Число уровней в дереве: *<n+1>* Некорректные входные данные!

Указание.

Породить от класса "Двоичное Дерево" класс "Особое Дерево", добавив метод "Построение Дерева От N".

8. Список рекомендуемой литературы

1. Лэнгсам Й. и др. Структуры данных для персональных ЭВМ. М.: Мир, 1989.
2. Вирт Н. Алгоритмы+Структуры данных=Программы. М.: Мир, 1985.
3. Вьюкова Н. и др. Систематический подход к программированию. М.: Наука, 1988.

Методические указания по вычислительной практике

Составитель: М.Л. Цымблер
Редактор: Е.А. Иванова

Подписано в печать 14.07.95. Формат 60×84/16.
Бумага типографская № 2.
Печать ротапринтная.
Уч.-изд. л.1,88. Усл. печ. л. 1,86.
Тираж 50 экз. Заказ . Бесплатно.

Челябинский государственный университет.
454136 Челябинск, ул.Братьев Кашириных, 129.

Участок оперативной полиграфии ЧелГУ.
454136 Челябинск, ул.Молодогвардейцев, 57а.

ГОСУДАРСТВЕННЫЙ КОМИТЕТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ПО ВЫСШЕМУ ОБРАЗОВАНИЮ

Челябинский государственный университет

Кафедра информатики и методов вычислений

Методические указания по вычислительной практике

для студентов 1 курса математического факультета

(специальность "Прикладная математика")

ЧЕЛЯБИНСК 1995

Одобрено учебно-методическим советом математического факультета Челябинского государственного университета.

Методические указания содержат необходимую информацию для студентов при прохождении вычислительной практики: обязанности студентов, примерное распределение часов по видам работ при выполнении заданий, требования к оформлению отчета, образец отчета, задания по вычислительной практике.

Предназначены для студентов 1 курса математического факультета (специальность "Прикладная математика").

Составитель:

ассистент кафедры информатики и методов вычислений ЧелГУ М.Л. Цымблер.

Рецензент:

доцент кафедры информатики и методов вычислений ЧелГУ, кандидат физ.-мат. наук В.А. Окороков.

1. Обязанности студента при прохождении практики

Во время прохождения практики студент обязан:

1. Разработать алгоритм решения задачи.
2. Составить программу, удовлетворяющую следующим требованиям:
   - модульная структура;
   - самодокументируемость;
   - естественный для пользователя вид печатаемых программой результатов.
3. Разработать набор тестов и провести тестирование программы.
4. Оформить отчет о прохождении практики с учетом требований к оформлению отчета.

2. Технологический цикл разработки программного обеспечения

Разработка любого серьезного программного обеспечения проходит по следующим этапам:

1. Определение требований к программе (или составление внешних спецификаций).

Определяется и в формализованном виде записывается, ***что*** должна делать программа.

2. Проектирование:

2.1. Разработка иерархии модулей.

Основная задача разбивается на более мелкие подзадачи-модули. Составляется иерархическая схема модульной структуры программы - графическое изображение связей по управлению между всеми используемыми в программе модулями.

2.2. Разработка внутренних спецификаций модулей.

В формализованном виде записывается, ***как*** модуль будет выполнять возложенную на него задачу.

3. Подготовка тестов:

3.1. Написание тестовых программ для каждого модуля.

Тестовая программа составляется так, чтобы проверить все функции, указанные во внутренней спецификации модуля.

3.2. Разработка набора тестов для каждого модуля.

Тест представляет собой некоторую совокупность входных данных и точное описание всех результатов и сообщений, которые должна выработать тестовая программа на этих данных. Набор тестов подбирается так, чтобы при тестировании пройти каждую ветвь программы.

4. Кодирование (программирование):

4.1. Разработка алгоритмов и их кодирование.

4.2. Отладка.

Цель отладки - устранить ошибки в программе (модуле). На этом этапе рассматриваются только те ошибки, которые проявляются (остальные могут оказаться нетронутыми).

5. Тестирование.

Цель тестирования - убедиться в том, что программа (модуль) функционирует как следует, что она (он) соответствует спецификациям и решает поставленную задачу. На этом этапе выявляются ошибки, которые должны быть исправлены.

3. Порядок работ и распределение времени

Порядок работ соответствует этапам технологического цикла разработки программного обеспечения[1].

№ п/п	Вид работы	Срок сдачи, день от начала практики (в конце рабочего дня)
1	Внешние спецификации	1
2	Иерархическая схема модульной структуры программы	1
3	Внутренние спецификации базового модуля (1-я часть задания)	3
4	Тестовая программа для базового модуля	5
5	Набор тестов для базового модуля	5
6	Реализация базового модуля	8
7	Тестовые прогоны базового модуля	8
8	Внутренние спецификации прикладного модуля (2-я часть задания)	9
9	Тестовая программа прикладного модуля	9
10	Набор тестов для прикладного модуля	9
11	Реализация прикладного модуля	11
12	Тестовые прогоны прикладного модуля	11
13	Отчет	11[2]
14	Зачет	12[3]

Студент обязан не позже даты контроля отчитаться за прохождение каждого этапа. Запрещается приступать к следующему этапу, не отчитавшись за предыдущий этап.

---

[1] О заданиях см.: 7. Задания по вычислительной практике, с. 12.

[2] В начале рабочего дня.

[3] В течение дня.

4. Содержание отчета. Требования к оформлению отчета

Отчет по вычислительной практике должен содержать следующие разделы:

1. Внешние спецификации.
2. Проектирование:
   - 2.1. Иерархическая схема модульной структуры программы.
   - 2.2. Внутренние спецификации каждого модуля программы.
3. Кодирование.

Список закодированных модулей программы.

4. Тестирование.

Набор тестов и тестовые прогоны каждого модуля программы.

5. Приложение.

Исходные тексты тестовых программ (для каждого модуля программы).

Отчет должен быть написан разборчивым почерком (напечатан) и аккуратно оформлен.

5. Критерии итоговой оценки

Критерии оценки за вычислительную практику:

1. Наличие отчета, оформленного согласно требованиям к оформлению отчета.
2. Соответствие исходных текстов требованиям к оформлению программных продуктов на языке Turbo Pascal.
3. Наличие полного набора тестов.
4. Работа тестовых программ.
5. Работа конечной программы.
6. Умение отвечать на вопросы относительно работы тестовых и конечной программ.

Студенту гарантирована оценка “удовлетворительно”, если он выполнил первую часть задания согласно требованиям (наличие отчета и т.д.).

