Используя представление списка как АТД выполнить следующие действия: заменить в тексте t все слова w1 на слово w2. Текст и слова представлены списками символов.

//Node.h

#pragma once

#include <iostream>

using namespace std;

struct node

{

char data;

node\* next;

};

void pushBack(node\*&, char);

bool EmptyNode(node\*);

void ReadSlovo(ifstream&, node\*&);

void WriteSlovo(node\*);

void DeleteNode(node\*&);

//List.h

#pragma once

#include <iostream>

#include <fstream>

#include "Node.h"

using namespace std;

struct List

{

node\* top;

node\* tail;

node\* marker;

};

void CreateList(List&);

void ResetList(List&);

void AddOne(List&, char);

void AddValue(List&, char);

void ReadList(ifstream&, List&);

void WriteList(List&);

bool EmptyList(List&);

void DeleteList(List&);

//Slovo.h

#pragma once

#include <iostream>

#include <fstream>

#include "Node.h"

#include "List.h"

using namespace std;

bool FindBKV(List&);

void FindTail(List&);

void GetSlovo(List&, bool&);

bool SameSlovo(List, node\*);

node\* PreviousSimvol(List&);

void DeleteSlovo(List&, node\*);

void ChangeOneSlovo(List&, node\*);

void CopySlovo(node\*, node\*&);

void ChangeAllSlovo(List&, node\*&, node\*&);

//Node.cpp

#include <iostream>

#include <fstream>

#include "Node.h"

using namespace std;

bool EmptyNode(node\* top)

{

return top == NULL;

}

void pushBack(node\*& top, char x)

{

node\* p = new node;

p->data = x;

p->next = NULL;

if (EmptyNode(top))

top = p;

else

{

node\* q = top;

while (!EmptyNode(q->next))

q = q->next;

q->next = p;

}

}

void ReadSlovo(ifstream& f, node\*& top)

{

char x;

while (f.peek() != EOF)

{

f.get(x);

pushBack(top, x);

}

}

void WriteSlovo(node\* top)

{

if (EmptyNode(top))

cout << "Слова нет";

else

{

node\* p = top;

while (!EmptyNode(p))

{

cout << p->data;

p = p->next;

}

}

cout << endl;

}

void DeleteNode(node\*& top)

{

while (!EmptyNode(top))

{

node\* p = top->next;

delete top;

top = p;

}

}

//List.cpp

#include <iostream>

#include <fstream>

#include "Node.h"

#include "List.h"

using namespace std;

void CreateList(List& s)

{

s.top = NULL;

s.marker = NULL;

s.tail = NULL;

}

void ResetList(List& s)

{

s.marker = s.top;

}

void AddOne(List& s, char x)

{

s.top = new node;

s.top->data = x;

s.top->next = NULL;

s.tail = s.top;

ResetList(s);

}

void AddValue(List& s, char x)

{

s.marker = new node;

s.marker->data = x;

s.marker->next = NULL;

s.tail->next = s.marker;

s.tail = s.marker;

}

void ReadList(ifstream& f, List& s)

{

ResetList(s);

char x;

if (f.peek() != EOF)

{

f.get(x);

AddOne(s, x);

}

while (f.peek() != EOF)

{

f.get(x);

AddValue(s, x);

}

}

void WriteList(List& s)

{

if (!s.top)

{

cout << "Пустой список";

}

else

{

node\* p = s.marker;

while (p != s.tail->next)

{

cout << p->data;

p = p->next;

}

}

cout << endl;

}

bool EmptyList(List& s)

{

return s.top == NULL;

}

void DeleteList(List& s)

{

ResetList(s);

while (s.top)

{

s.top = s.top->next;

delete s.marker;

ResetList(s);

}

}

//Slovo.cpp

#include <iostream>

#include <fstream>

#include "Node.h"

#include "List.h"

#include "Slovo.h"

using namespace std;

bool FindBKV(List& text)

{

bool b = false;

node\* p = text.marker;

while (p && !b)

{

if (p->data == ' ')

p = p->next;

else

{

b = true;

text.marker = p;

}

}

return b;

}

void FindTail(List& text)

{

node\* p = text.marker;

bool b = false;

text.tail = text.marker;

while (!b)

{

if (p == NULL || p->data == ' ')

b = true;

else

{

text.tail = p;

p = p->next;

}

}

}

void GetSlovo(List& text, bool& found)

{

found = FindBKV(text);

if (found)

FindTail(text);

}

bool SameSlovo(List text, node\* slovo)

{

bool b = true;

node\* p = text.marker;

while (p != text.tail->next && slovo != NULL && b)

{

if (p->data != slovo->data)

b = false;

p = p->next;

slovo = slovo->next;

}

if (slovo != NULL || p != text.tail->next)

b = false;

return b;

}

node\* PreviousSimvol(List& text)

{

node\* q = text.top, \*pr = NULL;

while (q && q != text.marker)

{

pr = q;

q = q->next;

}

if (q == text.marker)

return pr;

return NULL;

}

void DeleteSlovo(List& text, node\* p)

{

node\* q = text.tail->next;

while (text.marker != q)

{

p->next = text.marker->next;

delete text.marker;

text.marker = p->next;

}

text.tail = p;

}

void ChangeOneSlovo(List& text, node\* slovo)

{

node\* prev = PreviousSimvol(text);

node\* p = slovo;

while (p->next)

p = p->next;

p->next = text.marker;

if (text.top == text.marker)

text.top = slovo;

else

prev->next = slovo;

DeleteSlovo(text, p);

}

// slovof - Слова, которые мы ищем в тескте для замены

// slovoс - Слово, на которое мы заменяем искомые слова

void CopySlovo(node\*copyslovo, node\*& slovo)

{

slovo = NULL;

while (copyslovo)

{

pushBack(slovo, copyslovo->data);

copyslovo = copyslovo->next;

}

}

void ChangeAllSlovo(List& text, node\*& slovof, node\*& slovoc)

{

node\* copyslovoc = NULL;

CopySlovo(slovoc, copyslovoc);

bool foundNext;

GetSlovo(text, foundNext);

while (foundNext)

{

if (SameSlovo(text, slovof))

{

ChangeOneSlovo(text, slovoc);

CopySlovo(copyslovoc, slovoc);

}

else

text.marker = text.tail->next;

GetSlovo(text, foundNext);

}

}

//main()

#include <iostream>

#include <fstream>

#include "Node.h"

#include "List.h"

#include "Slovo.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Ru");

ifstream fin("text.txt");

ifstream fin1("slovo1.txt");

ifstream fin2("slovo2.txt");

List s;

node\* slovof = NULL, \* slovoc = NULL;

cout << "Исходный текст:" << endl;

CreateList(s);

ReadList(fin, s);

ResetList(s);

WriteList(s);

cout << "Слово, которое нужно изменить:" << endl;

ReadSlovo(fin1, slovof);

WriteSlovo(slovof);

cout << "Слово, накоторое нужно поменять найденные слова:" << endl;

ReadSlovo(fin2, slovoc);

WriteSlovo(slovoc);

ChangeAllSlovo(s, slovof, slovoc);

cout << "Измененный текст:" << endl;

ResetList(s);

WriteList(s);

DeleteList(s);

DeleteNode(slovof);

DeleteNode(slovoc);

system("pause");

return 0;

}









