СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ В ЭВМ

Практические задания

Раздел 1. Основные структуры данных

Тема 2. Структуры данных “стек” и “очередь”

Задание 1. Реализовать программу, выполняющую стандартный набор операций со стеком на основе массива:

- проверку пустоты стека
- проверку заполненности стекового массива
- добавление элемента в вершину стека
- удаление элемента из вершины стека
- вывод текущего состояния стека на экран

Требования:

- все действия должны быть оформлены как процедуры или функции
- добавлению/удалению должна предшествовать проверка возможности выполнения этих операций
- главная программа реализует следующий набор действий:
  - инициализация пустого стека
  - организация диалогового цикла с пользователем

Задание 2. Реализовать тот же набор действий на основе динамического распределения памяти.

Требования аналогичны заданию 1, за исключением того, что проверку заполненности стека проводить не надо. Пустой стек задается установкой sp := **nil**.

Задание 3. Добавить в предыдущую программу возможность занесения в стек сразу нескольких значений. Количество вводимых значений должно запрашиваться у пользователя, а сами значения можно формировать случайным образом с помощью функции **Random** (не забыть предварительно вызвать функцию **Randomize**). Проверить работоспособность программы при различных количествах вводимых элементов, в том числе – для больших значений (десятки тысяч элементов).

Задание 4 (дополнительно). Добавить в предыдущую программу следующие возможности:

- при удалении элемента из основного стека запросить у пользователя, что делать далее с этим элементом: действительно удалить с освобождением памяти или включить его в вершину вспомогательного стека удаленных элементов
- при добавлении нового элемента запросить у пользователя происхождение этого элемента: действительно создание нового элемента или выбор его с вершины вспомогательного стека
- вывод содержимого вспомогательного стека удаленных элементов

Задание 5. Реализовать программу, выполняющую стандартный набор операций с кольцевой очередью на основе массива:

- проверку пустоты очереди
- проверку заполненности очереди
- добавление элемента в конец очереди
- удаление элемента из начала очереди
- вывод текущего состояния очереди на экран

Требования к программе:

- все действия должны быть оформлены как процедуры или функции
- добавлению/удалению должна предшествовать проверка возможности выполнения этих операций
- главная программа реализует следующий набор действий:
  - инициализация пустой очереди
  - организация диалогового цикла с пользователем

Задание 6. Реализовать тот же набор действий на основе динамического распределения памяти.

Требования аналогичны заданию 1, за исключением того, что проверку заполненности очереди проводить не надо. Пустая очередь содержит только заголовочный элемент

Задание 7. Написать программу для моделирования работы очереди со случайным числом добавляемых и удаляемых элементов.

Пусть за единицу времени (например – 1 минуту) в очередь либо добавляется, либо удаляется случайное число элементов в количестве от 1 до 3-х. Одновременно добавление и удаление НЕ происходит. Для наглядности элементами пусть являются большие латинские буквы (коды ASCII от 65 до 90), выбираемые случайным образом. Тип операции с очередью (добавление или удаление) также задается случайно. Это приводит к тому, что очередь случайно растет и убывает. Программа должна наглядно показывать состояние очереди в каждый момент времени.

Рекомендации по разработке программы.

За основу взять предыдущую программу, внеся в ее процедуры следующие изменения:

- процедура добавления вместо запроса у пользователя информационной части нового элемента должна получать ее как входной параметр
- процедура удаления должна выполнять удаление одного элемента только если очередь НЕ пустая

Главная программа должна:

- после создания пустой очереди, содержащей только заголовок, инициировать датчик псевдослучайных чисел (процедура **Randomize**) и вывести соответствующее сообщение
- организовать цикл с выходом при вводе пользователем какого-либо специального символа (например – буквы q)
- сгенерировать случайное число в диапазоне от 1 до 100 и проверить его четность с помощью операции взятия остатка от деления этого числа на 2
- если число четное – реализовать операцию добавления, если нечетное – операцию удаления
- в том и другом случае выполнить:
    - генерацию случайного числа добавляемых или удаляемых символов в диапазоне от 1 до 3-х
    - вывод сообщения о выполняемом действии и количестве добавляемых/удаляемых элементов

    - выполнение цикла для добавления или удаления заданного числа элементов с вызовом соответствующих процедур работы с очередью, причем при добавлении новый символ должен генерироваться случайно с помощью его кода (диапазон 65 – 90) с последующим преобразованием кода в сам символ с помощью стандартной функции **CHR**
- вывести текущее состояние очереди
- запросить у пользователя символ для окончания цикла (при выполнении программы для продолжения работы цикла можно просто дважды нажать клавишу ввода)

После отладки программы надо **выполнить ее несколько раз** для следующих ситуаций:

- случайное число добавляемых и удаляемых элементов одинаково (например – от 1 до 3-х)
- число добавляемых элементов чуть больше (в среднем!) числа удаляемых (например, добавляется случайное количество элементов в диапазоне от 1 до 4-х, а удаляется – от 1 до 3-х)
- число удаляемых элементов чуть больше числа добавляемых

Для каждого случая выполнить программу при достаточно большом числе добавлений/удалений (30-40) и сделать вывод о поведении очереди.

