Разобрать все примеры. В качестве среды разработки использовать PascalABC (не .NET) или Lazarus (GUI или консольное приложение: Файл - Создать - Проект - Простая программа

Работа с указателями

Как правило, при обработке оператора объявления переменной
```
имя_переменной: тип_переменной
```
компилятор автоматически выделяет память под переменную `имя_переменной` в соответствии с указанным типом. Доступ к объявленной переменной осуществляется по ее имени. При этом все обращения к переменной заменяются адресом ячейки памяти, в которой хранится ее значение. При завершении подпрограммы, в которой была описана переменная, память автоматически освобождается.

Доступ к значению переменной можно получить иным способом – определить собственные переменные для хранения адресов памяти. Такие переменные называют указателями.

Указатель[30] – это переменная, значением которой является адрес памяти, где хранится объект определенного типа (другая переменная). Как и любая переменная, указатель должен быть объявлен. При объявлении указателей всегда указывается тип объекта, который будет храниться по данному адресу:

```
var имя_переменной: ^тип;
```

Такие указатели называют *типизированными*. Например:

```
var p: ^integer;
//По адресу, записанному в переменной p
//будет храниться переменная типа int
//или, другими словами p, указывает
//на тип данных integer.
```

В языке Free Pascal можно объявить указатель, не связанный с каким-либо конкретным типом данных. Для этого применяют служебное слово `pointer`:

```
var имя_переменной: pointer;
```

Например:

```
var x, y: pointer;
```

Подобные указатели называют *нетипизированными* и используют для обработки данных, структура и тип которых меняется в процессе выполнения программы, то есть динамически.

2.5.4 Операции над указателями

При работе с указателями используют операции получения адреса и разадресации.

Операция получения адреса `@` возвращает адрес своего операнда. Например:

```
Var
a:real; //Объявлена вещественная переменная a
adr_a:^real; //Объявлен указатель на тип real
…
//Оператор записывает в переменную adr_a
//адрес переменной a
adr_a:=@a;
```

Операция разадресации `^` возвращает значение переменной, хранящееся по заданному адресу:

```
Var
a:real; //Объявлена вещественная переменная a
adr_a:^real; //Объявлен указатель на тип real
…
//Оператор записывает в переменную a
//значение, хранящееся по адресу adr_a.
a:=adr_a^;
```

5.11 Использование указателей для работы с динамическими массивами

Все объявленные в программе статические переменные, которые мы рассматривали до этого момента, размещаются в одной непрерывной области оперативной памяти, которая называется сегментом данных. Для работы с массивами большой размерности можно воспользоваться так называемой динамической памятью, которая выделяется программе после запуска программы на выполнение. Размер динамической памяти можно варьировать в широких пределах. По умолча-

нию этот размер определяется всей доступной памятью ПК.

Динамическое размещение данных осуществляется компилятором непосредственно в процессе выполнения программы. При динамическом размещении заранее неизвестно количество размещаемых данных. Кроме того, к ним нельзя обращаться по именам, как к статическим переменным.

Оперативная память ПК представляет собой совокупность элементарных ячеек для хранения информации – байтов, каждый из которых имеет собственный номер. Эти номера называются адресами, они позволяют обращаться к любому байту памяти.

5.11.1 Работа с динамическими переменными и указателями

Free Pascal имеет гибкое средство управления памятью – указатели.

Указатель – переменная, которая в качестве своего значения содержит адрес байта памяти. Указатель занимает 4 байта.

Как правило, указатель связывается с некоторым типом данных. В таком случае он называется типизированным. Для его объявления используется знак ^, который помещается перед соответствующим типом, например:

```
type massiv=array [1..2500] of real;
var a:^integer; b,c:^real; d:^massiv;
```

В языке Free Pascal можно объявлять указатель, не связывая его с конкретным типом данных. Для этого служит стандартный тип `pointer`, например:

```
var p,c,h: pointer;
```

Указатели такого рода будем называть *нетипизированными*. Поскольку нетипизированные указатели не связаны с конкретным типом, с их помощью удобно динамически размещать данные, структура и тип которых меняются в ходе работы программы.

