Министерство образования и науки Российской Федерации

Государственное образовательное учреждение
профессионального образования Российской Федерации
«Ростовский государственный университет»

М. Э. Абрамян

1000 ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ

Часть III

Текстовые файлы, составные типы данных в процедурах и функциях, рекурсия, указатели и динамические структуры

Методические указания для студентов механико-математического, физического и экономического факультетов

Ростов-на-Дону
2004

Печатается по решению
кафедры алгебры и дискретной математики
механико-математического факультета РГУ
от 14 июня 2004 г. (протокол № 10)

Рецензенты:
к. ф.-м. н., доцент Столяр А. М.,
к. ф.-м. н., доцент Чечин Г. М.,
ст. преп. Мачулина Л. А.

Аннотация

Третья часть сборника учебных заданий по программированию содержит задания, посвященные обработке текстовых файлов, использованию сложных типов данных в процедурах и функциях, рекурсивным алгоритмам, а также указателям и динамическим структурам данных.

Задания формулируются таким образом, что их можно использовать при изучении любого из распространенных языков программирования, в частности, Pascal, C++, Basic.

Сборник предназначен для студентов механико-математического, физического и экономического факультетов.

Автор: М. Э. Абрамян.

17 Текстовые файлы: группа Text

Условие «дан текстовый файл» означает, что в наборе исходных данных указано *имя* данного файла (текстовая строка). Все исходные файлы в заданиях данной группы считаются существующими.

Если в задании требуется создать новый файл, то имя создаваемого файла также входит в набор исходных данных (и, как правило, является последним элементом этого набора).

Максимальный размер исходных файлов не устанавливается, поэтому при решении заданий не следует использовать вспомогательные массивы, содержащие все элементы исходных файлов, однако допускается использование *вспомогательных файлов*.

Используемые в заданиях двоичные файлы удовлетворяют условиям, которые перечислены в начале раздела «Двоичные (типизированные) файлы».

17.1 Основные операции с текстовыми файлами

Text1º. Дано имя файла и целые положительные числа N и K. Создать текстовый файл с указанным именем и записать в него N строк, каждая из которых состоит из K символов «*» (звездочка).

Text2. Дано имя файла и целое число N ($0 < N < 27$). Создать текстовый файл с указанным именем и записать в него N строк: первая строка должна содержать *строчную* (то есть маленькую) латинскую букву «a», вторая — буквы «ab», третья — буквы «abc» и т. д.; последняя строка должна содержать N начальных строчных латинских букв в алфавитном порядке.

Text3. Дано имя файла и целое число N ($0 < N < 27$). Создать текстовый файл с указанным именем и записать в него N строк длины N; строка с номером K ($K = 1, \ldots, N$) должна содержать K начальных *прописных* (то есть заглавных) латинских букв, дополненных справа символами «*» (звездочка). Например, для $N = 4$ файл должен содержать строки «A***», «AB**», «ABC*», «ABCD».

Text4º. Дан текстовый файл. Вывести количество содержащихся в нем символов и строк (маркеры концов строк EOLN и конца файла EOF при подсчете количества символов не учитывать).

Text5. Дана строка S и текстовый файл. Добавить строку S в конец файла.

Text6. Даны два текстовых файла. Добавить в конец первого файла содержимое второго файла.

Text7. Дана строка S и текстовый файл. Добавить строку S в начало файла.

Text8. Даны два текстовых файла. Добавить в начало первого файла содержимое второго файла.

Text9. Дано целое число K и текстовый файл. Вставить пустую строку перед строкой файла с номером K. Если строки с таким номером нет, то оставить файл без изменений.

Text10. Дано целое число K и текстовый файл. Вставить пустую строку после строки файла с номером K. Если строки с таким номером нет, то оставить файл без изменений.

Text11. Дан текстовый файл. Продублировать в нем все пустые строки.

Text12. Дана строка S и текстовый файл. Заменить в файле все пустые строки на строку S.

Text13. Дан непустой текстовый файл. Удалить из него первую строку.

Text14. Дан непустой текстовый файл. Удалить из него последнюю строку.

Text15. Дано целое число K и текстовый файл. Удалить из файла строку с номером K. Если строки с таким номером нет, то оставить файл без изменений.

Text16. Дан текстовый файл. Удалить из него все пустые строки.

Text17. Даны два текстовых файла. Добавить в конец каждой строки первого файла соответствующую строку второго файла. Если второй файл короче первого, то оставшиеся строки первого файла не изменять.

Text18. Дано целое число K и текстовый файл. Удалить из каждой строки файла первые K символов (если длина строки меньше K, то удалить из нее все символы).

Text19. Дан текстовый файл. Заменить в нем все прописные русские буквы на строчные, а все строчные — на прописные.

Text20. Дан текстовый файл. Заменить в нем все подряд идущие пробелы на один пробел.

Text21°. Дан текстовый файл, содержащий более трех строк. Удалить из него последние три строки.

Text22. Дано целое число K ($0 < K < 10$) и текстовый файл, содержащий более K строк. Удалить из файла последние K строк.

Text23. Дано целое число K ($0 < K < 10$) и текстовый файл, содержащий более K строк. Создать новый текстовый файл, содержащий K последних строк исходного файла.

17.2 Анализ и форматирование текста

Text24. Дан текстовый файл. Найти количество абзацев в тексте, если абзацы отделяются друг от друга одной или несколькими пустыми строками.

Text25. Дано целое число K и текстовый файл. Удалить из файла абзац с номером K (абзацы отделяются друг от друга одной или несколькими пустыми строками). Пустые строки, предшествующие и следующие за удаляемым

абзацем, не удалять. Если абзац с данным номером отсутствует, то оставить файл без изменений.

Text26. Дан текстовый файл. Найти количество абзацев в тексте, если первая строка каждого абзаца начинается с 5 пробелов («*красная строка*»). Пустые строки между абзацами не учитывать.

Text27. Дано целое число *K* и текстовый файл. Удалить из файла абзац с номером *K* (абзацы выделяются с помощью *красной строки* — см. задание Text26). Пустые строки между абзацами не учитывать и не удалять. Если абзац с данным номером отсутствует, то оставить файл без изменений.

Text28. Дан текстовый файл. Абзацы выделяются в нем с помощью *красной строки* (см. задание Text26), а пустых строк нет. Вставить между соседними абзацами по одной пустой строке (в начало и конец файла пустые строки не добавлять).

Text29. Дан текстовый файл. Вывести первое слово текста наибольшей длины. *Словом* считать набор символов, не содержащий пробелов и ограниченный пробелами или началом/концом строки.

Text30. Дан текстовый файл. Вывести последнее слово текста наименьшей длины. *Словом* считать набор символов, не содержащий пробелов и ограниченный пробелами или началом/концом строки.

Text31. Дано целое число *K* и текстовый файл. Создать строковый файл и записать в него все слова длины *K* из исходного файла. *Словом* считать набор символов, не содержащий пробелов, знаков препинания и ограниченный пробелами, знаками препинания или началом/концом строки. Если исходный файл не содержит слов длины *K*, то оставить результирующий файл пустым.

Text32. Дан символ *C* — *прописная* (заглавная) русская буква и текстовый файл. Создать строковый файл и записать в него все слова из исходного файла, начинающиеся на эту букву (прописную или строчную). *Словом* считать набор символов, не содержащий пробелов, знаков препинания и ограниченный пробелами, знаками препинания или началом/концом строки. Если исходный файл не содержит подходящих слов, то оставить результирующий файл пустым.

Text33. Дан символ *C* — *строчная* (маленькая) русская буква и текстовый файл. Создать строковый файл и записать в него все слова из исходного файла, содержащие хотя бы одну букву *C* (прописную или строчную). *Словом* считать набор символов, не содержащий пробелов, знаков препинания и ограниченный пробелами, знаками препинания или началом/концом строки. Если исходный файл не содержит подходящих слов, то оставить результирующий файл пустым.

Text34. Дан текстовый файл, содержащий текст, выровненный по левому краю. Выровнять текст по правому краю, добавив в начало каждой непустой строки нужное количество пробелов (ширину текста считать равной 50).

Text35. Дан текстовый файл, содержащий текст, выровненный по левому краю. Выровнять текст по центру, добавив в начало каждой непустой строки нужное количество пробелов (ширину текста считать равной 50). Строки нечетной длины перед центрированием дополнять слева пробелом.

Text36. Дан текстовый файл, содержащий текст, выровненный по правому краю. Выровнять текст по центру, удалив из каждой непустой строки половину начальных пробелов. В строках с нечетным количеством начальных пробелов перед центрированием удалять первый начальный пробел.

Text37. Дан текстовый файл, содержащий текст, выровненный по левому краю. Абзацы текста разделяются одной пустой строкой. Выровнять текст *по ширине* (то есть и по левому, и по правому краю), увеличив в каждой непустой строке (кроме последних строк абзацев) количество пробелов между словами, начиная с последнего пробела в строке (ширину текста считать равной 50).

Text38. Дано целое число K (> 25) и текстовый файл, содержащий текст, выровненный по левому краю. Абзацы текста отделяются друг от друга одной пустой строкой. Отформатировать текст так, чтобы его ширина не превосходила K позиций, и выровнять текст по левому краю, сохранив деление на абзацы. Пробелы в конце строк удалить. Сохранить отформатированный текст в новом текстовом файле.

Text39. Дано целое число K (> 25) и текстовый файл, содержащий текст, выровненный по левому краю. Абзацы выделяются в нем с помощью *красной строки* (5 начальных пробелов), а пустых строк нет. Отформатировать текст так, чтобы его ширина не превосходила K позиций, и выровнять текст по левому краю, сохранив деление на абзацы. Пробелы в конце строк удалить. Сохранить отформатированный текст в новом текстовом файле.

17.3 Текстовые файлы с числовой информацией

В заданиях на обработку текстовых файлов с числовой информацией предполагается, что изображения вещественных чисел, содержащиеся в текстовых файлах, удовлетворяют стандартным правилам используемого языка программирования (в частности, дробная часть отделяется от целой части десятичной *точкой*).

Text40. Даны два файла целых чисел одинакового размера. Создать текстовый файл, содержащий эти числа, расположенные в два столбца шириной по 30 символов (в первом столбце содержатся числа из первого исходного файла, во втором — из второго файла). В начало и конец каждой строки тек-

стового файла добавить разделитель «|» (код 124). Числа выравниваются по правому краю столбца.

Text41. Даны три файла целых чисел одинакового размера. Создать текстовый файл, содержащий эти числа, расположенные в три столбца шириной по 20 символов (в каждом столбце содержатся числа из соответствующего исходного файла). В начало и конец каждой строки текстового файла добавить разделитель «|» (код 124). Числа выравниваются по левому краю столбца.

Text42. Даны вещественные числа A, B и целое число N. Создать текстовый файл, содержащий таблицу значений функции $\sqrt{x}$ на промежутке [A, B] с шагом $(B - A)/N$. Таблица состоит из двух столбцов: с аргументами x (10 позиций, из них 4 под дробную часть) и со значениями $\sqrt{x}$ (15 позиций, из них 8 под дробную часть). Столбцы выравниваются по правому краю.

Text43. Даны вещественные числа A, B и целое число N. Создать текстовый файл, содержащий таблицу значений функций $\sin(x)$ и $\cos(x)$ на промежутке [A, B] с шагом $(B - A)/N$. Таблица состоит из трех столбцов: с аргументами x (8 позиций, из них 4 под дробную часть) и со значениями $\sin(x)$ и $\cos(x)$ (по 12 позиций, из них 8 под дробную часть). Столбцы выравниваются по правому краю.

Text44. Дан текстовый файл, каждая строка которого изображает целое число, дополненное слева и справа несколькими пробелами. Вывести количество этих чисел и их сумму.

Text45. Дан текстовый файл, каждая строка которого изображает целое или вещественное число, дополненное слева и справа несколькими пробелами (вещественные числа имеют ненулевую дробную часть). Вывести количество чисел с ненулевой дробной частью и их сумму.

Text46. Дан текстовый файл, каждая строка которого содержит изображения нескольких чисел, разделенные пробелами (вещественные числа имеют ненулевую дробную часть). Создать файл вещественных чисел, содержащий (в том же порядке) все числа из исходного файла, имеющие ненулевую дробную часть.