Тема 3. Основы реализации списковых структур

Задание 1. Реализовать программу для простейшего моделирования линейного списка с помощью массива. Должны быть реализованы все основные действия:

- проход по списку с выводом на экран информационных частей элементов
- поиск элемента с заданной информационной частью
- добавление нового элемента после заданного и перед заданным со сдвигом (при необходимости) хвостовой части вправо для освобождения ячейки массива

- удаление заданного элемента со сдвигом (при необходимости) хвостовой части влево для заполнения образовавшейся пустой ячейки.

Выполнение всех операций предусматривает необходимые проверки (наличие в списке хотя бы одного элемента, наличие свободных ячеек, наличие искомого элемента). Все основные операции оформляются как подпрограммы с параметрами. Главная программа создает пустой список, устанавливая счетчик числа элементов в списке в ноль, и организует диалог для реализации всех операций.

Проверить работу программы для небольшого массива (до 10 элементов).

Задание 2. Изменить предыдущую программу для создания упорядоченного списка. Для этого надо изменить логику работы подпрограммы добавления элемента. Подпрограмма должна находить соответствующее место для нового элемента, т.е. поиск должен останавливаться при обнаружении первого элемента, большего заданного. Предусмотреть обработку особых случаев:

- если в списке нет ни одного элемента, вставка производится в первую ячейку массива
- если все элементы меньше заданного, вставка производится в конец списка

Проверить работу программы для двух случаев: список целых чисел по возрастанию и список коротких текстовых строк по алфавиту.

Задание 3. Реализовать линейный список на базе массива с указателями-индексами. Список должен иметь заголовок (нулевая ячейка массива) с номером первого элемента списка. Набор операций - стандартный. Для отслеживания свободных ячеек использовать простейшую схему – отмечать свободные ячейки специальным значением индекса (-1). Главная программа при создании пустого списка должна отметить все ячейки массива (кроме нулевой) как свободные.

Задание 4. Реализовать линейный динамический список со стандартным набором операций. Пустой список содержит только элемент-заголовок, который создается главной программой в начале работы и содержит значение **nil** в ссылочной части. У непустого списка в ссылочной части хранится адрес первого реального элемента. Адрес заголовка сохраняется в глобальной ссылочной переменной. Все действия оформляются как подпрограммы.

Подпрограмма для добавления элемента **после** заданного должна работать и для пустого списка – в этом случае новый (он же первый реальный!) элемент должен добавляться сразу после заголовка. Для этого проверить пустоту списка, после чего для пустого списка установить глобальный указатель текущего элемента в адрес заголовка, для непустого списка вызвать процедуру поиска. Само добавление выполняется обычным образом.

Задание 5. Изменить предыдущую программу так, чтобы удаляемый из основного списка элемент добавлялся во вспомогательный список с возможностью просмотра вспомогательного списка. Работа со вспомогательным списком может выполняться по стековому принципу.

В предыдущую программу надо внести следующие изменения:

- добавить глобальную ссылочную переменную для хранения адреса вершины стека
- в начале главной программы создать пустой вспомогательный список (стек)
- в основное меню добавить возможность просмотра вспомогательного списка (стека)
- изменить процедуру удаления элемента из основного списка, заменив операцию освобождения памяти операциями добавления удаленного элемента во вспомогательный список (стек)
- добавить обычную процедуру вывода вспомогательного списка (стека)

Задание 6. Изменить предыдущую программу для поддержки упорядоченных списков (см. задание 2).

Тема 4. Усложненные списковые структуры

Задание 1. Реализовать линейный динамический двунаправленный список со следующим набором операций:

- просмотр списка в прямом и обратном направлениях
- поиск заданного элемента в прямом и обратном направлениях
- добавление элемента перед или после заданного
- удаление заданного элемента

Список должен иметь заголовок и быть кольцевым. Пустой список содержит только заголовок, оба ссылочных поля которого указывают на сам заголовок. Адрес заголовка задается глобальной ссылочной переменной. Все операции оформляются как подпрограммы. Добавление нового элемента после заданного должно работать и для пустого списка (см. задание 4 из предыдущей темы).