Значениями указателей являются адреса переменных памяти, поэтому следовало ожидать, что значение одного из них можно передавать другому. На самом деле это не совсем так. Эта операция проводится только среди указателей, связанных с одними и теми же типами данных.

Например:

```
Var
p1, p2:^integer; p3:^real; pp:pointer;
```

В этом случае присваивание `p1:=p2;` допустимо, в то время как `p1:=p3;` запрещено, поскольку `p1` и `p3` указывают на разные типы данных. Это ограничение не распространяется на нетипизированные указатели, поэтому можно записать `pp:=p3; p1:=pp;` и достичь необходимого результата.

Вся динамическая память во Free Pascal представляет собой сплошной массив байтов, называемый кучей. Физически куча располагается за областью памяти, которую занимает тело программы.

Начало кучи хранится в стандартной переменной `heaporg`, конец – в переменной `heapend`. Текущая граница незанятой динамической памяти хранится в указателе `heapprt`.

Память под любую динамическую переменную выделяется процедурой `new`, параметром обращения к которой является типизированный указатель. В результате обращения последний принимает значение, соответствующее динамическому адресу, начиная с которого можно разместить данные, например:

```
var
i,j:^integer;
r:^real;
begin
new(i);
new(R);
new(j)
```

В результате выполнения первого оператора указатель `i` принимает значение, которое перед этим имел указатель кучи `heapprt`. Сам `heapprt` увеличивает свое значение на 4, так как длина внутреннего представления типа `integer`, связанного с указателем `i`, составляет 4 байта. Оператор `new(r)` вызывает еще одно смещение указателя `heapprt`, но уже на 8 байтов, потому что такова длина внутреннего представления типа `real`. Аналогичная процедура применяется и для переменной любого другого типа. После того как указатель стал определять конкретный физический байт памяти, по этому адресу можно разместить любое значение соответствующего типа, для чего сразу за указателем без каких-либо пробелов ставится значок `^`, например:

```
i^:=4+3;
j^:=17;
r^:=2 * pi;
```

Таким образом, значение, на которое указывает указатель, то есть собственно данные, размещенные в куче, обозначаются значком ^. Значок ^ ставится сразу за указателем. Если после указателя значок ^ отсутствует, то имеется в виду адрес, по которому размещаются данные. Динамически размещенные данные (но не их адрес!) можно использовать для констант и переменных соответствующего типа в любом месте, где это допустимо, например:

```
r^:=sqr(r^)+sin(r^+i^)-2.3
```

Невозможен оператор

```
r:=sqr(r^)+i^;
```

так как указателю r нельзя присвоить значение вещественного типа.

Точно так же недопустим оператор

```
r^:=sqr(r);
```

поскольку значением указателя r является адрес, и его (в отличие от того значения, которое размещено по данному адресу) нельзя возводить в квадрат. Ошибочным будет и присваивание r^:=i, так как вещественным данным, на которые указывает r^, нельзя давать значение указателя (адрес). Динамическую память можно не только забирать из кучи, но и возвращать обратно. Для этого используется процедура dispose(p), где p – указатель, который не изменяет значение указателя, а лишь возвращает в кучу память, ранее связанную с указателем.

При работе с указателями и динамической памятью необходимо самостоятельно следить за правильностью использования процедур new, dispose и работы с адресами и динамическими переменными, так как транслятор эти ошибки не контролирует. Ошибки этого класса могут привести к зависанию компьютера, а то и к более серьезным последствиям!

Другая возможность состоит в освобождении целого фрагмента кучи. С этой целью перед началом выделения динамической памяти текущее значение указателя heapptr запоминается в переменной-указателе с помощью процедуры mark. Теперь можно в любой момент освободить фрагмент кучи, начиная с того адреса, который запомнила процедура mark, и до конца динамической памяти. Для этого используется процедура release.

