Билет № 1

1. Организация пользовательского интерфейса в среде Turbo Pascal (понятие пользовательского интерфейса, требования к пользовательскому интерфейсу, реализация пользовательского интерфейса на основе меню).
2. Алгоритм решения задачи сортировки методом обмена (схема алгоритма, пример)
3. Составить программу, которая устанавливает по центру экрана графическое окно размером 100x100, заливает его желтым фоном и заполняет синими точками, расположенные через 4 позиции.

Билет № 2

1. Организация пользовательского интерфейса в среде Turbo Pascal (структура меню, варианты выбора функции программы с помощью меню).
2. Алгоритм построения графика функций (определение масштаба, построение осей координат, вычерчивание графика).
3. Создать вещественный массив из 10000 чисел, заполнить его случайными числами в диапазоне от 0 до 1. Вычислить среднее значение массива. Очистить динамическую память. Создать целый массив размером 10000, заполнить его случайными целыми числами в диапазоне от -100 до 100 и вычислить его среднее значение.

Билет № 3

1. Реализация вывода информации на экран в диалоговом режиме (проектирование вывода информации на экран; реализация перемещения курсора в заданную позицию экрана).

2. Структура графической Pascal-программы (модуль GRAPH.tpu; определение графического режима, видеографического устройства, инициализация графического режима).
3. Составить программу вычисления функции y= (x+1)/(5-x) при x>-5 y=1/(x+5) при x<-5. Программа должна состоять из трех частей, каждая из которых активизируется выбором соответствующего пункта меню:
   - ввод данных
   - вычисление значения функции и вывод результатов
   - завершение работы программы

Билет № 4

1. Текстовый и графический режимы работы дисплеев (характеристики режимов, минимальные единицы управления режимами).
2. Методы решения задач сортировки (внутренние, внешние). Алгоритм решения задачи сортировки выбором (схема алгоритма, пример).
3. Найти с точностью ε=0,001 на отрезке [-2;1] корень уравнения $x^3+x^2+x+1=0$ методом половинного деления.

Билет № 5

1. Численные методы решения трансцендентных (нелинейных) уравнений. Метод половинного деления (дихотомия) .
2. Управляющие графические процедуры и функции (InitGraph., CloseGraph, GraphResult, GraphErrorMsg).
3. Дан двумерный массив среднемесячных температур за 3 года. Определить, в каком году было самое теплое лето, т.е. в каком году была

наибольшая средняя температура летних месяцев. Использовать сортировку методом пузырька (сортировка обменом).

Билет № 6

1. Реализация контроля вводимых данных в среде Turbo Pascal (контроль вводимых числовых данных, кода нажатой клавиши, символьных данных, вывод сообщений пользователю о некорректном вводе данных).
2. Динамическая цепочка (определение; однонаправленные списки, двунаправленные списки; графические модели).
3. Составить программу, которая рисует на экране прямоугольники, случайным образом , заливает их разным цветом.

Билет № 7

1. Динамические и статические переменные (определение статической и динамической переменной, распределение памяти, отличие статической переменной от динамической в языке Turbo Pascal).
2. Проектирование размещения текстовой и графической информации на экране, назначение координат размещения, определение типа линий и цвета элементов изображения.
3. Дано натуральное число n. Найти сумму первой и последней цифр этого числа (используйте внешние подпрограммы).

Билет № 8

1. Разработка Pascal-программ модульной структуры (понятие модульного программирования, структура программы с модулями, компиляция модулей).

2. Графические процедуры и функции для работы с дугами, фигурами (Arc, Pieclice, Ellipse, Circle, Rectangle, Bar, Bar3d, DrawPoly, FillPoly**).**

3. Составить программу на примере односвязного списка. Список содержит записи с информационной частью: ФИО пациента , давление и ссылочной частью (адрес следующей или предыдущей записи).

Билет № 9

1. Работа с экраном в текстовом режиме (текстовые окна, функции, используемые в текстовом режиме, использование символов псевдографики).
2. Алгоритм решения задачи сортировки цифровым методом (схема алгоритма, пример).
3. Получить массив целых чисел, заполнить его случайным образом, элементы, расположенные на главной диагонали выделить, синим и красным цветом, остальные элементы зеленым.

Билет № 10

1. Алгоритмизация и программирование задач численного интегрирования. Вычисление определенного интеграла методом прямоугольников
2. Графические процедуры и функции для работы с точечными изображениями (PutPixel, GetPixel, GetX, GetY).
3. Найти с точностью $\varepsilon=0{,}001$ на отрезке [-2;1] корень уравнения $x^3+x^2+x+1=0$ методом Ньютона (метод касательных).

Билет № 11

1. Переменная ссылочного типа (назначение, объявление, использование динамических переменных, безтиповый указатель).
2. Графические процедуры и функции для работы с текстом (OutText, SetTextJustify, SetTextStyle, SetUserCharSize, InstallUserFont**).**
3. Вычислить интеграл методом средних прямоугольников. Начальное значение x1=0, конечное значение x2=π, точность вычисления e=0.01, подынтегральная функция $1-0{,}25*\sin^2 x$

Билет № 12

1. Процедуры для работы с динамическими переменными (создание динамической переменной, освобождение памяти от динамической переменной, ввод\ вывод данных полей динамической переменной).
2. Графические процедуры и функции для формирования экрана, окна, страницы (ClearDevice, SetViewPort, ClearViewPort., SetVisualPage, SetVisualPage, SetActivePage, GetMaxX GetMaxY).

3. Найти с точностью ε=0,001 на отрезке [-2;1] корень уравнения $x^3+x^2+x+1=0$ методом хорд (метод ложного положения).

Билет № 13

1. Численные методы решения трансцендентных (нелинейных) уравнений. Метод хорд (метод ложного положения).
2. Графические процедуры и функции для работы со штриховкой, цветом и палитрой (SetFillStyle, SetFillPattern, FloodFill, SetBkColor, SetColor**).**
3. Составить программу построение графика функции y=sinx для x принадлежащего отрезку [0; 2π], используя точечный метод.

Билет № 14

1. Удаление динамической переменной из динамического списка (удаление динамической переменной из начала списка, из конца списка, с заданным атрибутом).
2. Алгоритм решения задачи сортировки вставками (схема алгоритма, пример).
3. Вычислить интеграл методом трапеций . Начальное значение x1=0, конечное значение x2=10, точность вычисления e=0.01, подынтегральная функция

1/(x*Ln(x)*0,43429)

Билет № 15

1. Алгоритмы добавления данных в динамические списки (в начало динамической цепочки, в конец динамической цепочки, после заданного элемента; графические модели алгоритмов).
2. Графические процедуры и функции для работы с графическими примитивами типа "линия" (Line, LineTo, LineRel, MoveTo, MoveRel, SetLineStyle**)**.

3. . Дан список, содержащий N записей (N≤100) следующей структуры:

№ рейса	Пункт отправления	Пункт назначения	День недели	Время отправления Час Мин	Цена билета
5 символов	15 символов	15 симв.	1..7	0..23 0..59	Real

1) Ввести заданый список с экрана в массив записей Spis.

2) Ввести искомый номер рейса - Isk_nom.

3) Найти в списке рейс с заданным номером.

4) Вывести информацию о найденном рейсе на экран.

Типы данных:

```
Type

    T_Time=record
            Hour:0..23;
            Min:0..59;
               end;
    tzap=record       {Описание типа для одной записи списка}
            nom:string[5];
            p1,p2:string[15];
            day:1..7;
            time:T_Time;
            price:real;
            end;
Var         Spis:array[1..100]of tzap;    {Описание списка}
    N,i:byte;
    Isk_nom:string[5];
```

Билет № 16

1. Структуры данных: общие понятия, задачи проектирования структур данных для решения различных задач

2. Алгоритм решения задачи сортировки методом подсчета (схема алгоритма, пример).

3. Составить программу, которая создает стек и выводит его на экран.

Билет № 17

1. Задачи и алгоритмы поиска данных.
2. Алгоритмизация и программирование задач численного интегрирования. Вычисление определенного интеграла методом трапеций.

3. Написать программу с использованием модуля, который вычислял сумму двух переменных c=a+в.

Билет № 18

1. Структура модуля (программные ресурсы модуля, видимые и невидимые объекты). Подпрограммы в модулях.
2. Алгоритмизация и программирование задач численного интегрирования. Вычисление определенного интеграла методом Симпсона
3. Построить гистограмму для сравнительного анализа выпуска деталей каким- то цехом за n месяцев текущего года . Исходные данные получить случайным образом, причем максимальное количество деталей не должно превышать 420.

Билет № 19

1. Процедуры языка Turbo Pascal для установки цвета фона и символов в текстовом режиме (TextColor, TextBackGround, параметр TextAttr).
2. Численные методы решения трансцендентных (нелинейных) уравнений. Метод Ньютона (метод касательных).
3. Составить программу с использованием динамических переменных для вычисления гипотенузы прямоугольного треугольника.

Билет № 20

1. Алгоритм создания динамической цепочки (этапы создания динамической цепочки, ключевое слово NIL, графическая модель динамической цепочки).
2. Алгоритмизация и программирование задач статистической обработки с использованием графического режима. Построение диаграмм.
3. Вычислить интеграл методом Симпсона. Начальное значение x1=0, конечное значение x2=10, точность вычисления e=0.01, подынтегральная функция 1/(x*Ln(x)*0,43429)

Билет № 1

1. ***Диалоговый режим*** - это обмен данными между человеком и компьютером в темпе, соизмеримом с темпом обработки данных человеком.

Интерфейс пользователя (User interface) - программные и аппаратные средства взаимодействия пользователя с программой или ЭВМ.

Одним из распространенных типов интерфейсов человека и компьютера является ***интерфейс на основе меню*** ("смотри и выбирай"). Интерфейс такого типа облегчает взаимодействие пользователя с компьютером, так как устраняет необходимость изучения пользователем языка общения с вычислительной системой.

На каждом шаге диалога пользователю предъявляются все возможные в данный момент команды в виде набора пунктов меню, из которого пользователь может выбрать нужный. Необходимые параметры затем запрашиваются программой с пояснительным текстом.

Структура меню может быть одно- и многоуровневой; в последнем случае часть пунктов текущего меню может иметь подчиненные меню.

Используются также меню в виде пиктограмм, но такие формы чаще используются в графическом режиме.

Примеры структур

Ввод данных	..... .	

Рисунок 8.2 - Меню в виде строки данных (горизонтальное)

Ввод данных
.....
.....

Окончание

Рисунок 8.3 - Меню в виде блока данных (вертикальное)

1. Ввод данных
2. .....
.....
5. Окончание

Качество определяется в ГОСТ Р ИСО/МЭК 9126-93 как «объем признаков и характеристик продукции или услуги, который относится к их способности удовлетворять установленным или предполагаемым потребностям». При комплексной оценке показателей качества программного продукта качество пользовательского интерфейса вносит определяющий вклад в субхарактеристику качества, как практичность (usability) (ГОСТ Р ИСО/МЭК 9126-93). Другими словами, качество характеризует содержание (смысл) и полезность текста, в то время как стандартизированность — грамотность (корректность).
В пользовательском интерфейсе можно выделить два аспекта интерфейса — функциональный и эргономический. О качестве функциональности интерфейса трудно говорить без указания предметной области, например, сформулировать «руководящие принципы функциональности» пользовательского интерфейса. Формально его можно связать со степенью «соответствия задаче»
Очень важным у интерфейса является эргономический аспект, который учитывает: комфортность экранного представления информации, достаточную оперативность реакции программного средства на действия пользователя, удобство манипулирования мышью и клавиатурой, удобство навигации, шрифты, цветовое оформление и т. д. Нормативные требования по эргономике пользовательского интерфейса относятся к психофизиологическим свойствам конкретной реализации

уже выбранного типа (стиля) пользовательского интерфейса (и соответствующего стандарта) в конкретном приложении. В этих условиях эргономические стандарты могут лишь требовать достижения некоторых общих руководящих эргономических принципов, которым должна удовлетворять реализация в приложении выбранного тип (стиля). Цель создания эргономичного интерфейса состоит в том, чтобы отобразить информацию настолько эффективно, насколько это возможно для человеческого восприятия и структурировать отображение на дисплее таким образом, чтобы привлечь внимание к наиболее важным единицам информации. Основная же цель состоит в том, чтобы минимизировать общую информацию на экране и представить только то, что является необходимым для пользователя.

Данные на экране следует располагать таким образом, чтобы пользователь знал, где найти и где ожидать вывода необходимой информации.

Количество информации, отображаемой на экране, называется экранной плотностью. Исследования показали, что чем меньше экранная плотность, тем отображаемая информация наиболее доступна и понятна для пользователя, и, наоборот, если экранная плотность большая, это может вызвать затруднения в усвоении информации и ее ясном понимании. Однако опытные пользователи могут предпочитать интерфейсы с большой экранной плотностью. Информация на экране может быть сгруппирована и упорядочена в значимые части. Это может быть достигнуто с использованием кадров (фреймов), методов типа цветового кодирования, рамок, негативного изображения или других методов для привлечения внимания.

Для привлечения внимания к каким-либо элементам интерфейса можно воспользоваться выделением этих элементов большей яркостью на фоне других – более темных.

Эргономические аспекты пользовательского интерфейса приложения

являются естественным расширением эргономики технических средств и рабочего места. Сегодня существует два подхода к оценке эргономического качества, которые можно отнести к методам «черного» и «белого» ящиков.

В первом подходе оценку производит конечный пользователь (или тестер), суммируя результаты работы с программой в рамках следующих показателей

- эффективности (effectiveness) влияния интерфейса на полноту и точность достижения пользователем целевых результатов;
- продуктивности (efficiency) или влияния интерфейса на производительность пользователя;
- степени (субъективной) удовлетворенности (satisfaction) конечного пользователя этим интерфейсом.

Эффективность является критерием функциональности интерфейса, а степень удовлетворенности и, косвенно, продуктивность — критерием эргономичности.

2. ***Сортировка с помощью прямого обмена (пузырьковая сортировка)***

Сортировка – это процесс перегруппировки заданного множества объектов в некотором определённом порядке.

Цель сортировки: облегчить поиск элементов.

Существует множество методов сортировки, каждый из которых имеет свои достоинства и недостатки. Выбор алгоритма зависит от структуры обрабатываемых данных. Алгоритмы сортировки имеют большое прикладное

значение, они интересны и сами по себе. Это достаточно глубоко исследованная область информатики используется в информационно-поисковых системах, в военном и банковском деле.

Кроме общенаучного интереса к алгоритмам сортировки, в каждом алгоритме интересно оценить и его так называемую сложность. Под

сложностью понимается максимальное число элементарных шагов алгоритма.

На примерах сортировок можно показать, как путём усложнения алгоритма,

хотя под рукой и есть уже очевидные методы, можно добиться значительного

выигрыша в эффективности.

Методы сортировки можно разбить на два класса – сортировку массивов и сортировку файлов. Иногда их называют внутренней и внешней сортировкой,

так как массивы хранятся в быстрой, оперативной, внутренней памяти машины

со случайным доступом, а файлы размещаются в более медленной, но и более ёмкой внешней памяти.

Идея метода: сравнение и смена мест для пары соседних элементов и в продолжении этого процесса до тех пор, пока не будут упорядочены все элементы. i = 1 2 3 4

4 1 1 1

1 4 4 3

7 3 3 4

3 7 6 6

6 6 7 7

Пусть требуется отсортировать массив в порядке убывания(самое маленькое в самом верху). Для этого проделаем следующее:

- будем просматривать по два рядом стоящих числа, начиная снизу и, если они располагаются в неправильном порядке, поменяем их местами;
- так вначале мы оставляем на своих местах 3 и 6;
- затем меняем местами 3 и 7;
- на следующем шаге сравниваем 3 и 1 и оставляем на своих местах;
- меняем местами 1 и 4.

10

Каждый проход обеспечивает всплытие самого «лёгкого» из оставшихся элементов до тех пор, пока он не займёт своё место (т.е. выше него будут только меньшие). Поэтому этот способ сортировки называют сортировкой

методом «пузырька».

Фрагмент программы:

for i: = 2 to n do

for j: = n downto i do

if a[j-1] > a[j] then begin x: =a[j-1]; a[j-1]: = a[j]; a[j]: =x end;

Если бы мы изменили направление просмотра, то за один просмотр добивались бы «затопления» самого тяжёлого элемента из оставшихся. И после

каждого прохода должны явиться к началу массива и снова повторить по парное сравнение. С каждым проходом в старшие позиции будут последовательно попадать элементы с максимальным значением. Фрагмент

программы:

for i: = n-1 downto 1 do

for j: = 1 to i do

if a[j] > a[j+1] then begin x: =a[j]; a[j]: = a[j+1]; a[j+1]: =x end;

Остановиться нужно после того прохода, в котором не было ни одной перестановки.

Алгоритм можно улучшить, если запоминать ещё и положение (индекс) последнего обмена. Ясно, что все пары соседних элементов выше этого индекса

k уже находятся в желаемом порядке. Так же можно увидеть здесь асимметрию.

Один, плохо расположенный пузырёк на «тяжёлом конце» в массиве за один

проход всплывает сразу (займёт своё место за один проход), а на «лёгком конце» - тонет очень медленно (за один проход на одну позицию). Можно улучшить этот метод, чередуя направление последовательных просмотров.

Полученный при этом алгоритм называют «шейкерной сортировкой».

Обменная сортировка

Выполняется последовательный анализ массива данных: если два соседних элемента (ключа) не удовлетворяют условию (1), то они меняются местами; если в процессе такого анализа выполнена, хотя бы одна перестановка, процесс сравнения повторяется с начала массива, иначе алгоритм заканчивает работу.

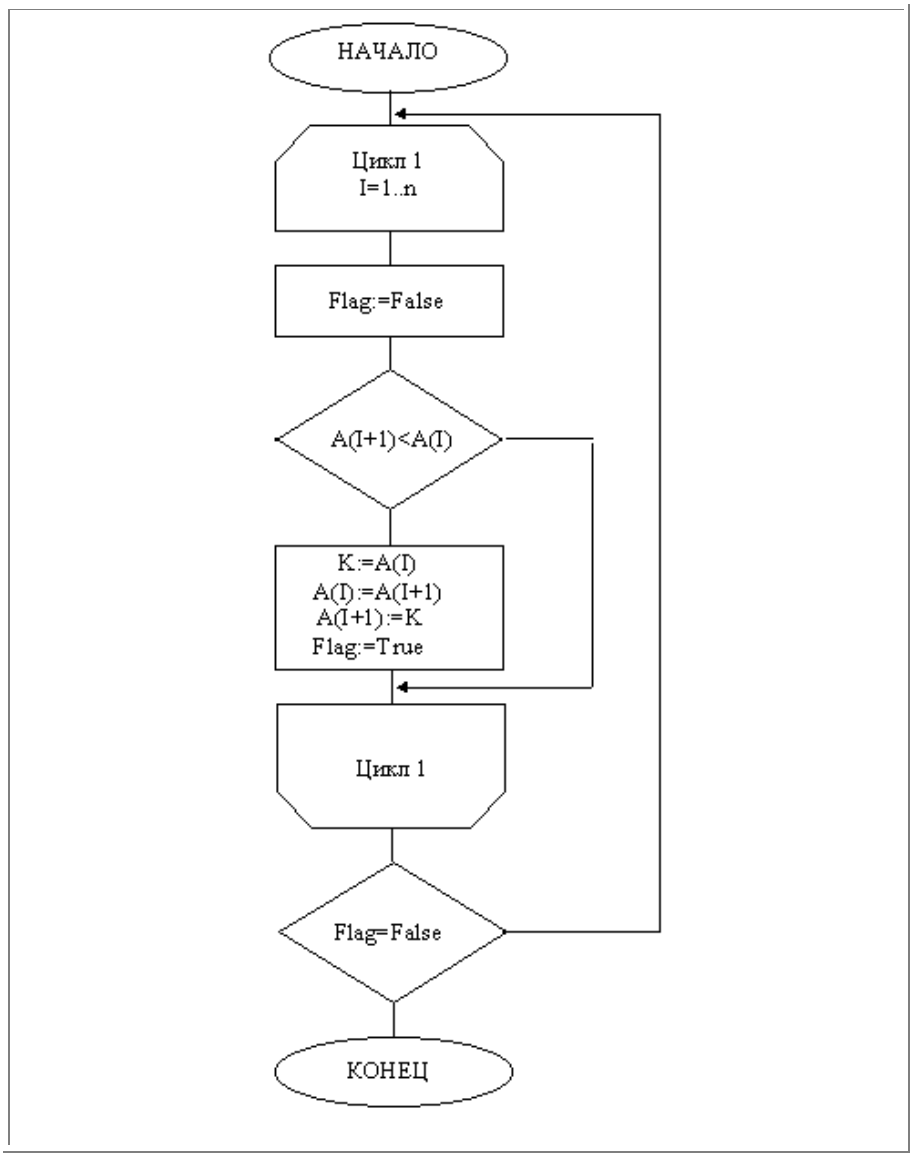
Исходные значения ключей в массиве: 7 8 5 4 1.