Задание 2. Реализовать набор подпрограмм для выполнения основных операций с массивом списков. Каждый элемент массива хранит только указатель на начало связанного списка. Сам базовый массив работает на основе сдвиговых операций. Основные операции:

- полный проход по всей структуре
- поиск заданного элемента
- добавление нового элемента в массив с пустым связанным списком
- добавление нового элемента в связанный список
- удаление элемента из связанного списка
- удаление элемента из базового массива

Задание 3. Реализовать набор подпрограмм для выполнения основных операций со списком списков. Требования аналогичны предыдущему заданию.

Тема 5. Основные понятия о древовидных структурах

Задание 1. Построение и обход идеально сбалансированных двоичных деревьев. Реализовать программу, выполняющую следующий набор операций:

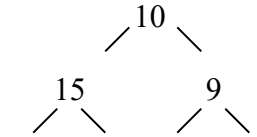
- построение идеально сбалансированного двоичного дерева с заданным числом вершин

- построчный вывод дерева на основе процедуры обхода в прямом порядке
- построчный вывод дерева на основе процедуры обхода в симметричном порядке
- построчный вывод дерева на основе процедуры обхода в обратно-симметричном порядке

Рекомендации:

- для простоты построения дерева можно информационную часть формировать как случайное целое число в интервале от 0 до 99
- глобальные переменные: указатель на корень дерева и число вершин
- алгоритмы построения ИСД и его обхода оформляются как подпрограммы, вызываемые из главной программы
- все процедуры обхода должны выводить вершины с числом отступов, пропорциональным уровню вершины: корень дерева не имеет отступов, вершины первого уровня выводятся на 5 отступов правее, вершины 2-го уровня – еще на 5 отступов правее и т.д. Для этого в рекурсивные подпрограммы обхода надо ввести второй формальный параметр - уровень этой вершины
- Все процедуры обхода имеют похожую структуру. Например, процедура обхода в прямом направлении должна:
  - проверить пустоту очередного поддерева
  - вывести в цикле необходимое число пробелов в соответствии с уровнем вершины
  - вывести информационную часть текущей вершины
  - вызвать рекурсивно саму себя для обработки своего левого поддерева с увеличением уровня на 1
  - вызвать рекурсивно саму себя для обработки своего правого поддерева с увеличением уровня на 1

Сравнение рассмотренных правил вывода двоичного дерева приводится в следующей таблице

Исходное дерево	Вывод в прямом порядке	Вывод в симметричном порядке	Вывод в обратно-симметричном порядке
	10 15 13 22 9 5 17	13 15 22 10 5 9 17	17 9 5 10 22 15 13

Главная программа реализует следующий набор действий:

- запрос числа вершин в дереве
- запуск рекурсивной подпрограммы построения идеально сбалансированного дерева со следующими фактическими параметрами: указатель на корень дерева (при построении дерева этот параметр является выходным!) и заданное число вершин
- последовательный вызов подпрограмм обхода дерева со следующими фактическими входными параметрами: указатель на корень дерева, ноль в качестве уровня корневой вершины дерева.

Задание 2. Добавить в программу **нерекурсивный** вариант процедуры обхода дерева в симметричном порядке.

Замена рекурсии циклом основана на использовании вспомогательного стека для хранения последовательности пройденных вершин от корня до текущей вершины и уровня этих вершин в дереве (напомним, что уровень используется только для задания правильного числа отступов при построчном выводе дерева). Исходя из этого, создаваемая подпрограмма должна объявить необходимые локальные типы данных для динамической реализации вспомогательного стека. Каждый элемент стека должен хранить: указатель на пройденную вершину дерева, уровень вершины, указатель на следующий элемент стека. Для реализации стека, как обычно, требуются две ссылочные переменные: указатель на вершину стека и вспомогательный указатель, используемый при добавлении или удалении элементов в стек.

Кодовая часть подпрограммы должна начинаться с инициализации необходимых переменных: текущая вершина дерева – его корень, вспомогательный стек - пустой, начальный уровень равен (-1). После этого запускается основной цикл обработки дерева, включающий выполнение следующих действий:

- циклическое сохранение очередной вершины в стеке (пока не будет достигнута пустая вершина) следующим образом:
  - увеличение уровня вершины на 1
  - создание нового элемента стека, заполнение всех его полей и добавление его в стек
  - переход к левому потомку текущей вершины
- проверка пустоты стека: если стек пуст, то основной цикл следует завершить, а иначе – перейти к обработке вершины
  - извлечь из стека адрес текущей обрабатываемой вершины и ее уровень
  - вывести вершину с необходимым числом пробелов
  - удалить элемент из стека с освобождением памяти
  - перейти к правому потомку текущей вершины

Для проверки правильности работы подпрограммы сравнить ее результаты с рекурсивным аналогом.

