Практическая работа 6

Типы данных

Ответьте на вопросы:

1. Что такое запись? Из чего она состоит. Приведите свой пример.

**Записи в Паскале – структурированный комбинированный тип данных**. Запись состоит из определенного числа компонент,**называемых полями, которые могут быть разного типа**.

**Пример**: создать запись для работы с датами. Запись должна быть с тремя полями, соответствующими месяцу, дню и году.

1. Как объявляются и создаются записи в Паскале? Как обращаться к полям записи? Приведите примеры.

Создание записи в Паскале:

Пример:

type mydate = record

month: 1..12;

day: 1..31;

year: integer

end;

var d: mydate;

В примере переменная mydate — запись, состоящая из трех полей: month, day и year. Каждое поле содержит соответственно данные: целое число в пределах от 1 до 12 — номер месяца (интервальный тип), целое число от 1 до 31 — число месяца (интервальный тип), целое число — год.

Обращение к полям записи:

Пример:

type mydate = record

month: 1..12;

day: 1..31;

year: integer

end;

var d: mydate;

begin

d.day:=1;

d.month:=12;

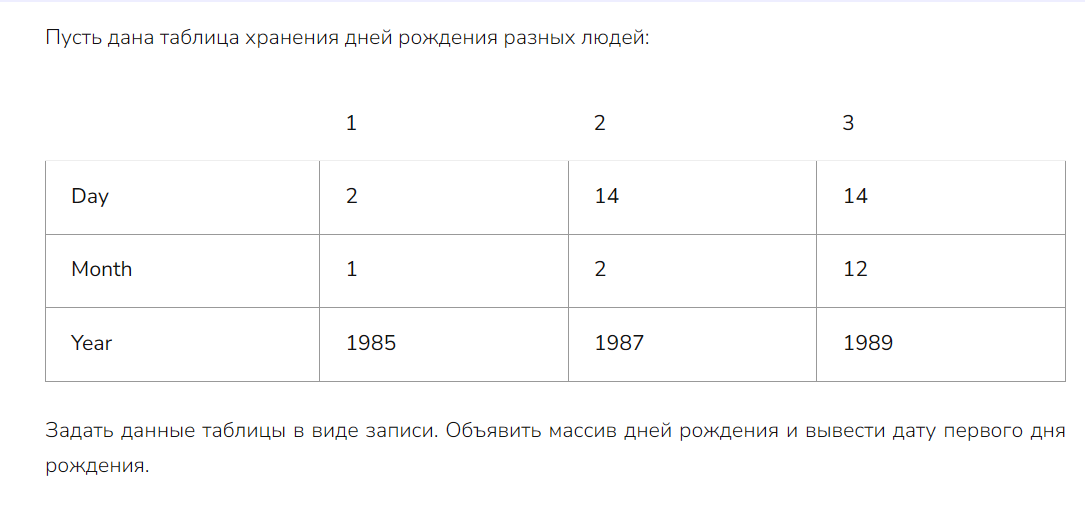
readln(d.year);

writeln(d.day,'/',d.month,'/',d.year);

end.

1. Зачем используются записи в виде двумерной таблицы? Приведите пример использования.

Часто записи используются **в виде двумерной таблицы**, каждый столбец которой имеет свой тип.  
Таким образом, если описана двумерная таблица, то ее начальные значения задаются как вектор, каждый компонент которого является записью.



1. Как использовать конструкции with при работе с записями?

При работе с записями есть возможность избавиться от постоянного префикса в виде обращения к названию переменной с помощью with.

type zap1= record

day:1..31;

month: 1..12;

year: 1900..2100;

end;

var my\_birthday: zap1;

begin

with my\_birthday do

begin

day:= 17;

month:= 3;

year:= 2004;

end;

...

end.

1. Как происходит считывание/запись записей в файл?

type

t\_subscriber = record

surname: string[20];

tel: LongInt;

end;

var

subscriber: t\_subscriber;

f: file of t\_subscriber;

i: Integer;

begin

Assign(f,'notebook.dat');

Rewrite(f);

for i:=1 to 5 do begin

with subscriber do begin

Write('Surname: ');

ReadLn(surname);

Write('Phone: ');

ReadLn(tel);

end;

Write(f, subscriber);

end;

Close(f);

end.

1. Что такое множество в Паскале? Как оно задается?

