Алфавитно-частотный словарь (alpha-frequency dictionary)

В файле записан текст. Нужно записать в другой файл все слова, встречающиеся в тексте, в алфавитном порядке, и количество повторений для каждого слова.

**Доработать -**

1. нужно убрать знаки препинания, а то получаются разные слова
2. Выводит не в алфавитном порядке, нужно по файлу построить.

program AFreqDict;

*// Алфавитно-частотный словарь (alpha-frequency dictionary). В файле записан текст.*

*// Нужно записать в другой файл все слова, встречающиеся в тексте, в алфавитном порядке, и количество повторений для каждого слова.*

type PNode = ^Node;

Node = record { *структура узла* }

word: string[40]; { *слово* }

count: integer; { *счетчик повторений* }

next: PNode; { *ссылка на следующий* }

end;

{ новые типы данных }

var Head: PNode; { *адрес головы списка* }

NewNode, q: PNode; { *вспомогательные указатели* }

w: string; { *слово из файла* }

F: text; { *файловая переменная* }

count: integer; { *счетчик разных слов* }

{ процедуры и функции }

function CreateNode(NewWord: string): PNode;

var NewNode: PNode;

begin

New(NewNode);

NewNode^.word := NewWord;

NewNode^.count := 1;

NewNode^.next := nil;

Result := NewNode;

end;

procedure AddFirst ( var Head: PNode; NewNode: PNode );

begin

NewNode^.next := Head;

Head := NewNode;

end;

procedure AddAfter ( p, NewNode: PNode );

begin

NewNode^.next := p^.next;

p^.next := NewNode;

end;

procedure AddBefore(var Head: PNode; p, NewNode: PNode);

var q: PNode;

begin

q := Head;

if p = Head then

AddFirst ( Head, NewNode )

else begin

while (q <> nil) and (q^.next <> p) do

q := q^.next;

if q <> nil then AddAfter ( q, NewNode );

end;

end;

function Find(Head: PNode; NewWord: string): PNode;

var q: PNode;

begin

q := Head;

while (q <> nil) and (NewWord <> q^.word) do

q := q^.next;

Result := q;

end;

function FindPlace(Head: PNode; NewWord: string): PNode;

var q: PNode;

begin

q := Head;

while (q <> nil) and (NewWord > q^.word) do

q := q^.next;

Result := q;

End;

function GetWord ( F: Text ) : string;

var c: char;

begin

Result := ''; { пустая строка }

c := ' '; { пробел – чтобы войти в цикл }

{ пропускаем спецсимволы и пробелы }

while not eof(f) and (c <= ' ') do

read(F, c); { читаем слово }

while not eof(f) and (c > ' ') do begin

Result := Result + c;

read(F, c);

end;

end;

begin { Основная программа }

Head := nil;

Assign ( F, 'input.txt' );

Reset ( F );

{ *читаем слова из файла, строим список* }

while True do begin { бесконечный цикл }

w := GetWord ( F ); { читаем слово }

if w = '' then break; { слова закончились, выход }

q := Find ( Head, w ); { ищем слово в списке }

if q <> nil then { нашли, увеличить счетчик }

q^.count := q^.count + 1

else begin { не нашли, добавить в список }

NewNode := CreateNode ( w );

q := FindPlace ( Head, w );

AddBefore ( Head, q, NewNode );

end;

end;

Close ( F );

{ выводим список в другой файл }

q := Head; { проход с начала списка }

count := 0; { обнулили счетчик слов }

Assign(F, 'output.txt');

Rewrite(F);

while q <> nil do begin { пока не конец списка }

count := count + 1; { еще одно слово }

writeln ( F, q^.word, ': ', q^.count );

q := q^.next; { перейти к следующему }

end;

writeln ( F, 'Найдено ',count:15, ' разных слов.' );

Close(F);

end.

{ процедуры и функции }

**function GetWord ( F: Text ) : string;**

var c: char;

begin

Result := ''; { пустая строка }

c := ' '; { пробел – чтобы войти в цикл }

{ пропускаем спецсимволы и пробелы }

while not eof(f) and (c <= ' ') do

read(F, c); { читаем слово }

while not eof(f) and (c > ' ') do begin

Result := Result + c;

read(F, c);

end;

end;

**function CreateNode(NewWord: string): PNode;**

Функция CreateNode (*создать узел*):

вход: новое слово, прочитанное из файла;

выход: адрес нового узла, созданного в памяти.

var NewNode: PNode;

begin

New(NewNode);

NewNode^.word := NewWord;

NewNode^.count := 1;

NewNode^.next := nil;

Result := NewNode;

end;

**procedure AddFirst ( var Head: PNode; NewNode: PNode );**

Добавление узла в начало списка

begin

NewNode^.next := Head;

Head := NewNode;

end;

