Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования

Владимирский государственный университет

Кафедра информационных систем и информационного менеджмента

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ НА ЯЗЫКАХ ВЫСОКОГО УРОВНЯ»

Составители:
В.С. ГРИШИН
Д.И. ГУСЕВ

Владимир 2009

УДК 004(07)
ББК 32.97 Я2
М54

Рецензент

Кандидат технических наук, профессор
зав. кафедрой информатики и компьютерной техники
Владимирского государственного гуманитарного университета
Ю.А. Медведев

Печатается по решению редакционного совета
Владимирского государственного университета

Методические указания к лабораторным работам по дисциплине «Программирование на языках высокого уровня» / Владим. гос. ун-т ; сост.: В.С. Гришин, Д.И. Гусев. – Владимир : Изд-во Владим. гос. ун-та, 2009. – 36 с.

М54

Содержат шесть лабораторных работ, которые позволяют получить практические навыки в области программирования. В соответствии с действующим образовательным стандартом в учебную программу по дисциплине «Программирование на языках высокого уровня» входят разделы, посвященные сортировке и поиску, рекурсии, основам модульного программирования. Способствуют овладению одним из популярных языков программирования – Паскаль, изучению и освоению методов и приемов программирования на этом языке, получению практических навыков работы.

Предназначены для студентов всех форм обучения специальности 230201 «Информационные системы и технологии» (системы поддержки принятия решений).

Ил. 2. Библиогр.: 4 назв.

УДК 004(07)
ББК 32.97 Я2

ВВЕДЕНИЕ

В любой сфере деятельности человеку для принятия правильного решения необходимо располагать какими-либо сведениями, документами, фактами, другими словами, иметь определенную информацию. В настоящее время широкое распространение получили информационные системы (ИС). При проектировании этих систем в большинстве случаев используют языки высокого уровня, к числу которых относятся, например, 1С, Паскаль, Delphi, Visual FoxPro.

Паскаль (*Pascal*) – язык программирования общего назначения. Один из наиболее известных языков программирования, широко применяется в промышленном программировании, обучении программированию в высшей школе, является базой для большого числа других языков. Был создан Николаусом Виртом в 1970 г. после его участия в работе комитета разработки стандарта языка Алгол-68.

Паскаль был создан как язык для обучения процедурному программированию (хотя, по словам Вирта, язык нельзя считать только учебным, поскольку язык, непригодный для написания реальных программ, для обучения использоваться не должен). Название языку дано в честь выдающегося французского математика, физика, литератора и философа Блеза Паскаля. Паскаль – один из первых языков, для которых была создана реализация «на самом себе» – компилятор Паскаля был написан на самом Паскале.

Многие идеи языка Паскаль были заимствованы создателями известных языков программирования (например, 1С, Java и др.).

Особенностями языка являются *строгая типизация* и наличие средств *структурного (процедурного) программирования*. Паскаль был одним из первых таких языков. По мнению Н. Вирта, язык должен способствовать дисциплинированию программирования, поэтому наряду со строгой типизацией в Паскале сведены к минимуму возможные синтаксические неоднозначности, а сам синтаксис автор постарался сделать интуитивно понятным даже при первом знакомстве с языком.

Развитием языка Паскаль стал Delphi (Object Pascal), который в настоящее время является достаточно популярным.

ОБЩИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Задания к лабораторным работам приведены после описания каждой из них. Преподаватель может дать дополнительные задачи, которые студент должен выполнить в ходе работы. При защите работ студент должен быть готов к ответу на контрольные вопросы и продемонстрировать созданное им приложение.

Оформление лабораторной работы

Отчет по лабораторной работе выполняется на писчей бумаге формата А4. Содержание отчета:

- Титульный лист.
- Задание.
- Содержание.
- Граф-схема алгоритма.
- Текст приложения с комментариями.
- Исходные данные.
- Полученные результаты.

Алгоритмы

Алгоритм – система правил, сформулированная на понятном исполнителю языке, которая определяет процесс перехода от допустимых исходных данных к некоторому результату и обладает свойствами массовости, конечности, определенности, детерминированности.

Слово «алгоритм» часто употребляется в связи с решением задач с помощью компьютеров.

Проблема определения понятия «алгоритм». На протяжении многих веков понятие алгоритма связывалось с числами и относительно простыми действиями над ними, да и сама математика была по большей части наукой о вычислениях, наукой прикладной. Чаще всего алгоритмы представлялись в виде математических формул. Порядок элементарных шагов алгоритма задавался расстановкой скобок, а сами шаги заключались в выполнении арифметических операций и операций отношения (проверки равенства, неравенства и т.д.). Часто

вычисления были громоздкими, а вычисления вручную – трудоемкими, но суть самого вычислительного процесса оставалась очевидной. У математиков не возникала потребность в осознании и строгом определении понятия алгоритма, в его обобщении. Но с развитием математики появлялись новые объекты, которыми приходилось оперировать: векторы, графы, матрицы, множества и др. Как определить для них однозначность или как установить конечность алгоритма, какие шаги считать элементарными? В 1920-х гг. задача точного определения понятия алгоритма стала одной из центральных проблем математики.

Наличие исходных данных и некоторого результата. Алгоритм – это точно определенная инструкция, последовательно применяя которую к исходным данным, можно получить решение задачи. Для каждого алгоритма есть некоторое множество объектов, допустимых в качестве исходных данных. Например, в алгоритме деления вещественных чисел делимое может быть любым, а делитель не может быть равен нулю.

Массовость означает возможность применять многократно один и тот же алгоритм. Алгоритм служит, как правило, для решения не одной конкретной задачи, а некоторого класса задач. Так, алгоритм сложения применим к любой паре натуральных чисел.

Детерминированность означает, что при применении алгоритма к одним и тем же исходным данным должен получаться всегда один и тот же результат, поэтому, например, процесс преобразования информации, в котором присутствует момент – бросание монеты, не является детерминированным и не может быть назван алгоритмом.

Результативность. Выполнение алгоритма должно обязательно приводить к его завершению. В то же время можно привести примеры формально бесконечных алгоритмов, широко применяемых на практике. Например, алгоритм работы системы сбора метеорологических данных состоит в непрерывном повторении последовательности действий («измерить температуру воздуха», «определить атмосферное давление»), выполняемых с определенной частотой (через минуту, час) во все время существования данной системы.

Определенность. На каждом шаге алгоритма у исполнителя должно быть достаточно информации, чтобы его выполнить. Кроме того, исполнителю нужно четко знать, каким образом он выполняет-

ся. Шаги инструкции должны быть достаточно простыми, элементарными, а исполнитель – однозначно понимать смысл каждого шага последовательности действий, составляющих алгоритм (при вычислении площади прямоугольника любому исполнителю нужно уметь умножать и трактовать знак «*» именно как умножение). Поэтому вопрос о выборе формы представления алгоритма очень важен. Фактически речь идет о том, на каком языке записан алгоритм.

