Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Редактирование текстов»**

**Выполнил**: студент группы 3822Б1ПР4

Фомин Владимир Михайлович

**Проверил:**

ассистент каф. МО ЭВМ, ВМК

Лебедев И.Г.

Нижний Новгород

2024

Содержание

[**Введение** 3](#_Toc25968134)

[**Постановка задачи** 3](#_Toc25968135)

[**Руководство пользователя** 4](#_Toc25968136)

[**Руководство программиста** 6](#_Toc25968137)

[***Описание структуры программы*** 6](#_Toc25968138)

[**Описание алгоритмов** 7](#_Toc25968139)

[**Заключение** 12](#_Toc25968147)

[**Приложение** 13](#_Toc25968149)

**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 учебной задачи**

**1.1. Основные понятия и определения**

*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.

*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).

**1.2. Требования к лабораторной работе**

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;
* запись подготовленного текста в файл.

При выполнении операций чтения (при выборе для редактирования уже существующего текста) и записи редактор должен использовать стандартный формат, принятый в файловой системе для представления текстовых файлов, обеспечивая тем самым совместимость учебного редактора текстов и существующего программного обеспечения.

**Руководство пользователя**

Данная программа написана с помощью программы Microsoft Visual Studio 2019 на языке C++.

При запуске программы, первым шагом работы пользователю предлагается выбор функций программы.