Студент обязан подготовить к зачету каталог на жестком диске со всеми исходными текстами, файлами данных и исполняемыми файлами.

6. Образец оформления отчёта

ГОСУДАРСТВЕННЫЙ КОМИТЕТ РОССИЙСКОЙ ФЕДЕРАЦИИ
ПО ВЫСШЕМУ ОБРАЗОВАНИЮ

Челябинский государственный университет

Кафедра информатики и методов вычислений

Отчет по вычислительной практике

Выполнил: Иванов П.Ф.

Группа: ПМ-101

Руководитель: Петров И.И.

ЧЕЛЯБИНСК 1995

1. Внешние спецификации

Под выражением понимается конструкция следующего вида:
<выражение> ::= <терм> | <терм> <знак> <выражение>
<знак> ::= + | -
<терм> ::= <множитель> * <терм>
<множитель> ::= <число> | <переменная> | <выражение>
<число> ::= 0 | 1 | 2 | ... | 9
<переменная> ::= A | B | C | ... | Z.

Польской записью выражения A [операция] B называется запись, в которой знак операции размещен за операндами: A B [операция].
Например:

Обычная запись	Польская запись	
a-b	ab-	
a*b+c	ab*c+	т.е. (ab*)c+
a*(b+c)	abc+*	т.е. a(c+b)*

Имеется текстовый файл infix.txt, в котором в одну строку записано выражение. Прочитать его из файла и перевести в польскую запись. Результат перевода записать в текстовый файл result.txt в следующем виде:

```
Исходное выражение:
<исходное выражение>.
Польская запись:
<польская запись выражения>.
```

Считать, что исходное выражение корректно.

2. Проектирование

2.1. Иерархическая схема модульной структуры программы

- СТЕК: Базовый модуль. Реализация класса "Стек элементов типа char".
- ПОЛЬСКАЯ ЗАПИСЬ: Прикладной модуль. Реализация класса "Польская запись", осуществляющего перевод выражения из обычной формы записи в польскую.
- Стрелка ↑: "вызывает".

2.2. Внутренние спецификации модулей программы

2.2.1. Внутренние спецификации базового модуля

```
{ CLSTACK.PAS
  ---------------------------------
  Дата      17.08.95
  Для       Turbo Pascal 6.0
  (c)оздал  Иванов П.Ф.
  ---------------------------------------------------
  Реализация класса "Стек элементов типа char". }

unit ClStack;

interface

type
  PNode = ^TNode;
  TNode = record
     Info: Char;
     Next: PNode;
  end;

  Stack = object
     Top: PNode;
       { вершина стека }
     procedure Init;
       { инициализация стека }
     procedure Done;
       { уничтожение стека }
     procedure Push(Elem: Char);
       { добавление элемента в стек }
     function Pop: Char;
       { удаление элемента из стека }
     function GetTop: Char;
       { взятие верхнего элемента стека }
     function Empty: Boolean;
       {·провер.ка стека на пустоту }
  end;

procedure WriteStack(S: Stack; F: Text);
     { запись содержимого стека S в текстовый файл F или на экран,
если F=Output }
```

2.2.2. Внутренние спецификации прикладного модуля

```
{ CLPOLISH.PAS
```



```
  ---------------------------------
  Дата      17.08.95
  Для       Turbo Pascal 6.0
  (c)оздал  Иванов П.Ф.
  ---------------------------------------------------
  Реализация класса "Польская запись". }

unit ClPolish;

interface

type
  Polish = object (Stack)
     function Infix2Postfix(S: String): String;
       { переводит выражение S из обычной формы записи в польскую }
  end;
```

3. Список закодированных модулей программы

1) CLSTACK.PAS - базовый модуль, реализация класса "Стек элементов типа char".
2) CLPOLISH.PAS - прикладной модуль, реализация класса "Польская запись".

4. Тестирование

4.1. Тестирование базового модуля

Входные данные: .сорпов меч в тоВ ?ьтыб ен или ьтыБ
Стек формируется последовательным считыванием входных данных.

№ п/п	Действие	Ожидаемый результат	Действительный результат
1	выдача стека на экран	Быть или не быть? Вот в чем вопрос.	Совпадает
2	Удаление элемента из стека	Удален символ 'Б'	Совпадает
3	Удаление элемента из стека	Удален символ 'ы'	Совпадает
4	Удаление элемента из стека	Удален символ 'т'	Совпадает
5	Удаление элемента из стека	Удален символ 'ь'	Совпадает
6	Добавление элемента 'ь' в стек	Добавлен символ 'ь'	Совпадает

№ п/п	Действие	Ожидаемый результат	Действительный результат
7	Добавление элемента 'т' в стек	Добавлен символ 'т'	Совпадает
8	Добавление элемента 'и' в стек	Добавлен символ 'и'	Совпадает
9	Добавление элемента 'П' в стек	Добавлен символ 'П'	Совпадает
10	Выдача стека на экран	Пить или не быть? Вот в чем вопрос.	Совпадает

4.2. Тестирование прикладного модуля

Перевод выражения из обычной формы записи в польскую.

№ п/п	Входные данные	Ожидаемый результат	Действительный результат
1	3	3	совпадает
2	a-b	ab-	совпадает
3	a*b+c	ab*c+	совпадает
4	a*(b-1)	ab1-*	совпадает
5	a+(2*b-c*d*4)	a2b*cd*4*-+	совпадает

5. Приложение

5.1. Тестовая программа для базового модуля

```
{ TSTSTACK.PAS
  ----------------------
  Дата      17/08/95
  Для       Turbo Pascal 6.0
  (c)оздал Иванов П.Ф.
  ---------------------------------------------------------------
  Тестовая программа для класса "Стек элементов типа Char". }

uses ClStack;

var
  S: Stack;
  F: Text;
  Ch: Char;

const
  InFName = 'stack.txt';
  OutFName = 'stack.new';

begin
```



```
  S.Init;
  WriteLn('Считывание стека из файла.');
  Assign(F, InFName);
  Reset(F);
  while not Eof(F) do
  begin
    Read(F, Ch);
    S.Push(Ch);
  end;
  Close(F);
  WriteLn('Исходный стек:');
  WriteStack(S, Output);
  WriteLn('Удаление элемента ', S.GetTop, ' из стека.');
  Ch := S.Pop;
  WriteLn('Удаление элемента ', S.GetTop, ' из стека.');
  Ch := S.Pop;
  WriteLn('Удаление элемента ', S.GetTop, ' из стека.');
  Ch := S.Pop;
  WriteLn('Удаление элемента ', S.GetTop, ' из стека.');
  Ch := S.Pop;
  WriteLn('Добавление элемента "ь" в стек.'); S.Push('ь');
  WriteLn('Добавление элемента "т" в стек.'); S.Push('т');
  WriteLn('Добавление элемента "и" в стек.'); S.Push('и');
  WriteLn('Добавление элемента "П" в стек.'); S.Push('П');
  WriteLn('Модифицированный стек:');
  WriteStack(S, Output);
  WriteLn('Запись стека в файл.');
  Assign(F, OutFName);
  Rewrite(F);
  WriteStack(S, F);
  Close(F);
  WriteLn('Удаление стека из памяти.');
  S.Done;
end.
```