**Множества в Паскале** — это некоторое собрание элементов, одно и того же базового типа.

Пример: Создать множество дней недели, базовым типом которого является перечисляемый тип из названий дней недели.

type week\_days = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);

var work\_days: set of week\_days;

begin

work\_days:=[Mon, Wed, Thu];

end.

1. Какие действия можно выполнять с множествами? Приведите примеры.

**Объединение двух множеств** A и B (A + B) – это новое множество, состоящее из элементов, принадлежащих множеству A или B либо тому и другому одновременно

Пример:

var ch1,ch2, ch3: set of char;

begin

ch1:=['a', 'b', 'd'];

ch2:=['m', 'd', 'e'];

ch3:=ch1 + ch2 + ['k', 'n'];

*{Результат: ch3 = ['a', 'b', 'd', 'm', 'e', 'k', 'n']}*

end.

**Пересечение двух множеств** A и B (A \* B) – это множество, состоящее из элементов, одновременно принадлежащих множествам A и B.

Пример:

var ch1, ch2, ch3: set of char;

begin

ch1:=['a', 'b', 'd'];

ch2:=['m', 'd', 'e'];

ch3:=ch1 \* ch2;

*{Результат: ch3 = ['d'] }*

end.

**Разность двух множеств** A и B (A – B) – это новое множество, состоящее из элементов множества A, не вошедших в множество B.

Пример:

var ch1, ch2, ch3: set of char;

begin

ch1 := ['a', 'e', 't'];

ch2 := ch1 – ['e']; *{ ['a', 't'] }*

ch3 := ['m', 'n', 't'] – ch2; *{ ['m', 'n'] }*

end.

1. Для чего используется операция IN при работе с множествами? Приведите примеры.

Операция in необходима для поиска определенного элемента в величине типа set, т.е. в множестве.

Так, если x есть элемент множества a, то (x in a) дает true.

**Пример:** создать перечисляемый тип из дней недели и множество из рабочих дней недели. Определить, является ли понедельник рабочим днем.

var

work\_days,days\_off: set of byte;

begin

work\_days:=[1, 2];

days\_off:=[6, 7];

if 1 in work\_days then

writeln ('понедельник - рабочий день')

else

writeln ('понедельник - не рабочий день');

if 6 in days\_off then

writeln ('суббота - выходной день')

else

writeln ('суббота - не выходной день');

end.

1. Опишите динамические данные, в чем их особенность?

О динамических данных

* размер динамических данных заранее неизвестен;
* память под них выделяется во время исполнения программы;
* динамические данные, как правило, не имеют идентификаторов (имени в стандартном понимании статических данных), но имеют адрес в памяти;
* обращение к динамическим данным осуществляется по их адресу в памяти.

О памяти

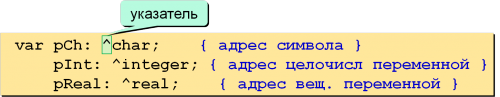
* Если говорить об оперативной памяти в программировании, то вводится понятие статической и динамической памяти. Оперативная память неодинакова, она имеет различные области: для статической памяти, для динамической памяти.
* Обращение к участку динамической памяти происходит с помощью специальной ссылочной переменной, которая называется указателем или ссылкой.

1. Что такое указатель? Как он объявляется? Как присваиваются значения? Приведите примеры.

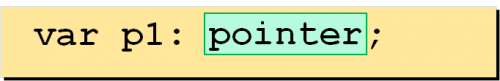
Указатели в Паскале необходимы при работе с динамической памятью. Переменная типа «указатель» в качестве своего значения содержит адрес участка динамической памяти, с которой связан этот указатель.

Объявление

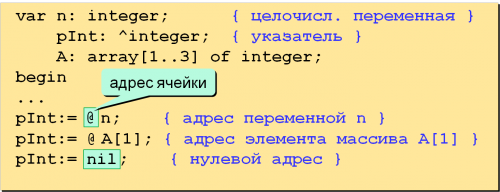
* Типизированные указатели:

[](https://labs-org.ru/wp-content/uploads/2017/08/1.png)

* Универсальные нетипизированные указатели могут хранить адрес переменной любого типа:

[](https://labs-org.ru/wp-content/uploads/2017/08/1_11.png)

Присваивание значений

[](https://labs-org.ru/wp-content/uploads/2017/08/1-1.png)

1. Что такое список? Какие виды списков бывают?

