Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

**Редактирование текстов (иерархический связный список)**

Выполнил:

студент ф-та ПРИН гр. 381908-4

Бутеску.В.В

Проверил:

ассистент каф. МОСТ, ИИТММ

Лебедев И.Г

Нижний Новгород

2021 г.

Содержание

[Введение 3](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962758)

[Постановка задачи 4](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962759)

[Руководство программиста 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962761)

[Описание структур данных 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962762)

[Описание алгоритмов 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962763)

[Описание структуры программы 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962764)

[Заключение 7](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962765)

[Литература 8](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962766)

[Приложения 9](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962767)

# Введение

Обработка текстовой информации на компьютере широко применяется в различных областях человеческой деятельности: образование, наука, документооборот, кадровый и бухгалтерский учет и др. Вне зависимости от назначения текста типовыми операциями обработки являются создание, просмотр, редактирование и сохранение информации. В связи с тем, что объем текстовой информации может являться очень большим, для эффективного выполнения операций с ней необходимо выбрать представление текста, обеспечивающее структурирование и быстрый доступ к различным элементам текста. Так, текст можно представить в виде линейной последовательности страниц, каждая из которых есть линейная последовательность строк, которые в свою очередь являются линейными последовательностями слов. Такое представление можно осуществлять с любой степенью детализации в зависимости от особенностей прикладной задачи.

В рамках лабораторной работы рассматривается задача разработки учебного редактора текстов, в котором для представления данных используется иерархический связный список. Подобная иерархическая структура представления может применяться при компьютерной реализации математических моделей в виде деревьев и, тем самым, может иметь самое широкое применение в самых различных областях приложений.

# Постановка задачи

Текст – это несколько предложений, связанных друг с другом по смыслу и грамматически. В рамках лабораторной работы в качестве примеров текстов рассматриваются тексты программ. Редактор текстов – программный комплекс, обеспечивающий выполнение операций обработки текста: создание, просмотр, редактирование и сохранение. Специализированные редакторы текстов могут поддерживать выполнение дополнительных операций (например, проверку синтаксиса или контекстный поиск). Иерархический связный список – это многосвязный список, в котором на каждое звено имеется ровно один указатель, а каждое звено содержит два указателя (один на следующее звено в том же уровне, другой на звено в нижерасположенном уровне).класс от базового класса поддержки списков.

В рамках лабораторной работы ставится задача разработки учебного редактора текстов с поддержкой следующих операций:

• выбор текста для редактирования (или создание нового текста);

• демонстрация текста на экране дисплея;

• поддержка средств указания элементов (уровней) текста;

• вставка, удаление и замена строк текста;

Для индикации позиции в тексте, относительно которой выполняются операции перемещения по тексту, вводится понятие текущей строки. Для перемещения по тексту предлагается реализовать следующие операции:

• переход к первой строке текста;

• переход к следующему элементу в том же уровне;

• переход к элементу в нижерасположенном уровне;

• переход к предыдущей позиции в тексте.

Для копирования текста необходимо осуществить обход текста. Так как структура текста является нелинейной, то копирование производится за два прохода, при этом для навигации по исходному тексту и тексту копии используется один объединенный стек. Первый проход производится при подъеме на строку из подуровня – для текущей строки выполняется:

• создание копии звена;

• заполнение в звене-копии поля указателя подуровня pDown (подуровень уже скопирован);

• запись в звене-копии в поле данных значения “Copy”, используемое как маркер для распознавания звена при попадании на него при втором проходе; предполагается, что в тексте данный маркер не встречается;

• запись в звене-копии в поле указателя следующего звена pNext указателя на звенооригинал (для возможности последующего копирования текста исходной строки);

• запись указателя на звено-копию в стек. Второй проход производится при извлечении звена-копии из стека (распознается по маркеру “Copy”)– в этом случае необходимо выполнить:

• заполнение в звене-копии полей данных и указателя следующего звена;

• указатель на звено-копию запоминается в служебной переменной.

**Руководство пользователя**

Пользователю нужно запустить файл main.exe.

Откроется консольное приложение для тестирования очереди.

Программа заполнит очередь числами и выведет их в консоль, так же выведет копирования очереди

Для повторного выполнения потребуется перезапустить программу.