5.2. Тестовая программа для прикладного модуля

```
{ TSTPOL.PAS
  ----------------------
  Дата      17/08/95
  Для       Turbo Pascal 6.0
  (c)оздал Иванов П.Ф.
  ---------------------------------------------------------------
```

```
  Тестовая программа для класса "Польская запись". }

uses Polish;

var
  P: Polish;
  F: Text;
  Infix, Postfix: String;

const
  InFName = 'infix.txt';
  OutFName = 'result.txt';

begin
  Assign(F, InFName);
  Reset(F);
  ReadLn(F, Infix);
  Close(F);
  P.Init;
  Postfix := P.Infix2Postfix(Infix);
  P.Done;
  Assign(F, OutFName);
  Rewrite(F);
  WriteLn(F, 'Исходное выражение:');
  WriteLn(F, Infix);
  WriteLn(F, 'Польская запись:');
  WriteLn(F, Postfix);
  Close(F);
end.
```

7. Задания по вычислительной практике

Задание состоит из двух частей. В первой части задания нужно реализовать структуру данных (стек, связанный список или двоичное дерево) в виде класса. Во второй части задания требуется запрограммировать решение конкретной задачи, используя класс-структуру данных из первой части.

Студенту гарантирована оценка "удовлетворительно", если он выполнил первую часть задания согласно требованиям (см.: 5. Критерии итоговой оценки, с. 5).

Задания по структуре данных стек имеют индекс S (S1, S2, ..., S5), связанный список - L (L1, L2), двоичное дерево - T (T1, T2, ... T6).

7.1. Задания с индексом S

7.1.1. Первая часть заданий S1, S2, S3

Написать модуль, реализующий структуру данных "Стек (эле типа string)" в виде класса. В качестве методов класса реали следующие операции над стеком: инициализация стека, уничтожение добавление элемента, удаление элемента, взятие верхнего элеме проверка стека на пустоту.

Написать тестовую программу, иллюстрирующую использова этого модуля:

- Создать текстовый файл ***stack.txt*** (с помощью любого редактора тексто в котором построчно (по одному слову в строке) записать фразу ***!пруда из рыбку и вынешь не труда Без***
- Сформировать стек из слов этой фразы, прочитав ее из файла.
- Выдать стек на экран (в строку).
- Удалить из стека 4 слова.
- Добавить 4 произвольных слова в стек.
- Выдать модифицированный стек на экран (в строку).
- Записать модифицированный стек в текстовый файл ***stack.new***.
- Удалить стек из памяти.
- Перед началом очередного действия выдавать на экран диагностическое сообщение:
  Считывание стека из файла, Исходный стек, Удаление элемента ... из стека, Добавление элемента ... в стек, Модифицированный стек, Запись стека в файл, Удаление стека из памяти.

7.1.2. Первая часть заданий S4, S5

Написать модуль, реализующий структуру данных "Стек (элементов типа char)" в виде класса. В качестве методов класса реализовать следующие операции над стеком: инициализация стека, уничтожение стека, добавление элемента, удаление элемента, взятие верхнего элемента, проверка стека на пустоту.

Написать тестовую программу, иллюстрирующую использование этого модуля:

- Создать текстовый файл ***stack.txt*** (с помощью любого редактора текстов), состоящий из одной строки символов:
  .сорпов меч в тоВ ?ытыб ен или ытыБ
- Сформировать стек из символов этой строки, прочитав ее из файла.
- Выдать стек на экран (в строку).
- Удалить из стека 4 символа.
- Добавить 4 произвольных символа в стек.
- Выдать модифицированный стек на экран (в строку).

- Записать модифицированный стек в текстовый файл ***stack.new***.
- Удалить стек из памяти.
- Перед началом очередного действия выдавать на экран диагностическое сообщение:

  Считывание стека из файла, Исходный стек, Добавление элемента ... в стек, Удаление элемента ... из стека, Модифицированный стек, Запись стека в файл, Удаление стека из памяти.

7.1.3. S1: Вычисление значения выражения в польской записи

Вторая часть.

Под выражением понимается конструкция следующего вида:

<выражение> ::= <терм> | <терм> + <выражение>
<терм> ::= <множитель> * <терм>
<множитель> ::= <число> | <выражение> | <множитель> ^ <число>[1]
<число> ::= 0 | 1 | 2 | ... | 9.

Польской записью выражения ***A [операция] B*** называется запись, в которой знак операции размещен за операндами: ***A B [операция]***. Например:

Обычная запись	Польская запись	
a-b	ab-	
a*b+c	ab*c+	т.е. (ab*)c+
a*(b+c)	abc+*	т.е. a(c+b)*

Имеется текстовый файл ***postfix.txt***, состоящий из одной строки, в которой записано выражение в польской записи. Прочитать выражение из файла и вычислить значение выражения (как целое число). Результат записать в текстовый файл ***result.txt*** в следующем виде:

Выражение в польской записи: ***<исходное выражение>*** Результат: ***<результат>***

Считать, что исходное выражение корректно.

Указания.

1. Использовать следующий алгоритм вычисления:
   Выражение просматривается слева направо. Если встречается операнд (число), то его значение (как целое) заносится в стек. Если встречается знак операции, то из стека извлекаются два последних элемента (это операнды данной операции) и над ними выполняется операция, а ее результат записывается в стек. В конце концов в стеке останется только одно число - значение всего выражения.

---

[1] Запись "x ^ y" означает возведение x в степень y.

2. Породить от класса "Стек" класс "Польская Запись", добавив метод "Вычисление Выражения В Польской Записи".

7.1.4. S2: Простейший вычислитель

Вторая часть.

