Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

А. В. Абрамян, М. Э. Абрамян

Использование базового набора заданий электронного задачника Programming Taskbook при проведении практических занятий

Методическая разработка
для преподавателей программирования

Ростов-на-Дону
2011

Печатается по решению
учебно-методической комиссии
факультета математики, механики и компьютерных наук ЮФУ
от 25 апреля 2011 г. (протокол № 8)

Рецензенты:
к. ф.-м. н., доцент С. С. Михалкович,
ст. преп. Л. А. Мачулина

Аннотация

В методической разработке дается обзор базового набора учебных заданий по программированию, входящего в электронный задачник Programming Taskbook. Рассматриваются задания начального уровня (на скалярные типы данных, управляющие операторы, процедуры и функции), задания на обработку сложных типов данных (одномерные и двумерные массивы, символьные строки, двоичные и текстовые файлы), а также задания на рекурсивные алгоритмы и динамические структуры (стеки, очереди, списки, деревья). Приводятся методические рекомендации для преподавателей, связанные с организацией и проведением практических занятий по рассматриваемым темам.

Для преподавателей программирования.

Авторы: А. В. Абрамян, М. Э. Абрамян.

Предисловие

Настоящее пособие предназначено для преподавателей программирования; его целью является ознакомление с базовым набором учебных заданий, входящим в состав электронного задачника по программированию Programming Taskbook, разработанного на факультете математики, механики и компьютерных наук ЮФУ (автор доц. М. Э. Абрамян). Базовый набор содержит 1100 учебных заданий и охватывает широкий диапазон тем по курсу программирования [2–4, 8]. Формулировки заданий не привязаны к определенному языку программирования, поэтому задания могут использоваться для большинства языков, применяемых в настоящее время при обучении программированию: Pascal [9, 10], Visual Basic [1], C# и Visual Basic.NET [6], C++.

Имеется большое число пособий и методических разработок, связанных с электронным задачником [1–4, 6, 8–10], однако они ориентированы, в первую очередь, на учащегося. Данное пособие описывает набор заданий, входящих в состав задачника, «с точки зрения преподавателя». В нем рассматриваются особенности заданий, связанных с каждой темой, и даются методические рекомендации по организации занятий. Для каждой темы приводится список заданий, которые рекомендуется рассмотреть на занятиях («Разбор в классе»), и список заданий для самостоятельного выполнения («Закрепление материала»). Кроме того, отмечены задания, имеющий уровень ниже среднего («Простые задания») и выше среднего («Сложные задания»; задания повышенной сложности помечены звездочкой). Указываются также *наборы однотипных заданий*, которые удобно использовать при составлении вариантов индивидуальных заданий.

Рассматриваемый в пособии базовый набор может быть расширен путем разработки новых групп заданий. Для этого предназначен *конструктор учебных заданий* PT4TaskMaker [7], входящий в состав программного комплекса

Teacher Pack. Программный комплекс Teacher Pack предназначен для преподавателя программирования и является дополнением к электронному задачнику Programming Taskbook. Помимо конструктора учебных заданий комплекс включает *конструктор вариантов* PTVarMaker, упрощающий процесс разработки вариантов индивидуальных заданий, и *контрольный центр*, предназначенный для автоматизации действий преподавателя при подготовке и проведении групповых практических занятий с применением электронного задачника Programming Taskbook [5].

Дополнительная информация, связанная с электронным задачником Programming Taskbook, содержится на его сайте http://ptaskbook.com/. На этом сайте, в частности, приведены формулировки всех заданий, входящих в базовый набор задачника. С этого же сайта можно загрузить как сам задачник, так и комплекс Teacher Pack.

1. Задания начального уровня

1.1. Ввод и вывод данных, оператор присваивания: группа Begin

Группа Begin является первой из групп начального уровня; она предназначена для ознакомления с процедурами ввода-вывода и оператором присваивания. В качестве исходных и результирующих данных во всех заданиях используются данные вещественного типа.

Первые задания из данной группы (1–9) не предполагают использования оператора присваивания. Главное в них — описание переменных и ввод-вывод данных, а также использование арифметических операций и простейших функций (в частности, функции *извлечения квадратного корня*). В качестве примера укажем задание 3 (о нахождении площади и периметра прямоугольника по его сторонам), которое подробно разбирается в пособиях [1, 6, 10]. Уже в заданиях 10–11 использование оператора присваивания является оправданным (например, в задании 10 требуется вычислить несколько выражений, содержащих квадраты исходных данных; в этой ситуации разумно завести вспомогательные переменные для всех требуемых квадратов). В серии таких заданий (10–28) следует особо выделить задачи, связанные с определением расстояния на прямой (16–18) и на плоскости (19–21), требующие использования функции *модуль*. Принципиальную важность имеют задания, связанные с переменой местами содержимого двух (или трех) переменных: 22–24. Полезно проанализировать также одно из заданий 27–28, в которых идет речь об эффективном вычислении степени данного числа (с применением минимального количества арифметических операций).

Все прочие задания данной группы (29–40) служат для закрепления материала; они отличаются только «предметной средой» (например, задания 35–37 имеют «физическое» содержание). В этих заданиях полезно особое внимание уделить организации наглядного ввода и вывода (при условии, что на занятиях

не используется электронный задачник Programming Taskbook, который автоматически обеспечивает наглядный ввод-вывод данных).

Группа: Begin

Разбор в классе: 3, 9, 10, 13, 17, 22, 27

Закрепление материала: 4, 7, 8, 11, 14, 18, 23, 28

Простые задания: 1, 2, 8, 16

Сложные задания: 19, 21, 39, 40

Наборы однотипных заданий: 1–7; 8–9; 10–11; 13–15; 16–18; 23–24; 25–26; 29–32; 35–37.

1.2. Целые числа: группа Integer

Основная цель заданий этой группы — познакомиться с особыми операциями целочисленного деления и узнать, в каких ситуациях их следует использовать. В этой группе заданий впервые вводится целочисленный тип данных.

Задания 1–5 являются вводными; в их формулировках явно говорится, какую из операций, связанных с целочисленным делением, надо использовать. Затем идет серия заданий на выделение цифр из многозначных целых чисел и на их перестановку. Рассматриваются двузначные (6–8), трехзначные (9-16) и многозначные (17-18) числа.

Два других аспекта применения целочисленных операций иллюстрируются группами задач 19–23 и 24–28. Первая группа посвящена задачам выделения различных промежутков времени, связанных с указанным количеством секунд, прошедших с начала суток (требуется определить количество полных часов, минут, секунд с начала последнего часа или с начала последней минуты и т. д.). Вторая группа посвящена определению дня недели для дня года с указанным номером, если дан день недели для 1 января и используется один из возможных вариантов нумерации дней недели (от 0 или от 1).

Следует также обратить внимание на задачу 29 (определение количества квадратов, которые можно разместить на данном прямоугольнике), поскольку

учащиеся, как правило, вначале предлагают неверный алгоритм ее решения: разделить площадь прямоугольника на площадь квадрата.

Группа: Integer

Разбор в классе: 4, 5, 6, 8, 9, 17, 20, 24, 29

Закрепление материала: 1, 3, 12, 13, 18, 21, 25, 26

Простые задания: 1, 2, 3, 7, 11, 19,

Сложные задания: 23, 27, 28, 29, 30

Наборы однотипных заданий: 1–5; 6–8; 9–16; 17–18; 19–23; 24–28

1.3. Логические выражения: группа Boolean

Перед изучением условного оператора представляется естественным рассмотреть *логический тип данных* и операции, связанные с ним. Впрочем, последовательность изложения материала может быть и другой: вначале рассмотреть условный оператор с простыми условиями, а затем обсудить более сложные вопросы, связанные с логическим типом данных.

Во всех заданиях группы Boolean требуется вывести логическое значение, соответствующее истинности некоторого утверждения, в которое входят введенные данные, например, требуется проверить истинность утверждения «Целое число A является положительным» (задание 1). Важно отметить, что для решения таких задач не нужно использовать условный оператор, так как достаточно вывести на экран значение соответствующего логического выражения.

Первые семь заданий знакомят с операциями отношения и логическими операциями, а также с проверкой числа на четность/нечетность (задания 2–3) и анализом двойных неравенств вида $A < B < C$ (задание 6). Следует обратить внимание на задание 7, в котором требуется проверить, что некоторое число лежит между двумя другими; желательно рассмотреть несколько вариантов его решения.

Задания 8–11 и 12–15 образуют две серии заданий на комбинацию логических операций. Задания 20–23 требуют анализа цифр данного числа, поэтому при их выполнении необходимо пользоваться знаниями, полученными при вы-

полнении заданий группы Integer. Задания 25–33 имеют геометрическое содержание: в 25–29 речь идет о координатах на плоскости, а в 30–33 — об определении типа треугольника по длинам его сторон. Последняя группа заданий (34–40) имеет «шахматное» содержание: в первых двух требуется проанализировать цвет одной или двух данных клеток шахматной доски, а в остальных требуется определить, может ли определенная шахматная фигура за один ход перейти с одной данной клетки на другую. Наибольший интерес здесь представляют задачи, связанные с ходом слона и коня, которые рекомендуется обсудить в классе.

Группа: Boolean

Разбор в классе: 2, 6, 7, 8, 16, 21, 27, 29, 32, 34, 38, 40

Закрепление материала: 3, 4, 9, 10, 17, 22, 28, 33, 35, 37, 39

Простые задания: 1, 2, 3, 4, 5, 24, 25, 26, 30, 31

Сложные задания: 12, 13, 14, 15, 23, 29, 33, 37, 38, 40

Наборы однотипных заданий: 2–3; 4–6; 8–11; 12–15; 17–18; 19–23; 25–28; 30–33; 36–40

1.4. Условный оператор: группа If

Начальные задания (1–5) знакомят с различными вариантами условных операторов (сокращенная и полная форма условного оператора, а также вложенные условные операторы). Следующая большая группа заданий (6–15) связана с нахождением минимальных и максимальных элементов и их номеров в простейших случаях (когда имеется два или три элемента). Отметим также задания на проверку упорядоченности трех чисел (16–17).

В группе заданий 18–23 эффективное решение можно получить с помощью вложенных условных операторов; приведем характерное задание из этой группы (номер 22): «определить номер координатной четверти, в которой находится точка с данными ненулевыми координатами». Для подобных заданий полезно проанализировать несколько вариантов их решения. Кроме того, полезно

отметить идентичность алгоритмов, применяемых для решения «разных» по формулировке заданий: 19 и 23.

Следующая группа заданий (24–27) посвящена вычислению функций, заданных различными выражениями на разных участках своей области определения. Целесообразно рассмотреть задание 26 для функции, заданной на трех участках, и показать, что количество сравнений для отбора требуемого участка не превосходит $n - 1$, где n — количество участков.

Задания 27–30 имеют повышенную сложность; в 29–30 дополнительно требуется использовать данные строкового типа и операцию сцепления строк.

Группа: If

Разбор в классе: 1, 2, 3, 7, 9, 12, 19, 21, 26

Закрепление материала: 4, 5, 8, 11, 14, 22, 23, 25

Простые задания: 1, 2, 4, 6, 24

Сложные задания: 13, 15, 23, 27, 28, 29, 30

Наборы однотипных заданий: 1–5; 9–11; 16–17; 18–22; 24–26; 29–30

1.5. Оператор выбора: группа Case

Во вводных заданиях этой группы (1–5) следует отметить задание 2, в котором естественно возникает вариант «по умолчанию» (else в Pascal и Visual Basic, default в C++ и C#), а также задания 3–4, в которых, в случае использования языков Pascal или Visual Basic, удобно использовать диапазоны значений (в задании 3 надо вывести строку с названием времени года для данного месяца, а в задании 4 — определить число дней в указанном месяце для невисокосного года).

Отметим также задания 8–9, в которых по данному номеру дня и месяца требуется определить номер дня и месяца для следующей или предыдущей даты. В этих заданиях надо совместно использовать оператор выбора и условный оператор.

Интересными являются задания 10–11, в которых впервые используется символьный тип данных. Например, в задании 10 требуется определить, как

изменится направление робота при выполнении указанной команды поворота (направление определяется по стороне света: «С», «З», «Ю», «В», а команда является числовой: 0 — продолжать движение, 1 — поворот налево, –1 — поворот направо). В этих заданиях наглядно проявляется важность правильного выбора формата представления данных для получения эффективного алгоритма: так, при выполнении задания 10 достаточно *закодировать* стороны света подходящими номерами, после чего прибавить к номеру стороны света число, соответствующее команде робота, и перевести полученный номер обратно в символьное представление.

Задания 15–19 связаны с формированием строковых данных; наиболее сложным из них является 19, в котором надо сформировать словесное описание числа, лежащего в диапазоне 100–999. Эти задания, как и задания 12–14 и 20, являются дополнительными.

Группа: Case

Разбор в классе: 2, 3, 4, 6, 8, 10, 16

Закрепление материала: 5, 7, 9, 11, 15

Простые задания: 1, 2, 5

Сложные задания: 10, 11, 16, 17, 18, 19, 20

Наборы однотипных заданий: 6–7; 8–9; 10–11; 12–14; 15–18

1.6. Цикл с параметром: группа For

В данной группе содержится много важных алгоритмов, обязательных для твердого усвоения. Следует отметить, что эту группу заданий можно рассмотреть до групп Boolean, If и Case, так как в заданиях группы For условные операторы применять не требуется. Такой подход реализован в пособии [10]; он особенно удобен при изучении языков Pascal и Visual Basic, так как в них конструкции цикла For не требуют использования логических выражений (в частности, операций отношения).

Задания 1–6 являются простейшими заданиями на оператор цикла с параметром. При этом в задании 3 удобно использовать цикл с убывающим пара-

метром, а в заданиях 5 и 6 возникает необходимость в выводе *формул*, содержащих параметр цикла (например, в задании 6 требуется вывести стоимость 0.1, 0.2, …, 1 кг конфет, если известна цена 1 кг).

Последующие задания (7–14) посвящены алгоритмам нахождения сумм и произведений при условии, что известно число слагаемых и их вид (например, в задании 10 требуется найти сумму чисел $1 + 1/2 + \ldots + 1/N$ для данного N). Закреплению данных алгоритмов служат серии заданий 15–18 и 19–21, первая из которых посвящена операции возведения в степень, а вторая — факториалам.

Группа заданий 22–28 посвящена вычислению элементарных математических функций с помощью их разложения в ряд (требуется использовать фиксированное количество членов разложения). В этих заданиях важно обратить внимание на наличие *рекуррентной зависимости* между членами суммы, позволяющей легко пересчитывать очередной член через предыдущий.

Задания 29–30 знакомят с приемами табулирования с фиксированным шагом значений функции на заданном отрезке, а задания 31–35 предназначены для ознакомления с приемами вычисления рекуррентных последовательностей (в том числе последовательности Фибоначчи).

Наконец, завершающая часть заданий данной группы (36–40) посвящена *вложенным циклам*. Ввиду сложности этой темы допустимо не рассматривать ее при первом знакомстве с циклами, а посвятить ей отдельное занятие уже после знакомства с другими вариантами циклов (как это сделано в [10]).

Группа: For (без вложенных циклов)

Разбор в классе: 3, 4, 6, 7, 8, 12, 13, 15, 16, 17, 22, 29, 33

Закрепление материала: 1, 2, 5, 9, 10, 11, 18, 19, 20, 21, 30, 34

Простые задания: 1, 2, 3, 4, 7, 8, 15, 19, 31, 32

Сложные задания: 23, 24, 25, 26, 27, 28, 34, 35

Наборы однотипных заданий: 4–6; 7–13; 15–18; 19–21; 22–28; 29–30; 31–35

Группа: For (вложенные циклы)

Разбор в классе: 37, 39

Закрепление материала: 38, 40

Простые задания: 16

Сложные задания: *нет*

Наборы однотипных заданий: 36–38; 39–40

1.7. Цикл с условием: группа While

Задания данной группы формально не зависят ни от темы «условный оператор», ни от темы «цикл с параметром»: для выполнения любого задания группы While достаточно использовать цикл с условием, оператор присваивания и операторы ввода-вывода. В частности, к данной теме можно перейти сразу после изучения темы Boolean (до изучения темы If). Однако, как показывает опыт преподавания, при выполнении заданий из данной группы учащиеся испытывают большие затруднения. По этой причине рекомендуется приступить к этой теме уже после знакомства с описанными выше темами, однако *не использовать* условные операторы и операторы цикла For при выполнении заданий.

Задания 1–5 являются вводными; полезно разобрать задание 3 («если данное целое число является степенью 3, то вывести True, иначе вывести False») и проанализировать два варианта его решения (первый вариант основан на последовательном умножении, второй — на последовательном целочисленном делении с анализом остатка). Интересным является задание 6 о вычислении двойного факториала; если уже изучена тема «Цикл с параметром», то целесообразно сравнить варианты решения задания 6, основанные на применении различных видов циклов.

Задания 7–14 посвящены закреплению приемов, связанных с использованием цикла с параметром. Все эти задания можно разбить на два типа: «найти наименьшее число такое, что …» (например, «найти наименьшее K такое, что $3^K > N$, где N дано» — задание 9) и «найти наибольшее число такое, что …»

(например, «найти наибольшее K такое, что $3^K < N$, где N дано» — задание 10). Следует обсудить конструкции, которые надо использовать для каждого типа заданий.

