компьютер в учебном процессе

Абрамян Михаил Эдуардович

УДК 004.42+37

ПРИМЕНЕНИЕ ЭЛЕКТРОННОГО ЗАДАЧНИКА ПРИ ПРОВЕДЕНИИ ПРАКТИКУМА ПО ДИНАМИЧЕСКИМ СТРУКТУРАМ ДАННЫХ

Аннотация

В статье рассматриваются пути повышения эффективности практического изучения динамических структур данных с помощью использования электронных задачников. Описывается реализация подобного задачника, включающая 180 заданий на линейные динамические структуры (стеки, очереди, двусвязные списки) и бинарные деревья. Приводятся примеры учебных заданий, иллюстрирующие особенности задачника..

Ключевые слова: электронный задачник, динамические структуры данных, бинарные деревья.

1. ОСОБЕННОСТИ ПРАКТИКУМА ПО ДИНАМИЧЕСКИМ СТРУКТУРАМ И ПУТИ ПОВЫШЕНИЯ ЕГО ЭФФЕКТИВНОСТИ

Изучение динамических структур данных является традиционной частью базового курса программирования. В соответствии с классической книгой Н. Вирта [1, гл. 4], в этом разделе курса рассматриваются линейные динамические структуры (стеки, очереди, списки и т. д.) и иерархические структуры (различные виды деревьев). Параллельно изучаются ссылочные типы данных, что обусловлено особенностями реализации динамических структур. Например, *стек* реализуется в виде односвязной цепочки элементов-узлов, содержащих поле данных (Data) и ссылочное поле (Next), указывающее на следующий элемент цепочки; при этом специальная ссылочная переменная содержит адрес начального узла цепочки – вершины стека (рис. 1). Аналогичным образом реализуется *очередь*, с которой связываются две ссылки: на ее начальный и конечный элементы. Для *списка* обычно используется двусвязная цепочка узлов, связанных не только с последующим, но и с предыдущим элементом (рис. 2), а для *бинарного дерева* –

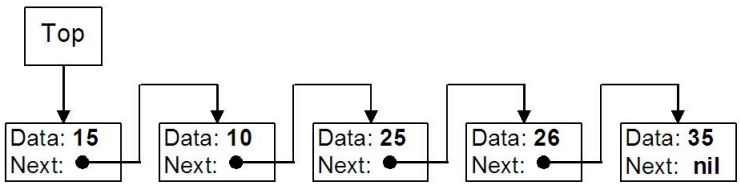

Рис. 1. Реализация стека

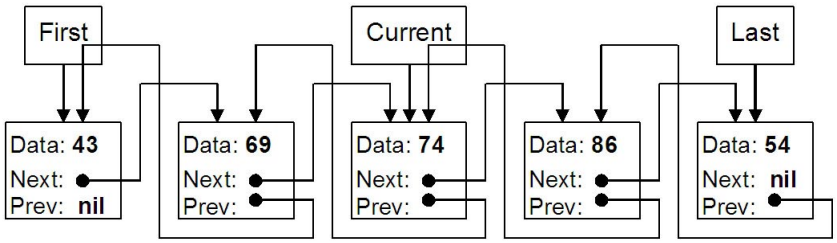

Рис. 2. Реализация двусвязного списка

иерархический набор узлов со ссылочными полями Left и Right (рис. 3).

На практических занятиях студентам предлагается реализовать отдельные операции для изучаемых структур данных и продемонстрировать их правильность на наборе тестовых примеров. В ряде пособий (см., например, [2, 3]) содержится достаточно представительный набор задач по данной теме.

При выполнении заданий, связанных с динамическими структурами, студенты сталкиваются с рядом дополнительных проблем. Трудности начинаются уже на этапе подготовки исходных данных. В самом деле, для того чтобы обработать требуемым образом динамическую структуру, ее надо сформировать в памяти, а эта задача сама по себе является достаточно сложной. Необходимо также учесть, что при обработке динамических структур часто требуется предусмотреть специальные действия в особых ситуациях,

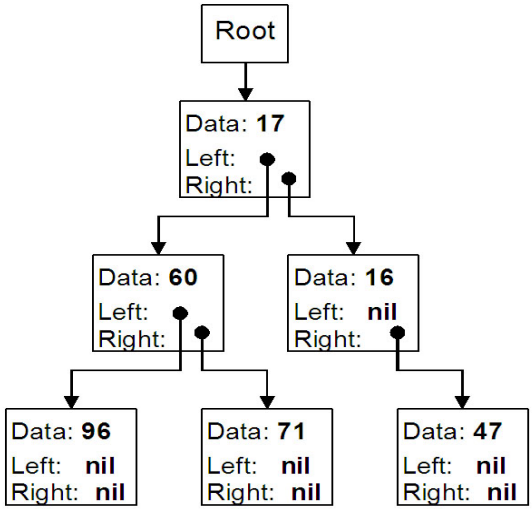

Рис. 3. Реализация бинарного дерева

поэтому при тестировании полученного решения студент должен реализовать несколько вариантов исходных данных, учитывающих возможные особые случаи. Далее, для проверки правильности решения необходимо отобразить на экране как исходную, так и преобразованную динамическую структуру, что также является непростой задачей.