# Руководство программиста

## **Описание структуры программы**

Реализует операции:

lass TextNode

{

protected:

static TextNode \* firstFree;

static TextNode\* textNodes;

TextNode\* next;

TextNode\* down;

char c;

int level; //Уровень. 0 - текст, 1 - строка, 2 - слово, 3 - буква

public:

TextNode(int \_l = 3, char \_c = 0);

TextNode(char\* s, int \_l);

TextNode(const TextNode& \_node);

~TextNode();

TextNode\* GetNext();

TextNode\* GetDown();

char GetC();

int GetLevel();

void SetNext(TextNode\* \_next);

void SetDown(TextNode\* \_down);

void SetC(char \_c);

void SetLevel(int \_l);

friend std::ostream& operator<<(std::ostream& o, TextNode& t);

static void InitMem(int \_size = 3);

static void FreeMem();

void\* operator new(size\_t size);

void operator delete(void\* p);

};

class TextIter;

class Text {

protected:

TextNode\* root;

public:

Text(char\* s = 0);

~Text();

TextIter GetRoot();

TextIter Find(char c);

TextIter Find(char\* c);

TextIter FindWord(char\* c);

void Insert(char\* c, TextIter i);

void InsertData(char\* c, TextIter i);

void InsertLine(char\* c, TextIter iter);

void Delete(int count, TextIter i);

char\* Copy(int count, TextIter i);

friend std::ostream& operator<<(std::ostream& o, Text& t);

};

class TextIter {

protected:

Text& text;

TextNode\* node;

TStack<TextNode\*> st;

public:

TextIter(Text& \_text, TextNode\* \_node, TStack<TextNode\*> \_st);

TextIter(const TextIter& t);

bool GoNext();

bool GoNextChar();

bool IsEnd(); //Првоерка конца

void ResetToString();

void ResetToWord();

TextNode\* Get();

};

# 

# Заключение

Этап 1. Реализация программ поддержки очереди.

Этап 2. Реализация системы имитации обслуживания заданий.

Этап 3. Выполнение дополнительных заданий лабораторной работы.

# Литература

1. Столлингс, В. Структурная организация и архитектура компьютерных систем, 5-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 896 с.: ил. — Парал. тит. англ.
2. Johnson M. Superscalar Microprocessor Design. — Englewood Cliff, New Jersey: Prentice Hall, 1991.
3. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
4. Stone H. High performance Computer Architecture. — Reading, MA: Addison-Wesley, 1993.
5. Tullsen D.M., Eggers S.J. Effective Cache Prefetching on a Bus-Based Multiprocessor. — ACM Transactions on Computer Systems, pp. 57-88, Feb 1995.
6. Chandra D., Guo F., Kim S., Solihin Y. Predicting inter-thread cache contention on a chip multi-processor architecture. — Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA), pp. 340–351, Feb 2005.
7. Press W., Teukolsky S., Vetterling W., Flannery B. Numerical Recipes in C. The Art of Scientific Computing. Second Edition. — Cambridge University Press, 1992.
8. Камаев А.М., Сиднев А.А., Сысоев А.В. Об одном подходе к анализу эффективности приложений // Труды 50-й научной конференции МФТИ «Современные проблемы фундаментальных и прикладных наук»: Часть I. Радиотехника и кибернетика. - М.: МФТИ, 2007.
9. Debugging and performance monitoring. Intel® 64 and IA-32 Architectures Software Developer’s Manual. Volume 3B: System Programming Guide, Part 2. May 2007. — [http://www.intel.com/products/processor/manuals/]
10. Юнаковский А.Д. Начала вычислительных методов для физиков. – Н. Новгород: ИПФ РАН, 2007.

# Приложения

#pragma once

#include "TextNode.h"

#include <fstream>

#include <string>

class TextIter;

class Text {

protected:

TextNode\* root;

public:

Text(char\* s = 0);

~Text();

TextIter GetRoot();

TextIter Find(char c);

TextIter Find(char\* c);

TextIter FindWord(char\* c);

void Insert(char\* c, TextIter i);

void InsertData(char\* c, TextIter i);

void InsertLine(char\* c, TextIter iter);

void Delete(int count, TextIter i);

char\* Copy(int count, TextIter i);

friend std::ostream& operator<<(std::ostream& o, Text& t);

};

class TextIter {

protected:

Text& text;

TextNode\* node;

TStack<TextNode\*> st;

public:

TextIter(Text& \_text, TextNode\* \_node, TStack<TextNode\*> \_st);

TextIter(const TextIter& t);

bool GoNext();

bool GoNextChar();

bool IsEnd(); //Првоерка конца

void ResetToString();

void ResetToWord();

TextNode\* Get();

};

#include "Text\_Iter.h"

#include <fstream>

#include <cstring>

Text::Text(char\* s)

{

root = new TextNode(s, 1);

}

Text::~Text()

{

TextNode::FreeMem();

}

TextIter Text::GetRoot()

{

return TextIter(\*this, root, TStack<TextNode\*>());

