**Практическая работа 6**

**Типы данных**

**Ответьте на вопросы:**

1. Что такое запись? Из чего она состоит. Приведите свой пример.
2. Как объявляются и создаются записи в Паскале? Как обращаться к полям записи? Приведите примеры.
3. Зачем используются записи в виде двумерной таблицы? Приведите пример использования.
4. Как использовать конструкции with при работе с записями?
5. Как происходит считывание/запись записей в файл?
6. Что такое множество в Паскале? Как оно задается?
7. Какие действия можно выполнять с множествами? Приведите примеры.
8. Для чего используется операция IN при работе с множествами? Приведите примеры.
9. Опишите динамические данные, в чем их особенность?
10. Что такое указатель? Как он объявляется? Как присваиваются значения? Приведите примеры.
11. Что такое список? Какие виды списков бывают?
12. Что такое односвязный список? Как он выглядит? Как происходит объявление типа данных?
13. Опишите алгоритм работы с алфавитно-частотным словарем. Опишите используемые подпрограммы.
14. Что такое двусвязные списки? В чем особенность работы с двусвязными списками?
15. **Измените алгоритмы для работы с алфавитно-частотным словарем, используя двусвязные списки.**
16. **Выведите список слов из словаря в порядке убывания частоты, то есть, сначала те слова, которые встречаются чаще всего.**
17. Что такое стек? Приведите пример.
18. Поясните основной принцип стека.
19. Опишите основные операции со стеком. Опишите используемые подпрограммы.
20. Что такое очередь? Приведите пример.
21. Что такое стек? Приведите пример.
22. Поясните основной принцип очереди.
23. Опишите основные операции с очередями.
24. Что такое дек? Приведите пример.
25. Опишите подпрограммы из задания с деком.
26. Что такое дерево? Перечислите основные элементы дерева.
27. Что такое двоичное дерево?
28. Опишите алгоритм поиска по дереву.
29. Опишите алгоритм поиска в бинарном дереве.
30. Улыбнитесь😊
31. **Запись** — структура данных, объединяющая переменные разных типов под одним именем. Состоит из **полей** (имя и тип).

type

TStudent = record

Name: string;

Age: integer;

end;

1. Объявление переменной:

type

TStudent = record

Name: string;

Age: integer;

end;

Создание и обращение:

var

Student: TStudent;

begin

Student.Name := 'Иван';

Student.Age := 20;

end;

1. Для хранения структурированных данных:

type

TEmployee = record

ID: integer;

Name: string;

end;

var

Employees: array[1..100] of TEmployee;

1. Упрощает доступ к полям записи:

with Student do

begin

Name := 'Мария';

Age := 22;

end;

1. Используются Read и Write
2. Множество — набор уникальных элементов одного типа. Задается через set of:

type

TCharSet = set of char;

var

Letters: TCharSet;

begin

Letters := ['A', 'B'];

end;

1. Объединение, пересечение, разность, проверка на вхождение:

var

A, B, C: TCharSet;

begin

A := ['A', 'B'];

B := ['B', 'C'];

C := A + B; // C = ['A', 'B', 'C']

end;

1. Операция in используется для проверки, принадлежит ли элемент множеству:

if 'A' in Letters then

WriteLn('A входит в множество');

1. Динамические данные — это данные, память для которых выделяется во время выполнения программы. Их особенность в том, что размер таких данных может изменяться в процессе работы программы.
2. Указатель — это переменная, которая хранит адрес другой переменной:

type

PInteger = ^integer;

var

P: PInteger;

begin

New(P);

P^ := 10;

end;

1. Список — это структура данных, состоящая из узлов, каждый из которых содержит данные и ссылку на следующий узел. Виды списков: односвязные, двусвязные, кольцевые.
2. Односвязный список — узел содержит данные и ссылку на следующий узел.

type

PNode = ^TNode;

TNode = record

Data: integer;

Next: PNode;

end;

1. Создание списка слов, подсчет частоты, сортировка по частоте: AddWord, CountFrequency, SortByFrequency
2. Двусвязный список — узел содержит данные, ссылку на следующий и предыдущий узлы. Можно перемещаться в обе стороны.
3. type

PNode = ^TNode;

TNode = record

Data: string;

Frequency: integer;

Next, Prev: PNode;

end;

1. Для вывода списка в порядке убывания частоты нужно отсортировать список по полю Frequency и вывести его.
2. Для вывода списка в порядке убывания частоты нужно отсортировать список по полю Frequency и вывести его.  
   var

Stack: array[1..100] of integer;

Top: integer;

1. Последний добавленный элемент извлекается первым.
2. 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>

**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**;

1. Очередь — это структура данных, работающая по принципу FIFO (First In, First Out).
2. Уже было такое 
3. Основной принцип очереди: первый добавленный элемент извлекается первым.
4. добавить элемент в конец очереди (PushTail)

удалить элемент с начала очереди (Pop)

1. **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.

var

Deque: array[1..100] of integer;

Front, Rear: integer;

1. AddFront — добавление в начало.

AddRear — добавление в конец.

RemoveFront — извлечение из начала.

RemoveRear — извлечение из конца.

1. 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. Элементы: узел, корень, лист, ключ, высота узла, глубина узла
2. В двоичном (бинарном) дереве каждый узел имеет не более двух дочерних узлов (сыновей).
3. Алгоритм поиска по дереву:
   * Начните с корня
   * Если значение меньше текущего узла, идите влево, иначе вправо
   * Повторяйте до нахождения или достижения листа
4. Алгоритм поиска в бинарном дереве аналогичен поиску по дереву, но с учетом того, что дерево двоичное

function Search(Node: PTreeNode; Value: integer): PTreeNode;

begin

if (Node = nil) or (Node^.Data = Value) then

Search := Node

else if Value < Node^.Data then

Search := Search(Node^.Left, Value)

else

Search := Search(Node^.Right, Value);

end;