Отмеченные выше проблемы характерны в той или иной степени для всех разделов практикума по программированию, связанных со *сложными структурами данных* (например, с массивами или файлами), однако при изучении динамических структур они проявляются особенно остро.

Следует принимать во внимание и дополнительные трудности, связанные с исправлением ошибок, которые возникают в процессе решения задач на обработку динамических структур. Проблемы здесь двоякого рода: с одной стороны, большинство ошибок при работе со ссылочными типами (в частности, указателями) немедленно приводит к аварийному завершению программы, что затрудняет определение состояния динамической структуры в момент возникновения ошибки; с другой стороны, ошибки, связанные с утечкой памяти, в учебных программах обычно никак не проявляются, что также затрудняет их выявление.

Одним из способов преодоления (или, по крайней мере, смягчения) перечисленных проблем является использование вспомогательных программных средств – *электронных задачников*. В [4] было отмечено, что электронный задачник представляет собой компьютерную систему, которая взаимодействует с программой студента, выполняя следующие действия:

– передача программе студента набора исходных данных и получение от нее результатов;

– автоматическая проверка результатов путем их сравнения с контрольными данными;

– дополнительный контроль правильности операций ввода-вывода;

– наглядное отображение всей информации, связанной с учебным заданием.

Таким образом, задачник обеспечивает автоматизацию именно тех действий, которые особенно сложно реализовать при обработке динамических структур. Поэтому можно ожидать, что его использование окажется наиболее эффективным при изучении данного раздела курса программирования, позволяя студенту выполнять задания в более быстром темпе, более надежно тестировать разработанные алгоритмы и, в конечном итоге, прочнее усваивать изучаемый материал.

Надо, однако, отметить, что использование электронного задачника для поддержки практикума по динамическим структурам требует включения в него дополнительных возможностей. Прежде всего, необходимо, чтобы задачник выполнял автоматическую генерацию исходных динамических структур в оперативной памяти и передавал ссылку на созданные структуры программе студента. Далее, необходимо, чтобы задачник мог получать от программы студента ссылки на созданные или преобразованные динамические структуры; при этом задачник должен обеспечивать проверку их правильности, корректно обрабатывая различные ошибки, которые могут быть выявлены в ходе проверки. Наконец, задачник должен отображать на экране в наглядном виде все используемые в задании динамические структуры вместе со связанными с ними ссылками, причем следует принимать во внимание, что результирующие структуры могут быть созданы с ошибками, информацию о которых также желательно выводить на экран.

Существующие сетевые сервисы для автоматической проверки учебных программ (см., например, [5]) не обладают подобными возможностями, поскольку они позволяют обмениваться с программой студента только данными, представленными в текстовом виде (через стандартные потоки ввода-вывода). Кроме того, в подобных системах обычно не предусматриваются средства визуализации полученных результатов; это обстоятельство не является препятствием для финальной проверки уже разработанной программы, однако затрудняет использование подобных систем для поддержки *процесса* решения задачи на практических занятиях.

В настоящей работе описывается реализация электронного задачника по динамическим структурам данных, выполненная на базе универсального электронного задачника Programming Taskbook [4].

2. СОСТАВ ЭЛЕКТРОННОГО ЗАДАЧНИКА ПО ДИНАМИЧЕСКИМ СТРУКТУРАМ

В состав электронного задачника Programming Taskbook входят две группы заданий, связанные с динамическими структурами данных: Dynamic (80 заданий на линейные динамические структуры) и Tree (100 заданий на бинарные деревья).

Задания группы Dynamic разбиты на 4 подгруппы, каждая из которых посвящена линейной динамической структуре определенного вида (в скобках указано число заданий в подгруппе):

– стеки (13);

– очереди (15);

– двусвязные списки (41);

– списки с барьерным элементом (11).

Задания в подгруппах располагаются по возрастанию уровня сложности и включают задачи на анализ элементов динамической структуры, на удаление или добавление одного или нескольких элементов, на объединение и разбиение однотипных структур, а также (для списков) на перегруппировку элементов.

Каждая подгруппа завершается заданиями, в которых требуется оформить некоторую операцию для рассматриваемой динамической структуры в виде отдельной под-

программы (процедуры, функции) или метода соответствующего класса.

В последней подгруппе рассматривается вариант реализации списка, в котором вместо двусвязной цепочки, ограниченной по краям «пустыми» ссылками (см. рис. 2), используется замкнутая (циклическая) цепочка узлов с дополнительным «барьерным» элементом. Пустой список в данной реализации представляет собой единственный барьерный элемент, замкнутый на себя. Важной особенностью списка с барьерным элементом является отсутствие у его элементов пустых ссылок, что позволяет единообразно обрабатывать элементы и тем самым упрощает реализацию различных операций.