}

TextIter Text::Find(char c)

{

TextIter iter = TextIter(\*this, root, TStack<TextNode\*>());

while (iter.GoNextChar())

{

if (iter.Get()->GetC() == c)

return iter;

}

return iter;

}

TextIter Text::FindWord(char\* c)

{

TextIter iter = GetRoot();

while (iter.GoNext())

{

TextIter result = iter;

bool word = true;

int i = 0;

int n = strlen(c);

while (iter.GoNext() && i < n)

{

if (iter.Get()->GetC() != c[i])

{

word = false;

break;

}

i++;

}

if (word)

return result;

}

return iter;

}

TextIter Text::Find(char\* s)

{

TextIter iter = GetRoot();

while (iter.GoNextChar())

{

TextIter result = iter;

bool letter = true;

bool next = true;

int n = strlen(s);

for (int i = 0; i < n; i++)

{

if (s[i] != result.Get()->GetC()) {

letter = false;

break;

}

next = result.GoNextChar();

}

if (letter)

return iter;

}

}

void Text::Insert(char\* c, TextIter iter)

{

TextNode\* node = new TextNode(c, iter.Get()->GetLevel());

TextNode\* next = iter.Get()->GetNext();

node->SetNext(next);

iter.Get()->SetNext(node);

}

void Text::InsertData(char\* c, TextIter iter)

{

if (iter.Get()->GetNext()->GetC() == ' ' && iter.Get()->GetNext()->GetC() == '.' && iter.Get()->GetNext()->GetC() == '\n')

{

iter.GoNext();

TextNode\* node = new TextNode(c, 2);

iter.Get()->SetNext(node);

int k = strlen(c) + 1;

for (int i = 0; i < k; i++)

iter.GoNext();

node = new TextNode(" ", 2);

iter.Get()->SetNext(node);

}

else

{

TextNode\* next = iter.Get()->GetNext();

TextNode\* tmp = iter.Get()->GetNext();

int w = 0;

while (next->GetC() != ' ' && next->GetC() != '.' && next->GetC() != '\n')

{

w++;

next = next->GetNext();

}

char\* word = new char[w + 2];

word[0] = ' ';

word[w + 2] = NULL;

for (int i = 1; i < w + 2; i++)

{

word[i] = tmp->GetC();

tmp = tmp->GetNext();

}

TextNode\* node = new TextNode(" ", 2);

iter.Get()->SetNext(node);

iter.GoNext();

iter.GoNext();

node = new TextNode(c, 2);

iter.Get()->SetNext(node);

int k = strlen(c) + 1;

for (int i = 0; i < k; i++)

iter.GoNext();

node = new TextNode(word, 2);

iter.Get()->SetNext(node);

}

}

void Text::InsertLine(char\* c, TextIter iter)

{

while (iter.Get()->GetC() != '\n')

{

iter.GoNext();

}

TextNode\* node = new TextNode(c, 1);

iter.Get()->SetNext(node);

iter.GoNext();

node = new TextNode("\n", 2);

iter.Get()->SetNext(node);

}

void Text::Delete(int count, TextIter iter)

{

if (count == 0)

return;

TStack<TextNode\*> stack[3];

int del = 0;

int level = iter.Get()->GetLevel();

if (level < 3)

iter.GoNextChar();

if (level >= 1 && level <= 3)

{

stack[level - 1].Push(iter.Get());

if (level == 3)

del++;

}

TextIter it = iter;

while (it.GoNext() && del < count)

{

level = it.Get()->GetLevel();

if (level >= 1 && level <= 3)

{

stack[level - 1].Push(it.Get());

if (level == 3)

del++;

}

}

if (del < count)

throw logic\_error("ERROR");

if (stack[0].IsEmpty() && stack[1].IsEmpty())

{

TextIter prev = iter;

prev.ResetToWord();

while (prev.Get()->GetNext() != iter.Get() && prev.Get()->GetDown() != iter.Get())

prev.GoNext();

TextNode\* last = stack[2].Get();

if (prev.Get()->GetLevel() == 2)

prev.Get()->SetDown(last->GetNext());

else

prev.Get()->SetNext(last->GetNext());

delete last;

return;

}

TextIter prev = iter;

prev.ResetToWord();

while (prev.Get()->GetNext() != iter.Get() && prev.Get()->GetDown() != iter.Get())

prev.GoNext();

if (prev.Get()->GetNext() == iter.Get())

prev.Get()->SetNext(nullptr);

else

prev.Get()->SetDown(nullptr);

if (stack[0].IsEmpty())

{

TextNode\* last\_word = stack[1].Get();