Под выражением понимается конструкция следующего вида:

<выражение> ::= <число> | <знак операции> (<выражение>, <выражение>)
<знак операции> ::= m | M | s | p | e | v | o
<число> ::= 0 | 1 | 2 | ... | 9,

где

M(a,b) вычисляет максимум из a и b,
m(a,b) вычисляет минимум из a и b,
s(a,b) вычисляет сумму a и b,
p(a,b) вычисляет произведение a и b,
e(a,b) вычисляет a в степени b,
v(a,b) вычисляет результат от деления нацело a на b,
o(a,b) вычисляет остаток от деления нацело a на b.

Например: M(5,m(3,4)) = 5,
e(s(0,2),v(3,p(2,2))) = 1.

Имеется текстовый файл ***example.txt***, состоящий из одной строки, в которой записано выражение. Прочитать выражение из файла и вычислить значение выражения (как целое число). Результат записать в текстовый файл ***result.txt*** в следующем виде:

Исходное выражение: ***<исходное выражение>***. Результат: ***<результат>***.

Считать, что исходное выражение корректно.

Указания.

1. Использовать следующий алгоритм вычисления:
   Выражение просматривается слева направо. Если встречается число или знак операции, то оно заносится в стек. Если встречается закрывающая скобка, то из стека извлекаются три последних элемента (числа-операнды операции и знак операции) над ними выполняется операция, и ее результат записывается в стек. В конце концов в стеке останется только одно число - значение всего выражения.
2. Породить от класса "Стек" класс "Вычислитель", добавив метод "Вычисление Выражения".

7.1.5. S3: Компилятор с языка AGA

Вторая часть.

Язык программирования AGA имеет алфавит, идентичный алфавиту языка Pascal. Прописные и строчные буквы в идентификаторах и зарезервированных словах этого языка не различаются (например, идентификаторы 'Stack', 'STACK', 'stack' не различаются). Подпрограммы (процедуры и функции) языка AGA имеют следующий формат:

```
proc | func <имя подпрограммы>
    <операторы>
finish <имя подпрограммы>
```

Например:

a) **proc PRACT**
   <операторы процедуры PRACT>
   finish PRACT

b) **func STUDY**
   <операторы функции STUDY>
   finish STUDY

Подпрограммы могут быть вложенными. Например:

a) **proc AnyPROC**
   func AnyFUNC
   <операторы функции AnyFUNC>
   finish AnyFUNC
   <операторы процедуры AnyPROC>
   finish AnyPROC

b) **proc Main**
   func Second
   func SubFUNC
   <операторы функции SubFUNC>
   finish SubFUNC
   <операторы функции Second>
   finish Second
   <операторы процедуры Main>
   finish Main

Имеется текстовый файл ***program.aga***, в котором записана программа на языке AGA. Проверить эту программу на выполнение следующих условий:

1) Для каждой декларации **proc *<имя подпрограммы>*** или **func *<имя подпрограммы>*** ниже имеется декларация **finish *<имя подпрограммы>***; и наоборот: для каждой декларации **finish *<имя подпрограммы>*** выше имеется декларация **proc *<имя подпрограммы>*** или **func *<имя подпрограммы>***.

2) Соответствующие пары деклараций **proc *<имя подпрограммы>*** **func *<имя подпрограммы>*** и **finish *<имя подпрограммы>*** правильно вложены друг в друга. Например:

Правильное вложение	Неправильное вложение
proc AnyPROC **func AnyFUNC** ***<операторы>*** **finish AnyFUNC** ***<операторы>*** **finish AnyPROC**	**proc AnyPROC** **func AnyFUNC** ***<операторы>*** **finish AnyPROC** ***<операторы>*** **finish AnyFUNC**

Результат проверки записать в текстовый файл ***result.txt*** в следующем виде:

Если в программе нет ошибок	Если в программе найдена ошибка
Проверяемый файл: PROGRAM.AGA ОШИБОК НЕТ.	Проверяемый файл: PROGRAM.AGA ОШИБКА В СТРОКЕ ***<номер строки с первой найденной ошибкой>***

Считать, что файл ***program.aga*** существует и программа в нем корректна в следующем смысле:

1) каждый оператор, в т.ч. декларации ***proc***, ***func***, ***finish*** занимает одну строку;
2) декларации ***proc***, ***func***, ***finish*** имеют нулевой отступ от левого края строки;
3) имя подпрограммы начинается через один пробел после деклараций ***proc***, ***func***, ***finish***;
4) нет подпрограмм с одинаковыми именами;
5) имя подпрограммы - корректный идентификатор.

Указания.

1. Обрабатывать каждую строку программы на языке AGA, считывая ее из файла и преобразовывая к верхнему регистру. Для этого написать подпрограмму

```
function StrUpcase(S: String): String;
  { возвращает строку символов, преобразованную к
  верхнему регистру },
```

   в которой использовать стандартную функцию работы с символами Upcase.
2. Проверять наличие деклараций ***proc***, ***func*** или ***finish*** с помощью стандартной функции работы со строками Pos.
3. Для получения имени подпрограммы из строки декларации использовать стандартную функцию работы со строками Copy.

4. При разработке алгоритма учесть, что имя подпрограммы не может быть пустым и совпадать с зарезервированными словами *'func'*, *'proc'*, *'finish'*.

7.1.6. S4: Перевод выражения из обычной формы записи в польскую

Вторая часть.

Под выражением понимается конструкция следующего вида:

<выражение> ::= <терм> | <терм> <знак> <выражение>
<знак> ::= + | -
<терм> ::= <множитель> * <терм>
<множитель> ::= <число> | <переменная> | <выражение>
<число> ::= 0 | 1 | 2 | ... | 9
<переменная> ::= A | B | C | ... | Z.

Польской записью выражения ***A [операция] B*** называется запись, в которой знак операции размещен за операндами: ***A B [операция]***. Например:

Обычная запись	Польская запись	
a-b	ab-	
a*b+c	ab*c+	т.е. (ab*)c+
a*(b+c)	abc+*	т.е. a(c+b)*

Имеется текстовый файл ***infix.txt***, в котором в одну строку записано выражение. Прочитать его из файла и перевести в польскую запись. Результат перевода записать в текстовый файл ***result.txt*** в следующем виде:

Исходное выражение:
<исходное выражение>.
Польская запись:
<польская запись выражения>.

Считать, что исходное выражение корректно.

Указания.