Задания 15–16 относятся к заданиям с «реальным» содержанием (так, задание 15 посвящено вычислению роста банковского вклада).

Задания 17–21 связаны с анализом цифр целых чисел, и являются обобщением аналогичных заданий из группы Integer.

Отметим задания 22 (проверка числа на простоту) и 23 (нахождение наибольшего общего делителя двух целых чисел). Связанные с ними алгоритмы желательно обсудить в классе.

Задания 24–29 посвящены рекуррентным последовательностям: числам Фибоначчи (24–27) и последовательностям общего вида (28–29). В заданиях 28–29 впервые возникает условие проверки близости двух вещественных чисел. Задание 30 (на определение числа квадратов, которые можно разместить на данном прямоугольнике, причем все размеры являются *вещественными*) полезно сравнить с аналогичным заданием 29 из группы Integer.

Следует заметить, что группа While не содержит заданий на вложенные циклы с условием. Однако задания подобного типа в достаточном количестве имеются в следующей группе (Series).

Группа: While

Разбор в классе: 4, 6, 9, 10, 15, 17, 22, 23, 25, 28

Закрепление материала: 5, 11, 12, 16, 18, 24, 26, 29

Простые задания: 1, 2, 3, 17, 30

Сложные задания: 20, 21, 22, 23, 28, 29

Наборы однотипных заданий: 7–14; 15–16; 17–21; 24–27; 28–29

1.8. Последовательности: группа Series

В заданиях данной группы впервые потребуется совместно использовать различные управляющие операторы (как правило, условный оператор и один из вариантов оператора цикла). Поскольку в каждом из заданий требуется ввести

некоторую последовательность чисел, целесообразно генерировать такую последовательность с помощью датчика псевдослучайных чисел (при условии, что на занятиях не используется электронный задачник Programming Taskbook, который генерирует наборы исходных данных автоматически).

Первые задания данной группы (1–7) посвящены алгоритмам нахождения суммы и произведения; кроме того, в них требуется вывести целые (5) или дробные (6) части данных чисел, а также их округленные значения (7). Таким образом, обсуждение этих заданий — хороший повод познакомить учащихся с соответствующими функциями используемого языка программирования.

Задания 8–11 посвящены алгоритмам отбора требуемых элементов из исходного набора с *фиксированным* числом элементов; в заданиях 10–11 удобно использовать операторы явного выхода из цикла (break в Pascal, C++ и C#, Exit For в Visual Basic и VB.NET).

Серия заданий 12–16 посвящена наборам ненулевых целых чисел с заранее неизвестным числом элементов (признаком завершения набора является число 0). Стандартную схему перебора элементов в таком наборе, использующую цикл с условием, можно обсудить на примере первого задания этой серии (12), в котором требуется определить количество чисел в исходном наборе.

Особенностью следующей группы заданий (17–23) является то, что для их успешного выполнения надо хранить в памяти два (или три) последовательных элемента из данного набора. При выполнении этих заданий следует напомнить аналогичные задания из групп For и While на обработку рекуррентных последовательностей (в частности, чисел Фибоначчи).

Отдельную подгруппу заданий (26–40) составляют задания на вложенные циклы. Перед обсуждением заданий этой подгруппы полезно вернуться к группе For и рассмотреть ее подгруппу, связанную с вложенными циклами. Задания 30–40 фактически представляют собой модификации предыдущих заданий группы Series, в которых надо обработать не один, а *несколько* числовых наборов.

Группа: Series (без вложенных циклов)

Разбор в классе: 5, 6, 7, 9, 10, 12, 15, 17, 22

Закрепление материала: 3, 4, 8, 11, 13, 16, 18, 21

Простые задания: 1, 2, 3, 4

Сложные задания: 23, 24, 25

Наборы однотипных заданий: 1–4; 5–7; 8–11; 12–16; 17–23

Группа: Series (вложенные циклы)

Разбор в классе: 27, 29, 30, 36

Закрепление материала: 28, 31, 35, 37

Простые задания: 26, 27, 28

Сложные задания: 38, 39, 40

Наборы однотипных заданий: 26–28; 29–34; 35–40

1.9. Процедуры и функции: группа Proc

Данная группа состоит из трех подгрупп: первая содержит задания на разработку процедур (1–15), вторая — на разработку функций (16–36), а третья — дополнительные задания на процедуры и функции (37–60).

Первые две подгруппы группы Proc могут изучаться в любом порядке. Например, в [10] вначале рассматриваются процедуры, а затем — функции. Однако это связано с тем, что предварительно, с применением специальных исполнителей (Робот и Чертежник), изучаются процедуры без параметров, после которых переход к процедурам с параметрам выполняется достаточно просто. Если вспомогательные исполнители не используются, то, возможно, целесообразнее начать с рассмотрения функций, поскольку в этом случае не потребуется сразу изучать различия между входными и выходными параметрами (в заданиях на разработку функций все параметры функций являются входными). Для языков C++ и C#, в которых нет разделения на процедуры и функции, начало изучения данной темы с функций также представляется более оправданным. Отметим, что при изучении данных языков термин «процедура», используемый

в заданиях, следует понимать просто как сокращение для функции, не возвращающей значение (аналогичное соглашение используется и в других группах заданий, связанных с разработкой процедур и функций, в частности, в группе Param — см. п. 2.6).

1.9.1. Процедуры с числовыми параметрами

В начальных заданиях данной подгруппы (1–5) требуется, по сути, реализовать процедурные «оболочки» для простейших линейных алгоритмов, подобных тем, которые были рассмотрены в группе Begin (см. п. 1.1). Основное внимание при выполнении этих заданий надо обратить на различия в описании и использовании входных и выходных параметров, а также на «разделение ролей» основной программы и разрабатываемой процедуры: ввод данных должен выполняться в *основной* программе, после чего основная программа вызывает процедуру, получает из нее результаты и выводит их. Таким образом, процедура не должна содержать операторов ввода-вывода: все необходимые исходные данные она получает из входных параметров, а все полученные результаты она возвращает в виде выходных параметров.

Следующая серия заданий (6–9) связана с алгоритмами выделения цифр из целых чисел; таким образом, эти задания представляют собой процедурные обертки заданий из группы Integer (см. п. 1.2). Задания 7–9 имеют дополнительную особенность: в процедуру, которую надо разработать, входит параметр, являющийся одновременно входным и выходным.

Наконец, последняя серия заданий (10–15) связана с обменом содержимым между переменными и включает задания на простой обмен (10), обмен с размещением минимального и максимального значения в указанном порядке (11), простейшую сортировку трех чисел (12–13) и циклический сдвиг трех чисел (14–15). Следует обратить внимание на то, что процедуры, разработанные в некоторых заданиях данной серии, оказываются полезными при выполнении других заданий (процедура Swap из задания 10 может использоваться в задании 11, процедура Minmax из задания 11 может использоваться в заданиях 12 и 13 и

т. д.). На возможность повторного использования процедур следует обратить внимание учащихся; кроме того, полезно в связи с необходимостью повторного использования рассмотреть правила объединения подобных процедур в *модули*.

Группа: Proc (процедуры с числовыми параметрами)

Разбор в классе: 4, 7, 10, 11, 12

Закрепление материала: 3, 6, 8, 13, 14

Простые задания: 1, 2, 3, 4, 10

Сложные задания: 7, 9

Наборы однотипных заданий: 1–5; 6–9; 10–15

1.9.2. Функции с числовыми параметрами

Подобно начальным заданиям из подгруппы, посвященной процедурам, задания 16–20, а также 32–33, являются «обертками» простейших линейных алгоритмов. Как и для заданий на процедуры, имеется серия заданий (29–31), связанная с выделением цифр из целых чисел.

Большой набор заданий (23–28) посвящен функциям, возвращающим *логические значения*. Алгоритмы, используемые в этих заданиях, ранее рассматривались в группах Boolean (24), If (23), While (25–28). Прочие задания данной подгруппы также имеют аналоги среди ранее рассмотренных заданий: задание 22 связано с оператором выбора, задания 21, 34 и 36 связаны с циклами с параметром (нахождение суммы чисел из заданного диапазона, вычисление факториала, вычисления числа Фибоначчи), задание 35 — с циклом с условием (вычисление двойного факториала).

Группа: Proc (функции с числовыми параметрами)

Разбор в классе: 16, 17, 21, 25, 30, 36

Закрепление материала: 18, 19, 27, 31, 34

Простые задания: 16, 17, 18, 24, 32, 33

Сложные задания: *нет*

Наборы однотипных заданий: 18–20; 24–28; 29–31; 32–33; 34–35

1.9.3. Дополнительные задания на процедуры и функции

Характерной особенностью данного раздела является объединение многих заданий в серии таким образом, что процедуры и функции, реализованные при выполнении предыдущих заданий, оказываются полезными при выполнении последующих. Таковы серии 37–39 (реализация различных вариантов функций для возведения в степень), 46–49 (реализация алгоритмов нахождения наибольшего общего делителя, наименьшего общего кратного и родственных им алгоритмов), 50–51 (обработка промежутков времени), 52–55 (обработка дат), 56–60 (обработка геометрических объектов на координатной плоскости). Целесообразно рассмотреть какую-либо одну серию (например, задания 52–55 на обработку дат), оформив все реализованные процедуры и функции в виде модуля, и предложить по аналогичной схеме выполнить задания из другой серии.

Кроме того, имеется набор заданий (40–45), посвященных реализации функций для вычисления элементарных математических функций с помощью их разложения в ряд. В отличие от аналогичных заданий группы For (см. п. 1.6), здесь требуется использовать не фиксированное количество слагаемых, а то количество, которое обеспечивает требуемую точность результата (точность определяется по величине последнего слагаемого).

Группа: Proc (дополнительные задания на процедуры и функции)

Разбор в классе: 40, 52, 53, 54

Закрепление материала: 41, 50, 51, 55, 56, 57, 58, 59

Простые задания: 37, 38, 56

Сложные задания: 39, 41–45, 47, 60

Наборы однотипных заданий: 37–39; 40–45; 46–49; 50–51

1.10.Минимумы и максимумы: группа Minmax

Данная группа является логическим продолжением группы Series, поскольку в ней также обрабатываются последовательности чисел. Однако все за-

дания в ней посвящены нахождению минимумов, максимумов и связанных с ними характеристик исходных последовательностей.

Следует отметить, что возможен вариант, при котором данная группа рассматривается уже после знакомства с массивами; в этом случае для выполнения заданий группы Minmax можно использовать массивы, что во многих случаях приводит к более простым алгоритмам. При этом учащиеся должны понимать, что все задания группы Minmax (как и группы Series) можно выполнить и без массивов, причем подобный способ будет более эффективным по объему используемой памяти.

Начальный набор заданий (1–11) связан с нахождением «обычных» минимумов и максимумов, а также их номеров. В заданиях 2–3 и 5 дополнительно требуется *вычислять* те элементы, среди которых надо найти минимумы или максимумы. Отдельная серия заданий (6–11) посвящена ситуации, когда в наборе может оказаться *несколько* минимальных или максимальных элементов (подобным свойством обладают наборы *целых* чисел). Заметим, что все элементы наборов *вещественных* чисел в заданиях предполагаются *различными*, и на это надо обратить внимание учащихся.

Следующий набор заданий (12–15) посвящен *условным* минимумам и максимумам, т. е. алгоритмам нахождения минимальных или максимальных элементов среди тех, которые удовлетворяют дополнительным условиям. Важно обратить внимание учащихся на ряд особенностей данных алгоритмов по сравнению с алгоритмами нахождения «обычных» минимумов/максимумов.

Отметим задачи 16–18 на нахождение количества элементов, расположенных перед, после или между экстремальными элементами (которые легко сводятся к задачам о нахождении *номеров* соответствующих экстремальных элементов), а также задачи 19–20 на нахождение *количества* экстремальных элементов (эти задачи, как и все предыдущие, следует решить *за один проход* исходного набора данных).

Прочие задания данной группы являются дополнительными. Среди них интерес представляет задание 21 (о нахождении среднего значения всех элементов набора, за исключением минимального и максимального) и задания 22–23 (о нахождении *нескольких* наименьших или наибольших элементов набора). В заданиях 24–30 предварительно требуется определить сами элементы, среди которых надо найти экстремальные (укажем в качестве примера задание 29, в котором надо найти максимальное количество подряд идущих минимумов).

Группа: Minmax

Разбор в классе: 1, 6, 12, 16, 19, 22

Закрепление материала: 7, 8, 13, 18, 20, 21, 23

Простые задания: 1, 2, 3, 4, 5

Сложные задания: 26, 27, 28, 29, 30

Наборы однотипных заданий: 1–5; 6–11; 12–15; 16–18; 19–20; 22–23; 24–25; 26–28; 29–30

2. Обработка массивов, строк и файлов

2.1. Одномерные массивы: группа Array

Об особой важности данной группы свидетельствует тот факт, что в нее входит наибольшее количество заданий: 140. Следует сразу обратить внимание учащихся на то, что хотя правил для описания и использования массивов немного, имеется большое число алгоритмов, которые надо усвоить для успешной работы с массивами.

Учитывая большой размер группы Array, обзор заданий будем проводить по ее подгруппам (аналогичным образом мы будем описывать и все последующие группы задачника).

2.1.1. Формирование массива и вывод его элементов

Задания 1–6 посвящены формированию массива. При выполнении этих заданий следует обратить внимание на то, что в них можно определять очередной

элемент массива через предыдущие (один или несколько), что позволяет упростить алгоритм. В этих же заданиях следует обсудить вопросы, связанные с наглядным выводом массивов на экран (эту тему необходимо обсудить и в том случае, если при выполнении заданий используется электронный вариант задачника, упрощающий действия по вводу-выводу).

Оставшиеся задания данной подгруппы (7–17) посвящены выводу элементов массива в различном порядке. Начинает эту серию задание на перебор элементов массива в обратном порядке (7). В заданиях 8–10 требуется выполнить перебор элементов массива с четными и/или нечетными значениями (в прямом или обратном порядке), а в заданиях 12–15 — с четными и/или нечетными номерами. В последнем случае естественно использовать выражения 2**i* и 2**i* – 1 для перебора всех четных и нечетных номеров соответственно, однако при этом важно правильно определить диапазон изменения параметра *i*.

Необходимо учитывать следующую особенность формулировок учебных заданий: в них всегда говорится о *порядковых номерах* элементов (а не об их индексах), причем предполагается, что *первый элемент массива имеет порядковый номер 1*. Эта особенность позволяет единообразно формулировать задания для языков с различным способом индексирования. В то же время она приводит к дополнительным сложностям в тех языках, в которых индексация массивов жестко задана и начинается от 0 (как в C++, C# и Visual Basic .NET), поскольку для этих языков значения *индексов* элементов отличаются от их *порядковых номеров* (первый по порядку элемент имеет индекс 0, второй — индекс 1 и т. д.). Для языков Pascal и Visual Basic подобные проблемы не возникают, так как в них можно индексировать элементы массивов от 1.

Задания 16–17 являются дополнительными заданиями повышенного уровня сложности. Например, в задании 17 следует вывести элементы массива *A* (размера *N*) в следующем порядке: A_1, A_2, A_N, A_{N-1}, A_3, A_4, A_{N-2}, A_{N-3}, … . В этих заданиях впервые возникает необходимость перебирать в цикле не отдельные элементы массива, а группы элементов (в задании 17 это четверки элементов).

Далее, в подобной ситуации следует выполнять дополнительные действия после завершения цикла, чтобы обработать оставшиеся элементы, которые не удалось включить в группу (в задании 17 количество таких элементов равно $N \bmod 4$, где mod обозначает операцию взятия остатка от деления).

Группа: Array (формирование массива и вывод его элементов)

Разбор в классе: 2, 4, 5, 7, 9, 14

Закрепление материала: 3, 6, 8, 11, 15

Простые задания: 1, 2, 7

Сложные задания: 14, 15, 16, 17

Наборы однотипных заданий: 1–6; 8–10; 11–15; 16–17

2.1.2. Анализ элементов массива

В данной подгруппе не требуется преобразовывать исходный массив: достаточно проанализировать его элементы.

Простейшими заданиями такого рода являются задания 18–19, связанные со сравнением элементов массива между собой. В заданиях 20–23 необходимо обработать элементы массива, расположенные между элементами с указанными порядковыми номерами (или вне указанного диапазона номеров).

Задания 24–27 связаны с анализом закономерности чередования элементов; во всех этих заданиях может оказаться полезным использование оператора явного выхода из цикла.

Особо следует выделить задания 32–36 на нахождение локальных экстремумов (минимумов и максимумов) и задания 37–39 на определение участков монотонного убывания/возрастания, которые также сводятся к нахождению локальных экстремумов. В этих заданиях впервые можно применить полезный прием «фиктивных», или барьерных, элементов, который в данном случае позволит единообразно обработать все элементы массива, несмотря на то что первый и последний элемент имеют по одному соседу, а прочие элементы — по два. Кроме того, в заданиях 36 и 39 полезно указать условие, позволяющее лег-

ко проверить, является ли элемент A_K локальным экстремумом: для этого выражение $(A_K - A_{K-1})*(A_K - A_{K+1})$ должно быть *положительным*.