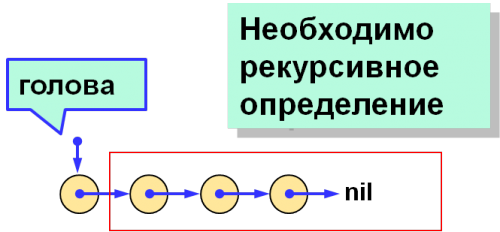
Список состоит из конечного множества динамических элементов, размещающихся в разных областях памяти. Благодаря указателям элемент списка объединены в логически упорядоченную последовательность. Каждый элемент списка организован следующим образом: содержит поля с полезной информацией; содержит специальное поле (или несколько полей), которое хранит адрес другого элемента списка.

1. Односвязный (линейный) список.
2. Двусвязный список.
3. Кольцевой список.
4. Что такое односвязный список? Как он выглядит? Как происходит объявление типа данных?

Список состоит из начального узла — головы — и связанного с ним списка.

Каждый элемент списка содержит информационную и ссылочную части. Т.е. каждый элемент имеет информационное поле (поля) — полезная информация — и ссылку (ссылки), то есть адрес на другой элемент списка.

Односвязный (линейный) список: структура, каждый элемент которой «знает» адрес только следующего за ним элемента.

[](https://labs-org.ru/wp-content/uploads/2017/08/1-4.png)

Объявление:

type PNode = ^Node; *{ указатель на узел }*

Node = record *{ структура узла }*

word: string[40]; *{ слово }*

count: integer; *{ счетчик повторов слов }*

next: PNode; *{ ссылка на следующий }*

end;

1. Опишите алгоритм работы с алфавитно-частотным словарем. Опишите используемые подпрограммы.

Алгоритм алфавитно-частотного словаря

1. Открываем файл в режиме на чтение (var F: Text;)
2. Считывание очередного слова из файла.
3. Если слов больше не осталось (достигнут конец файла), то переходим к шагу номер 7.
4. Слово нашлось — значит, увеличиваем счетчик слов.
5. Если в списке искомого слова не существует, то выполняем:
   * создание нового узла с заполнением необходимых полей: функция CreateNode();
   * поиск узла, перед которым добавляем слово: функция MakePlace();
   * добавление узла: функция AddBefore().
6. Возвращаемся к шагу номер 2.
7. Закрываем рабочий файл.
8. Печатаем на экран список слов: для этого используем алгоритм перемещения по списку.

Подпрограммы:

Создание узла списка

Для решения нашей задачи запрограммируем функцию для создания узла. На вход функции подается новое слово, возвращает функция адрес нового узла, созданного в памяти.

|  |
| --- |
| function CreateNode(NewWord: string): PNode;  var NewNode: PNode;  begin  New(NewNode);  NewNode^.word := NewWord;  NewNode^.count := 1;  NewNode^.next := nil;  Result := NewNode;  end; |

Добавление узла списка

Выполним добавление узла списка на Паскале в виде процедуры:

|  |
| --- |
| procedure AddFirst ( var Head: PNode; NewNode: PNode );  begin  NewNode^.next := Head;  Head := NewNode;  end; |

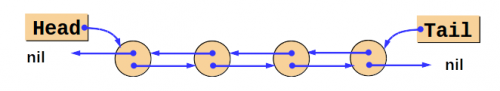
Считывание одного слова из файла

Для этого создадим функцию TakeWord().  
Будем игнорировать все ненужные символы, код которых <= 32.

|  |
| --- |
| function TakeWord ( F: Text ) : string;  var c: char;  begin  Result := ''; *{ пустая строка }*  c := ' '; *{ пробел – чтобы войти в цикл }*  *{ пропускаем спецсимволы и пробелы }*  while not eof(f) and (c <= ' ') do  read(F, c);  *{ читаем слово }*  while not eof(f) and (c > ' ') do begin  Result := Result + c;  read(F, c);  end;  end; |

1. Что такое двусвязные списки? В чем особенность работы с двусвязными списками?

Двусвязный список позволяет двигаться в обоих направлениях.  
При работе с двусвязными списками создаются два указателя.