1. Использовать следующий алгоритм перевода:
   В стек записывается открывающая скобка, и выражение просматривается слева направо. Если встречается операнд (число или переменная), то сразу добавить его справа к выходной строке. Если встречается открывающая скобка, то она заносится в стек. Если встречается закрывающая скобка, то из стека извлекаются находящиеся там знаки операций до ближайшей открывающей скобки, которая также удаляется из стека, и все эти знаки в порядке их извлечения добавляются справа к выходной строке. Если встречается знак операции, то из конца стека извлекаются (до ближайшей скобки, которая сохраняется в стеке) знаки операций, старшинство которых больше или равно старшинству да операции, и они добавляются справа к выходной строке, после ч рассматриваемый знак заносится в стек. В заключение выполня те же действия, как если бы встретилась закрывающая скобка.
2. Породить от класса "Стек" класс "Польская Запись", добавив ме "Перевод В Польскую Запись".

7.1.7. S5: Баланс скобок

Вторая часть.

Заданы следующие определения:

<открывающая скобка> ::= (| { | [| < ;
<закрывающая скобка> ::=) | } |] | > ;
<соответствующие скобки> ::= (и) | { и } | [и] | < и > .

Имеется текстовый файл ***expr.txt***, состоящий из одной строки символов, в которой записан некий текст. Проверить, соблюдается ли в этом тексте требования баланса скобок:

1) Для каждой открывающей скобки слева имеется закрывающая скобка справа; и наоборот: для каждой закрывающей скобки справа имеется открывающая скобка слева.
2) Соответствующие пары скобок правильно вложены друг в друга. Например:

Правильное вложение скобок в тексте	Неправильное вложение скобок в тексте
(abc)	(abc>
(89[5])	(89[5)]
(2{[abcd<OK>efgh}]2)	(2{[abcd<OK)efgh}]2>

Результат проверки записать в текстовый файл ***result.txt*** в следующем виде:

Если есть баланс скобок	Если нет баланса скобок
Проверяемый текст: ***<проверяемый текст>*** БАЛАНС СКОБОК ЕСТЬ.	Проверяемый текст: ***<проверяемый текст>*** БАЛАНСА СКОБОК НЕТ. ОШИБКА В СИМВОЛЕ ***<номер символа с первой найденной ошибкой>***

Указание.

Породить от класса "Стек" класс "Контролер", добавив метод "Проверка Баланса Скобок".

7.2. Задания с индексом L

7.2.1. L1: Считалка

Первая часть.

Написать модуль, реализующий структуру данных "Двунаправленный список (элементов типа string)" в виде класса. В качестве методов класса реализовать следующие операции над двунаправленным списком: инициализация списка, уничтожение списка, добавление элемента, удаление элемента, обход списка в "прямом порядке", обход списка в "обратном порядке".

Написать тестовую программу, иллюстрирующую использование этого модуля:

- Создать текстовый файл ***ring.txt*** (с помощью любого редактора текстов), где построчно (по одному слову в строке) записать фразу:
  А РОЗА УПАЛА НА ЛАПУ АЗОРА
- Сформировать двунаправленный список из слов этой фразы, прочитав ее из файла.
- Выдать список на экран (в строку, без пробелов между словами) в "прямом порядке".
- Выдать список на экран (в строку, без пробелов между словами) в "обратном порядке".
- Удалить из списка 2 любых слова.
- Вставить на их место 2 любых (непустых) слова.
- Выдать модифицированный список на экран (в строку, без пробелов между словами) в "прямом порядке".
- Выдать модифицированный список на экран (в строку, без пробелов между словами) в "обратном порядке".
- Записать модифицированный список в текстовый файл ***ring.new*** в первой строке - в "прямом порядке", во второй строке - в "обратном порядке".
- Удалить список из памяти.
- Перед началом очередного действия выдавать на экран диагностическое сообщение:
  Считывание списка из файла. Исходный список в прямом порядке, Исходный список в обратном порядке, Удаление элемента ... из списка, Добавление элемента ... в список, Модифицированный список в прямом порядке, Модифицированный список в обратном порядке, Запись списка в файл, Удаление списка из памяти.

Вторая часть.

Игра "Считалка" предполагает следующее: n человек (n - натуральное) располагаются по кругу. Начав отсчет от первого (слева направо), удаляют из круга каждого k-го (k - натуральное), смыкая круг после каждого удаления.

Имеется текстовый файл ***kids.txt***, в первой строке которого записано натуральное число слов в считалке, а в последующих - строки символов - имена игроков в круге. Например:


```
3
Лена
Света
Коля
Дима
```

Требуется определить порядок удаления игроков из круга и записать результат в текстовый файл ***result.txt*** в следующем виде:

```
Список игроков в круге:
<имена игроков в круге построчно>
Число слов в считалке:
<число слов в считалке>
Порядок удаления игроков из круга:
<имена игроков в порядке удаления из круга построчно>
```

Считать, что в файле ***kids.txt*** не менее двух строк. В случае, если в первой строке файла ***kids.txt*** записано целое отрицательное число или 0, файл ***result.txt*** должен иметь вид:

```
Список игроков в круге:
<имена игроков в круге построчно>
Число слов в считалке:
<число слов в считалке>
ОШИБКА! НЕКОРРЕКТНЫЕ ВХОДНЫЕ ДАННЫЕ.
```

Указание.

Породить от класса "Двунаправленный список" класс "Считалка", добавив метод "Протокол Выбывания".

7.2.2. L2: Сортировка списка методом вставок

Первая часть.

Написать модуль, реализующий структуру данных "Связанный список (элементов типа integer)" в виде класса. В качестве методов класса реализовать следующие операции над связанным списком: инициализация списка, уничтожение списка, добавление элемента, удаление элемента, обход списка.

Написать тестовую программу, иллюстрирующую использование этого модуля:

- Создать текстовый файл ***list.txt*** (с помощью любого редактора текстов), в каждой строке которого записано одно целое число; в файле должно быть более четырех строк.
- Сформировать связанный список из этих чисел, прочитав их из файла.
- Выдать список на экран (в строку, через пробел).
- Подсчитать среднее арифметическое элементов списка и выдать ответ на экран.

- Удалить последний элемент списка.
- Удалить элемент из середины списка.
- Добавить элемент в начало списка.
- Добавить элемент в конец списка.
- Добавить элемент в середину списка.
- Выдать модифицированный список на экран (в строку, через пробел).
- Записать модифицированный список в файл ***list.new***.
- Удалить список из памяти.
- Перед началом очередного действия выдавать на экран диагностическое сообщение:
  Считывание списка из файла, Исходный список, Удаление элемента ... из списка, Добавление элемента ... в список, Модифицированный список, Запись списка в файл, Удаление списка из памяти.

Вторая часть.

Имеется текстовый файл ***numbers.txt***, в каждой строке которого записано целое число. Требуется прочитать эти числа из файла, сформировать из них связанный список и отсортировать его по убыванию, используя метод вставок (описание метода см., например, в [2, с. 81-83] - "Сортировка простым выбором").