Решения задач 40–43 сводятся к перебору элементов массивов в простом цикле, тогда как для решения задач 44–50 необходимо использовать *вложенные циклы*, и в этом состоит их главная сложность. По указанной причине в пособиях [1, 6, 9] дается полное решение типовой задачи 47. Следует заметить, что задачи 40–43 можно решить и не используя массивы; учащимся можно предложить реализовать такое решение и сравнить полученный алгоритм с алгоритмом, использующим массив.

Группа: Array (анализ элементов массива)

Разбор в классе: 19, 25, 26, 32, 37, 39, 43, 44, 47

Закрепление материала: 24, 27, 33, 36, 38, 42, 45, 49

Простые задания: 18, 19, 20, 21, 28, 29, 30, 31

Сложные задания: 47, 48

Наборы однотипных заданий: 18–19; 20–23; 24–27; 28–29; 30–31; 32–36; 37–39; 40–42; 45–50

2.1.3. Работа с несколькими массивами

Данная подгруппа содержит достаточно простые задания (51–62), в которых требуется по исходному массиву (одному или нескольким) сформировать новый массив. При выполнении этих заданий следует использовать переменную, содержащую информацию о текущей «заполненности» создаваемого массива (после завершения формирования массива в этой переменной будет содержаться *размер* созданного массива).

Особое место в этой подгруппе занимают задания 63–64, посвященные алгоритму *слияния* упорядоченных массивов, который обязательно следует разобрать на занятиях. В частности, следует указать на возможность применения в данном алгоритме приема барьерного элемента (для автоматической обработки ситуации, в которой оказывается программа при достижении конца одного из

исходных массивов). В то же время следует отметить, что данную задачу можно решить, не используя барьерные элементы.

Группа: Array (работа с несколькими массивами)

Разбор в классе: 54, 60, 63

Закрепление материала: 55, 59, 64

Простые задания: 51, 52, 53

Сложные задания: 63, 64

Наборы однотипных заданий: 55–57; 58–61; 63–64

2.1.4. Преобразование массива

Данная подгруппа начинается с заданий, в которых требуется преобразовать элементы массива по определенному правилу (65–70); при этом надо сразу указать на опасность «преждевременной порчи» изменяемых элементов массива и на вызванную этим необходимость использования вспомогательных переменных. Вместе с тем надо отметить, что все задания этой группы следует выполнять без использования вспомогательных массивов.

Отметим задание 71 об инвертировании массива (изменении порядка его элементов на обратный) и аналогичные задания 72–73. Эти задания сводятся к перемене местами элементов, симметричных относительно середины массива. При реализации данного алгоритма для массива из N элементов учащиеся часто делают ошибку, выполняя указанную перемену не N div 2 раз (где div обозначает операцию целочисленного деления), а N раз. В результате преобразованный массив не будет отличаться от исходного, поскольку каждая пара симметричных элементов будет меняться местами дважды.

Следующая серия задания посвящена простому (79–82) и циклическому (83–86) *сдвигу* элементов массива на одну или несколько позиций. При выполнении этих заданий следует обратить внимание на необходимость перебора элементов с конца массива при выполнении правого сдвига и с начала массива при выполнении левого сдвига, а также на необходимость использования вспомогательной переменной при выполнении циклического сдвига. Особо надо ос-

тановиться на проблемах, связанных с циклическим сдвигом на несколько позиций, и привести несколько вариантов их решения.

Отметим задания 87–89 о вставке элемента в упорядоченный массив с сохранением его упорядоченности. В задании 89 заранее неизвестно, какой элемент нарушает упорядоченность, что повышает сложность данного задания.

Задания 90–100 посвящены алгоритмам *удаления* элементов из массива: от самых простых («Удалить элемент с указанным номером» — задание 90) до достаточно сложных (например, «Удалить из массива все элементы, встречающиеся менее трех раз» — задание 98). Следует обратить внимание учащихся на то, что удаление сводится к левому сдвигу оставшихся элементов. Рассмотрев какое-либо задание средней сложности (например, «Удалить из массива все нечетные числа» — задание 92), следует показать, как эффективно реализовать подобное удаление, просматривая массив один раз и сразу перемещая оставляемые элементы на их окончательные позиции.

Задания 101–111 посвящены *вставке* элементов в массив. Здесь следует отметить, что вставка сводится к правому сдвигу элементов и что важно различать ситуации «вставки перед» и «вставки после» данного элемента. В данном случае, как и при изучении заданий на удаление элементов, следует рассмотреть какое-либо из заданий средней сложности (например, «Перед каждым положительным элементом массива вставить элемент с нулевым значением» — задание 108) и обсудить эффективный алгоритм вставки, обеспечивающий немедленное перемещение каждого элемента на его окончательную позицию. Заметим, что при реализации подобного алгоритма может потребоваться предварительно просмотреть весь массив (в задании 108 в ходе этого предварительного просмотра массива надо подсчитать количество его положительных элементов). Все задания на вставку элементов следует выполнять без использования вспомогательных массивов.

Завершающая серия заданий, посвященных преобразованию массива, связана с сортировкой (112–115). Задания 112–114 сформулированы таким обра-

зом, что позволяют по самой формулировке ознакомиться с требуемым алгоритмом: сортировкой *простым обменом* (пузырьковой сортировкой) в задании 112, сортировкой *простым выбором* в задании 113, сортировкой *простыми вставками* в задании 114. При выполнении каждого задания требуется выводить содержимое массива после каждого прохода, что обеспечит контроль за выполняемыми действиями. Дополнительное задание 115 описывает способ сортировки, не требующий изменения порядка элементов исходного массива (преобразуется вспомогательный *индексный массив*).

Группа: Array (преобразование массива)

Разбор в классе: 65, 69, 71, 79, 82, 84, 86, 87, 90, 92, 101, 105, 108, 112

Закрепление материала: 70, 72, 80, 81, 84, 88, 91, 93, 102, 109, 113

Простые задания: 65, 79, 80, 90, 101, 102

Сложные задания: 76, 77, 89*, 96, 97, 98, 99, 100, 115

Наборы однотипных заданий: 65–67; 69–73; 79–82; 83–86; 87–89; 92–94; 95–100; 101–105; 106–111; 112–114

2.1.5. Серии целых чисел

Данная подгруппа заданий специально предназначена для того, чтобы предоставить преподавателю большой набор однотипных задач среднего уровня сложности, посвященных преобразованию массива. Понятие *серии* элементов массива (набор идущих подряд одинаковых элементов) и ее *длины* (количество этих элементов; длина может быть равна 1) дается в задаче 116. Задачи 117–122 связаны с изменением длины одной или нескольких серий, задачи 123–124 — с переменой местами двух серий, задачи 125–127 — с заменой серии на один нулевой элемент, задачи 128–130 — с обработкой серий наибольшей длины. В [1, 6, 9] приводится решение задачи 116. Алгоритм этого решения (использующий два вспомогательных массива) после несложных модификаций можно применить и для решения остальных задач данной подгруппы. В качестве альтернативного варианта решения задач 117–130 можно предложить разработать алго-

ритм, не использующий вспомогательные массивы, однако для ряда заданий (например, 128–130) подобный алгоритм будет достаточно сложным.

Группа: Array (серии целых чисел)

Разбор в классе: 116, 119, 124

Закрепление материала: 117, 120, 128

Простые задания: 116

Сложные задания: 128, 129, 130

Наборы однотипных заданий: 117–120; 121–124; 125–130

2.1.6. Множества точек на плоскости

Эта подгруппа заданий также является дополнительной. Ее особенность — в том, что требуется обрабатывать набор точек на плоскости, для хранения которого можно использовать либо два числовых массива (*X* и *Y*), либо один массив *A записей* (структур) с полями *X* и *Y*.

Задачи 131–133 решаются простым перебором исходного набора точек с применением алгоритма нахождения минимума или максимума.

Задачи 134–138 также посвящены нахождению минимумов и максимумов, однако для их решения необходимо организовать перебор пар или троек точек во *вложенных циклах*.

Наконец, задачи 139–140 посвящены сортировке наборов точек (в каждой из этих задач указывается свой вариант отношения порядка для точек на плоскости).

Группа: Array (множества точек на плоскости)

Разбор в классе: 132, 136, 139

Закрепление материала: 133, 135, 138, 140

Простые задания: 131

Сложные задания: 139, 140

Наборы однотипных заданий: 131–133; 134–138; 139–140

2.2. Двумерные массивы (матрицы): группа Matrix

Данная группа является продолжением группы Array и имеет аналогичную структуру. Во всех подгруппах данной группы (кроме последней) фактически применяются те же алгоритмы, что и для заданий группы Array, однако в данной ситуации они должны быть реализованы для строк или столбцов матрицы. При этом естественным образом возникают вложенные циклы.

Для того чтобы уменьшить количество ошибок, часто возникающих при организации вложенных циклов, рекомендуется предложить учащимся придерживаться ряда соглашений, связанных с именованием параметров циклов. Так, параметру цикла, перебирающего строки, следует присваивать имя *i* (или данное имя с последующим номером: *i1*, *i2* и т. д.), а параметру цикла, перебирающего столбцы, следует присваивать имя *j* (возможно, с последующим номером). В индексах элементов матриц следует придерживаться «математического» варианта нумерации строк и столбцов: $A[i, j]$ (первыми указывается номер *строки*). Следует обратить внимание учащихся на связь между именем индекса и диапазоном его изменения: в циклах по *i* (т. е. по строкам) параметр *i* практически всегда будет изменяться от 1 до *M*, а в циклах по *j* (по столбцам) — от 1 до *N* (если в языке используется индексирование от 0, то диапазон изменится соответствующим образом: от 0 до $M - 1$ для *i*, от 0 до $N - 1$ для *j*). Следование подобным соглашениям существенно уменьшит число ошибок, связанных с неверной индексацией, а также упростит проверку текстов программ.

2.2.1. Формирование матрицы и вывод ее элементов

Формированию матрицы посвящены первые шесть заданий. В заданиях 1–4 требуется сформировать матрицу с одинаковыми строками или столбцами, причем в заданиях 1–2 значения этих строк (столбцов) должны вычисляться по указанной простой формуле (например, 10**i* для *i*-й строки в задании 1), а в заданиях 3–4 эти значения должны браться из исходного одномерного массива. В заданиях 5–6 матрица должна быть сформирована таким образом, чтобы эле-

менты каждой ее строки (или столбца) образовывали арифметическую (соответственно, геометрическую) прогрессию.

Остальные задания (7–16) посвящены выводу элементов матрицы в различном порядке. Помимо этих заданий желательно обсудить приемы, позволяющие вывести на экран *всю матрицу*, обеспечив при этом правильное выравнивание ее элементов по столбцам.

Наряду с простыми заданиями 7–13, в которых требуется вывести одну или несколько конкретных строк (столбцов) матрицы в определенном порядке, подгруппа содержит сложные (дополнительные) задания, в которых обход элементов матрицы надо проводить другими способами: «уголками» (13–14) и «по спирали» (15–16).

Группа: Matrix (формирование матрицы и вывод ее элементов)

Разбор в классе: 1, 4, 7, 10

Закрепление материала: 2, 3, 5, 8, 12

Простые задания: 1, 2, 7, 8

Сложные задания: 13, 14, 15, 16

Наборы однотипных заданий: 1–2; 3–4; 5–6; 7–8; 9–12; 13–16

2.2.2. Анализ элементов матрицы

В большинстве заданий данной подгруппы необходимо определить некоторую характеристику каждой строки или столбца матрицы, а затем обработать полученные характеристики (типичным в этом отношении является задание 27: «Найти максимальный элемент среди минимальных элементов строк данной матрицы»). При выполнении подобных заданий желательно не использовать вспомогательные массивы, ограничиваясь несколькими вспомогательными простыми переменными. Следует обратить внимание на важность выбора правильного места для *инициализации* вспомогательных переменных.

В некоторых заданиях (32–35) требуется найти первую или последнюю строку (или столбец) матрицы, удовлетворяющую некоторому условию. В по-

добных заданиях целесообразно использовать оператор явного выхода из внешнего цикла.

Следует обратить внимание на задания 36–37, в которых требуется определить количество строк (столбцов), «похожих» на первую строку (столбец) данной матрицы, причем похожими считаются строки (столбцы), имеющие одинаковые множества значений элементов: например, строки (1, 1, 5, 3) и (5, 5, 1, 3) являются похожими. Эти задания целесообразно рассмотреть в случае, если используемый язык программирования предоставляет средства для работы с *множествами* (как, например, Pascal). В противном случае решение заданий будет довольно сложным, так как потребует сравнений каждого элемента одной строки со всеми элементами другой строки (и наоборот).

Задания 38–41 связаны с нахождением одинаковых элементов в строках/столбцах, а задания 42–45 — с проверкой *упорядоченности* элементов в строках/столбцах.

Отметим также задание 46, в котором в целочисленной матрице требуется найти элемент, являющийся максимальным в своей строке и минимальным в своем столбце. Учащимся можно предложить доказать корректность этого задания (т. е. тот факт, что если даже в матрице имеется *несколько* подобных элементов, то все они имеют одинаковое значение).

Группа: Matrix (анализ элементов матрицы)
Разбор в классе: 19, 32, 36, 43
Закрепление материала: 20, 23, 34, 37, 44
Простые задания: 17, 18, 19, 20
Сложные задания: 36, 37, 40, 41, 46
Наборы однотипных заданий: 17–18; 19–22; 23–28; 29–31; 32–35; 36–37; 38–41; 42–45

2.2.3. Преобразование матрицы

При выполнении заданий из данной подгруппы не следует применять вспомогательные двумерные массивы. Алгоритмы, используемые при выпол-

нении заданий, ранее уже были рассмотрены для одномерных массивов; в данной ситуации в качестве объектов преобразования выступают не отдельные элементы массива, а строки (или столбцы) матрицы, поэтому соответствующие алгоритмы следует дополнить внутренним циклом, позволяющим обработать требуемым образом все элементы этих строк/столбцов.

Задания 47–60 посвящены *перемене местами* строк (или столбцов) данной матрицы. При этом в заданиях 55–60 подобная перемена должна выполняться для всех строк/столбцов; например, в задании 59 следует зеркально отразить элементы матрицы относительно ее горизонтальной оси симметрии.

Задания 61–67 посвящены удалению строк (или столбцов) матрицы, а задания 68–73 — вставке в данную матрицу новых строк/столбцов.

Следует обратить внимание на задания 74–75, в которых надо найти и определенным образом изменить локальные минимумы (максимумы) данной матрицы. Поскольку граничные элементы матрицы имеют меньше соседних элементов, чем внутренние (угловые элементы имеют 3 соседа, остальные граничные элементы — 5 соседей, а внутренние элементы — 8 соседей), для единообразной обработки всех элементов матрицы очень удобно использовать фиктивные элементы, которыми следует «окружить» исходную матрицу. Вторая проблема, которую следует решить в этих заданиях, связана с тем, что изменение элементов матрицы может привести к неправильной обработке соседних с ними элементов. Простейший способ решения этой проблемы — использование вспомогательной матрицы, и в данном случае такой способ можно считать допустимым (об этом явно сказано в условии), так как вариант решения, не использующий вспомогательную матрицу, является существенно более сложным для реализации.

Задания 76–79 посвящены сортировке строк (столбцов) матрицы по определенному критерию. Алгоритм сортировки в заданиях не уточняется. При выполнении этих заданий полезно вспомнить об индексной сортировке, описанной в задании 115 из группы Array (см. п. 2.1.4), и обсудить ее преимущества

при сортировке «больших объектов» (в данном случае — строк или столбцов). В заданиях 78–79 полезно предварительно сформировать вспомогательный «массив ключей», который использовать при проведении сортировки.

Группа: Matrix (преобразование матрицы)

Разбор в классе: 56, 59, 65, 72, 74, 76

Закрепление материала: 55, 58, 60, 66, 70, 75, 77

Простые задания: 47, 48, 61, 62, 68, 69

Сложные задания: 67, 74, 75, 78, 79

Наборы однотипных заданий: 47–48; 49–54; 55–60; 61–66; 68–73; 74–75; 76–79

2.2.4. Диагонали квадратной матрицы

Данная подгруппа менее тесно связана с заданиями на одномерные массивы, чем предыдущие подгруппы группы Matrix. В заданиях этой подгруппы требуется обработать фрагменты матрицы, связанные с ее *диагоналями*. Определения главной и побочной диагонали квадратной матрицы даются во вводных заданиях 80–81. Следующий набор заданий (82–87) посвящен обработке всех диагоналей, параллельных главной (или побочной). Ввиду сложности такой обработки, в [1, 6, 9] приводится решение типовой задачи 82, причем в двух вариантах.

Необходимо обратить особое внимание на серию заданий 89–93, в которых требуется обнулить часть элементов, расположенных выше/ниже главной/побочной диагонали (рассматриваются различные комбинации указанных условий) *без использования условных операторов*. Эти задания позволяют еще раз обсудить приемы, связанные с составлением вложенных циклов в такой ситуации, когда диапазон изменения параметра внутреннего цикла зависит от текущего значения параметра внешнего цикла.

Завершающая часть заданий на матрицы (96–100) связана с их зеркальным отражением относительно главной или побочной диагонали или поворотом на угол 90, 180 или 270 градусов. При выполнении заданий желательно не исполь-

зовать вспомогательные матрицы; вместо этого следует организовать перебор *пар* элементов (или даже *четверок* в случае заданий 99–100), циклически переходящих друг в друга.

