Процедура удаления с конца четных элементов

**procedure** delete(pCur:pList; n,counter:integer; **var** pFirst,pNext:pList; **var** fileOut:text);

**begin**

**if** counter = 1 **then**

pFirst:=pCur;

**if** pCur^.next<>nil **then**

delete(pCur^.next, n, counter + 1, pFirst, pNext, fileOut);

**if** (counter **mod** 2) <> (n **mod** 2) **then begin**

pNext:=pCur^.next;

writeln(fileOut, pCur^.data);

dispose(pCur);

**if** counter = 1 **then**

pFirst:=pNext;

**end**

**else**

pCur^.next:=pNext;

**end**;

Процедура удаления с конца элементов двунаправленного списка

**procedure** delete(pCur:pList; counter, n:integer; **var** pFirst:pList; **var** fileOut:text);

**begin**

**if** count = 1 **then**

pFirst:=pCur;

**if** pCur^.next <> nil **then**

delete(pCur^.next, counter + 1, n, pFirst, fileOut);

**if** (counter **mod** 2) = (n **mod** 2) **then begin**

**if** counter = 1 **then**

pFirst:=pCur^.next;

**if** pCur^.next <> nil **then**

pCur^.next^.prev:=pCur^.prev;

**if** pCur^.prev <> nil **then**

pCur^.prev^.next:=pCur^.next;

writeln(fileOut, pCur^.data);

dispose(pCur);

**end**;

**end**;

Процедура выгрузки из дерева целых на k-ом уровне

**Procedure** writeLevelItem(Tree:Pnode; Level, NeedLevel:integer);

**begin**

**if** Tree <> nil **then**

**begin**

Level:=Level + 1;

**if** Level=NeedLevel **then**

writeln('item=',Tree^.data,'Level=',Level);

**else begin**

writeLevelItem(Tree^.left,Level,NeedLevel);

writeLevelItem(Tree^.right,Level,NeedLevel);

**end**;

**end**;

**end**;

Процедура вывода стека с конца

**Procedure** stwrit(Stack:pt);

**begin**

**if** Stack <> nil **then**

**begin**

stwrite(stack^.next);

writeln(stack^.data);

**end**;

**end**;

Процедура вывода количества листов

**Procedure** WriteList(Tree:PNode;Level:integer);

**begin**

**if** Tree <> nil **then**

**begin**

Level:=Level + 1;

**if** (Tree^.left=nil)**and**(Tree^.right=nil)**then**

writeln('List=',Tree^.data,'CurrentLevel=',Level);

WriteList(Tree^.left,Level);

WriteList(Tree^.right,Level);

**end**;

**end**;

Реализация двунаправленнего списка с выводом через один

**program** double\_linked\_list;

**type** pList = ^listRec;

listRec = **record**

data:integer;

prev:pList;

next:pList;

**end**;

**var** pCur,p:pList;

**function** putList(**var** pCurrent:pList; tFile:text):pList;

**begin**

**new**(pCurrent);

pCurrent^.prev:=nil;

**while not** eof(tFile) **do begin**

readln(pCurrent^.data);

**new**(pCurrent^.next);

pCurrent^.next^.prev:=pCurrent;

putList:=pCurrent;

pCurrent:=pCurrent^.next;

**end**;

pCurrent:=nil;

**end**;

**procedure** getList(pCurrent:pList);

**begin**

**if** pCurrent <> nil **then begin**

writeln(pCurrent^.data);

**if** pCurrent^.prev <> nil **then**

getList(pCurrent^.prev^.prev);

**end**;

**end**;

**begin**

assign(input,'list.txt');

reset(input);

assign(output,'res.txt');

rewrite(output);

getList(putList(pCur, input));

close(input);

close(output);

**end**.

# **Выгрузка дерево по правилу ЛКП в очередь**

**program** ideone;

**type**

Line=string[5];

PNode = ^Node;

Node = **record**

data: Line;

left, right: PNode;