Задание 3. Обработка произвольных двоичных деревьев

Реализовать программу, выполняющую следующий набор операций с двоичными деревьями:

- поиск вершины с заданным значением информационной части
- добавление левого или правого потомка для заданной вершины
- построчный вывод дерева с помощью основных правил обхода
- уничтожение всего дерева

Рекомендации:

- объявить необходимые глобальные переменные: указатель на корень дерева, указатель на родительскую вершину, признак успешности поиска

- объявить и реализовать рекурсивную подпрограмму поиска. В качестве основы можно взять любой из известных вариантов обхода дерева. Основное отличие рекурсивного поиска от рекурсивного обхода состоит в необходимости прекращения обхода дерева в случае совпадения информационной части текущей вершины с заданным значением. Один из способов прекращения обхода основан на использовании булевского признака, который перед запуском обхода устанавливается в **false** и переключается в **true** при обнаружении искомой вершины. Для каждой текущей вершины подпрограмма поиска должна выполнять следующие основные действия:
  - проверить необходимость продолжения поиска
  - проверить текущую ссылочную переменную на **nil**
  - сравнить информационную часть текущей вершины с заданным значением
  - если эти величины совпадают, то установить признак окончания поиска и установить адрес родительской переменной в адрес текущей вершины
  - в противном случае продолжить поиск сначала в левом поддереве текущей вершины, вызвав рекурсивно саму себя с адресом левого потомка, а потом – в правом поддереве, вызвав саму себя с адресом правого потомка

  Результат поиска можно проверить с помощью глобального признака. В случае успешного поиска становится известен адрес найденной вершины, который можно использовать в дальнейшем для добавления к этой вершине одного из потомков.
- Объявить и реализовать подпрограмму добавления новой вершины в дерево как потомка заданной вершины.

  Подпрограмма должна:
  - проверить пустоту дерева: если указатель корня имеет значение **nil**, то надо создать корень дерева

- выделить память
- запросить значение информационной части корня
- сформировать пустые ссылочные поля на потомков

➢ если дерево не пустое, то организовать поиск родительской вершины:

- запросить искомое значение информационной части родительской вершины
- установить признак поиска и вызвать подпрограмму поиска
- если поиск удачен, то проверить число потомков у найденной родительской вершины
- если вершина-родитель имеет двух потомков, то добавление невозможно
- если родительская вершина имеет только одного потомка, то сообщить о возможности добавления одного из потомков, выделить память для новой вершины и заполнить все три ее поля, настроить соответствующее ссылочное поле у родительской вершины
- если родительская вершина не имеет ни одного потомка. то запросить тип добавляемой вершины (левый или правый потомок) и выполнить само добавление

- Объявить и реализовать рекурсивную подпрограмму для построчного вывода дерева в обратно-симметричном порядке. Эту подпрограмму без каких-либо изменений можно взять из предыдущей работы.
- Объявить и реализовать подпрограмму для уничтожения всего дерева с освобождением памяти. Основой подпрограммы является рекурсивный обход дерева, причем – по правилу **обратного** обхода: сначала посещается и удаляется левый потомок текущей вершины, потом посещается и удаляется правый потомок, и лишь затем удаляется сама текущая вершина. Такой обход позволяет не разрывать связи между родительской вершиной и потомками до тех пор, пока не будут удалены оба потомка. Подпрограмма

удаления имеет один формальный параметр – адрес текущей вершины. Подпрограмма должна проверить указатель на текущую вершину, и если он не равен **nil**, то:

- вызвать рекурсивно саму себя с адресом левого поддерева
- вызвать рекурсивно саму себя с адресом правого поддерева
- вывести для контроля сообщение об удаляемой вершине
- освободить память с помощью процедуры **Dispose** и текущего указателя

Главная программа должна:

- создать пустое дерево
- организовать цикл для добавления вершины с вызовом соответствующей подпрограммы и последующим построчным выводом дерева
- предоставить возможность в любой момент вызвать подпрограмму удаления дерева с фактическим параметром, равным адресу корня дерева, что запускает механизм рекурсивного удаления всех вершин, включая и корневую; поскольку после удаления корневой вершины соответствующий указатель становится неопределенным, можно инициировать его пустым значением, что позволит повторно создать дерево с новой корневой вершиной

Тема 6. Реализация поисковых деревьев

Задание 1. Построение и обработка двоичных деревьев поиска.