Группа: Matrix (диагонали квадратной матрицы)

Разбор в классе: 81, 82, 88, 90, 92, 96

Закрепление материала: 80, 85, 89, 91, 94, 97

Простые задания: 80, 81

Сложные задания: 98, 99*, 100*

Наборы однотипных заданий: 80–81; 82–87; 88–91; 92–95; 96–100

2.3. Символы и строки: группа String

Задания данной группы охватывают большинство алгоритмов, связанных с обработкой символов и строк; при этом они сформулированы таким образом, чтобы их можно было использовать для любого из распространенных языков программирования: Pascal, Visual Basic, C++, C#. При использовании языка Visual Basic версий 5 и 6 в качестве символьных данных рекомендуется применять данные типа String * 1 (хотя допустимым является и обычный тип String).

2.3.1. Символы и их коды. Формирование строк

Данная подгруппа начинается с серии заданий на обработку отдельных символов (1–6): преобразование кода в символ и обратно (1–2), перебор символов, идущих подряд в кодовой таблице (3–5), распознавание цифровых и буквенных символов (6).

Затем идут задания на формирование и простейшую обработку строк (7–12), в которых требуется использовать лишь операцию сцепления, а также функцию, возвращающую длину строки. Следует обратить внимание на задачу 10, в которой требуется сформировать строку, содержащую символы исходной строки в обратном порядке. Три варианта решения этой задачи приводятся в [1, 6, 9]; полезно сравнить эти варианты с решением задачи на инвертирование массива (задача 71 из группы Array — см. п. 2.1.4).

Группа: String (символы и их коды, формирование строк)

Разбор в классе: 1, 2, 3, 4, 6, 7, 10

Закрепление материала: 5, 8, 9, 11

Простые задания: 1, 2, 3, 7

Сложные задания: *нет*

Наборы однотипных заданий: 8–9; 10–12

2.3.2. Посимвольный анализ и преобразование строк. Строки и числа

Задачи 13–18 решаются с помощью посимвольного просмотра исходной строки; при этом в задачах 16–18 требуется соответствующим образом *преобразовать* некоторые символы. Следует отметить, что в некоторых языках для преобразования символов в строке требуется использовать особые средства (например, в Visual Basic для этого предназначен специальный *оператор* Mid, а в C# для возможности изменения символов строки необходимо использовать особый класс StringBuilder, поскольку строки типа string являются *неизменяемыми*). При выполнении указанных заданий в некоторых языках (например, Pascal и Visual Basic) удобно использовать оператор выбора.

Оставшиеся задачи данной подгруппы (19–25) связаны с преобразованием строк в числа и чисел в строки. Отметим задачу 19, в которой требуется распознать, содержит ли исходная строка число определенного типа: целого или вещественного (с дробной частью); решение этой задачи приводится в [1, 6, 9]. Следует обратить внимание учащихся на тот факт, что необходимость в подобном распознавании часто возникает в реальных программах при организации надежного ввода числовых данных.

В задачах 19–23, связанных с преобразованием цифровых символов в соответствующие числовые значения, полезно указать простой способ подобного преобразования (достаточно вычесть из кода цифрового символа код символа «0»), а также обратного преобразования однозначного числа в соответствую-

щий ему символ (достаточно взять символ с кодом, равным сумме кода символа «0» и данного числа).

Задачи 24–25, связанные с преобразованием десятичного представления числа в двоичное и обратно, могут рассматриваться как дополнительные.

Группа: String (посимвольный анализ и преобразование строк; строки и числа)

Разбор в классе: 13, 17, 19, 20, 22

Закрепление материала: 14, 18, 21, 23

Простые задания: 13, 14

Сложные задания: 19, 24, 25

Наборы однотипных заданий: 13–15; 16–18; 20–21; 24–25

2.3.3. Обработка строк с помощью стандартных функций. Поиск и замена

Начальные задания данной подгруппы (26–30) позволяют продемонстрировать применение стандартных функций, предназначенных для обработки строк и имеющихся в любом языке программирования (*удаление* фрагмента строки, получение *копии* подстроки исходной строки, *вставка* в строку новой подстроки в требуемой позиции). При выполнении заданий 28–30, связанных с вставкой новых символов или строк в требуемые позиции исходной строки, следует обратить внимание на важность перебора символов исходной строки *в обратном порядке*.

Завершающая серия заданий данной подгруппы (31–40) посвящена различным вариантам *поиска и замены*, основанным на применении стандартных функций используемого языка программирования. Она содержит задания на проверку наличия требуемой подстроки (31), определение количества вхождений требуемой подстроки в исходную строку (32), удаление или замену первого, последнего или всех вхождений требуемой подстроки (33–38). Задания 39–40, связанные с поиском в строке пробелов, могут рассматриваться как дополнительные. Следует обратить внимание на сложности в организации поиска *по-*

следнего вхождения подстроки для тех языков, в которых отсутствуют соответствующие стандартные функции (например, Pascal или Visual Basic версии 5), а также на стандартную схему, использующую цикл с условием при организации удаления или замены *всех* вхождений требуемой подстроки в исходной строке.

Группа: String (обработка строк с помощью стандартных функций; поиск и замена)

Разбор в классе: 26, 27, 28, 31, 32, 34, 38

Закрепление материала: 29, 33, 35, 36, 37

Простые задания: 31

Сложные задания: *нет*

Наборы однотипных заданий: 28–30; 33–38; 39–40

2.3.4. Анализ и преобразование слов в строке

Задания данной подгруппы (41–57) посвящены алгоритмам, относящимся к разбиению исходной строки на отдельные слова. При решении большинства заданий можно использовать «универсальный» алгоритм подобного разбиения, основанный на применении стандартных строковых функций (данный алгоритм подробно описан в [1, 6, 9]). Однако во многих ситуациях более эффективными будут «специализированные» алгоритмы, основанные на посимвольном анализе исходной строки (в частности, на распознавании начала слова по двум соседним символам, первый из которых является пробелом, а второй — не является, а конца слова — по паре символов «не пробел»–«пробел»). Подобные варианты решений также рассмотрены в [1, 6, 9]. Задания 53 и 55–56 связаны с распознаванием в строке *знаков препинания*; для быстрого распознавания подобных символов можно использовать подходящее множество (в языке Pascal) или строку (в языке Visual Basic), содержащие все знаки препинания; можно также использовать соответствующую функцию, если она имеется в языке (например, метод IsPunctuation класса char в языках платформы .NET).

Группа: String (анализ и преобразование слов в строке)

Разбор в классе: 41, 42, 47, 50, 52, 53

Закрепление материала: 43, 44, 46, 51, 54, 57

Простые задания: 41, 53, 54

Сложные задания: 48, 49, 51, 55, 56

Наборы однотипных заданий: 41–46; 48–50; 53–54; 55–56

2.3.5. Дополнительные задания на обработку строк

Задания 58–61 в данной подгруппе посвящены обработке имен файлов. Подобную обработку (выделение собственно имени, расширения или требуемого фрагмента пути) можно проводить либо с помощью функций поиска или посимвольного анализа, либо с применением специальных функций для работы с файловыми именами, предусмотренными в используемом языке программирования (например, в языке Delphi Pascal для этих целей служит набор функций вида ExtractFileDir, ExtractFileName, ExtractFileExt и т. д., а в языках платформы .NET — специальный класс Path).

Следующий набор заданий (62–67) посвящен шифрованию строк. Рассматриваются два вида шифрования: циклическая замена каждой буквы на букву, расположенную в алфавите на *K*-й позиции после шифруемой буквы (задания 62–65) и «перемешивание» символов в строке (66–67). При реализации циклической замены символов полезно проанализировать вариант, использующий общую формулу для *всех* символов (в том числе и для перемещаемых «назад», как, например, буква «я»); в эту формулу будет входить операция взятия остатка от целочисленного деления.

Наконец, отметим задания 69–70, связанные с проверкой правильности расстановки скобок в исходной строке (в задании 69 следует анализировать скобки одного типа «()», а в задании 70 — скобки трех типов «()», «[]» и «{}»). Если в задании 69 достаточно завести счетчик открывающих скобок (который надо увеличивать при появлении открывающей скобки и уменьшать при появлении закрывающей скобки, и при этом следить, чтобы он никогда не принимал

отрицательные значения и после просмотра строки был равен 0), то в задании 70 необходимо воспользоваться вспомогательной строкой-«стеком» для хранения открывающих скобок всех типов (верхняя скобка из «стека» удаляется при обнаружении закрывающей скобки того же типа) и учесть различные особые ситуации, которые могут при этом возникать (например, появление закрывающей скобки в случае пустого «стека»).

Группа: String (дополнительные задания на обработку строк)

Разбор в классе: 58, 62, 63, 66, 70

Закрепление материала: 60, 64, 67, 69

Простые задания: 58, 59

Сложные задания: 63, 64, 65, 70

Наборы однотипных заданий: 58–61; 62–65; 66–67

2.4. Двоичные файлы: группа File

Во всех рассматриваемых языках программирования имеется четкое разделение файловых структур на два класса: двоичные файлы и текстовые файлы. Двоичным файлам посвящена группа заданий File, текстовым — группа Text. Начинать изучение файлов целесообразно с группы File, так как в некоторых заданиях группы Text используются не только текстовые, но и двоичные файлы.

Основная особенность, отличающая двоичные файлы от текстовых, состоит в возможности *прямого доступа* к содержимому файла, из которой вытекает также возможность одновременного открытия двоичного файла на чтение и на запись. Кроме того, как правило, двоичные файлы хранят элементы одного типа (что позволяет быстро определять размер файла не в байтах, а в «элементах», и быстро переходить к элементу с требуемым номером). По этой причине в языке Pascal двоичные файлы, содержащие однотипные элементы, называются *типизированными*.

В языках платформы .NET имеется принципиальная возможность записывать в двоичные файлы данные разных типов, однако при этом могут возник-

нуть проблемы с правильной интерпретацией содержимого этих файлов при их последующем чтении. В заданиях на обработку двоичных файлов всегда предполагается, что все элементы файлов имеют один и тот же тип (т. е. что все двоичные файлы являются «типизированными» в смысле файлов языка Pascal).

2.4.1. Основные операции с двоичными файлами

Данная подгруппа может быть разделена на две большие части.

Первая часть (задания 1–24) знакомит со стандартными операциями по созданию/открытию/закрытию файла и вводу-выводу его элементов.

Особое место в подгруппе занимает задание 1, в котором предлагается проверить, является ли указанная строка допустимым именем файла. При первом знакомстве с файлами это задание можно не рассматривать, поскольку его оптимальный метод решения заключается в том, чтобы «переложить» проверку допустимости имени на операционную систему, попытавшись создать файл с требуемым именем; при этом основная проблема состоит в «перехвате» сообщения о возможной ошибке, если имя не является допустимым. Таким образом, обсуждение этого задания естественным образом подводит к механизму пользовательской обработки ошибок, возникающих в программе (конструкция On Error в «традиционном» Visual Basic, директива компилятора {$I–} в Delphi Pascal, try-блоки во всех языках, допускающих обработку исключений). Следует заметить, что при выполнении других заданий на обработку файлов (как двоичных, так и текстовых) всегда можно реализовать алгоритм, не требующий пользовательской обработки ошибок.

Однотипные задания 2–3 посвящены созданию нового файла, задание 4 — проверке существования требуемого файла (при этом достаточно воспользоваться соответствующей стандартной функцией, предусмотренной во всех рассматриваемых языках, — Dir в Visual Basic, FileExists в Delphi Pascal и т. д.), задание 5 — определению количества элементов в файле (следует обратить внимание на то, что во всех заданиях под размером двоичного файла понимается его размер «в элементах»). Заметим, что в заданиях 4–5 не требуется организо-

вывать считывание данных из файла (а при выполнении задания 4 нет необходимости даже в открытии файла). Наконец, в заданиях 6–7 требуется выполнить чтение элементов с указанными номерами; это приводит к рассмотрению прямого способа доступа к элементам файла. Таким образом, при выполнении заданий 2–7 учащиеся знакомятся практически со всеми приемами работы с двоичными файлами (за исключением нескольких специальных возможностей, которые будут рассмотрены позднее).

Задания 8–24 посвящены закреплению полученных знаний. Заметим, что практически во всех этих и последующих заданиях, связанных с обработкой файлов (за исключением задания 9), не требуется проверять существование исходных файлов: предполагается, что они всегда существуют. При этом задания 8–15 можно считать основными, а оставшиеся задания (16–24) — дополнительными. Дополнительные задания представляют собой переформулировки «на языке файлов» ранее рассмотренных для массивов задач на нахождение локальных экстремумов, анализ возрастающих/убывающих последовательностей и нахождение серий одинаковых чисел.

Вторая часть (задания 25–41) содержит задачи на *преобразование* существующего файла. Поскольку двоичные файлы можно открыть одновременно на чтение и запись, в ряде простых случае можно выполнить нужное преобразование непосредственно в исходном файле. Однако более простой и универсальный способ преобразования файла состоит во введении вспомогательного (временного) файла, в который записываются преобразованные данные, после чего исходный файл уничтожается, а имя вспомогательного файла заменяется на имя исходного. Оба способа преобразования файла подробно рассмотрены в [1, 6, 9] на примере решения задачи 25. Следует отметить, что в задаче 26 проще использовать непосредственное преобразование исходного файла, тогда как в задачах 27–28 (как и почти во всех последующих задачах этой части) — способ, основанный на применении вспомогательного файла.

Задания 29–34 связаны с удалением элементов из файла. Заметим, что для удаления *конечной* (завершающей) части файла (задания 29–30) во многих языках имеются стандартные процедуры, позволяющие обойтись без вспомогательного файла (процедура Truncate в языке Pascal, метод SetLength класса FileStream в языках платформы .NET).

Задания 35–39 связаны с добавлением в файл новых элементов, а 40–41 — с заменой определенных элементов на наборы других элементов. Во всех этих заданиях удобно использовать вспомогательные файлы, хотя задания, связанные с добавлением элементов *в конец* файла (36–37), можно выполнить и без применения вспомогательного файла.

Группа: File (основные операции с двоичными файлами)
Разбор в классе: 2, 4, 5, 6, 7, 25, 29, 31, 36, 40, а также 1
Закрепление материала: 3, 8, 10, 12, 15, 28, 30, 37, 41
Простые задания: 4, 5, 6, 7, 14, 15, 29, 30
Сложные задания: 1, 17, 23, 24, 27
Наборы однотипных заданий: 11–13; 14–15; 18–19; 21–22; 23–24; 29–32; 33–34; 36–36; 38–39; 40–41

2.4.2. Обработка нетипизированных двоичных файлов

Данная подгруппа содержит задания на обработку нетипизированных файлов. В этих заданиях (42–47) заранее неизвестен тип элементов исходных файлов, причем для выполнения задания эта информация и не требуется, поскольку файлы надо обработать «как единое целое»: поменять местами содержимое двух файлов (задание 42), создать копию существующего файла (43), заменить содержимое одного из исходных файлов на содержимое другого (44–45), дописать к одному файлу содержимое другого (46–47). Заметим, что при выполнении задания 42 вместо перемены местами содержимого файлов достаточно выполнить *переименование* исходных файлов, воспользовавшись при этом вспомогательным промежуточным именем. Для эффективного выполнения заданий на копирование содержимого целесообразно воспользоваться специальными

средствами, предусмотренными в большинстве языков программирования (например, процедурами BlockRead и BlockWrite в языке Pascal или аналогичными методами блочного чтения/записи в языках платформы .NET). В случае отсутствия подобных средств (как в Visual Basic) можно работать с нетипизированными файлами как с *типизированными файлами, состоящими из отдельных байт*, и выполнять их побайтное копирование. В [1, 6, 9] в качестве примера приведены различные варианты решения задачи 43.

Следует заметить, что в дальнейшем нетипизированные файлы в заданиях не используются, поэтому при первоначальном знакомстве с двоичными файлами данную подгруппу можно не рассматривать.

Группа: File (обработка нетипизированных двоичных файлов)
Разбор в классе: 43
Закрепление материала: 42, 44
Простые задания: 42
Сложные задания: 47
Наборы однотипных заданий: 44–45

2.4.3. Работа с несколькими числовыми файлами. Файлы-архивы

Во всех заданиях данной подгруппы требуется обрабатывать несколько типизированных файлов (входных или выходных). Задание 48, связанное с поэлементным копированием нескольких исходных файлов в один результирующий файл, подробно проанализировано в [1, 6, 9]; в нем для обработки набора исходных файлов целесообразно использовать *массив* файловых переменных. Аналогичным, хотя и более сложным, является задание 49 (в нем, в отличие от задания 48, исходные файлы могут иметь различный размер).

В заданиях 50–51 требуется выполнить слияние отсортированных файлов в результирующий файл с сохранением упорядоченности элементов. Полезно сравнить разработанный алгоритм решения этих задач с алгоритмом решения задач на слияние массивов (задания 63–64 из группы Array — см. п. 2.1.3).

Задания 52–57 связаны с объединением данных из нескольких однотипных файлов в один файл-*архив* и последующим извлечением из архива сохраненных в нем данных. Эти шесть заданий составляют две серии по три задания; в каждой серии используется свой формат хранения данных в файле-архиве. Первое задание серии (52 и 55) связано с формированием файла-архива, остальные задания — с извлечением из него данных, причем последнее задание серии (54 и 57) является более сложным.