1. Поочередно сравниваются два соседних элемента (7 сравнивается с 8 - перестановки не происходит; 8 сравнивается с 5 – происходит перестановка: 8 ставится на место 5, а 5 – на место 8; 8 сравнивается с 4 – происходит перестановка). Цикл заканчивается после сравнения с последним элементом. После выполнения цикла получили массив

2. Если была хотя бы одна перестановка в цикле, то алгоритм продолжается с шага 1. Во время сортировки данного массива цикл был выполнен 4 раза.

Массив после сортировок:

1) 7 5 4 1 8
2) 5 4 1 7 8
3) 4 1 5 7 8
4) 1 4 5 7 8



Билет № 2

1. Структура меню может быть одно- и многоуровневой; в последнем случае часть пунктов текущего меню может иметь подчиненные меню.

Используются также меню в виде пиктограмм, но такие формы чаще используются в графическом режиме.

Ввод данных	....	

Рисунок 8.2 - Меню в виде строки данных (горизонтальное)

Ввод данных
.....
.....
Окончание

Рисунок 8.3 - Меню в виде блока данных (вертикальное)

1. Ввод данных
2. .....
.....
5. Окончание

Рисунок 8.4 - Меню для выбора функции с помощью ввода номера пункта (числа или цифры как символа)

F1. Ввод данных
F2. .....
.....
F5. Окончание

Рисунок 8.5 - Меню для выбора функции с помощью функциональных клавиш

Ввод данных
Сортировка
.....
Окончание

Рисунок 8.6 - Меню для выбора функции с помощью выделенных символов

В примере выбираемые пункты меню выделены прописными буквами.

Рисунок 8.7 - Меню для выбора функции с помощью мнемонических кодов (команд)

Рисунок 8.8 - Выбор с помощью указателя (выделенной строки, курсора)

В примере пункт выделен прямоугольником.

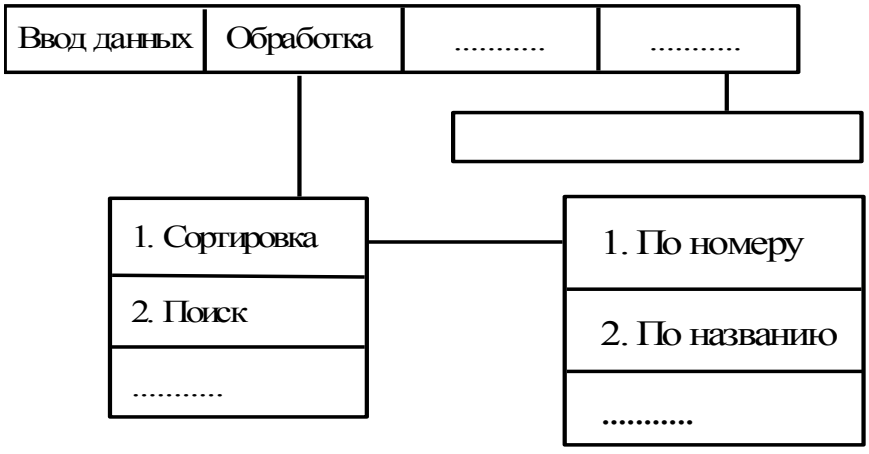


Рисунок 8.9 - Многоуровневое меню

2. лекция, стр 95 И.Г. Семакин , А.П. Шестаков Основы программирования Алгоритм построения графика функции

Для получения графика требуемой функции рекомендуется следующий алгоритм построения графика:

1. Ввод границ изменения аргумента ($X_н$ = A, $X_к$ = B) и количества точек графика N. Вычисление шага изменения аргумента

$$h = \frac{B - A}{N}.$$

2. Оценка значений функции $Y_{фmin}$ и $Y_{фmax}$. Для линейных функций значения Y_{min}, Y_{max} находятся на границах интервала ($X_н$, $X_к$). Для нелинейных функций значения Y_{min}, Y_{max} следует найти, используя методы приближенного поиска экстремумов.

3. Определение масштаба по осям OX и OY соответственно:

$$M_x = (X_{эк} - X_{эн}) / (X_к - X_н); \quad (2)$$

$$M_y = (Y_{эк} - Y_{эн}) / (Y_{фmax} - Y_{фmin}) \quad (3)$$

4. Построение осей координат. Если нулевые значения аргумента и функции входят в диапазоны ($X_н$, $X_к$), ($Y_{фmin}$, $Y_{фmax}$), то следует строить оси по точкам: (0, $Y_{фmin}$), (0, $Y_{фmax}$) и ($X_н$, 0), ($X_к$, 0). В противном случае можно строить оси, начинающиеся в левом нижнем углу экрана (рисунок 14.3).

Нанести штрихи длиной D_s на оси 0X и 0Y, отмечающие минимальное и максимальное значения аргумента и функции.

5. Задание начального значения аргумента X = $X_н$; вычисление FX = f(x).

6. Преобразование координат начальной точки графика A=(X,FX) в систему координат экрана по формулам:

$$X'_a = X_a - X_н; \quad (4)$$

$$Y'_a = Y_a - Y_{фmin}; \quad (5)$$

$$X_{эа} = X'_a \bullet M_x + X_{эн}; \quad (6)$$

$$Y_{э,а} = Y_{э,к} - Y'_a \bullet M_y. \quad (7)$$

В программе при выполнении вычислений следует учесть, что значения координат $X_{э,а}$, $Y_{э,а}$ должны быть целыми; вычисление $X_{э,а}$ и $Y_{э,а}$ рекомендуется оформить в виде функций.

7. Перевод курсора в начальную точку графика ($X_{э,а}$, $Y_{э,а}$).

8. Вычисление следующего значения аргумента X = X + h.

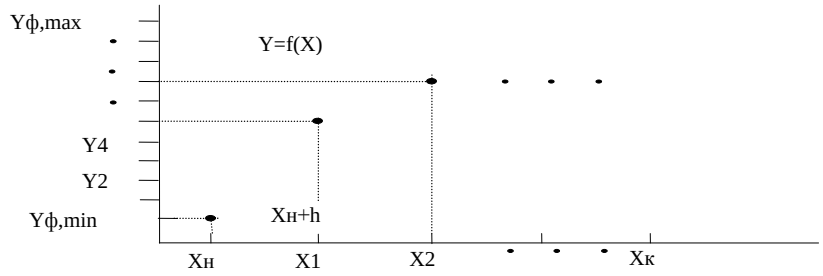
9. Если $X > X_к$, то закончить построение графика.

10. Вычисление FX = f(x). Преобразование координат очередной точки графика (X, FX) в систему координат экрана по формулам (4,5,6,7).

11. Вычерчивание отрезка графика функции от предыдущей точки до очередной с использованием графических процедур и функций модуля Graph Turbo Pascal. Повторять операции пп.8 -11, пока не выполнится условие п.9.

12. В верхней части экрана вывод текста:

График функции Y = ...".



Перед выполнением операций вывода какой-либо информации или изображения на экран следует установить тип, толщину и цвет линии, цвет фона, цвет и шрифт символов.

Билет № 3

1. Для отображения меню на экране используется вывод в заданное место на экране текстового сообщения, которое состоит из нескольких групп слов (группа слов или одно слово - пункт меню). Вывод текста может быть реализован с помощью следующих операторов:

- установка кодов цвета фона и символов с помощью процедур *TextBackGround* и *TextColor* или с помощью присвоения кода параметру *TextAttr* (см. пункт 8.9);

- перемещение курсора в позицию экрана (X,Y) для вывода очередного пункта меню с помощью процедуры *GoToXY(X,Y);*

- вывод на экран текста очередного пункта меню с помощью процедуры *Write (Name_P[i])* при использовании горизонтального меню или

Writeln (Name_P[i]) при использовании вертикального меню.

При использовании пятого варианта выбора пункта меню (рисунок 8.8) следует повторить вывод текста первого пункта меню, предварительно изменив цвет фона и символов, обеспечив реализацию выделения пункта.

Способы выбора пункта меню, показанные на рисунках 8.4 - 8.8, могут быть реализованы следующим образом.

В первом случае (см. рисунок 8.4) следует выполнить ввод целого числа и присвоить его значение переменной типа *integer* или *byte*.

Например:

Readln(Kod_Kl);

Во втором и третьем - нажать функциональную или символьную клавишу; код клавиши может быть определен с помощью функции *ReadKey* и присвоен переменной типа *Char*.

Например:

Kod_Kl:=ReadKey;

В четвертом случае нужно ввести группу символов (мнемонический код) и присвоить ее переменной типа *String* нужной длины.

Например:

Readln(Kod_Com);

Реализация пятого варианта (с помощью выделенной строки) сложнее предыдущих, но проще при использовании. Она рассмотрена в примере программы Prog_Menu_1 (см. пункт 8.8).

Проектирование структуры программы

Если планируется разработка программы с иерархической структурой, включающей головную программу и процедуры для реализации обрабатывающих и других функций. Основные процедуры должны взаимно однозначно соответствовать пунктам меню.

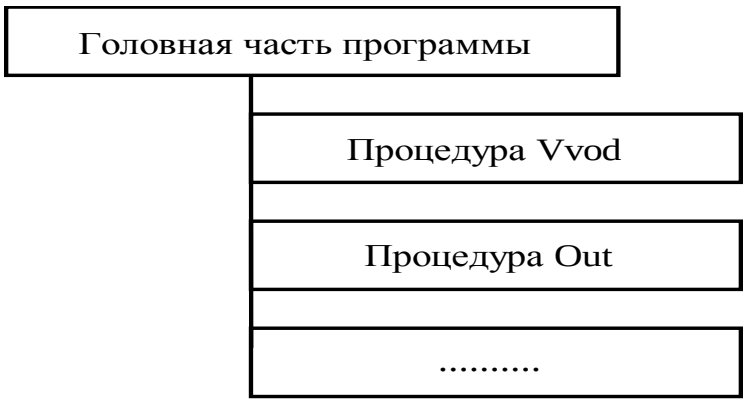


Рисунок 8.11

Головная часть программы может быть построена по различным вариантам. Например:

а) головная часть содержит операторы, реализующие работу меню и обращение к соответствующим процедурам (см. примеры в данном разделе);

б) все основные операции, включая работу с меню, реализуются в виде процедур и функций, а в головной части организуется обращение к ним.

2. Для построения графического изображения на экране дисплея в библиотеку процедур и функций языка Паскаль включен набор графических процедур и функций, которые могут быть вызваны из программы на языке Паскаль.

Любая графическая Паскаль-программа *организуется по следующей схеме:*

- установка графического режима;

- создание и манипулирование графическими объектами;

- закрытие графического режима.

В Паскаль-программу, использующую графические процедуры и функции, должен быть включен файл GRAPH.TPU, где эти процедуры и функции описаны. Для включения любого файла с расширением TPU в программу используется оператор Паскаля USES.

Формат: USES <имя файла TPU>;

Этот оператор должен быть включен в программу сразу же после оператора program. Таким образом, в графических программах вторым оператором должен быть оператор USES GRAPH.

Установка графического режима

Первым шагом любой графической программы является установка соответствующего **графического режима.** Для установки графического режима используется процедура InitGraph.

Формат: InitGraph (<гарф. драйвер>, <граф. режим>, <путь>);

где **<граф. драйвер>** - переменная целого типа, задающая номер графического драйвера, используемого для данного типа терминала. Вы всегда присваиваете этой переменной значение 0, что обеспечивает автоматическое распознавание драйвера и установку графического режима;

<граф. режим> - переменная целого типа, которая принимает значение установленного графического режима. Графический режим определяет размер экрана в пикселях по вертикали и горизонтали и количество допустимых цветов. При задании первого параметра = 0 устанавливается автоматически;

<путь> - это маршрут к каталогу, где находится графический драйвер. Вы будете использовать драйвер CGA.BGI. Для нормальной работы вышей программы необходимо поместить файл CGA.BGI в ваш каталог, а параметр <путь> задать пустой строкой ". Если вы сомневаетесь в правильности выполнения той или иной графической операции, то можете узнать ее код завершения. Для этого используется функция GraphResult.

Формат: GraphResult 0;

Параметров нет. Тип результата - целый. Эта функция возвращает код ошибки для последней графической операции. Если код ошибки равен нулю, то операция выполнена успешно. В противном случае необходимо распечатать код ошибки и посмотреть в документации ее причины.

Для выхода из графического режима используется процедура CloseGraph.

Формат: CloseGraph 0;

По этой процедуре произойдет выход из графического режима в режим, который был установлен до вызова процедуры InitGraph.

Теперь вы знаете три графические операции, которые составляют костяк любой графической программы. Любая написанная вами графическая программа должна содержать в себе следующие операторы: см лекции.

Билет № 4 лекции

1. **Дисплей** служит основным средством отображения вводимой и выводимой информации во время работы на ПЭВМ. Для эффективного управления работой дисплея необходимо ознакомиться с тем, как формируется изображение на экране.

Изображение на экране формируется из светящихся точек (пикселей), которые получаются при столкновении пучка электронов, испускаемого электронной пушкой, с фосфоресцирующей поверхностью лучевой трубки дисплея.
Большинство дисплеев ПЭВМ позволяют работать в текстовом или графическом режимах.
Текстовый режим служит для отображения символов кодовой таблицы и характеризуется максимальным числом символов в строке (для нашего дисплея 80) и количеством строк на экране (25).
Графический режим используется при отображении графической информации и характеризуется разрешающей способностью экрана, т.е. максимальным количеством точек по горизонтали и по вертикали (обычно это 320 или 640 точек по горизонтали и 200 точек по вертикали).
Минимальной единицей управления в текстовом режиме является символ. Символ составляется из нескольких пикселей, преобразование которых в символ происходит на аппаратном уровне. Вся выводимая на экран информация хранится в буфере экрана. Вы можете себе

представить буфер экрана как массив 25x80 в текстовом режиме или 200x640 в графическом, каждой ячейке памяти которого соответствует определенная позиция экрана, символ в текстовом режиме и точка в графическом.

Для хранения одного символа в текстовом режиме требуется два байта. Первый содержит непосредственно символ кодовой таблицы, второй указывает, как символ должен быть выведен на экран (мерцающим, черным, белым и т.д.).

Минимальной единицей управления в графическом режиме является пиксель. Для установки цвета пикселя требуется от 1 до 4 битов (всего 16 цветов), т.к. он не имеет цвета фона и мерцания, как символ в текстовом режиме.

Для выбора подходящего режима необходимо учитывать *следующие особенности текстового и графического режимов:*

- вывод графиков и сложных изображений возможен только на графический дисплей;
- программы, ориентированные на текстовый режим, более универсальны и имеют более широкое применение;
- зависимость программы от цвета в любом режиме сокращает поле ее применения;
- в текстовом режиме текст выводится быстрее, чем в графическом;
- текстовые режимы используют меньше памяти и более широкий выбор цветов;
- в текстовом режиме, в отличие от графического, возможно мерцание выводимых символов.

Для построения графического изображения на экране дисплея в библиотеку процедур и функций языка Паскаль включен набор графических процедур и функций, которые могут быть вызваны из программы на языке Паскаль.

2. Методы решения задачи сортировки делят на внешние и внутренние в зависимости от места расположения данных: если данные располагаются в оперативной памяти, то задача относится к внутренней сортировке; если, преимущественно, во внешней памяти (накопители на магнитных дисках или лентах), то - ко внешней. Методы решения задач сортировки отличаются быстродействием (или временной сложностью), требуемым объемом памяти (или емкостной сложностью).

Сортировка выбором

Сортировка с помощью прямого выбора

Основная идея:

1. Выбирается элемент с минимальным ключом.
2. Он меняется местами с первым элементом.
3. Этот процесс повторяется с оставшимися (n – 1) элементами, (n – 2) элементами и т. д. до тех пор, пока не останется один, самый большой элемент.

Алгоритм сортировки прямого выбора можно сформулировать так:

for i:=1 to n-1 do begin

1. Присвоить к индекс наименьшего из a[i], …, a[n];
2. Поменять местами a[i] и a[k];

end;

Этот алгоритм в некотором смысле противоположен прямому включению. При прямом включении на каждом шаге рассматривается только
один очередной элемент исходной последовательности и все элементы готовой
последовательности, среди которых отыскивается точка включения. При прямом выборе для поиска одного наименьшего элемента рассматриваются все
элементы исходной последовательности и, найденный помещается как

очередной элемент в готовую последовательность.

Пример. 3 5 8 1 2 n=5

i = 1 1 5 8 3 2

i = 2 1 2 8 3 5

i = 3 1 2 3 8 5

i = 4 1 2 3 5 8

Фрагмент программы:

```
for i: =1 to n-1 do
begin k: =i; x: = a[i];
For j: = i+1 to n do if a[j] < x then begin k :=j; x: = a[j] end;
a[k]:= a[i]; a[i]:=x
end;
```

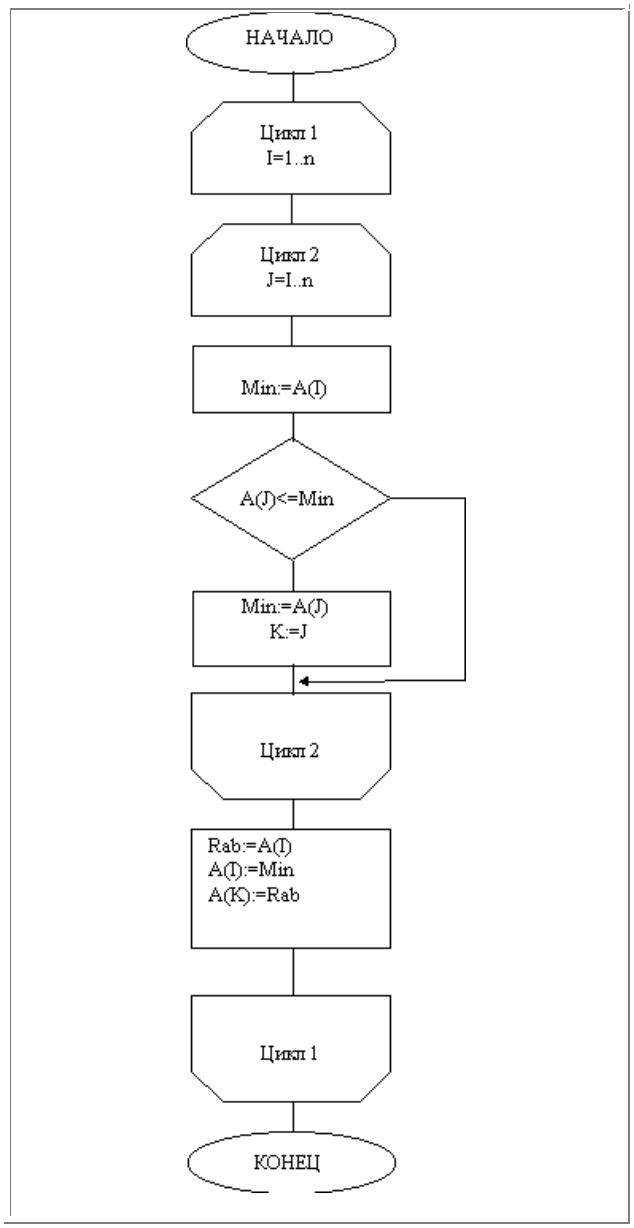
Вначале отыскивается запись с наименьшим значением ключа и помещается на первое место. Эта запись исключается из дальнейшего анализа. Затем отыскивается запись с наименьшим значением ключа в оставшейся части массива, помещается на второе место и исключается из дальнейшего анализа и т.д.