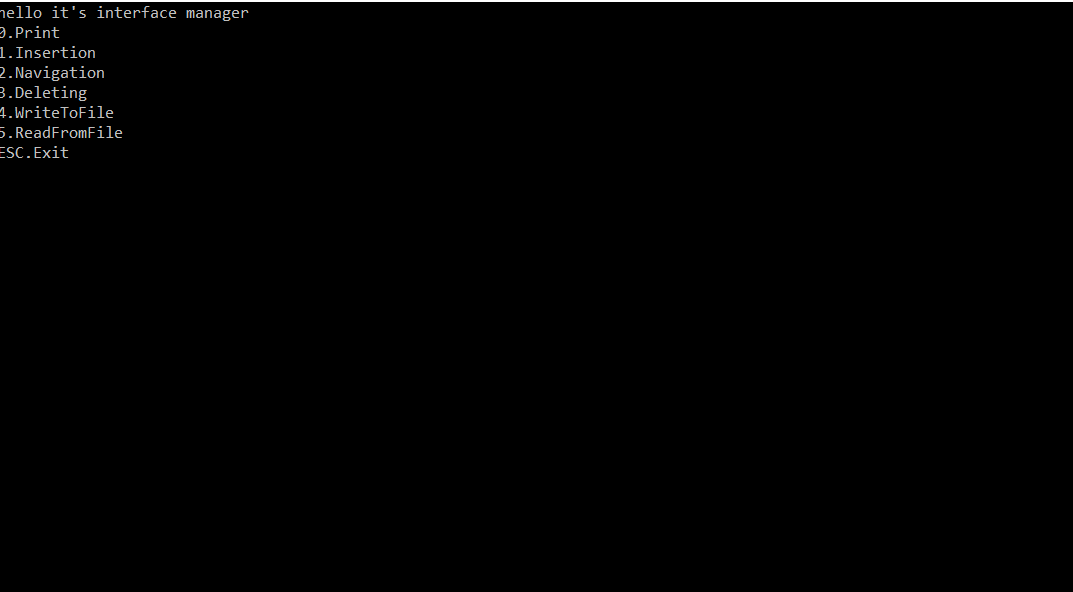


Рисунок 1 Старт программы

Пользователь может выгрузить текст из файла и вывести его на экран

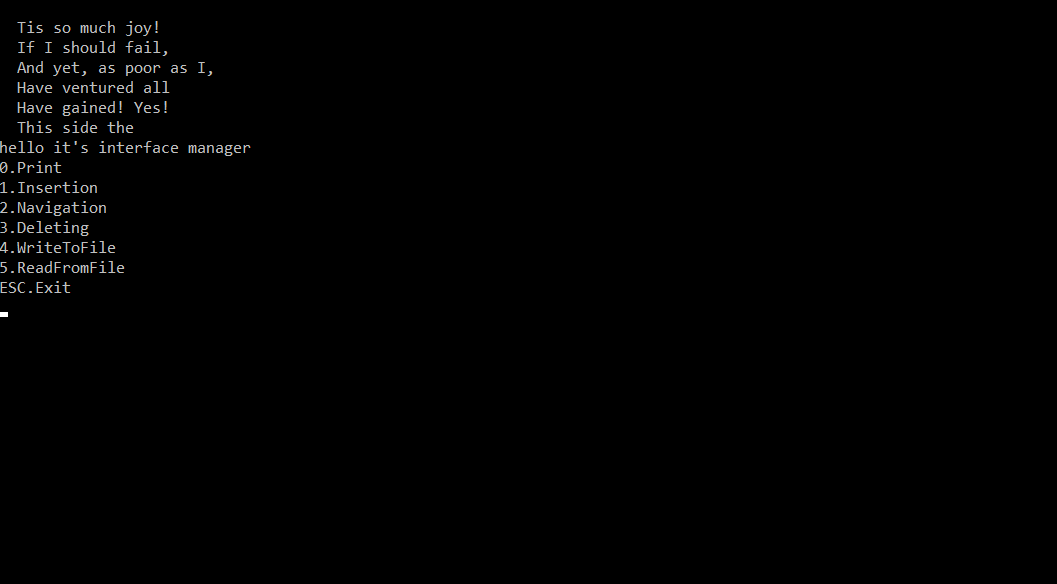


Рисунок 2 Вывод текста из файла

После этого пользователь может редактировать полученный текст

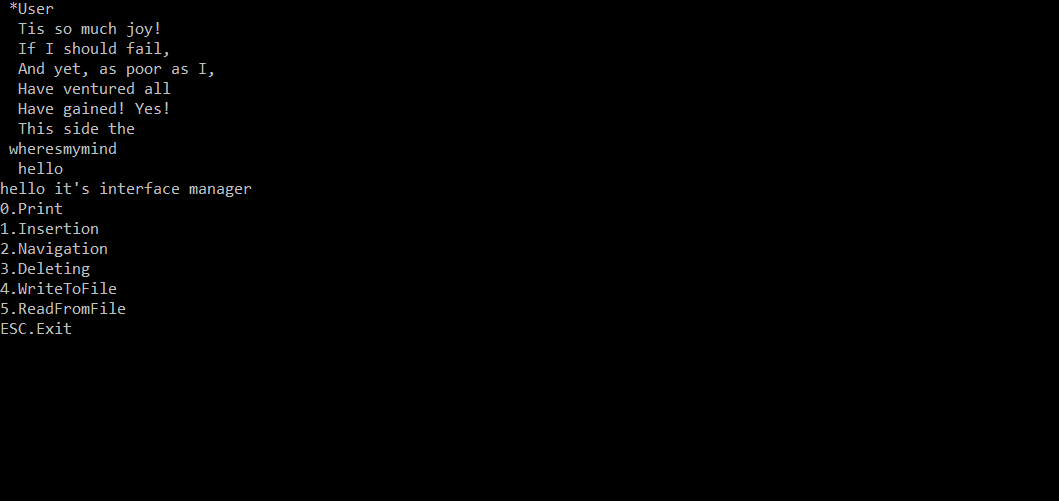


Рисунок 3. Редактирование текста пользователем

И наконец, выгрузить текст в файл.

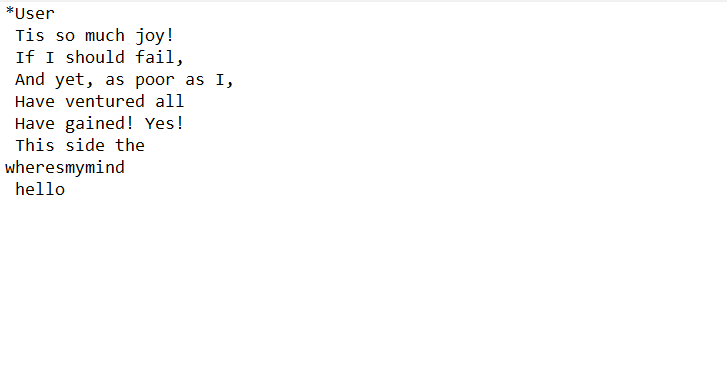
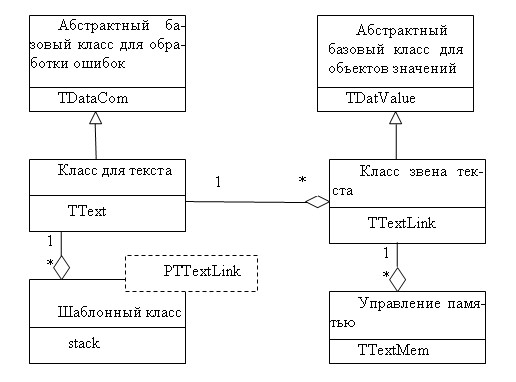