Группа: File (работа с несколькими числовыми файлами, файлы-архивы)

Разбор в классе: 48, 50, 52, 53

Закрепление материала: 49, 51, 55, 56

Простые задания: 48

Сложные задания: 50, 51, 54, 57

Наборы однотипных заданий: 50–51; 52–57

2.4.4. Символьные и строковые файлы

Данная подгруппа содержит задания на обработку двоичных файлов с символьной информацией (в предыдущих подгруппах группы File использовались только файлы с числовой информацией; исключение составляли некоторые задания на нетипизированные файлы, в которых тип элементов заранее не указывался).

Задания на обработку файлов, элементами которых являются символы (58–62), подобны аналогичным заданиям на обработку числовых файлов.

Двоичные файлы, содержащие строки (в задачнике такие файлы называются *строковыми*), имеют ряд особенностей. Прежде всего, надо учитывать, что строковые файлы отличаются от стандартных *текстовых файлов*, которым посвящена следующая группа заданий (Text, п. 2.5). Отличие состоит в *формате хранения строк*: в заданиях на обработку двоичных строковых файлов предполагается, что все строки в этих файлах занимают участки памяти одинакового размера (что дает возможность прямого доступа к требуемой строке по ее номеру), в то время как в текстовых файлах каждая строка занимает участок

памяти, соответствующий ее фактической длине, а признаком конца каждой строки является специальный маркер (в операционной системе Windows маркер конца строки состоит из двух символов с кодами 13 и 10).

При выполнении заданий, связанных со строковыми файлами (63–73), с использованием электронного задачника на языке Pascal в средах Delphi, Lazarus и PascalABC.NET соответствующие файловые переменные должны быть описаны как file of ShortString; тем самым для хранения каждого элемента-строки в файле будет выделено 256 байт. В случае языка Visual Basic элементы строковых файлов должны иметь тип String * 80, в случае языка C++ — тип char[80] (при этом «реальный» размер строки при считывании данных из строковых файлов будет определяться правильно). При использовании языков платформы .NET (VB.NET и C#) требуется явным образом дополнять справа пробелами каждую строку, записываемую в строковый файл, чтобы в результате она имела длину 80 символов, а при считывании строк перед их обработкой может потребоваться удалить их завершающие пробелы. При организации прямого доступа к элементам строковых файлов для языков платформы .NET следует учитывать, что каждый строковый элемент занимает в файле 81 байт.

Информация об особенностях работы со строковыми файлами, ввиду ее важности, приведена в окне справки задачника Programming Taskbook на вкладке «Ввод-вывод» (окно справки можно вызвать из окна задачника, нажав клавишу [F1]).

Среди заданий на обработку двоичных строковых файлов следует отметить серию, связанную с обработкой дат (задания 67–73). Даты хранятся в формате «день/месяц/год», где день и месяц задаются двузначным числом, а год — четырехзначным. В некоторых из этих заданий (71–73) требуется сравнивать даты между собой; чтобы избежать большого числа проверок, удобно при подобных сравнениях использовать те же даты, преобразованные к «американскому» формату «год/месяц/день», для которых можно использовать обычное лексикографическое сравнение строк (в языках платформы .NET можно также

преобразовать исходные строки к стандартному типу DateTime, для которого уже реализованы операции сравнения).

Группа: File (символьные и строковые файлы)

Разбор в классе: 58, 60, 63, 67, 69

Закрепление материала: 59, 61, 68, 71

Простые задания: 58, 59

Сложные задания: 73

Наборы однотипных заданий: 58–59; 60–61; 64–65; 67–68; 69–72

2.4.5. Использование файлов для работы с матрицами

Последняя подгруппа группы File (задания 74–90) является дополнительной и содержит задания, в которых двоичные файлы используются для хранения элементов *матриц*, т. е. прямоугольных таблиц чисел. Выполнение этих заданий позволит учащимся изучить альтернативный способ хранения матриц, отличный от ранее рассмотренного способа (в виде двумерных массивов). Как и для других заданий группы File, при их выполнении не следует использовать вспомогательные массивы (поскольку в этом случае задачи сведутся к уже изученным задачам на обработку массивов и, кроме того, решения будут неэффективными по объему используемой памяти).

В заданиях данной подгруппы используются два формата хранения матриц. Первый применяется для квадратных матриц и не требует хранения дополнительных сведений о размере матрицы, так как ее порядок однозначно определяется по размеру файла (с квадратными матрицами связано большинство заданий: 74, 76, 80–90). Второй формат обеспечивает хранение прямоугольных матриц; для того чтобы можно было восстановить размер матрицы, в первый элемент соответствующего файла записывается количество столбцов матрицы (с прямоугольными матрицами связаны задания 75, 77–79). Кроме того, рассматриваются три специальных вида квадратных матриц: *верхнетреугольные* (80, 83, 86, 89), *нижнетреугольные* (81, 84, 87, 90) и *трехдиагональные* (82, 85, 88); с этими видами матриц связываются дополнительные форматы, не содер-

жащие нулевую часть матриц. В заданиях используются также две матричные операции: транспонирование матрицы (75, 78) и перемножение двух матриц (76, 79, 89, 90).

При выполнении заданий из данной подгруппы следует обратить внимание на реализацию двух основных подзадач: 1) определение размера матрицы по размеру файла и (для прямоугольных матриц) по дополнительной информации о числе столбцов, содержащейся в первом файловом элементе; 2) определение файловой позиции требуемого матричного элемента по номеру его строки и столбца (в заданиях предполагается, что строки и столбцы нумеруются от 1).

Поскольку как специальные виды матриц, так и матричные операции могут быть незнакомы учащимся, в преамбуле к данной подгруппе приводятся их определения. Для просмотра преамбулы достаточно отобразить на экране html-страницу для группы File (например, запустив модуль PT4Demo, выбрав в нем группу File и нажав клавишу [F2] или кнопку ).

Группа: File (использование файлов для работы с матрицами)

Разбор в классе: 74, 75

Закрепление материала: 77, 78

Простые задания: 74, 77

Сложные задания: 76, 79, 89*, 90*

Наборы однотипных заданий: 80–82; 83–85; 86–88; 89–90

2.5. Текстовые файлы: группа Text

Обработка текстовых файлов во всех рассматриваемых языках программирования имеет ряд особенностей, отличающих ее от обработки двоичных файлов. В частности, при обработке текстовых файлов нельзя использовать средства, связанные с прямым доступом к файловым элементам; кроме того, текстовые файлы нельзя одновременно открыть на чтение и запись, а для дополнения текстового файла его надо открыть в специальном режиме. По этой причине для изучения текстовых файлов разработана специальная группа Text.

В некоторых заданиях этой группы используются не только текстовые, но и двоичные файлы, поэтому приступать к изучению группы Text рекомендуется после изучения группы File.

При выполнении заданий на языках VB.NET и C# с использованием стандартных файловых классов платформы .NET следует учитывать, что по умолчанию при записи текстовых данных эти классы применяют кодировку UTF-8, которая отличается от однобайтной ANSI-кодировки, используемой задачником при формировании исходных файлов и проверке результирующих файлов. Для настройки ANSI-кодировки при работе с файлами надо явно указать в конструкторах соответствующих классов параметр System.Text.Encoding.Default. Поскольку при использовании электронного задачника это действие является необходимым для правильной обработки большинства файлов с текстовыми данными, информация о нем приводится в разделе «Ввод-вывод» окна справки задачника. Впрочем, в случае, если текстовое содержимое файла не включает русские буквы, параметр System.Text.Encoding.Default можно не указывать, так как ANSI-кодировка символов ASCII (с кодами до 127) совпадает с их кодировкой UTF-8.

2.5.1. Основные операции с текстовыми файлами

Первая подгруппа группы Text знакомит с основными приемами, связанными с созданием и преобразованием текстовых файлов. Задания 1–3 посвящены формированию текстовых файлов, задание 4 — анализу их содержимого, а прочие задания — преобразованию имеющихся файлов. Задания 1 и 4 подробно разбираются в [1, 6, 9]. Поскольку открыть текстовый файл одновременно на чтение и запись нельзя, в большинстве заданий на преобразование файла необходимо использовать вспомогательный файл (как и в аналогичных заданиях на преобразование двоичных файлов — см. п. 2.4.1). Однако в случае добавления в конец существующего файла новых строк (задания 5–6) можно обойтись без вспомогательного файла, используя специальный режим *дополнения* (append), на который следует обратить внимание учащихся.

С использованием вспомогательного файла можно легко выполнять такие действия по преобразованию текстового файла, как вставка в его начало или середину новых строк (задания 7–11) или обработка имеющихся строк: замена (12), удаление (13–16) или преобразование (17–18). Следует обратить внимание на задания 19–20, которые можно выполнить двумя способами, используя *построчную* или *посимвольную* обработку файла (заметим, что оба указанных способа обработки текстовых файлов приводятся в [1, 6, 9] при разборе заданий 1 и 4).

Завершает данную подгруппу серия заданий 21–23, в которых надо определенным образом обработать завершающую часть исходного файла: удалить требуемое число последних строк из файла (21–22) или скопировать их в новый файл (23). Рекомендуется обсудить эффективный, хотя и достаточно сложный, алгоритм выполнения этих заданий, использующий *однократный* проход по исходному файлу (различные варианты решения задачи 21 приведены в [1, 6, 9]).

Группа: Text (основные операции с текстовыми файлами)

Разбор в классе: 1, 4, 5, 7, 10, 14, 19

Закрепление материала: 2, 6, 11, 13, 15, 20

Простые задания: 1, 4, 5, 6

Сложные задания: 21, 22, 23

Наборы однотипных заданий: 9–10; 11–12; 15–16; 22–23

2.5.2. Анализ и форматирование текста

Данная подгруппа группы Text включает задания двух типов.

Задания первого типа связаны с анализом и обработкой фрагментов текста, отличных от строк и символов: это *абзацы* (задания 24–28) и отдельные *слова* (задания 29–33). Рассмотрены два способа выделения абзацев: с помощью одной или нескольких пустых строк (24–25) и с помощью абзацного отступа — красной строки (26–28). При обсуждении заданий 24–25 следует обратить внимание учащихся на необходимость анализа *пар* соседних файловых строк (те-

кущая непустая строка является началом абзаца, если предшествующая ей строка является пустой). Задания на обработку слов также представлены в двух вариантах: в более простом варианте словом считается последовательность непробельных символов, которая ограничена пробелами или началом/концом строки (29–30), в более сложном варианте слова могут отделяться друг от друга не только пробелами, но и знаками препинания (31–33). Во всех заданиях на обработку слов удобно использовать посимвольный анализ исходного текста.

Задания второго типа связаны с форматированием исходного текста: изменением способа его горизонтального выравнивания (34–37) и изменением ширины текста путем переноса «выступающих» слов на новую строку (38–39). Задачи на изменение ширины текста являются сложными, в то время как в серию задач на выравнивание входят как сравнительно простые задачи 34–36 (на выравнивание текста по левому или правому краю и на его центрирование), так и сложная задача 37, связанная с выравниванием *по ширине* (за счет добавления между словами дополнительных пробелов).

Группа: Text (анализ и форматирование текста)

Разбор в классе: 24, 25, 34

Закрепление материала: 26, 27, 35, 36

Простые задания: 26, 29, 30

Сложные задания: 37, 38*, 39*

Наборы однотипных заданий: 29–30; 31–33; 34–36

2.5.3. Текстовые файлы с числовой информацией

При работе с текстовыми файлами, содержащими числовую информацию, возникают дополнительные проблемы, связанные, во-первых, с необходимостью *форматирования* чисел при их записи в текстовый файл и, во-вторых, с необходимостью выделения из текстовой строки числовых данных при их чтении из файла. Форматированию числовых данных при их записи в файл посвящены задания 40–43 и 49, разбору исходных файлов с целью выделения из них числовых данных — задания 44–48 и 50–52.

Во многих заданиях данной подгруппы наряду с текстовыми файлами используются ранее изученные *двоичные* файлы с числовой информацией.

Задания на форматирование выполняются с применением стандартных средств форматирования, имеющихся в любом из используемых языков программирования. В частности, для языка Pascal можно применять как «старый» способ, основанный на указании форматных настроек через двоеточие, так и новые способы, появившиеся в средах Delphi и Lazarus и связанные с функцией Format (функция Format имеется и в среде PascalABC.NET, хотя в ней применяются форматные настройки другого вида, используемые в языках платформы .NET). В заданиях 40–41 требуется выполнить форматирование целочисленных данных, в заданиях 42–43 — данных вещественного типа (задания 42–43 связаны с *табулированием функции*, т. е. с формированием таблицы значений функции на некотором наборе равноотстоящих узлов).

Способ выполнения заданий, связанных с выделением из строки числовых данных, зависит от наличия в языке программирования или его стандартной библиотеке соответствующих средств. Если язык позволяет считывать числовые данные из текстовых файлов непосредственно в переменные числовых типов (например, с помощью процедур Read и Readln языка Pascal, оператора Input языка Visual Basic или оператора >> языка C++), то этой возможностью, разумеется, надо воспользоваться. Если же специальных средств для чтения числовых данных из текстовых файлов не предусмотрено (как в стандартной библиотеке ввода-вывода .NET), то необходимо считывать данные в строковые переменные и затем выполнять их преобразование к требуемому числовому типу.

При изучении особенностей чтения числовых данных из текстовых файлов на языке Pascal необходимо обратить внимание учащихся на наличие функции SeekEof, позволяющей вовремя распознать конец файла при считывании из него числовых данных, даже если после чтения последнего числа в файле остаются незначащие символы, например, пробелы.

Группа: Text (текстовые файлы с числовой информацией)

Разбор в классе: 40, 42, 44, 47, 49

Закрепление материала: 41, 43, 46, 51

Простые задания: 40

Сложные задания: 52

Наборы однотипных заданий: 42–43; 44–48

2.5.4. Дополнительные задания на обработку текстовых файлов

Задания этой подгруппы связаны с выделением из исходных текстовых файлов требуемых категорий символов и анализом частоты их появления (задания 53–58), а также с шифрованием/дешифрованием исходного файла (59–60).

При анализе символов, содержащихся в исходном файле, важно использовать оптимальную структуру данных для хранении информации об уже прочитанных и обработанных символах. Если при выполнении задания надо лишь хранить информацию о *наличии* в исходном файле того или иного символа (задания 54–56), то на языке Pascal в качестве соответствующей структуры можно использовать *множество*, содержащее символы, прочитанные из файла. Если же необходимо подсчитывать *количество* появлений каждого символа (задания 57–58), то можно использовать *числовой массив*, каждый элемент которого соответствует одному из возможных символов, а значение этого элемента равно числу появлений данного символа (на языке Pascal в качестве индексов массива можно использовать *символы*; таким образом, информация о числе появлений, например, символа «a», будет содержаться в элементе с индексом «a», информация о символе «b» будет содержаться в элементе с индексом «b» и т. д.). Аналогичные структуры можно использовать и для других языков программирования; в частности, если язык не поддерживает работу с множеством, то вместо множества можно использовать массив логических элементов, каждый из которых соответствует одному из возможных символов (элемент равен значе-

нию True, если соответствующий символ был прочитан из исходного текста, и False в противном случае).

Для шифрования/дешифрования файла в заданиях 59–60 используется усложненный вариант способа шифрования, основанного на циклическом сдвиге букв алфавита (ранее этот способ был рассмотрен в задачах 62–65 группы String — см. п. 2.3.5).

Группа: Text (дополнительные задания на обработку текстовых файлов)

Разбор в классе: 54, 55, 57

Закрепление материала: 56, 58

Простые задания: 53

Сложные задания: 58, 59, 60

Наборы однотипных заданий: 54–56; 59–60

2.6. Составные типы данных в процедурах и функциях: группа Param

Группа Param содержит задания на обработку одномерных и двумерных массивов, строк и файлов, т. е. тех же структур данных, которые ранее изучались в группах Array, Matrix, String, File и Text. Особенностью заданий группы Param является то, что алгоритмы решения необходимо оформить в виде некоторой процедуры или функции с указанными в условии параметрами. Таким образом, выполняя эти задания, учащиеся, с одной стороны, должны повторить и закрепить материал, связанный с обработкой сложных структур данных, а с другой стороны — вспомнить правила описания и использования процедур и функций (и, кроме того, изучить особенности передачи сложных структур данных в качестве параметров).

Задания группы Param разделены на четыре подгруппы, первые три из которых связаны с одной из ранее изученных структур данных: массивами, строками и файлами. Задания из каждой подгруппы можно предложить для выполнения сразу после изучения соответствующей структуры данных. Можно также

организовать «итоговое повторение», обратившись к группе Param после изучения всех ранее рассмотренных групп, связанных со сложными структурами данных.

Особое место занимают задания четвертой подгруппы, предназначенные для изучения новых структурных типов, называемых в языке Pascal *записями* (в языке Visual Basic подобные типы называются *пользовательскими типами данных*, а в языках C++, C#, VB.NET — *структурами*).