Реализовать программу, выполняющую следующий набор операций с деревьями поиска:

- поиск вершины с заданным значением ключа с выводом счетчика числа появлений данного ключа
- добавление новой вершины в соответствии со значением ее ключа или увеличение счетчика числа появлений
- построчный вывод дерева в наглядном виде с помощью обратно-симметричного обхода

- вывод всех вершин в одну строку по порядку следования ключей с указанием для каждой вершины значения ее счетчика появлений
- удаление вершины с заданным значением ключа

Рекомендации:

- Объявить и реализовать подпрограмму поиска. Поиск начинается с корня дерева и в цикле для каждой вершины сравнивается ее ключ с заданным значением. При совпадении ключей поиска заканчивается с выводом значения счетчика числа появлений данного ключа. При несовпадении поиск продолжается в левом или правом поддереве текущей вершины
- Объявить и реализовать рекурсивную подпрограмму добавления новой вершины в дерево. Подпрограмма использует один параметр-переменную, определяющую адрес текущей вершины. Если при очередном вызове подпрограммы этот адрес равен **nil**, то производится добавление нового элемента с установкой всех необходимых полей. В противном случае продолжается поиск подходящего места для новой вершины за счет рекурсивного вызова подпрограммы с адресом левого или правого поддерева. При совпадении ключей надо просто увеличить значение счетчика появлений
- Объявить и реализовать не-рекурсивный вариант подпрограммы добавления новой вершины в дерево. Необходимы две ссылочные переменные – адрес текущей вершины и адрес ее родителя. Сначала в цикле ищется подходящее место за счет сравнения ключей и перехода влево или вправо. Поиск заканчивается либо по пустому значению текущего адреса, либо в случае совпадения ключей (здесь просто увеличивается счетчик числа появлений). После окончания поиска либо создается корневая вершина, либо добавляется новая вершина как левый или правый потомок родительской вершины.
- Объявить и реализовать рекурсивную подпрограмму для построчного вывода дерева в обратно-симметричном порядке. Эту подпрограмму без каких-либо изменений можно взять из предыдущей работы.

- Объявить и реализовать рекурсивную подпрограмму для вывода всех вершин в одну строку в соответствии с возрастанием их ключей. Основа – обход в симметричном порядке. Дополнительно рядом со значением ключа в скобках должно выводится значение счетчика повторений данного ключа. Например:

  5(1) 11(2) 19(5) 33(1) 34(4) . . . . .

- Объявить и реализовать подпрограмму удаления вершины: запрашивается ключ вершины, организуется ее поиск, при отсутствии вершины выводится сообщение, при нахождении вершины проверяется число ее потомков и при необходимости выполняется поиск вершины-заменителя

Главная программа должна предоставлять следующие возможности:

- создание дерева с заданным числом вершин со случайными ключами
- добавление в дерево одной вершины с заданным пользователем значением ключа
- поиск в дереве вершины с заданным ключом
- построчный вывод дерева в наглядном виде
- вывод всех вершин в порядке возрастания их ключей
- удаление вершины с заданным ключом

Задание 2. Построение таблицы символических имен с помощью дерева поиска. Постановка задачи формулируется следующим образом.

Деревья поиска широко используются компиляторами при обработке текстов программ на языках высокого уровня. Одной из важнейших задач любого компилятора является выделение во входном тексте программы всех **символических имен** (ключевых слов, пользовательских идентификаторов для обозначения типов, переменных, подпрограмм) и построение их в виде **таблицы** с запоминанием номеров строк, где эти имена встречаются. Поскольку каждое выделенное из текста имя должно отыскиваться в этой таблице, а число повторений одного и того же имени в тексте программы может быть весьма большим, важным для скорости всего процесса компиляции становится скорость поиска в таблице имен. Поэтому очень часто подобные

таблицы реализуются в виде дерева поиска. **Ключом** поиска в таких деревьях являются **текстовые имена**, которые добавляются в дерево поиска в соответствии с обычными правилами упорядочивания по алфавиту.

Необходимо реализовать программу, выполняющую следующие действия:

- запрос имени исходного файла с текстом анализируемой программы
- чтение очередной строки и выделение из нее всех символических имен (будем считать, что имя – любая непрерывная последовательность букв и цифр, начинающаяся с буквы и не превышающая по длине 255 символов)
- запоминание очередного имени в дереве поиска вместе с номером строки исходного текста, где это имя найдено; поскольку одно и то же имя в тексте может встречаться многократно в разных строках, приходится с каждым именем-вершиной связывать вспомогательный линейный список номеров строк
- вывод построенной таблицы имен по алфавиту с помощью процедуры обхода дерева в симметричном порядке
- построчный вывод построенного дерева в наглядном представлении с помощью процедуры обхода дерева в обратно-симметричном порядке

Для упрощения программной реализации будем считать, что обрабатываемая программа удовлетворяет следующим непринципиальным условиям:

- в именах используются только строчные (малые) буквы
- отсутствуют комментарии
- отсутствуют текстовые константы, задаваемые с помощью кавычек ‘’

