```
1       Program obj2tree;

2       {
          Задача.  Построить двоичное дерево, элементами которого являются
                   слова.
                   Слова, которые уже помещены в дерево, повторно не
                   включать.
                   Определить процедуры и работы с деревом:
                     - поиск слова в дереве;
                     - добавление к дереву нового узла;
                     - обход дерева (сортировка дерева);
                     - уничтожение дерева.

                   Определить object-дерево.

          Формат.  Сергей Митрофанов,
                   Центр НИТ "Северная Звезда",

                   11.11.03, 09:40-14:25
                   29.05.05,
                   17.02.06, 12:30-16:08
                   30.11.06, 17:49
        }
23
24      {
          Теория
          ------
          Двоичное дерево схематично можно определить следующим образом:
          имеется набор вершин, соединенных стрелками.

          Из каждой вершины выходит не более двух стрелок (ветвей),
          направленных влево-вниз или вправо-вниз. Должна существовать
          единственная вершина, в которую не входят ни одна стрелка -
          эта вершина называется КОРНЕМ дерева.

          В каждую из оставшихся вершин входит в точности одна стрелка.

          Каждое звено (вершина дерева) будет записью из 3 полей.
          1 поле - ключ записи (их последовательность записана в массив A)
          2 поле - адрес на вершину влево-вниз
          3 поле - адрес на вершину вправо-вниз

          Способ построения дерева:
          Если поступает очередная запись с ключом k, то - начиная с корня
          дерева - в зависимости от сравнения ключа k с ключом в очередной
          вершине мдем влево или вправо от нее, до тех пор, пока не найдем
          подходящую вершину, к которой можно присоединить новую вершину с
          ключом k. В зависимости от результата сравнения ключа в этой
          вершине с поступившим ключом k делаем вновь сформированную
          вершину левой или правой для найденной вершины.
```

```
        }
51
52      Uses Crt;
53
54      Type
55        keys = string;
56
57      Const
58        N = 12;
59
60        A  { данный массив строк, константа }
61        : array [1..N] of string [20]
62        = ('line', 'comma', 'array', 'name',
63          'type', 'string', 'union', 'size', 'case',
64          'line', 'comma', 'array');
65
66        {
          A { данный массив-константа }
68        {: array [1..N] of real
          = (-6, 5, -8, 10, -12, 4, -2, -11, -3, 1, -9, 7);
          }
71
72      Type
73        PNode = ^TNode;
74        TNode = record
75                  Data { строка }
76                  : keys; { string; }
77                  Left, { адрес левого узла }
78                  Right { адрес правого узла }
79                  : PNode;
80                end;
81
82        d_tree = object
83                   Root { вершина дерева }
84                   : PNode;
85
86                   Function poisk (X: keys { string }): PNode;
87                   Procedure add_node (X: keys { string });
88                   Procedure traverse_sort (p: PNode);
89                   Procedure del_tree (p: PNode);
90                   Constructor init;
91                   Destructor done;
92                 end;
93
94
95      Var
96        tree { объект дерева }
97        : d_tree;
98
99        i, { номер элемента массива }
100       R { результат Count () }
```

```
101         : integer;
102
103         n_node { какое число ищем в дереве }
104         : keys;
105
106         uzel { узел, который надо найти }
107         : PNode;
108
109
110       Constructor d_tree.init;
111       {
            создает экземпляр объекта-дерева
          }
114       begin
115         Root := nil;
116         WriteLn ('Сработал Constructor');
117         ReadLn;
118       end;
119
120       Destructor d_tree.done;
121       {
            уничтожает дерево
          }
124       begin
125         del_tree (Root);
126         WriteLn ('Сработал Destructor');
127         ReadLn;
128       end;
129
130       Function d_tree.poisk (
131                               X { искомый стринг }
132                               : keys { string }
133                             )
134                             : PNode;
135       {
            поиск адреса узла, в котором ключ равен X
          }
138       var
139         t { текущий адрес узла, искомый адрес }
140         : PNode;
141
142       begin
143         { начнем поиск с корня дерева }
144         t := Root;
145         while (t <> nil)
146               and
147               (t^.Data <> X)
148           do
149             if t^.Data < X
150               then
151                 t := t^.Right
```

```
152             else
153               t := t^.Left;
154         poisk := t;
155       end; { end функции поиска }
156
157
158       Procedure d_tree.add_node (
159                                   X { строка, помещаемая в дерево }
160                                   : keys {string}
161                                  );
162       {
          Построение дерева.
          Включение в дерево нового узла.
          Но если строка X уже в дереве есть, то ничего не делаем,
          новую такую же строку в дерево не добавляем.
        }
168       var
169         t, { адрес текущего узла дерева }
170         Parent { родительский узел }
171         : PNode;
172
173       begin
174         { начнем просмотр дерева с его корня }
175         t := Root;
176
177         {
            Найдем адрес t, в котором стринг равен X,
            а в Parent - адрес родителя этого X.