Формы представления алгоритмов. Для записи алгоритмов необходим некоторый язык, при этом очень важно, какой именно язык выбран. Записывать алгоритмы на русском языке (или любом другом естественном языке) громоздко и неудобно. Например, упрощенное описание алгоритма Евклида нахождения наибольшего общего делителя (НОД) двух целых положительных чисел может быть представлено в виде трех шагов.

Шаг 1. Разделить m на n. Пусть p – остаток от деления.

Шаг 2. Если p равно нулю, то n и есть искомый НОД.

Шаг 3. Если p не равно нулю, то сделаем m равным n, а n равным p. Возврат к шагу 1.

Для описания алгоритмов часто служат языки программирования высокого уровня.

Граф-схема алгоритма (ГСА) – ориентированный связный граф, содержащий одну *начальную* вершину, одну *конечную* и произвольное конечное множество *условных* и *операторных* вершин.

Граф-схемы алгоритмов. Являются одним из распространенных способов записи алгоритмов.

На рис. 1 показаны графические обозначения вершин.

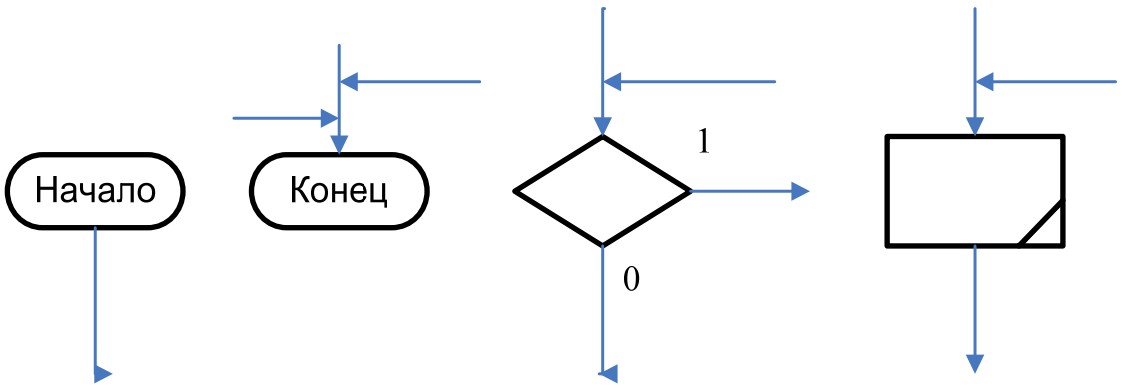

Рис. 1. Вершины ГСА

Правила построения ГСА. Конечная, операторная и условная вершины имеют по одному входу, начальная вершина входов не имеет. Начальная и операторная вершины имеют по одному выходу, а условная – два выхода, помеченных символами 1 (да) и 0 (нет). Конечная вершина выходов не имеет. ГСА удовлетворяет следующим условиям:

1. Входы и выходы вершин соединяются друг с другом с помощью дуг, направленных всегда от выхода ко входу.
2. Каждый выход соединен точно с одним входом.
3. Любой выход соединен точно с одним входом.

На рис. 2 представлена граф-схема алгоритма нахождения НОД.

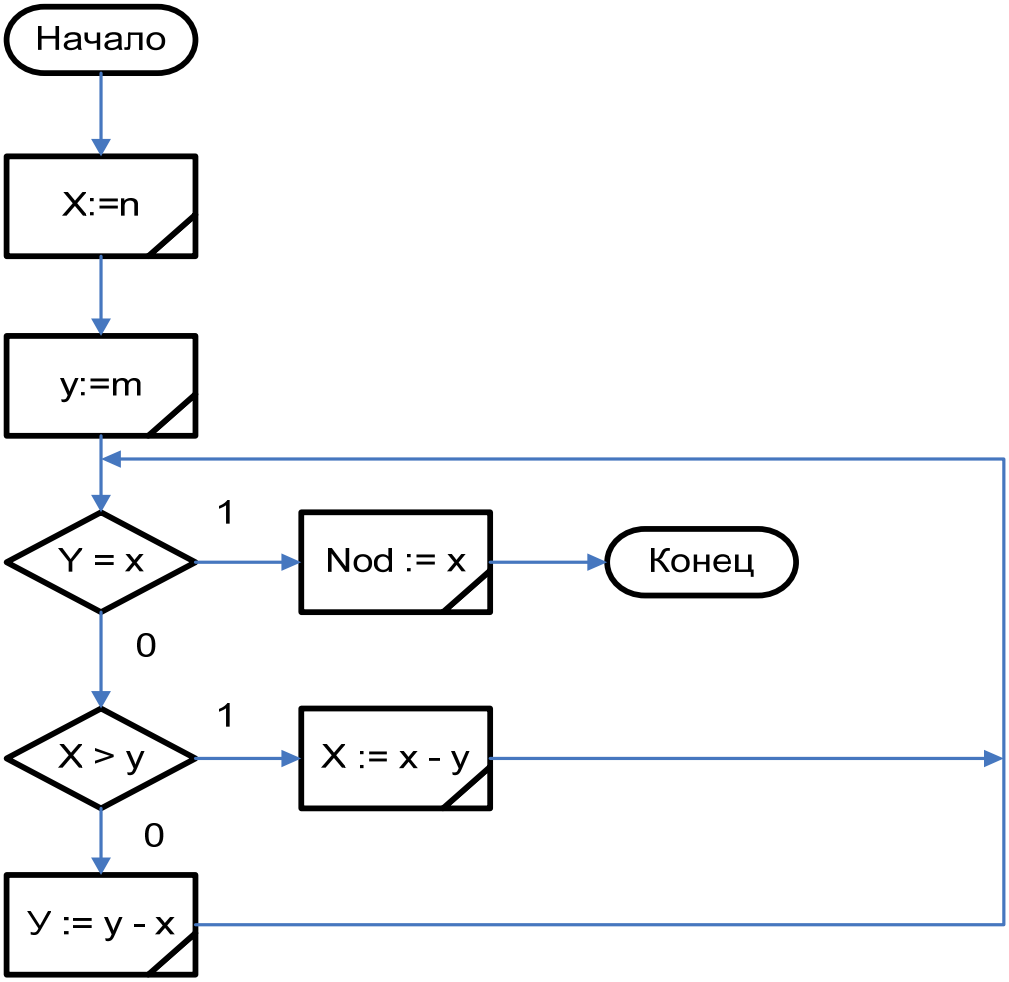

Рис. 2. Граф-схема алгоритма нахождения НОД

4. Любая вершина графа лежит, по крайней мере, на одном пути из начальной вершины к конечной вершине.

5. Один из выходов условной вершины может соединяться с ее входом, что недопустимо для операторной вершины.
6. В каждой условной вершине записывается логическое условие.
7. В каждой операторной вершине записывается оператор.