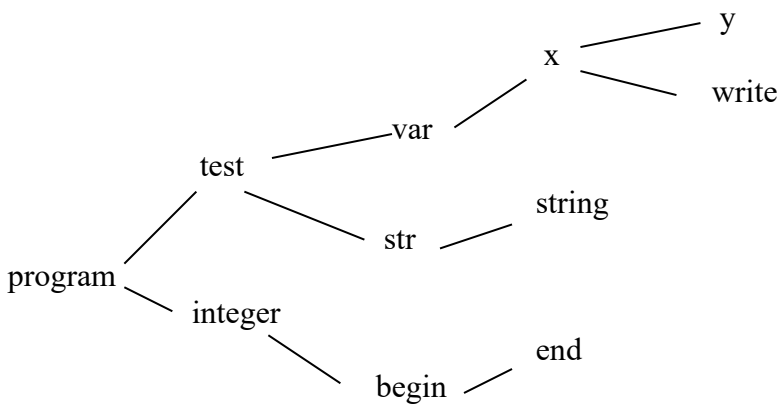
Например, пусть исходная программа имеет следующий вид:

```
program test;
var x, y : integer;
     str : string;
begin
  x := 1;
```

```
    y := x + 2;
    write(x, y);
end.
```

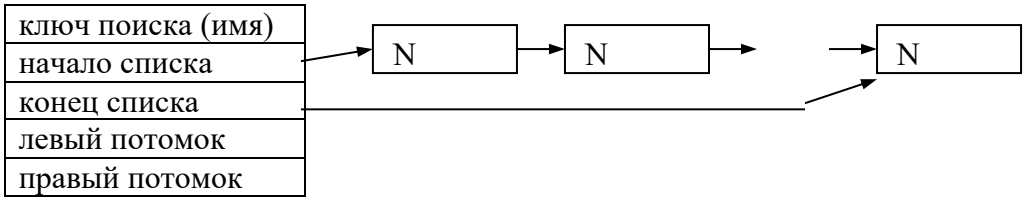
Тогда для нее должна быть построена следующая таблица имен и дерево поиска:

begin	4
end	8
integer	2
program	1
str	3
string	3
test	1
var	2
write	7
x	2 5 6 7
y	2 6 7



Рекомендации:

1. В разделе описания типов ввести два ссылочных типа данных для динамической реализации дерева поиска и вспомогательных линейных списков. Описать связанные с ними базовые типы, определяющие структуру вершин дерева и узлов списка. Вершина дерева должна содержать строковое ключевое поле, указатели на двух потомков и два указателя на начало и конец вспомогательного линейного списка.



2. Объявить необходимые глобальные переменные, такие как номер обрабатываемой строки исходного текста и ее длина, номер обрабатываемого символа в строке (счетчик символов), сама текущая строка, текущее формируемое имя, имя исходного файла, файловая переменная, указатель на корень дерева

3. Объявить и реализовать рекурсивную подпрограмму добавления вершины в дерево поиска. За основу можно взять рассмотренную в предыдущей работе процедуру добавления, внеся в нее следующие небольшие дополнения:
   - при вставке новой вершины создать первый (пока единственный!) узел вспомогательного линейного списка, заполнить его поля и поля созданной вершины дерева необходимыми значениями
   - в случае совпадения текущего имени с ключом одной из вершин дерева добавить в конец вспомогательного списка новый узел с номером строки, где найдено данное имя
4. Объявить и реализовать рекурсивную подпрограмму для вывода построенной таблицы в порядке возрастания имен. Основа – обход дерева в симметричном порядке. Дополнительно для каждой вершины выводится список номеров строк, где используется данное имя. Для этого организуется проход по вспомогательному линейному списку от его начала до конца.
5. Объявить и реализовать рекурсивную подпрограмму для построчного вывода дерева в обратно-симметричном порядке. Эту подпрограмму без каких-либо изменений можно взять из предыдущей работы.
6. Реализовать главную программу, выполняющую основную работу по выделению имен из строк обрабатываемого текста. Формируемое имя объявляется как **String**, но обрабатывается как массив символов. Основной механизм формирования имени – посимвольная обработка текущей строки. Как только в строке после не-буквенных символов распознается буква, начинается выделение всех последующих буквенно-цифровых символов и формирование очередного имени. Обрабатываемая строка (тип **String**) рассматривается как массив символов. Удобно перед обработкой строки добавить в ее конце символ пробела. Номер текущего обрабатываемого символа задается переменной-счетчиком. Для

отслеживания символов в формируемом имени также необходима переменная-счетчик.