            Если стринга X еще в дереве нет,
            то в t будет nil
            Если после работы цикла в Parent останется nil,
              то это означает:
                 либо дерева еще нет, оно пустое
                 либо стринг X находится в корне дерева
          }
187         Parent := nil;
188         while (t <> nil)
189               and
190               (t^.Data <> X)
191           do
192             begin
193               Parent := t;
194               if t^.Data < X
195                 then
196                   t := t^.Right
197                 else
198                   t := t^.Left;
199             end;
200
201         {
            if (Parent = nil)
```

```
           and
           (Root <> nil)
         then
           WriteLn (X, ' находится в корне дерева...');

       if t <> nil
         then
           WriteLn (X, ' был обнаружен в дереве');
       }
212
213    {
         если в дереве не был обнаружен стринг X,
              ИЛИ
              дерева еще нет, оно пустое
           то добавляем новый узел
       }
219    if t = nil
220      then
221        begin
222          New (t);
223          t^.Data := X;
224          t^.Left := nil;
225          t^.Right := nil;
226          if Root = nil { если дерево еще пустое }
227            then
228              Root := t
229            {
               известен Parent - родитель,
               куда прикрепим t, к левой связи или правой
             }
233          else
234            if Parent^.Data < X
235              then
236                Parent^.Right := t
237              else
238                Parent^.Left := t;
239        end;
240  end; { end add_node - добавление нового узлав дерево }
241
242
243  Procedure d_tree.traverse_sort (
244                                    p { адрес узла дерева}
245                                    : PNode
246                                  );
247  {
       Обход дерева.
       Необходимо посетить все узлы дерева,
       обойти эти узлы в порядке возрастания стрингов
       Обход по правилу:
       Left - Self - Right
     }
```

```
254       begin
255         if p <> nil
256           then
257             begin
258               {
                    найдем самый нижний левый узел.
                    И напечатаем его стринг
                  }
262               traverse_sort (p^.Left);
263               Write (p^.Data, ' ');
264               {
                    ищем самый нижний правый узел
                    и после окончания рекурсии печатаем стринги
                  }
268               traverse_sort (p^.Right);
269             end;
270       end;
271
272
273       Procedure d_tree.del_tree (
274                                   p { адрес узла дерева }
275                                   : PNode
276                                 );
277       {
            Уничтожение дерева.
          }
280       begin
281         if p <> nil
282           then
283             begin
284               del_tree (p^.Left);
285               del_tree (p^.Right);
286               Dispose (p);
287             end;
288         Root := nil;
289       end;
290
291
292       Function Count_nodes (
293                               p { адрес узла дерева }
294                               : PNode
295                             )
296                             : integer;
297       {
            MSP, 17.02.06, 15:40
            Эта функция возвращает число узлов дерева

            Причем,
              лист - дает ...
              узел с одним nil - дает ...
              узел без nil - дает ...
```

```
        }
306     begin
307       if p = nil
308         then
309           Count_nodes := 0
310         else
311           Count_nodes :=
312           Count_nodes (p^.Left) + 1 + Count_nodes (p^.Right);
313     end;
314
315
316     Begin
317       ClrScr;
318
319       {
            Напечатаем массив слов из которых будут состоять улы дерева
          }
322       WriteLn ('Размер массива: ', N);
323       WriteLn;
324       WriteLn ('Данный массив слов:');
325       WriteLn;
326       for i := 1 to N do
327         Write (A [i], ' ');
328       WriteLn;
329       WriteLn;
330
331       {
            Двоичное дерево еще не построено
            Дерево отсутствует полностью, оно пустое
          }
335       tree.init;
336
337
338       { Создадим дерево }
339       Write ('Создадим дерево, не включая в него снова те слова, ');
340       WriteLn ('которые в нем уже есть.');
341       WriteLn;
342       for i := 1 to N do
343         tree.add_node (A [i]);
344
345       WriteLn ('Отсортированное дерево:');
346       WriteLn;
347       { обход-сортировка дерева, вывод отсортированного списка ключей
          }
349       tree.traverse_sort (tree.Root);
350       WriteLn;
351
352       WriteLn;
353       WriteLn ('Число узлов дерева: ');
354       WriteLn;
355       R := Count_nodes (tree.Root);
```

```
356         WriteLn ('N_nodes = ', R);
357
358         WriteLn;
359         Write ('Какой узел найти: ');
360         ReadLn (n_node);
361
362         uzel := tree.poisk (n_node);
363
364         if (uzel <> nil)
365           then
366             begin
367               WriteLn;
368               WriteLn ('Узел ', n_node, ' в дереве найден.');
369               WriteLn;
370             end
371           else
372             begin
373               WriteLn;
374               WriteLn ('Узел ', n_node, ' в дереве не обнаружен!');
375               WriteLn;
376             end;
377
378         WriteLn ('Уничтожим дерево.');
379         WriteLn;
380         tree.done;
381
382         ReadLn;
383       End.
```

Listing данной задачи опубликован в сети Internet по адресу
http://www.Best-Listing.ru/color-1-task-573.html

Sergey Mitrofanov, 24.08.14, 20:44
E-mail: infostar@mail.ru