TextNode\* last\_letter = stack[2].Get();

TextIter cur = iter;

cur.ResetToWord();

cur.Get()->SetNext(last\_word);

last\_word->SetDown(last\_letter->GetNext());

delete last\_letter;

for (int i = 1; i < 3; i++)

{

while (!stack[i].IsEmpty())

{

TextNode\* del = stack[i].Get();

delete del;

}

}

}

else

{

TextNode\* last\_string = stack[0].Get();

TextNode\* last\_word = stack[1].Get();

TextNode\* last\_letter = stack[2].Get();

TextIter cur = iter;

cur.ResetToString();

cur.Get()->SetNext(last\_string);

cur.ResetToWord();

cur.Get()->SetNext(last\_word);

last\_word->SetDown(last\_letter->GetNext());

delete last\_letter;

for (int i = 0; i < 3; i++)

{

while (!stack[i].IsEmpty())

{

TextNode\* temp = stack[i].Get();

delete temp;

}

}

}

}

char\* Text::Copy(int count, TextIter iter)

{

if (count < 0)

throw logic\_error("Error");

char\* res = new char[count ++];

while (iter.Get()->GetLevel() < 3)

iter.GoNext();

if (count > 0)

{

res[0] = iter.Get()->GetC();

for (int i = 0; i < count; i++)

{

iter.GoNextChar();

res[i] = iter.Get()->GetC();

}

}

res[count] = 0;

return res;

}

std::ostream& operator<<(std::ostream& out, Text& t)

{

out << \*(t.root);

return out;

}

#include "Text\_Iter.h"

#include <fstream>

#include <cstring>

TextIter::TextIter(Text& \_text, TextNode\* \_node, TStack<TextNode\*> \_st) : text(\_text)

{

node = \_node;

st = \_st;

}

TextIter::TextIter(const TextIter& t) : text(t.text)

{

node = t.node;

st = t.st;

text = t.text;

}

bool TextIter::GoNext()

{

if (node->GetLevel() == 3)

{

if (node->GetNext() != 0)

node = node->GetNext();

else

{

node = st.Get();

if (node->GetNext() != 0)

node = node->GetNext();

else

{

node = st.Get();

if (node->GetNext() != 0)

node = node->GetNext();

else

{

node = 0;

return false;

}

}

}

}

else

{

st.Push(node);

node = node->GetDown();

}

return true;

}

bool TextIter::GoNextChar()

{

if (node->GetLevel() == 1)

{

if (node->GetDown() == nullptr)

return false;

else

{

st.Push(node);

node = node->GetDown();

st.Push(node);

node = node->GetDown();

return true;

}

}

if (node->GetLevel() == 2)

{

if (node->GetDown() == nullptr)

return false;

else

{

st.Push(node);

node = node->GetDown();

return true;

}

}

if (node->GetLevel() == 3 && node->GetNext() != nullptr)

{

if (node->GetNext() == nullptr)

return false;

else

{

node = node->GetNext();

return true;

}

}

if (node->GetLevel() == 3 && node->GetNext() == nullptr)

{

if (st.IsEmpty())

throw logic\_error("Error");

TextNode\* temp = st.Get();

if (temp->GetNext() != nullptr)

{

st.Push(temp->GetNext());

node = temp->GetNext()->GetDown();

return true;

}

}

return false;

}

bool TextIter::IsEnd()

{

if (node == nullptr)

return true;

if (node->GetLevel() == 1 || node->GetLevel() == 2)

return false;

if (node->GetLevel() == 3 && node->GetNext() != nullptr)

return false;

if (node->GetLevel() == 3 && node->GetNext() == nullptr)

{

if (st.IsEmpty())

throw logic\_error("Error");

TextNode\* prev = st.Get();

if (prev->GetNext() != nullptr)

return true;

}

return true;

}

void TextIter::ResetToString()

{

while (node->GetLevel() > 1) {

st.Get();

if (st.IsEmpty())

throw logic\_error("Error");

node = st.Get();

}

}

void TextIter::ResetToWord()

{

if (node->GetLevel() == 1)

{

st.Push(node);

node = node->GetDown();

}

else if (node->GetLevel() == 3)

node = st.Get();

}

TextNode\* TextIter::Get() {

return this->node;

}

#pragma once

#include <iostream>

#include "Stack.h"

class TextNode

{

protected:

static TextNode \* firstFree;

static TextNode\* textNodes;

TextNode\* next;

TextNode\* down;

char c;

int level; //Уровень. 0 - текст, 1 - строка, 2 - слово, 3 - буква

public:

TextNode(int \_l = 3, char \_c = 0);