Комментарии

При написании программы обязательно вставляйте в нее предложения (на любом распространенном языке, например, русском или английском), чтобы объяснить, для чего служит данный фрагмент кода. Эти комментарии очень пригодятся, если вы будете налаживать программу или расширять ее возможности. Не забывайте о комментариях и не откладывайте работу над ними на будущее, иначе потом не вспомните, для чего было нужно то или иное конкретное действие.

Иногда примечания пишут в расчете на последующее использование. Прокомментировав код с применением стандартных соглашений, вы сможете позже извлечь эти заметки из программы и вставить их в документацию.

Соглашения о форматировании кодов

Форматирование кода делает его более удобным для других разработчиков. Сравним два сегмента кода.

Первый листинг.

```
{Определяется текст сообщения}
if lnCount = 100 then
begin
lcMessage = ‘Сообщение 1’;
lnValue = 200;
end;
if lnCount2 = 300 then
lcMessage = ‘Сообщение 2’;
```

Второй листинг.

```
{Определяется текст сообщения}

if lnCount = 100 then
   begin
        lcMessage = 'Сообщение 1';
        lnValue = 200;
   end;

if lnCount2 = 300 then
   lcMessage = 'Сообщение 2';
```

Разумеется, вторую запись читать проще: текст напечатан с отступами и содержит достаточно много свободного пространства. Человеческому глазу нужны промежутки между объектами, тогда он может их различать. Второй листинг вполне удовлетворяет этому условию.

Лабораторная работа № 1

РЕКУРСИЯ

Цель работы. Познакомиться с одним из эффективных способов решения сложных задач – рекурсией.

Общие сведения

Очень часто, разрабатывая программу, удается свести исходную задачу к более простым задачам. Среди этих задач может оказаться и первоначальная задача, но в упрощенной форме. Например, вычисление функции $F(n)$ может потребовать вычисления $F(n - 1)$ и еще каких-то операций. Иными словами, частью алгоритма вычисления функции будет вычисление этой же функции.

Алгоритм называется рекурсивным, если он прямо или косвенно обращается к самому себе. Часто в основе такого алгоритма лежит рекурсивное определение какого-то понятия. Например, о факториале

числа N можно сказать, что $N! = N*(N-1)!$, если $N > 0$ и $N! = 1$, если $N = 0$ – это рекурсивное определение.

Вот еще одно рекурсивное определение.

1. Три коровы – это стадо коров.
2. Стадо из n коров – это стадо из $n-1$ коровы и еще одна корова.

Попробуем применить это определение для проверки, является ли стадом группа из пяти коров (обозначим ее К5). Объект К5 не удовлетворяет первому пункту определения, поскольку пять коров – это не три коровы. Согласно второму пункту К5 – стадо, если там есть одна корова, а остальная часть К5, назовем ее К4, – тоже стадо коров. Решение относительно объекта К5 откладывается, пока не будет принято решение относительно К4. Объект К4 снова не подходит под первый пункт, а второй пункт гласит, что К4 – стадо, если объект К3, полученный из К4 путем отделения одной коровы, тоже стадо. Решение о К4 тоже откладывается. Наконец, объект К3 удовлетворяет первому пункту определения, и мы можем смело утверждать, что К3 – стадо коров. Теперь и о К4 можно утверждать, что это стадо, а значит, и К5 является стадом коров.

Любое рекурсивное определение состоит из двух частей. Эти части принято называть базовой и рекурсивной частями. Базовая часть является нерекурсивной и задает определение для некоторой фиксированной части объектов. Рекурсивная часть определяет понятие через него же и записывается так, чтобы при цепочке повторных применений она редуцировалась бы к базе.

Примеры

Задача 1. Написать рекурсивную программу поиска минимального элемента массива.

Решение. Опишем функцию *P*min, которая определяет минимум среди первых n элементов массива a. Параметрами этой функции являются: количество элементов в рассматриваемой части массива – n и значение последнего элемента этой части – $a[n]$. При этом если $n > 2$, то результатом является минимальное из двух чисел – $a[n]$ и минимального числа из первых $(n-1)$ элементов массива. В этом заключается рекурсивный вызов. Если же $n = 2$, то результатом является ми-

нимальное из первых двух элементов массива. Чтобы найти минимум всех элементов массива, нужно обратиться к функции *P*min, указав в качестве параметров значение размеров массива и значение последнего его элемента. Минимальное число из двух определяется с помощью функции Min, параметрами которой являются эти числа.

```
Program Example_1;
Const
        n = 10;
Type
        MyArray = Array[1..n] of Integer;
Const
       A : MyArray = (4,2,-1,5,2,9,4,8,5,3);

Function Min (a, b : Integer) : Integer;
Begin
       if a > b then
              Min := b
       else
              Min := a;
End;

Function Pmin(n, b : Integer) : Integer;
Begin
        if n = 2 then
                Pmin := Min(n, a[1])
        else
                Pmin := Min(a[n], Pmin(n-1,a[n]));
End;
Begin
        WriteLn('Минимальный элемент массива  - ', Pmin(n, a[n]));
End.
```

Задача 2. *Ханойские башни*. Имеется три стержня *A*, *B*, *C*. На стержень *A* нанизано *n* дисков радиуса 1, 2, ..., *n* таким образом, что диск радиуса *i* является *i*-м сверху. Требуется переместить все диски на стержень *B*, сохраняя их порядок расположения (диск с большим

радиусом находится ниже). За один раз можно перемещать только один диск с любого стержня на любой другой стержень. При этом должно выполняться следующее условие: на каждом стержне ни в какой момент времени никакой диск не может находиться выше диска с меньшим радиусом.

Решение. Предположим, что мы умеем перекладывать пирамиду из (n – 1) дисков. Рассмотрим пирамиду из n дисков. Переместим первые (n – 1) дисков на стержень C (это мы умеем). Затем перенесем последний n-й диск со стержня A на стержень B. Далее перенесем пирамиду из (n – 1) дисков со стержня C на стержень B. Так как n-й диск самый большой, то условие задачи не будет нарушено. Таким образом, вся пирамида будет на стержне B. Аналогичным образом можно перенести n – 2, n – 3 и т. д. дисков. Когда n = 1, осуществить перенос очень просто: непосредственно с первого стержня на второй. При этом для решения задачи будет достаточно 2^n – 1 перекладываний.

```
Program Example_2;
Const
        k = 3;
Var
        a, b, c : Char;

Procedure Disk(n : Integer; a, b, c: Char);
Begin
        if n > 0 then
                begin
                        Disk(n-1, a, c, b);
                        WriteLn('Диск ', n, ' с ', a, '-> ', b);
                       Disk(n-1, c, b, a);
                end;
End;

Begin
        a := 'A'; b := 'B'; c := 'C';
        Disk(k, a, b, c);
        ReadLn;
End.
```

Контрольные вопросы

1. На чем основан рекурсивный метод программирования?
2. В чем разница между циклическим и рекурсивным способами определения? Какой элемент является обязательным в рекурсивном определении?
3. Что такое фрейм активации?
4. К каким последствиям приводит рекурсивное зацикливание?
5. Какое условие должно обязательно присутствовать в любой рекурсивной процедуре?
6. Что такое явная и косвенная рекурсии?
7. Дайте рекурсивное определение целой степени числа *N*.