Процедура mark запоминает текущее указание кучи heapptr

(обращение `mark(ptr)`, где `ptr` – указатель любого типа, в котором будет возвращено текущее значение `heapptr`). Процедура `release(ptr)`, где `ptr` – указатель любого типа, освобождает участок кучи от адреса, хранящегося в указателе до конца кучи.

5.11.2 Работа с динамическими массивами с помощью процедур getmem и freemem

Для работы с указателями любого типа используются процедуры `getmem`, `freemem`. Процедура `getmem(p,size)`, где `p` – указатель, `size` – размер в байтах выделяемого фрагмента динамической памяти (`size` типа `word`), резервирует за указателем фрагмент динамической памяти требуемого размера.

Процедура `freemem(p,size)`, где `p` – указатель, `size` – размер в байтах освобождаемого фрагмента динамической памяти (`size` типа `word`), возвращает в кучу фрагмент динамической памяти, который был зарезервирован за указателем. При применении процедуры к уже освобожденному участку памяти возникает ошибка.

После рассмотрения основных принципов и процедур работы с указателями возникает вопрос: а зачем это нужно? В основном для того, чтобы работать с так называемыми динамическими массивами. Последние представляют собой массивы переменной длины, память под которые может выделяться (и изменяться) в процессе выполнения программы, как при каждом новом запуске программы, так и в разных её частях. Обращение к `i`-му элементу динамического массива `x` имеет вид `x^[i]`.

Рассмотрим процесс функционирования динамических массивов на примере решения следующей задачи.

ЗАДАЧА 5.2. Найти максимальный и минимальный элементы массива X(N).

Вспомним решение задачи традиционным способом.

```
Program din_mas1;
Var
 x:array [1..150] of real;
i,n:integer; max,min:real;
begin
 writeln('введите размер массива');
 readln (n);
for i:=1 to N do
```

```
begin
 write('x[',i,']='); readln(x[i]);
end;
max:=x[1]; min:=x[1];
for i:=2 to N do
begin
 if x[i] > max then max:=x[i];
 if x[i] < min  then  min:=x[i];
end;
 writeln('максимум=',max:1:4);
 writeln('минимум=',min:1:4);
end.
```

Теперь рассмотрим процесс решения задачи с использованием указателей. Распределение памяти проводим с помощью процедур new-dispose (программа din_mas2) или getmem-freemem (программа din_mas2).

```
 type massiw= array[1..150]of real;
 var
 x:^massiw;
 i,n:integer;
 max,min:real;
begin
{Выделяем память под динамический
массив из 150 вещественных чисел.}
 new(x);
 writeln('Введите размер массива');
 readln(n);
for i:=1 to N do
begin
 write('x(',i,')=');
 readln(x^[i]);
end;
max:=x^[1];min:=x^[1];
for i:=2 to N do
begin
 if x^[i] > max then max:=x^[i];
 if x^[i] < min then min:=x^[i];
end;
```

```
writeln('максимум=',max:1:4,
                    ' минимум=',min:1:4);
{ Освобождаем память. }
dispose(x);
end.
type
 massiw=array[1..150]of real;
var
 x:^massiw;
 i,n:integer;max,min:real;
begin
writeln('Введите размер массива');
readln(n);
{ Выделяем память под n элементов массива. }
getmem(x,n*sizeof(real));
for i:=1 to N do
begin
 write('x(',i,')=');
 readln(x^[i]);
end;
max:=x^[1];min:=x^[1];
for i:=2 to N do
begin
 if x^[i] > max then max:=x^[i];
 if x^[i] < min then min:=x^[i];
end;
writeln('максимум=',max:1:4,
                    ' минимум=',min:1:4);
{ Освобождаем память. }
freemem(x,n*sizeof(real));
end.
```

При работе с динамическими переменными необходимо соблюдать следующий порядок работы:

1. Описать указатели.
2. Выделить память под массив (функции `new` или `getmem`).
3. Обработать динамический массив.
4. Освободить память (функции `dispose` или `freemem`).

Примеры

Необходимость использования переменных ссылочного типа возникает при различных ситуациях. Рассмотрим наиболее часто встречающиеся ситуации.