2.6.1. Одномерные и двумерные массивы

Данную подгруппу можно разделить на две части. В первую часть входят задания, связанные с обработкой *одномерных массивов*: поиск минимальных/максимальных элементов в массиве (задания 1–3), инвертирование и сглаживание массива (4–7), удаление и вставка элементов (8–10), сортировка (11–13), разбиение исходного массива на два результирующих (14–15). Вторая часть содержит задания на обработку *двумерных массивов-матриц*: формирование матрицы (16–18), вычисление матричной нормы (19–20), вычисление сумм элементов строк или столбцов (21–22), преобразование матрицы, связанное с переменой местами требуемых строк или столбцов (23–24), транспонированием (25). удалением строк и/или столбцов (26–28) и сортировкой столбцов (29).

Многие задания имеют близкие аналоги в группах Minmax, Array и Matrix. Из числа «оригинальных» заданий отметим задания, посвященные сглаживанию массива (5–7), формированию матрицы по исходному одномерному массиву (16–17) и вычислению нормы матрицы (19–20).

При выполнении заданий желательно не использовать вспомогательные массивы, хотя в ряде случаев это ограничение приводит к усложнению алгоритма (см., например, задания 7–10).

Общей чертой многих заданий на обработку матриц является необходимость проверки корректности некоторых параметров. Например, в процедуре нахождения суммы *K*-й строки матрицы из задания 21 необходимо дополни-

тельно проверять, что параметр *K* лежит в допустимом диапазоне номеров строк (при нарушении этого условия требуется вывести особое значение 0).

Группа: Param (одномерные и двумерные массивы)

Разбор в классе: 1, 4, 5, 8, 21, 24

Закрепление материала: 2, 3, 10, 22, 23

Простые задания: 1, 2, 3, 18

Сложные задания: 7, 9, 12, 13, 29

Наборы однотипных заданий: 5–6; 14–15; 16–17; 19–20; 21–22; 23–24; 26–27

2.6.2. Строки

Некоторые из заданий данной подгруппы являются не «процедурными» вариантами ранее рассмотренных заданий группы String, а вполне оригинальными заданиями, среди которых можно выделить задания на проверку корректности идентификатора (задание 30), формирование строки по повторяющемуся шаблону (31), сжатие строки текста с повторяющимися символами по определенной схеме и ее последующее восстановление (42–43).

В ряде заданий требуется разработать процедуры и функции, которые выполняют те же действия, что и подпрограммы, входящие в некоторые стандартные библиотеки для работы со строками. К таким заданиям можно отнести задания на изменение регистра символов (32–33), удаление начальных или конечных символов с указанным значением (34-35), инвертирование и поиск (36–39), выделение из строки слов (40–41). Однако даже если при выполнении подобных заданий учащийся воспользуется стандартными подпрограммами, от него все равно потребуется адаптировать их к условию задания; описав процедуру или функцию с требуемым именем и указанными в задании параметрами.

Последние четыре задания (44–47) посвящены алгоритмам перевода из десятичной системы счисления в двоичную и шестнадцатеричную (и обратно); ранее подобные задания рассматривались в группе String (см. п. 2.3.2, задания 24–25).

Группа: Param (строки)

Разбор в классе: 30, 34, 41

Закрепление материала: 31, 33, 40

Простые задания: 32, 33, 34, 35

Сложные задания: 42, 43, 45, 47

Наборы однотипных заданий: 32–33; 34–35

2.6.3. Файлы

В отличие от заданий подгрупп, посвященных массивам и строкам, в заданиях на обработку файлов не требуется передавать данные соответствующего структурного типа (в данном случае — файлового типа) в качестве параметров. Вместо этого в процедуры или функции должны передаваться *имена* обрабатываемых или создаваемых файлов, что соответствует принятой практике разработки подпрограмм для работы с файлами и существенно упрощает использование подобных подпрограмм.

Данная подгруппа включает ряд заданий, являющихся процедурными «оболочками» ранее рассмотренных заданий на обработку двоичных и текстовых файлов: это задания на определение числа элементов в двоичном файле (48) и числа строк в текстовом файле (49), а также на инвертирование элементов двоичного файла (50). Во всех перечисленных заданиях дополнительно требуется выполнять проверку существования файла с указанным именем, и обрабатывать особую ситуацию, связанную с отсутствием данного файла.

Задачи на шифрование/дешифрование текстового файла (57–58) также являются процедурными оболочками заданий 59–60 из группы Text (см. п. 2.5.4), причем в этих задачах предлагается использовать более простой способ шифрования.

К оригинальным заданиям можно отнести задачи на добавление или удаление нумерации строк текстового файла (51–52), на разбиение двоичного или текстового файла на две части (53–54) и на преобразование двоичного строкового файла в текстовый и обратно (55–56). Все эти задания (за исключением за-

дания 52), являются достаточно простыми, однако требуют использования большинства ранее изученных приемов работы с файлами (задание 51 дополнительно требует применения форматного вывода числовых данных). Сложность задания 52, в котором требуется удалить ранее добавленную в текстовый файл нумерацию строк, связана с необходимостью предварительной проверки того, существует ли в исходном файле нумерация строк, и определения формата нумерации, если она существует.

Группа: Param (файлы)

Разбор в классе: 48, 54, 55

Закрепление материала: 49, 50, 53, 56

Простые задания: 48, 49, 55, 56

Сложные задания: 51, 52, 57, 58

Наборы однотипных заданий: 48–49; 51–52; 53–54; 55–56; 57–58

2.6.4. Записи

Задания в данной подгруппе можно разбить на две серии, в каждой из которых используется свои типы-записи: в серии заданий 59–63 это тип TDate, содержащий информацию о дате, в серии 64–70 — типы TPoint (точка на координатной плоскости) и TTriangle (треугольник, заданный своими вершинами). В каждой серии задания должны выполняться последовательно, поскольку в большинстве заданий каждой серии используются типы и подпрограммы, реализованные в предыдущих заданиях этой серии. Следует заметить, что аналогичные задания, в которых, однако, не требовалось использовать записи, входят в завершающую подгруппу группы Proc (см. п. 1.9.3): это задания 52–55 на обработку дат и задания 56–60, связанные с геометрией на плоскости. Полезно сравнить соответствующие задания из групп Proc и Param и обсудить преимущества, которые дает использование новых типов-записей.

Типы и подпрограммы, разработанные в заданиях данной подгруппы, целесообразно сохранять в отдельном *модуле*, что упростит их использование при выполнении последующих заданий из той же серии. В языках C++, C#, VB.NET

можно оформлять подпрограммы, связанные с записями (называемыми в этих языках структурами), в виде *методов*, входящих в описание этих структур.

Группа: Param (записи)

Разбор в классе: 59, 60, 61, 62

Закрепление материала: 63, 64, 65, 66

Простые задания: *нет*

Сложные задания: *нет*

Наборы однотипных заданий: 62–63; 69–70

3. Рекурсия и динамические структуры данных

3.1. Рекурсия: группа Recur

Помимо группы Recur рекурсивные алгоритмы рассматриваются в группе Tree, связанной с обработкой деревьев (см. п. 3.3).

3.1.1. Простейшие рекурсивные алгоритмы

Изучение темы «Рекурсия» традиционно начинается с рассмотрения простейших задач (подобных вычислению факториала или двойного факториала — см. задания 1–2), которые, наряду с рекурсивным решением, допускают и «обычное» итерационное решение. Эти задачи позволяют получить первоначальные навыки разработки рекурсивных алгоритмов; однако учащиеся должны четко понимать, что для подобных задач использование рекурсии нередко является не самым лучшим (с точки зрения эффективности) вариантом решения. Для наглядной демонстрации этого обстоятельства полезно рассмотреть хотя бы одно из заданий 4, 6, в которых требуется реализовать рекурсивный алгоритм для вычисления чисел Фибоначчи (задание 4) или числа перестановок C_n^k (задание 6), дополнительно определив *количество вызовов рекурсивной функции*, и убедиться в том, что это количество будет очень быстро расти при увеличении значений параметров рекурсивных функций. В заданиях 5, 7 приводится вариант решения проблемы, связанной с быстрым ростом числа рекур-

сивных вызовов; он состоит в использовании вспомогательного массива, содержащего *уже вычисленные* значения рекурсивной функции.

Большинство заданий данной подгруппы связано с обработкой чисел; исключение составляют последние три задания, в которых в качестве параметра рекурсивной функции требуется использовать массив (задание 11 на нахождение минимального элемента массива) или строку (задание 12 на подсчет количества цифр в строке и задание 13 на проверку того, является ли строка палиндромом).

Группа: Recur (простейшие рекурсивные алгоритмы)

Разбор в классе: 1, 4, 5, 11, 13

Закрепление материала: 2, 6, 7, 10, 12

Простые задания: 1, 2

Сложные задания: *нет*

Наборы однотипных заданий: 1–2

3.1.2. Разбор выражений

Разбор выражений представляет собой один из типов задач, для эффективного решения которых необходимо использовать рекурсию. Следует отметить, что задачи этого типа являются достаточно сложными (требующими, как правило, реализации набора связанных подпрограмм, в которых, к тому же, может возникать *косвенная рекурсия*). Ввиду сложности подобных задач в пособиях [1, 6, 9] приводятся подробные решения нескольких задач из этой подгруппы (14–16, 18), связанных с разбором арифметических выражений (в частности, рассматриваются выражения, в которых необходимо учитывать приоритет операций, и выражения, содержащие скобки).

В большинстве заданий в качестве исходных данных предлагаются заведомо правильные выражения, однако имеются задания (18, 19), в которых необходимо проверить правильность исходного выражения и вывести результат этой проверки либо в виде логического значения (18), либо в виде числа, указывающего позицию первого ошибочного символа (19).

Наряду с заданиями на разбор арифметических выражений (14–19) подгруппа включает задания на разбор выражений, содержащих вызовы функций нахождения минимальных/максимальных значений (20, 22) или вызовы логических функций (21, 23–24). Дополнительная особенность заданий 22–24 состоит в том, что вызовы функций, входящие в анализируемые выражения, имеют переменное число параметров.

Группа: Recur (разбор выражений)

Разбор в классе: 17, 18, 21

Закрепление материала: 19, 20, 23

Простые задания: 14, 17

Сложные задания: 16, 22, 23, 24

Наборы однотипных заданий: 20–21; 22–23

3.1.3. Перебор с возвратом

Данная подгруппа содержит шесть заданий (25–30), в которых необходимо реализовать рекурсивный перебор «разветвляющихся» данных, представленных в виде деревьев. Соответствующий алгоритм называется алгоритмом *перебора с возвратом*. В отличие от заданий, входящих в группу Tree (см. п. 3.3), в данной подгруппе рассматриваются деревья специального вида — *полные деревья степени K*, в которых все внутренние вершины имеют *K* непосредственных потомков, а все листья находятся на одном и том же уровне *N*. Это позволяет рассмотреть сущность алгоритма перебора с возвратом, не «отвлекаясь» на вопросы, связанные с организацией хранения в памяти деревьев произвольной структуры.

Алгоритм решения первой из задач данной подгруппы (на перебор *всех* маршрутов, ведущих от корня дерева к его листьям) подробно описывается в [1, 6, 9]; для решения остальных задач, в которых требуется найти маршруты, удовлетворяющие дополнительным условиям, достаточно использовать модификации этого алгоритма.

В заданиях 27–30 с вершинами дерева связываются определенные *весовые значения* (равные –1, 0 и 1), которые следует учитывать при отборе допустимых маршрутов. В частности, в заданиях 28–29 требуется отобрать маршруты, для любого начального участка которых суммарный вес вершин неотрицателен (28) или неположителен (29); в задании 29 дополнительно требуется, чтобы суммарный вес всех вершин маршрута был равен 0. На примере этих заданий можно обсудить прием *отсечения*, позволяющий отбрасывать некоторые недопустимые маршруты, не доводя их построение до конца, что существенно ускоряет перебор вариантов.

В заданиях данной подгруппы, в отличие от заданий других подгрупп группы Recur, требуется умение работать с текстовыми файлами (в текстовый файл записываются допустимые маршруты для обрабатываемого дерева).

Группа: Recur (перебор с возвратом)

Разбор в классе: 25, 27

Закрепление материала: 26, 28

Простые задания: 25

Сложные задания: 29, 30

Наборы однотипных заданий: 27–28; 29–30

3.2. Динамические структуры данных: группа Dynamic

В заданиях группы Dynamic и следующей за ней группы Tree (см. п. 3.3) используются *цепочки узлов-записей*, связанных между собой ссылочными полями. С помощью подобных цепочек можно реализовать различные динамические структуры: стек, очередь (односвязные цепочки, использующие единственное поле связи Next), список (двусвязные цепочки, использующие два поля связи: Next и Prev), бинарное дерево (иерархические цепочки, позволяющие связывать каждый узел-вершину с двумя потомками, используя поля связи Left и Right). Таким образом, задания этих групп позволяют познакомиться с особенностями использования ссылочных типов и изучить основные динамические структуры данных.

В языках программирования, поддерживаемых электронным задачником Programming Taskbook, работа со ссылочными типами организуется одним из двух различных способов. Языки Pascal и C++ используют для этого *указатели*, в то время как для языков платформы .NET (C# и VB.NET) стандартным является другой подход, основанный на использовании *объектов* классового (т. е. ссылочного) типа. По этой причине группы Dynamic и Tree, в отличие от всех остальных базовых групп задачника, реализованы в двух вариантах.

В первом варианте задания формулируются в терминах указателей, а для создания динамических структур используется специальный тип-запись TNode, содержащий поле данных Data целого типа и одно или несколько полей связи типа PNode, являющихся указателями на записи типа TNode. Этот вариант реализован для языков Pascal и C++.

Во втором варианте задания формулируются в терминах объектов ссылочного типа, а для создания динамических структур используется специальный ссылочный тип — класс Node с набором свойств, включающих свойство Data целого типа и одно или несколько свойств связи типа Node. Этот вариант реализован для языков C# и VB.NET.

В языке Visual Basic отсутствуют средства создания динамических структур данных, основанных на использовании ссылочных типов. Поэтому в базовый набор заданий для языка Visual Basic группы Dynamic и Tree не включены. В наборе для языка PascalABC.NET, напротив, эти группы присутствуют в двух вариантах, поскольку язык PascalABC.NET сочетает в себе возможности как «стандартного» языка Pascal (и тем самым позволяет использовать указатели), так и языка платформы .NET. Варианты групп Dynamic и Tree, связанные со ссылочными объектами и предназначенные для использования в системе PascalABC.NET, имеют специальные имена: ObjDyn и ObjTree. Таким образом, при изучении линейных динамических структур и бинарных деревьев в системе PascalABC.NET преподаватель имеет возможность выбирать между двумя подходами: с использованием указателей или объектов ссылочного типа.

Следует подчеркнуть, что несмотря на использование различных типов данных (записей с указателями или ссылочных объектов), алгоритмы обработки получаемых динамических структур в обоих вариантах любого задания практически не отличаются.

3.2.1. Стек

Первые два задания данной подгруппы (1–2) имеют вводный характер и знакомят с понятиями узла и цепочки узлов. В них не требуется создавать или модифицировать узлы и их связи — достаточно лишь проанализировать состав предложенной односвязной цепочки узлов.

В остальных заданиях подгруппы цепочка узлов интерпретируется как динамическая структура «стек». В них требуется либо добавить к стеку один или несколько новых элементов-узлов (3–4), либо извлечь из стека один или несколько имеющихся элементов (5–7), либо перераспределить элементы между несколькими исходными стеками (8–10).

При выполнении заданий на добавление элементов в стек необходимо использовать операцию *выделения динамической памяти* (для языков Pascal и C++) или вызывать *конструктор объекта* (для языков C# и VB.NET). При выполнении заданий на извлечение элементов из стека для языков Pascal и C++ требуется *освобождать динамическую память*, вызывая соответствующую процедуру (при использовании языков платформы .NET также требуется выполнять специальные действия по освобождению выделенных ресурсов, вызывая специальный метод Dispose класса Node). При перераспределении имеющихся элементов не требуется ни выделять, ни освобождать память.

Три завершающих задания данной подгруппы посвящены разработке вспомогательных подпрограмм, упрощающих работу со стеком: Push (добавление элемента в стек — задание 11), Pop (извлечение элемента из стека — задание 12), Peek и IsEmpty (просмотр верхнего элемента стека и проверка стека на непустоту — задание 13). Способ оформления этих подпрограмм зависит от используемого языка программирования: для языков Pascal и C++ подпрограм-

мы должны работать со специальным типом — записью с единственным полем Top (указатель на вершину стека), для языков платформы .NET требуется реализовать новый класс с закрытым полем top, включив в этот класс все указанные выше подпрограммы в виде открытых методов.

Группа: Dynamic (стек)

Разбор в классе: 2, 3, 4, 6, 11

Закрепление материала: 5, 7, 8, 12

Простые задания: 1, 3, 5

Сложные задания: *нет*

Наборы однотипных заданий: 6–7

3.2.2. Очередь

Состав заданий, посвященных обработке динамической структуры «очередь», подобен составу заданий, связанных с обработкой стека: вначале идут задания, связанные с формированием очереди и добавлением в нее новых элементов (14–17), затем — с извлечением элементов из очереди (18–19) и наконец — с обработкой существующих очередей, связанной с перераспределением их элементов (20–25).