Text47. Дан текстовый файл, каждая строка которого изображает целое или вещественное число, дополненное слева и справа несколькими пробелами (вещественные числа имеют ненулевую дробную часть). Вывести количество целых чисел и их сумму.

Text48. Дан текстовый файл, каждая строка которого содержит изображения нескольких чисел, разделенные пробелами (вещественные числа имеют ненулевую дробную часть). Создать файл целых чисел, содержащий все целые числа из исходного файла (в том же порядке).

Text49. Дан текстовый файл и файл целых чисел. Добавить в конец каждой строки текстового файла изображение соответствующего числа из файла целых чисел. Если файл целых чисел короче текстового файла, то оставшиеся строки текстового файла не изменять.

Text50. Дан текстовый файл. В каждой его строке первые 30 позиций отводятся под текст, а оставшаяся часть — под вещественное число. Создать два файла: строковый файл, содержащий текстовую часть исходного файла, и файл вещественных чисел, содержащий числа из исходного файла (в том же порядке).

Text51. Дан текстовый файл, содержащий таблицу из трех столбцов вещественных чисел. Ширина столбцов таблицы и способ их выравнивания являются произвольными, специальных символов-разделителей таблица не содержит. Создать три файла вещественных чисел, каждый из которых содержит числа из соответствующего столбца таблицы (в том же порядке).

Text52. Дан текстовый файл, содержащий таблицу из трех столбцов целых чисел. В начале и в конце каждой строки таблицы, а также между ее столбцами располагается *символ-разделитель*. Ширина столбцов таблицы, способ их выравнивания и вид символа-разделителя являются произвольными. Создать файл целых чисел, содержащий сумму чисел из каждой строки исходной таблицы.

17.4 Дополнительные задания на обработку текстовых файлов

Text53. Дан текстовый файл. Создать символьный файл, содержащий все знаки препинания, встретившиеся в текстовом файле (в том же порядке).

Text54. Дан текстовый файл. Создать символьный файл, содержащий все символы, встретившиеся в тексте, включая пробел и знаки препинания (без повторений). Символы располагать в порядке их первого появления в тексте.

Text55. Дан текстовый файл. Создать символьный файл, содержащий все символы, встретившиеся в тексте, включая пробел и знаки препинания (без повторений). Символы располагать в порядке возрастания их кодов.

Text56. Дан текстовый файл. Создать символьный файл, содержащий все символы, встретившиеся в тексте, включая пробел и знаки препинания (без повторений). Символы располагать в порядке убывания их кодов.

Text57. Дан текстовый файл. Подсчитать число появлений в нем каждой *строчной* (то есть маленькой) русской буквы и создать строковый файл, элементы которого имеют вид «<буква>–<число ее появлений>» (например, «а–25»). Буквы, отсутствующие в тексте, в файл не включать. Строки упорядочить по возрастанию кодов букв.

Text58. Дан текстовый файл. Подсчитать число появлений в нем каждой *строчной* (то есть маленькой) русской буквы и создать строковый файл, элементы которого имеют вид «<буква>–<число ее появлений>» (например, «а–25»). Буквы, отсутствующие в тексте, в файл не включать. Строки упорядочить по убыванию числа появлений букв, а при равном числе появлений — по возрастанию кодов букв.

Text59. Дана строка S, состоящая из 10 цифр, и файл с русским текстом. Зашифровать файл, выполнив циклическую замену каждой русской буквы, стоящей на K-й позиции строки, на букву того же регистра, расположенную в алфавите на S_K-м месте после шифруемой буквы (для $K = 11$ снова используется смещение S_1 и т. д.). Букву «ё» в алфавите не учитывать, знаки препинания и пробелы не изменять.

Text60. Дана строка и файл с русским текстом, зашифрованным по правилу, описанному в задании Text59. Данная строка представляет собой первую расшифрованную строку текста. Расшифровать остальные строки и заменить в файле зашифрованный текст на расшифрованный. Если информации для расшифровки недостаточно, то исходный файл не изменять.

18 Составные типы данных в процедурах и функциях: группа Param

В каждом задании данного раздела требуется описать процедуру или функцию и затем использовать ее для обработки исходных данных.

Все параметры любой *функции* считаются входными. Для *процедур* всегда указывается, какие параметры являются выходными (или одновременно входными и выходными); если о виде параметра процедуры ничего не сказано, то он считается входным.

18.1 Одномерные и двумерные массивы

При вводе исходного массива вначале следует ввести его размер (одно число для одномерных массивов, два числа — количество строк и столбцов — для двумерных массивов-матриц), а затем — все его элементы.

Если в задании явно не указывается размер одномерного массива, являющегося параметром процедуры или функции, то предполагается, что этот размер может меняться в пределах от 1 до 10. Для двумерных массивов-матриц также предполагается, что число их строк и столбцов может меняться от 1 до 10. Порядковые номера начальных элементов как одномерных, так и двумерных массивов всегда считаются равными 1.

При описании процедур, выполняющих преобразование массива, не следует использовать вспомогательный массив того же размера.

Param1º. Описать функцию MinInt(A, N) целого типа, находящую минимальный элемент целочисленного массива A размера N. С помощью этой функции найти минимальные элементы массивов A, B, C размера N_A, N_B, N_C соответственно.

Param2. Описать функцию NMax(A, N) целого типа, находящую номер максимального элемента вещественного массива A размера N. С помощью этой функции найти номера максимальных элементов массивов A, B, C размера N_A, N_B, N_C соответственно.

Param3. Описать процедуру NMinmax(A, N, *NMin*, *NMax*), находящую номера минимального и максимального элемента вещественного массива A размера N. Выходные параметры целого типа: *NMin* (номер минимального элемента) и *NMax* (номер максимального элемента). С помощью этой процедуры найти номера минимальных и максимальных элементов массивов A, B, C размера N_A, N_B, N_C соответственно.

Param4. Описать процедуру Invert(A, N), меняющую порядок следования элементов вещественного массива A размера N на противоположный (*инвертирование* массива). Массив A является входным и выходным параметром. С помощью этой процедуры инвертировать массивы A, B, C размера N_A, N_B, N_C соответственно.

Param5. Описать процедуру Smooth1(A, N), выполняющую *сглаживание* вещественного массива A размера N следующим образом: элемент A_K заменяется на среднее арифметическое первых K исходных элементов массива A. Массив A является входным и выходным параметром. С помощью этой процедуры выполнить пятикратное сглаживание данного массива A размера N, выводя результаты каждого сглаживания.

Param6. Описать процедуру Smooth2(A, N), выполняющую *сглаживание* вещественного массива A размера N следующим образом: элемент A_1 не изменяется, элемент A_K ($K = 2, \ldots, N$) заменяется на полусумму исходных элементов A_{K-1} и A_K. Массив A является входным и выходным параметром. С помощью этой процедуры выполнить пятикратное сглаживание данного массива A размера N, выводя результаты каждого сглаживания.

Param7. Описать процедуру Smooth3(A, N), выполняющую *сглаживание* вещественного массива A размера N следующим образом: каждый элемент массива заменяется на его среднее арифметическое с соседними элементами (при вычислении среднего арифметического используются *исходные* значения соседних элементов). Массив A является входным и выходным параметром. С помощью этой процедуры выполнить пятикратное сглаживание данного массива A размера N, выводя результаты каждого сглаживания.

Param8. Описать процедуру RemoveX(A, N, X), удаляющую из целочисленного массива A размера N элементы, равные целому числу X. Массив A и число

N являются входными и выходными параметрами. С помощью этой процедуры удалить числа X_A, X_B, X_C из массивов A, B, C размера N_A, N_B, N_C соответственно и вывести размер и содержимое полученных массивов.

Param9. Описать процедуру RemoveForInc(A, N), удаляющую из вещественного массива A размера N «лишние» элементы так, чтобы оставшиеся элементы оказались упорядоченными по возрастанию: первый элемент не удаляется, второй элемент удаляется, если он меньше первого, третий — если он меньше предыдущего элемента, оставленного в массиве, и т. д. Например, массив 5.5, 2.5, 4.6, 7.2, 5.8, 9.4 должен быть преобразован к виду 5.5, 7.2, 9.4. Массив A и число N являются входными и выходными параметрами. С помощью этой процедуры преобразовать массивы A, B, C размера N_A, N_B, N_C соответственно и вывести размер и содержимое полученных массивов.

Param10. Описать процедуру DoubleX(A, N, X), дублирующую в целочисленном массиве A размера N элементы, равные целому числу X. Массив A и число N являются входными и выходными параметрами. С помощью этой процедуры продублировать числа X_A, X_B, X_C в массивах A, B, C размера N_A, N_B, N_C соответственно и вывести размер и содержимое полученных массивов.

Param11. Описать процедуру Sort(A, N), выполняющую сортировку по возрастанию вещественного массива A размера N. Массив A является входным и выходным параметром. С помощью этой процедуры отсортировать массивы A, B, C размера N_A, N_B, N_C соответственно.

Param12. Описать процедуру SortIndex(A, N, I), формирующую для вещественного массива A размера N *индексный массив* I — массив целых чисел того же размера, содержащий номера элементов массива A в том порядке, который соответствует возрастанию элементов массива A (сам массив A при этом не изменяется). Индексный массив I является выходным параметром. С помощью этой процедуры создать индексные массивы для массивов A, B, C размера N_A, N_B, N_C соответственно.

Param13. Описать процедуру Bell(A, N), меняющую порядок элементов вещественного массива A размера N на следующий: наименьший элемент массива располагается на первом месте, наименьший из оставшихся элементов — на последнем, следующий по величине располагается на втором месте, следующий — на предпоследнем и т. д. (в результате график значений элементов будет напоминать *колокол*). Массив A является входным и выходным параметром. С помощью этой процедуры преобразовать массивы A, B, C размера N_A, N_B, N_C соответственно.

Param14. Описать процедуру Split1(A, N_A, B, N_B, C, N_C), формирующую по вещественному массиву A размера N_A два вещественных массива B и C размера N_B и N_C соответственно; при этом массив B содержит все элементы

массива A с нечетными порядковыми номерами (1, 3, …), а массив C — все элементы массива A с четными номерами (2, 4, …). Массивы B и C и числа N_B и N_C являются выходными параметрами. Применить эту процедуру к данному массиву A размера N_A и вывести размер и содержимое полученных массивов B и C.

Param15. Описать процедуру Split2(A, N_A, B, N_B, C, N_C), формирующую по целочисленному массиву A размера N_A два целочисленных массива B и C размера N_B и N_C соответственно; при этом массив B содержит все четные числа из массива A, а массив C — все нечетные числа (в том же порядке). Массивы B и C и числа N_B и N_C являются выходными параметрами. Применить эту процедуру к данному массиву A размера N_A и вывести размер и содержимое полученных массивов B и C.

Param16. Описать процедуру ArrayToMatrix1(A, K, M, N, B), формирующую по вещественному массиву A размера K матрицу B размера $M \times N$ (матрица заполняется элементами массива A по строкам). «Лишние» элементы массива игнорируются; если элементов массива недостаточно, то оставшиеся элементы матрицы полагаются равными 0. Двумерный массив B является выходным параметром. С помощью этой процедуры на основе данного массива A размера K и целых чисел M и N сформировать матрицу B размера $M \times N$.

Param17. Описать процедуру ArrayToMatrix2(A, K, M, N, B), формирующую по вещественному массиву A размера K матрицу B размера $M \times N$ (матрица заполняется элементами массива A по столбцам). «Лишние» элементы массива игнорируются; если элементов массива недостаточно, то оставшиеся элементы матрицы полагаются равными 0. Двумерный массив B является выходным параметром. С помощью этой процедуры на основе данного массива A размера K и целых чисел M и N сформировать матрицу B размера $M \times N$.

Param18. Описать процедуру Chessboard(M, N, A), формирующую по целым положительным числам M и N матрицу A размера $M \times N$, которая содержит числа 0 и 1, расположенные в «шахматном» порядке, причем $A_{1,1} = 0$. Двумерный целочисленный массив A является выходным параметром. С помощью этой процедуры по данным целым числам M и N сформировать матрицу A размера $M \times N$.

Param19. Описать функцию Norm1(A, M, N) вещественного типа, вычисляющую *норму* вещественной матрицы A размера $M \times N$:

$$\text{Norm1}(A, M, N) = \max \{|A_{1,J}| + |A_{2,J}| + \ldots + |A_{M,J}|\},$$

где максимум берется по всем J от 1 до N. Для данной матрицы A размера $M \times N$ найти Norm1(A, K, N), $K = 1, \ldots, M$.