Задания группы Tree разбиты на 7 подгрупп, каждая из которых освещает какой-либо аспект обработки деревьев или посвящена деревьям специального вида:

– анализ бинарного дерева (24);

– формирование бинарного дерева (10);

– преобразование бинарного дерева (13);

– бинарные деревья с обратной связью (9);

– бинарные деревья поиска (15);

– бинарные деревья разбора выражений (14);

– деревья общего вида (15).

В заданиях рассматривается ряд типовых алгоритмов обработки деревьев, в том числе организация перебора вершин в инфиксном, префиксном и постфиксном порядке; формирование идеально сбалансированного дерева; дополнение существующего дерева до полного; добавление и удаление вершины в дереве поиска; сортировка набора данных с использованием дерева поиска; преобразование выражения, представленного в префиксном или постфиксном бесскобочном формате, в соответствующее дерево разбора выражения и выполнение обратного преобразования.

Рассматриваются также бинарные деревья с обратной связью, в которых каждая вершина содержит дополнительную ссылку Parent на свою родительскую вершину, и деревья общего вида (деревья с множественным ветвлением), для вершин которых ссылка Left определяет первую (левую) дочернюю вершину, а ссылка Right – правую *сестру*, то есть вершину, имеющую того же родителя.

3. ВАРИАНТЫ РЕАЛИЗАЦИИ ДЛЯ РАЗЛИЧНЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Задачник Programming Taskbook может использоваться при выполнении заданий на разных языках программирования. В версии 4.11 задачника поддерживаются языки Pascal, Visual Basic, C++, C#, Visual Basic .NET, Python, Java. Задачник интегрирован во многие среды программирования, среди которых можно выделить следующие:

– PascalABC.NET [6], Turbo Delphi 2006; Free Pascal Lazarus 1.0 (язык Pascal);

– Visual Studio 2003, 2005, 2008, 2010 (языки C++, C#, Visual Basic .NET);

– IDLE 2.5, 2.6, 2.7, 3.2 (язык Python),

– NetBeans 6.x и 7.x (язык Java).

Задачник также интегрирован в веб-среду программирования Programming-ABC.NET [7], в которой может использоваться для языков Pascal, C#, Visual Basic. NET и Python.

Среди языков, поддерживаемых задачником, лишь язык Visual Basic не обладает средствами, позволяющими разрабатывать структуры, основанные на ссылочных типах данных. Для языков Pascal и C++ для таких целей можно использовать *записи* (record в Pascal, struct в C++) и указатели на них. В варианте задачника для этих языков описан тип TNode, содержащий поле данных и все поля связи, которые могут потребоваться для реализации различных структур, используемых в заданиях. Поле данных Data, определяющее значение элемента динамической структуры, имеет целочисленный тип; все поля связи имеют тип PNode — указатель на TNode. В листинге 1 приведено описание этих типов для языка Pascal (в языке C++ типы TNode и PNode описываются аналогичным образом).

В языках Python и Java указатели отсутствуют. В языках C# и Visual Basic .NET указатели имеются, однако их применение нехарактерно для данных языков. В то же вре-

мя во всех перечисленных языках реализована ссылочная объектная модель, поэтому в качестве ссылочных переменных можно использовать *объекты*, то есть экземпляры классового типа. В варианте задачника для указанных языков определен класс Node с открытыми свойствами Data, Next, Prev, Left, Right, Parent и набором конструкторов, обеспечивающих инициализацию требуемых свойств. С помощью экземпляров данного класса можно моделировать все виды структур, используемых в заданиях. В вариантах групп Dynamic и Tree, ориентированных на использование объектов, несколько изменены формулировки ряда заданий, однако эти изменения имеют чисто терминологический характер (например, вместо слова «указатель» используется слово «ссылка»).

Таким образом, задания, связанные с динамическими структурами данных, можно выполнять для всех языков, поддерживаемых задачником (за исключением языка Visual Basic), причем для реализации подобных структур применяются средства, наиболее характерные для каждого языка.

4. ВИЗУАЛИЗАЦИЯ ДИНАМИЧЕСКИХ СТРУКТУР

Одной из основных особенностей задачника, существенно ускоряющих процесс выполнения учебных заданий, является автоматизация действий по отображению данных на экране: все данные, выведенные программой студента, автоматически форматируются и отображаются в предусмотренных для них позициях в соответствующем разделе окна задачника. Отмеченная возможность реализована и для таких данных, как динамические структуры и связанные с ними ссылки (указатели или объекты).

В ходе выполнения задания на динамические структуры задачник предоставляет программе студента ссылки на исходные структуры (автоматически созданные задачником в динамической памяти), а программа студента передает задачнику ссылки на созданные (или преобразованные) ею структуры, что позволяет выполнить проверку их правильности. При этом требуется обеспечить визуализацию как ссылочных данных, которыми обмениваются задачник и программа студента, так и связанных с ними структур. Однако вывод на экран самого значения ссылки (то есть адреса некоторого участка памяти) не будет нести для студента никакой содержательной информации. Вместо этого необходимо каким-либо наглядным способом показать *связь* ссылки с тем элементом структуры, на которую она указывает. В задачнике Programming Taskbook проблема визуализации ссылок решена следующим образом: каждая ссылка, используемая в задании, автоматически снабжается *меткой с индексом*, имеющей вид P_K (для указателей) или A_K (для объектов); если ссылка не является нулевой, то ее метка выводится рядом с адресуемым ею элементом динамической структуры.