Варианты заданий

1. Ввести последовательность чисел (окончание ввода – 0) и вывести их в обратной последовательности. Входные данные взять из текстового файла.
2. Используя команды *Write*(*x*) лишь при $x = 0...9$, написать рекурсивную программу печати десятичной записи целого положительного числа *n*.
3. Напишите рекурсивную функцию, которая возвращает среднее из *n* элементов массива чисел.
4. Найти первые *N* чисел Фибоначчи двумя способами: с помощью рекурсии и с помощью итерации. Сравнить эффективность алгоритмов.
5. Написать функцию сложения двух чисел, используя только прибавление единицы.
6. Написать функцию умножения двух чисел, используя только операцию сложения.
7. Вычислить сумму элементов одномерного массива.
8. Найти НОД двух натуральных чисел.
9. Вычислить несколько значений функции Аккермана для неотрицательных чисел *m* и *n*: $A(n,m) = \begin{cases} m+1, & n = 0, \\ A(n-1,1), & n \neq 0,\ m = 0, \\ A\big(n-1, A(n,m-1)\big), & n > 0,\ m \geq 0. \end{cases}$
10. Напишите рекурсивную функцию, которая вычисляет длину строки.

11. Написать функцию $C(m, n)$ вычисления биномиальных коэффициентов C_n^m по следующей формуле: $C_n^0 = C_n^n = 1$, $C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$ при $0 < m < n$.
12. Проверить, является ли фрагмент строки с i-го по j-й символ палиндромом.
13. Вычислить произведение элементов одномерного массива.
14. Написать процедуру сортировки массива методом простого выбора.
15. Подсчитать количество цифр в заданном числе.
16. Написать функцию, проверяющую правильность имени в языке Pascal.
17. Написать функцию *Root*(*f*, *a*, *b*, *eps*), которая методом деления отрезка пополам (методом дихотомии) находит с точностью *eps* корень уравнения $f(x) = 0$ на отрезке $[a, b]$ ($eps > 0$, $a < b$, $f(a)*f(b) < 0$). Метод дихотомии определяется следующим образом. Если $f(a)$ и $f(b)$ имеют разные знаки, то между точками a и b существует корень R. Пусть $m = (a + b) / 2$ – средняя точка в интервале $a <= x <= b$. Если $f(m) = 0$, то корень $R = m$. Если нет, то либо $f(a)$ и $f(m)$ имеют разные знаки ($f(a)*f(m) < 0$), либо $f(m)$ и $f(b)$ имеют разные знаки ($f(b)*f(m) < 0$). Если ($f(a)*f(m) < 0$), то корень лежит в интервале $a <= x <= m$. В противном случае он лежит в интервале $m <= x <= b$. Теперь выполним это действие для нового интервала – половины исходного интервала. Процесс продолжается до тех пор, пока интервал не станет меньше *eps*.
18. В текстовом файле задана последовательность положительных чисел, за которой следует отрицательное число. Написать функцию без параметров для нахождения суммы этих положительных чисел.
19. Написать функцию без параметров, которая подсчитывает количество цифр в тексте, заданном в текстовом файле (за текстом следует точка).
20. Расстояния между городами заданы матрицей (если между городами i, j есть прямой путь с расстоянием N, то элементы матрицы $A(i, j)$ и $A(j, i)$ содержат значение N, иначе 0). Написать программу поиска минимального пути для произвольной пары городов.

21. Вычислить определитель матрицы, пользуясь формулой разложения по первой строке: $\det A = \sum_i (-1)^{i+1} a_{1i} * \det B_i$, где матрица *Bk* получается из *A* вычеркиванием первой строки и *k*-го столбца.
22. Написать функцию определения, является ли заданное натуральное число простым.
23. Реализовать рекурсивный алгоритм построения цепочки из имеющегося набора костей домино.
24. Расстояния между городами заданы матрицей (если между городами *i*, *j* есть прямой путь с расстоянием *N*, то элементы матрицы *A*(*i*, *j*) и A(*j*, *i*) содержат значение *N*, иначе 0). Написать программу поиска минимального пути обхода всех городов без посещения дважды одного и того же города (задача коммивояжера).
25. Задан набор слов. Построить из них любую цепочку таким образом, чтобы символ в конце слова совпадал с символом в начале следующего.
26. Задан массив целых чисел. Построить из них любую последовательность таким образом, чтобы последняя цифра предыдущего числа совпадала с первой цифрой следующего.
27. Для данного *n* напечатайте коэффициенты разложения полинома (1 + *x*)^*n*.
28. Вычислить, используя рекурсию, выражение $\sqrt{6+2\sqrt{7+3\sqrt{8+4\sqrt{9+K}}}}$.
29. Написать процедуру печати всех перестановок из *n* символов.
30. Написать процедуру перевода числа из десятичной системы счисления в двоичную систему счисления.

Лабораторная работа № 2

СОРТИРОВКА МЕТОДОМ ПРОСТОГО ВЫБОРА

Цель работы. Исследовать сортировку массива методом простого выбора и оценить эффективность этого алгоритма.

Общие сведения

Сортировка методом простого выбора сводится к следующим шагам:

1. Установить номер наибольшего элемента массива.
2. Поменять местами наибольший и последний элементы массива.
3. Оставив в покое последний элемент, выполнить п. 1 и 2 над остатком массива (массивом без последнего элемента), п. 3 повторять, пока остаток массива не сократится до одного элемента.

Пример

Опишем процедуру сортировки на языке проектирования программ (псевдокоде).

```
For i := n downto 2 do
 Begin
       найти максимальный элемент из a[1], ..., a[i];
       запомнить его индекс в переменной k;
      если i<>k поменять местами a[i] и a[k];
 End;
```

Вот как изменяется значение массива из пяти элементов (30, 20, 10, 50, 40):

30	20	10	50	40
30	20	10	40	50
30	20	10	40	50
10	20	30	40	50
10	20	30	40	50

Подчеркнута область поиска наибольшего элемента.

Контрольные вопросы

1. Что понимается под сортировкой массива?
2. Чем отличается сортировка по убыванию от сортировки по невозрастанию?

3. Сформулировать идею сортировки массива методом простого выбора.
4. Почему время выполнения одного шага прямо пропорционально размеру неупорядоченной части массива?