Общей чертой заданий на стеки и очереди является то, что в обоих случаях структуры моделируются односвязной цепочкой узлов. Однако задания, связанные с очередями, являются более сложными, чем аналогичные задания для стеков. Это объясняется, прежде всего, необходимостью специальной обработки *особых ситуаций*: извлечения из очереди последнего элемента, в результате чего очередь становится пустой, и добавления к пустой очереди первого элемента. В подобных ситуациях для стека особой обработки не требовалось, так как ссылка на вершину стека изменялась автоматически. Для очереди требуются дополнительные действия, обусловленные тем, что с очередью связываются две ссылки: на начало очереди (из которого извлекаются элементы) и на ее конец (к которому добавляются новые элементы).

Из заданий на преобразование очередей следует выделить задание 25, в котором требуется выполнить слияние двух упорядоченных очередей в новую очередь с сохранением упорядоченности. Полезно сравнить разработанный алгоритм с аналогичными алгоритмами для слияния массивов (задания 63–64 группы Array — см. п. 2.1.3) и файлов (задания 50–51 группы File — см. п. 2.4.2).

Завершают данную подгруппу (как и подгруппу, посвященную стекам) три задания на разработку подпрограмм, предназначенных для обработки очередей: Enqueue (добавление элемента в очередь — задание 26), Dequeue (извлечение элемента из очереди — задание 27), IsEmpty (проверка очереди на непустоту — задание 28). Как и в случае со стеком, способ оформления данных подпрограмм зависит от используемого языка программирования. При разработке подпрограмм Enqueue и Dequeue следует обратить внимание на необходимость дополнительной обработки отмеченных выше особых ситуаций.

Группа: Dynamic (очередь)

Разбор в классе: 16, 19, 22, 25, 26

Закрепление материала: 15, 20, 23, 24, 27

Простые задания: 17, 18

Сложные задания: 19, 20, 22, 23, 24, 25

Наборы однотипных заданий: 19–20; 22–23; 24–25

3.2.3. Двусвязный список

Основное отличие заданий на двусвязные списки от ранее рассмотренных заданий на стеки и очереди состоит в том, что в цепочках элементов-узлов, моделирующих списки, каждый элемент связывается с обоими своими соседями: со следующим элементом посредством поля Next, а с предыдущим — посредством поля Prev (в случае языков платформы .NET вместо полей используются одноименные свойства класса Node). Первый элемент списка содержит пустую ссылку Prev, а последний — пустую ссылку Next. Наличие двойной связи и необходимость особым образом обрабатывать «концевые» элементы списков

приводит к усложнению алгоритмов обработки списков (по сравнению с аналогичными алгоритмами обработки стеков и очередей). Кроме того, для списков вводится понятие *текущего элемента*, для которого определяются дополнительные действия, отсутствующие у стеков и очередей: добавление нового элемента перед или после текущего и удаление текущего элемента (все эти действия должны сохранять целостность двусвязной цепочки).

Подобно подгруппе, связанной со стеком, подгруппа для списка начинается с двух вводных заданий (29–30), посвященных знакомству с двусвязными цепочками. Следует обратить внимание на задание 30, в котором требуется преобразовать исходную односвязную цепочку в двусвязную. В других заданиях подобные действия выполнять не придется, поскольку все исходные списки уже являются двусвязными.

Последующие задания можно разбить на несколько серий:

- простейшие задания на перемещение по списку в различных направлениях и вставку в его начало/конец или в требуемую позицию нового элемента (31–36);
- более сложные задания на вставку элементов, в которых требуется продублировать в списке элементы с определенными свойствами, например, имеющие четный/нечетный порядковый номер или четное/нечетное значение (37–40);
- задания на удаление одного (41) или нескольких (42-43) требуемых элементов из списка;
- задания, связанные с перемещением некоторого элемента списка вперед или назад на указанное число позиций (44–47);
- более сложные задания на перегруппировку элементов списка, включающие перемену местами двух указанных элементов (48) и перемещение элементов с требуемыми свойствами в конец списка (49–50);
- задания, связанные с обработкой нескольких списков: вставка всех элементов одного списка в указанную позицию другого (51-52) и обратная

задача: извлечение отмеченного фрагмента из списка и оформление извлеченного фрагмента как нового списка (53–54).

Задания 55–58 посвящены особой разновидности списков — *циклическим спискам*, которые моделируются замкнутой цепочкой узлов (замкнутые цепочки рассматриваются также в следующей подгруппе, посвященной другому варианту реализации двусвязного списка, — см. п. 3.2.4). Следует отметить задания 57–58, в которых с помощью «сшивания» исходного списка в циклический и последующего «разрыва» этого списка в новой позиции требуется реализовать циклический сдвиг элементов списка на указанное количество позиций вперед или назад.

Как и в предыдущих подгруппах, заключительные задания (59–69) посвящены реализации подпрограмм, выполняющих различные стандартные операции над списком. Количество подобных заданий достаточно велико, поскольку для списков можно определить много различных операций. При реализации всех операций предполагается, что со списком связываются три ссылки: на начало списка (First), на его конец (Last) и на его текущий элемент (Current); для пустого списка все эти ссылки должны быть пустыми. Перечислим операции, которые требуется реализовать:

- различные операции, связанные со вставкой элемента: InsertLast/InsertFirst (добавление в конец/начало списка — задания 59–60), InsertBefore/InsertAfter (вставка перед/после текущего элемента списка — задания 61–62);
- операции, обеспечивающие перемещение по списку вперед/назад, а также доступ к значениям элементов списка на чтение/запись (ToFirst/ToLast, ToNext/ToPrev, IsLast/IsFirst, GetData/SetData — задания 63–64);
- операция, связанная с удалением текущего элемента списка (DeleteCurrent — задание 65); сложность этой операции обусловлена необходимостью обработки большого числа различных особых ситуаций;

- операции, связанные с обработкой нескольких списков: Split (расщепление списка на два — задание 66), Add (добавление одного списка в конец другого — задание 67), Insert (вставка одного списка перед текущим элементом другого — задание 68), MoveCurrent (перемещение текущего элемента одного списка в другой список — задание 69); последняя операция, подобно операции DeleteCurrent, является сложной для реализации.

Группа: Dynamic (двусвязный список)
Разбор в классе: 30, 31, 34, 41, 55, 57, 59, 63
Закрепление материала: 32, 33, 40, 56, 58, 60, 64
Простые задания: 29, 31, 33, 34, 35, 36, 41, 55
Сложные задания: 46, 47, 48, 49, 50, 65, 69
Наборы однотипных заданий: 33–34; 35–36; 37–40; 42–43; 44–45; 46–47; 49–50; 51–52; 53–54; 57–58; 59–60; 61–62; 63–64; 67–68

3.2.4. Список с барьерным элементом

Последняя подгруппа группы Dynamic может рассматриваться как дополнительная, поскольку в ней не вводится новая динамическая структура, а по-прежнему изучается двусвязный список. Эта подгруппа посвящена варианту реализации двусвязного списка, в котором для хранения элементов используется *замкнутая* цепочка узлов, снабженная особым узлом — *барьерным элементом* списка. Элемент, следующий за барьерным, считается первым элементом списка, элемент, предшествующий барьерному, — последним элементом. Пустой список моделируется единственным барьерным элементом, «замкнутым на себя».

При реализации списка с использованием барьерного элемента нет необходимости хранить ссылки на его начало и конец; достаточно хранить *одну* ссылку на барьерный элемент, используя которую можно сразу перейти и к первому, и к последнему элементу списка. Еще одним преимуществом этой реализации является отсутствие пустых ссылок: даже при извлечении из списка

всех «настоящих» элементов в нем останется барьерный элемент. Кроме того, при любых преобразованиях списка ссылка не его барьерный элемент не изменяется (в отличие от прежней реализации, в которой требуется хранить ссылки на первый и последний элементы списка, которые могут изменяться).

В первом задании данной подгруппы (70) требуется преобразовать «обычную» двусвязную цепочку с пустыми ссылками на концах в замкнутую цепочку с барьерным элементом, т .е. перейти от старого к новому представлению двусвязного списка. В последующих заданиях исходные списки уже представлены в виде замкнутых цепочек с барьерными элементами. В задании 71 требуется разбить список на два, в заданиях 72–73 — объединить два списка в один с сохранением первоначальной структуры.

Заключительные задания (74–80) посвящены, как обычно, реализации подпрограмм, выполняющих стандартные операции над списком:

- вставку элемента: InsertLast/InsertFirst (добавление в конец/начало списка — задания 74–75), InsertBefore/InsertAfter (вставка перед/после текущего элемента списка — задания 76–77);
- перемещение по списку вперед/назад и доступ к значениям элементов списка на чтение/запись (ToFirst/ToLast, ToNext/ToPrev, IsBarrier, GetData/SetData — задания 78–79);
- удаление текущего элемента списка (DeleteCurrent — задание 80).

Полезно сравнить реализации этих подпрограмм с реализациями аналогичных подпрограмм для «старого» представления списка. Новые варианты будут существенно проще, поскольку в них не потребуется предусматривать обработку особых ситуаций, связанных с пустыми ссылками (особенно сильно упростится реализация подпрограммы DeleteCurrent). Следует также обратить внимание на функцию IsBarrier (задание 78), используемую вместо пары функций IsFirst/IsLast. Применение функции IsBarrier упрощает перебор элементов списка, так как не требует особой обработки первого или последнего элемента (которую надо выполнять при использовании функций IsFirst/IsLast). Полезно

сравнить функцию IsBarrier с аналогичной функцией, позволяющей проверить, что достигнут конец файла (например, Eof для языка Pascal).

Группа: Dynamic (список с барьерным элементом)

Разбор в классе: 71, 74, 78, 80

Закрепление материала: 72, 75, 76, 79

Простые задания: 70

Сложные задания: *нет*

Наборы однотипных заданий: 72–73; 74–75; 76–77; 78–79

3.3. Бинарные деревья: группа Tree

В заданиях группы Tree, как и в заданиях группы Dynamic, используются цепочки элементов-записей, связанных между собой ссылочными полями, поэтому все, сказанное об особенностях группы Dynamic (см. п. 3.2), относится и к группе Tree. В частности, реализованы два варианта заданий этой группы: в первом варианте задания формулируются в терминах *записей* и *указателей* на них (этот вариант предназначен для языков Pascal и C++), во втором — в терминах *объектов* ссылочного типа (этот вариант предназначен для языков платформы .NET). Для языка Visual Basic группа Tree, как и группа Dynamic, не реализована. В языке PascalABC.NET можно использовать оба варианта группы Tree; при этом название Tree оставлено для варианта, связанного с указателями, а вариант, связанный со ссылочными объектами, получил название ObjTree.

Основное отличие цепочек узлов, рассматриваемых в группе Tree, от цепочек группы Dynamic состоит в том, что в группе Tree цепочки являются *иерархическими*: с каждым узлом (*вершиной дерева*) могут быть связаны два непосредственных потомка (левая и правая *дочерняя вершина*); связь осуществляется с помощью ссылочных полей узла-предка Left и Right; возможна также «обратная связь» от потомка к предку (*родительской вершине*), которая обеспечивается полем Parent потомка (такой вариант деревьев называется в задачнике *деревьями с обратной связью* — см. п. 3.3.4). Общий предок всех вершин непустого дерева называется *корнем*. С помощью подобных иерархических цепо-

чек можно моделировать не только бинарные деревья, но и *деревья с произвольным ветвлением*, вершины которых могут иметь произвольное число дочерних вершин (такой вариант деревьев называется в задачнике *деревьями общего вида* — см. п. 3.3.7).

Вершины дерева в задачнике моделируются тем же типом данных, что и элементы линейных динамических структур (для языков Pascal и C++ это запись TNode и указатель на нее PNode, для языков платформы .NET — класс Node). Различие состоит в полях, используемых для связи вершин: если для линейных структур это поля Next и, возможно, Prev (для двусвязных списков), то для деревьев это поля Left, Right и, возможно, Parent (для деревьев с обратной связью).

Как и в случае подгрупп группы Dynamic, связанных с различными видами линейных динамических структур (стек, очередь, список), большинство подгрупп группы Tree связано с различными видами деревьев. Основному объекту данной группы — бинарному дереву — посвящены три первые подгруппы (см. п. 3.3.1–3.3.3), затем рассматриваются бинарные деревья с обратной связью (п. 3.3.4), бинарные деревья поиска (п. 3.3.5), бинарные деревья разбора выражений (п. 3.3.6) и, наконец, деревья общего вида (п. 3.3.7).

При изучении темы «Деревья» следует начинать с рассмотрения заданий первых трех групп, содержащих основные алгоритмы, связанные с анализом, формированием и преобразованием бинарных деревьев. Прочие подгруппы можно рассматривать в любом порядке; кроме того, некоторые из этих подгрупп можно исключить из рассмотрения.

В заключение обзора группы Tree отметим, что она связана не только с группой Dynamic, но и с группой Recur, поскольку большинство алгоритмов анализа и обработки деревьев являются рекурсивными.

3.3.1. Анализ бинарного дерева

В первой подгруппе группы Tree не требуется создавать или преобразовывать деревья; достаточно проанализировать содержимое исходного дерева.

Первые два задания (1–2) знакомят учащегося со структурой бинарного дерева и связанными с ним понятиями. В задании 2 требуется определить общее количество вершин дерева; рекурсивный алгоритм, решающий эту задачу (см. [8, 9]), с некоторыми модификациями будет использоваться и для последующих заданий 3–5, связанных с перебором всех вершин (нахождение количества вершин, удовлетворяющих определенному условию, и нахождение суммы значений всех вершин дерева).

В задании 6 вводится понятие *листа* дерева (вершины, не имеющей дочерних вершин); задания 6–8 связаны с обработкой листьев.

В задании 9 вводится понятие *уровней* дерева; задания 9–11 (а также задание 18) связаны с анализом дерева по его уровням. По сравнению с остальными заданиями данной подгруппы задания 9–11 имеют повышенную сложность, поскольку требуют существенной модификации базового рекурсивного алгоритма перебора вершин. В частности, при выполнении заданий 10–11, в которых для каждого уровня дерева требуется определить некоторую его характеристику (количество вершин, расположенных на этом уровне, или сумму их значений), необходимо использовать вспомогательный массив, элементы которого будут содержать требуемые характеристики уровней. Этот массив должен быть описан как внешний по отношению к рекурсивной процедуре, обеспечивающей перебор вершин.

Задания 12–14 вводят понятия *инфиксного*, *префиксного* и *постфиксного* порядков перебора элементов дерева. Данные задания являются достаточно простыми (требуется вывести *все* элементы дерева в указанном порядке); задания 15–17 являются их более сложными модификациями.

Завершающие задания данной подгруппы (19–24) предназначены для закрепления навыков рекурсивного перебора вершин дерева. Все они связаны с нахождением минимальных/максимальных значений вершин дерева, в том числе условных минимумов/максимумов (например, в задании 21 требуется найти минимальное значение среди листьев дерева).

Группа: Tree (анализ бинарного дерева)

Разбор в классе: 2, 6, 9, 12, 23

Закрепление материала: 3, 7, 10, 13, 24

Простые задания: 1, 2, 3, 4, 5, 6, 7, 12, 13, 14

Сложные задания: 9, 10, 11, 18

Наборы однотипных заданий: 2–7; 10–11; 12–14; 15–17; 19–22; 23–24

3.3.2. Формирование бинарного дерева

При создании бинарного дерева необходимо выделять память для хранения его вершин; поэтому в заданиях этой подгруппы требуется использовать либо соответствующие стандартные процедуры выделения памяти (для языков Pascal и C++), либо подходящие варианты конструктора класса Node (для языков платформы .NET).

Первые шесть заданий этой подгруппы (25–30) связаны с формированием бинарных деревьев специального вида (например, в задании 25 требуется создать дерево, каждая вершина которого, кроме корня, является правой дочерней вершиной). В этих заданиях, в отличие от последующих, не требуется использовать рекурсию.

Завершающие задания посвящены трем алгоритмам, связанным с созданием дерева требуемой глубины (31), *идеально сбалансированного дерева* (32–33) и *копии* существующего дерева (34). Поскольку идеально сбалансированное дерево с указанными значениями можно создать различными способами, в задании 32 описывается один из вариантов алгоритма, который требуется использовать при выполнении заданий 32–33 (решение задачи 32 приведено в [8, 9]).

Группа: Tree (формирование бинарного дерева)

Разбор в классе: 26, 32, 34

Закрепление материала: 27, 31, 33

Простые задания: 25

Сложные задания: 31

Наборы однотипных заданий: 26–27; 28–29; 32–33

3.3.3. Преобразование бинарного дерева

В данной подгруппе содержатся задания на различные виды преобразований бинарных деревьев. В заданиях 35–39 рассматриваются простейшие преобразования, не связанные с добавлением или удалением вершин (изменение значений и перемена местами вершин). Задания 40–43 связаны с удалением части вершин. В этих заданиях, как и в аналогичных заданиях на удаление элементов линейных структур, требуется выполнять дополнительные действия по *освобождению выделенных ресурсов* (например, для языка Pascal надо вызывать стандартную процедуру Dispose, а для языков платформы .NET — метод Dispose класса Node). Следует обратить внимание на задание 40, в котором требуется реализовать алгоритм удаления *всех* вершин дерева, кроме корня.

Завершающие задания данной подгруппы (44–47) связаны с добавлением к дереву новых вершин. Среди этих заданий следует выделить достаточно сложное задание 47, в котором требуется, не изменяя глубины исходного дерева, дополнить его до *полного* дерева (дерево глубины L называется полным, если все его внутренние вершины имеют по два непосредственных потомка, а все листья находятся на уровне L).