[](https://labs-org.ru/wp-content/uploads/2017/08/1-7.png)

Рассмотрим структуру:

|  |
| --- |
| type PNode = ^Node; *{ указатель на узел }*  Node = record *{ узел и его структура }*  word: string[40]; *{ поле для слова }*  count: integer; *{ поле - счетчик повторений }*  next: PNode; *{ поле - ссылка на следующий узел }*  prev: PNode; *{ поле - ссылка на предыдущий узел }*  end; |

«Обнуляем» адреса головы и хвоста:

|  |
| --- |
| var Head, Tail: PNode;  ...  Head := nil;  Tail := nil; |

1. **Измените алгоритмы для работы с алфавитно-частотным словарем, используя двусвязные списки.**
2. **Выведите список слов из словаря в порядке убывания частоты, то есть, сначала те слова, которые встречаются чаще всего.**
3. Что такое стек? Приведите пример.

Стек — динамическая структура данных, в которой добавление и удаление элементов доступно только с одного конца (с верхнего (последнего) элемента).

Существуют такие сокращения:

* LIFO = Last In -> First Out — с английского языка «Кто последним вошел, тот первым вышел»
* FILO = First In -> Last Out — «первым вошел, последним вышел»  
  const MAXSIZE = 50;

type Stack = record *{ стек рассчитан на 50 символов }*

tags: array[1..MAXSIZE] of char;

size: integer; *{ число элементов }*

end;

1. Поясните основной принцип стека.  
   Стек — это структура данных, работающая по принципу LIFO (Last In, First Out), что переводится как "последним пришел — первым ушел". Это значит, что элементы добавляются и удаляются из одной стороны стека, причем последним добавленным элементом будет тот, который будет удалён первым.
2. Опишите основные операции со стеком. Опишите используемые подпрограммы.

Над стеком выполняются следующие операции:

* добавление в стек нового элемента;

add(<нач\_стека>,<новый\_элемент>):<нач\_стека>

* определение пуст ли стек;

empty(<нач\_стека>):boolean

* доступ к последнему включенному элементу, вершине стека;

take(<нач\_стека>):<тип\_элементов\_стека>

* исключение из стека последнего включенного элемента.

del(<нач\_стека>):<нач\_стека>

Создание структуры узла:

|  |
| --- |
| type PNode = ^Node; *{ указатель на узел }*  Node = record  data: char; *{ полезная }*  next: PNode; *{ указатель на след. элемент }*  end; |

Добавление элемента в стек:

|  |
| --- |
| procedure Push( var Head: PNode; x: char);  var NewNode: PNode;  begin  New(NewNode); *{ выделение памяти }*  NewNode^.data := x; *{ запись символа }*  NewNode^.next := Head; *{ сделать первым узлом }*  Head := NewNode;  end; |

Забор элемента с вершины:

|  |
| --- |
| function Pop ( var Head: PNode ): char;  var q: PNode;  begin  if Head = nil then begin *{ если стек пустой }*  Result := char(255); *{ неиспользуемый символ, т.е. ошибка }*  Exit;  end;  Result := Head^.data; *{ берем верхний символ }*  q := Head; *{ запоминаем вершину }*  Head := Head^.next; *{ удаляем вершину }*  Dispose(q); *{ удаление из памяти }*  end; |

Проверка, пустой ли стек:

|  |
| --- |
| function isEmptyStack ( S: Stack ): Boolean;  begin  Result := (S = nil);  end; |

Объявления в основной программе:

|  |
| --- |
| var S: PNode;  ...  S := nil; |

1. Что такое очередь? Приведите пример.

**Очередь** — динамическая структура данных, у которой в каждый момент времени доступны только два элемента: первый и последний. Добавление элементов возможно только с одного конца (конца очереди), а удаление элементов – только с другого конца (начала очереди).

type Queue = record

data: array[1..MAXSIZE] of integer;

head, tail: integer;

end;

1. Поясните основной принцип очереди.

Очередь — это структура данных, которая работает по принципу FIFO (First In, First Out), что переводится как "первым пришёл — первым ушёл". Это значит, что элементы добавляются в очередь с одного конца (называется "хвост"), а извлекаются с другого конца (называется "голова").

1. Опишите основные операции с очередями.

Как добавить в очередь:

|  |
| --- |
| procedure PushTail( var Q: Queue; x: integer);  begin  if Q.head = (Q.tail+1) mod MAXSIZE + 1  then Exit; *{ очередь уже полна }*  Q.tail := Q.tail mod MAXSIZE + 1;  Q.data[Q.tail] := x;  end; |

Как выбрать из очереди:

|  |
| --- |
| function Pop ( var S: Queue ): integer;  begin  if Q.head = Q.tail mod MAXSIZE + 1 then begin  Result := MaxInt;  Exit;  end;  Result := Q.data[Q.head];  Q.head := Q.head mod MAXSIZE + 1;  end; |

1. Что такое дек? Приведите пример.