Param20. Описать функцию Norm2(A, M, N) вещественного типа, вычисляющую *норму* вещественной матрицы A размера $M \times N$:

$$\text{Norm2}(A, M, N) = \max \{|A_{I,1}| + |A_{I,2}| + \ldots + |A_{I,N}|\},$$

где максимум берется по всем I от 1 до M. Для данной матрицы A размера $M \times N$ найти Norm2(A, K, N), $K = 1, \ldots, M$.

Param21. Описать функцию SumRow(A, M, N, K) вещественного типа, вычисляющую сумму элементов вещественной матрицы A размера $M \times N$, расположенных в K-й строке (если $K > M$, то функция возвращает 0). Для данной матрицы A размера $M \times N$ и трех данных K найти SumRow(A, M, N, K).

Param22. Описать функцию SumCol(A, M, N, K) вещественного типа, вычисляющую сумму элементов вещественной матрицы A размера $M \times N$, расположенных в K-м столбце (если $K > N$, то функция возвращает 0). Для данной матрицы A размера $M \times N$ и трех данных K найти SumCol(A, M, N, K).

Param23. Описать процедуру SwapRow(A, M, N, K_1, K_2), осуществляющую перемену местами строк вещественной матрицы A размера $M \times N$ с номерами K_1 и K_2. Матрица A является входным и выходным параметром; если K_1 или K_2 больше M, то матрица не изменяется. Используя эту процедуру, поменять для данной матрицы A размера $M \times N$ строки с данными номерами K_1 и K_2.

Param24. Описать процедуру SwapCol(A, M, N, K_1, K_2), осуществляющую перемену местами столбцов вещественной матрицы A размера $M \times N$ с номерами K_1 и K_2. Матрица A является входным и выходным параметром; если K_1 или K_2 больше N, то матрица не изменяется. Используя эту процедуру, поменять для данной матрицы A размера $M \times N$ столбцы с данными номерами K_1 и K_2.

Param25. Описать процедуру Transp(A, M), выполняющую *транспонирование* (то есть зеркальное отражение относительно главной диагонали) квадратной вещественной матрицы A порядка M. Матрица A является входным и выходным параметром. Используя эту процедуру, транспонировать данную матрицу A порядка M.

Param26. Описать процедуру RemoveRows(A, M, N, K_1, K_2), удаляющую из вещественной матрицы A размера $M \times N$ строки с номерами от K_1 до K_2 включительно (предполагается, что $1 < K_1 \le K_2$). Если $K_1 > M$, то матрица не изменяется; если $K_2 > M$, то удаляются строки матрицы с номерами от K_1 до M. Двумерный массив A и числа M, N являются входными и выходными параметрами. Используя процедуру RemoveRows, удалить из данной матрицы A размера $M \times N$ строки с номерами от K_1 до K_2 и вывести размер полученной матрицы и ее элементы.

Param27. Описать процедуру RemoveCols(A, M, N, K_1, K_2), удаляющую из вещественной матрицы A размера $M \times N$ столбцы с номерами от K_1 до K_2

включительно (предполагается, что $1 < K_1 \le K_2$). Если $K_1 > N$, то матрица не изменяется; если $K_2 > N$, то удаляются столбцы матрицы с номерами от K_1 до N. Двумерный массив A и числа M, N являются входными и выходными параметрами. Используя процедуру RemoveCols, удалить из данной матрицы A размера $M \times N$ столбцы с номерами от K_1 до K_2 и вывести размер полученной матрицы и ее элементы.

Param28. Описать процедуру RemoveRowCol(A, M, N, K, L), удаляющую из вещественной матрицы A размера $M \times N$ строку и столбец, которые содержат элемент $A_{K,L}$ (предполагается, что $M > 1$ и $N > 1$; если $K > M$ или $L > N$, то матрица не изменяется). Двумерный массив A и числа M, N являются входными и выходными параметрами. Дана матрица A размера $M \times N$ и числа K, L. Применить к матрице A процедуру RemoveRowCol и вывести размер полученной матрицы и ее элементы.

Param29. Описать процедуру SortCols(A, M, N), выполняющую сортировку по возрастанию столбцов целочисленной матрицы A размера $M \times N$ (столбцы сравниваются *лексикографически*: если первые элементы столбцов различны, то меньшим считается столбец, содержащий меньший первый элемент; если первые элементы столбцов равны, то анализируются их вторые элементы и т. д.). Двумерный массив A является входным и выходным параметром. Используя процедуру SortCols, отсортировать столбцы данной матрицы A размера $M \times N$.

18.2 Строки

Param30. Описать функцию IsIdent(S) целого типа, проверяющую, является ли строка S допустимым *идентификатором*, то есть непустой строкой, которая содержит только латинские буквы, цифры и символ подчеркивания «_» и не начинается с цифры. Если S является допустимым идентификатором, то функция возвращает 0. Если S является пустой строкой, то возвращается –1, если S начинается с цифры, то возвращается –2. Если S содержит недопустимые символы, то возвращается номер первого недопустимого символа. Проверить с помощью функции IsIdent пять данных строк.

Param31. Описать функцию FillStr(S, N) строкового типа, возвращающую строку длины N, заполненную повторяющимися копиями строки-*шаблона* S (последняя копия строки-шаблона может входить в результирующую строку частично). Используя эту функцию, сформировать по данному числу N и пяти данным строкам-шаблонам пять результирующих строк длины N.

Param32. Описать процедуру UpCaseRus(S), преобразующую все строчные русские буквы строки S в прописные (остальные символы строки S не изменяются). Строка S является входным и выходным параметром. Используя процедуру UpCaseRus, преобразовать пять данных строк.

Param33. Описать процедуру LowCaseRus(*S*), преобразующую все прописные русские буквы строки *S* в строчные (остальные символы строки *S* не изменяются). Строка *S* является входным и выходным параметром. Используя процедуру LowCaseRus, преобразовать пять данных строк.

Param34. Описать процедуру TrimLeftC(*S*, *C*), удаляющую в строке *S* начальные символы, совпадающие с символом *C*. Строка *S* является входным и выходным параметром. Дан символ *C* и пять строк. Используя процедуру TrimLeftC, преобразовать данные строки.

Param35. Описать процедуру TrimRightC(*S*, *C*), удаляющую в строке *S* конечные символы, совпадающие с символом *C*. Строка *S* является входным и выходным параметром. Дан символ *C* и пять строк. Используя процедуру TrimRightC, преобразовать данные строки.

Param36. Описать функцию InvertStr(*S*, *K*, *N*) строкового типа, возвращающую *инвертированную подстроку* строки *S*, содержащую в обратном порядке *N* символов строки *S*, начиная с ее *K*-го символа. Если *K* превосходит длину строки *S*, то возвращается пустая строка; если длина строки меньше $K + N$, то инвертируются все символы строки, начиная с ее *K*-го символа. Вывести значения функции InvertStr для данной строки *S* и каждой из трех пар положительных целых чисел: (K_1, N_1), (K_2, N_2), (K_3, N_3).

Param37. Описать функцию PosSub(S_0, *S*, *K*, *N*) целого типа, возвращающую номер позиции, начиная с которой в строке *S* содержится первое вхождение строки S_0, причем анализируются только *N* символов строки *S*, начиная с ее *K*-го символа (таким образом, PosSub обеспечивает *поиск в подстроке*). Если *K* превосходит длину строки *S*, то возвращается 0, если длина строки меньше $K + N$, то анализируются все символы строки, начиная с ее *K*-го символа. Если в требуемой подстроке строки *S* вхождения S_0 отсутствуют, то функция возвращает 0. Вывести значения функции PosSub для данных строк S_0, *S* и каждой из трех пар положительных целых чисел: (K_1, N_1), (K_2, N_2), (K_3, N_3).

Param38. Описать функцию PosLast(S_0, *S*) целого типа, возвращающую номер позиции, начиная с которой в строке *S* содержится последнее вхождение подстроки S_0. Считать, что перекрывающихся вхождений подстрок S_0 строка *S* не содержит. Если в строке *S* отсутствуют подстроки S_0, то функция возвращает 0. Вывести значения этой функции для пяти данных пар строк S_0 и *S*.

Param39. Описать функцию PosK(S_0, *S*, *K*) целого типа, возвращающую номер позиции, начиная с которой в строке *S* содержится *K*-е вхождение подстроки S_0 ($K > 0$). Если количество вхождений S_0 в строке *S* меньше *K*, то функция возвращает 0. Считать, что перекрывающихся вхождений подстрок S_0 строка *S* не содержит. Вывести значения этой функции для пяти данных троек: S_0, *S* и *K*.

Param40. Описать функцию WordK(*S*, *K*) строкового типа, возвращающую *K*-е слово строки *S* (*словом* считается набор символов, не содержащий пробелов и ограниченный пробелами или началом/концом строки). Если количество слов в строке меньше *K*, то функция возвращает пустую строку. Используя эту функцию, выделить из данной строки *S* слова с данными номерами K_1, K_2, K_3.

Param41. Описать процедуру SplitStr(*S*, *W*, *N*), которая формирует по данной строке *S* массив *W* слов, входящих в *S* (массив *W* и его размер *N* являются выходными параметрами). *Словом* считается набор символов, не содержащий пробелов и ограниченный пробелами или началом/концом строки; предполагается, что строка *S* содержит не более 10 слов. Используя функцию SplitStr, найти количество слов *N*, содержащихся в данной строке *S*, и сами эти слова.

Param42. Описать функцию CompressStr(*S*) строкового типа, выполняющую *сжатие* строки *S* по следующему правилу: каждая подстрока строки *S*, состоящая из более чем четырех одинаковых символов *C*, заменяется текстом вида «*C*{*K*}», где *K* — количество символов *C* (предполагается, что строка *S* не содержит фигурных скобок «{» и «}»). Например, для строки *S* = «bbbccccce» функция вернет строку «bbbc{5}e». С помощью функции CompressStr сжать пять данных строк.

Param43. Описать функцию DecompressStr(*S*) строкового типа, восстанавливающую строку, сжатую процедурой CompressStr (см. задание Param42). Параметр *S* содержит сжатую строку; восстановленная строка является возвращаемым значением функции. С помощью функции DecompressStr восстановить пять данных сжатых строк.

Param44. Описать функцию DecToBin(*N*) строкового типа, возвращающую строковое представление целого неотрицательного числа *N* в двоичной системе счисления. Результирующая строка состоит из символов «0»–«1» и не содержит ведущих нулей (за исключением представления числа 0). Используя эту функцию, получить двоичные представления пяти данных чисел.

Param45. Описать функцию DecToHex(*N*) строкового типа, возвращающую строковое представление целого неотрицательного числа *N* в 16-ричной системе счисления. Результирующая строка состоит из символов «0»–«9», «A»–«F» и не содержит ведущих нулей (за исключением представления числа 0). Используя эту функцию, получить 16-ричные представления пяти данных чисел.

Param46. Описать функцию BinToDec(*S*) целого типа, определяющую целое неотрицательное число по его строковому представлению *S* в двоичной системе счисления. Параметр *S* имеет строковый тип, состоит из символов «0»–«1» и не содержит ведущих нулей (за исключением значения «0»).

Используя эту функцию, вывести пять чисел, для которых даны их двоичные представления.

Param47. Описать функцию HexToDec(*S*) целого типа, определяющую целое неотрицательное число по его строковому представлению *S* в 16-ричной системе счисления. Параметр *S* имеет строковый тип, состоит из символов «0»–«9», «A»–«F» и не содержит ведущих нулей (за исключением значения «0»). Используя эту функцию, вывести пять чисел, для которых даны их 16-ричные представления.

18.3 Файлы

Param48. Описать функцию IntFileSize(*S*) целого типа, возвращающую количество элементов в файле целых чисел с именем *S*. Если файл не существует, то функция возвращает –1. С помощью этой функции найти количество элементов в трех файлах с данными именами.

Param49. Описать функцию LineCount(*S*) целого типа, возвращающую количество строк в текстовом файле с именем *S*. Если файл не существует, то функция возвращает –1. С помощью этой функции найти количество строк в трех файлах с данными именами.