Варианты заданий

1. Изменить процедуру сортировки так, чтобы сортировка производилась по убыванию элементов.
2. Проверить, является ли данная последовательность целых чисел упорядоченной по убыванию. Если нет, упорядочить ее.
3. Отсортировать данный массив и подсчитать количество уникальных чисел в массиве.
4. Изменить процедуру сортировки так, чтобы значение параметра *i* с каждым шагом увеличивалось.
5. Отсортировать четные элементы массива с помощью простого выбора.
6. Отсортировать с помощью простого выбора элементы массива, стоящие на нечетных местах.
7. Отсортировать положительные элементы массива с помощью простого выбора.
8. Отсортировать отрицательные элементы массива с помощью простого выбора.
9. В матрице $n*m$ отсортируйте столбцы в порядке возрастания.
10. Дан список футбольных команд высшей лиги России и количество очков, набранных каждой командой в чемпионате России. Известно, что нет команд с равным числом очков. Распечатать список призеров.
11. В неупорядоченном массиве могут быть совпадающие элементы. Из каждой группы одинаковых элементов оставить только один, удалив остальные и «поджав» массив к его началу.
12. Турнирная таблица соревнований представлена квадратной матрицей A, каждый элемент которой $a[i, j]$ есть число голов, забитых i-й командой в ворота j-й команды. По диагонали расположить место каждой команды (по числу побед за вычетом числа поражений; в случае равенства – по разности забитых и пропущенных голов).

Лабораторная работа № 3

СОРТИРОВКА МЕТОДОМ ПРОСТОГО ОБМЕНА

Цель работы. Исследовать сортировку массива методом простого обмена и оценить эффективность этого алгоритма.

Общие сведения

Как и предыдущий, этот алгоритм состоит из отдельных шагов. На каждом шаге массив проходят от начала к концу, сравнивая пары соседних элементов. Если очередная пара нарушает требуемый порядок, ее элементы меняют местами. Шаги повторяют до тех пор, пока очередной проход не вызовет ни одного обмена.

Пример

Отсортируем по возрастанию методом простого обмена массив из пяти элементов [5, 1, 8, 4, 9]. Длина текущей части массива – $n - k + i$, где k – номер просмотра, i – номер проверяемой пары, $n - k$ – номер последней пары.

Первый просмотр: рассматриваем весь массив.

$i = 1$ → 5 4 8 2 9 → > меняем
$i = 2$ → 4 5 8 2 9 → < не меняем
$i = 3$ → 4 5 8 2 9 → > меняем
$i = 4$ → 4 5 2 8 9 → < не меняем

9 стоит на своем месте.

Второй просмотр: рассматриваем часть массива с первого до четвертого элемента.

$i = 1$ → 4 5 2 8 9 → < не меняем
$i = 2$ → 4 5 2 8 9 → > меняем
$i = 3$ → 4 2 5 8 9 → < не меняем

8 стоит на своем месте.

Третий просмотр: рассматриваемая часть массива содержит первые три элемента.

$i = 1$ → 4 2 5 8 9 → > меняем
$i = 2$ → 2 4 5 8 9 → < не меняем

5 стоит на своем месте.

Четвертый просмотр: рассматриваем последнюю пару.

$i = 1$ → 2 4 5 8 9 → < не меняем

4 стоит на своем месте.

Для самого маленького элемента (2) остается только одно место – первое.

Итак, наш массив отсортирован по возрастанию элементов методом простого обмена. Этот метод также называют методом «пузырька». Название происходит от образной интерпретации, при которой в процессе выполнения сортировки более «легкие» элементы (элементы с заданным свойством) мало-помалу всплывают на «поверхность».

Контрольные вопросы

1. Как поменять местами два элемента массива?
2. Что такое вложенные циклы?
3. В чем сходство и отличие методов простого выбора и простого обмена?

Варианты заданий

Для всех вариантов предварительно написать процедуру «пузырьковой» сортировки.

1. Подсчитать количество выполненных сравнений и перестановок в лучшем, худшем случаях и в среднем.
2. Упорядочить первые *n* элементов данного ряда в порядке возрастания. Напечатать эти элементы в порядке убывания.

3. В нашем примере два последних прохода не влияют на порядок элементов, так как они уже отсортированы. Следовательно, можно улучшить наш алгоритм, если запоминать, производились ли перестановки элементов в процессе некоторого прохода. Если их не было, то сортировку можно закончить. Модифицировать процедуру с учетом этой возможности улучшения.
4. Если известен не только факт последнего обмена, но и его место, то нетрудно заметить, что все пары соседних элементов, расположенные правее этого места, уже находятся в нужном порядке. Поэтому просмотр можно закончить на этом индексе, а не продолжать до конца. Модифицировать процедуру с учетом этой возможности улучшения.
5. Написать процедуру «пузырьковой» сортировки по убыванию.
6. Найти второй по величине (после наименьшего) элемент данного ряда.
7. Найти так называемую медиану ряда, т. е. такой его элемент, который больше любого из одной половины элементов и меньше любого из другой (если число элементов ряда четно, следует взять среднее значение из двух значений, обладающих указанным свойством).
8. В методе «пузырьковой» сортировки, если какие-то два соседние элементы ряда (например, $a[4]$ и $a[5]$) не упорядочены, они переставляются, после чего алгоритм предписывает перейти к сравнению следующей пары – $a[5]$ и $a[6]$. Вместо этого можно попытаться проследить за элементом $a[4]$, дав ему возможность «всплывать» в сторону меньших индексных позиций ряда. Для этого сравним $a[4]$ с $a[3]$. Если окажется, что $a[3]$ больше, осуществим перестановку и затем сравним $a[3]$ с $a[2]$ и т. д. Осуществить указанный метод.
9. В целочисленном массиве найти наибольшее число одинаковых элементов.
10. Дано n целых чисел. Сколько чисел лежит между данными a и b?
11. Дано n отрезков $[a[i], b[i]]$ на прямой ($i = 1...n$). Найти максимальное k, для которого существует точка прямой, покрытая k отрезками («максимальное число слоев»). Подсказка. Упорядочим все левые и правые концы отрезков вместе (при этом левый конец считается меньше правого конца, расположенного в той же

точке прямой). Далее двигаемся слева направо, считая число слоев. Встреченный левый конец увеличивает число слоев на 1, правый – уменьшает.

12. Дано n точек на плоскости. Указать (n – 1)-звенную несамопересекающуюся незамкнутую ломаную, проходящую через все эти точки. (Соседним отрезкам ломаной разрешается лежать на одной прямой.) Подсказка. Упорядочим точки по координате x, а при равных x-координатах – по y-координате. В таком порядке и можно проводить ломаную.

Лабораторная работа № 4

СОРТИРОВКА МЕТОДОМ ПРЯМОГО ВКЛЮЧЕНИЯ

Цель работы. Исследовать сортировку массива методом прямого включения и оценить эффективность этого алгоритма.