Пример:

Исходные значения ключей в массиве: 7 3 5 6 1.

После первого просмотра массива найден наименьший элемент (ключ) 1; установлен на первое место; а элемент, занимавший это место (7), установлен на место выбранного: 1 3 5 6 7. (Рис. 11.2)

Остальные шаги алгоритма не привели к изменениям (алгоритм продолжал работу независимо от промежуточного результата).



Вначале отыскивается запись с наименьшим значением ключа и она помещается на первое место. Эта запись исключается из дальнейшего анализа. Затем отыскивается запись с наименьшим значением ключа в оставшейся части массива, помещается на второе место и исключается из дальнейшего анализа и т.д.

Пример:

Исходные значения ключей в массиве: 7 8 5 4 1.

1. После первого просмотра массива найден наименьший элемент (ключ=1).

2. Он установлен на первое место, а элемент, занимавший это место (7),

установлен на место выбранного. Получили новый массив 1 8 5 4 7.

3. Находится наименьший элемент в массиве ключей, начиная со второго, находится наименьший (ключ=4), устанавливается на второе место. Получили массив 1 4 5 8 7. Далее поиск продолжается с третьего элемента и т.д. (алгоритм продолжает работу независимо от промежуточного результата, т.е. были ли перестановки или нет).

Пример:

Исходный массив: 2 5 7 4 1 6

1 шаг

Находим наименьший элемент, он равен 1, ставим его напервое место, а первый элемент - на пятое место (вместо 1).

1 5 7 4 2 6

2 шаг

Находим наименьший элемент в массиве, начиная со второго

1 2 7 4 5 6

3 шаг

Находим наименьший элемент в массиве, начиная с третьего

1 2 4 7 5 6

4 шаг

Находим наименьший элемент в массиве, начиная с четвертого

1 2 4 5 7 6

5 шаг

Находим наименьший элемент в массиве, начиная с пятого

1 2 4 5 6 7

3.см практику

Билет № 5

1. см лекции

2. Для установки графического режима используется процедура InitGraph.

Формат: InitGraph (<гарф. драйвер>, <граф. режим>, <путь>);

где **<граф. драйвер>** - переменная целого типа, задающая номер графического драйвера, используемого для данного типа терминала. Вы всегда присваиваете этой переменной значение 0, что обеспечивает автоматическое распознавание драйвера и установку графического режима;

<граф. режим> - переменная целого типа, которая принимает значение установленного графического режима. Графический режим определяет размер экрана в пикселях по вертикали и горизонтали и количество допустимых цветов. При задании первого параметра = 0 устанавливается автоматически;

<путь> - это маршрут к каталогу, где находится графический драйвер. Вы будете использовать драйвер CGA.BGI. Для нормальной работы вышей программы необходимо поместить файл CGA.BGI в ваш каталог, а параметр <путь> задать пустой строкой ". Если вы сомневаетесь в правильности выполнения той или иной графической операции, то можете узнать ее код завершения. Для этого используется функция GraphResult.

Формат: GraphResult 0;

Параметров нет. Тип результата - целый. Эта функция возвращает код ошибки для последней графической операции. Если код ошибки равен нулю, то операция выполнена успешно. В противном случае необходимо распечатать код ошибки и посмотреть в документации ее причины.

Для выхода из графического режима используется процедура CloseGraph.

Формат: CloseGraph 0;

По этой процедуре произойдет выход из графического режима в режим, который был установлен до вызова процедуры InitGraph.

Теперь вы знаете три графические операции, которые составляют костяк любой графической программы. Любая написанная вами графическая программа должна содержать в себе следующие операторы: см лекции. 3. стр 111 И.Г. Семакин , А.П. Шестаков Основы программирования (3 года)

Билет № 6

1. Контроль правильности ввода при выборе пункта необходим во всех случаях, где выбор выполняется вводом кода (числа, символа, команды и т.п.). Контроль целых чисел может быть выполнен с помощью проверки значения переменной на соответствие заданному диапазону; например: если Kod_Kl < 1 или > 5, то - ошибка (следует выдать сообщение об этом и перейти к повторению ввода). Контроль кода клавиши или символа может быть выполнен с помощью проверки принадлежности кода заданному множеству; например:

if Kod_Kl in ['В', 'Ы', 'О']

then ...

{ввод без ошибок; можно выполнять выбор процедуры для выполнения}

else …{ошибка}

Рекомендуется оформить в виде процедуры группу операций по перемещению курсора в заданное место экрана, установка цветов и выводу текста на экран.

Например:

Procedure Proc1 (Color_Code, Back_Code: byte;

X, Y: integer; Text: string);

Begin

TextColor (Color_Code);

TextBackGround (Back_Code);

{или TextAttr:=16*Back_Code+Color_Code) вместо двух предыдущих операторов}

GoToXY (X, Y);

Write (Text)

End;

……

В процедурах используются параметры:

Color_Code - код цвета символов,

Back_Code - код цвета фона,

X, Y - координаты начала строки Text, выводимой на экран.

Координаты позиций символов отсчитываются от левого верхнего угла экрана.

2. Динамические цепочки (списки, массивы)

Динамические цепочки - это последовательности взаимосвязанных динамических переменных, которые можно предоставить в виде графической модели: (рис. 10.5)

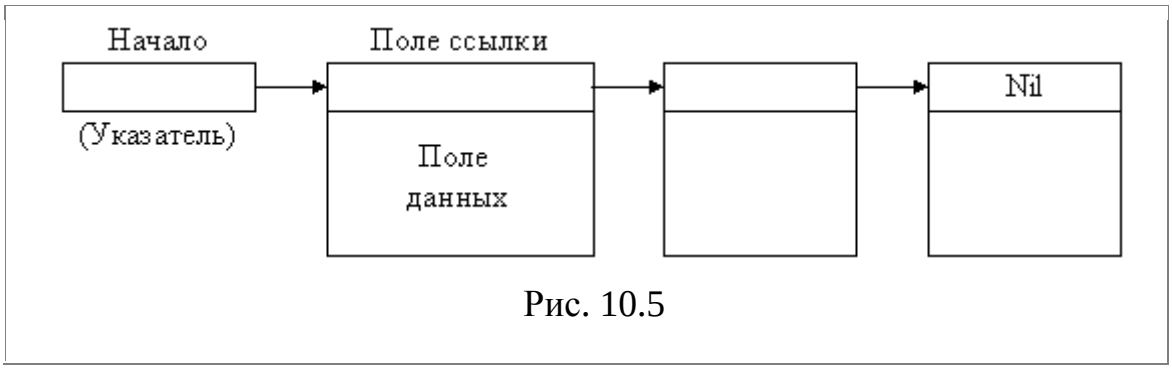


Рис. 10.5

Эта модель называется однонаправленным списком. Начало динамической цепочки должно быть зафиксировано. Если адрес первого элемента потерян, то доступ ко всей динамической цепочке будет утерян, и она станет недоступной.

Такие динамические цепочки можно использовать, как массивы переменной длины. Для обозначения окончания динамической цепочки в поле ссылки последнего элемента помещают специальную константу Nil ссылочного типа, которая не куда не указывает.

Также можно использовать двунаправленный список, с указанием на предыдущий элемент (рис. 10.6).

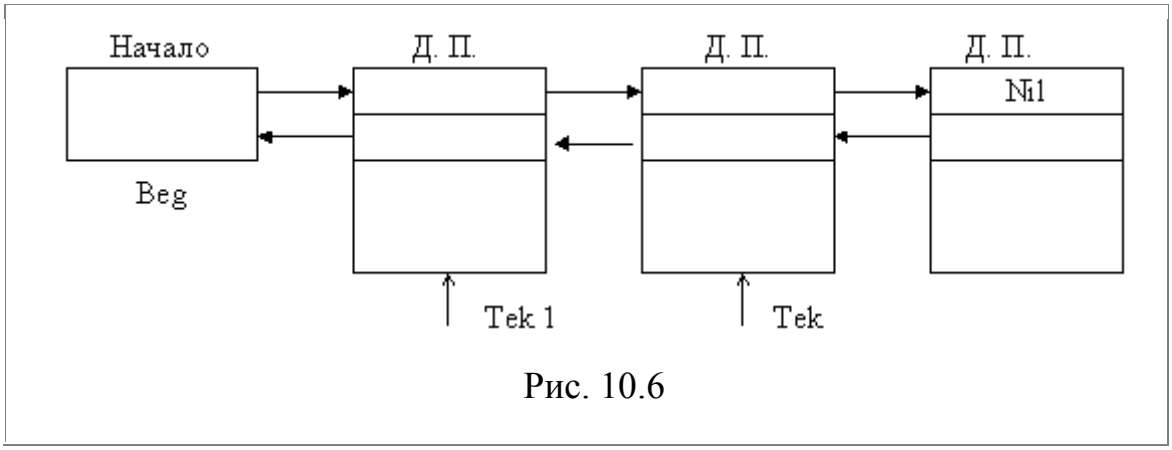


Рис. 10.6

Для работы с динамическими списками требуется несколько ссылочных переменных:

1. Указатель на начало списка (например, вед),

2. Указатель на текущий элемент, с которым выполняются действия в данный момент: чтение, запись, сравнение и др.(тек.);
3. Tek 1 - указывает на элемент, предшествующий текущему,
4. New(P) - указатель для добавления нового элемента динамической цепочки.

Стр 140-143 И.Г. Семакин , А.П. Шестаков Основы программирования (3 года) стр 138-149 с.а. Немнюгин турбо паскаль практикум 2-е издание 3.1 pas

Билет № 7

1. (лекция) Для выполнения любой программы сама программа (её код) и необходимые данные размещаются в оперативной памяти, размеры которой ограничены. Поэтому этот ресурс ЭВМ использовать, по возможности, эффективно, т.е. обеспечить решение задач максимально возможной размерности, с наибольшей производительностью. При решении многих задач заранее неизвестно: - потребуется ли значения некоторых переменных вообще в процессе решения, - сколько элементов массива (списка) будет обрабатываться. При разработке программы необходимо резервировать место в ОП, исходя из предположение наихудшего случая, в частности, определять массивы максимально необходимой длины. В процессе работы программы одна часть данных используется с начала и до конца периода работы программы (например, некоторые константы), а другая часть данных требуется эпизодически, временно. Для первой группы данных должно быть выделено место в ОП на весь период работы программного модуля. По отношению ко второй группе данных можно поступить двояко: либо для них постоянно выделять место в оперативной памяти (в этом случае оперативная память часть периода работы ЭВМ не будет использоваться из-за ненужности временных данных), либо выделять место в ОП по необходимости (только когда необходимо использовать временные данные), а затем освобождать его. Второй вариант предпочтительнее, так как

обеспечивается эффективное использование ОП в процессе работы программного модуля (свободные участки ОП могут использоваться различными данными в разные моменты времени, что позволяет решать задачи максимально возможной размерности, организовывать массивы данных переменной длины).

Статические и динамические переменные в языке Паскаль. Модели переменных

Статическими называются переменные, для которого выделено место в ОП в течение модуля всего периода работы программного модуля (процедуры, функции, программы). Выделение места в ОП для статистических переменных можно представить в виде табл.10.1 и рис.10.1.

Таблица 10.1

Имя	Адрес	Тип	Длина
x	1000	Real	6
y	1006	Integer	4
z	1010	Array	40
z1	1050	[1..10]	. . . .
		of	
		integer	
		. . . .	

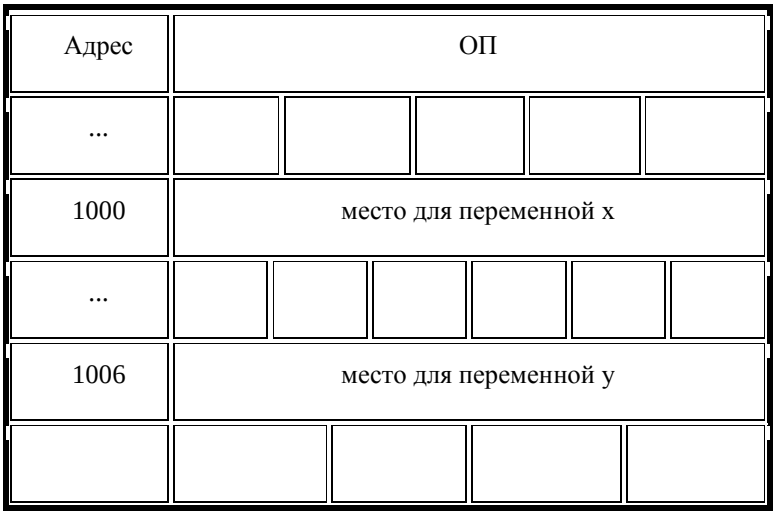


Рис.10.1

Такой способ распределения требует выделения памяти, для переменных исходя из наихудшего случая (рассчитывая на максимально возможное количество данных); особенно это относится к массивам.

Другой способ распределения ОП называется **динамическим.** В этом случае используются динамические переменные, которые создаются и уничтожаются по необходимости (т.е. для них выделяется ОП или от них освобождается ОП).

В этом случае одни и те же участки ОП в различные моменты времени могут быть заняты различными переменными в течение периода работы программного модуля. Использование динамических переменных позволяет уменьшить требуемый объем ОП по сравнению с использованием статических переменных. Для работы с динамическими переменными используются переменные ссылочного типа (указатели), которые содержат адреса динамических переменных.

Указатели являются статическими переменными. Модель использования динамических переменных представлена на рис 10.2

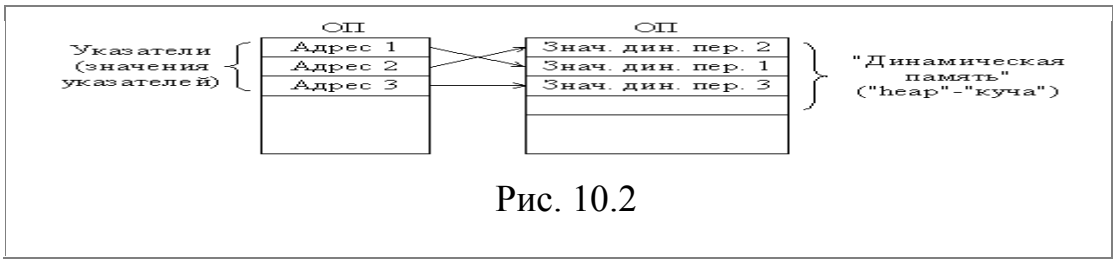


Рис. 10.2

Указатели содержат адреса динамических переменных.

В течение периода работы программного модуля один и тот же указатель может содержать адреса различных динамических переменных.

2. Проектирование размещения информации на экране

При использовании интерфейса типа «меню» на экране должны быть запланированы области (окна, поля) для вывода названия программы, меню, сообщений об ошибках, подсказок. Схемы расположения этих окон представлены на рисунке. 8.10. На рисунке 8.10.а дана примерная схема при вертикальном расположении меню, на рис. 8.10.б - при горизонтальном расположении (меню в виде блока). Во втором случае напротив каждого пункта меню может располагаться подсказка, поясняющая назначение пункта.

При проектировании образа экрана следует стремиться к равномерному и, по возможности, симметричному расположению данных на экране.

Как правило, окна обрамляются одиночными или двойными рамками из псевдографических символов, которые вводятся с помощью нажатия клавиши *Alt* и кодов, данных в таблице 8.3 Коды вводятся с дополнительной (правой) клавиатуры при нажатой клавише *Alt*.

Рекомендуется использовать для оформления не более 4 - 5 цветов; при этом:

все пункты меню должны быть одного цвета, за исключением выделенного; выделение пункта меню может выполняться инверсией цвета символов и фона относительно остальных строк меню, или изменением цвета фона строки и цвета символов, или изменением только цвета фона или только цвета символов,

- выделение отдельных символов выполняется размером и/или цветом,
- при выделении пункта меню цветом не использовать плохо различимые, близкие по спектру сочетания цветов для фона и символа (сиреневые на синем, коричневые на красном и т.п.).

Окно названия программы может быть просто строкой текста, содержащей название программы; например, "Подготовка результатов сессии".

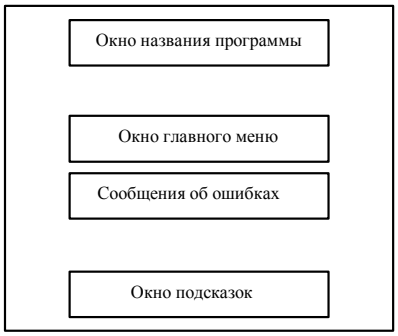
Окно подсказок должно содержать общие указания типа:

“ Left, Right - перемещение по меню”;

“Enter - выполнение пункта".

В него же, не стирая общей подсказки, должны выводиться подсказки, поясняющие содержание операции, которая будет выполняться при выборе выделенного пункта меню.

а)



б)

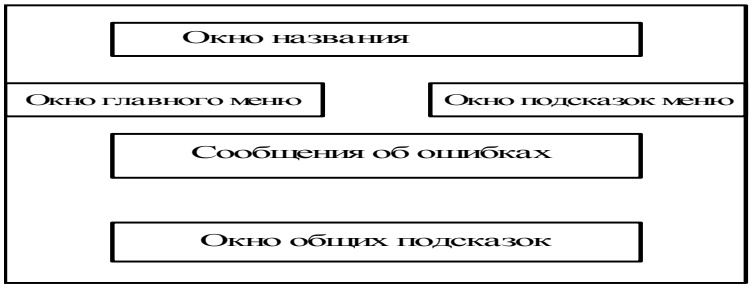


Рисунок 8.10

Билет № 8

1. см лекция стр. 123-129 СА Немнюгин турбо паскаль практикум , стр 146-152 И Г Семакин А.П. Шестаков Основы программирования

Модуль компилируется точно таким же образом, как и обычные программы, но так как модуль не является непосредственно выполняемой

единицей, то в результате его компиляции образуется дисковый файл с расширением .TPU (Turbo Pascal Unit), а не с расширением .EXE. При компиляции модуля имя файла (NAME.TPU) берется из имени файла с исходным текстом модуля (NAME.PAS).

Чтобы основная программа могла воспользоваться константами, переменными, процедурами и другими объектами, описанными в интерфейсной секции используемого модуля, необходимо указать в программе имя нужного *TPU*-файла. Соответствующая конструкция называется *спецификацией используемых модулей* и имеет следующий общий вид:

USES *Name1, | Name2...*; { *Name1, Name2 - имена используемых модулей}*

Эта спецификация должна идти непосредственно после заголовка программы. В модуле типа *UNIT* такая спецификация должна следовать сразу после служебного слова *INTERFACE*. При этом имя модуля (например, *Unit1*) должно совпадать с именем файла, содержащего его текст и объектный код (например, *UNIT1.PAS* и, соответственно, *UNIT1.TPU*).

Если имя модуля (например, *Unit_M*) отличается от имени файла с его текстом (например, *FILE_UN.PAS*), то программа должна содержать директиву *{$U}* для переопределения имени файла. Директива должна находиться непосредственно перед именем модуля в спецификации использования. Например, конструкция

USES { $U FILE_UN } Unit_M;

приведет к тому, что компилятор будет искать код модуля *Unit_M* в дисковом файле *FILE_UN.TPU*.

При наличии спецификации использования в данной программе считаются известными все описания из интерфейсной секции подключенного модуля. К интерфейсным объектам модуля можно

обращаться в программе точно так же, как если бы они были описаны в самой программе.

Ниже приведен пример программы с использованием модулей, а также текст самих модулей.