СЛУЧАЙ 1. Программа должна работать с большими объемами данных, причем нет необходимости выделять память сразу под все переменные. Память под объект отводится тогда, когда в этом возникает необходимость, и это место освобождается, когда необходимость в использовании соответствующего объекта исчезает.

Задача 1. Дано 10000 вещественных чисел $a_1, a_2, \ldots, a_{10000}$ и 8000 целых положительных чисел $b_1, b_2, \ldots, b_{8000}$. Вывести числа $a_1, a_2, \ldots, a_{10000}$ в обратном порядке. Вывести то число b_k, номер которого k равен минимальному из чисел $b_1, b_2, \ldots, b_{8000}$.

Программа

```
program Project1;
uses crt;
type
        ra = array [1 .. 10000] of real;
        ia = array [1 .. 8000] of integer;
        pr = ^ra; pi = ^ia; {Определили ссылочный тип}
var  k, i ,min: Integer;
     f : pr;
     g : pi; {Описали ссылочные переменные}
begin
new (f); {Выделили память под массив из 10000 элементов}
for i := 1 to 10000 do f^[i]:=random(10);
for i := 10000 downto 1 do write(f^[i]:3:1);
writeln;
dispose (f); {Освободили выделенную память}
new(g); {Выделили память под массив из 8000 элементов}
read(g^[1]); {ввели с клавиатуры первый элемент массива}
min:=g^[1]; k :=1; {полагаем,что минимальный элемент равен первому элементу, и его номер равен единице}
for i := 2 to 8000 do
begin
   g^[i]:=1+random(10);
   if g^[i] < min then
         begin min:=g^[i]; k := min; end;
end;
writeln('min = ',min);
```

```
    writeln('число с индексом =',k);
    writeln(g^[k]:3);
    dispose(g);
    readkey;
    end.
```

В этой программе сначала отводится место для массива из 10000 элементов. Переменные f^[1], ..., f^[10000] получают значения с помощью оператора f^[i]:=. После решения первой задачи – вывода этих чисел в обратном порядке, место в памяти, выделенное для объекта типа ra, освобождается с помощью dispose(f). После этого отводится место в памяти для массива из 8000 элементов. Переменные g^[1], ..., g^[8000] получают значения с помощью оператора g^[i]:=, и решается вторая часть задачи. Память для объектов типа ra и ia выделяется поочередно, так что эти объекты могут располагаться в пересекающихся областях памяти вычислительной машины.

СЛУЧАЙ 2. Программа во время компиляции использует данные, для которых заранее неизвестен необходимый объем памяти. Некоторые элементы данных, например, строки и массивы, требуют задания максимально возможного их размера во время компиляции. Объем памяти, необходимый для хранения этого максимально возможного количества данных, будет зарезервирован даже в том случае, если в этих массивах или строках не хранится никаких данных. Если расположить переменные в динамически распределяемой области памяти во время выполнения программы, то для хранения данных будет выделено столько байтов памяти, сколько потребуется.

Задача 2. Рассчитать сумму элементов вектора a_1, состоящего из n элементов. Сформировать вектор a_2 из элементов вектора a_1, расположенных в обратном порядке и уменьшенных на единицу.

Объявим тип Vector для массива, состоящего из одного элемента. Переменную типа Vector объявлять не будем, а объявим типизированный указатель, базовым типом данных для которого будет Vector. В процессе работы программы определим количество элементов вектора с помощью оператора readln(n) и выделим память необходимого размера. Выделять память будем при помощи процедуры Getmem, освобождать – при помощи процедуры Freemem.

```
Программа
type
   vector = array [1..1] of integer;
var  a1, a2: ^vector;
     p : pointer;
     n, i, s : integer;
begin
     write('Введите число элементов вектора: ');
     readln(n);
     S := 0;
     getmem(a1, n*sizeof(integer)); {Выделяем память под вектор a1}
     for i := 1 to n do
         begin
           a1^[i]:=random(15); {Вводим элементы вектора a1}
           S := S+a1^[i]; {Суммируем элементы вектора}
         end;
     writeln('S=',S);
     Getmem(a2, n*sizeof(integer)); {Выделяем память под вектор a2}
     for i:= 1 to n do
         begin
             a2^[n+1-i]:= a1^[i] - 1; {Формируем вектор a2}
             writeln(a2^[i]); {Выводим элементы вектора a2}
         end;
     freemem(a1, n*sizeof(integer));
     freemem(a2, n*sizeof(integer));
end.
```

Тип Vector объявлен как массив, индексы которого изменяются в диапазоне от 1 до 1.