Результат записать в текстовый файл ***result.txt*** в следующем виде:

Исходный список:
<исходный список (в строку через пробел)>
(***<количество элементов в списке>*** элементов)
Отсортированный по убыванию список:
<отсортированный список (в строку через пробел)>
Время работы:
<время сортировки, часов:минут:секунд:сотыхсекунд>

Указания.

1. Породить от класса "Связанный Список" класс "Специальный Список", добавив метод "Сортировка".
2. Написать программу, которая создает файл ***numbers.txt*** с тестовым набором чисел для сортировки:
   a) 256 чисел, отсортированных по убыванию;
   b) 256 случайных чисел;
   c) 256 чисел, отсортированных по возрастанию;
   d) то же, что в a) - c), для 512 чисел.
   Таким образом, тестовый набор должен состоять из шести примеров.
3. Для получения случайных чисел использовать стандартные подпрограммы Randomize и Random.
4. Для вычисления времени сортировки использовать стандартные подпрограммы GetTime и SetTime.

7.3. Задания с индексом Т

7.3.1. Первая часть заданий Т1, Т2, Т3

Написать модуль, реализующий структуру данных "Двоичное дерево (элементов типа string)" в виде класса. В качестве методов класса реализовать следующие операции над двоичным деревом: инициализация дерева, уничтожение дерева, добавление элемента, удаление элемента, обход дерева в порядке "корень→левая ветвь→правая ветвь", обход дерева в порядке "левая ветвь→правая ветвь→корень", обход дерева в порядке "левая ветвь→корень→правая ветвь".

Написать тестовую программу, иллюстрирующую использование этого модуля:

- Сформировать двоичное дерево, изображенное на рисунке:

- Реализовать обход сформированного дерева в порядке "корень→левая ветвь→правая ветвь". Результат обхода выдать на экран (в строку, через пробел) и записать в текстовый файл ***tree.new*** в виде:

Обход дерева в порядке "корень ->левая ветвь ->правая ветвь"
<фраза, полученная в результате обхода>

- Реализовать обход сформированного дерева в порядке "левая ветвь→правая ветвь→корень". Результат обхода выдать на экран (в строку, через пробел) и записать в текстовый файл ***tree.new*** в виде:

Обход дерева в порядке "левая ветвь ->правая ветвь ->корень"
<фраза, полученная в результате обхода>

Реализовать обход сформированного дерева в порядке "левая ветвь→ корень→правая ветвь". Результат обхода выдать на экран (в строку, через пробел) и записать в текстовый файл ***tree.new*** в виде:

Обход дерева в порядке "левая ветвь ->корень ->правая ветвь" ***<фраза, полученная в результате обхода>***

Удалить дерево из памяти.

Перед началом очередного действия выдавать на экран диагностическое сообщение:

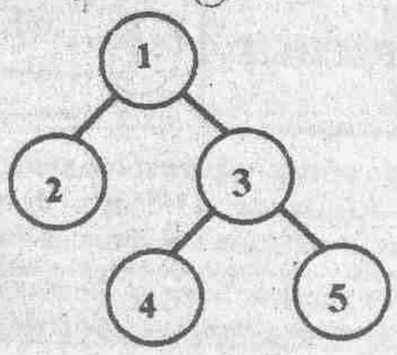
Формирование дерева, Обход дерева в порядке "корень->левая ветвь->правая ветвь", Обход дерева в порядке "левая ветвь->правая ветвь->корень", Обход дерева в порядке "левая ветвь->корень->правая ветвь", Удаление дерева из памяти.

7.3.2. Первая часть заданий Т4, Т5, Т6

Написать модуль, реализующий структуру данных "Двоичное дерево (элементов типа integer)" в виде класса. В качестве методов класса реализовать следующие операции над двоичным деревом: инициализация дерева, уничтожение дерева, добавление элемента, удаление элемента, обход дерева в порядке "корень→левая ветвь→правая ветвь", обход дерева в порядке "левая ветвь→правая ветвь→корень", обход дерева в порядке "левая ветвь→корень→правая ветвь".

Написать тестовую программу, иллюстрирующую использование этого модуля:

Сформировать двоичное дерево, изображенное на рисунке:



Реализовать обход сформированного дерева в порядке "корень→ левая ветвь→правая ветвь". Результат обхода выдать на экран (в строку, через пробел) и записать в текстовый файл ***tree.new*** в виде:

Обход дерева в порядке "корень ->левая ветвь ->правая ветвь" ***<фраза, полученная в результате обхода>***

- Реализовать обход сформированного дерева в порядке "левая ветвь→ правая ветвь→корень". Результат обхода выдать на экран (в строку, через пробел) и записать в текстовый файл ***tree.new*** в виде:

Обход дерева в порядке "левая ветвь ->правая ветвь ->корень" ***<фраза, полученная в результате обхода>***

- Реализовать обход сформированного дерева в порядке "левая ветвь→ корень →правая ветвь". Результат обхода выдать на экран (в строку, через пробел) и записать в текстовый файл ***tree.new*** в виде:

Обход дерева в порядке "левая ветвь ->корень ->правая ветвь " ***<фраза, полученная в результате обхода>***

- Удалить дерево из памяти.
- Перед началом очередного действия выдавать на экран диагностическое сообщение:

Формирование дерева, Обход дерева в порядке "корень->левая ветвь->правая ветвь", Обход дерева в порядке "левая ветвь->правая ветвь->корень", Обход дерева в порядке "левая ветвь->корень->правая ветвь", Удаление дерева из памяти.

7.3.3. Т1: Интерактивное построение двоичного дерева

Вторая часть.

Требуется в режиме "вопрос машины - ответ пользователя" построить двоичное дерево.

Первый вопрос: ***Введите элемент-корень =>_***, пользователь должен ввести строку-корень дерева. Затем выдается запрос: ***Будете вводить элементы следующего уровня? (Y/N)***. При положительном ответе выдается сообщение: ***Уровень n_1, ввод n_2 вершин*** (где n_1 нужно заменить на номер уровня, а n_2 — на число вершин на этом уровне), а затем n_2 раз запрос: ***Введите ...-ю вершину*** и считывание ввода пользователя. Если строка оканчивается символом #, то данную вершину дерева считать листом. Процесс ввода дерева прекращается, если пользователь не захочет вводить элементы следующего уровня либо все введенные элементы уровня оказались листьями.