Главная программа

Файл *OSNOV.PAS*

```
Program Osn;                { Главная часть модульной программы }
 uses Crt, Unit1, Unit2;
 begin                      {начало блока операторов}
 ClrScr;                    { Процедура очистки экрана из модуля Crt }
{----------Использование модулей----------}
 repeat
  run1;                     {Коды процедур run1 и run2 расположены в}
  run2                      {модулях Unit1 и Unit2}
 until KeyPressed
end.                        {Конец программы
```

При трансляции программы, использующей модули типа *UNIT*, компилятор последовательно отыскивает файлы, содержащие коды используемых модулей (с расширением .*TPU*),с тем чтобы подключить их к компилируемой программе. При этом компилятор работает по следующему алгоритму:

- компилятор просматривает содержимое системного библиотечного файла модулей *TURBO.TPL* (Turbo Pascal Library);

- если искомый модуль не найден в файле *TURBO.TPL*, то компилятор осуществляет поиск соответствующего TPU-файла в текущем каталоге;

- если в текущем каталоге нужный файл не найден, то поиск продолжается в каталогах, заданных в альтернативе *Options*/*Directories*/*Unit Directories* для интегрированной среды;

- если на предыдущих шагах файл не найден, то компилятор прекращает работу и выдает диагностическое сообщение об ошибке;

- если компилятор активизирован посредством альтернатив *Compile/Make* или *Compile/Build*, то вышеуказанные шаги проводятся в поисках исходных текстов используемых модулей, которые будут, оттранслированы перед рансляциией самой программы при этом подразумевается, что имя файла с текстом модуля совпадает с именем модуля и имеет расширение *.PAS.*

Процесс трансляции программы, использующей модули типа *UNIT*, можно представить следующей схемой:

Исходный текст

PAS – файлы Результат компиляции

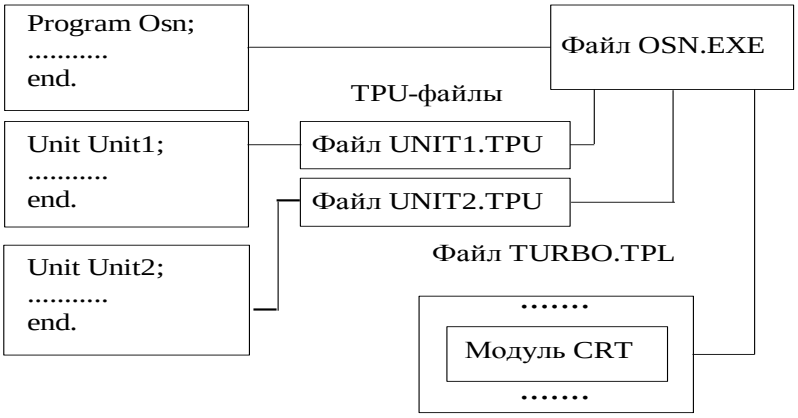


Рисунок 9.1

2. лекция **В графической библиотеке языка Паскаль** существуют процедуры, позволяющие вычерчивать следующие фигуры: прямоугольник, закрашенный столбец, трехмерный параллелепипед, окружность, сектор круга. Рассмотрим более подробно каждую из этих процедур.

Для обрисовки прямоугольников используется процедура Rectangle.

Формат: Rectangle 0(x1, y1, x2, y2);

Эта процедура рисует прямоугольник, используя текущий цвет и тип линии. Параметры - целые. (x1,y1) - координата верхнего левого угла

прямоугольника, (x2, y2) - координата правого нижнего угла прямоугольника.

Эта же процедура позволяет вычерчивать и квадраты. Однако, при вычерчивании этих фигур надо учитывать, что экран терминала устроен так, что на единичном отрезке по горизонтали пикселей больше, чем на единичном отрезке по вертикали. Поэтому, для достижения пропорциональности сторон необходимо использовать коэффициент сжатия для координаты Y, либо коэффициент расширения для координаты X.

Для вычисления коэффициента относительного удлинения необходимо использовать процедуру GetAspectRatio.

Формат: GetAspectRatio 0(Xk, Yk);

где Xk, Yk - целые параметры, которые получают реальную для данного терминала разрешающую способность по оси X и по оси Y. Отношение Xk/Yk или Yk/Xk и есть коэффициент сжатия или расширения.

Для примера приведу фрагмент программы, отрисовывающий квадрат, сторона которого равна 20 пикселам, в точке (100,100). x:=20;

Для отрисовки закрашенного столбца используется процедура Bar.

Формат: Bar 0(x1, y1, x2, y2);

Эта процедура рисует столбец используя стандартный тип и цвет закраски. Параметры целого типа. (x1,y1) - координата левого верхнего угла, (x2,y2) - координата правого нижнего угла. Процедура используется для отрисовки гистограмм.

Для установки образца и цвета закраски используется процедура SetFillStyle.

Формат: SetFillStyle 0(<образец>, <цвет>);

где **<цвет>** - один из цветов палитры (0 - фоновый цвет);

<образец>: 0 - фоновый цвет; 2 - ---; 3 - ///; 4 - жирными ///; 5 - жирными \\\; 7 - редкой штриховкой; 8 - частой пересекающейся штриховкой; 9 - прерывистой линией; 10 - редкими точками; 11 - частыми точками.

В Паскале есть еще одна процедура установки образца закраски, заданного пользователем, но мы не будем ее рассматривать. Таким образом используя процедуру SetFillStyle с различными параметрами мы можем получить столбцы с различной закраской.

Для отрисовки трехмерного параллелепипеда используется процедура BAR3D.

Формат: BAR3D 0(x1, y1, x2, y2, <глубина>, <вершина>);

Первые четыре параметра целые и определяют координаты передней панели параллелепипеда. Параметр **<глубина>** также целого типа и определяет в пикселах глубину параллелепипеда. Последний парамет **<вершина>** - параметр логического типа. Его значение определяет, будет ли отрисовываться трехмерная вершина, или нет. Если параметр - True, вершина отрисовывается.

Передняя стенка параллелепипеда также закрашивается типом закраски, установленным процедурой SetFillStyle.

Для отрисовки и заполнения цветом сектора круга используется процедура PieSlice.

Формат: PieSlice 0(x, y, <нач. угол>, <кон. угол>, <радиус>);

Точка (x,y) - центр окружности. Сектор рисуется от начального до конечного угла заданным радиусом. Точка отсчета для угла, от оси X вверх. Сектор закрашивается типом закраски, заданным процедурой SetFillStyle.

3. см практ раб

билет№9

1. лекция После включения ПЭВМ дисплей по умолчанию находится в одном из текстовых режимов. Для переустановки текстового режима или выхода из графического режима используется процедура *TextMode*.

Назначение: устанавливает указанный текстовый режим.

Описание: *TextMode(режим)*.

Тип параметра: целый.

Допустимы следующие режимы:

0 - 40x25, черно-белый;

1 - 40x25, цветной;

2 - 80x25, черно-белый;

3 - 80x25, цветной.

По умолчанию на наших ПЭВМ установлен режим 2.

При работе с монохромным (не цветным) дисплеем возможно использование только двух цветов - черного и белого (у нас черного и зеленого). Однако Паскаль предоставляет рая *дополнительных возможностей вывода на экран:*

- символы могут отличаться интенсивностью свечения;
- белые символы могут выводиться на черном фоне, а черные на белом;
- символы могут мерцать.

В Турбо-Паскале имеется байтовая переменная *TextAttr*, устанавливая значение которой мы можем регулировать режим вывода на экран. Эти режимы вы легко узнаете, запустив следующую программу.

Зная о возможности изменения способа вывода информации вы можете более оригинально оформлять вывод информации на экран.

Текстовые окна

Понятие 'окно' является одним из основных в области создания программного обеспечения для ПЭВМ.

Окно - это ограниченная область экрана, выполняющая те же функции, что и полный экран.

Язык Паскаль позволяет организовывать окна в любой части экрана с помощью процедуры *Window*. Параметры процедуры задают координаты

левого верхнего и правого нижнего угла окна. Максимальный размер окна - полный экран, минимальный - два столбца в двух строках.

Описание: *Window(x1,y1,x2,y2).*

Верхнему левому углу экрана соответствует координата (1,1). В режиме 2 (80x25) по умолчанию окно определено как (1,1,80,25).

На экране может находиться несколько окон, но в один момент времени активным может быть только одно окно. С момента ограничения окна процедурой *Window* оно ведет себя как полный экран. Оставшаяся часть экрана остается неприкосновенной до конца программы, или до организации нового окна. Окна могут налагаться друг на друга, пересекающаяся часть стирается текущим окном, непересекающаяся часть старого окна остается на экране, но является недоступной.

Не следует думать, что после выполнения процедуры *Window*, на экране явно появится некоторый прямоугольник. Это не так. Экран не меняется, происходит усечение полного экрана до размеров, указанных при последнем обращении к этой процедуре.

При работе с окнами используются те же процедуры и функции, что и при работе с полным экраном. Однако следует учитывать, что все координаты задаются относительно активного окна , а не полного экрана.

Функции, используемые в текстовом режиме

Рассмотрим еще раз процедуры и функции, используемые в текстовом режиме для работы с экраном или окном экрана. При рассмотрении этих функций может быть удобнее представлять весь экран как окно максимальных размеров.

Процедура *GoToXY*

Назначение: позиционирование курсора.

Описание: *GoToXY(x,y).*

Тип параметров: байтовый.

Курсор перемещается в ту позицию внутри текущего окна, которая задана координатами x и y (x - столбец, y - строка). Зависит от текущего окна. Если заданы недопустимые координаты, то обращение к процедуре игнорируется.

Функция *WhereX*

Назначение: возвращает для текущей позиции курсора относительно текущего окна координату X.

Описание: WhereX.

Тип результата: байт.

Например, a:=WhereX.

Функция WhereY

Назначение: возвращает для текущей позиции курсора относительно текущего окна координату Y.

Описание: *WhereY*

Тип результата: байт.

Например, a:=WhereY.

Процедура *ClrScr*

Назначение: очищает окно и помещает курсор в его верхний левый угол. Все позиции символов заполняются пробелами. При этом используется текущее значение атрибута TextAttr. Если в атрибуте TextAttr установлен фон символов зеленый, то происходит закраска области окна зеленым цветом, если фон черный, то окно просто очищается.

Процедура *ClrEol*

Назначение: стирает или закрашивает (в зависимости от значения атрибута TextAttr) символы от позиции курсора, до конца строки. Курсор при этом не перемещается. Процедура также зависит от используемого окна.

Процедура *DelLine*

Назначение: удаляется строка, в которой расположен курсор. Все строки, расположенные ниже, перемещаются вверх. Внизу окна добавляется новая строка, причем она заполняется пробелами или закрашивается.

Процедура *IntLine*

Назначение: начиная с позиции курсора вставляет пустую строку. Все строки, расположенные ниже добавляемой строки, перемещаются на одну строку вниз, а нижняя исчезает с экрана. Всем позициям символов новой строки присваивается значение пробелов или она закрашивается. Зависит от используемого окна.

Процедуры *Write, Writeln, Read, Readlln* работают также, как и с полным окном, но ввод и вывод осуществляется только в пределах активного окна. Так, если выводимая строка длиннее строки окна, то осуществляется перенос остатка выводимой строки на следующую строку окна.

Для пояснения работы с окнами составим программу, которая поддерживает на экране два окна. В одном информация выводится в обычном режиме снизу вверх, во втором, сверху вниз.

. Использование символов псевдографики

Средства языка Паскаль позволяют строить и выводить на экран в текстовом режиме довольно сложные изображения: гистограммы, рамки и т.д. Для этого используются т.н. символы псевдографики:

горизонтальные и вертикальные линии, заштрихованные области разной формы, стыковочные соединения и т.д.

Доступ к символам псевдографики может быть осуществлен различными путями. С помощью функции Chr, которая возвращает символ, соответствующий указанному коду. Например Chr(122) возвратит символ z. Вы можете посмотреть всю кодовую таблицу символов с использованием данной функции с помощью следующей программы.

Символы псевдографики можно получить с помощью знака #. Например, оператор writeln(#165) выдаст на экран символ ¦.

Кроме того, символы псевдографики можно получить непосредственно с помощью клавиши <ALT>. Для этого надо держа клавишу <ALT> в нажатом состоянии набрать код символа и отпустить клавишу. Например, ALT 149 вызовет появление на экране символа ¦.

Для построения изображений символы псевдографики применяются обычно в операторах вывода или набираются

Таблица 8.3- Примеры кодов псевдографических символов

Символ	Код	Символ	Код	Символ	Код
┌	218	╖	187	╬	206
┬	194	├	195	╣	185
┐	191	┼	197	└	192
╔	201	┤	180	┴	193
╦	203	╠	204	┘	217
╚	200	╩	202	╝	188
┃	179	═	205	╟	186

2. ***Цифровая сортировка***

Этот метод еще называется сортировкой "вычерпыванием". Он удобен для сортировки целых чисел (ключей), имеющих сравнительно небольшой диапазон.

Пусть даны *N* целых чисел в массиве *A*, принадлежащих диапазону от A_{min} до *Amax* включительно.

Каждому числу A_i поставим в соответствие j-й элемент массива L; при этом индекс элемента определяется по формуле

$j = A_i - A_{min} + 1$.

Необходимо, чтобы массив L имел длину не менее $K = A_{max} - A_{min} + 1$.

Алгоритм состоит из следующих шагов (рис. 11. 6):

1) Поиск в массиве A минимального A_{min} и максимального A_{max} числа.

2) Обнуление массива L.

3) Просмотр массива A и заполнение массива L по правилу:

$j = A_i - A_{min} + 1$,

$L_j = L_j + 1$,

4) Просмотр массива L и заполнение массива результата (A или другого):

- если $L_j = 0$, то перейти к следующему элементу массива L,

- если L_j <.> 0, то записать в выходной массив L_j чисел,

равных $j + A_{min} - 1$.

Пример:

Массив исходных чисел (A):

2 6 3 2 12 7 2 5 12 13 6

Массив L:

3 1 0 1 2 1 0 0 0 0 2 1

Массив A после сортировки

2 2 2 3 5 6 6 7 12 12 13

{===== Программный пример 3.15 =====}

{ **Цифровая сортировка (распределение)** }

```
const D=...; { максимальное количество цифр в числе }
P=...; { основание системы счисления }
Function Digit(v, n : integer) : integer;
{ возвращает значение n-ой цифры в числе v }
begin
for n:=n downto 2 do v:=v div P;
Digit:=v mod P;
```

```
end;
Procedure Sort(var a : Seq);
Var b : array[0..P-2] of integer; { индекс элемента,
следующего за последним в i-ой группе }
i, j, k, m, x : integer;
begin
for m:=1 to D do begin { перебор цифр, начиная с младшей }
for i:=0 to P-2 do b[i]:=1; { нач. значения индексов }
for i:=1 to N do begin { перебор массива }
k:=Digit(a[i],m); { определение m-ой цифры }
x:=a[i];
{ сдвиг - освобождение места в конце k-ой группы }
for j:=i downto b[k]+1 do a[j]:=a[j-1];
{ запись в конец k-ой группы }
a[b[k]]:=x;
{ модификация k-го индекса и всех больших }
for j:=k to P-2 do b[j]:=b[j]+1;
end;
end;
```

Результаты трассировки программного примера 3.15 при P=10 и D=4 представлены в таблице 3.9.

Цифровая

Этой сортировкой можно сортировать *целые неотрицательные числа большого диапазона*. Идея состоит в следующем: отсортировать числа по младшему разряду, потом устойчивой сортировкой сортируем по второму, третьему, и так до старшего разряда. В качестве устойчивой сортировки можно выбрать сортировку подсчетом, в виду малого времени работы. Реализация такова:

```
Program RadixSort;
Var A,B   : array[1..1000] of word;
```

```
  N,i    : integer;
  t      : longint;
Procedure Sort; {сортировка подсчетом}
Var C    : array[0..9] of integer;
   j     : integer;
Begin
 For j:=0 to 9 do
 C[j]:=0;
 For j:=1 to N do
 C[(A[j] mod (t*10)) div t]:= C[(A[j] mod (t*10)) div t]+1;
 For j:=1 to 9 do
 C[j]:=C[j-1]+C[j];
 For j:=N downto 1 do
 begin
  B[C[(A[j] mod (t*10)) div t]]:=A[j];
  C[(A[j] mod (t*10)) div t] := C[(A[j] mod (t*10)) div t]-1;
 end;
End;
Begin
{Определение размера массива A (N) и его заполнение}
 …
{сортировка данных}
 t:=1;
 for i:=1 to 5 do
 begin
  Sort;
  A:=B;
  t:= t*10;
 end;
{Вывод массива A}
```

…

End.

.3.2 паскаль

билет № 10

1. см лекцию, практику При использовании метода прямоугольников для вычисления суммы S1 используются формулы (рис. 1):

$$S1 = h[f(X_1) + f(X_2) + ... f(X_n)], \quad (1)$$

где $X_1 = A + h/2$, $X_1 = X_{i-1} + h$.

2. лекция ***Нанесение точек и вычерчивание линий***

Простейшие графические процедуры и функции оперируют с одним пикселем. При этом можно выполнять *следующие операции:*

1. Проанализировать пиксел, т.е. проверить, установлен ли он на какой-либо цвет. Для этого используется функция GetPixel.

Формат: GetPixel 0(x,y);

где x, y - параметры целого типа, определяющие координаты анализируемого пикселя. Результат - номер цвета из палитры цветов, в который установлен данный пиксел.

2. Установить пиксел на указанный цвет или отрисовать точку заданным цветом. Для этого используется процедура PutPixel.

Формат: PutPixel 0(x,y,<цвет>);

где все параметры целые, (x,y) - координаты пикселя, <цвет> его цвет.

3. Погасить пиксел. Погасить пиксел можно отрисовав его используя цвет фона.

Для демонстрации этих функций приведу пример процедуры, которая для заданного пикселя либо гасит его, если он нарисован цветом, отличным от фонового, либо устанавливает его в указанный цвет.

Формат: GetX; GetY;

Параметров нет. Тип результата - целый. Эти функции возвращают X- или Y-координату текущего указателя. Обе функции зависят от "окна". Следующие две процедуры используются для установки или перемещения текущего указателя.

3.см практику

Билет № 11

1. лекция ***Определение переменных ссылочного типа в языке Паскаль***

Динамические переменные не имеют индетификаторов, для доступа к ним используются переменные ссылочного типа. Для обозначения ссылочного типа используется специальный символ “^” (caret), который указывается перед базовым типом. Базовый тип - это тип динамических переменных, на которые может указывать ссылочная переменная.

Пример:

Работа с динамическими переменными выполняется с помощью переменных ссылочного типа следующим образом:

P1 - значение переменной ссылочного типа,

P1 ^ - значение динамической переменной; на которую указывает переменная P1.

Динамические переменные могут использоваться во всех операторах в соответствии с правилами использования базового типа, например, A^ - динамическая переменная типа Real, на которую указывает ссылочная переменная A; R1^ - динамическая переменная типа Rec, на которую указывает ссылка R1.

При использовании динамических переменных типа "запись" доступ к отдельным полям происходит так же, как рассмотренов разделе 7▶; например:

Значения переменных ссылочного типа можно присваивать друг другу, сравнивать с друг другом (на равенство, и неравенство) только для одинаковых базовых типов данных; например:

if P2 = P1 then ... ; {допустимо}

if R1 <> P1 then ... {недопустимо}

В языке Паскаль имеется бестиповой указатель, который обозначается словом **"Pointer";** его можно использовать для присвоения значений ссылок на любые динамические переменные.

2. **Графические процедуры и функции работы с текстом :**

OutText (str:string) - процедура. С позиции курсора выводится строка str. Автоматической переверстки строк не производится. Шрифт устанавливается с помощью процедур *SetTextJustify, SetTextStyle, SetUserCharSize*.

OutTextXY (x,y:integer;str:string) - процедура.

Строка *str* выдается, начиная с позиции (x,y). Прочее как для ***OutText.***

SetTextStyle (font,dir,gr:word) - процедура. Для следующего вывода текста задаются шрифтовой фонт, наклон и размер символов. Параметр Font может принимать значения 0, 1, ... 7 (или соответствующие обозначения констант:

defaulfont = 0, *sansseriffont = 3,*

triplexfont = 1, *gothicfont = 4,*

smallfont = 2 и др.).

Параметр Dir может принимать значения *HorizDir = 0* (слева направо) или *VertDir = 1* (снизу вверх, строка повернута на 90 градусов против часовой стрелки). Параметр gr может принимать значения *NormSize = 1* (или 0) или 2, 3, ...; в первом случае используются минимальные размеры выбранного шрифта (размеры по умолчанию).

SetUserCharSize (mx,dx,my,dy:word) - процедура.

Устанавливает коэффициенты увеличения символов по осям OX, OY:

"Ширина символа новая" = "Ширина символа" • Mx / Dx,

"Высота символа новая" = "Высота символа" • My / Dy.

InstallUserFont (font:string) - процедура.