**procedure AddAfter ( p, NewNode: PNode );**

Добавление узла после заданного

begin

NewNode^.next := p^.next;

p^.next := NewNode;

end;

**procedure AddLast ( var Head: PNode; NewNode: PNode );**

Добавление узла в конец списка

var q: PNode;

begin

if Head = nil then

AddFirst ( Head, NewNode )

else begin

q := Head;

while q^.next <> nil do

q := q^.next;

AddAfter ( q, NewNode );

end;

end;

**procedure AddBefore(var Head: PNode; p, NewNode: PNode);**

Добавление узла перед заданным

var q: PNode;

begin

q := Head;

if p = Head then

AddFirst ( Head, NewNode )

else begin

while (q <> nil) and (q^.next <> p) do

q := q^.next;

if q <> nil then AddAfter ( q, NewNode );

end;

end;

**procedure AddBefore2 ( p, NewNode: PNode );**

var temp: Node;

begin

temp := p^; p^ := NewNode^;

NewNode^ := temp;

p^.next := NewNode;

end;

**function Find(Head: PNode; NewWord: string): PNode;**

Функция Find:

вход: слово (символьная строка);

выход: адрес узла, содержащего это слово или nil.

var q: PNode;

begin

q := Head;

while (q <> nil) and (NewWord <> q^.word) do

q := q^.next;

Result := q;

end;

**function FindPlace(Head: PNode; NewWord: string): PNode;**

Функция FindPlace:

вход: слово (символьная строка);

выход: адрес узла, перед которым нужно вставить это слово или  
 nil, если слово нужно вставить в конец списка.

var q: PNode;

begin

q := Head;

while (q <> nil) and (NewWord > q^.word) do

q := q^.next;

Result := q;

End

**procedure DeleteNode ( var Head: PNode; p: PNode );**

var q: PNode;

begin

if Head = p then

Head := p^.next

else begin

q := Head;

while (q <> nil) and (q^.next <> p) do

q := q^.next;

if q <> nil then q^.next := p^.next;

end;

Dispose(p);

end;

Построение словаря без использования Списков

procedure cleanString(var s: string);

var i: integer;

begin

for i := 1 to length(s) do

if s[i] in ['.', ',', '!', '?']

then delete(s, i, 1);

end;

function getToken(delim: char; var s: string): string;

var p: byte;

begin

getToken := '';

p := pos(delim, s);

if p <> 0 then

begin

getToken := Copy(s, 1, pred(p));

delete(s, 1, p); exit

end;

getToken := s; s := ''

end;

type

LexType = (lxPredict, lxName, lxWord);

const

WhichType: array[LexType] of string =

(' > Predicate', ' > Name', ' > Word');

function analyzeToken(s: string): LexType;

const

Vowels = ['A', 'E', 'I', 'O', 'U', 'Y'];

Consonants = ['B', 'C', 'D', 'F', 'G', 'H', 'J', 'K', 'L',

'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'V', 'Z'];

var

hasDouble: boolean;

i: byte;

begin

hasDouble := false;

analyzeToken := lxName;

if upcase(s[length(s)]) in Consonants then exit;

for i := 1 to length(s) - 1 do

hasDouble := hasDouble or

((upcase(s[i]) in Consonants) and (upcase(s[i+1]) in Consonants));

if hasDouble then analyzeToken := lxPredict

else analyzeToken := lxWord

end;

const

maxString = 30;

maxWords = 1000;

type

arrRec =

record

s: string[maxString];

freq: word;

lex: LexType;

end;

var

arr: array[1 .. maxWords] of arrRec;

const

arrCount: Integer = 0;

function checkInArray(str: string): boolean;

var i: integer;

begin

checkInArray := true;

for i := 1 to arrCount do

if arr[i].s = str then

begin

inc(arr[i].freq); exit

end;

checkInArray := false

end;

function putToArray(str: string; lx: LexType): boolean;

begin

putToArray := false;

inc(arrCount);

if arrCount > maxWords then exit;

arr[arrCount].s := str;

arr[arrCount].lex := lx;

arr[arrCount].freq := 1;

putToArray := true

end;

const

f\_name = 'TEXT.LGL';

var

f: text;

st, sToken: string;

lex: LexType;

i: integer;

begin

assign(f, f\_name);

{$i-} reset(f); {$i+}

while not seekeof(f) do

begin

readln(f, st);

cleanString(st);

while st <> '' do

begin

sToken := getToken(' ', st);

if not checkInArray(sToken) then

begin

lex := analyzeToken(sToken);

if not putToArray(sToken, lex) then

begin

writeln( 'word amount exceeded' ); exit;

end;

end;

end;

end;

for i := 1 to arrCount do

writeln(arr[i].freq:4, ': ', arr[i].s, whichType[arr[i].lex] );

close(f);

end.