Результат записать в текстовый файл ***result.txt*** в следующем виде:

Построенное дерево (обход "левая ветвь->корень->правая ветвь"): ***<распечатка построенного дерева в порядке "левая ветвь→корень→правая ветвь">*** Чйсло уровней в дереве: ***<число уровней в дереве>***

7.3.4. Т2: Игра "Угадай слово"

Вторая часть.

Описание игры.

Понятие - слово (существительное) или выражение, обозначающее одушевленный или неодушевленный предмет. Например: ***дискета, вирус СПИД.***

Признак понятия - слово (прилагательное) или выражение, обозначающее свойство, которое характеризует понятие. Например: ***компьютерное, очень опасное*** (соответственно для понятий ***дискета, вирус СПИД***).

База понятий и признаков - двоичное дерево, элементами которого являются понятия и признаки. Понятия - листья этого дерева. Признаки какого-либо понятия располагаются в узлах дерева по правилу: если у понятия присутствует этот признак, то он должен быть в левой ветви дерева, иначе - в правой. Корень дерева - признак произвольного понятия. Например:

Играют пользователь и машина. Берется начальная база понятий и признаков, состоящая из двух понятий и одного признака. Пользователь задумывает понятие. Машина, используя (пополняющуюся) базу понятий и признаков, пытается отгадать это понятие.

Порядок игры следующий:

1) Выдается сообщение:
   Задумайте слово. Для начала игры нажмите любую клавишу.
2) Берется корень дерева и выдается запрос:
   Оно <признак понятия-корень>? (Y/N)
3) В зависимости от положительного или отрицательного ответа пользователя берется признак-корень левого или правого поддерева соответственно и выдается запрос:
   Оно <признак понятия>? (Y/N).
4) Когда, продолжая аналогичным образом, машина доходит до листа дерева, выдается сообщение:
   Знаю, знаю! Это <понятие-лист>. Я угадала? (Y/N).
5) При положительном ответе пользователя выдается запрос: ***Играем еще? (Y/N)*** и в зависимости от ответа пользователя осуществляется переход к 1) (продолжение игры) либо выдается сообщение: ***Приятно было с Вами поиграть.***, и игра заканчивается.
6) При отрицательном ответе выдается сообщение:

Вот как, а что же это? Введите новое понятие =>_ ,
и пользователь вводит новое понятие. Далее выдается сообщение:
А как отличить, где <понятие-лист>, а где <новое понятие>?
Введите признак =>_ ,
и пользователь вводит признак нового понятия. Затем выдается запрос:
Значит, если <признак нового понятия>, то <новое понятие>? (Y/N),
и в зависимости от ответа пользователя новое понятие и его признак включается в базу понятий и признаков (в левую или правую ветвь).

Задание.

Написать программу, реализующую игру "Угадай слово". В качестве начальной базы понятий и признаков взять дерево:

Пополняющуюся базу понятий и признаков хранить в файле ***gametree.dat***. Обеспечить ввод Y/N-ответов пользователя путем нажатия клавиши Y либо N. Обеспечить запись протокола игры в текстовый файл ***protocol.gam*** согласно следующему примеру:

```
База понятий и признаков (левый обход дерева):
(лошадь) (живое) (вирус СПИД)
-------------------------------------------------------------
Задумайте слово. Для начала игры нажмите любую клавишу.
Оно живое? (Y/N) Y
Знаю, знаю! Это лошадь. Я угадала? (Y/N) N
Вот как, а что же это? Введите новое понятие =>_ таракан
А как отличить, где лошадь, а где таракан? Введите признак =>_ с усами
Значит, если с усами, то таракан? (Y/N) Y
Играем еще? (Y/N) Y
-------------------------------------------------------------
База понятий и признаков (левый обход дерева):
(таракан) (с усами) (лошадь) (живое) (вирус СПИД)
-------------------------------------------------------------
Задумайте слово. Для начала игры нажмите любую клавишу.
Оно живое? (Y/N) N
Знаю, знаю! Это вирус СПИД. Я угадала? (Y/N) N
Вот как, а что же это? Введите новое понятие =>_ ОРЗ
А как отличить, где СПИД, а где ОРЗ? Введите признак =>_ очень опасное
Значит, если очень опасное, то ОРЗ? (Y/N) N
Играем еще? (Y/N) N
```

Приятно было с Вами поиграть.

База понятий и признаков (левый обход дерева):
(таракан) (с усами) (лошадь) (живое) (вирус СПИД) (очень опасное) (ОРЗ)

Указания.

1. Породить от класса "Двоичное Дерево" класс "Игровое Дерево", добавив методы "Считывание Дерева Из Файла" и "Запись Дерева В Файл" и поле "Файл Для Хранения Дерева".
2. Подключить к головной программе модули с реализацией классов "Двоичное Дерево" и "Игровое Дерево".
3. Самостоятельно разработать формат хранения двоичного дерева в файле, учтя следующие замечания:
   - При построении дерева путем считывания из файла легче различать его элементы как листья и не-листья. Для этого в начало строки-элемента дерева можно добавлять (сделав его невидимым для пользователя) символ-индикатор. Например, символ chr(255)='_' обозначает не-лист, его отсутствие означает лист, и элементы игрового дерева

будут храниться как *'_очень опасное'*, *'вирус СПИД'*, *'ОРЗ'*.

   - Считывание и запись дерева в файл легче реализовать, используя рекурсию.
4. Распечатки файла ***protocol.gam*** приложить к отчету по вычислительной практике в качестве тестовых прогонов модуля с реализацией класса "Игровое Дерево" и головной программы.

7.3.5. Т3: Вычисление значения выражения с помощью двоичного дерева

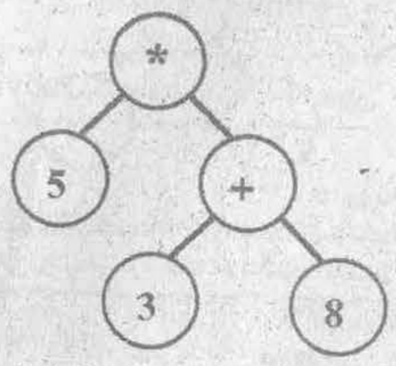
Вторая часть.

Под выражением понимается конструкция следующего вида:

<выражение> ::= <число> | (<выражение> <знак> <выражение>)
<знак> ::= + | - | *
<число> ::= 0 | 1 | 2 | ... | 9.