Общие сведения

Сортировка «методом прямого включения», так же как и сортировка «методом простого выбора», обычно применяется для массивов, не содержащих повторяющихся элементов.

Сортировка этим методом производится последовательно шаг за шагом. На k-м шаге считается, что часть массива, содержащая первые k – 1 элементов, уже упорядочена, то есть $a[1] <= a[2] <= ... <= a[k - 1]$. Далее необходимо взять k-й элемент и подобрать для него место в отсортированной части массива такое, чтобы после его вставки упорядоченность не нарушилась, то есть надо найти такое j ($1 <= j <= k - 1$), что $a[j] <= a[k] <= a[j + 1]$. Затем надо вставить элемент $a[k]$ на найденное место.

С каждым шагом отсортированная часть массива увеличивается. Для выполнения полной сортировки потребуется выполнить n – 1 шаг.

Осталось ответить на вопрос, как осуществить поиск подходящего места для элемента x. Поступим следующим образом: будем просматривать элементы, расположенные левее x (то есть те, которые

уже упорядочены), двигаясь к началу массива. Нужно просматривать элементы $a[j]$, j изменяется от $k - 1$ до 1. Такой просмотр закончится при выполнении одного из следующих условий:

- найден элемент $a[j] < x$, что говорит о необходимости вставки x между $a[j - 1]$ и $a[j]$;
- достигнут левый конец упорядоченной части массива, следовательно, нужно вставить x на первое место.

До тех пор, пока одно из этих условий не выполнится, будем смещать просматриваемые элементы на одну позицию вправо, в результате чего в отсортированной части будет освобождено место под x.

Пример

Коротко опишем фрагмент алгоритма сортировки с помощью прямого включения:

```
For k := 2 to n do
 Begin
      x := a[k]; j := k-1;
      { вставить x на подходящее место в a[1], ..., a[k] }
      { для этого организуем цикл, который выполняется, пока }
      { j > 0 и x <= a[j] }
      { в цикле смещаем элементы массива на одну позицию вправо }
      { по выходу из цикла вставляем x в позицию j + 1 массива }
 End;
```

Контрольное задание

Написать программу вставки последнего элемента массива после первого отрицательного элемента этого же массива.

Варианты заданий

Для всех заданий предварительно написать процедуру сортировки массива методом прямого включения.

1. Дан ряд, содержащий n элементов. Отсортировать их в порядке возрастания, отбрасывая при этом все повторяющиеся элементы.

2. Определить моду данного ряда – значение, встречающееся среди его элементов чаще всего.
3. Исходный набор данных представляет собой последовательность записей, состоящих из фамилии, возраста и стажа работы. Распечатать этот список:

1) в алфавитном порядке;
2) порядке увеличения возраста;
3) порядке увеличения стажа работы.

4. Написать процедуру сортировки по убыванию.
5. Дан ряд целых чисел. Получить в порядке возрастания все различные числа, входящие в этот ряд.
6. Дан ряд из n неповторяющихся целых чисел. Получить различные целые числа $i1, i2, ..., in$ такие, что $a[i1] < a[i2] < ... < a[in]$.
7. Даны целые $a1, a2, ..., an$. Найти наибольшее значение в этой последовательности после выбрасывания из нее всех членов со значением $\max(a1, a2, ..., an)$.
8. Даны натуральные $n, a1, ..., an$ ($n >= 4$). Числа $a1, ..., an$ – это измеренные в сотых долях секунды результаты n спортсменов в беге на 100 м. Составить команду из четырех лучших бегунов для участия в эстафете 4×100, т. е. указать одну из четверок натуральных чисел i, j, k, l такую, что сумма $ai + aj + ak + al$ будет иметь наименьшее значение.
9. Дано n независимых случайных точек, с координатами $(xi; yi)$, равномерно распределенных в круге радиуса 1 с центром в начале координат. Напишите программу, располагающую точки в порядке возрастания расстояния от центра.
10. Дано n целых положительных двузначных чисел. Трактуя каждое число как пару цифр из интервала 0 – 9, отсортировать их (цифры) по возрастанию.
11. Дано n точек на плоскости. Указать $(n – 1)$-звенную несамопересекающуюся замкнутую ломаную, проходящую через все эти точки. (Соседним отрезкам ломаной разрешается лежать на одной прямой.) Подсказка. Возьмем самую левую точку (т.е. точку с наименьшей координатой x) и проведем из нее лучи во все остальные точки. Теперь упорядочим эти лучи снизу вверх, а точки на одном луче упорядочим по расстоянию от начала луча (это делается для всех лучей, кроме нижнего и верхнего). Ломаная

выходит из выбранной (самой левой) точки по нижнему лучу, затем по всем остальным лучам (в описанном порядке) и возвращается по верхнему лучу.

12. Дано *n* точек на плоскости. Построить их выпуклую оболочку – минимальную выпуклую фигуру, их содержащую. (Резиновое колечко, натянутое на вбитые в доску гвозди, – их выпуклая оболочка.) Указание. Упорядочим точки. Затем, рассматривая точки по очереди, будем строить выпуклую оболочку уже рассмотренных точек.

Лабораторная работа № 5

БИНАРНЫЙ ПОИСК

Цель работы. Исследовать один из наиболее распространенных методов поиска данных – бинарный (двоичный) поиск.

Общие сведения

Пусть $a1$, $a2$, ..., an – целочисленный массив и $a1 < a2 < ... < an$; b – целое число. Рассмотрим задачу: выяснить, входит ли данное число в массив $k = \text{ceil}(\log2(n + 1))$ и если входит, то каково значение p, для которого $ap = b$? Эту задачу мы назовем задачей поиска элемента. Областью поиска служит n-элементный ряд, предварительно отсортированный. Вначале диапазон поиска определяется граничными значениями $low = 1$ и $high = n$. Строим первую «гипотезу», полагая, что искомое значение находится в позиции *test* – где-то посередине между границами *low* и *high*. Если проверка подтверждает это, то поиск успешно завершен. Если значение в позиции *test* меньше искомого, нижней границей (*low*) становится *test* + 1, а если больше, то модифицируется верхняя (*high*) граница – для нее принимается значение, равное *test* – 1. Описанный процесс постепенного сближения границ поиска с последующей проверкой среднего элемента повторяется многократно, завершаясь одним из двух исходов: либо нужное значение обнаруживается (успех), либо *Low* становится больше *high* (неудача).

Максимальное количество проверок, необходимое для завершения бинарного поиска (при любом исходе – удачном или неудачном), определяется по формуле, где ceil обозначает функцию, возвращающую ближайшее целое число, большее или равное ее аргументу. Другими словами, k есть наименьшее целое, такое, что 2^k равно или больше, чем $n + 1$. Например, если $n = 100$, для сортировки потребуется не более семи проверок, поскольку $2^7 = 128$.