Рисунок 4. Вывод текста в файл

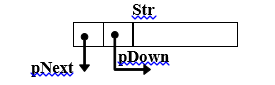
**Руководство программиста**

**Описание структуры программы**

* TTextLink.h, TTextLink.cpp – модуль с классом для звена текста;
* TText.h, TText.cpp – модуль с классом, реализующим операции над текстом;
* TTextViewer.h, TTextViewer.cpp – модуль с классом, реализующим визуализацию текста;
* TText\_test.cpp – модуль программы тестирования.
* TDataCom.h – абстрактный базовый класс для обработки ошибок
* 

**Описание структур данных**

Выбор структуры звеньев состоит в следующем:

* каждое звено структуры хранения содержит два поля указателей и поле для хранения данных;
* нижний уровень иерархической структуры звеньев ограничивается уровнем строк, а не символов. Это повышает эффективность использования памяти, так как в случае отдельного уровня для хранения символов, затраты на хранение служебной информации (указатели) в несколько раз превышают затраты на хранение самих данных. При использовании строк в качестве атомарных элементов поле для хранения данных является массивом символов заданного размера;
* 

При выбранном способе представления для следующего примера текста

Раздел 2

2.1. Полиномы

* + 1. Определение
    2. Структура

2.2. Тексты

1.Определение

2.Структура

структура хранения текста будет иметь вид:

**Раздел 2**

**. Полиномы**

**2.1**

**1**

**. Определение**

**. Структура**

**2**

**2.2**

**. Тексты**

**1**

**. Определение**

**2**

**. Структура**

**Описание алгоритмов программы**

В программе используются следующие функции:

* + 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 последовательного доступа ко всем элемента текста (например, для печати текста) необходимо совершить обход текста.

Схема обхода состоит в следующем. При начале обхода следует перейти от начала текста (корня дерева) до атомарного уровня по указателям нижерасположенных уровней, запоминая при этом все пройденные звенья в стеке. После обработки найденной строки, далее следует переходить по строкам того же уровня, до последней строки в этом уровне. После обхода уровня текущего уровня необходимо извлечь звено из стека и повторить всю выше приведенную последовательность действий. Обход текста будет завершен, когда стек пуст.

int TText::GoFirstLink()

{

while (!Path.empty())

Path.pop();

pCurrent = pFirst;

if (pCurrent == NULL)

SetRetCode(TextError);

else

SetRetCode(TextOK);

return RetCode;

}

int TText::GoDownLink() //go to next string on Down

{

SetRetCode(TextError);

if (pCurrent != NULL)

if (pCurrent->pDown != NULL)

{

Path.push(pCurrent);

pCurrent = pCurrent->pDown;

SetRetCode(TextOK);

}

return RetCode;

}

int TText::GoNextLink() // go to next string on Next

{

SetRetCode(TextError);

if (pCurrent != NULL)

if (pCurrent->pNext != NULL)

{

Path.push(pCurrent);

pCurrent = pCurrent->pNext;

SetRetCode(TextOK);

}

return RetCode;

}

int TText::GoPrevLink() // go to previous string

{

if (Path.empty())

SetRetCode(TextNoPrev);

else

{

pCurrent = Path.top();

Path.pop();

SetRetCode(TextOK);

}

return RetCode;

}

**Итератор**

Как и при работе с линейными списками, использование итератора позволяет упростить реализацию операций с текстом и обеспечивает унифицированный способ обработки элементов структуры данных.

Итератор включет следующие методы:

* инициализация (установка на корневое звено)
* проверка завершения текста
* переход к следующему звену

**Копирование**

* 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 звено-оригинал (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.