Такое выражение можно представить в виде двоичного дерева по правилам: выражение из одного числа представляется деревом из одной вершины с этим числом, а выражение вида (f_1 s f_2) - деревом, в котором корень - это знак s, а левое и правое поддеревья - это соответствующие

представления выражений f_1 и f_2. Например, выражению 5*(3+8) соответствует дерево:



Имеется текстовый файл ***formule.txt***, состоящий из одной строки символов, в которой записано выражение. Прочитать выражение из файла и, построив соответствующее двоичное дерево, вычислить значение выражения (как целое число). Результат записать в текстовый файл ***result.txt*** в виде:

Исходное выражение:
<исходное выражение>
Результат:
<результат>

Считать, что исходное выражение корректно.

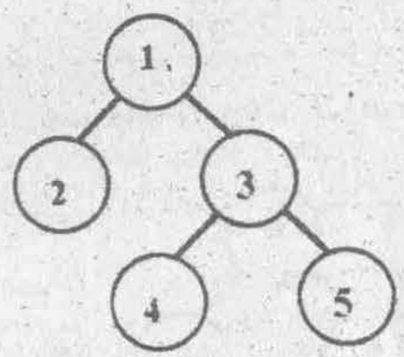
Указания.

1. Породить от класса "Двоичное дерево" класс "Калькулятор", добавив метод "Вычисление Значения Выражения" (который по аргументу типа String - выражение - строит дерево и вычисляет значение выражения как целое).
2. При вычислении значения выражения легче различать элементы дерева как знаки математических операций и константы. Для этого в начало строки-элемента дерева можно добавлять (не учитывая при вычислениях) символ-индикатор. Например, символ chr(255) = '_' обозначает знак операции, а его отсутствие означает константу.
3. Значение выражения вычисляется обходом дерева в порядке "левая ветвь→правая ветвь→корень", попутно модифицируя исходное дерево.

7.3.6. Т4: Глубина двоичного дерева

Вторая часть.

Максимальная (минимальная) глубина двоичного дерева - это число ветвей в самом длинном (в самом коротком) из путей от корня дерева до листьев. Например, для дерева



max глубина = 2, min глубина = 1.

Требуется найти max и min глубину заданного двоичного дерева и записать результат в текстовый файл ***result.txt*** в следующем виде:

Исходное дерево (обход в порядке "левая ветвь->корень->правая ветвь"): ***<распечатка исходного дерева в порядке "левая ветвь→корень→правая ветвь">*** max глубина = ***<max глубина>*** min глубина = ***<min глубина>***

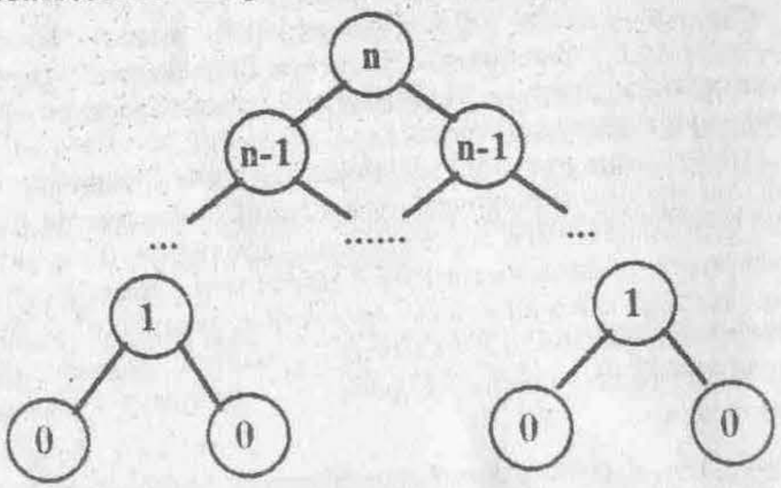
Указание.

Породить от класса "Двоичное Дерево" класс "Особое Дерево", добавив методы "Поиск Max Глубины", "Поиск Min Глубины".

7.3.7. Т5: Построение двоичного дерева

Вторая часть.

Имеется текстовый файл ***levels.txt***, состоящий из одной строки, в которой записано целое число ***n*** из отрезка ***[0;10]***. Требуется по этому числу построить двоичное дерево вида



Записать результат в текстовый файл ***result.txt*** в следующем виде:

Число уровней в дереве: ***<n+1>*** Построенн ∍ дерево (обход в порядке "левая ветвь->корень->правая ветвь"): ***<распечатка построенного дерева в порядке "левая ветвь→корень→правая ветвь">***

Если n ∉ [0;10], то файл ***result.txt*** должен иметь вид

Число уровней в дереве: ***<n+1>*** Некорректные входные данные!

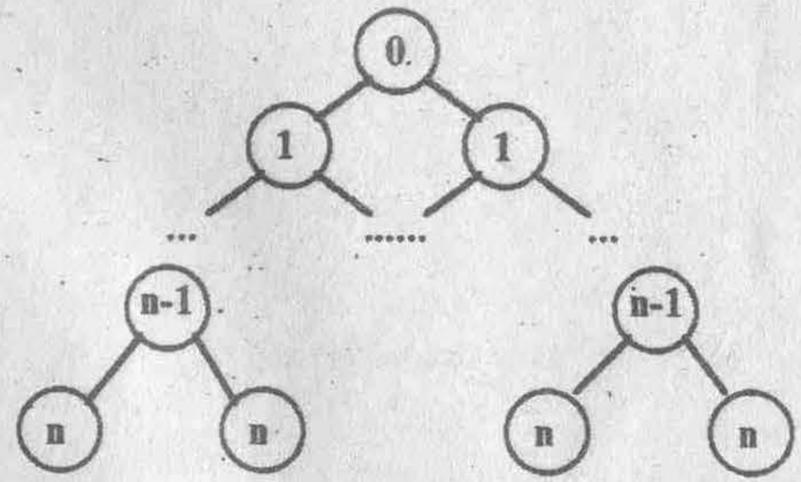
Указание.

Породить от класса "Двоичное Дерево" класс "Особое Дерево", добавив метод "Построение Дерева От N".

7.3.8. Т6: Построение двоичного дерева

Вторая часть.

Имеется текстовый файл ***levels.txt***, состоящий из одной строки, в которой записано целое число ***n*** из отрезка ***[0;10]***. Требуется по этому числу построить двоичное дерево вида



Записать результат в текстовый файл ***result.txt*** в следующем виде:

Число уровней в дереве: ***<n+1>*** Построенное дерево (обход в порядке "левая ветвь->корень->правая ветвь"): ***<распечатка построенного дерева в порядке "левая ветвь→корень→правая ветвь">***

Если n ∉ [0;10], то файл ***result.txt*** должен иметь вид