Param50. Описать процедуру InvertIntFile(*S*), меняющую порядок следования элементов файла целого типа с именем *S* на противоположный. Если файл не существует или содержит менее двух элементов, то процедура не выполняет никаких действий. Обработать с помощью этой процедуры три файла с данными именами.

Param51. Описать процедуру AddLineNumbers(*S*, *N*, *K*, *L*), добавляющую в начало каждой строки существующего текстового файла с именем *S* ее порядковый номер: первая строка получает номер *N*, вторая — $N + 1$ и т. д. Номер отображается в *K* позициях, выравнивается по правому краю и отделяется от последующего текста *L* пробелами ($K > 0$, $L > 0$). Если строка файла является пустой, то она также нумеруется, но пробелы после номера не добавляются. Применить эту процедуру к данному файлу, используя указанные значения *N*, *K* и *L*.

Param52. Описать процедуру RemoveLineNumbers(*S*), удаляющую из начала каждой строки существующего текстового файла с именем *S* ее порядковый номер, добавленный процедурой AddLineNumbers (см. задание Param51), а также пробелы, отделяющие номер от последующего текста. Если строки не содержат номеров, то процедура не выполняет никаких действий. Применить эту процедуру к файлу с данным именем.

Param53. Описать процедуру SplitIntFile(S_0, *K*, S_1, S_2), копирующую первые *K* (≥ 0) элементов существующего файла целых чисел с именем S_0 в новый файл целых чисел с именем S_1, а остальные элементы — в новый файл це-

лых чисел с именем S_2. Один из созданных файлов может остаться пустым. Применить эту процедуру к файлу с данным именем S_0, используя указанные значения K, S_1 и S_2.

Param54. Описать процедуру SplitText(S_0, K, S_1, S_2), копирующую первые K (≥ 0) строк существующего текстового файла с именем S_0 в новый текстовый файл с именем S_1, а остальные строки — в новый текстовый файл с именем S_2. Один из созданных файлов может остаться пустым. Применить эту процедуру к файлу с данным именем S_0, используя указанные значения K, S_1 и S_2.

Param55. Описать процедуру StringFileToText(S), преобразующую двоичный строковый файл с именем S в текстовый файл с тем же именем. Используя эту процедуру, преобразовать два данных строковых файла с именами S_1 и S_2 в текстовые.

Param56. Описать процедуру TextToStringFile(S), преобразующую текстовый файл с именем S в двоичный строковый файл с тем же именем. Используя эту процедуру, преобразовать два данных текстовых файла с именами S_1 и S_2 в строковые.

Param57. Описать процедуру EncodeText(S, K), которая шифрует текстовый файл с именем S, выполняя циклическую замену каждой русской буквы на букву того же регистра, расположенную в алфавите на K-й позиции после шифруемой буквы ($0 < K < 10$). Например, при $K = 3$ «А» перейдет в «Г», «я» — в «в». Букву «ё» в алфавите не учитывать, считая, что за буквой «е» сразу идет «ж». Символы, не являющиеся русскими буквами, при шифровании не изменять. Используя эту процедуру и зная кодовое смещение K, зашифровать файл с указанным именем.

Param58. Описать процедуру DecodeText(S, K), которая дешифрует текстовый файл с именем S, зашифрованный с использованием кодового смещения K (способ шифрования описан в задании Param57). Используя эту процедуру и зная кодовое смещение K, расшифровать файл с указанным именем.

18.4 Записи

При вводе и выводе каждой даты в заданиях Param59–Param63 вначале указывается день, затем номер месяца, затем год. При вводе каждой точки в заданиях Param64–Param70 вначале указывается ее абсцисса (x-координата), затем ее ордината (y-координата).

Param59. Описать тип TDate — запись с полями целого типа Day (день), Month (месяц) и Year (год) — и функцию LeapYear(D) логического типа с параметром типа TDate, которая возвращает True, если год в дате D является високосным, и False в противном случае. Вывести значение функции LeapYear для пяти данных дат (предполагается, что все даты являются

правильными). *Високосным* считается год, делящийся на 4, за исключением тех годов, которые делятся на 100 и не делятся на 400.

Param60. Используя тип TDate и функцию LeapYear (см. задание Param59), описать функцию DaysInMonth(*D*) целого типа с параметром типа TDate, которая возвращает количество дней для месяца, указанного в дате *D*. Вывести значение функции DaysInMonth для пяти данных дат (предполагается, что все даты являются правильными).

Param61. Используя тип TDate и функцию DaysInMonth (см. задания Param59 и Param60), описать функцию CheckDate(*D*) целого типа с параметром типа TDate, которая проверяет правильность даты, указанной в параметре *D*. Если дата *D* является правильной, то функция возвращает 0; если в дате указан неверный номер месяца, то функция возвращает 1; если в дате указан неверный день для данного месяца, то возвращается 2. Вывести значение функции CheckDate для пяти данных дат.

Param62. Используя тип TDate и функции DaysInMonth и CheckDate (см. задания Param59–Param61), описать процедуру PrevDate(*D*) с параметром типа TDate, которая преобразует дату *D* к предыдущей дате (если дата *D* является неправильной, то она не изменяется). Запись *D* является входным и выходным параметром. Применить процедуру PrevDate к пяти данным датам.

Param63. Используя тип TDate и функции DaysInMonth и CheckDate (см. задания Param59–Param61), описать процедуру NextDate(*D*) с параметром типа TDate, которая преобразует дату *D* к следующей дате (если дата *D* является неправильной, то она не изменяется). Запись *D* является входным и выходным параметром. Применить процедуру NextDate к пяти данным датам.

Param64. Описать тип TPoint — запись с полями вещественного типа *X* и *Y* (координаты точки на плоскости) — и функцию Leng(*A*, *B*) вещественного типа, находящую длину отрезка *AB* на плоскости по координатам его концов:

$$|AB| = \sqrt{(A.X - B.X)^2 + (A.Y - B.Y)^2}$$

(*A* и *B* — параметры типа TPoint). С помощью этой функции найти длины отрезков *AB*, *AC*, *AD*, если даны координаты точек *A*, *B*, *C*, *D*.

Param65. Используя тип TPoint и функцию Leng (см. задание Param64), описать тип TTriangle — запись с полями *A*, *B*, *C* типа TPoint (вершины треугольника) — и функцию Perim(*T*) вещественного типа, находящую периметр треугольника *T* (*T* — параметр типа TTriangle). С помощью этой функции найти периметры треугольников *ABC*, *ABD*, *ACD*, если даны координаты точек *A*, *B*, *C*, *D*.

Param66. Используя типы TPoint, TTriangle и функции Leng и Perim (см. задания Param64 и Param65), описать функцию Area(*T*) вещественного типа,

находящую площадь треугольника T (T — параметр типа TTriangle) по *формуле Герона*:

$$S_{ABC} = \sqrt{p \cdot (p - |AB|) \cdot (p - |AC|) \cdot (p - |BC|)},$$

где p — *полупериметр*. С помощью этой функции найти площади треугольников ABC, ABD, ACD, если даны координаты точек A, B, C, D.

Param67. Используя типы TPoint, TTriangle и функции Leng и Area (см. задания Param64–Param66), описать функцию Dist(P, A, B) вещественного типа (P, A, B — параметры типа TPoint), находящую расстояние $D(P, AB)$ от точки P до прямой AB по формуле

$$D(P, AB) = 2 \cdot S_{PAB} / |AB|,$$

где S_{PAB} — площадь треугольника PAB. С помощью этой функции найти расстояния от точки P до прямых AB, AC, BC, если даны координаты точек P, A, B, C.

Param68. Используя типы TPoint, TTriangle и функцию Dist (см. задания Param64, Param65, Param67), описать процедуру Heights(T, h_1, h_2, h_3), находящую высоты h_1, h_2, h_3 треугольника T (T — входной параметр типа TTriangle, h_1, h_2, h_3 — выходные вещественные параметры), проведенные соответственно из вершин $T.A$, $T.B$, $T.C$. С помощью этой процедуры найти высоты треугольников ABC, ABD, ACD, если даны координаты точек A, B, C, D.

Param69. Используя тип TPoint и функцию Leng (см. задание Param64), описать функцию PerimN(P, N) вещественного типа, находящую периметр N-угольника, вершины которого (в порядке их обхода) передаются в массиве P размера N (> 2) с элементами типа TPoint. С помощью этой функции найти периметры трех многоугольников, если дано число их сторон и координаты их вершин.

Param70. Используя типы TPoint, TTriangle и функцию Area (см. задания Param64–Param66), описать функцию AreaN(P, N) вещественного типа, находящую площадь выпуклого N-угольника, вершины которого (в порядке их обхода) передаются в массиве P размера N (> 2) с элементами типа TPoint. С помощью этой функции найти площади трех многоугольников, если дано число их сторон и координаты их вершин.

19 Рекурсия: группа Recur

19.1 Простейшие рекурсивные алгоритмы

Задания этого пункта можно легко решить и *без использования рекурсии*. Данное обстоятельство связано с тем, что в заданиях рассматриваются *простейшие* примеры рекурсии, легко сводимые к итерационным алгоритмам. Более того, в некоторых случаях непосредственные вычисления по рекурсивным

формулам оказываются весьма неэффективными (см., например, задания Recur4 и Recur6). Однако именно на подобных примерах проще всего получить первоначальные навыки разработки рекурсивных алгоритмов.

Recur1°. Описать рекурсивную функцию Fact(N) вещественного типа, вычисляющую значение *факториала*

$$N! = 1 \cdot 2 \cdot \ldots \cdot N$$

($N > 0$ — параметр целого типа). С помощью этой функции вычислить факториалы пяти данных чисел.

Recur2. Описать рекурсивную функцию Fact2(N) вещественного типа, вычисляющую значение *двойного факториала*

$$N!! = N \cdot (N-2) \cdot (N-4) \cdot \ldots$$

($N > 0$ — параметр целого типа; последний сомножитель в произведении равен 2, если N — четное число, и 1, если N — нечетное). С помощью этой функции вычислить двойные факториалы пяти данных чисел.

Recur3. Описать рекурсивную функцию PowerN(X, N) вещественного типа, находящую значение N-й степени числа X по формулам:

$X^0 = 1$,
$X^N = (X^{N/2})^2$ при *четных* $N > 0$,
$X^N = X \cdot X^{N-1}$ при *нечетных* $N > 0$,
$X^N = 1/X^{-N}$ при $N < 0$

($X \neq 0$ — вещественное число, N — целое; в формуле для четных N должна использоваться операция *целочисленного деления*). С помощью этой функции найти значения X^N для данного X при пяти данных значениях N.

Recur4. Описать рекурсивную функцию Fib1(N) целого типа, вычисляющую N-й элемент последовательности *чисел Фибоначчи* (N — целое число):

$$F_1 = F_2 = 1, \qquad F_K = F_{K-2} + F_{K-1}, \quad K = 3, 4, \ldots .$$

С помощью этой функции найти пять чисел Фибоначчи с данными номерами, и вывести эти числа вместе с количеством рекурсивных вызовов функции Fib1, потребовавшихся для их нахождения.

Recur5. Описать рекурсивную функцию Fib2(N) целого типа, вычисляющую N-й элемент последовательности *чисел Фибоначчи* (N — целое число):

$$F_1 = F_2 = 1, \qquad F_K = F_{K-2} + F_{K-1}, \quad K = 3, 4, \ldots .$$

Считать, что номер N не превосходит 20. Для уменьшения количества рекурсивных вызовов по сравнению с функцией Fib1 (см. задание Recur4) создать вспомогательный массив для хранения *уже вычисленных* чисел Фибоначчи и обращаться к нему при выполнении функции Fib2. С помощью функции Fib2 найти пять чисел Фибоначчи с данными номерами.

Recur6. Описать рекурсивную функцию Combin1(N, K) целого типа, находящую $C(N, K)$ — *число сочетаний* из N элементов по K — с помощью рекуррентного соотношения:

$$C(N, 0) = C(N, N) = 1,$$
$$C(N, K) = C(N - 1, K) + C(N - 1, K - 1) \quad \text{при } 0 < K < N.$$

Параметры функции — целые числа; $N > 0$, $0 \le K \le N$. Дано число N и пять различных значений K. Вывести числа $C(N, K)$ вместе с количеством рекурсивных вызовов функции Combin1, потребовавшихся для их нахождения.