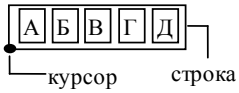
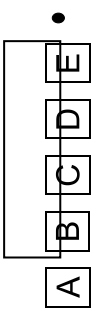
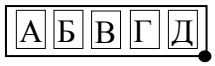
Устанавливает шрифт символов, определенный пользователем. Все шрифты символов хранятся в файлах с расширением *.CHR* и по мере

необходимости пользователь может устанавливать тот или иной шрифтовой фонт. В переменной *font* должно представлено имя файла с расширением .*CHR*, который необходимо использовать, например,

font := InstallUserFont ('RUSS.CHR');

SetTextStyle (font,vertdir,2);

SetTextJustify (hor,vert:word) - процедура. Устанавливает параметры расположения строки относительно курсора.

Значения параметров			
Hor	Расположение строки	Vert	Расположение строки
LeftText = 0		BottomText = 0	
CenterText = 1		CenterText = 1	
RightText = 2		TopText = 2	

Билет № 12

1. лекция ***Создание и уничтожение динамических переменных***

Общий порядок использования динамических переменных следующий:

- создание ДП,
- применение (использование в программе),
- уничтожение ДП (освобождение ОП).

Для создания динамических переменных используется процедура:

New(P1);

P1 - имя ссылочной переменной. После выполнения этой процедуры переменной P1, будет присвоено значения адреса динамической переменной, для которой выделено место в “куче” в соответствии с базовым типом. Результат выполнения этой процедуры можно представить в виде следующей модели (рис. 10.3).

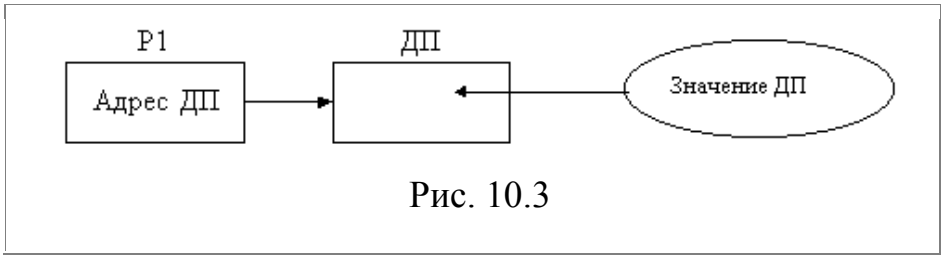


Рис. 10.3

После создания динамической переменной с ней можно выполнять необходимые действия, например:

Для освобождения памяти от динамической переменной используется процедура:

Dispose(P1);

После того как переменная становится не нужной следует освободить место в ОП. Перед созданием динамических переменных следует проверить **наличие свободного места** (в куче). Состояние динамической памяти можно представить в виде графической модели (рис. 10.4)

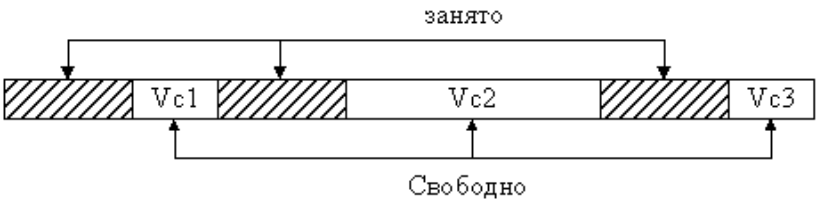


Рис. 10.4

Общий объем свободной динамической памяти складывается из суммы свободных участков:

$$Vc = \sum_{i=1}^{n} Vc,i$$

В свободной области можно найти свободный участок максимальной длины:

Vmax = max(Vc,i), i=1,2, ..., n

В языке Турбо Паскаль имеются функции, которые определяют общий объем свободной динамической памяти и объем наибольшего свободного участка: **Memavail (Memory available); MaxAvail**. Перед созданием динамической переменной рекомендуется использовать **MaxAvail** для проверки достаточного объема памяти для создаваемой переменной. Требуемый объем памяти может быть определен с помощью функции SizeOf(P1), которая определяет длину аргумента в байтах. Аргумент P1 - либо базовый тип, либо имя ссылочной переменной. Проверка наличия свободной ОП для динамической переменной может быть выполнена следующим образом.

2. лекция **Графические процедуры и функции для формирования экрана, окна, страницы:**

ClearDevice - процедура.

Гасит экран и устанавливает параметры графического драйвера на стандартное значение.

SetViewPort (x1,y1,x2,y2:integer;clip:boolean) - процедура.

Создает окно с левым верхним углом (x1,y1) и правым нижним углом (x2,y2), устанавливая при этом курсор в верхний левый угол окна. Параметр ***clip*** устанавливает, пересекают ли символы границы окна.

ClearViewPort - процедура.

Содержимое окна гасится и окрашивается в цвет, заданный фоновым.

SetVisualPage (p:word) - процедура.

Некоторые графические адаптеры (например, *EGA, VGA, Hercules*) поддерживают несколько страниц. Процедура ***SetVisualPage*** устанавливает, какая страница видна.

SetActivePage (p:word) - процедура.

Устанавливает, на какой странице строится изображение.

GetMaxX - функция.

Возвращает максимально возможную координату X для установленного драйвера и режима.

GetMaxY - функция.

Возвращает максимально возможную координату Y для установленного драйвера и режима.

Билет № 13

1. лекция практика *Исходные данные:*

- границы интервала, содержащего корень уравнения (A,B),
- R - абсолютная погрешность результата (корня уравнения),
- E - допустимое значение функции, близкое к нулю,
- Nmax - максимально допустимое число итераций.

Результаты:

- X_n, X_{n-1} - два последних приближенных значения корня,
- $f(x_n)$, $f(x_{n-1})$ - два последних значения функции,

$$- L = \begin{cases} -1, & \text{если корень не найден} \\ 0, & \text{если выполнен останов по условию } |B - A| \le R \\ 1, & \text{если выполнен останов по условию } |f(x)| \le E \end{cases}$$

Основные шаги алгоритма:

а) вычислить f(A), f(B);

б) если f(A) • f(B) > 0, то корень отсутствует, L = -1, конец работы,

в) вычислить значение абсциссы точки пересечения хорды с осью 0X

x1 = A - f(A) • (B-A)/[f(B) - f(A)];

г) если f(A) • f(X1) > 0, то A = x1, f(A)=f(x1), иначе B=x1, f(B)=f(x1);

д) если f(x1) < E, то L =1, конец работы;

е) если |B - A|<=R, то L=0, конец работы, иначе перейти к пункту в).

В конце работы выполнить вывод результатов.

Необходимо также считать число итераций n и сравнивать с N_{max}; если n >= N_{max}, то закончить работу.

2. ***Управление цветом в графическом режиме***

В графическом режиме для представления пиксела может быть использовано различное количество видеопамяти. Пиксел может быть представлен 1, 2, 4 или 8-ю битами. Значения, которые могут быть представлены в этих битах, называются **значениями пиксела.** Значение пиксела определяет цвет, которым пиксел высвечивается на экране дисплея. Следовательно, количество битов, используемых для представления пиксела, определяет количество различных цветов, которыми может быть отрисован пиксел. Однобитовое представление определяет два цвета, двухбитовое 4 и т.д.

Каждый цвет соответствует уникальному значению или номеру в множестве значений пиксела. Такое нумерное представление цветов называется **палитрой цветов.** Первоначально палитра (т.е. соответствие номера цвету) задана по умолчанию. Однако в Паскале есть возможность переустановки палитры.

Установка той или иной палитры важна в первую очередь для цветного дисплея. Палитрой мы как бы выделяет из множества всех цветов только те цвета, которыми будем пользоваться при рисовании. Для монохромного дисплея определено только два цвета: черный и белый (или зеленый). Поэтому устанавливать или переустанавливать палитру вам не придется. Как вы увидите ниже, все функции и процедуры, работающие с цветом, используют цвета из текущей палитры выбирая их по номеру. Рассмотрим эти процедуры.

Для установки цвета рисунка используется процедура SetColor.

Формат: SetColor 0(<цвет>);

где **<цвет>** - это номер цвета из текущей палитры. После того, как вы установили с помощью этой процедуры цвет рисунка, все процедуры отрисовки будут рисовать графические объекты именно этим цветом (пока вы его не переопределите).

Таким образом, параметр процедуры SetColor является индексом в палитре цветов. Нулевое значение параметра означает, что выбран цвет, совпадающий с цветом фона (например, при стирании линии). Цвет по умолчанию является цветом с наибольшим индексом в текущей палитре. При работе с монохромным дисплеем использовать процедуру установки цвета рисунка нет необходимости, если вы не хотите стереть линию или часть рисунка.

Для установки фона в Паскале используется процедура SetBkColor.

Формат: SetBkColor 0(<цвет>);

где **<цвет>** - номер цвета из текущей палитры.

Если в текстовом режиме мы говорили о фоне символа, то в графическом режиме пиксел не имеет фона. Когда мы устанавливаем фон в графическом режиме, то речь идет о фоне закраски, которая используется при очистке экрана или "окна" или закраске некоторых графических объектов.

SetFillStyle (muster,f:word) - процедура. Устанавливает образец для заполнения площадки. Для muster существуют следующие константы:

```
const
  emptyfill = 0;          { Заполнение цветом фона }
  solidfill = 1;          { Сплошное заполнение }
  linefill = 2 ;          { ----------}
  Ltslashfill = 3;        {\\\\}
  slashfill = 4;          {\\\\, линии утолщенные}
  bkslashfill = 5;        {////,линии утолщенные }
  Ltbkslashfill = 6;      {////}
  hatchfill = 7;          { легкая штриховка }
  xhatchfill = 8;         { частая штриховка, пересекающаяся }
  interleavefill = 9;     { чередующиеся линии }
  widedotill = 10;        { далеко отстоящие одна от другой точки }
  closedotfill = 11;      { жирные точки }
```

Переменная f определяет цвет заполнения.

SetFillPattern (muster:fillpatterntype;f:word) - процедура.

Устанавливает образец заполнения и цвет для одной из вызываемых процедур ***FillPoly***, ***FloodFill***, ***Bar***, ***Bar3d***, ***Pieclice***. Для этого в модуле *Graph* имеется тип

```
type
  fillpatterntype = array[1..8] of byte;
```

При этом каждому биту этого массива соответствует один элемент изображения (пиксел). Каждый байт определяет восемь расположенных рядом точек. Восемь байтов устанавливаются один за другим. Переменная f устанавливает цвет заполнения.

FloodFill (x,y,rand:word) - процедура. Если точка (x,y) находится внутри ограниченной некоторыми линиями цвета rand поверхности, она закрашивается в соответствии с образцом, определенными процедурами *SetFillStyle* или *SetFillPattern*.

Pieslice (x,y:integer;w1,w2,r:word) - процедура.

Из центра (x,y) вычерчивает дугу радиусом r от угла w1 до угла w2. Угол задается в градусах. Затем такая "вырезка" заполняется согласно установленному с помощью *SetFillStyle* или *SetFillPattern* образцу.

Ellipse (x,y:integer;a1,a2,rX,rY:word) - процедура. Вычерчивает эллиптическую дугу радиусами rX, rY от угла a1 до угла a2, координаты точки (x,y) задают центр эллипса.

Графические процедуры и функции для работы с цветом и палитрой:

SetBkColor (f:word) - процедура.

Переменная f устанавливает фоновый цвет.

SetColor (f:word) - процедура.

Переменная *f* устанавливает текущий цвет.

3. стр 97 семакин и др основы программирования

билет № 14

1. лекция Удаление элемента динамической цепочки

Операция выполняется с помощью двух операторов (после успешного поиска): в начале в поле ссылки элемента, предшествующему удаленному записывается ссылка на элемент, следующий за удаляемым; затем освобождается место в ОП; на рис. 10.8 первое действие помечено цифрой 1, а второе цифрой 2.

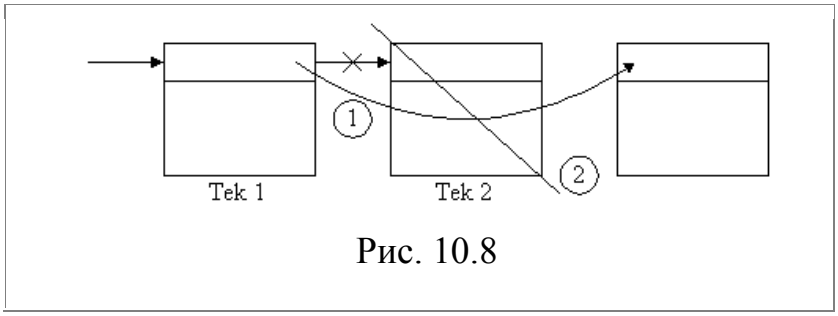


Рис. 10.8

Соответствующие операторы:

Дополнение динамической цепочки

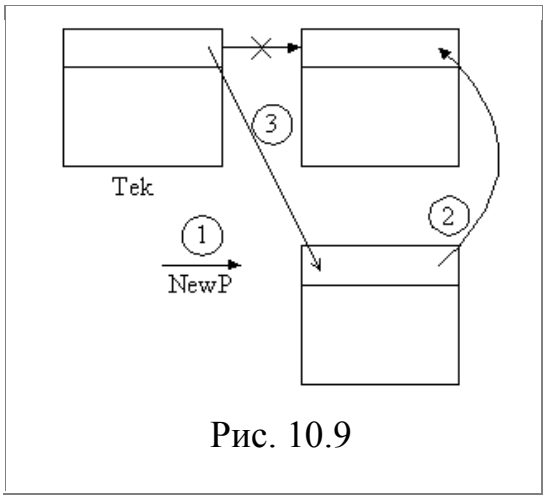


Рис. 10.9

Для дополнения после заданного элемента вначале создается новая динамическая переменная, на которую указывает NewP, затем осуществляется изменение ссылок: из новой динамической переменной на последующий элемент, из текущей на новую (рис 10.9).

Соответствующие операторы:

Удаление элемента из начала цепочки:

```
.....
Tek := Beg;          {Сохранение указателя на первый элемент}
Beg:= Beg^.P;        {Изменение указателя начала цепочки: "второй становится
 первым"}
Dispose(Tek);        {Удаление "старого первого"}
.....
```

Удаление элемента из конца цепочки:

```
.....
Tek:=Beg;
while Tek^.P <> Nil do    {"Продвижение" до конца}
begin                     {цепочки; Tek1 указывает на элемент,}
Tek1:=Tek;           {предшествующий элементу Tek^}
Tek:=Tek^.P
```

end;

Tek1^.P:=Nil; {Замена ссылки предпоследнего элемента на NIL;

"предпоследний становится последним"}

Dispose(Tek); {Освобождение памяти от последнего элемента}

.....

Исключение элемента, следующего за заданным (на заданный элемент указывает *Tek*):

.....

NewP:= Tek^.P; {Сохранение указателя на удаляемый элемент}

Tek^.P:= NewP^.P; {Замена ссылки заданного элемента на ссылку

на элемент, следующий за удаляемым}

Dispose(NewP); {Освобождение памяти от удаляемого элемента}

.....

В конце работы программы следует **освободить память от динамических переменных**, например, следующим образом:

.....

Tek:=Beg; {Присвоение начального адреса}

while Tek <> NIL do

begin

Tek:=Tek^.P; {Присвоение следующего адреса}

dispose(Beg); {Удаление элемента, предшествующего текущему (на который указывает *Tek*)}

Beg:=Tek {Сохранение указателя на предыдущий элемент}

end

..... Для удаления динамического объекта из кучи используется особый метод – ***деструктор***, описываемый с помощью

зарезервированного слова **DESTRUCTOR.** В этом методе можно предусмотреть все действия, связанные с ликвидацией динамического объекта (т.е. переменной объектного типа, размещенной в динамической памяти), например, осуществить. нужную коррекцию списка динамических объектов. Обращение к деструктору указывается вторым параметром при вызове процедуры *DISPOSE,* например:

```
Type
  TLine = object(Point)
      Constructor Init;
      Destructor Done;
  end;
. . .
New(PLine, Init); {Размещение динамического объекта}
...
Dispose(Pline, Done);{Удаление динамического объекта}
```

При необходимости деструктор, как и любой другой метод объекта (кроме конструктора!) , можно объявить виртуальным

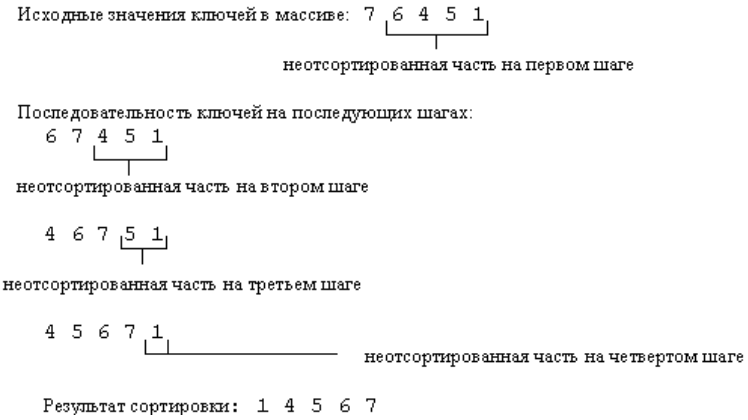
2. лекция ***Сортировка вставками***

Массив рассматривается состоящим из двух частей: отсортированной и не отсортированной.

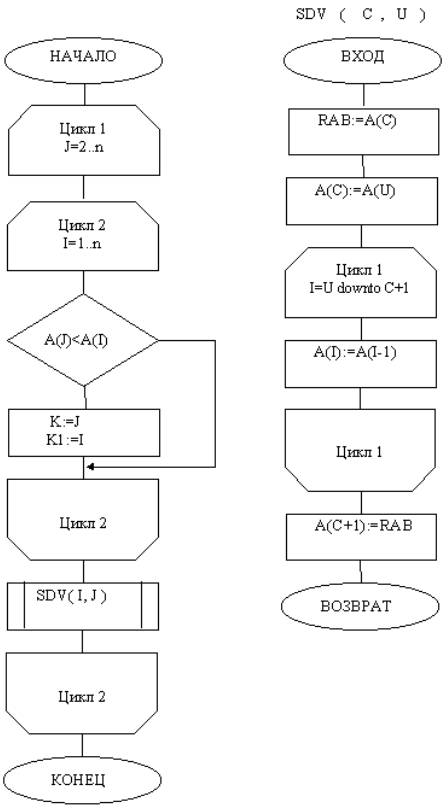
На первом шаге отсортированная часть содержит первый по порядку элемент (запись) массива, не отсортированная - остальные. Шаг в данном случае - это последовательность операций по добавлению (вставке) одного элемента (первого из не отсортированной части) в то место отсортированной части, которое после этого будет удовлетворять условию ▶.На втором шаге первый элемент из не отсортированной части сравнивается поочередно с элементами отсортированной части и устанавливается на место, удовлетворяющее условию ▶; при этом может потребоваться сдвиг всех расположенных ниже места вставки элементов

на одну позицию вниз; после этого отсортированная часть содержит два элемента и т.д.

Пример:



Для реализации этого алгоритма необходима процедура сдвига элементов массива вниз (вправо) для освобождения места для вставки элемента данных (записи). Рис. 11. 4



Сортировка вставками.

Это изящный и простой для понимания метод. Вот в чем его суть: создается новый массив, в который мы последовательно вставляем элементы из исходного массива так, чтобы новый массив был упорядоченным. Вставка происходит следующим образом: в конце нового массива выделяется свободная ячейка, далее анализируется элемент, стоящий перед пустой ячейкой (если, конечно, пустая ячейка не стоит на первом месте), и если этот элемент больше вставляемого, то подвигаем элемент в свободную ячейку (при этом на том месте, где он стоял, образуется пустая ячейка) и сравниваем следующий элемент. Так мы прейдем к ситуации, когда элемент перед пустой ячейкой меньше вставляемого, или пустая ячейка стоит в начале массива. Помещаем вставляемый элемент в пустую ячейку. Таким образом, по очереди вставляем все элементы исходного массива. Очевидно, что если до вставки элемента массив был упорядочен, то после вставки перед вставленным элементом расположены все элементы, меньшие его, а после - большие. Так как порядок элементов в новом массиве не меняется, то сформированный массив будет упорядоченным после каждой вставки. А значит, после последней вставки мы получим упорядоченный исходный массив. Вот как такой алгоритм можно реализовать на языке программирования *Pascal*:

```
Program InsertionSort;
Var A,B   : array[1..1000] of integer;
   N,i,j  : integer;
Begin
{Определение размера массива A (N) и его заполнение}
 …
{сортировка данных}
 for i:=1 to N do
 begin
```

```
 j:=i;
 while (j>1) and (B[j-1]>A[i]) do
  begin
   B[j]:=B[j-1];
   j:=j-1;
  end;
 B[j]:=A[i];
end;
{Вывод массива B}
…
End.
```

В принципе, данную сортировку можно реализовать и без дополнительного массива *B*, если сортировать массив *A* сразу при считывании, т. е. осуществлять вставку нового элемента в массив *A*.