Алгоритм работы главной программы:

- инициализация обработки исходного файла (запрос имени файла, открытие для чтения)
- цикл обработки строк исходного файла:
  - ✓ чтение очередной строки и определение ее длины
  - ✓ добавление в конец строки дополнительного символа пробела
  - ✓ инициализация счетчика символов
  - ✓ организация циклической обработки символов строки по следующему алгоритму:
    a. если очередной символ есть буква, то:
       - ➢ запомнить символ как первую букву имени
       - ➢ организовать цикл просмотра следующих символов в строке, пока они являются буквами или цифрами, с добавлением этих символов к формируемому имени
       - ➢ после окончания цикла установить длину сформированного имени (можно использовать нулевой элемент массива символов)
       - ➢ вызвать подпрограмму для поиска сформированного имени в дереве
    b. увеличить значение счетчика символов для перехода к анализу следующего символа
- вывод построенной таблицы в алфавитном порядке
- построчный вывод дерева поиска

Тема 7. Дополнительные вопросы обработки деревьев. Графы.

Задание 1. Реализовать основные операции с недвоичными деревьями, представленными с помощью списка родителей и потомков. Будем считать, что начальное дерево содержит единственную корневую вершину. Необходимо реализовать следующие операции:

- добавление новой вершины как потомка заданной вершины
- удаление заданной вершины
- вывод всех вершин с указанием родительских связей
- поиск заданной вершины

Требования к подпрограммам и главной программе – стандартные.

Задание 2. Реализовать основные операции с простыми графами с использованием матричного и спискового представлений:

- формирование матрицы смежности
- преобразование матрицы смежности в список смежности
- формирование списка смежности
- преобразование списка смежности в матрицу смежности
- добавление нового ребра
- удаление заданного ребра
- добавление новой вершины
- удаление заданной вершины
- обход графа в глубину
- обход графа в ширину

Требования к подпрограммам и главной программе – стандартные.

Раздел 2. Алгоритмы сортировки и поиска

Тема 1. Классификация методов. Простейшие методы сортировки

Задание. Реализовать программу, объединяющую простейшие методы сортировки массивов:

- сортировку обменом (метод пузырька)
- сортировку выбором
- сортировку вставками

Каждый метод реализуется своей подпрограммой, добавляемой в основную программу по мере разработки. Кроме того, необходима вспомогательная подпрограмма генерации исходного массива случайных целых чисел с заданным числом элементов (не более 10 000) и выводом этого массива на экран

Каждый исходный массив должен обрабатываться всеми подпрограммами сортировки с подсчетом и выводом фактического числа выполненных сравнений и пересылок. Поскольку каждый из универсальных методов выполняет сортировку “на месте”, т.е. изменяет исходный массив, то для наглядности работы можно передавать в подпрограмму сортировки копию исходного массива, объявив его как параметр-значение.

После завершения разработки программы необходимо выполнить всеми методами сортировку нескольких массивов с разным числом элементов (10, 100, 1.000, 10.000) и провести сравнительный анализ эффективности рассматриваемых методов.

Главная программа должна реализовать диалог с пользователем для выбора метода сортировки.

Тема 2. Улучшенные методы сортировки массивов

Задание. Добавить в ранее созданную программу с простейшими методами сортировки подпрограммы, реализующие все три улучшенных метода сортировки массивов.

Каждый метод реализуется своей подпрограммой, добавляемой в основную программу по мере разработки. Каждый исходный массив должен обрабатываться всеми программами сортировки с подсчетом и выводом фактического числа выполненных сравнений и пересылок. После завершения разработки программы необходимо выполнить всеми методами сортировку нескольких массивов с разным числом элементов (10, 100, 1.000, 10.000) и провести сравнительный анализ эффективности рассматриваемых методов.

Главная программа должна реализовать диалог с пользователем для выбора метода сортировки.

Рекомендации по реализации метода Шелла. Для хранения шагов группировки можно ввести вспомогательный массив. После генерации исходного массива надо организовать цикл, в котором запросить и ввести количество шагов и величины самих шагов. Выполнить сортировку исходного массива для нескольких **разных наборов шагов** и найти среди этих наборов наилучший по числу выполненных сравнений.

Тема 3. Специальные методы сортировки

Задание. Реализовать специальные методы сортировки:

- Простейшую карманную с использованием второго массива и без него
- Обобщенную карманную сортировку с повторяющимися ключами и дополнительными списками
- Поразрядную сортировку

Все методы реализуются как подпрограммы и поэтапно добавляются в главную программу.

Тема 4. Поиск с использованием хеш-функций

Задание 1. Построить бесконфликтную хеш-таблицу для заданного набора текстовых ключей. Число ключей (и размер таблицы) равен 10. В качестве ключей взять 10 любых служебных слов языка Паскаль. Для преобразования текстовых ключей в числовые значения использовать суммирование кодов символов текстового ключа: код (End) = код (E) + код (n) + код (d).