Recur7. Описать рекурсивную функцию Combin2(N, K) целого типа, находящую $C(N, K)$ — *число сочетаний* из N элементов по K — с помощью рекуррентного соотношения:

$$C(N, 0) = C(N, N) = 1,$$
$$C(N, K) = C(N - 1, K) + C(N - 1, K - 1) \quad \text{при } 0 < K < N.$$

Параметры функции — целые числа; $N > 0$, $0 \le K \le N$. Считать, что параметр N не превосходит 20. Для уменьшения количества рекурсивных вызовов по сравнению с функцией Combin1 (см. задание Recur6) описать вспомогательный двумерный массив для хранения *уже вычисленных* чисел $C(N, K)$ и обращаться к нему при выполнении функции Combin2. С помощью функции Combin2 найти числа $C(N, K)$ для данного значения N и пяти различных значений K.

Recur8. Описать рекурсивную функцию RootK(X, K, N) вещественного типа, находящую приближенное значение корня K-й степени из числа X по формуле:

$$Y_0 = 1, \qquad Y_{N+1} = Y_N - (Y_N - X/(Y_N)^{K-1})/K,$$

где Y_N обозначает RootK(X, K, N) при фиксированных X и K. Параметры функции: X (> 0) — вещественное число, K (> 1) и N (> 0) — целые. С помощью функции RootK найти для данного числа X приближенные значения его корня K-й степени при шести данных значениях N.

Recur9. Описать рекурсивную функцию NOD(A, B) целого типа, находящую *наибольший общий делитель* (НОД) двух натуральных чисел A и B, используя *алгоритм Евклида*:

$$\text{НОД}(A, B) = \text{НОД}(B, A \bmod B), \quad \text{если } B \ne 0; \qquad \text{НОД}(A, 0) = A.$$

С помощью этой функции найти НОД(A, B), НОД(A, C), НОД(A, D), если даны числа A, B, C, D.

Recur10. Описать рекурсивную функцию DigitSum(K) целого типа, которая находит сумму цифр целого числа K, не используя оператор цикла. С помощью этой функции найти суммы цифр для пяти данных целых чисел.

Recur11. Описать рекурсивную функцию MaxInt(A, N) целого типа, которая находит максимальный элемент целочисленного массива A размера N ($1 \le N \le 10$), не используя оператор цикла. С помощью этой функции найти максимальные элементы массивов A, B, C размера N_A, N_B, N_C соответственно.

Recur12. Описать рекурсивную функцию DigitCount(*S*) целого типа, которая находит количество цифр в строке *S*, не используя оператор цикла. С помощью этой функции найти количество цифр в каждой из пяти данных строк.

Recur13. Описать рекурсивную функцию Palindrom(*S*) логического типа, возвращающую True, если строка *S* является *палиндромом* (то есть читается одинаково слева направо и справа налево), и False в противном случае. Оператор цикла в теле функции не использовать. Вывести значения функции Palindrom для пяти данных строк.

19.2 Разбор выражений

Во всех заданиях данного пункта предполагается, что исходные строки, определяющие выражения, не содержат пробелов. При выполнении заданий рекомендуется обойтись без операторов цикла.

Recur14º. Вывести значение целочисленного выражения, заданного в виде строки *S*. Выражение определяется следующим образом:

<выражение> ::= <цифра> | <выражение> + <цифра> |
<выражение> – <цифра>

Recur15º. Вывести значение целочисленного выражения, заданного в виде строки *S*. Выражение определяется следующим образом:

<выражение> ::= <терм> | <выражение> + <терм> |
<выражение> – <терм>
<терм> ::= <цифра> | <терм> * <цифра>

Recur16º. Вывести значение целочисленного выражения, заданного в виде строки *S*. Выражение определяется следующим образом:

<выражение> ::= <терм> | <выражение> + <терм> |
<выражение> – <терм>
<терм> ::= <элемент> | <терм> * <элемент>
<элемент> ::= <цифра> | (<выражение>)

Recur17. Вывести значение целочисленного выражения, заданного в виде строки *S*. Выражение определяется следующим образом:

<выражение> ::= <цифра> |
(<выражение><знак><выражение>)
<знак> ::= + | – | *

Recur18º. Проверить правильность выражения, заданного в виде непустой строки *S* (выражение определяется по тем же правилам, что и в задании Recur17). Если выражение составлено правильно, то вывести True, иначе вывести False.

Recur19. Проверить правильность выражения, заданного в виде непустой строки *S* (выражение определяется по тем же правилам, что и в задании Recur17). Если выражение составлено правильно, то вывести 0, в противном случае вывести номер первого ошибочного (или лишнего) символа в строке *S*.

Recur20. Вывести значение целочисленного выражения, заданного в виде строки *S*. Выражение определяется следующим образом (функция M возвращает максимальный из своих параметров, а функция m — минимальный):

<выражение> ::= <цифра> | M(<выражение> , <выражение>) |
m(<выражение> , <выражение>)

Recur21. Вывести значение логического выражения, заданного в виде строки *S*. Выражение определяется следующим образом («T» — True, «F» — False):

<выражение> ::= T | F | And(<выражение> , <выражение>) |
Or(<выражение> , <выражение>)

Recur22. Вывести значение целочисленного выражения, заданного в виде строки *S*. Выражение определяется следующим образом (функция M возвращает максимальный из своих параметров, а функция m — минимальный):

<выражение> ::= <цифра> | M(<параметры>) | m(<параметры>)
<параметры> ::= <выражение> | <выражение> , <параметры>

Recur23. Вывести значение логического выражения, заданного в виде строки *S*. Выражение определяется следующим образом («T» — True, «F» — False):

<выражение> ::= T | F | And(<параметры>) | Or(<параметры>)
<параметры> ::= <выражение> | <выражение> , <параметры>

Recur24. Вывести значение логического выражения, заданного в виде строки *S*. Выражение определяется следующим образом («T» — True, «F» — False):

<выражение> ::= T | F | And(<параметры>) |
Or(<параметры>) | Not(<выражение>)
<параметры> ::= <выражение> | <выражение> , <параметры>

19.3 Перебор с возвратом

Recur25°. Дано дерево глубины *N*, каждая внутренняя вершина которого имеет *K* (< 10) непосредственных потомков (нумеруются от 1 до *K*). Корень дерева имеет номер 0. Записать в текстовый файл с данным именем все возможные пути, ведущие от корня к листьям. Перебирать пути, начиная с «самого левого» и заканчивая «самым правым» (при этом первыми заменять конечные элементы пути).

Recur26. Дано дерево глубины *N*, каждая внутренняя вершина которого имеет *K* (< 10) непосредственных потомков (нумеруются от 1 до *K*). Корень дерева имеет номер 0. Записать в текстовый файл с данным именем все пути, ведущие от корня к листьям и удовлетворяющие следующему условию: никакие соседние элементы пути не нумеруются одной и той же цифрой. Порядок перебора путей такой же, как в задании Recur25.

Recur27. Дано дерево глубины *N* (*N* — четное), каждая внутренняя вершина которого имеет 2 непосредственных потомка: *A* с весом 1 и *B* с весом –1. Корень дерева *C* имеет вес 0. Записать в текстовый файл с данным именем все пути от корня к листьям, удовлетворяющие следующему условию: суммарный вес элементов пути равен 0. Порядок перебора путей такой же, как в задании Recur25.

Recur28. Дано дерево глубины *N* того же типа, что и в задании Recur27. Записать в текстовый файл с данным именем все пути от корня к листьям, удовлетворяющие следующему условию: суммарный вес элементов для любого начального отрезка пути неотрицателен. Порядок перебора путей такой же, как в задании Recur25.

Recur29. Дано дерево глубины *N*, каждая внутренняя вершина которого имеет 3 непосредственных потомка: *A* с весом 1, *B* с весом 0 и *C* с весом –1. Корень дерева *D* имеет вес 0. Записать в текстовый файл с данным именем все пути от корня к листьям, удовлетворяющие следующим условиям: суммарный вес элементов для любого начального отрезка пути неположителен, а суммарный вес всех элементов пути равен 0. Порядок перебора путей такой же, как в задании Recur25.

Recur30. Дано дерево глубины *N* того же типа, что и в задании Recur29. Записать в текстовый файл с данным именем все пути от корня к листьям, удовлетворяющие следующим условиям: никакие соседние элементы пути не обозначаются одной и той же буквой, а суммарный вес всех элементов пути равен 0. Порядок перебора путей такой же, как в задании Recur25.

20 Указатели и динамические структуры данных: группа Pointer

Все числа, упоминаемые в заданиях данной группы, являются *целыми*. Все указатели имеют тип PNode и указывают на записи типа TNode.

На языке Pascal типы PNode и TNode описываются следующим образом:

```
type
  PNode = ^TNode;
  TNode = record
    Data: Integer;
    Next: PNode;
```

```
    Prev: PNode;
  end;
```

Приведем также описание этих типов на языке C++:

```
struct TNode
{
  int Data;
  TNode* Next;
  TNode* Prev;
};
typedef TNode* PNode;
```

В заданиях на стеки и очереди (Pointer1–Pointer28) при работе с записями типа TNode используются только поля Data и Next (см. задание Pointer1); в заданиях на двусвязные списки (Pointer29–Pointer80) используются все поля записи TNode (см. задание Pointer29).

Так как переменные типа «указатель» предназначены для хранения *адресов*, в формулировках заданий слова «указатель» (на элемент данных) и «адрес» (элемента данных) используются как синонимы. Имя nil для *нулевого указателя* в формулировках заданий заимствовано из языка Pascal. В языке C++ в качестве нулевого указателя можно использовать обычное число 0 или макроопределение NULL.

20.1 Стеки

Pointer1. Дан адрес P_1 записи типа TNode, содержащей поле Data (целого типа) и поле Next (типа PNode — указателя на TNode). Эта запись связана полем Next со следующей записью того же типа. Вывести значения полей Data обеих записей, а также адрес P_2 следующей записи.

Pointer2º. Дан адрес P_1 записи типа TNode. Эта запись связана полем Next со следующей записью того же типа, она, в свою очередь, — со следующей, и так далее до записи, поле Next которой равно nil (таким образом, возникает *цепочка* связанных записей). Вывести значения полей Data для всех элементов цепочки, *длину цепочки* (то есть число ее элементов) и адрес ее последнего элемента.

В заданиях Pointer3–Pointer13 структура *«стек»* (stack) моделируется цепочкой связанных узлов-записей типа TNode (см. задание Pointer2). Поле Next последнего элемента цепочки равно nil. *Вершиной стека* (top) считается первый элемент цепочки. Для доступа к стеку используется указатель на его вершину (для пустого стека данный указатель полагается равным nil). *Значением* элемента стека считается значение его поля Data.

Pointer3º. Дано число D и указатель P_1 на вершину непустого стека. Добавить элемент со значением D в стек и вывести адрес P_2 новой вершины стека.

Pointer4. Дано число N (> 0) и набор из N чисел. Создать стек, содержащий исходные числа (последнее число будет вершиной стека), и вывести указатель на его вершину.

Pointer5º. Дан указатель P_1 на вершину непустого стека. Извлечь из стека первый (верхний) элемент и вывести его значение D, а также адрес P_2 новой вершины стека. Если после извлечения элемента стек окажется пустым, то положить P_2 = nil. После извлечения элемента из стека освободить память, занимаемую этим элементом.

Pointer6. Дан указатель P_1 на вершину стека, содержащего не менее десяти элементов. Извлечь из стека первые девять элементов и вывести их значения. Вывести также адрес новой вершины стека. После извлечения элементов из стека освобождать память, которую они занимали.

Pointer7. Дан указатель P_1 на вершину стека (если стек пуст, то P_1 = nil). Извлечь из стека все элементы и вывести их значения. Вывести также количество извлеченных элементов N (для пустого стека вывести 0). После извлечения элементов из стека освобождать память, которую они занимали.

Pointer8º. Даны указатели P_1 и P_2 на вершины двух непустых стеков. Переместить все элементы из первого стека во второй (в результате элементы первого стека будут располагаться во втором стеке в порядке, обратном исходному) и вывести адрес новой вершины второго стека. Операции выделения и освобождения памяти не использовать.

Pointer9. Даны указатели P_1 и P_2 на вершины двух непустых стеков. Перемещать элементы из первого стека во второй, пока значение вершины первого стека не станет четным (перемещенные элементы первого стека будут располагаться во втором стеке в порядке, обратном исходному). Если в первом стеке нет элементов с четными значениями, то переместить из первого стека во второй все элементы. Вывести адреса новых вершин первого и второго стека (если первый стек окажется пустым, то вывести для него константу nil). Операции выделения и освобождения памяти не использовать.