**end**;

pt=^elem;

elem=**record**

data:Line;

next:pt;

**end**;

ListQueue=**record**

First:pt;

Last:pt;

**end**;

**var**

f1:Text;

List:ListQueue;

Tree, p1: PNode;

n, x, i: integer;

**procedure** AddToTree(**var** Tree: PNode; x:Line);

**begin**

**if** Tree = nil **then**

**begin**

**New**(Tree;

Tree^.data := x;

Tree^.left := nil;

Tree^.right := nil;

**exit**;

**end**;

**if** x < Tree^.data **then**

AddToTree(Tree^.left, x)

**else**

AddToTree(Tree^.right, x);

**end**;

**procedure** AddToQueue(**var** ListQ:ListQueue; str:Line);

**var**

PNew : pt;

**begin**

**New**(PNew);

PNew^.data := str;

PNew^.next := nil;

**if** ListQ.First = nil **then**

ListQ.First := PNew

**else**

ListQ.Last^.next := PNew;

ListQ.Last := PNew;

**end**;

**procedure** Lkp(Tree: PNode; **var** List:ListQueue);

**begin**

**if** Tree <> nil **then begin**

Lkp(Tree^.left, List);

AddToQueue(List, Tree^.data);

Lkp(Tree^.right, List);

**end**;

**end**;

**procedure** ReadTree(f1:Text; **var** Tree:PNode);

**var**

str:Line;

**begin**

**while** eof(f1) = false **do**

**begin**

readln(f1, str);

AddToTree(Tree, str);

**end**;

**end**;

**procedure** WriteList(**var** ListQ:ListQueue);

**var**

First:pt;

**begin**

writeln;

First:=ListQ.First;

**while** First <> nil **do**

**begin**

writeln(First^.data);

First := First^.next;

**end**;

**end**;

**begin**

assign(f1, 'input.txt');

assign(output, 'output.txt');

reset(f1);

rewrite(output);

ReadTree(f1, Tree);

Lkp(Tree, List);

WriteList(List);

close(f1);

close(output);

**end**.

# **Посчитать количество элементов на K-ом уровне**

**program** ideone;

**type**

Line=integer;

PNode = ^Node;

Node = **record**

data: Line;

left, right: PNode;

**end**;

pt=^elem;

elem=**record**

data:Line;

next:pt;

**end**;

ListQueue=**record**

First:pt;

Last:pt;

**end**;

**var**

f1:Text;

List:ListQueue;

Tree, p1: PNode;

n, RResult, K: integer;

**procedure** AddToTree(**var** Tree: PNode; x:Line);

**begin**

**if** Tree = nil **then**

**begin**

**New**(Tree);

Tree^.data := x;

Tree^.left := nil;

Tree^.right := nil;

**exit**;

**end**;

**if** x < Tree^.data **then**

AddToTree(Tree^.left, x)

**else**

AddToTree(Tree^.right, x);

**end**;

**procedure** FindCountLevel(Tree: PNode; Level:integer; NeedLevel:integer; **var** ResultCount:integer);

**begin**

**if** Tree <> nil **then begin**

Level := Level + 1;

**if** Level = NeedLevel **then**

ResultCount := ResultCount + 1

**else**

**begin**;

FindCountLevel(Tree^.left, Level, NeedLevel, ResultCount);

FindCountLevel(Tree^.right, Level, NeedLevel, ResultCount);

**end**;

**end**;

**end**;

**procedure** ReadTree(f1:Text; **var** Tree:PNode);

**var**

str:Line;

**begin**

**while** eof(f1) = false **do**

**begin**

readln(f1, str);

AddToTree(Tree, str);

**end**;

**end**;

**begin**

K := 5;

assign(f1, 'input.txt');

assign(output, 'output3.txt');

reset(f1);

rewrite(output);

ReadTree(f1, Tree);

FindCountLevel(Tree, 0, K, RResult);

Writeln(RResult);

close(f1);

close(output);

**end**.