Задача 3. Считать 1000 строк (максимально возможное число) из файла и записать их в динамическую память. Неизвестно, насколько длинной будет каждая из строк, поэтому потребуется описать строковый тип такого размера, который будет соответствовать максимально возможной строке. Чтобы решить эту проблему, можно считать каждую строку в буфер, затем выделить столько памяти, сколько требуется для фактических данных в строке.

Программа

```
type
    PString = ^string;
var
    buf : string;
    L : array[1..1000] of PString;
    ff : text;
    i: integer;
begin
    assign(ff, 'dat.txt');
    reset(ff);
    i:=0;
    while not eof(ff) do
    begin
        readln(ff, Buf);
        getmem(L[i], length(buf) + 1);
        writeln('выделено памяти под строку ', length(buf) + 1);
        L[i]^ := buf;
        inc(i);
    end;
    writeln('количество строк в файле = ', i);
end.
```

Вместо выделения для строк 256К (256 символов на строку 1000 раз) программа выделила 4К (4 байта на указатель 1000 раз).

Самостоятельно создать текстовый файл из большого количества строк разной длины и проверить работу программы.

СЛУЧАЙ 3. В программе необходимо использовать массив большой размерности.

Задача 4. Найти среднее значение элементов массива d, состоящего из 200 строк и 200 столбцов. Под элементы двухмерного массива такого размера потребуется 200x200x6 байтов памяти, что составит более 234К. Предположим, что переменная d:array[1..200, 1..200] of Real такого размера не может быть объявлена. Нужно реорганизовать массив так, чтобы он распался на части, не превышающие по отдельности 64К.

Программа

```
type
     d200 = array [1 .. 200] of real;{Объявлен тип строки матрицы}
     dPtr = ^d200;{Объявлен тип ссылки на строку}
     dPtr200 = array [1..200] of dPtr;{Тип для массива указателей}
var  d : dPtr200; {Объявили массив указателей, каждый элемент кото-
рого указывает на одномерный массив d200}
     s : real;
     i, j : integer;
begin
     s:=0;
     for i := 1 to 200 do
     begin
          New(d[i]); {Выделили память под строку из 200 элементов
     типа real, на которую указывает ссылка d[i]}
          for j := 1 to 200 do
          begin
               d[i]^[j] := random(10); {Формирование элемента с помо-
          щью датчика случайных чисел и подсчет суммы элементов }
               s := s + d[i]^[j];
          end;
          dispose(d[i]);
     end;
     s := s/40000; writeln('S = ', s:2:2);
end.
```

Матрица 200 на 200 в программе определяется как статический массив из 200 ссылок на динамические массивы по 200 элементов. Одновременно память выделяется только для одной строки. Выделение памяти для каждой следующей строки происходит после освобождения памяти, занятой предыдущей строкой. Для каждого массива понадобится блок длиной 200x6 байтов. Обращение к элементу массива d (разыменование) имеет вид: d[i]^[j], где i – номер строки, а j – номер столбца.

СЛУЧАЙ 4. В программе необходимо применять ссылки на данные, имеющие разную структуру.

Задача 5. Компоненты файла f1 являются объектами типа “игрушка”. Компоненты

файла f2 являются объектами типа “багаж”. “Игрушка” – это запись, состоящая из следующих полей: наименование игрушки, цена игрушки, возрастные границы. “Багаж” – это запись, состоящая из полей: количество предметов, вес предметов. В текстовом файле f расположена последовательность $a_1 b_1 a_2 b_2 \dots a_n b_n$, где каждое a_i – это символ “i” или “б”, а каждое b_i –натуральное число.