Pointer10º. Дан указатель P_1 на вершину непустого стека. Создать два новых стека, переместив в первый из них все элементы исходного стека с четными значениями, а во второй — с нечетными (элементы в новых стеках будут располагаться в порядке, обратном исходному; один из этих стеков может оказаться пустым). Вывести адреса вершин полученных стеков (для пустого стека вывести nil). Операции выделения и освобождения памяти не использовать.

Pointer11º. Дан указатель P_1 на вершину стека (если стек пуст, то P_1 = nil). Также дано число N (> 0) и набор из N чисел. Описать тип TStack — запись с одним полем Top типа PNode (поле указывает на *вершину стека*) — и процедуру Push(S, D), которая добавляет в стек S новый элемент со значе-

нием *D* (*S* — входной и выходной параметр типа TStack, *D* — входной параметр целого типа). С помощью процедуры Push добавить в исходный стек данный набор чисел (последнее число будет вершиной стека) и вывести адрес новой вершины стека.

Pointer12. Дан указатель P_1 на вершину стека, содержащего не менее пяти элементов. Используя тип TStack (см. задание Pointer11), описать функцию Pop(*S*) целого типа, которая извлекает из стека *S* первый (верхний) элемент, возвращает его значение и освобождает память, которую занимал извлеченный элемент (*S* — входной и выходной параметр типа TStack). С помощью функции Pop извлечь из исходного стека пять элементов и вывести их значения. Вывести также указатель на новую вершину стека (если результирующий стек окажется пустым, то этот указатель должен быть равен nil).

Pointer13. Дан указатель P_1 на вершину стека. Используя тип TStack (см. задание Pointer11), описать функции StackIsEmpty(*S*) логического типа (возвращает True, если стек *S* пуст, и False в противном случае) и Peek(*S*) целого типа (возвращает значение вершины непустого стека *S*, не удаляя ее из стека). В обеих функциях переменная *S* является входным параметром типа TStack. С помощью этих функций, а также функции Pop из задания Pointer12, извлечь из исходного стека пять элементов (или все содержащиеся в нем элементы, если их менее пяти) и вывести их значения. Вывести также значение функции StackIsEmpty для результирующего стека и, если результирующий стек не является пустым, значение и адрес его новой вершины.

20.2 Очереди

В заданиях Pointer14–Pointer28 структура *«очередь»* (queue) моделируется цепочкой связанных узлов-записей типа TNode (см. задание Pointer2). Поле Next последнего элемента цепочки равно nil. *Началом очереди* («головой», head) считается первый элемент цепочки, *концом* («хвостом», tail) — ее последний элемент. Для возможности быстрого добавления в конец очереди нового элемента удобно хранить, помимо указателя на начало очереди, также и указатель на ее конец. В случае пустой очереди указатели на ее начало и конец полагаются равными nil. Как и для стека, *значением* элемента очереди считается значение его поля Data.

Pointer14. Дан набор из 10 чисел. Создать очередь, содержащую данные числа в указанном порядке (первое число будет размещаться в начале очереди, последнее — в конце), и вывести указатели P_1 и P_2 на начало и конец очереди.

Pointer15. Дан набор из 10 чисел. Создать две очереди: первая должна содержать числа из исходного набора с нечетными номерами (1, 3, …, 9), а вто-

рая — с четными (2, 4, …, 10); порядок чисел в каждой очереди должен совпадать с порядком чисел в исходном наборе. Вывести указатели на начало и конец первой, а затем второй очереди.

Pointer16. Дан набор из 10 чисел. Создать две очереди: первая должна содержать все нечетные, а вторая — все четные числа из исходного набора (порядок чисел в каждой очереди должен совпадать с порядком чисел в исходном наборе). Вывести указатели на начало и конец первой, а затем второй очереди (одна из очередей может оказаться пустой; в этом случае вывести для нее две константы nil).

Pointer17. Дано число D и указатели P_1 и P_2 на начало и конец очереди (если очередь является пустой, то $P_1 = P_2 =$ nil). Добавить элемент со значением D в конец очереди и вывести новые адреса начала и конца очереди.

Pointer18. Дано число D и указатели P_1 и P_2 на начало и конец очереди, содержащей не менее двух элементов. Добавить элемент со значением D в конец очереди и извлечь из очереди первый (начальный) элемент. Вывести значение извлеченного элемента и новые адреса начала и конца очереди. После извлечения элемента из очереди освободить память, занимаемую этим элементом.

Pointer19. Дано число N (> 0) и указатели P_1 и P_2 на начало и конец непустой очереди. Извлечь из очереди N начальных элементов и вывести их значения (если очередь содержит менее N элементов, то извлечь все ее элементы). Вывести также новые адреса начала и конца очереди (для пустой очереди дважды вывести nil). После извлечения элементов из очереди освобождать память, которую они занимали.

Pointer20. Даны указатели P_1 и P_2 на начало и конец непустой очереди. Извлекать из очереди элементы, пока значение начального элемента очереди не станет четным, и выводить значения извлеченных элементов (если очередь не содержит элементов с четными значениями, то извлечь все ее элементы). Вывести также новые адреса начала и конца очереди (для пустой очереди дважды вывести nil). После извлечения элементов из очереди освобождать память, которую они занимали.

Pointer21. Даны две очереди; адреса начала и конца первой равны P_1 и P_2, а второй — P_3 и P_4 (если очередь является пустой, то соответствующие адреса равны nil). Переместить все элементы первой очереди (в порядке от начала к концу) в конец второй очереди и вывести новые адреса начала и конца второй очереди. Операции выделения и освобождения памяти не использовать.

Pointer22. Дано число N (> 0) и две непустые очереди; адреса начала и конца первой равны P_1 и P_2, а второй — P_3 и P_4. Переместить N начальных элементов первой очереди в конец второй очереди. Если первая очередь содержит менее N элементов, то переместить из первой очереди во вторую

все элементы. Вывести новые адреса начала и конца первой, а затем второй очереди (для пустой очереди дважды вывести nil). Операции выделения и освобождения памяти не использовать.

Pointer23. Даны две непустые очереди; адреса начала и конца первой равны P_1 и P_2, а второй — P_3 и P_4. Перемещать элементы из начала первой очереди в конец второй, пока значение начального элемента первой очереди не станет четным (если первая очередь не содержит четных элементов, то переместить из первой очереди во вторую все элементы). Вывести новые адреса начала и конца первой, а затем второй очереди (для пустой очереди дважды вывести nil). Операции выделения и освобождения памяти не использовать.

Pointer24. Даны две непустые очереди; адреса начала и конца первой равны P_1 и P_2, а второй — P_3 и P_4. Очереди содержат одинаковое количество элементов. Объединить очереди в одну, в которой элементы исходных очередей чередуются (начиная с первого элемента первой очереди). Вывести указатели на начало и конец полученной очереди. Операции выделения и освобождения памяти не использовать.

Pointer25. Даны две непустые очереди; адреса начала и конца первой равны P_1 и P_2, а второй — P_3 и P_4. Элементы каждой из очередей упорядочены по возрастанию (в направлении от начала очереди к концу). Объединить очереди в одну с сохранением упорядоченности элементов. Вывести указатели на начало и конец полученной очереди. Операции выделения и освобождения памяти не использовать, поля Data не изменять.

Pointer26. Даны указатели P_1 и P_2 на начало и конец очереди (если очередь является пустой, то $P_1 = P_2$ = nil). Также дано число N (> 0) и набор из N чисел. Описать тип TQueue — запись с двумя полями типа PNode: Head (*начало очереди*) и Tail (*конец очереди*) — и процедуру Enqueue(Q, D), которая добавляет в конец очереди Q новый элемент со значением D (Q — входной и выходной параметр типа TQueue, D — входной параметр целого типа). С помощью процедуры Enqueue добавить в исходную очередь данный набор чисел и вывести новые адреса ее начала и конца.

Pointer27. Даны указатели P_1 и P_2 на начало и конец очереди, содержащей не менее пяти элементов. Используя тип TQueue (см. задание Pointer26), описать функцию Dequeue(Q) целого типа, которая извлекает из очереди первый (начальный) элемент, возвращает его значение и освобождает память, занимаемую извлеченным элементом (Q — входной и выходной параметр типа TQueue). С помощью функции Dequeue извлечь из исходной очереди пять начальных элементов и вывести их значения. Вывести также адреса начала и конца результирующей очереди (если очередь окажется пустой, то эти адреса должны быть равны nil).

Pointer28. Даны указатели P_1 и P_2 на начало и конец очереди. Используя тип TQueue (см. задание Pointer26), описать функцию QueueIsEmpty(Q) логического типа, которая возвращает True, если очередь Q пуста, и False в противном случае (Q — входной параметр типа TQueue). Используя эту функцию для проверки состояния очереди, а также функцию Dequeue из задания Pointer27, извлечь из исходной очереди пять начальных элементов (или все содержащиеся в ней элементы, если их менее пяти) и вывести их значения. Вывести также значение функции QueueIsEmpty для полученной очереди и новые адреса ее начала и конца.

20.3 Двусвязные списки

Pointer29. Дан адрес P_2 записи типа TNode, содержащей поле Data (целого типа) и поля Prev и Next (типа PNode — указателя на TNode). Эта запись связана полями Prev и Next соответственно с предыдущей и последующей записью того же типа. Вывести значения полей Data предыдущей и последующей записи, а также адреса P_1 и P_3 предыдущей и последующей записи.

Pointer30º. Дан указатель P_1 на начало непустой цепочки элементов-записей типа TNode, связанных между собой с помощью поля Next. Используя поле Prev записи TNode, преобразовать исходную (*односвязную*) цепочку в *двусвязную*, в которой каждый элемент связан не только с последующим элементом (с помощью поля Next), но и с предыдущим (с помощью поля Prev). Поле Prev первого элемента положить равным nil. Вывести указатель на последний элемент преобразованной цепочки.

В заданиях Pointer31–Pointer69 структура *«двусвязный список»* (doubly linked list) моделируется цепочкой узлов-записей типа TNode, связанных как с предыдущим, так и с последующим узлом (см. задание Pointer30). Поле Next последнего элемента цепочки и поле Prev первого элемента цепочки равны nil. Для доступа к любому элементу двусвязного списка достаточно иметь указатель на один из его элементов, однако для ускорения операций со списком обычно хранят три указателя: на *первый* элемент списка (first), на его *последний* элемент (last) и на *текущий* элемент (current). Для пустого списка все эти указатели полагаются равными nil. Как в случае стека и очереди, *значением* элемента списка считается значение его поля Data.

Pointer31. Дан указатель P_0 на один из элементов непустого двусвязного списка. Вывести число N — количество элементов в списке, а также указатели P_1 и P_2 на первый и последний элементы списка.

Pointer32. Даны числа D_1 и D_2 и указатель P_0 на один из элементов непустого двусвязного списка. Добавить в начало списка новый элемент со значением D_1, а в конец — новый элемент со значением D_2. Вывести адреса первого и последнего элементов полученного списка.

Pointer33. Дано число D и указатель P_0 на один из элементов непустого двусвязного списка. Вставить перед данным элементом списка новый элемент со значением D и вывести указатель на добавленный элемент списка.

Pointer34. Дано число D и указатель P_0 на один из элементов непустого двусвязного списка. Вставить после данного элемента списка новый элемент со значением D и вывести указатель на добавленный элемент списка.

Pointer35. Даны указатели P_1 и P_2 на первый и последний элементы двусвязного списка, содержащего не менее двух элементов. Продублировать в списке первый и последний элементы (новые элементы добавлять перед существующими элементами с такими же значениями) и вывести указатель на первый элемент преобразованного списка.

Pointer36. Даны указатели P_1 и P_2 на первый и последний элементы двусвязного списка, содержащего не менее двух элементов. Продублировать в списке первый и последний элементы (новые элементы добавлять после существующих элементов с такими же значениями) и вывести указатель на последний элемент преобразованного списка.

Pointer37. Дан указатель P_1 на первый элемент непустого двусвязного списка. Продублировать в списке все элементы с нечетными номерами (новые элементы добавлять перед существующими элементами с такими же значениями) и вывести указатель на первый элемент преобразованного списка.

