Ответы на вопросы:

1. **Запись** — это структура, состоящая из наборов ключей и значений, ключ уникален в одной запись, значения могут быть не уникальны. Пример Запись о ученике Возраст пол,группа обучения
2. Объявление записи происходит с помощью type, а переменная типа записи объявляется как обычная переменная.
3. Записи используются в виде двумерной таблицы для удобного хранения и обработки структурированных данных, например, базы данных студентов или сотрудников.
4. Оператор with упрощает доступ к полям записи, позволяя обращаться к ним без указания имени переменной.
5. Assign,rewrite,write,close
6. **Множество** – это структурированный тип данных, представляющий собой набор значений одного типа.
7. Сложение,вычитание,уникализация
8. Оператор IN проверяет принадлежность элемента множеству.
9. **Динамические данные** – это переменные, память для которых выделяется во время выполнения программы с помощью new и освобождается через dispose.
10. **Указатель** – это переменная, содержащая адрес другой переменной. В паскале обозначается как ^
11. **Список** – это динамическая структура данных, состоящая из элементов, связанных указателями. Списки бывают односвязными,двусвязными,кольцевыми
12. **Односвязный список** – это структура, где каждый элемент хранит указатель на следующий.
13. **Алгоритм работы**

**Инициализация списка**

* Указатели head и tail инициализируются как nil, обозначая пустой список.
* Переменная wordCount (счетчик слов) устанавливается в 0.

**Чтение слов**

* Пользователь вводит слова по одному.
* Если введено слово "стоп", ввод прекращается.

**Добавление слова в список (AddWord)**

* Поиск слова в списке:Если слово уже есть в списке, увеличивается его счетчик (count).
* Если слова нет, создается новый узел (TWordNode).
* Вставка нового узла в список:
* Если список пуст, новый узел становится head и tail.
* Если слово меньше первого слова в списке, новый узел становится новым head.
* В противном случае поиск идет по списку, и слово вставляется в нужное место.

**Вывод результатов**

* Выводится количество уникальных слов.

**Освобождение памяти**

* Все узлы списка последовательно удаляются.

**Используемые подпрограммы**

**procedure AddWord(word: string);**

* Добавляет слово в связный список в алфавитном порядке.
* Увеличивает счетчик, если слово уже есть.

1. **Двусвязные списки** — это тип динамической структуры данных, в которой каждый элемент (узел) содержит не только ссылку на следующий элемент, но и на предыдущий. Это позволяет легко перемещаться как вперед, так и назад по списку.

**Особенности работы с двусвязными списками**

**Доступ к элементам**

* Можно перемещаться по списку в обоих направлениях, используя ссылки Next и Prev.

**Вставка и удаление**

* Упрощается удаление узла, так как можно легко обновить ссылки соседних элементов.
* Вставка элемента также эффективна, особенно если известна его позиция.

**Использование памяти**

* Требует больше памяти, чем односвязный список, так как каждый узел хранит две ссылки.

1. **Стек** — это структура данных, работающая по принципу **LIFO** (Last In, First Out – «последний вошел, первый вышел»). Это означает, что последний добавленный элемент удаляется первым. Стек можно представить как стопку тарелок: последняя положенная тарелка будет убрана первой.
2. Основной принцип стека – **LIFO** (Last In, First Out). Данные добавляются (Push) сверху, а удаляются (Pop) также сверху.
3. Основные операции со стеком

* Push (добавление) – помещает элемент на вершину стека.
* Pop (удаление) – удаляет и возвращает верхний элемент.Peek (просмотр верхнего элемента) – возвращает верхний элемент без удаления.
* IsEmpty (проверка на пустоту) – проверяет, пуст ли стек.
* IsFull (проверка на заполненность) – проверяет, полон ли стек.

1. Очередь – это структура данных, работающая по принципу **FIFO** (First In, First Out – «первый пришел, первый вышел»). Очередь можно представить как очередь в магазине: первый пришедший обслуживается первым.
2. Принцип **FIFO** (First In, First Out): элементы добавляются в конец
3. Основные операции с очередями.

* (Enqueue) и удаляются из начала (Dequeue).
* Enqueue (добавление в конец очереди)
* Dequeue (удаление из начала очереди)
* Peek (просмотр первого элемента)
* IsEmpty (проверка на пустоту)
* IsFull (проверка на заполненность)

1. Дек (двусторонняя очередь) – это структура, в которой элементы можно добавлять и удалять как с начала, так и с конца. Пример использования: в браузерах история страниц реализуется с помощью дека.
2. Описание подпрограмм из 17 Лабораторной

**PushFront(var head, tail: PNode; value: integer)** – добавляет элемент в начало дека.

* Создает новый узел.
* Устанавливает next на старый head.Если head существовал, обновляет его prev.
* Если дек был пуст, обновляет tail.

**PushBack(var head, tail: PNode; value: integer)** – добавляет элемент в конец дека.

* Создает новый узел.
* Устанавливает prev на старый tail.
* Если tail существовал, обновляет его next.
* Если дек был пуст, обновляет head.

**PopFront(var head, tail: PNode): integer** – удаляет и возвращает первый элемент.

* Проверяет, пуст ли дек.
* Сохраняет данные и передвигает head на следующий узел.
* Обновляет prev нового head или tail (если дек пуст).
* Освобождает память.

**PopBack(var head, tail: PNode): integer** – удаляет и возвращает последний элемент.

* Проверяет, пуст ли дек.
* Сохраняет данные и передвигает tail на предыдущий узел.
* Обновляет next нового tail или head (если дек пуст).
* Освобождает память.

1. Дерево – это иерархическая структура данных, состоящая из узлов. Основные элементы: Корень (Root)

* Вершина (Node)
* Ребро (Edge)
* Лист (Leaf)
* Дочерние узлы (Children)
* Родитель (Parent)

1. Двоичное дерево – это дерево, в котором каждый узел имеет не более двух потомков (левый и правый).
2. Алгоритм поиска по дереву

**Обход в глубину (DFS)**:

* Прямой (Preorder): Корень → Левый → Правый
* Центрированный (Inorder): Левый → Корень → Правый
* Обратный (Postorder): Левый → Правый → Корень

**Обход в ширину (BFS)**:

* посещает узлы уровня за уровнем.

1. Алгоритм поиска элемента в бинарном дереве:

* Начать с корня. Если ключ искомого узла меньше текущего, двигаться влево.
* Если ключ больше, двигаться вправо.
* Если ключ найден – завершить поиск.
* Если узел не найден – вернуть "не найдено".