**Дек** — англ. double ended queue, т.е. очередь с двумя концами – это динамическая структура данных, добавлять и удалять элементы в которой можно с обоих концов.

type

TDequeItem = Integer; // Можно заменить на любой нужный тип

PDequeNode = ^TDequeNode;

TDequeNode = record

Data: TDequeItem;

Next, Prev: PDequeNode;

end;

var

Head, Tail: PDequeNode;

begin

InitDeque(Head, Tail);

1. Опишите подпрограммы из задания с деком.

Для очереди доступны **следующие операции:**

* добавить элемент в конец очереди (PushTail);
* удалить элемент с начала очереди (Pop).

Как добавить в очередь:

|  |
| --- |
| **procedure** PushTail( **var** Q: Queue; x: **integer**);  **begin**  **if** Q.head = (Q.tail+1) **mod** MAXSIZE + 1  **then** Exit; *{ очередь уже полна }*  Q.tail := Q.tail **mod** MAXSIZE + 1;  Q.data[Q.tail] := x;  **end**; |

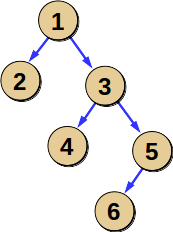
Как выбрать из очереди:

|  |
| --- |
| **function** Pop ( **var** S: Queue ): **integer**;  **begin**  **if** Q.head = Q.tail **mod** MAXSIZE + 1 **then** **begin**  Result := MaxInt;  Exit;  **end**;  Result := Q.data[Q.head];  Q.head := Q.head **mod** MAXSIZE + 1;  **end**; |

1. Что такое дерево? Перечислите основные элементы дерева.

Дерево – это структура данных, которая состоит из узлов и соединяющих их направленных ребер или дуг; в каждый узел за исключением корневого ведет только одна дуга.

* Корнем называется главный или начальный узел дерева.
* Листом называется узел, из которого не выходят дуги.

[](https://labs-org.ru/wp-content/uploads/2017/08/1-14.png)

* Перечислим некоторые иерархические отношения в дереве:

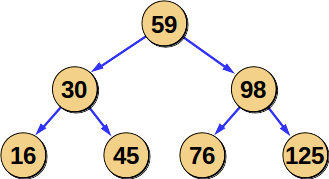
1 — предок для всех других узлов в дереве  
6 — потомок для узлов 5, 3 и 1  
3 — родитель узлов 4 и 5  
5 — сын узла 3  
5 — брат узла 4

* Высота дерева определяется как наибольшее расстояние от корня до листа (количество дуг).

1. Что такое двоичное дерево?

В двоичном (бинарном) дереве каждый узел имеет не более двух дочерних узлов (сыновей).

|  |
| --- |
| type PNode = ^Node; *{ указатель на узел }*  Node = record  data: integer; *{ данные }*  left, right: PNode;  end; |

1. Опишите алгоритм поиска по дереву.  
   При поиске в дереве используется понятие ключ. Ключ — это характеристика узла, по которой осуществляется поиск.  
   В левую сторону отходят узлы с меньшими ключами, а в правую — с большими.  
   [](https://labs-org.ru/wp-content/uploads/2017/08/2-6.png)

Пример: Найти в дереве ключ, равный k.

Алгоритм решения:

* если дерево пустое, значит, ключ не найден;
* если ключ узла равен k, то остановка;
* если ключ узла меньше k, то искать k в левом поддереве;
* если ключ узла больше k, то искать k в правом поддереве.

1. Опишите алгоритм поиска в бинарном дереве.

Создадим функцию для поиска. На вход функции из главной программы подаются два параметра: tree — адрес корня дерева и x — искомое число. Функция возвращает адрес узла с искомым значением или nil, если ничего не найдено.

|  |
| --- |
| function SearchInTree(Tree: PNode; x: integer): PNode;  begin  if Tree = nil then begin  Result := nil;  Exit;  end;  if x = Tree^.data then  Result := Tree  else  if x < Tree^.data then  Result := SearchInTree(Tree^.left, x)  else Result := SearchInTree(Tree^.right, x);  end; |

1. Улыбнитесь😊