Pointer38. Дан указатель P_1 на первый элемент непустого двусвязного списка. Продублировать в списке все элементы с нечетными номерами (новые элементы добавлять после существующих элементов с такими же значениями) и вывести указатель на последний элемент преобразованного списка.

Pointer39. Дан указатель P_1 на первый элемент непустого двусвязного списка. Продублировать в списке все элементы с нечетными значениями (новые элементы добавлять перед существующими элементами с такими же значениями) и вывести указатель на первый элемент преобразованного списка.

Pointer40. Дан указатель P_1 на первый элемент непустого двусвязного списка. Продублировать в списке все элементы с нечетными значениями (новые элементы добавлять после существующих элементов с такими же значениями) и вывести указатель на последний элемент преобразованного списка.

Pointer41. Дан указатель P_0 на один из элементов непустого двусвязного списка. Удалить из списка данный элемент и вывести два указателя: на элемент, предшествующий удаленному, и на элемент, следующий за удаленным (один или оба этих элемента могут отсутствовать; для отсутствующих

элементов выводить nil). После удаления элемента из списка освободить память, занимаемую этим элементом.

Pointer42. Дан указатель P_1 на первый элемент двусвязного списка, содержащего не менее двух элементов. Удалить из списка все элементы с нечетными номерами и вывести указатель на первый элемент преобразованного списка. После удаления элементов из списка освобождать память, которую они занимали.

Pointer43. Дан указатель P_1 на первый элемент непустого двусвязного списка. Удалить из списка все элементы с нечетными значениями и вывести указатель на первый элемент преобразованного списка (если в результате удаления элементов список окажется пустым, то вывести nil). После удаления элементов из списка освобождать память, которую они занимали.

Pointer44. Дан указатель P_0 на один из элементов непустого двусвязного списка. Переместить данный элемент в конец списка и вывести указатели на первый и последний элементы преобразованного списка. Операции выделения и освобождения памяти не использовать, поля Data не изменять.

Pointer45. Дан указатель P_0 на один из элементов непустого двусвязного списка. Переместить данный элемент в начало списка и вывести указатели на первый и последний элементы преобразованного списка. Операции выделения и освобождения памяти не использовать, поля Data не изменять.

Pointer46. Дано число K (> 0) и указатель P_0 на один из элементов непустого двусвязного списка. Переместить в списке данный элемент на K позиций вперед (если после данного элемента находится менее K элементов, то переместить его в конец списка). Вывести указатели на первый и последний элементы преобразованного списка. Операции выделения и освобождения памяти не использовать, поля Data не изменять.

Pointer47. Дано число K (> 0) и указатель P_0 на один из элементов непустого двусвязного списка. Переместить в списке данный элемент на K позиций назад (если перед данным элементом находится менее K элементов, то переместить его в начало списка). Вывести указатели на первый и последний элементы преобразованного списка. Операции выделения и освобождения памяти не использовать, поля Data не изменять.

Pointer48. Даны указатели P_X и P_Y на два различных элемента двусвязного списка (элемент с адресом P_X находится в списке перед элементом с адресом P_Y, но не обязательно рядом с ним). Поменять местами данные элементы и вывести указатель на первый элемент преобразованного списка. Операции выделения и освобождения памяти не использовать, поля Data не изменять.

Pointer49°. Дан указатель P_1 на первый элемент непустого двусвязного списка. Перегруппировать его элементы, переместив все элементы с нечетными номерами в конец списка (в том же порядке) и вывести указатель на пер-

вый элемент преобразованного списка. Операции выделения и освобождения памяти не использовать, поля Data не изменять.

Pointer50. Дан указатель P_1 на первый элемент непустого двусвязного списка. Перегруппировать его элементы, переместив все элементы с нечетными значениями в конец списка (в том же порядке) и вывести указатель на первый элемент преобразованного списка. Операции выделения и освобождения памяти не использовать, поля Data не изменять.

Pointer51. Даны два непустых двусвязных списка и связанные с ними указатели: P_A и P_B указывают на первый и последний элементы первого списка, P_C — на один из элементов второго. Объединить исходные списки, поместив все элементы первого списка (в том же порядке) перед данным элементом второго списка, и вывести указатели на первый и последний элементы объединенного списка. Операции выделения и освобождения памяти не использовать.

Pointer52. Даны два непустых двусвязных списка и связанные с ними указатели: P_A и P_B указывают на первый и последний элементы первого списка, P_C — на один из элементов второго. Объединить исходные списки, поместив все элементы первого списка (в том же порядке) после данного элемента второго списка, и вывести указатели на первый и последний элементы объединенного списка. Операции выделения и освобождения памяти не использовать.

Pointer53. Даны указатели P_X и P_Y на два различных элемента двусвязного списка; элемент с адресом P_X находится в списке перед элементом с адресом P_Y, но не обязательно рядом с ним. Переместить элементы, расположенные между данными элементами (включая данные элементы) в новый список (в том же порядке). Вывести указатели на первые элементы преобразованного и нового списков. Если преобразованный список окажется пустым, то связанный с ним указатель положить равным nil. Операции выделения и освобождения памяти не использовать.

Pointer54. Даны указатели P_X и P_Y на два различных элемента двусвязного списка; элемент с адресом P_X находится в списке перед элементом с адресом P_Y, но не обязательно рядом с ним. Переместить элементы, расположенные между данными элементами (не включая данные элементы) в новый список (в том же порядке). Вывести указатели на первые элементы преобразованного и нового списков. Если новый список окажется пустым, то связанный с ним указатель положить равным nil. Операции выделения и освобождения памяти не использовать.

Pointer55°. Дан указатель P_1 на первый элемент непустого двусвязного списка. Преобразовать список в *циклический*, связав его последний элемент с помощью поля Next с первым, а первый элемент с помощью поля Prev — с

последним, и вывести указатель на элемент, который был последним элементом исходного списка.

Pointer56. Даны указатели P_1 и P_2 на первый и последний элементы непустого двусвязного списка, содержащего четное количество элементов. Преобразовать список в два *циклических* списка (см. задание Pointer55), первый из которых содержит первую половину элементов исходного списка, а второй — вторую половину. Вывести указатели P_A и P_B на два средних элемента исходного списка (элемент с адресом P_A должен входить в первый циклический список, а элемент с адресом P_B — во второй). Операции выделения и освобождения памяти не использовать.

Pointer57. Дано число K (> 0) и указатели P_1 и P_2 на первый и последний элементы непустого двусвязного списка. Осуществить *циклический сдвиг* элементов списка на K позиций вперед (то есть в направлении от начала к концу списка) и вывести указатели на первый и последний элементы полученного списка. Для выполнения циклического сдвига преобразовать исходный список в циклический (см. задание Pointer55), после чего «разорвать» его в позиции, соответствующей данному значению K. Операции выделения и освобождения памяти не использовать.

Pointer58. Дано число K (> 0) и указатели P_1 и P_2 на первый и последний элементы непустого двусвязного списка. Осуществить *циклический сдвиг* элементов списка на K позиций назад (то есть в направлении от конца к началу списка) и вывести указатели на первый и последний элементы полученного списка. Для выполнения циклического сдвига преобразовать исходный список в циклический (см. задание Pointer55), после чего «разорвать» его в позиции, соответствующей данному значению K. Операции выделения и освобождения памяти не использовать.

Pointer59. Даны указатели P_1, P_2 и P_3 на первый, последний и текущий элементы двусвязного списка (если список является пустым, то $P_1 = P_2 = P_3 = \text{nil}$). Также дано число N (> 0) и набор из N чисел. Описать тип TList — запись с полями First, Last и Current типа PNode (поля указывают соответственно на *первый*, *последний* и *текущий* элементы списка) — и процедуру InsertLast(L, D), которая добавляет новый элемент со значением D в конец списка L (L — входной и выходной параметр типа TList, D — входной параметр целого типа). Добавленный элемент становится текущим. С помощью этой процедуры добавить в конец исходного списка данный набор чисел (в том же порядке) и вывести новые адреса его первого, последнего и текущего элементов.

Pointer60. Даны указатели P_1, P_2 и P_3 на первый, последний и текущий элементы двусвязного списка (если список является пустым, то $P_1 = P_2 = P_3 = \text{nil}$). Также дано число N (> 0) и набор из N чисел. Используя тип TList (см. задание Pointer59), описать процедуру InsertFirst(L, D), которая добавляет

новый элемент со значением D в начало списка L (L — входной и выходной параметр типа TList, D — входной параметр целого типа). Добавленный элемент становится текущим. С помощью этой процедуры добавить в начало исходного списка данный набор чисел (добавленные числа будут располагаться в списке в обратном порядке) и вывести новые адреса его первого, последнего и текущего элементов.

Pointer61. Дан непустой двусвязный список, первый, последний и текущий элементы которого имеют адреса P_1, P_2 и P_3. Также даны пять чисел. Используя тип TList (см. задание Pointer59), описать процедуру InsertBefore(L, D), которая вставляет новый элемент со значением D перед текущим элементом списка L (L — входной и выходной параметр типа TList, D — входной параметр целого типа). Вставленный элемент становится текущим. С помощью этой процедуры вставить пять данных чисел в исходный список и вывести новые адреса его первого, последнего и текущего элементов.

Pointer62. Дан непустой двусвязный список, первый, последний и текущий элементы которого имеют адреса P_1, P_2 и P_3. Также даны пять чисел. Используя тип TList (см. задание Pointer59), описать процедуру InsertAfter(L, D), которая вставляет новый элемент со значением D после текущего элемента списка L (L — входной и выходной параметр типа TList, D — входной параметр целого типа). Вставленный элемент становится текущим. С помощью этой процедуры вставить пять данных чисел в исходный список и вывести новые адреса его первого, последнего и текущего элементов.

Pointer63. Дан непустой двусвязный список, первый, последний и текущий элементы которого имеют адреса P_1, P_2 и P_3. Используя тип TList (см. задание Pointer59), описать процедуры ToFirst(L) (делает текущим первый элемент списка L), ToNext(L) (делает текущим в списке L следующий элемент, если он существует), SetData(L, D) (присваивает текущему элементу списка L значение D целого типа) и функцию IsLast(L) логического типа (возвращает True, если текущий элемент списка L является его последним элементом, и False в противном случае). Параметр L имеет тип TList; в процедурах ToFirst и ToNext он является входным и выходным. С помощью этих процедур и функций присвоить нулевые значения элементам исходного списка с нечетными номерами и вывести количество элементов в списке, а также новый адрес текущего элемента списка.

Pointer64. Дан непустой двусвязный список, первый, последний и текущий элементы которого имеют адреса P_1, P_2 и P_3. Используя тип TList (см. задание Pointer59), описать процедуры ToLast(L) (делает текущим последний элемент списка L), ToPrev(L) (делает текущим в списке L предыдущий элемент, если он существует) и функции GetData(L) целого типа (возвращает значение текущего элемента списка L), IsFirst(L) логического типа

(возвращает True, если текущий элемент списка L является его первым элементом, и False в противном случае). Параметр L имеет тип TList; в процедурах ToLast и ToPrev он является входным и выходным. С помощью этих процедур и функций вывести все четные значения элементов исходного списка, просматривая список с конца. Вывести также количество элементов в списке.

Pointer65. Даны указатели P_1, P_2 и P_3 на первый, последний и текущий элементы двусвязного списка, содержащего не менее пяти элементов. Используя тип TList (см. задание Pointer59), описать функцию DeleteCurrent(L) целого типа, удаляющую из списка L текущий элемент и возвращающую его значение (L — входной и выходной параметр типа TList). После удаления элемента текущим становится следующий элемент или, если следующего элемента не существует, последний элемент списка. Функция также освобождает память, занимаемую удаленным элементом. С помощью этой функции удалить из исходного списка пять элементов и вывести их значения. Вывести также новые адреса первого, последнего и текущего элементов списка.

Pointer66. Даны указатели P_1, P_2 и P_3 на первый, последний и текущий элементы непустого двусвязного списка. Используя тип TList (см. задание Pointer59), описать процедуру SplitList(L_1, L_2), которая переносит элементы списка L_1 от текущего до последнего в новый список L_2 (таким образом, список L_1 делится на две части, причем первая часть может оказаться пустой). Параметры процедуры имеют тип TList; первый параметр является входным и выходным, второй — выходным. Текущими элементами непустых результирующих списков становятся их первые элементы. Операции выделения и освобождения памяти в процедуре не использовать. С помощью этой процедуры разбить исходный список на два и вывести адреса первого, последнего и текущего элементов полученных списков.