Группа: Tree (преобразование бинарного дерева)

Разбор в классе: 37, 40, 45, 47

Закрепление материала: 38, 43, 46

Простые задания: 35, 36, 37, 38, 39

Сложные задания: 47

Наборы однотипных заданий: 35–37; 38–39; 42–43; 44–46

3.3.4. Бинарные деревья с обратной связью

В данной подгруппе рассматривается вариант представления бинарного дерева, в котором у дочерних вершин имеется поле Parent, связанное с их родительской вершиной. Такие деревья в задачнике называются *деревьями с обратной связью*.

Первые два задания подгруппы (48–49) посвящены знакомству с новым представлением бинарного дерева. Следует обратить внимание на задание 49, в котором требуется преобразовать исходное бинарное дерево в дерево с обратной связью (решение задачи 49 приведено в [8, 9]). В других заданиях подгруппы подобные преобразования выполнять не придется, поскольку все исходные деревья уже имеют обратную связь.

Далее следуют четыре простых задания, в которых по данным вершинам исходного дерева с обратной связью требуется определить какие-либо характеристики этих вершин или дерева в целом: найти корень дерева (50), уровень данной вершины (51), «степень родства» (разность уровней) или ближайшего общего предка двух данных вершин (52–53). Во всех этих заданиях достаточно использовать итерационные алгоритмы, не прибегая к рекурсии.

Три завершающих задания являются более сложными вариантами ранее рассмотренных заданий на бинарные деревья без обратной связи. В задании 54 требуется создать копию дерева с обратной связью (ср. с заданием 34). Задание 55 является «комбинированным» заданием на преобразование дерева с обратной связью, в котором требуется как добавлять новые, так и удалять существующие вершины (ср. с заданиями из п. 3.3.3). Наконец, задание 56 посвящено формированию дерева с обратной связью требуемой глубины (ср. с заданием 31).

Группа: Tree (бинарные деревья с обратной связью)

Разбор в классе: 49, 52, 54

Закрепление материала: 51, 53, 56

Простые задания: 48, 50, 51

Сложные задания: 56

Наборы однотипных заданий: 50–51; 52–53

3.3.5. Бинарные деревья поиска

С бинарными деревьями поиска связан ряд важных алгоритмов, в том числе один из быстрых алгоритмов сортировки. Поэтому в группу Tree включена

подгруппа, посвященная деревьям поиска. Следует отметить, что большинство заданий этой подгруппы являются достаточно сложными.

Понятия *дерева поиска* и *дерева поиска без повторяющихся элементов* вводятся в первых двух заданиях подгруппы (57–58). В этих заданиях требуется проверить, является ли исходное дерево деревом поиска соответствующего вида, и в случае, если не является, вывести первую (в инфиксном порядке) вершину, нарушающую требуемую закономерность.

Следующие задания (59–60) посвящены алгоритмам нахождения требуемых элементов в дереве поиска. Чтобы оценить эффективность этих алгоритмов, в заданиях необходимо вывести не только найденные вершины, но и количество вершин, которые потребовалось проанализировать для выполнения задания (информация о количестве вершин позволяет также проверить, что при поиске действительно был использован эффективный алгоритм, учитывающий специальный вид бинарного дерева).

Задания 61–64 связаны с алгоритмом добавления нового элемента к дереву поиска (61, 63) или к дереву поиска без повторяющихся элементов (62, 64). Поскольку добавить новый элемент к дереву поиска можно различными способами, в заданиях 61–62 приводится описание того варианта алгоритма, который требуется использовать. Задания 63–64 являются модификациями предыдущих заданий; в них требуется добавить не один, а несколько элементов с указанными значениями.

Так как при инфиксном переборе вершин дерева поиска значения его вершин образуют неубывающую последовательность, для сортировки набора чисел достаточно разместить их в дереве поиска, после чего организовать перебор вершин дерева в инфиксном порядке. В этом и заключается идея алгоритма *сортировки деревом*. Данному виду сортировки посвящены задания 65–66 (в задании 66 требуется получить отсортированный набор исходных чисел *без повторений*). В заданиях требуется вывести не только отсортированную последовательность чисел, но и использованное в ходе алгоритма дерево поиска; это

позволяет проверить, что сортировка действительно была выполнена с помощью дерева. Решение задачи 65 приводится в [8, 9].

Завершающие задания данной подгруппы посвящены алгоритму *удаления* вершин из дерева поиска. В задании 67 рассматривается вариант алгоритма, который надо использовать в случае, если удаляемая вершина содержит не более одной дочерней вершины. В заданиях 68–69 рассматриваются два варианта алгоритма, которые можно использовать в случае, если удаляемая вершина содержит две дочерние вершины. Сами алгоритмы приводятся в формулировках заданий. Наконец, в двух последних заданиях (70–71) заранее не оговариваются свойства удаляемой вершины, поэтому в них необходимо реализовать полный алгоритм удаления (еще одной особенностью этих двух заданий является то, что обрабатываемое дерево является деревом с обратной связью — см. п. 3.3.4).

Группа: Tree (бинарные деревья поиска)

Разбор в классе: 58, 59, 61, 65

Закрепление материала: 57, 60, 62, 66

Простые задания: 59

Сложные задания: 57, 58, 65, 66, 67, 68, 69, 70*, 71*

Наборы однотипных заданий: 57–58; 59–60; 61–62; 63–64; 65–66; 68–69; 70–71

3.3.6. Бинарные деревья разбора выражений

Данная подгруппа дополняет тему, связанную с разбором выражений, которая ранее рассматривалась в разделе «Рекурсия» (см. п. 3.1.2). Под *деревом разбора* арифметического выражения понимается бинарное дерево, вершины которых соответствуют операндам (в частности, числам) и операциям. При этом вершины, соответствующие операциям, кодируются специальными значениями, например, –1 для операции сложения, –2 для операции вычитания и т. д.

Содержащиеся в подгруппе задания можно отнести к одной из двух категорий. В первую категорию входят задания, в которых требуется по строке-описанию некоторого выражения сформировать дерево разбора этого выраже-

ния (72, 74, 76–78, 83). Во вторую категорию входят «обратные» задания: в них дается дерево разбора выражения, по которому требуется либо вычислить данное выражение (79, 84), либо сформировать его строковое описание (73, 75, 80–82, 85).

Задания, связанные с *формированием* дерева разбора выражений, являются достаточно сложными и требуют организации просмотра исходного строкового выражения и его анализа с применением вспомогательных рекурсивных подпрограмм. Указанные действия следует выполнять по схеме, аналогичной той, которая была использована для заданий из п. 3.1.2 (см. соответствующие разделы пособий [6, 9]). В [8, 9] приведено подробное решение задания 74, являющееся «модельным» для всех заданий данной категории.

Задания, связанные с *использованием* данного дерева разбора, являются более простыми; в них достаточно организовать перебор вершин дерева в требуемом порядке, заполняя при этом строку-описание для соответствующего выражения или вычисляя его значение. В [8, 9] приведено решение одной из таких задач (75), прочие задачи решаются аналогично.

В заданиях рассматриваются три вида выражений:

- выражения, описывающие бинарное дерево (72–75);
- арифметические выражения, включающие цифры и операции сложения, вычитания и умножения (76–82);
- выражения, содержащие цифры и вызовы функций $M(x, y)$ и $m(x, y)$, возвращающих соответственно максимальное и минимальное из чисел x и y (83–85).

В заданиях 72–75 используются два варианта описания бинарных деревьев. В более простом варианте (задания 72–73) все вершины описываются единообразно, независимо от количества их непосредственных потомков, например, «4(2(,),6(,7(3(,),)))». В скобках вначале указывается значение левой дочерней вершины, а затем — правой (при условии, что они существуют). В более сложном варианте (74–75) описание вершины зависит от числа ее непосредственных

потомков, например, «4(2,6(,7(3)))». В этом варианте при описании листьев скобки не используются; кроме того, при отсутствии правой дочерней вершины запятая также не указывается.

При описании арифметических выражений применяются либо скобки, явно определяющие порядок выполнения операций (задания 76, 79–80), либо *бесскобочные форматы*: префиксный (77, 81) и постфиксный (78, 82).

Группа: Tree (бинарные деревья разбора выражений)

Разбор в классе: 74, 75, 77, 81

Закрепление материала: 72, 73, 78, 82

Простые задания: *нет*

Сложные задания: 72, 74

Наборы однотипных заданий: 76–78; 79–82

3.3.7. Деревья общего вида

В завершающей подгруппе группы Tree рассматриваются *деревья с произвольным ветвлением*, т. е. деревья, вершины которых могут содержать произвольное число непосредственных потомков. Для краткости в формулировках заданий такие деревья называются *деревьями общего вида*. Деревья общего вида моделируются «обычными» бинарными деревьями, связи в которых интерпретируются особым образом: ссылка Left для любой вершины *N* указывает на ее первую дочернюю вершину, а ссылка Right — на ее правую *сестру*, т. е. вершину, имеющую того же родителя, что и вершина *N*. Таким образом, данная структура содержит как иерархические компоненты типа «предок-потомок» (по ссылкам Left), так и линейные компоненты — односвязные списки сестер (по ссылкам Right).

При выполнении заданий из данной подгруппы важно учитывать особенности интерпретации ссылок Left и Right для деревьев общего вида. Чтобы облегчить эту задачу, в электронном задачнике Programming Taskbook используются различные визуальные представления «обычных» бинарных деревьев и деревьев общего вида.

Как обычно, начальные задания подгруппы (86–87) вводят новые понятия и связывают их с ранее изученными. В задании 86 требуется преобразовать исходное бинарное дерево в дерево общего вида (при этом будет потеряна часть информации, поскольку в дереве общего вида нельзя по единственному потомку определить, является ли он левой или правой дочерней вершиной). Задание 87 посвящено обратному преобразованию дерева общего вида, все вершины которого имеют не более двух непосредственных потомков, в «обычное» бинарное дерево (здесь предполагается, что единственный потомок является *левой* дочерней вершиной). Решение задачи 86 приводится в [8, 9].

Остальные задания подгруппы можно разбить на следующие три категории (большинство этих заданий представляют собой варианты аналогичных заданий для бинарных деревьев):

- *анализ дерева общего вида*, в том числе нахождение его глубины (задание 88 — ср. с заданием 9), анализ вершин по уровням (89–91 — ср. с 10–11), перебор всех вершин в требуемом порядке (92–93 — ср. с 12–14), поиск вершины с требуемым количеством дочерних вершин (94–96);
- *преобразование дерева общего вида* путем изменения значений дочерних вершин (97–98);
- *работа с выражениями, описывающими дерево общего вида*, в том числе создание дерева общего вида по его описанию (99) и формирование такого описания по исходному дереву (100).

Практически во всех алгоритмах, связанных с деревьями общего вида, приходится использовать комбинацию итерации и рекурсии: для перебора дочерних вершин некоторой вершины применяется цикл (как при обработке линейных структур), для обработки каждой дочерней вершины, как правило, применяются рекурсивные подпрограммы. Таким образом, изучение данной подгруппы может рассматриваться как итоговое повторение алгоритмов, связанных с различными динамическими структурами.

Группа: Tree (деревья общего вида)

Разбор в классе: 86, 88, 92, 95, 97

Закрепление материала: 87, 89, 93, 96, 98

Простые задания: *нет*

Сложные задания: 99

Наборы однотипных заданий: 89–90; 92–93; 94–96; 97–98

Литература

1. *Абрамян А. В., Абрамян М. Э.* Практикум по программированию на языке Visual Basic. — Ростов н/Д: «ЦВВР», 2007. — 228 с.
2. *Абрамян М. Э.* 1000 задач по программированию. Часть I: Скалярные типы данных, управляющие операторы, процедуры и функции. — Ростов н/Д: УПЛ РГУ, 2004. — 43 с.
3. *Абрамян М. Э.* 1000 задач по программированию. Часть II: Минимумы и максимумы, одномерные и двумерные массивы, символы и строки, двоичные файлы. — Ростов н/Д: УПЛ РГУ, 2004. — 42 с.
4. *Абрамян М. Э.* 1000 задач по программированию. Часть III: Текстовые файлы, составные типы данных в процедурах и функциях, рекурсия, указатели и динамические структуры. — Ростов н/Д: УПЛ РГУ, 2004. — 43 с.
5. *Абрамян М. Э.* Проведение практических занятий с использованием задачника Programming Taskbook. Методическая разработка для преподавателей программирования. — Банк компьютерных изданий РГУ, 2006. — 83 с. URL: http://open-edu.rsu.ru/upload/pub/37607100_1149508000Archive.zip.
6. *Абрамян М. Э.* Практикум по программированию на языках C# и VB .NET. 2-е изд. — Ростов н/Д: «ЦВВР», 2007. — 220 с.
7. *Абрамян М. Э.* Конструктор учебных заданий для электронного задачника Programming Taskbook. Методическая разработка для преподавателей программирования. — Банк компьютерных изданий ЮФУ, 2009. — 76 с. URL: http://open-edu.sfedu.ru/files/abramyan_taskmaker.pdf.

8. *Абрамян М. Э.* Бинарные деревья: Задачи, решения, указания. — Банк компьютерных изданий ЮФУ, 2009. — 71 с. URL: open-edu.sfedu.ru/files/abramyan-bintrees.zip.

9. *Абрамян М. Э.* Практикум по программированию на языке Паскаль: Массивы, строки, файлы, рекурсия, линейные динамические структуры, бинарные деревья. — 7-е изд., перераб. и доп. — Ростов н/Д: Изд-во ЮФУ, 2010. — 276 с.

10. *Абрамян М. Э., Михалкович С. С.* Основы программирования на языке Паскаль: Скалярные типы данных, управляющие операторы, процедуры и функции, работа с графикой в системе PascalABC.NET. 5-е изд. — Ростов н/Д: «ЦВВР», 2009. — 223 с.

Содержание

Предисловие 3
1. Задания начального уровня 5
1.1. Ввод и вывод данных, оператор присваивания: группа Begin 5
1.2. Целые числа: группа Integer 6
1.3. Логические выражения: группа Boolean 7
1.4. Условный оператор: группа If 8
1.5. Оператор выбора: группа Case 9
1.6. Цикл с параметром: группа For 10
1.7. Цикл с условием: группа While 12
1.8. Последовательности: группа Series 13
1.9. Процедуры и функции: группа Proc 15
1.9.1. Процедуры с числовыми параметрами 16
1.9.2. Функции с числовыми параметрами 17
1.9.3. Дополнительные задания на процедуры и функции 18
1.10.Минимумы и максимумы: группа Minmax 18
2. Обработка массивов, строк и файлов 20
2.1. Одномерные массивы: группа Array 20
2.1.1. Формирование массива и вывод его элементов 20
2.1.2. Анализ элементов массива 22
2.1.3. Работа с несколькими массивами 23
2.1.4. Преобразование массива 24
2.1.5. Серии целых чисел 26
2.1.6. Множества точек на плоскости 27
2.2. Двумерные массивы (матрицы): группа Matrix 28
2.2.1. Формирование матрицы и вывод ее элементов 28

2.2.2. Анализ элементов матрицы 29
2.2.3. Преобразование матрицы 30
2.2.4. Диагонали квадратной матрицы 32
2.3. Символы и строки: группа String 33
2.3.1. Символы и их коды. Формирование строк 33
2.3.2. Посимвольный анализ и преобразование строк. Строки и числа 34
2.3.3. Обработка строк с помощью стандартных функций. Поиск и замена 35
2.3.4. Анализ и преобразование слов в строке 36
2.3.5. Дополнительные задания на обработку строк 37
2.4. Двоичные файлы: группа File 38
2.4.1. Основные операции с двоичными файлами 39
2.4.2. Работа с несколькими числовыми файлами. Файлы-архивы 42
2.4.3. Символьные и строковые файлы 43
2.4.4. Использование файлов для работы с матрицами 45
2.5. Текстовые файлы: группа Text 46
2.5.1. Основные операции с текстовыми файлами 47
2.5.2. Анализ и форматирование текстов 48
2.5.3. Текстовые файлы с числовой информацией 49
2.5.4. Дополнительные задания на обработку текстовых файлов 51
2.6. Составные типы данных в процедурах и функциях: группа Param 52
2.6.1. Одномерные и двумерные массивы 53
2.6.2. Строки 54
2.6.3. Файлы 55
2.6.4. Записи 56
3. Рекурсия и динамические структуры данных 57
3.1. Рекурсия: группа Recur 57
3.1.1. Простейшие рекурсивные алгоритмы 57
3.1.2. Разбор выражений 58

3.1.3. Перебор с возвратом 59
3.2. Динамические структуры данных: группа Dynamic 60
3.2.1. Стек 62
3.2.2. Очередь 63
3.2.3. Двусвязный список 64
3.2.4. Список с барьерным элементом 67
3.3. Бинарные деревья: группа Tree 69
3.3.1. Анализ бинарного дерева 70
3.3.2. Формирование бинарного дерева 72
3.3.3. Преобразование бинарного дерева 73
3.3.4. Бинарные деревья с обратной связью 73
3.3.5. Бинарные деревья поиска 74
3.3.6. Бинарные деревья разбора выражений 76
3.3.7. Деревья общего вида 78
Литература 81