Массив рассматривается состоящим из двух частей: отсортированной части и неотсортированной части.

На первом шаге отсортированная часть содержит первый по порядку элемент (запись) массива, неотсортированная - остальные. Шаг в данном случае - это последовательность операций по добавлению (вставке) одного элемента (первого из неотсортированной части) в то место отсортированной части, которое после этого будет удовлетворять условию (1).

На втором шаге первый элемент из неотсортированной части сравнивается поочередно с элементами отсортированной части и устанавливается на место, удовлетворяющее условию (1); при этом может потребоваться

сдвиг всех расположенных ниже места вставки элементов на одну позицию вниз; после этого отсортированная часть содержит два элемента и т.д.

Пример:

Исходные значения ключей в массиве: 7 6 4 5 1, неотсортированная часть на первом шаге -6 4 5 1).

Последовательность ключей на последующих шагах:

6 7 4 5 1

(неотсортированная часть на втором шаге – 4 5 1)

4 6 7 5 1

(неотсортированная часть на третьем шаге - 5 1)

4 5 6 7 1

(неотсортированная часть на третьем шаге - 1)

Результат сортировки: 1 4 5 6 7

Для реализации этого алгоритма необходима процедура сдвига элементов массива вниз (вправо) для освобождения места для вставки элемента данных (записи).

3.практика лекция

билет № 15

1. **<u>Добавление элемента в начало цепочки:</u>**

.....

New(NewP); {Создание новой динамической переменной}

NewP^.P:= Beg; {Запись в поле ссылки нового элемента адреса элемента, который был первым}

Beg:= NewP; {Установка указателя на новый первый элемент}

.....

<u>Включение нового элемента в цепочку данных после заданного</u>

(на заданный элемент указывает *Tek*):

New(NewP); {Создание новой динамической переменной}

NewP^.P:= Tek^.P; {Запись в поле ссылки нового элемента адреса

элемента, который будет следовать за ним}

Tek^.P:= NewP; {Замена ссылки заданного элемента на ссылку на новый, включаемый в цепочку элемент}

Типовыми операциями с динамическими списками являются:

- создание и первоначальное заполнение динамического списка (ввод данных),
- вывод значений элементов динамического списка,
- поиск заданного элемента списка,
- дополнение после заданного элемента списка,
- удаление заданного элемента списка.

Все операции следует рассматривать в начале на графических моделях с целью освоения происходящих процессов. Рассматриваемые ниже примеры используют динамические переменные типа “запись”, в которых имеются три поля:

- Д1 и Д2 – поля обрабатываемых данных (тип string [15]); например, Д1 – для записи фамилий сотрудников,
- Д2 – для записи названий профессий;
- Р – поле ссылки на следующий элемент динамической цепочки.

Ввод, создание и заполнение динамического списка

В программе в разделе Type определяется во-первых ссылочный тип. Затем необходимые переменные (см. п. 10.5).

```
Type
Ptr=^Zap; {Ptr - Ссылочный тип; для указания динамических переменных типа Zap;
определения типа Zap даётся следующим оператором}
Zap=record
D1,D2: string[15];
P: Ptr {поле ссылки на следующий элемент динамической цепочки}
end;
var
Bag,Tek,tek1,NewP: Prt; {вспомогательная переменная для ввода данных}
S: string[15];
flag: boolean; {вспомогательная переменная для использования в операторе цикле}
```

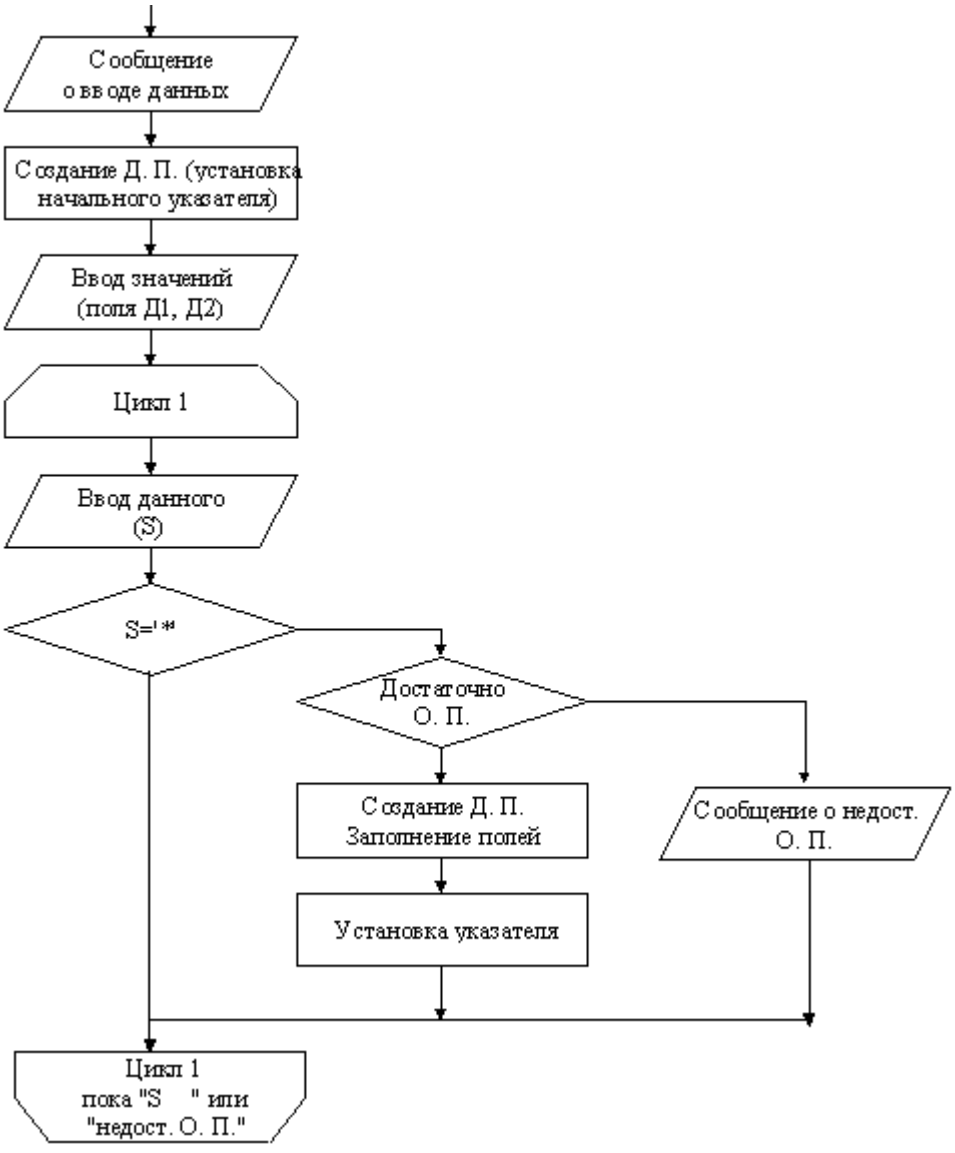


Рис. 10.7

2. Для отрисовки линий в Паскале используются три различных процедуры. Рассмотрим их более подробно.

Процедура Lin.

Формат: Lin 0(x1, y1, x2, y2);

Эта процедура рисует линию из точки (x1,y1) в точку (x2,y2). Процедура не меняет положения текущего указателя.

Процедура LineRel.

Формат: LineRel 0(Dx, Dy);

Процедура проводит прямую линию в точку, заданную относительным расстоянием от текущего указателя.

Процедура LineTo.

Формат: LineTo 0(x, y);

Рисует прямую линию из точки, в которой находится текущий графический указатель, в точку (x,y). Процедура перемещает текущий указатель в точку (x,y).

Линии, отрисованные любой из этих процедур, рисуются цветом, заданным с помощью процедуры SetColor. Кроме того, при отрисовке линий может устанавливаться ее толщина и тип. Для этого служит процедура SetLineStyle.

Формат: SetLineStyle 0(<тип линии>, <образец>, <толщина>);

Все параметры целые и могут принимать следующие значения:

Тип линии: 0 - непрерывная линия; 1 - линия из точек; 2 - линия из точек и тире; 4 - тип линии, определяемый пользователем. Если тип линии - 4, то в качестве типа линии выбирается заданный во втором параметре образец.

Толщина линии: 1 - нормальная толщина; 3 - жирная линия.

В качестве примера использования процедур отрисовки линий можно привести программу, рисующую прямоугольный треугольник в заданной точке (100,100). Катеты треугольника равны 50 и 70 пикселам.

3.

Билет № 16

1. Pascal. Структуры данных в Паскале

Язык Паскаль относится к парадигме структурного программирования. Это значит, что понятие «структура» входит в программу не только на уровне общего ее построения, но и предусматривает структурирование самой логики и оперируемых данных. Прежде всего, данный принцип нашел свое применение в типизации языка, т.е. наделении его строгой системой предопределенных типов, комбинируя которые программист мог создавать свои собственные типы, причем уже не только простые (атомарные), но и представляющие собой сложные конструкции – структуры данных.

Это было очень важно, поскольку моделирование всевозможных реальных процессов становилось одной из основных областей применения ЭВМ. Моделирование приводило к абстракции, т.е. к упрощенному (или обобщенному) представлению объектов, их свойств и отношений в зависимости от поставленной задачи. Некоторые такие абстрактные объекты из-за своих полезных качеств стали необычайно «популярны». Они получили точную спецификацию и с тех пор стали называться абстрактными типами данных (или АТД). Однако абстракция на то и абстракция, чтобы абстрагироваться от многих несущественных факторов, в том числе и от того, как этот абстрактный объект представить в ЭВМ. Этим собственно и занялись языки программирования, которые, по сути, являются неким переходником между идеями человека и возможностями машины. И вот здесь как раз и пригодились вышеупомянутые структуры данных.

Структурные типы в Паскале

Массив

Массив – это одно- или многомерная таблица данных одного типа. Каждая ячейка таблицы имеет свой индекс (в одномерном случае) или

набор индексов (в многомерном). Массив называют структурой данных со случайным доступом, поскольку к любому элементу массива можно обратиться, просто указав его индексы, т.е. все элементы одинаково доступны в любой момент времени. Массив определяется, прежде всего, общим типом его элементов и их количеством. Количество элементов массива, в свою очередь, определяется количеством индексов и диапазоном их изменения. При описании переменной типа массив под каждый элемент выделяется фиксированный объем памяти, что и является главным недостатком этой структуры, во-первых, потому что в некоторых системах программирования Паскаль под переменную выделяется ограниченное количество памяти, что не позволяет использовать массивы больших размеров, во-вторых, потому что такая организация не дает возможности изменять размер массива, что приводит к определенного рода неудобствам, когда приходится впустую копировать большие объемы памяти, например, при передаче параметра-массива подпрограмме.

Записи

Запись – связанная структура, состоящая из нескольких элементов (полей) разных (можно и одинаковых) типов. По сути, запись очень похожа на одномерный массив, но с элементами разных типов, кроме того, доступ к конкретному полю записи осуществляется уже не через индекс, а указанием идентификатора (т.е. имени) этого поля. Более того, в Паскале существует возможность менять тип конкретного поля в зависимости от ситуации. Такие структуры называются записями с вариантами. Правда, в любой записи может быть только одна вариантная часть, и, если она есть, ее описание должно располагаться за всеми фиксированными частями. Замечательная особенность вариантной части состоит в том, что все варианты как бы «накладываются» друг на друга, т.е. каждому из них выделяется одна и та же область памяти. Это открывает дополнительные возможности преобразования типов.

Файлы

Файл – динамическая структура данных, размер которой может меняться в процессе выполнения над ним каких-либо действий (он может быть равен нулю, что соответствует пустому файлу). Вообще-то, понятие файла используют и как абстракцию данных, хранящихся в памяти, что позволяет единообразно определить их общие характеристики и операции над ними. Однако свойства физического файла почти полностью совпадают со свойствами АТД файла, из-за чего абстракция в данном случае теряет весь смысл. Файл – структура, состоящая из последовательности компонент одного типа. Свойства последовательности определяет последовательный доступ к элементам, т.е. в каждый момент времени может быть доступен только один элемент файла. Основная операция над файлами – конкатенация (или слияние) файлов – позволяет создавать файлы неограниченной длины. Существует несколько видов файлов в Паскале. Текстовые файлы трактуются как последовательности строк переменной длины. В конце каждой строки ставится специальный признак конца строки EOLN. Типизированные файлы – последовательности компонент определенного типа. Длина любого элемента типизированного файла строго постоянна, что дает возможность организовать прямой доступ к каждому компоненту. В некоторых СП Паскаля это действие реализуется процедурой SEEK. Также, в отличие от текстовых файлов, здесь не работают процедуры READLN и WRITELN, поскольку нет нужды в переводе строки. Нетипизированные файлы объявляются предложением FILE и отличаются тем, что для них не указан тип компонент. Такой подход делает нетипизированные файловые переменные совместимыми с файлами любых типов, а также позволяет организовать высокоскоростной обмен данными между оперативной и внешней памятью. При инициации таких файлов процедурами RESET или REWRITE нужно указывать их длину в байтах, причем для достижения

максимальной скорости обмена лучше, если эта длина кратна размеру физического сектора на внешней памяти.

Динамические структуры

Хотя динамика и не является отличительной чертой языка Паскаль, все же существует возможность создавать динамические объекты и оперировать с ними. Динамический объект представляет собой область памяти без идентификатора. Для обращения к этой области заводится специальная ссылочная переменная, которая описывается в программе. Элементами множества значений ссылочного типа являются значения адресов оперативной памяти. Чаще всего динамические структуры состоят из объектов, являющихся записями из нескольких полей, одна или несколько из которых являются ссылками на такой же объект. Таким образом можно составлять цепочки или более сложные структуры ссылающихся друг на друга объектов. Размер таких структур ограничивается только объемом свободной памяти и может легко меняться при удалении и создании объектов, что и определило основную область их применения – моделирование нелинейных последовательных структур данных (например, деревьев). Однако, несмотря на кажущееся отсутствие недостатков, динамические структуры тоже имеют свои минусы. Главный из них – отсутствие наглядности программы – вызывает трудности при создании особо крупных структур.

Представление абстрактных типов данных (АТД) в Паскале

Моделирование абстрактных типов данных на конкретных структурах данных Паскаля не является однозначным. Практически любой АТД может быть эффективным образом представлен как на массиве, так и на файле или на динамической структуре, но чаще всего конкретное представление диктуется особенностью доступа к элементам рассматриваемого абстрактного типа.

Стек

Стек – структура данных, представляющая собой последовательность

элементов. Добавление и удаление элементов происходит только с одного конца последовательности, т.е. при изменении структуры предыдущая последовательность остается неизменной. Это значит, что по идее идеальной структурой представления для стека должен быть файл. Действительно, добавление элементов происходит только в конец файла, однако при удалении последнего элемента придется два раза переписывать весь файл, причем с подсчетом элементов. Это главный недостаток файла – для удаления компоненты требуется слишком много действий. С другой стороны время удаления элементов из файла не зависит ни от положения этого элемента в файле, ни от количества удаляемых элементов – это плюс! Стек на массиве позволяет уравнять время выполнения действий по добавлению и удалению элемента. Нужно всего лишь хранить в отдельной переменной длину последовательности и увеличивать ее или уменьшать соответственно при добавлении и удалении. Единственный недостаток такого представления – недостаток самого массива в Паскале – ограниченный размер, приводящий к переполнению стека. Использование динамических структур полностью решает эту проблему. В этом случае при добавлении и удалении элемента модифицируется только ссылка на начало цепочки динамических объектов, что снижает до минимума количество используемых для действия операторов. При этом размер стека почти неограничен.

Очередь, дек

Очередь – структура данных, как и стек представляющая собой последовательность элементов. Добавление элементов происходит с одной стороны последовательности, удаление – с другой. Дек – двусторонняя очередь, т.е. и добавление и удаление осуществляется с обеих сторон. Представление на файле этих АТД имеет такой же характер и «преимущества» как и у стека, т.е. переписывание файла происходит во всех случаях кроме добавления компоненты в его конец. Представление очередей и деков на массиве отличается тем, что последовательность

элементов может «перемещаться» по массиву, следовательно, чтобы последовательность не выскочила за границы, нужно границы соединить вместе (зациклить). При этом индекс элементов (и первого и последнего) вычисляется как (x mod n), где n – размер массива. Проблемы все те же – возможность переполнения. Реализация на динамике использует ту же особенность (цикличность) и лучше всего представляется в виде кольцевого двунаправленного списка, о котором речь пойдет ниже.

Линейный список

Линейный список – одна из тех структур данных, которая если не приравнивается, то хотя бы ассоциируется с динамическими структурами. Он представляет собой упорядоченное множество (возможно пустое), в котором добавление и удаление элементов может происходить на любом месте списка. Элементы списков чаще всего представляют в виде записи, состоящей из поля информационной части и одного или двух полей-ссылок на другие узлы. Списки имеют последовательную структуру, т.е. для того чтобы перейти к какому-либо элементу, нужно пройти все элементы, предшествующие искомому. Причем если каждый элемент имеет ссылку только на следующий за ним элемент, то каждый такой проход нужно начинать с «головы» списка. Этот недостаток устраняется либо зацикливанием списка (ссылка последнего элемента указывает на первый) – в этом случае вообще теряются понятия начала и конца списка – либо использованием второй ссылки (на предыдущий элемент), чтобы можно было перемещаться в обе стороны. Представление списка на массиве является простым моделированием на массиве оперативной памяти. Элементом списка опять будет запись, состоящая из поля информационной части и поля, хранящего индекс следующего элемента в массиве (аналог ссылки). Для того чтобы смоделировать «кучу» свободной памяти, из которой берется память под новые объекты, создается список свободных элементов.

Линейные списки Pascal – курсовая работа

Деревья

Дерево – множество, состоящее из элемента, называемого корнем, и конечного (возможно пустого) множества деревьев, называемых поддеревьями данного дерева. Дерево, так же как и список, реализуется прежде всего на динамических структурах, т.е. узел дерева содержит два поля-ссылки – на левое и правое поддерево для двоичного дерева и на брата и старшего сына для дерева общего вида. Дерево уже не является линейной структурой, поэтому представление деревьев на последовательной памяти (файлах) представляет собой определенную проблему. Однако все данные хранятся во внешней памяти в виде файлов, поэтому решение этой проблемы необходимо. Тем не менее, сразу стоит оговориться, что это представление не будет в полной мере реализацией АТД дерева на структуре данных файле, поскольку требуется лишь создать обратимое отображение дерева в файл – никаких операций с файлом-деревом совершаться не будет. Создадим новый тип узла дерева (он станет компонентом файла) – запись информационного поля и поля, хранящего число поддеревьев данного узла. В файл будем последовательно записывать каждый узел так, чтобы поддеревья записывались после своего корня. Обратная операция производится с использованием рекурсии. Читаем из файла узел дерева, создаем его и ссылки на его поддеревья (столько, сколько указано в соответствующем поле). Затем для всех указанных поддеревьев повторяем ту же процедуру.

Дерево Pascal – лабораторная работа

Множества и деревья

Бинарное дерево называется упорядоченным, если для каждого его узла, имеющего поддеревья, корень больше левого поддерева и меньше правого. Используя упорядоченные бинарные деревья, можно легко реализовать математическое множество данных, для которого существует отношение порядка. При добавлении элемента нужно сохранить свойство упорядоченности, поэтому у добавляемого элемента существует

единственное место в дереве (конечно, если такой элемент еще не присутствует в дереве). Проверка наличия элемента в дереве осуществляется тем же способом – переход от узла к узлу происходит по принципу максимального приближения к искомому значению. Если на том месте, где должен находиться искомый элемент, отсутствует поддерево, значит можно утверждать, что такого элемента нет во множестве.