Написать программу, в результате выполнения которой для каждого $i = 1, 2, \dots, n$ выводится: если a_i есть символ “i” – название и стоимость игрушки, информация о которой содержится в b_i-й компоненте файла f1; если a_i есть символ “б” – количество мест и вес багажа, информация о котором содержится в b_i-й компоненте файла f2. Программу составить так, чтобы в каждый момент ее выполнения было выделено место в памяти не более, чем для одного объекта типа “игрушка” или “багаж”.

Программа

```
type
     Igru = record {Объявили записной тип “игрушка”}
          name : string[15];
          cost : integer;
          age1, age2 : integer;
     end;
     Bagaz = record {Объявили записной тип “багаж”}
          count : integer;
          w : real;
     end;
var
     ig : ^Igru;
     ba : ^Bagaz; {Объявили ссылочные переменные, которые будут ссылаться на переменные типа запись}
     f1 : file of Igru;
     f2 : file of Bagaz;
     f : text;
     i, k : integer;
     a, b: char;
     code : word;
begin
assign(f,’dat3.txt'); assign(f1,’dat1.dat'); assign(f2,’dat2.dat');
```

```
    {Связали файловые переменные с физическими файлами на диске}
    reset(f); reset(f1); reset(f2); {Открыли файлы на чтение}
    while eof(f) = false do {Организуем цикл}
    begin
        read(f, a, b); {Считали из текстового файла два символа}
        val(b, k, code); {Превратили второй символ в число}
        if a = 'i' then
        begin {Блок для работы с объектом типа "игрушка"}
            new(Ig);
            seek(f1, k); {Установили указатель файла на компонент
        k}
            read(f1, Ig^); writeln;
            with Ig^ do
            begin
                writeln('Наименование ',name);
                writeln('Цена ',cost);
                writeln('Нижняя возрастная граница ', age1);
                writeln('Верхняя возрастная граница ', age2);
            end;
        dispose(Ig);
        end
        else
        begin {Блок для работы с объектом типа "багаж"}
            new(Ba);
            seek(f2, k); read(f2, Ba^);
            with Ba^ do
            begin
                writeln('Количество ', count);
                writeln('Вес ', w:8:2);
            end;
        dispose(Ba);
        end;
    end;
    close(f); close(f1); close(f2);
end.
```

До тех пор, пока не будет достигнут конец файла f, считываются два символа a и b. Символ a указывает на то, с какой структурой придется работать. Символ b содержит номер компоненты типизированного файла. Перемещение на k-ую компоненту файла осуществляется с помощью функции seek.

Пример. Расширение по ходу решения задачи матрицы A с элементами типа real.

```
type
    tip = real;
const
      m=65520 div SizeOf(tip);
      mem = 1000;
type
     stroka = array [1..m] of tip;
var   i,j,k:word;
      z:char;
      n:word;
      a:array[1..200] of ^stroka;{положим, что число строк не более
200}
begin
     writeln('Укажите число столбцов матрицы A');
     readln(n);
     i:=1;
     getmem(a[1], n*sizeof(tip)); {Выделяем память под 1-ую строку
матрицы A}
     repeat
          for k:=1 to 100 do
          for j:=1 to n do a[i]^[j]:=j; {заполнение строки}
          writeln('Добавить в матрицу еще строку? Ответ: Y,N');
          readln(z);
          if z in ['Y','y'] then
          begin
               i:=i+1;
               getmem(a[i],n*sizeof(tip))
          end;
```

```
          until n*sizeof(tip)>mem;
writeln('Заданная память (mem) исчерпана. Создано', i:3,' строк');
readln;
end.
```

Оператором FreeMem(A[r], N*SizeOf(tip)) можно освободить память, занятую любой r-й строки, но перенумерации следующих за нею строк не произойдет. После оператора FreeMem нужно выполнить цикл копирования указателей:

For r:=r to i-1 do a[r] :=a[r+1]

и задать значение nil i-му (теперь ненужному) указателю.