Пример

Пусть $b = 6$, а массив a состоит из 10 элементов [3, 5, 6, 8, 12, 15, 17, 18, 20, 25].

Шаг 1. Найдем номер среднего элемента: $test = (1 + 10)/2 = 5$ ($low = 1$, $high = 10$). Так как $a[5] > 6$, то результат находится в левой части массива.

Шаг 2. Рассматриваем лишь первые четыре элемента массива; находим индекс среднего элемента этой части $test = (1 + 4)/2 = 2$ ($low = 1$, $high = 4$). $6 > a[2]$, следовательно, первый и второй элементы из рассмотрения исключаем.

Шаг 3. Рассматриваем два элемента; значение $test = (3 + 4)/2 = 3$ ($low = 3$, $high = 4$). $a[3] = 6$! Элемент найден, его номер – 3.

Контрольные вопросы

1. Как в Паскале разделить нацело два числа?
2. Есть ли в Паскале аналог функции ceil?
3. За сколько шагов наверняка завершится поиск элемента в массиве из 1000 элементов?
4. Когда число элементов ряда удваивается, насколько увеличивается максимально необходимое количество проверок?

Варианты заданий

Для всех вариантов предварительно написать процедуру поиска.

1. Опишите массив записей, содержащих фамилию абонента и номер его телефона. Запрограммируйте двоичный поиск в телефонном справочнике.
2. Индексом называется таблица, содержащая отсортированные значения некоторых ключей и их местоположение в массиве за-

писей. Индексом пользуются для ускорения поиска в массиве (сам массив может быть не отсортирован). Запрограммируйте процедуру составления индекса и:

1) последовательного поиска при помощи индекса;
2) бинарного поиска при помощи индекса.

3. Пусть задан в виде последовательности символов некоторый текст T, состоящий из слов, и есть два списка из нескольких слов в виде двух массивов A и B. Написать программу, преобразующую текст T в текст S путем замены каждого вхождения слова $A[i]$ на соответствующее слово $B[i]$.
4. В нашем примере поиск проводился в массиве, упорядоченном по возрастанию. Напишите процедуру бинарного поиска в массиве, упорядоченном по убыванию.
5. Дан массив из строк (например фамилий). Отсортировать его по алфавиту и написать процедуру вставки новой фамилии после заданной так, чтобы алфавитный порядок не нарушился. Предусмотреть ситуацию, когда массив заполнен «до отказа» и вставка нового элемента невозможна.
6. Имеется железнодорожное расписание, содержащее номер рейса поезда, время отправления, прибытия и станцию прибытия. Организовать поиск номера поезда, время отправления и прибытия, если задана станция.
7. Компания с целью определения спроса на свою продукцию организует некоторый опрос. Продукция – компакт-диски с записями шлягеров. Все опрашиваемые делятся на две группы в соответствии с полом. Каждый опрашиваемый должен назвать три песни, которые идентифицируются номерами от 1 до N (пусть $N = 10$). Файл с данными обрабатывается программой, которая должна печатать:
    1. Список песен в порядке их популярности. Каждая строка содержит название песни и число упоминаний ее при опросе. Песни, которые ни разу не упоминались, в список не включаются;
    2. Три наиболее популярные песни (хиты) и два списка (в соответствии с группами) с именами всех тех ответивших, которые поставили на первое место один из этих трех хитов.

Лабораторная работа № 6

МОДУЛИ

Цель работы. Освоить использование модулей в программах, написанных на языке Паскаль.

Общие положения

Модуль (UNIT) в Турбо Паскале – это особым образом оформленная библиотека подпрограмм. Модуль в отличие от программы не может быть запущен на выполнение самостоятельно, он может только участвовать в построении программ и других модулей.

Модули можно разделить на две группы:

- стандартные модули, содержащиеся в Турбо Паскале;
- модули, создаваемые разработчиком и включающие личные библиотеки процедур и функций.

Стандартные модули:

SYSTEM – содержит все процедуры и функции стандартного Паскаля и ряд дополнительных процедур (*GETDIR*). Этот модуль всегда подключается к программе.

PRINTER – делает доступным вывод на матричный принтер. Приведем фрагмент программы с использованием этого модуля.

```
Uses Printer
begin
      Writeln(LST, ‘Турбо Паскаль’)
end.
```

CRT – содержит процедуры и функции, которые управляют текстовым режимом работы экрана.

GRAPH – набор процедур и функций для управления графическим режимом.

DOS – процедуры и функции, открывающие доступ к средствам дисковой операционной системы MS-DOS.

TURBO3, GRAPH3 – обеспечивают совместимость с ранней версией 3.0 системы Турбо Паскаль.

Модули позволяют создавать и строить программы практически любой сложности.

Модуль в Турбо Паскале представляет собой отдельно хранимую и независимо компилируемую программную единицу.

В общем случае модуль – это совокупность программных ресурсов, предназначенных для использования другими программами. Под программными ресурсами понимаются любые элементы языка Турбо Паскаль: константы, типы, переменные, подпрограммы. Модуль сам по себе не является выполняемой программой, его элементы используются другими программными единицами.

В модуле можно выделить две группы программных элементов:

- программные элементы, предназначенные для использования другими программами или модулями, такие элементы называют видимыми вне модуля;
- программные элементы, необходимые только для работы самого модуля, их называют невидимыми или скрытыми.

В соответствии с этим модуль кроме заголовка содержит две основные части, называемые интерфейсом и реализацией.

В общем случае модуль имеет следующую структуру:

```
unit <имя модуля>; {заголовок модуля}
interface
        { описание видимых программных элементов модуля }
       { описание скрытых программных элементов модуля }
begin
        { операторы инициализации элементов модуля }
end.
```

В частном случае модуль может не содержать части реализации и части инициализации, тогда структура модуля будет такой:

```
unit <имя модуля>; {заголовок модуля}
interface
        { описание видимых программных элементов модуля }
implementation

end.
```

Использование в модулях процедур и функций имеет свои особенности. Заголовок подпрограммы содержит все сведения, необходимые для ее вызова: имя, перечень и тип параметров, тип результата для функций – эта информация должна быть доступна для других программ и модулей. Однако текст подпрограммы, реализующий ее алгоритм, другими программами и модулями не может быть использован. Поэтому заголовок процедур и функций помещают в интерфейсную часть модуля, а текст – в часть реализации.

Интерфейсная часть модуля содержит только видимые (доступные для других программ и модулей) заголовки процедур. Полный текст процедуры или функции помещают в часть реализации, причем заголовок может не содержать список формальных параметров.

Исходный текст модуля должен быть откомпилирован с помощью директивы Make подменю Compile и записан на диск. Результатом компиляции модуля является файл с расширением TPU (Turbo Pascal Unit). Основное имя модуля берется из заголовка модуля.