TextNode(char\* s, int \_l);

TextNode(const TextNode& \_node);

~TextNode();

TextNode\* GetNext();

TextNode\* GetDown();

char GetC();

int GetLevel();

void SetNext(TextNode\* \_next);

void SetDown(TextNode\* \_down);

void SetC(char \_c);

void SetLevel(int \_l);

friend std::ostream& operator<<(std::ostream& o, TextNode& t);

static void InitMem(int \_size = 3);

static void FreeMem();

void\* operator new(size\_t size);

void operator delete(void\* p);

};

TextNode\* TextNode::firstFree = 0;

TextNode\* TextNode::textNodes = 0;

#include "TextNode.h"

#include <cstring>

std::ostream& operator<<(std::ostream& o, TextNode& t)

{

if (t.level == 3)

{

o << t.c;

if (t.next != nullptr)

o << \*(t.next);

}

else

{

if (t.down != nullptr)

o << \*(t.down);

if (t.next != nullptr)

o << \*(t.next);

}

return o;

}

TextNode::TextNode(int \_l, char \_c)

{

next = nullptr;

down = nullptr;

c = \_c;

level = \_l;

}

TextNode::TextNode(char\* s, int l) //Принимает слово

{

next = nullptr;

c = 0;

level = l;

down = nullptr;

if (l < 1 || l > 3)

throw std::exception();

if (l == 1)

{

int sz = strlen(s);

if (sz > 0) {

int start = 0;

TextNode\* wcur = new TextNode(2);

this->SetDown(wcur);

for (int i = 0; i < sz; i++) {

if (s[i] == ' ' || i == sz - 1)

{

if (start != 0) {

wcur->SetNext(new TextNode(2));

wcur = wcur->GetNext();

}

TextNode\* lcur = new TextNode(3, s[start]);

wcur->SetDown(lcur);

for (int j = start + 1; j <= i; j++) {

lcur->SetNext(new TextNode(3, s[j]));

lcur = lcur->GetNext();

}

lcur->SetNext(nullptr);

start = i + 1;

}

}

wcur->SetNext(nullptr);

}

}

if (l == 2)

{

next = nullptr;

c = 0;

level = l;

int sz = strlen(s);

if (sz > 0) {

TextNode\* lcur = new TextNode(3, s[0]);

this->SetDown(lcur);

for (int i = 1; i < sz; i++) {

lcur->SetNext(new TextNode(3, s[i]));

lcur = lcur->GetNext();

}

lcur->SetNext(nullptr);

}

}

if (l == 3)

{

c = s[0];

level = l;

next = nullptr;

}

}

TextNode::TextNode(const TextNode& \_node)

{

next = nullptr;

down = nullptr;

c = \_node.c;

level = \_node.level;

}

TextNode::~TextNode()

{

}

TextNode\* TextNode::GetNext()

{

return next;

}

TextNode\* TextNode::GetDown()

{

return down;

}

char TextNode::GetC()

{

return c;

}

int TextNode::GetLevel()

{

return level;

}

void TextNode::SetNext(TextNode\* \_next)

{

next = \_next;

}

void TextNode::SetDown(TextNode\* \_down)

{

down = \_down;

}

void TextNode::SetC(char \_c)

{

c = \_c;

}

void TextNode::SetLevel(int \_l)

{

if (\_l >= 1 && \_l <= 3)

level = \_l;

else

throw logic\_error("Error");

}

void TextNode::InitMem(int \_size)

{

if (\_size <= 0)

throw logic\_error("Error");

if ((firstFree == 0) && (textNodes == 0))

{

textNodes = new TextNode[\_size];

firstFree = textNodes;

}

else if ((firstFree != 0) && (textNodes == 0))

throw logic\_error("Error");

else

textNodes = new TextNode[\_size \* 2];

firstFree = textNodes;

}

void TextNode::FreeMem() //Освобождение памяти

{

if (textNodes != 0)

{

delete[] textNodes;

}

textNodes = 0;

firstFree = 0;

}

void\* TextNode::operator new(size\_t size)

{

if ((firstFree == 0) && (textNodes == 0))

InitMem();

if ((firstFree == 0) && (textNodes != 0))

InitMem();

TextNode\* node = firstFree;

firstFree = firstFree->next;

node->next = 0;

return node;

}

void TextNode::operator delete(void\* p)

{

TextNode\* node = (TextNode\*)p;

if ((firstFree == 0) && (textNodes == 0))

throw logic\_error("Error");

node->next = nullptr;

node->down = nullptr;

node->c = 0;

node->level = 3;

node->next = firstFree;

firstFree = node;

}