2. ***Сортировка подсчетом***

Идея алгоритма заключается в том, чтобы попарно сравнить значения всех ключей и при этом для каждого ключа K 4i 0 подсчитать количество меньших его ключей C 4i 0; затем каждый i-й элемент массива (запись) устанавливается на место, номер которого равен C 4i 0 + 1. Рис. 11. 5

При реализации алгоритма можно использовать три массива:

- для исходных данных,
- для подсчета результатов сравнения ключей,
- для результата сортировки.

Алгоритм сортировки подсчетом состоит из следующих процедур:

- сравнение значения очередного i-го элемента массива (ключа) с остальными и подсчет C 4i 0,
- запись очередного i-го элемента (записи) в массив результатов в позицию с номером C 4i 0 + 1.

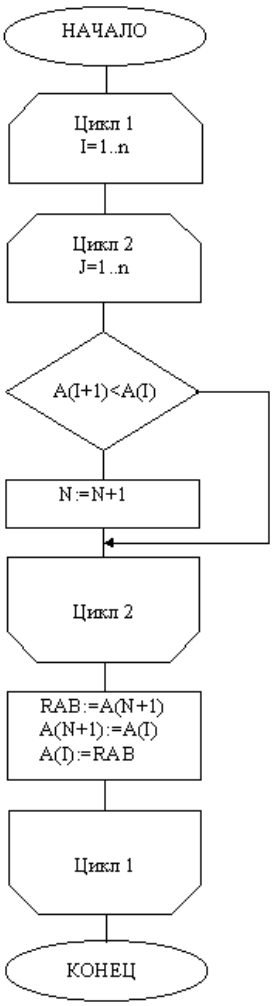
Пример:

Исходный массив: 7 3 5 6 1

Результаты сравнения: 4 1 2 3 0

Результат сортировки: 1 3 5 6 7

< table border="1" width="78%" bgcolor="#FFFFFF"
bordercolor="#FFFFFF" bordercolordark="#FFFFFF"
bordercolorlight="#FFFFFF">



Идея алгоритма заключается в том, чтобы попарно сравнить значения всех ключей массива K и при этом для каждого ключа K_i подсчитать количество меньших его ключей C_i. Затем каждый i-й элемент (запись) исходного массива K устанавливается на место, номер которого равен C_{i+1} в массиве A.

При реализации алгоритма можно использовать три массива:

- для исходных данных;

- для подсчета результатов сравнения ключей;

- для результата сортировки.

Алгоритм сортировки подсчетом состоит из следующих процедур:

- сравнение значения очередного i-го элемента массива (ключа) с остальными и подсчет C_i,

- запись очередного i-го элемента (записи) в массив результатов в

позицию с номером C_{i+1}.

Пример:

Исходный массив: 7 3 5 6 1 (массив *K*)

Результаты сравнения: 4 1 2 3 0 (массив *C*, определяющий позицию элемента массива K_i в массиве *A*)

Результат сортировки: 1 3 5 6 7 (массив *A*)

3. стр 142 немнюгин турбо паскаль практикум

Билет № 17

1. Наряду с задачей сортировки актуальной при работе с данными является поиск данных по заданному признаку. Если данные не упорядочены, то для поиска приходится просматрить весь объем информации. В том случае, если данные упорядочены, для поиска используют один из двух известных алгоритмов поиска: линейный или двоичный поиск. Рассмотрим эти алгоритмы и реализующие их программы.

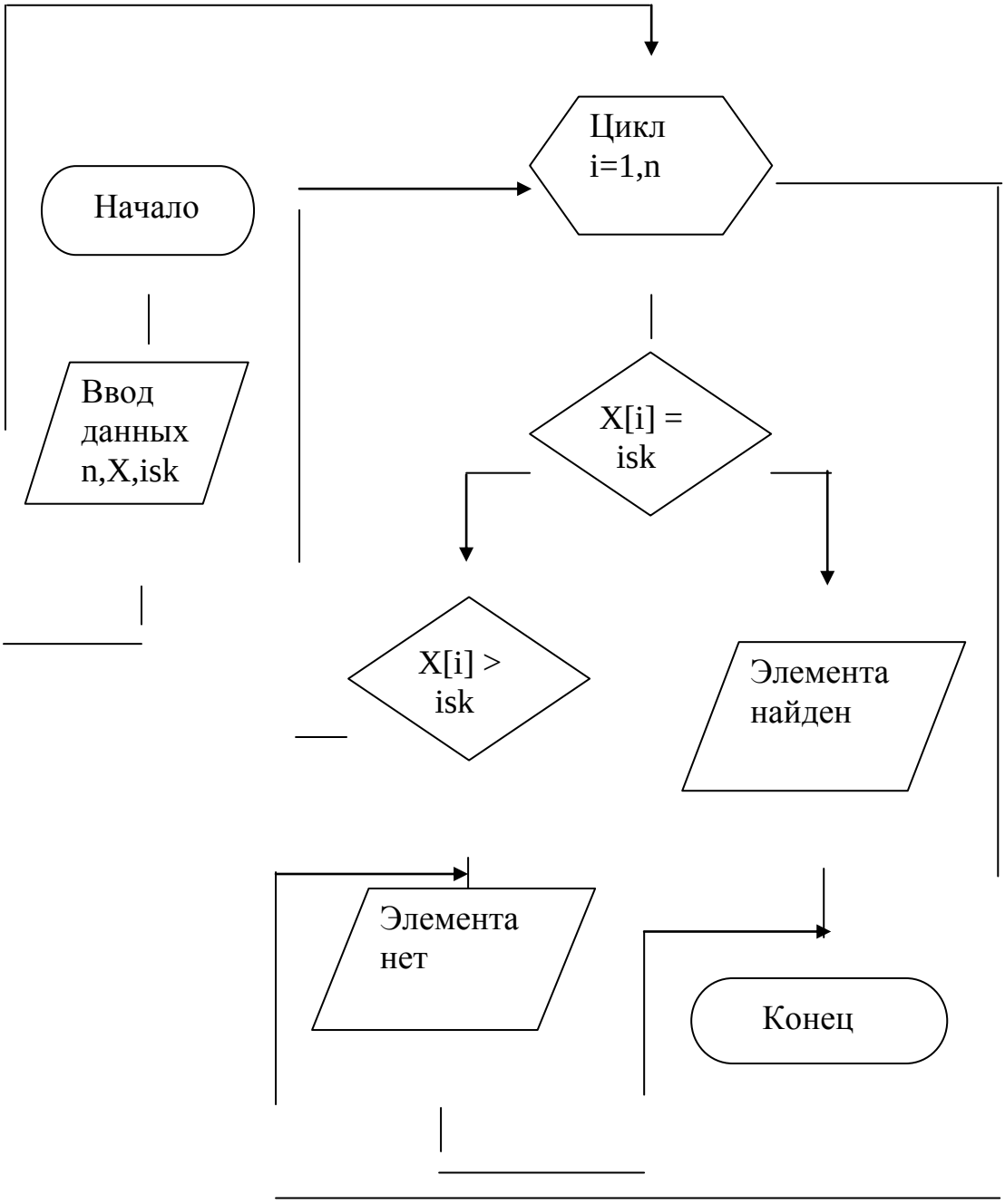
Пусть задан массив X, содержащий n целых чисел, расположенных в порядке возрастания значений. Требуется найти заданное число isk.

Линейный поиск

Линейный поиск заключается в последовательном сравнении элементов упорядоченного массива X с искомым значением isk. Поиск в этом случае заканчивается в двух случаях:

1. Очередной элемент массива X равен искомому значению isk, в этом случае элемент найден.
2. Очередной элемент массива X больше искомого значения, а так как все последующие элементы также больше isk, то искомого элемента не может быть в X.

Схема алгоритма линейного поиска



В худшем случае для поиска заданного элемента алгоритмом линейного поиска приходится проводить n сравнений, т.е. временная сложность алгоритма ~ n.

Текст программы линейного поиска

```
Uses crt;
Var
        X:array[1..10000] of integer;
```

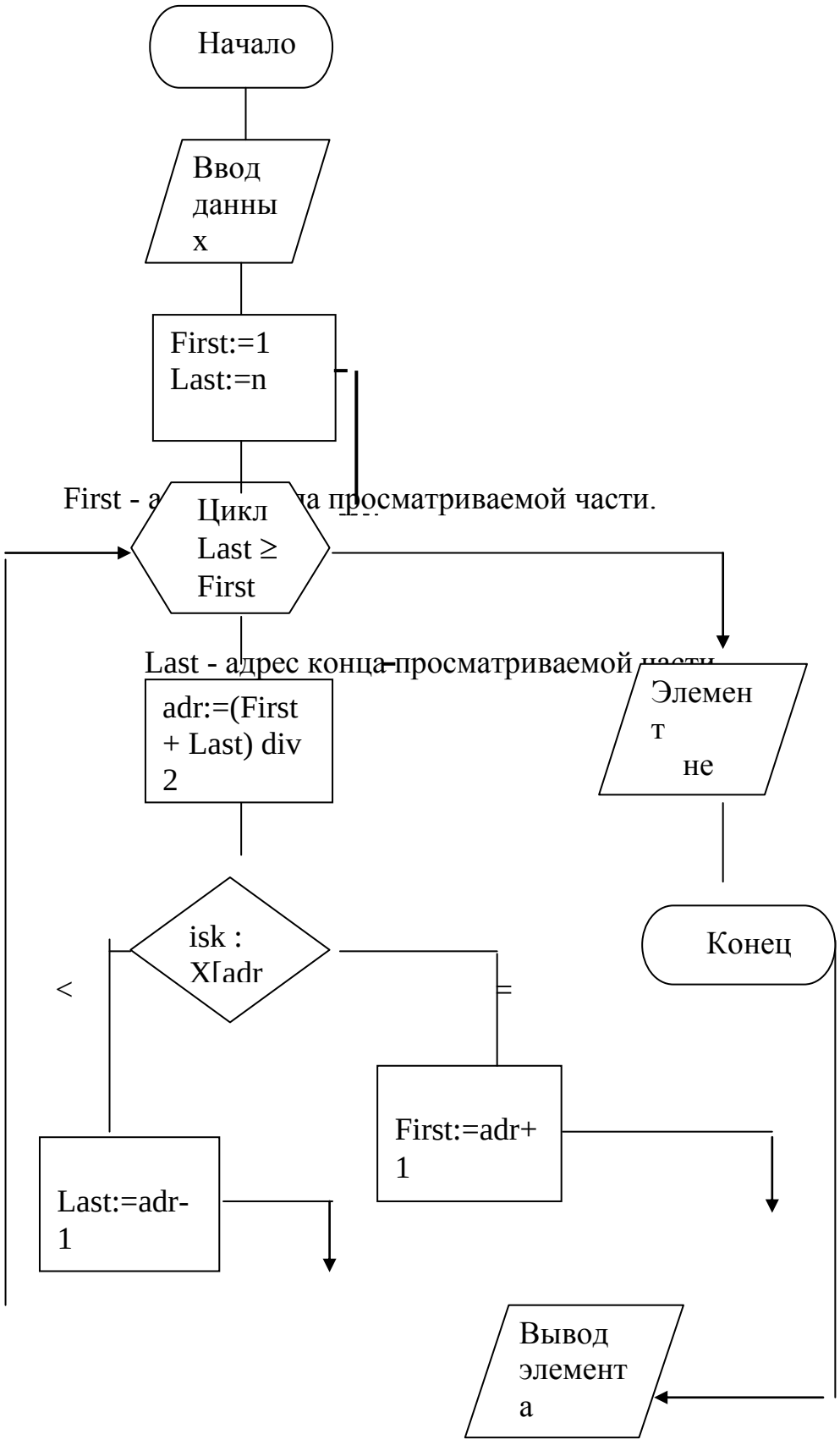
```
        i, n, isk:integer;
Begin
        write('   Введите число элементов в массиве -  n ');
          readln(n);
          write('   Введите элементы массива X ');
          for i:=1 to n do read(X[i]);                    writeln;
          write('    Введите искомое число');        readln(isk);
    for i:=1 to n do
        if X[i]=isk then
                    begin
                            writeln('    Элемент найден по адресу  i = ' ,i );
                            readln;   exit
                                end
        else if X[i]>isk then break;
    writeln('    Заданного элемента нет');
    readln;
End.
```

Двоичный поиск

Двоичный поиск заключается в следующем: вычисляется середина массива и элемент x_i, находящийся по найденному адресу, сравнивается с искомым. Если элементы x_i и isk равны, то поиск закончен. Если x_i < isk, то искомый элемент может находиться в последующей x_i части массива, в противном случае поиск надо осуществлять в предшествующей x_i части массива. Такой подход позволяет сократить число просматриваемых элементов в 2 раза. Далее в выделенной части массива также определяется средний элемент, и он сравнивается с искомым. Подобная процедура повторяется до тех пор, пока ни будет найден заданный элемент или пока не совпадут начальный и конечный адрес просматриваемой, на очередном шаге, части массива. Если эти адреса совпадут и к этому моменту элемент не найден, то его в массиве нет.

Временная сложность алгоритма двоичного поиска ~ $\log_2 n$.

Схема алгоритма двоичного поиска



First - а[illegible]а просматриваемой части.

Last - адрес конца просматриваемой части.

Текст программы

```
Uses crt;
        Var X:array[1..10000] of integer;
            isk,i,n,adr,First,Last:integer;
            Begin
              write('   Введите число элементов в массиве -  n ');
              readln(n);
              write('   Введите элементы массива X ');
             for i:=1 to n do read(X[i]);
             writeln;
             write('    Введите искомое число');
             readln(isk);
             First:=1; Last:=n;
             while First <= Last do begin
               adr:=(First+Last) div 2;
               if isk=X[adr] then
                     begin  writeln('Yes! isk = ',isk,'   adr = ',adr);
                          exit
                     end
               else if isk<X[adr] then Last:=adr-1
                              else First:=adr+1;
                 end;
                 writeln('  Not found');
           end.
```

2. лекция При использовании метода трапеций для вычисления суммы S_1 используются формулы (рис. 2):

$$S1 = 0.5\ h\ \{f(A) + f(B) + 2[f(X_1) + f(X_2) + \ ... + f(X_{n-1})]\}, \qquad (2)$$

где $X_1 = A + h,\ X_1 = X_{i-1} + h.$

3.см тетрадь

билет № 18

1. Модуль типа *UNIT* в Turbo Pascal - это отдельно хранимая и независимо компилируемая программная единица, в отличие от подпрограмм, которые, являясь структурным элементом Pascal-программы, не могут существовать вне ее. Но модуль не является исполняемой программой; константы, типы, переменные, процедуры и функции, входящие в состав модуля типа *UNIT*, используются другими программами.

Все программные ресурсы модуля можно разбить на две части: объекты, прямо предназначенные для использования другими программами или модулями, и объекты рабочего характера. В соответствии с этим модуль, кроме заголовка, имеет две основные части, называемые интерфейсом и реализацией.

В интерфейсной части модуля сосредоточены описания объектов, доступных из других программ; такие объекты называют видимыми вне модуля. В части реализации помещаются рабочие объекты, называемые также невидимыми или скрытыми.

В соответствии с вышеизложенным модуль типа *UNIT* имеет следующую структуру:

Unit** {служебное слово} **Unit_Name** {имя модуля}**;

***Interface** { начало интерфейсной части }*

Описание объектов, видимых из других программных модулей:

***Const** .....* *(определение констант)*

***Type** .....* *(определение типов переменных)*

***Var** .....* *(определение переменных)*

***Procedure** .....* *(только заголовки процедур)*

***Function** .....* *(только заголовки функций)*

***Implementation** { начало части реализации }*

Описание объектов, скрытых от других программных модулей

***Const** .....* *(определение констант)*

.....

Procedure *.....* *(полные описания процедур и*

Function *.....* *функций, включая процедуры*

и функции из интерфейсной секции)

Begin

Операторы инициализации переменных

перед использованием модуля UNIT_NAME

End;

End. *{ Окончание модуля }*

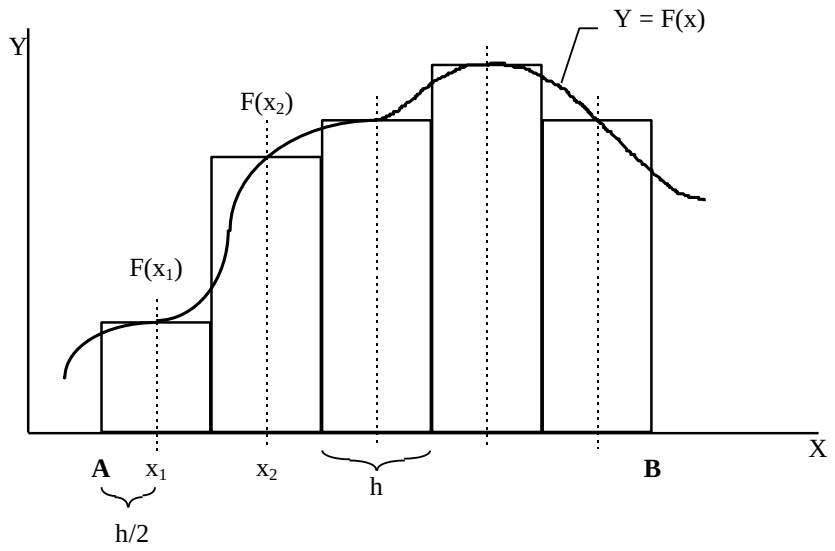
2. лекция практика При использовании метода Симпсона для вычисления суммы S_1 используется формула:

$$S1 = \frac{h}{3}\{f(A) + f(B) + ... + 2[f(A+h) + f(A+3h) + ... + f(A+(n-1)h] + 4[f(A+2h) + f(A+4h) + ... + f(A+(n-2)h)]\}, \quad (3)$$

где n - четное число.

Для вычисления суммы S2 используются те же формулы, только при измененных значениях шага аргумента h и количества интервалов n.

4.



3. лекция

Билет № 19

Справки по процедурам, функциям и управляющим параметрам

TextColor (Код цвета) - процедура установки цвета символов; коды цветов смотри в таблице 8.4.

TextBackGround (Код цвета) - процедура установки цвета фона.

TextAttr - параметр, определяющий коды цветов символа и фона для него; значение TextAttr определяется как (Код цвета фона) x 16 + (Код цвета символа).

Например, для вывода на экран желтых символов на синем фоне следует установить **TextAttr**:=30.

ReadKey: Char - функция чтения кода нажатой клавиши.

ClrScr - процедура очистки экрана.

Таблица 8.4

Код цвета	Цвет	Код цвета	Цвет
0	Черный	8	Серый, темно-серый
1	Синий	9	Ярко синий
2	Зеленый	10	Ярко зеленый
3	Бирюзовый, голубой	11	Ярко голубой
4	Красный	12	Ярко красный
5	Вишневый, сиреневый	13	Ярко сиреневый
6	Коричневый	14	Желтый
7	Белый, светло-серый	15	Белый, ярко-белый

Таблица 8.5 – Примеры кодов клавиш

Название клавиши	Код клавиши (десятичный)	Название клавиши	Код клавиши (десятичный)
Enter	13	F1	59
Home	71	F2	60
Up (Стрелка вверх)	72	F3	61
PgUp	73	F4	62
Left(Стрелка влево)	75	F5	63
Right(Стрелка вправо)	77	F6	64
End	79	F7	65
Down (Стрелка вниз)	80	F8	66
PgDown	81	F9	67
		F10	68

Если символ не имеет графического представления, то в программе на языке Turbo Pascal используется обозначение вида “#K”, где К - код символа.

Например**,** константа #75 соответствует коду клавиши "Стрелка влево" (Left arrow). Этот прием используется для анализа кода нажатой управляющей клавиши.

2. лекция ***5 Решение уравнений по методу касательных (Ньютона)***

Предварительно следует найти первую производную f ′ (x).

Исходные данные и результаты те же, что в предыдущих случаях.