Pointer67. Даны указатели на первый, последний и текущий элементы двух непустых двусвязных списков. Используя тип TList (см. задание Pointer59), описать процедуру AddList(L_1, L_2), которая добавляет все элементы из списка L_2 (в том же порядке) в конец списка L_1; в результате список L_2 становится пустым. Текущим элементом списка L_1 становится первый из добавленных элементов. Оба параметра процедуры имеют тип TList и являются входными и выходными. Операции выделения и освобождения памяти в процедуре не использовать. С помощью этой процедуры добавить второй из исходных списков в конец первого и вывести адреса первого, последнего и текущего элементов объединенного списка.

Pointer68. Даны указатели на первый, последний и текущий элементы двух непустых двусвязных списков. Используя тип TList (см. задание Pointer59), описать процедуру InsertList(L_1, L_2), которая вставляет все элементы из

списка L_2 (в том же порядке) в список L_1 перед его текущим элементом; в результате список L_2 становится пустым. Текущим элементом списка L_1 становится первый из вставленных элементов. Оба параметра процедуры имеют тип TList и являются входными и выходными. Операции выделения и освобождения памяти в процедуре не использовать. С помощью этой процедуры вставить второй из исходных списков в текущую позицию первого и вывести адреса первого, последнего и текущего элементов объединенного списка.

Pointer69. Даны указатели на первый, последний и текущий элементы двух двусвязных списков (второй список может быть пустым). Используя тип TList (см. задание Pointer59), описать процедуру MoveCurrent(L_1, L_2), которая перемещает текущий элемент списка L_1 в список L_2 (элемент вставляется после текущего элемента списка L_2 и сам становится текущим; в списке L_1 текущим становится следующий элемент или, если следующего элемента не существует, последний элемент). Оба параметра процедуры имеют тип TList и являются входными и выходными. Операции выделения и освобождения памяти в процедуре не использовать. С помощью этой процедуры переместить текущий элемент первого списка во второй и вывести адреса первого, последнего и текущего элементов полученных списков.

Использованная в заданиях Pointer31–Pointer69 реализация двусвязного списка в виде цепочки узлов, ограниченной по краям нулевыми указателями, не является единственно возможной. Двусвязный список можно также реализовать в виде *замкнутой* цепочки узлов с дополнительным *фиктивным*, или *барьерным*, элементом. Этот барьерный элемент связан своими полями Next и Prev с первым и последним элементом списка соответственно, поэтому, имея указатель на барьерный элемент, можно сразу перейти как к первому, так и к последнему элементу списка (естественно, первый и последний элементы также связаны с барьерным элементом своими полями Prev и Next соответственно). Для работы с двусвязным списком, снабженным барьерным элементом, достаточно хранить два указателя: Barrier, указывающий на барьерный элемент, и Current, указывающий на текущий элемент (который может быть как «настоящим», так и барьерным элементом). Поле Data барьерного элемента может быть произвольным; для определенности будем полагать его равным 0. *Пустой список* в данной реализации представляет собой единственный барьерный элемент, «замкнутый на себя»; это означает, что для пустого списка поля Next и Prev барьерного элемента равны *адресу барьерного элемента*, то есть значению указателя Barrier.

Задания Pointer70–Pointer80 посвящены *двусвязным спискам с барьерным элементом*.

Pointer70. Даны указатели P_1 и P_2 на первый и последний элементы двусвязного списка, реализованного в виде цепочки узлов, ограниченной по краям нулевыми указателями (если список пуст, то $P_1 = P_2 =$ nil). Преобразовать исходный список в *циклический список* (см. задание Pointer55), снабженный *барьерным элементом*. Барьерный элемент должен иметь значение 0 и быть связан своими полями Next и Prev с первым и последним элементом исходного списка (в случае пустого исходного списка поля Next и Prev барьерного элемента должны указывать на сам барьерный элемент). Вывести указатель на барьерный элемент полученного списка. Операцию выделения памяти использовать только для создания барьерного элемента.

Pointer71. Даны указатели P_1 и P_2 на барьерный и текущий элементы двусвязного списка (о списке с *барьерным элементом* см. задание Pointer70). Разбить список на два, перенеся во второй список все элементы от текущего до последнего и добавив ко второму списку барьерный элемент. Если текущий элемент исходного списка является барьерным элементом, то второй список должен быть пустым (то есть состоять только из барьерного элемента). Вывести указатель на барьерный элемент второго списка. Операцию выделения памяти использовать только для создания барьерного элемента второго списка.

Pointer72. Даны указатели P_1 и P_2 на барьерные элементы двух двусвязных списков (о списке с *барьерным элементом* см. задание Pointer70). Объединить исходные списки, связав конец первого и начало второго списка (барьерным элементом объединенного списка должен остаться барьерный элемент первого списка). Вывести указатели на первый и последний элементы объединенного списка (если объединенный список является пустым, то дважды вывести указатель на его барьерный элемент). После удаления лишнего барьерного элемента освободить занимаемую им память.

Pointer73. Даны указатели P_1 и P_2 на барьерные элементы двух двусвязных списков (о списке с *барьерным элементом* см. задание Pointer70). Объединить исходные списки, связав конец первого и начало второго списка (барьерным элементом объединенного списка должен остаться барьерный элемент второго списка). Вывести указатели на первый и последний элементы объединенного списка (если объединенный список является пустым, то дважды вывести указатель на его барьерный элемент). После удаления лишнего барьерного элемента освободить занимаемую им память.

Pointer74. Даны указатели P_1 и P_2 на барьерный и текущий элементы двусвязного списка (о списке с *барьерным элементом* см. задание Pointer70). Также дано число N (> 0) и набор из N чисел. Описать тип TListB — запись с полями Barrier и Current типа PNode (поля указывают соответственно на барьерный и текущий элементы списка) — и процедуру LBInsertLast(L, D), которая добавляет новый элемент со значением D в конец списка L (L —

входной и выходной параметр типа TListB, *D* — входной параметр целого типа). Добавленный элемент становится текущим. С помощью этой процедуры добавить в конец исходного списка данный набор чисел (в том же порядке) и вывести адрес текущего элемента полученного списка.

Pointer75. Даны указатели P_1 и P_2 на барьерный и текущий элементы двусвязного списка. Также дано число N (> 0) и набор из N чисел. Используя тип TListB (см. задание Pointer74), описать процедуру LBInsertFirst(*L*, *D*), которая добавляет новый элемент со значением *D* в начало списка *L* (*L* — входной и выходной параметр типа TListB, *D* — входной параметр целого типа). Добавленный элемент становится текущим. С помощью этой процедуры добавить в начало исходного списка данный набор чисел (добавленные числа будут располагаться в списке в обратном порядке) и вывести адрес текущего элемента полученного списка.

Pointer76. Даны указатели P_1 и P_2 на барьерный и текущий элементы двусвязного списка. Также даны пять чисел. Используя тип TListB (см. задание Pointer74), описать процедуру LBInsertBefore(*L*, *D*), которая вставляет новый элемент со значением *D* перед текущим элементом списка *L* (*L* — входной и выходной параметр типа TListB, *D* — входной параметр целого типа). Вставленный элемент становится текущим. С помощью этой процедуры вставить пять данных чисел в исходный список и вывести новый адрес его текущего элемента.

Pointer77. Даны указатели P_1 и P_2 на барьерный и текущий элементы двусвязного списка. Также даны пять чисел. Используя тип TListB (см. задание Pointer74), описать процедуру LBInsertAfter(*L*, *D*), которая вставляет новый элемент со значением *D* после текущего элемента списка *L* (*L* — входной и выходной параметр типа TListB, *D* — входной параметр целого типа). Вставленный элемент становится текущим. С помощью этой процедуры вставить пять данных чисел в исходный список и вывести новый адрес его текущего элемента.

Pointer78. Даны указатели P_1 и P_2 на барьерный и текущий элементы двусвязного списка. Используя тип TListB (см. задание Pointer74), описать процедуры LBToFirst(*L*) (делает текущим первый элемент списка *L*), LBToNext(*L*) (делает текущим в списке *L* следующий элемент), LBSetData(*L*, *D*) (присваивает текущему элементу списка *L* значение *D* целого типа, если данный элемент не является барьерным) и функцию IsBarrier(*L*) логического типа (возвращает True, если текущий элемент списка *L* является его барьерным элементом, и False в противном случае). Параметр *L* имеет тип TListB; в процедурах LBToFirst и LBToNext он является входным и выходным. С помощью этих процедур и функций присвоить нулевые значения элементам исходного списка с нечетными номерами и вывести количество элементов в списке, а также новый адрес текущего

элемента списка. Барьерный элемент при подсчете элементов не учитывать.

Pointer79. Даны указатели P_1 и P_2 на барьерный и текущий элементы двусвязного списка. Используя тип TListB (см. задание Pointer74), описать процедуры LBToLast(L) (делает текущим последний элемент списка L), LBToPrev(L) (делает текущим в списке L предыдущий элемент) и функцию LBGetData(L) целого типа (возвращает значение текущего элемента списка L). Параметр L имеет тип TListB; в процедурах LBToLast и LBToPrev он является входным и выходным. С помощью этих процедур и функций, а также с использованием функции IsBarrier из задания Pointer78, вывести все четные значения элементов исходного списка, просматривая список с конца. Вывести также количество элементов в списке. Барьерный элемент при подсчете элементов не учитывать.

Pointer80. Даны указатели P_1 и P_2 на барьерный и текущий элементы непустого двусвязного списка, причем текущий элемент не совпадает с барьерным. Используя тип TListB (см. задание Pointer74), описать функцию LBDeleteCurrent(L) целого типа, удаляющую из списка L текущий элемент и возвращающую его значение (L — входной и выходной параметр типа TListB). Текущим становится следующий элемент или, если следующий элемент является барьерным, предыдущий элемент списка. Функция также освобождает память, занимаемую удаленным элементом. Если текущим элементом является барьерный элемент, то функция не выполняет никаких действий и возвращает 0. С помощью этой функции, а также функции IsBarrier из задания Pointer78, удалить из исходного списка пять элементов (или все элементы, если их менее пяти) и вывести их значения. Вывести также новый адрес текущего элемента списка.

Литература

1. Амелина Н. И., Королева Е. М. Методические указания для студентов механико-математического факультета «Задачи по программированию». Часть 2. — Ростов-на-Дону: УПЛ РГУ, 1993. — 31 с.
2. Гусева А. И. Учимся программировать: PASCAL 7.0. Задачи и методы их решения. — М.: Диалог–МИФИ, 1997. — 256 с.
3. Дагене В. А., Григас Г. К., Аугутис К. Ф. 100 задач по программированию. — М.: Просвещение, 1993. — 255 с.
4. Задачи по программированию / С. А. Абрамов, Г. Г. Гнездилова, Е. Н. Капустина, М. И. Селюн. — М.: Наука, 1988. — 224 с.
5. Златопольский Д. М. Я иду на урок информатики: Задачи по программированию. 7–11 классы: Книга для учителя. — М.: Изд-во «Первое сентября», 2001. — 208 с.
6. Касьянов В. Н., Сабельфельд В. К. Сборник заданий по практикуму на ЭВМ. — М.: Наука, 1986. — 272 с.
7. Пильщиков В. Н. Сборник упражнений по языку Паскаль. — М.: Наука, 1989. — 160 с.
8. Практикум по Турбо Паскалю / И. А. Бабушкина, Н. А. Бушмелева, С. М. Окулов, С. Ю. Черных. — М.: АБФ, 1998. — 384 с.
9. Программирование на языке Паскаль: задачник / Под ред. О. Ф. Усковой. — СПб.: Питер, 2002. — 336 с.
10. Юркин А. Г. Задачник по программированию. — СПб.: Питер, 2002. — 192 с.
11. Абрамян М. Э., Михалкович С. С. Основы программирования на языке Паскаль: Скалярные типы данных, управляющие операторы, процедуры и функции. — Ростов-на-Дону: Изд-во «ЦВВР», 2004. — 198 с.
12. Абрамян М. Э. Практикум по программированию на языке Паскаль. 4-е изд. — Ростов-на-Дону: Изд-во «ЦВВР», 2004. — 187 с.

Содержание