Листинг 1

```
type
  PNode = ^TNode;
  TNode = record
    Data: integer;
    Next, Prev, Left,
    Right, Parent: PNode;
  end;
```

Приведем примеры, которые иллюстрируют не только описанный способ визуализации ссылок, но и варианты представления различных динамических структур.

На рис. 4 приведена односвязная структура, моделирующая стек; при этом указатель P_1 содержит адрес вершины стека. Тот факт, что структура является односвязной, отмечается одинарными линиями, соединяющими числа – значения соответствующих узлов. Кроме того, явным образом отмечается, что последний элемент стека содержит в своем поле Next значение nil. Таким образом, данное представление является упро-

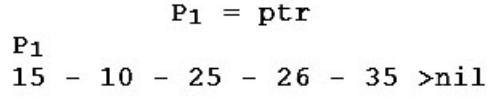

Рис. 4. Представление стека и указателя на его вершину

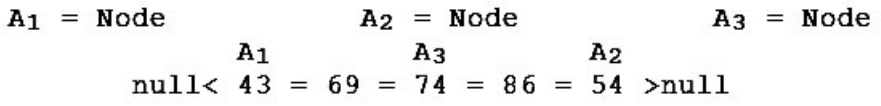

Рис. 5. Представление двусвязного списка и связанных с ним объектов

щенным вариантом традиционного изображения стека, приведенного на рис. 1. В качестве «значения» указателя P_1 приводится текст «ptr» (сокращение слова «pointer»), поскольку, как уже было сказано, явное указание адреса в данном случае не добавило бы никакой содержательной информации. Однако в результате ввода программой студента указателя P_1 она получит «настоящий» адрес вершины стека, созданного задачником (для ввода указателей в задачнике предусмотрены специальные средства, зависящие от используемого языка программирования).

На рис. 5 приведена двусвязная структура, моделирующая список, и три связанные с ней ссылки: на начальный, конечный и текущий элементы списка. Использованные в качестве «значений» ссылок имена «Node» показывают, что данный пример взят из варианта задания, ориентированного на использование объектов. Наличие двойной связи между элементами списка отмечается двойными линиями, соединяющими значения элементов. Показано также, что начальный и конечный элементы содержат нулевые ссылки в своих свойствах Prev и Next соответственно (имя нулевой ссылки «null» свидетельствует о том, что используется язык C# или Java). Приведенное представление соответствует списку, изображенному на рис. 2.

Аналогичные способы визуализации используются и в заданиях, посвященных бинарным деревьям. Приведем три примера.

На рис. 6 изображено бинарное дерево вместе с указателем P_1 на его корень. Для большей наглядности рядом с деревом отображаются уровни его вершин (уровень корня считается равным 0). В отличие от линейных структур, для деревьев нулевые поля связи не отображаются: если поле Left или Right некоторой вершины равно nil, то на изображении дерева у этой вершины просто отсутствует левая или правая связь. Приведенное представление соответствует бинарному дереву, изображенному на рис. 3.

На рис. 7 изображено так называемое *бинарное дерево с обратной связью*, в котором все вершины содержат ссылки Parent на свою родительскую вершину (для корня эта ссылка является нулевой). Наличие обратной связи изображается двойными горизонтальными линиями. Заметим, что для доступа к дереву с обратной связью достаточно иметь ссылку на какую-либо его вершину (не обязательно корень).

Наконец, на рис. 8 изображено *дерево с произвольным ветвлением*, для которого используется представление «левая дочерняя