Основные шаги алгоритма:

а) вычислить f(A), f (B);

б) если f(A) • f(B) > 0, то корень отсутствует, L = -1, конец работы;

в) вычислить координату пересечения касательной к графику функции с осью OX: x1= $A-\frac{f(A)}{f'(A)}$ A и значение функции в этой точке f(x1);

д) если f(x1) <= E, то L = 1 и конец работы;

г) если |x1-A| < R, то L = 0, конец работы, иначе A = x1, переход к пункту в).

В конце работы выполнить вывод результатов.

Необходимо также считать число итераций n и сравнивать с N_{max}; если n >= N_{max}, то закончить работу.

3.тетрадь

Билет № 20

1. лекция Вывод значений элементов динамической цепочки

Выполняется с помощью изменение текущего указателя от начального значения (адреса 1-го элемента) до тех пор пока в поле ссылки очередного элемента не встретится константа Nil.

Поиск элемента динамической цепочки

Выполняется путем просмотра списка и сравнения значений полей данных с заданным значением.

Tek - После окончания поиска переменная указывает на искомый элемент, *Tek1* - на предыдущий.

Пример схемы алгоритма ввода данных приведен на рис. 10.7.

Алгоритм состоит из следующих действий:

1. Вывод сообщения о вводе данных; оно должно содержать указания о том, какие данные должен ввести пользователь программы, например: “Введите фамилию и профессию сотрудника”;
2. Создание первой динамической переменной из предполагаемого списка; на нее должна указывать переменная ссылочного типа, например, Beg (см. п. 10. 5▶);
3. Ввод значений полей Д1 и Д2 созданной динамической переменной, например:

   Эта переменная является текущей и на нее следует установить указатель

   Tek:=Beg;
4. Начало цикла для продолжения формирования и заполнения динамического списка;

5. Ввод значения поля Д1 следующей динамической переменной; в данном варианте алгоритма используется вариант ввода данных, в котором признаком окончания ввода является символ “*”; поэтому вводимая с клавиатуры строка символов предварительно присваивается переменной S, затем анализируется, и в зависимости от значения переменной S выбираются последующие действия (если был введен символ ‘*’, это процесс ввода данных заканчивается, иначе - продолжается);
6. Если S=‘*’, это переход к блоку окончания цикла, иначе - к блоку 7;
7. Проверка наличия свободной ОП для следующей динамической переменной: если памяти недостаточно, то выполняется вывод сообщения об этом и переход к блоку окончания цикла, иначе – к блоку 8,
8. Создание следующей динамической переменной и заполнение ее полей;

   Например:
9. Установка указателя из предыдущего элемента на вновь созданный:
10. Блок окончания цикла: проверка условия завершения процесса ввода данных и создания динамической цепочки: если S=‘*’ или недостаточно ОП, то процесс заканчивается, иначе – переход к началу цикла 1.

Формат программы для реализации рассмотренного алгоритма:

Стр 135 Семакин

2. Столбиковая диаграмма (гистограмма)

Гистограмма - графическое изображение оценки плотности распределения случайной величины в диапазоне имеющихся опытных, экспериментальных значений [Mmin, Mmax], при этом диапазон делится на L интервалов, в каждом из которых распределение считается равномерным и равным Fi, (i = 1, 2, ..., L); Fi = ki / n; n - количество значений случайной величины; ki - количество попаданий случайной величины в i - й интервал (рис.13.1).

Другими словами, эта диаграмма показывает, какая доля значений какой-либо величины (цена товара, заработная плата, возраст пациентов и др.) попадает в каждый из заданных интервалов.

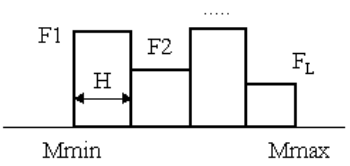


Рис. 13.1

Порядок построения гистограммы можно определить следующим образом:

- подготовлен массив статистических данных M (n чисел); как правило, статистические данные накапливаются в файлах, а для обработки данные могут быть переписаны в массив; примем, что M: array [1...d] of real, d □ □ n;
- задается количество интервалов для построения гистограммы (L: byte); как правило, значение L зависит от количества данных, цели анализа и др.; 5 <=L □ □ 20.
- необходимо построить столбиковую диаграмму (гистограмму) по данным из массива M, *пользуясь следующими правилами:*

1. Ширина основания “столбика” (прямоугольника) равна H = (Mmax - Mmin)/L, где Mmax, Mmin - соответственно максимальное и минимальное число в массиве M,
2. Высота i-го “столбика” (i = 1, 2, ..., L) пропорциональна величине Fi = (k_i / n)• 100 , где k_i - количество чисел из массива M, попадающих в i - й интервал; если не требуется значение k_i в процентах, то Fi не вычисляется, а высота “столбика” будет пропорциональна k_i.

Таким образом, для решения задачи необходимо найти Mmin и Mmax, определить массивы для хранения k_i, Fi (i = 1, 2, ..., L) и границ интервалов Gi (i = 1, 2, ..., L + 1).

4. Алгоритм построения гистограммы

1. Определение массивов M, K, G, F и других переменных.
2. Чтение данных из файла в массив M.
3. Ввод с клавиатуры количества интервалов L.
4. Поиск Mmin и Mmax в массиве M.
5. Вычисление ширины основания прямоугольника H.
6. Обнуление массива K.
7. Заполнение массива G (определение границ интервалов):

```
G[1]: = Mmin;
for i: = 2 to L do
g[i]: = G[i-1] + H;
G[L + 1]: = Mmax;
```

7. Определение значений Ki (i = 1, 2, ..., L):

```
for i: = 1 to n do
for j: = 1 to L do
if (j < L) and (M[i] >= G[j]) and (M[i] < G[j + 1]) then
Inc K[j]
else
{для последнего интервала немного изменяем условие: включаем верхнюю границу интервала}
if (M[i] >= G[j]) and (M[i] <= G[j =1])
then Inc(K[j]);
```

8. Вычисление Fi и поиск Fmax или Kmax; минимальное значение Fi равно нулю.
9. Установка параметров для графического режима (цвет фона, тип линии и др.) и изображения на экране:

XNE, XKE, YNE, YKE - координаты линий прямоугольника, ограничивающих график.

Переход в графический режим.

10. Вычисление масштаба изображения:

```
Mx = (XKE - XNE) / (Mmax - M min);
My = (YKE - YNE) / Fmax (или Kmax)
```

11. Цикл построения гистограммы; “столбики” закрашиваются различными цветами:

```
YE1:=X_Ekr (G[j]);
for j:=1 to L do
begin
{SetFillPattern (Mask[j], Col[j]);- если требуется пользовательская штриховка; Mask - массив кодов пользовательских штриховок; Col - массив кодов цветов для заполнения прямоугольников}
SetFillStyle (Mask1[j], Col1[j]);
{Mask1 - массив кодов для заполнения, включая стандартные; Col1 - то же, что Col; при сплошном заполнении прямоугольника вместо Mask1 подставить константу. Например, SolidFill}
XE1:=X_Ekr(G[j]);
XE2:=X_Ekr(G[j+1]);
YE2:=Y_Ekr(F[j]);
Bar(XE1, YE1, XE2, YE2)
end;
```

12. Вывод осей координат, поясняющих надписей:

```
{Вычерчивание осей координат }
SetColor(White); XE1:=X_Ekr(Mmin);
Line(XE1, YE1, X_Ekr(Mmax + H), YE1);
Line(XE1, YE1, XE1, X_Ekr(Fmax + H));
{Вывод заполнения Mmin под осью OX}
Str(Mmin:6:2,S);
```

```
SetTextStyle(Sansseriffont, Horizdir, NormSize);
OutTextXY(XE1, YE1 + 10, S);
{Вывод других надписей выполняется аналогично}
```

5. Круговая диаграмма

В круговой диаграмме центральные углы секторов Ai пропорциональны значениям Fi (или k_i), гле Fi и k_i определяются как определено выше (рис. 13.2):

$$Ai=(3,\ 6\bullet\ Fi).$$

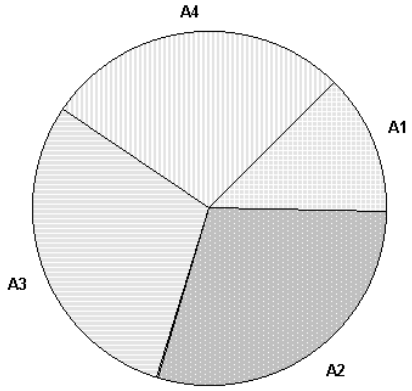


Рис. 13.2

.6. Алгоритм построения круговой диаграммы

Для построения круговой диаграммы необходимо вначале вычислить Fi, как это выполняется при построении гистограммы, затем - строить (последовательно вычерчивать) секторы диаграммы.

```
A1:=0; {угол первого луча сектора}
XC:=(GetMaX + 1)div 2; {координаты центра диаграммы}
YC:=(GetMaY + 1)div 2;
R:=YC - 40; {радиус окружности}
RN:=R+5; {радиус для “привязки” надписи Fi}
begin
A2:=A1+3.6*F(L); {угол второго луча}
```

```
SetFillStyle(SolidFill, i); {установка параметров для заполнения сектора}
PilSlice(XC, YC, A1, A2, R);
end;
{Вывод числовых данных к каждому сектору: значение Fi - в процентах,
“привязанное” к медиане i-го угла}
A:=(A1 + A2)/2;
XN:=XC + Round(RN*cos(A);
YN:=YC - Round(RN*sin(A);
Str(F[i]:5:2, S);
if (A>90) and (A<270) then
SetTextJustifi(RightText,O);
else
SetTextJustifi(LeftText,O);
OutTextXY(XN, YN, S);
```

Столбиковая диаграмма (гистограмма)

Гистограмма - графическое изображение оценки плотности распределения случайной величины в диапазоне имеющихся опытных, экспериментальных значений $[M_{min}, M_{max}]$, при этом диапазон делится на L интервалов, в каждом из которых распределение считается равномерным и равным F_i, (i = 1, 2, ..., L);

$F_i = \frac{k_i}{n}$; n - количество значений случайной величины; k_i - количество попаданий случайной величины в i - й интервал (рис.1).

Другими словами, эта диаграмма показывает, какая доля значений какой-либо величины (цена товара, заработная плата, возраст пациентов и др.) попадает в каждый из заданных интервалов.

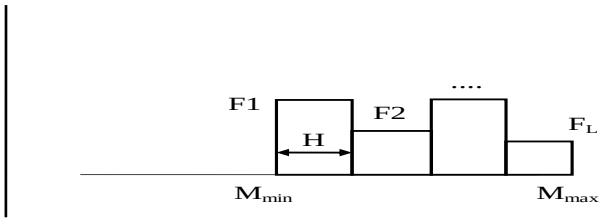


Рисунок 15.1

Порядок построения гистограммы можно определить следующим образом:

- подготовлен массив статистических данных *M* (n чисел); как правило, статистические данные накапливаются в файлах, а для обработки данные могут быть переписаны в массив; примем, что *M: array [1...d] of real,* d ≥ n;

- задается количество интервалов для построения гистограммы (L: byte); как правило, значение *L* зависит от количества данных, цели анализа и др.; $5 \le L \le 20$.

- необходимо построить столбиковую диаграмму (гистограмму) по данным из массива *M*, пользуясь следующими правилами:

- ширина основания «столбика» (прямоугольника) равна $H = \frac{M_{\max} - M_{\min}}{L}$,

где M_{max}, M_{min} - соответственно максимальное и минимальное число в массиве *M*,

- высота i-го «столбика» (i = 1, 2, ..., L) пропорциональна величине $F_i = \frac{k_i}{n} 100$, где k_i - количество чисел из массива *M*, попадающих в i - й интервал; если не требуется значение k_i в процентах, то Fi не вычисляется, а высота «столбика» будет пропорциональна k_i.

Таким образом, для решения задачи необходимо найти M_{min} и M_{max}, определить массивы для хранения k_i, F_i (i = 1, 2, ..., L) и границ интервалов G_i (i = 1, 2, ..., L + 1).

Алгоритм построения гистограммы

5. Определение массивов *M, K, G, F* и других переменных.
6. Чтение данных из файла в массив *M*.
7. Ввод с клавиатуры количества интервалов *L*.
8. Поиск *Mmin* и *Mmax* в массиве *M*.
9. Вычисление ширины основания прямоугольника *H*.
10. Обнуление массива *K*.
11. Заполнение массива *G* (определение границ интервалов):

G[1]: = Mmin;

for i: = 2 to L do

g[i]: = G[i-1] + H;

G[L + 1]: = Mmax;

12. Определение значений Ki (i = 1, 2, ..., L):

for i: = 1 to n do

for j: = 1 to L do

if (j < L) and (M[i] >= G[j]) and (M[i] < G[j + 1]) then

Inc K[j]

else

if (M[i] >= G[j]) and (M[i] <= G[j =1]) then Inc(K[j]);

изменяем условие: включаем верхнюю границу интервала}

13. Вычисление *Fi* и поиск *Fmax* или *Kmax*; минимальное значение *Fi* равно нулю.

14. Установка параметров для графического режима (цвет фона, тип линии и др.) и изображения на экране:

XNE, XKE, YNE, YKE - координаты линий прямоугольника, ограничивающих график.

15. Переход в графический режим.

16. Вычисление масштаба изображения:

Mx := (XKE - XNE) / (Mmax - M min);

My: = (YKE - YNE) / Fmax (или Kmax);

17. Цикл построения гистограммы; «столбики» закрашиваются различными цветами:

YE1:=X_Ekr (G[j]);

for j:=1 to L do

begin

SetFillPattern (Mask[j], Col[j]);

{если требуется пользовательская штриховка; *Mask* - массив кодов пользовательских штриховок; *Col* - массив кодов цветов для заполнения

прямоугольников}

SetFillStyle (Mask1[j], Col1[j]);

{*Mask1* - массив кодов для заполнения, включая стандартные; Col1 - то же, что *Col*; при сплошном заполнении прямоугольника вместо *Mask1* подставить константу, например *SolidFill*}

XE1:=X_Ekr(G[j]);

XE2:=X_Ekr(G[j+1]);

YE2:=Y_Ekr(F[j]);

Bar(XE1, YE1, XE2, YE2)

end;

18. Вывод осей координат, поясняющих надписей:

SetColor(White);

XE1:=X_Ekr(M_{min});

Line(XE1, YE1, X_Ekr(Mmax + H), YE1);

Line(XE1, YE1, XE1, X_Ekr(Fmax + H));

Str(Mmin:6:2,S)

SetTextStyle(Sansseriffont, Horizdir, NormSize);

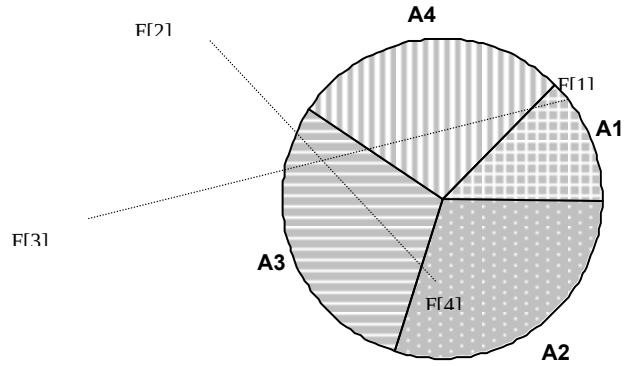
OutTextXY(XE1, YE1 + 10, S);

19. Вывод других надписей выполняется аналогично.

Круговая диаграмма

В круговой диаграмме центральные углы секторов *Ai* пропорциональны значениям *Fi* (или *Ki*), гле *Fi* и *Ki* определяются как определено выше (рис. 2):

Ai=(3, 6•Fi).



.

Рисунок 15.2

Алгоритм построения круговой диаграммы

Для построения круговой диаграммы необходимо вначале вычислить *Fi*, как это выполняется при построении гистограммы, затем - строить (последовательно вычерчивать) секторы диаграммы.

A1:=0; {угол первого луча сектора}

XC:=(GetMaX + 1)div 2; {координаты центра диаграммы}

YC:=(GetMaY + 1)div 2;

R:=YC - 40; {радиус окружности}

RN:=R+5; {радиус для «привязки» надписи Fi}

begin

*A2:=A1+3.6*F(L);* {угол второго луча}

SetFillStyle(SolidFill, i); {установка параметров для заполнения сектора}

PilSlice(XC, YC, A1, A2, R);

end;

{Вывод числовых данных к каждому сектору: значение Fi - в процентах, «привязанное» к медиане i-го угла}

A:=(A1 + A2)/2;

*XN:=XC + Round(RN*cos(A);*

*YN:=YC - Round(RN*sin(A);*

Str(F[i]:5:2, S);

if (A>90) and (A<270) then

SetTextJustifi(RightText,∅);

else

SetTextJustifi(LeftText,∅);

OutTextXY(XN, YN, S);

3. лекция практика

4. **Дополнительные вопросы**

1. . ***Диалоговый режим*** - это обмен данными между человеком и компьютером в темпе, соизмеримом с темпом обработки данных человеком.

2. Интерфейс пользователя (User interface) - программные и аппаратные средства взаимодействия пользователя с программой или ЭВМ.

3. Одним из распространенных типов интерфейсов человека и компьютера является ***интерфейс на основе меню*** ("смотри и выбирай"). Интерфейс такого типа облегчает взаимодействие пользователя с компьютером, так как устраняет необходимость изучения пользователем языка общения с вычислительной системой.

4. Сортировка – это процесс перегруппировки заданного множества объектов в некотором определённом порядке.

Цель сортировки: облегчить поиск элементов.

5. Методы сортировки можно разбить на два класса – сортировку массивов

и сортировку файлов. Иногда их называют внутренней и внешней сортировкой,

так как массивы хранятся в быстрой, оперативной, внутренней памяти машины

со случайным доступом, а файлы размещаются в более медленной, но и более ёмкой внешней памяти.

Сортировка: выбором, подсчетом, вставками, обменная, цифровая, слиянием.

6. Структуры меню: горизонтальное, вертикальное Меню для выбора функции с помощью выделенных символов Меню для выбора функции с помощью функциональных клавиш Меню для выбора функции с помощью мнемонических кодов (команд)

7. Способы построения графиков: точечный, кусочно- линейный

8. Что называется модулем ?

9. Массив – это одно- или многомерная таблица данных одного типа. Каждая ячейка таблицы имеет свой индекс (в одномерном случае) или набор индексов (в многомерном). Массив называют структурой данных со случайным доступом, поскольку к любому элементу массива можно обратиться, просто указав его индексы, т.е. все элементы одинаково доступны в любой момент времени.

10. Запись – связанная структура, состоящая из нескольких элементов (полей) разных (можно и одинаковых) типов. По сути, запись очень похожа на одномерный массив, но с элементами разных типов, кроме того, доступ к конкретному полю записи осуществляется уже не через индекс, а указанием идентификатора (т.е. имени) этого поля

11. Файл – динамическая структура данных, размер которой может меняться в процессе выполнения над ним каких-либо действий (он может быть равен нулю, что соответствует пустому файлу)

12. Стек – структура данных, представляющая собой последовательность элементов. Добавление и удаление элементов происходит только с одного конца последовательности, т.е. при изменении структуры предыдущая последовательность остается неизменной

13. Очередь – структура данных, как и стек представляющая собой последовательность элементов. Добавление элементов происходит с

одной стороны последовательности, удаление – с другой. Дек – двусторонняя очередь, т.е. и добавление и удаление осуществляется с обеих сторон.

14 Линейный список – одна из тех структур данных, которая если не приравнивается, то хотя бы ассоциируется с динамическими структурами. Он представляет собой упорядоченное множество (возможно пустое), в котором добавление и удаление элементов может происходить на любом месте списка.

15. Дерево – множество, состоящее из элемента, называемого корнем, и конечного (возможно пустого) множества деревьев, называемых поддеревьями данного дерева. Дерево, так же как и список, реализуется прежде всего на динамических структурах, т.е. узел дерева содержит два поля-ссылки – на левое и правое поддерево для двоичного дерева и на брата и старшего сына для дерева общего вида. Дерево уже не является линейной структурой, поэтому представление деревьев на последовательной памяти (файлах) представляет собой определенную проблему.

16. Виды диаграмм: круговая, столбиковая, полигон (линейная)

17. Методы целочисленного программирования: прямоугольников, трапеций, Симпсона

18 Методы решения нелинейных уравнений: хорд, касательной (Ньютона), дихотомии (деление пополам)