Преобразование числового кода ключа в значение индекса выполнить с помощью простейшей хеш-функции, которая берет остаток от целочисленного деления кода на размер хеш-таблицы (в задании – 10).

Для подсчета индексов ключей написать вспомогательную программу, которая в диалоге многократно запрашивает исходный текстовый ключ (служебное слово языка Паскаль) и подсчитывает значение хеш-функции. Программа должна вычислять длину очередного текстового ключа с помощью функции Length, в цикле вычислять сумму кодов всех символов ключа и брать остаток от деления этой суммы на 10.

Если некоторые исходные ключи будут конфликтовать друг с другом, можно изменить исходное слово, например – сменить регистр начальной буквы или всех букв в слове, полностью заменить слово на близкое по значению (End на Stop и.т.д.), ввести какие-либо спецсимволы или придумать другие способы

После подбора неконфликтующих ключей написать основную программу, которая должна:

- ввести подобранные ключи и расположить их в ячейках хеш-таблицы в соответствии со значением хеш-функции
- вывести хеш-таблицу на экран
- организовать циклический поиск разных ключей, как имеющихся в таблице (с выводом местоположения), так и отсутствующих:
  вычислить для ключа значение хеш-функции и сравнить содержимое соответствующей ячейки таблицы с исходным ключом

Задание 2. Реализовать метод внутреннего хеширования. Исходные ключи – любые слова (например – фамилии). Размер хеш-таблицы должен задаваться в программе с помощью константы m. Хеш-функция – такая же, что и в задании 1, но делить надо на константу m. В случае возникновения конфликта при попытке размещения в таблице нового ключа, для него ищется первое свободное по порядку место по формуле

$$j = ((h (\text{ключ}) + i) \bmod m) + 1, \quad \text{где } i = 0, 1, 2, \ldots, m-2$$

Программа должна выполнять следующие действия:

- добавление нового ключа в таблицу с подсчетом сделанных при этом сравнений
- поиск заданного ключа в таблице с подсчетом сделанных при этом сравнений
- вывод текущего состояния таблицы на экран

После отладки программы необходимо выполнить ее для разных соотношений числа исходных ключей и размерности таблицы: взять 10 ключей и разместить их поочередно в таблице размерности 11, 13 и 17. Для каждого случая найти суммарное число сравнений, необходимое для размещения ключей и их поиска. Сделать вывод о влиянии количества пустых мест в таблице на эффективность поиска.

Задание 3. Реализовать метод открытого хеширования. Исходные ключи – любые слова (например – фамилии). Размер хеш-таблицы должен задаваться в программе с помощью константы m. Хеш-функция – такая же, что и в задании 1, но делить надо на константу m. В случае возникновения конфликта при попытке размещения в таблице нового ключа этот ключ добавляется в конец вспомогательного списка. Это требует включения в каждую ячейку хеш-таблицы двух указателей на начало и конец вспомогательного списка.

Программа должна выполнять следующие действия:

- добавление нового ключа в таблицу с подсчетом сделанных при этом сравнений
- поиск заданного ключа в таблице с подсчетом сделанных при этом сравнений
- вывод текущего состояния таблицы на экран
- удаление заданного ключа из таблицы

Алгоритм удаления:

- вычислить хеш-функцию и организовать поиск удаляемого элемента в таблице
- если удаляемый элемент найден в ячейке таблицы, то эта ячейка либо становится пустой (если связанный с ней список пуст), либо в нее

записывается значение из первого элемента списка с соответствующим изменением указателей

- если удаляемый элемент найден в списке, то производится его удаление с изменением указателей

После отладки программы необходимо выполнить ее для разных соотношений числа исходных ключей и размерности таблицы: взять 20 ключей и разместить их поочередно в таблице размерности 9, 17 и 23. Для каждого случая найти суммарное число сравнений, необходимое для размещения ключей и их поиска. Сделать вывод о влиянии размерности таблицы на эффективность поиска.

Тема 5. Внешний поиск и внешняя сортировка

Задание 1. Разработать программу для имитации обработки простейшего Б-дерева второго порядка. Программа должна выполнять следующие действия:

- построение Б-дерева для заданного набора вершин, начиная с пустого дерева
- добавление отдельной вершины
- поиск заданной вершины
- удаление заданной вершины
- вывод Б-дерева в наглядном виде

Имитация обработки определяется тем, что файлы для хранения элементов дерева не используются.

Задание 2. Разработать программу сортировки файлов методом естественного слияния. Для простоты этап предварительной сортировки фрагментов для получения начальных серий не реализуется. Должны использоваться 5 файлов – входной и 4 вспомогательных. Исходный набор заполняется случайными числами в диапазоне от 1 до 1 миллиона.