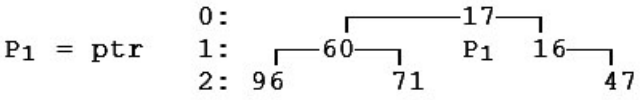

Рис. 6. Представление бинарного дерева с указателя на его корень

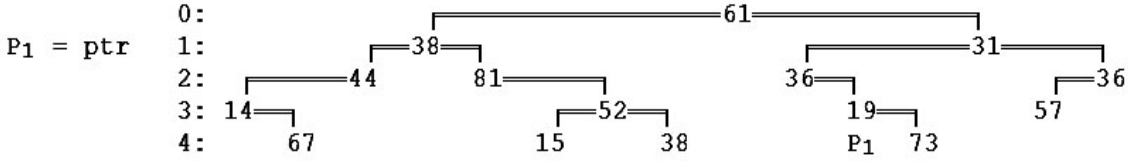

Рис. 7. Представление бинарного дерева с обратной связью

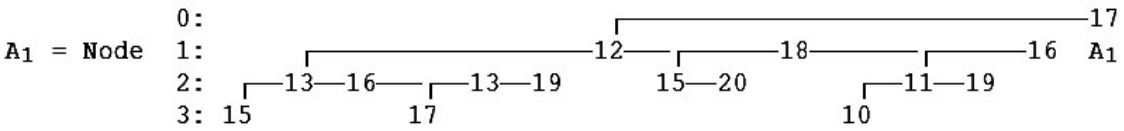

Рис. 8. Представление дерева с произвольным ветвлением

вершина – правая сестра» («left child – right sibling»). В данном случае все вершины-«сестры» изображаются на одном уровне, что соответствует реальной структуре дерева. В частности, корень (на который ссылается объект A_1) имеет три дочерних вершины (со значениями 12, 18 и 16), вершина 12 – четыре дочерних вершины (13, 16, 13, 19), а вершина 18 – две (15 и 20).

Некоторые дополнительные обозначения, используемые при визуализации динамических структур, будут описаны в пунктах, посвященных обработке ошибок и примерам выполнения типовых заданий.

5. ОБРАБОТКА ОШИБОК

Электронный задачник распознает основные типы ошибок, связанных с вводом-выводом, а именно ввод-вывод недостаточного числа данных, попытку ввода-вывода лишних данных и ввод-вывод данных неверного типа. Кроме того, он перехватывает все ошибки времени выполнения, возникающие в программе студента, и выводит соответствующие сообщения в своем окне. При проверке задач на динамические структуры задачник также выявляет ошибки в структурах, созданных или преобразованных программой студента, и отображает их в наглядной форме.

На рис. 9 приведен результат выполнения задания, в котором требовалось определенным образом перегруппировать элементы исходного двусвязного списка и вывести ссылку на его последний элемент. При выполнении задания было допущено несколько ошибок, которые можно выявить, анализируя представление полученного списка. Во-первых, был выведен указатель не на последний элемент списка, а на предпоследний (метка P_2 располагается над четвертым элементом, а в списке содержится пять элементов); во-вторых, между вторым и третьим элементом отсутствует обратная связь (это видно по одинарной линии); в-третьих, в первом элементе списка поле Prev не равно nil (в противном случае перед этим элементом был бы выведен текст «nil<»); в-четвертых, значение поля Next у последнего элемента является неверным и указывает на посторонний участок памяти (об этом свидетельствуют символы-звездочки, которые дополнительно выделяются красным цветом). Приведенный пример является условным, так как на практике такое количество ошибок редко возникает одновременно. Он лишь демонстрирует различные способы, которыми задачник информирует студента о допущенных ошибках. Аналогичные способы визуализации ошибок предусмотрены и для бинарных деревьев.

При каждом запуске программы, выполняющей задание, в окне задачника отображаются не только результаты, полученные программой, но и пример правильных результатов. Сравнение этих наборов данных (в частности, сравнение изображений динамических структур) также дает возможность студенту быстро обнаружить ошибки в своем решении.

Особо следует остановиться на ошибках, связанных с утечкой памяти. Как было отмечено выше, при традиционном способе

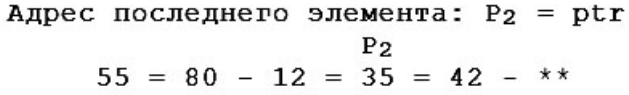

Рис. 9. Пример ошибочной динамической структуры

выполнения заданий эти ошибки никак не проявляются в учебных программах, что существенно затрудняет их обнаружение. В задачнике реализован механизм, автоматизирующий выявление подобных ошибок. При выполнении задания, в котором требуется удалить из исходной динамической структуры некоторые элементы, решение не будет считаться правильным, если для всех удаленных элементов программа студента не вызовет соответствующую операцию освобождения памяти (для языка Pascal это процедура Dispose, которая переопределяется в задачнике, для языка C++ это переопределенный в структуре TNode оператор delete). Если для каких-либо элементов, которые необходимо удалить, память не освобождена, то выводится соответствующее сообщение, а изображения этих элементов в исходной динамической структуре выделяются красным цветом.

Аналогичный механизм реализован и для языков, в которых для выполнения заданий вместо указателей используются ссылки на объекты типа Node. Для любого такого языка (Visual Basic .NET, C#, Python, Java) в описание класса Node включен метод Dispose, который определяется как *метод освобождения ресурсов*, выделенных для объекта данного класса. Для правильного выполнения задания программа студента должна вызвать метод Dispose для всех объектов, удаляемых из исходных динамических структур. Данный подход представляется оправданным, так как даже для языков, в которых реализовано автоматическое освобождение неиспользуемой памяти (автоматическая сборка мусора), проблема своевременного освобождения других (неуправляемых) ресурсов по-прежнему остается актуальной.

