элемента}
      .  .  .
    end
    else {удаление из «середины» списка}
    begin
      t := first;
      q := nil;
      k := 1;
      while ( k < i ) do
      begin {поиск адресов (i-1)-го и i-го элементов}
        k := k + 1;
        q := t;
        t := t^.next;
      end;
      r := t^.next;
      {найдены адреса i-го (t), (i-1)-го (q) и (i+1)-го (r) элементов
}
      q^.next := r;
      dispose ( t );   {удален i-ый элемент }
    end;
  end;
end;
```

Удаление всего списка с освобождением памяти.

```
procedure Delete_List(Var First : el);
var
  P, q : el;
  Answer : string;
```

```
begin
  If First <> Nil Then
  begin  { список не пуст }
    writeln ( ' Вы хотите удалить весь список ? (да/нет)' );
    readln ( answer );
    if answer = 'да' then
    begin
      q:=First;
      p:=nil;
      while ( q <> nil ) do
      begin
        p:=q;
        q:=q^.Next;
        Dispose(p);
      end;
      First:=Nil;
    end;
  end
  else
  writeln ('список  пуст  ');
end;
```

В рамках данной лабораторной работы, студентам предлагается реализовать однонаправленный список.

Для организации элемента списка, можно создать следующую структуру

Задание на лабораторную работу

Создать приложение со следующим функционалом:

– Приложение предназначено для создания и редактирования однонаправленных списков;

– Выбор операции над списком производится пользователем в режиме меню.

– Ввод параметров производится с клавиатуры.

Порядок выполнения лабораторной работы

Вывести приглашение пользователю на ввод выбор операции над списком:

1) создание нового списка;

2) добавление нового звена в список;

3) удаление звена из списка;

4) удаление списка;

5) выход.

При выборе первого пункта вывести приглашение на ввод количества элементов списка. Вывести приглашение на ввод значений элементов. Сохранить введенные данные в переменные типа «структура». Вывести содержимое списка на экран.

При выборе второго пункта вывести приглашение на ввод информационного поля нового элемента, а также номера звена, после которого нужно вставить новое звено. Создать новое звено. Вывести содержимое списка на экран.

При выборе третьего пункта вывести приглашение на ввод звена списка, которое нужно удалить. Произвести удаление звена. Вывести содержимое списка на экран.

При выборе четвертого пункта меню осуществить очистку памяти

Лабораторная работа №5

Сортировка списков

Цель работы

Закрепление навыков работы со ссылочными реализациями структур данных. Создание приложения, осуществляющего ввод, редактирование, сортировку и удаление однонаправленного списка.

Задание на лабораторную работу

Создать приложение со следующим функционалом:

- приложение предназначено для создания и редактирования однонаправленных списков;
- выбор операции над списком производится пользователем в режиме меню;
- ввод данных производится с клавиатуры;
- сортировка производится по строковому полю методом пузырька.

Варианты заданий

Список организовать согласно варианту из лабораторной работы № 3

Порядок выполнения работы

Преобразовать программу из лабораторной работы №4 согласно варианту задания. Добавить процедуру сортировки списка.

В основной части программы вывести приглашение пользователю на ввод выбор операции над списком:

1) создание нового списка;

2) сортировка списка;

3) добавление нового звена в список;

4) удаление звена из списка;

5) удаление списка;

6) выход.

Сделать обработчик выбора пунктов меню с вызовом соответствующих процедур и функций работы со списком.

Лабораторная работа №6

Введение в объектно-ориентированное программирование. Наследование

Цель работы

Знакомство с основными принципами объектно-ориентированного программирования и синтаксисом языка Pascal для создания объектно-ориентированных приложений. Практическое применение принципа наследования – построение иерархической схемы классов.

Введение

В языке Pascal основным элементом объектно-ориентированных приложений является *объект*. Синтаксис описания объекта:

type

<имя объекта> = object

<список членов(поля и заголовки методов)>;

end;

Члены класса можно разделить на две группы:

Поля данных – данные, определяющие состояние оюхекта

Методы – процедуры и функции, которые могут использовать поля объекта и внешние данные.

Каждый член класс имеет атрибут доступа, при помощи которого определяется “зона видимости” для члена класса. Всего таких атрибутов три:

1. public – член объекта может использоваться любой функцией(процедурой);
2. private – член объекта может использоваться только функциями-

членами объекта;

3. protected – член объекта может использоваться функциями-членами объекта, а также членами объектов, для которого данный объект является базовым(предком).

Основные принципы ООП:

Инкапсуляция(encapsulation). Комбинирование записей с процедурами и функциями, манипулирующими полями этих записей, формирует новый тип данных – объект.

Инкапсуляция производится таким образом, чтобы пользователь оюъекта мог видеть и использовать только интерфейсную часть класса (т. е. список декларируемых свойств и методов объекта и не вникать в его внутреннюю реализацию. Поэтому данные принято инкапсулировать в классе таким образом, чтобы доступ к ним по чтению или записи осуществлялся не напрямую, а с помощью методов. Принцип инкапсуляции (теоретически) позволяет минимизировать число связей между объектами и, соответственно, упростить независимую реализацию и модификацию классов.

2. Наследование(inheritance). Наследованием называется возможность порождать один объект от другого с сохранением всех свойств и методов объекта-предка (прародителя) и добавляя, при необходимости, новые свойства и методы. Набор объектов, связанных отношением наследования, называют иерархией.

Наследование призвано отобразить такое свойство реального мира, как иерархичность.

Важно помнить то, что если характеристика однажды определена на каком-то уровне иерархии, то все объекты, расположенные ниже данного уровня, содержат эту характеристику.

3. **Полиморфизм(polymorphism).** Присваивание действию одного имени, которое затем совместно используется вниз и вверх по иерархии объектов, причем

каждый объект иерархии выполняет это действие способом, именно ему подходящим.

Экземпляры объектных типов

Экземпляры объектных типов описываются в точности так же, как в Паскале описывается любая переменная, либо статическая, либо указатель, ссылающийся на размещенную в динамической памяти переменную:

```
var
    Emp: TEmployee;
```

Поля объектов

К полю объекта можно обратиться так же, как к полю обычной записи, либо с помощью оператора with, либо путем уточнения имени с помощью точки. Например:

```
Emp.Rate := 10;
with Emp do
begin
    Name := 'Ivanov';
    Title := 'programmer';
end;
```

Методы

Внутри объекта метод определяется заголовком процедуры или функции, действующей как метод:

```
type
    TEmployee = object
    Name, Title: string[25];
        Rate: Real;
        procedure Init (AName, ATitle: String; ARate:
Real);
    end;
```

Примечание: Поля данных должны быть описаны перед первым описанием

метода.

При определении метода после описания объекта имени метода должно предшествовать имя типа объекта, которому принадлежит этот метод, с последующей точкой:

```
procedure   TEmployee.Init(AName,   ATitle:   string;ARate:
Real);
begin
     Name := AName;
    Title := ATitle;
    Rate := ARate;
end;
```

Создание объекта-наследника

```
<имя объекта> = object(<имя обекта-родителя>)
    <список членов>;
end;
```

При создании объекта-наследника, он наследует все поля и методы базового объекта, описанные атрибутами public и protected. Для этого класса нет необходмсоти снова определять эти поля и методы.

Задание на лабораторную работу

Создать приложение, реализующее следующую иерархию объектов

TPerson

/ \

TStudent TTeacher

Объекты TStudent и TTeacher должны наследовать поля и методы объекта TPerson.

Порядок выполнения работы

Создать объект типа TPerson со следующими полями: фамилия, имя,

отчество, дата рождения – и двумя методами, выводящими на экран ФИО(фамилию, имя, отчество) и дату рождения.

Создать объект TStudent – потомок класса TPerson, для которого определить новые поля (год зачисления в вуз, номер группы) и методы, выводящие этиполя на экран.

Создать объект TTeacher – потомок класса TPerson, для которого определить новое поле (должность) и метод, выводящие это поле.

Создать одну переменную типа TPerson, две переменные типа TStudent и одну переменную типа TTeacher.

В основном блоке программы заполнить все поля всех экземпляров объектов (информацию пользователь вводит с клавиатуры) и вызвать методы.

Результат работы оформить в виде отчета, в котором обязательно привести

- описание всех объектов, их полей и методов;
- скриншоты программы;
- листинг программы.

Лабораторная работа №7

Введение в объектно-ориентированное программирование. Конструкторы и деструкторы

Введение

Существуют специальные методы объектов, которые отвечают за создание экземпляров объекта и их удаление. Это так называемые конструкторы и деструкторы. Конструкторы – методы, основная цель которых заключается в инициализации объекта и распределении памяти для хранения объекта. Как и любой другой метод, конструктор может иметь или не иметь параметров. Конструктор без параметров называется конструктором по умолчанию.

Деструктор разрушает созданный экземпляр.

По своей форме конструкторы и деструкторы являются процедурами, но объявляются с помощью зарезервированных слов constructor и destructor:

Объекты могут размещаться в динамической памяти и ими можно манипулировать с помощью указателей. Паскаль включает несколько мощных расширений для выполнения динамического размещения и удаления объектов более легкими и более эффективными способами.

Free Pascal поддерживает расширенный синтаксис процедур new и dispose. В случае, когда нужно выделить память под динамическую переменную объектного типа, при вызове процедуры new может быть указано имя конструктора объекта:

```
Type
   TObj = object;
    Constructor Init
    Destructor Destroy;
    ...
    end;
   Pobj = ^TObj;
Var PP : Pobj;
```

Следующие 3 вызова эквивалентны:

```
pp := new (Pobj,Init);
new(pp,Init);
new (pp);
```

Таким же образом, для выполнения освобождения памяти деструктор можно вызывать как часть расширенного синтаксиса процедуры dispose:

```
dispose(pp, Destroy);
```

Задание на лабораторную работу

На основе лабораторной работы № 5 создать приложение для работы с объектами типа “список”: создание объекта в динамической памяти, добавление и удаление элементов из списка, уничтожение объекта из динамической памяти.

Порядок выполнения работы

Создать объект типа TList со следующими обязательными методами: Init(конструктор объекта), Print(вывести список на экран), Add_Item(включение нового элементав список), Delete_Item(удаление элемента из списка), Destroy(деструктор объекта) – и обязательным полем First, в котором хранится указатель на первый элемент.

В основном блоке программы в динамической памяти создать экземпляр объекта. Осуществить редактирование списка в форме диалога с пользователем (интерфейс должен быть эргономичным!!!). При выходе из программы очистить память, занимаемую списком.

Результат работы оформить в виде отчета, в котором кроме обязательных пунктов (титульный лист, цель работы, задание на работу с вариантом, заключение

и листинг программы) обязательно привести:

- подробное описание всех объектов (назначение), их полей(назначение и тип данных) и методов (назначение и расширенный список формальных параметров);
- описание использования конструктора и деструктора объекта;
- скриншоты программы(на каждую операцию со списком).