Для подключения модуля к программе необходимо указать его имя в разделе описания модулей, например:

uses CRT, Graph;

В том случае, если имена переменных в интерфейсной части модуля и в программе, использующей этот модуль, совпадают, обращение будет происходить к переменной, описанной в программе. Для обращения к переменной, описанной в модуле, необходимо применить составное имя, состоящее из имени модуля и имени переменной, отделенных друг от друга точкой.

Например, пусть имеется модуль, в котором описана переменная *K*:

```
unit M;
interface
        var K: Integer;
```

```
implementation
.....
end.
```

Пусть программа, использующая этот модуль, также содержит переменную *K*:

```
Program P;
    uses M;
    var K: Char;
begin
  .....
end.
```

Для того чтобы в программе *P* иметь доступ к переменной *K* из модуля *M*, необходимо задать составное имя *M.K*.

Использование составных имен применяется не только к именам переменных, а ко всем именам, описанным в интерфейсной части модуля.

Приведем пример модуля.

```
Unit Cmp;
Interface
type
       complex = record
                      re, im : real
       end;

var
       a, b, c : complex;

Procedure ADDC(x, y : complex; var z : real);
Procedure MulC(x, y : complex; var z : real);

{Исполняемая часть}

IMPLEMENTATION
```

```
Procedure ADDC;
begin
      z.re := x.re + y.re
end;
{Инициирующая часть}
begin
      a.re := 1;
      a.im:= 2;
end;
end.
```

Рекурсивное использование модулей запрещено.

Если в модуле имеется раздел инициализации, то операторы из этого раздела будут выполнены перед началом выполнения программы, в которой используется этот модуль.

Контрольные вопросы

1. Для чего нужны модули?
2. Назовите стандартные модули Турбо Паскаля.
3. Структура модуля.
4. Что включается в интерфейсную часть модуля?
5. Для чего служит инициирующая часть модуля?

Варианты заданий

1. Написать программу сортировки четных элементов массива с помощью простого выбора. Процедура сортировки расположена в модуле.
2. Написать программу сортировки положительных элементов массива с помощью простого выбора. Процедура сортировки расположена в модуле.
3. Написать программу сортировки отрицательных элементов массива с помощью простого выбора. Процедура сортировки расположена в модуле.
4. Написать программу сортировки столбцов матрицы по первой ее строке $n*m$ в порядке возрастания. Процедура сортировки расположена в модуле.

5. Сортировка методом простого обмена. Написать программу подсчета количества выполненных сравнений и перестановок в лучшем, худшем случаях и в среднем. Процедура должна быть расположена в модуле.

6. Сортировка методом простого обмена. Упорядочить первые *n* элементов данного ряда в порядке возрастания. Напечатать эти элементы в порядке убывания. Процедуры сортировки и печати должны содержаться в модуле.

7. Написать программу сортировки, в модуле которой содержится процедура "пузырьковой" сортировки по убыванию.

8. Написать программу поиска в массиве чисел второго по величине (после наименьшего) элемента. Процедура поиска расположена в модуле.

9. Написать программу, в которой необходимо в целочисленном массиве найти наибольшее число одинаковых элементов. Процедуры и функции должны располагаться в модуле.

10. Сортировка методом прямого включения. Дан ряд, содержащий *n* элементов. Отсортировать их в порядке возрастания, отбрасывая при этом все повторяющиеся элементы. Процедуры и функции программы должны располагаться в модуле.

11. Сортировка методом прямого включения. Написать программу сортировки по убыванию. Процедура сортировки должна располагаться в модуле.

12. Опишите массив записей, содержащих фамилию абонента и номер его телефона. Запрограммируйте двоичный поиск в телефонном справочнике. Процедура поиска должна располагаться в модуле программы.

13. Напишите программу бинарного поиска в массиве, упорядоченном по убыванию. Процедура поиска должна располагаться в модуле.

14. Ввести последовательность чисел (окончание ввода – 0) и вывести их в обратной последовательности. Входные данные взять из текстового файла. Процедуры и функции должны располагаться в модуле программы.

15. Напишите программу с рекурсивной функцией, которая возвращает среднее из *n* элементов массива чисел. Функция расположена в модуле.

16. Напишите программу с функцией сложения двух чисел, используя только прибавление единицы. Функция расположена в модуле.

17. Написать функцию умножения двух чисел, используя только операцию сложения. Функция расположена в модуле программы.

18. Вычислить сумму элементов одномерного массива. Функция вычисления суммы должна быть расположена в модуле программы.

19. Подсчитать количество цифр в заданном числе. Функция должна быть расположена в модуле.

20. Написать функцию, проверяющую правильность имени в языке Паскаль. Функция расположена в модуле.

21. В текстовом файле задана последовательность положительных чисел, за которой следует отрицательное число. Написать функцию без параметров для нахождения суммы этих положительных чисел. Функция расположена в модуле.

22. Написать функцию без параметров, которая подсчитывает количество цифр в тексте, заданном в текстовом файле (за текстом следует точка). Функция расположена в модуле.

23. Написать программу перевода числа из десятичной системы счисления в двоичную. Процедура расположена в модуле.

24. Написать программу поиска в массиве чисел третьего по величине (после наименьшего) элемента. Процедура поиска расположена в модуле.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Галкин, А.А. Программирование и основы алгоритмизации в примерах и задачах : учеб. пособие / А.А. Галкин и [др.] ; под ред. А.А. Галкина ; Владим. гос. ун-т. – Владимир, 2001. – 148 с. – ISBN 5-89368-236-X.
2. Фаронов, В.В. Турбо Паскаль 7.0. Начальный курс : учеб. пособие / В.В. Фаронов. – М. : Нолидж, 1997. – 616 с. – ISBN 5-89251-012-3.
3. Баранов, С.И. Синтез микропрограммных автоматов (граф-схемы и автоматы) / С.И. Баранов. – 2-е изд., стер. – Л. : Энергия, Ленингр. отд-ние, 1979. – 232 с.
4. Культин, Н. Delphi 4. Программирование на Object Pascal / Н. Культин. – СПб. : БХВ, 1999. – 480 с. – ISBN 5-8206-0041-X.

ОГЛАВЛЕНИЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К ЛАБОРАТОРНЫМ РАБОТАМ ПО ДИСЦИПЛИНЕ
«ПРОГРАММИРОВАНИЕ НА ЯЗЫКАХ ВЫСОКОГО УРОВНЯ»

Составители
ГРИШИН Владимир Семенович
ГУСЕВ Дмитрий Игоревич

Ответственный за выпуск – зав. кафедрой профессор А.В. Костров

Подписано в печать 02.06.09.
Формат 60x84/16. Усл. печ. л. 2,09. Тираж 100 экз.
Заказ
Издательство
Владимирского государственного университета.
600000, Владимир, ул. Горького, 87.