Листинг 2

```
uses PT4;
var
  p1, p2, p: PNode;
begin
  Task('Dynamic8');
  GetP(p1);
  GetP(p2);
  while p1 <> nil do
  begin
    p := p1;
    p1 := p1^.Next;
    p^.Next := p2;
    p2 := p;
  end;
  PutP(p2);
end.
```

6. ПРИМЕРЫ ВЫПОЛНЕНИЯ УЧЕБНЫХ ЗАДАНИЙ

В качестве иллюстрации описанных выше возможностей приведем несколько примеров выполнения простых заданий групп Dynamic и Tree на различных языках программирования. Заметим, что в задачнике предусмотрен вспомогательный программный модуль, позволяющий создавать программы-заготовки для требуемого задания на выбранном языке (краткое описание процесса создания заготовки приведено, например, в [8]).

Первое из рассматриваемых заданий посвящено обработке стеков.

Dynamic8. Даны указатели P_1 и P_2 на вершины двух непустых стеков. Переместить все элементы из первого стека во второй (в результате элементы первого стека будут располагаться во втором стеке в порядке, обратном исходному) и вывести адрес новой вершины второго стека. Операции выделения и освобождения памяти не использовать.

Таким образом, в этом задании не требуется выполнять действия по созданию или разрушению элементов.

Пример программы с правильным решением на языке Pascal приведен в листинге 2. Эта программа может запускаться в любой из Pascal-сред, поддерживаемых задачником (PascalABC.NET, Delphi, Free Pascal Lazarus). Директива uses подключает модуль PT4 с дополнительными подпрограммами задачника, процедура Task обеспечивает инициализацию указанного задания, процедуры GetP и PutP осуществляют соответственно ввод и вывод данных-указателей типа PNode. Прочие операторы програм-

мы в дополнительных пояснениях не нуждаются.

Окно задачника, отображаемое на экране при первом запуске данной программы, приведено на рис. 10. При каждом тестовом запуске программы задачник генерирует новый набор исходных данных. Задание считается выполненным, если программа успешно пройдет требуемое количество тестовых испытаний; при ошибочной обработке некоторого тестового набора отсчет успешных тестовых запусков начинается заново.

Во втором задании требуется дополнить существующий двусвязный список. Приведем вариант формулировки этого задания, ориентированный на применение ссылочных объектов и поэтому не использующий понятие указателя.

Dynamic35. Даны ссылки A_1 и A_2 на первый и последний элементы двусвязного списка, содержащего не менее двух элементов. Продублировать в списке первый и последний элементы (новые элементы добавлять перед существующими элементами с такими же значениями) и вывести ссылку на первый элемент преобразованного списка.

Программа с правильным решением на языке Python приведена в листинге 3. В этой программе также импортируется модуль pt4

Листинг 3

```
# -*- coding: cp1251 -*-
from pt4 import *
task("Dynamic35")
a1 = get()
a2 = get()
a = Node(a2.Data, a2, a2.Prev)
a2.Prev.Next = a
a2.Prev = a
a = Node(a1.Data, a1, None)
a1.Prev = a
put(a)
```

и используются функции задачника task, get и put (поскольку язык Python является языком с динамической типизацией, в варианте задачника для этого языка предусмотрены универсальные функции ввода и вывода). В программе используются объекты типа Node; для создания таких объектов вызывается конструктор с тремя параметрами: значениями свойств Data, Next и Prev создаваемого объекта.

Вид окна задачника приведен на рис. 11. Отметим особенность отображения полученных динамических структур, которая ранее не упоминалась: элементы, созданные программой студента, дополнительно обрамляются точками (в окне на рис. 11 таким

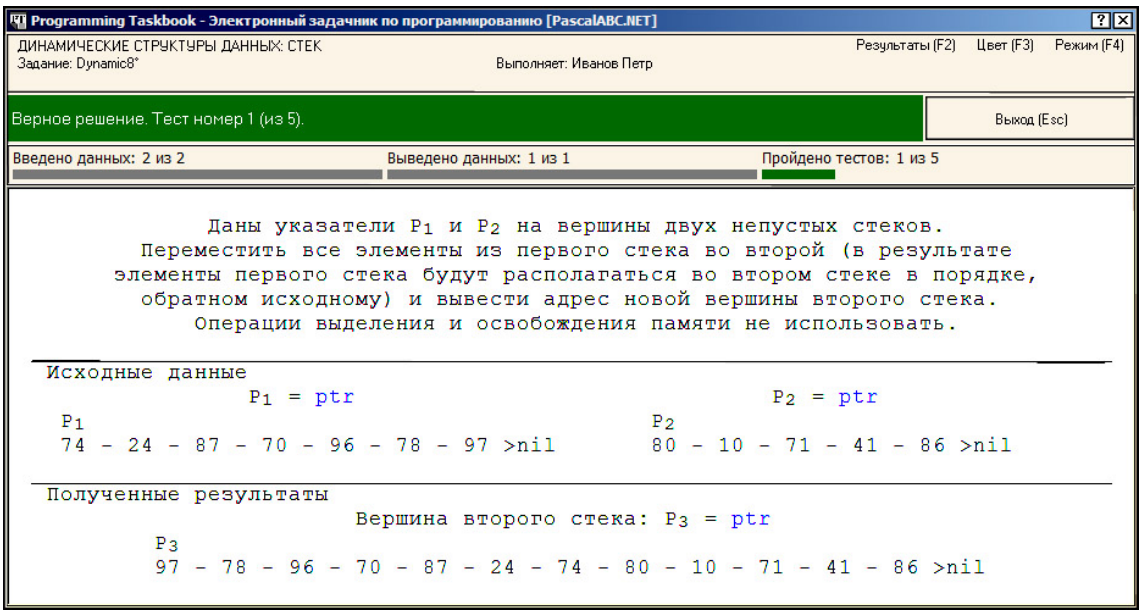

Рис. 10. Первый успешный запуск программы с решением задачи Dynamic8 (язык Pascal)

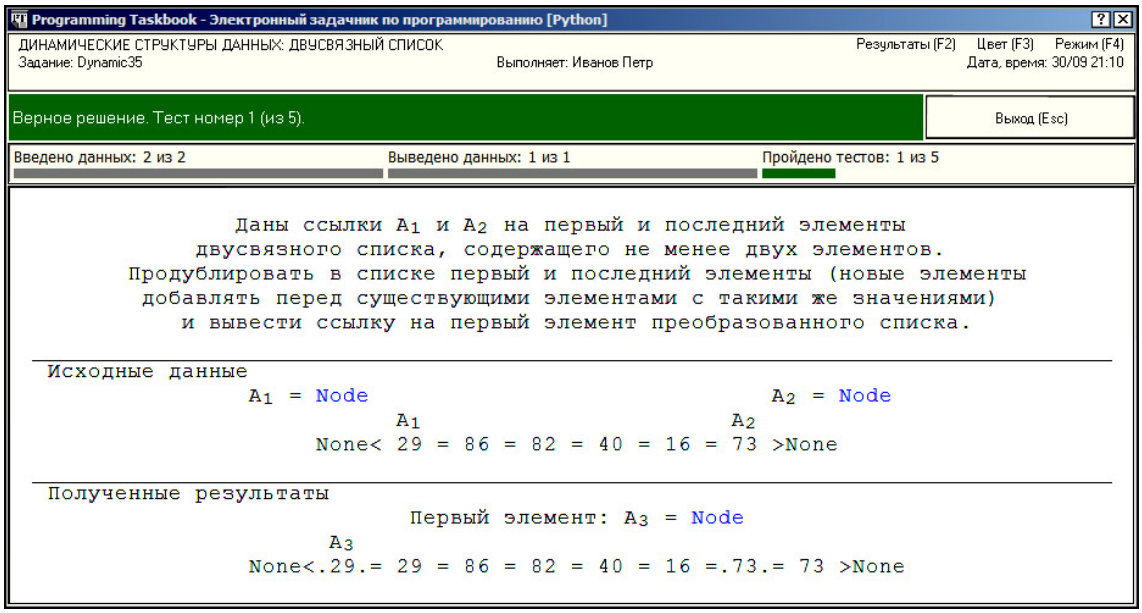

Рис. 11. Первый успешный запуск программы с решением задачи Dynamic35 (язык Python)

образом выделены первый и предпоследний элемент преобразованного списка).

Последний пример демонстрирует решение задачи из группы Tree на языке C#.

Tree34. Дан корень A_1 непустого дерева. Создать копию данного дерева и вывести ссылку A_2 на корень созданной копии.

При выполнении заданий на языке C# решение оформляется в виде метода Solve вспомогательного класса MyTask – потомка класса PT, который импортируется из библиотеки pt4net.dll, входящей в состав задачника. Если в решении требуется использовать вспомогательные подпрограммы, то они оформляются как статические методы класса MyTask. Описание класса MyTask с решением задачи Tree34 приведено в листинге 4 (прочие элементы проекта, в том числе класс со стартовым методом Main, создаются задачником автоматически и не требуют изменения). В данном решении используются определенные в классе PT методы Task (инициализация задания), GetNode (ввод объектов типа Node) и Put (вывод результатов любого типа). Для создания новой вершины дерева используется вариант конструктора класса Node с параметрами, определяющими значения свойств Left, Right, Data (в указанном порядке); класс Node импортируется из библиотеки pt4net.dll.

Листинг 4

```
public class MyTask: PT
{
  public static Node CopyTree(Node p)
  {
    if (p == null)
      return null;
    return
      new Node(CopyTree(p.Left),
      CopyTree(p.Right), p.Data);
  }
  public static void Solve()
  {
    Task("Tree34");
    Put(CopyTree(GetNode()));
  }
}
```

Вид окна задачника приведен на рис. 12. В данном случае обрамляются точками все вершины дерева-копии, поскольку все эти вершины созданы в программе, решающей задачу. Заметим, что подобное выделение позволяет распознать недобросовестный вариант решения, при котором программа студента просто выводит ссылку на корень *исходного* дерева (не создавая копии). В этом случае элементы результирующего дерева не

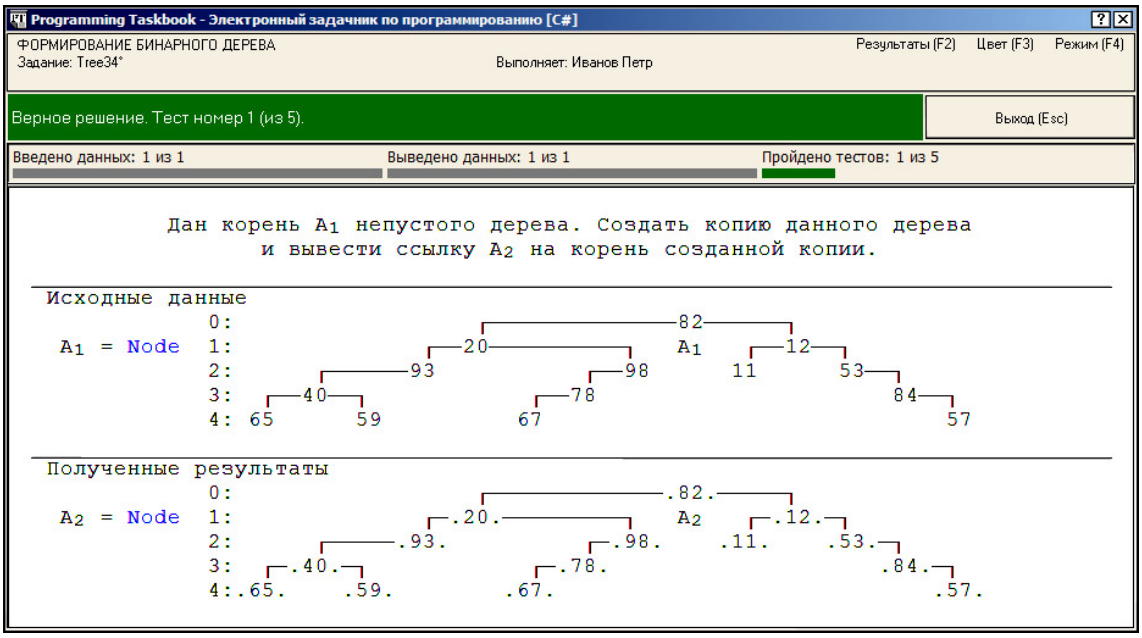

Рис. 12. Первый успешный запуск программы с решением задачи Tree34 (язык C#)

будут обрамляться точками, и задачник оценит это решение как ошибочное.

В заключение отметим, что с помощью конструктора учебных заданий PT4TaskMaker [9] для задачника Programming Taskbook можно разрабатывать новые группы учебных заданий, в том числе и связанные с динамическими структурами данных. Новые группы оформляются в виде динамических библиотек (dll-файлов), что делает их доступными для любых языков и сред программирования, поддерживаемых задачником.

Формулировки всех заданий групп Dynamic и Tree и решения типовых задач для различных языков программирования приводятся на сайте электронного задачника http://ptaskbook.com. Раздел сайта, посвященный конструктору PT4TaskMaker, содержит подробное описание средств конструктора, связанных с разработкой заданий на динамические структуры.

Литература

1. *Вирт Н.* Алгоритмы и структуры данных. М.: Мир, 1989.
2. Программирование на языке Паскаль: задачник / *Под ред. О.Ф. Усковой.* СПб.: Питер, 2002.
3. Задачи по программированию / *С.М. Окулов, Т.В. Ашихмина, Н.А. Бушмелева и др. Под ред. С.М. Окулова.* М.: БИНОМ, 2006.
4. *Абрамян М.Э.* Реализация универсального электронного задачника по программированию // Информатика и образование, 2009. № 6. С. 118–120.
5. Ejudge contest management system / http://ejudge.ru/ (дата обращения: 01.12.2012).
6. PascalABC.NET / http://pascalabc.net/ (дата обращения: 01.12.2012).
7. Веб-среда программирования ProgrammingABC.NET / http://pascalabc.net/WDE/ (дата обращения: 01.12.2012).
8. *Абрамян М.Э.* Использование электронного задачника по строковым алгоритмам биоинформатики // Компьютерные инструменты в образовании, 2012. № 3. С. 47–56.
9. *Абрамян М.Э.* Использование специализированного программного обеспечения для преподавателя при организации и проведении лабораторных занятий по программированию // Информатика и образование, 2011. № 5. С. 78–80.

Abstract

We discuss some ways to improve efficiency of dynamic data structures studies based on the application of the dedicated educational software. Then we describe the electronic book of educational training tasks that includes 180 tasks on dynamic data structures (stacks, queues, doubly linked lists and binary trees). We also provide solutions for exemplary problems.

Keywords: educational software, dynamic data structures, binary trees.

Абрамян Михаил Эдуардович,
кандидат физико-математических наук, доцент кафедры алгебры и дискретной математики Южного федерального университета,
mabr@math.sfedu.ru