PTText TText::getCopy() //text copying

{

PTTextLink pl1, pl2, pl = pFirst, cpl = NULL;

if (pFirst != NULL)

{

while (!St.empty())

St.pop(); // очистка стека

while (true)

{

if (pl != NULL)

{ // переход к первому атому

pl = GetFirstAtom(pl);

St.push(pl);

pl = pl->GetDown();

}

else if (St.empty())

break;

else

{

pl1 = St.top();

St.pop();

if (strstr(pl1->Str, (char \*)"Copy") == NULL)

{ // первый этап создания копии

// создание копии - pDown на уже скопированный подуровень

pl2 = new TTextLink((char \*)"Copy", pl1, cpl); // pNext на оригинал

St.push(pl2);

pl = pl1->GetNext();

cpl = NULL;

}

else

{ // второй этап создания копии

pl2 = pl1->GetNext();

strncpy(pl1->Str, pl2->Str, TextLineLength);

pl1->pNext = cpl;

cpl = pl1;

}

}

}

}

* return new TText(cpl);

**Тесты**

Для тестирования программы использьзовался модуль Google test

#include <gtest.h>

#include "TText.h"

TEST(TText, can\_create\_empty\_text)

{

TTextLink::InitMemSystem(1);

ASSERT\_NO\_THROW(TText tmp);

}

TEST(TText, copy\_has\_its\_own\_memory)

{

TTextLink::InitMemSystem(3);

PTTextLink pl = new TTextLink((char \*)"copy", NULL, NULL);

TText t1(pl);

PTText cpy = t1.getCopy();

EXPECT\_NE(&t1, cpy);

}

TEST(TText, can\_getcopy)

{

TTextLink::InitMemSystem(3);

PTTextLink pl = new TTextLink((char \*)"copy", NULL, NULL);

TText t1(pl);

PTText cpy = t1.getCopy();

string s = t1.GetLine();

s = cpy->GetLine();

EXPECT\_EQ(t1.GetLine(), cpy->GetLine());

}

TEST(TText, new\_text\_ret\_code\_OK)

{

TTextLink::InitMemSystem(1);

PTTextLink pLink = new TTextLink((char \*)"str");

TText tmp(pLink);

EXPECT\_EQ(TextOK, tmp.GetRetCode());

}

/\* Access \*/

TEST(TText, can\_get\_set\_line)

{

TTextLink::InitMemSystem(1);

PTTextLink pLink = new TTextLink((char \*)"str");

TText tmp(pLink);

tmp.SetLine((char \*)"fire");

EXPECT\_EQ("fire", tmp.GetLine());

}

/\* Insertion \*/

TEST(TText, can\_ins\_next\_line)

{

TTextLink::InitMemSystem(2);

PTTextLink pLink = new TTextLink((char \*)"str");

TText tmp(pLink);

tmp.InsNextLine((char \*)"str11");

EXPECT\_EQ(TextOK, tmp.GetRetCode());

}

TEST(TText, can\_ins\_next\_section)

{

TTextLink::InitMemSystem(2);

PTTextLink pLink = new TTextLink((char \*)"str");

TText tmp(pLink);

tmp.InsNextSection((char \*)"str11");

EXPECT\_EQ(TextOK, tmp.GetRetCode());

}

TEST(TText, can\_ins\_down\_line)

{

TTextLink::InitMemSystem(2);

PTTextLink pLink = new TTextLink((char \*)"str");

TText tmp(pLink);

tmp.InsDownLine((char \*)"str11");

EXPECT\_EQ(TextOK, tmp.GetRetCode());

}

TEST(TText, can\_ins\_down\_section)

{

TTextLink::InitMemSystem(2);

PTTextLink pLink = new TTextLink((char \*)"str");

TText tmp(pLink);

tmp.InsDownSection((char \*)"str11");

EXPECT\_EQ(TextOK, tmp.GetRetCode());

}

/\*Navigation\*/

TEST(TText, can\_go\_frist\_link)

{

TTextLink::InitMemSystem(2);

PTTextLink pNext = new TTextLink((char \*)"strNext", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.GoFirstLink();

EXPECT\_STREQ("str", tmp.GetLine().c\_str());

}

TEST(TText, can\_go\_down\_link)

{

TTextLink::InitMemSystem(2);

PTTextLink pDown = new TTextLink((char \*)"str2", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", NULL, pDown);

TText tmp(pLink);

tmp.GoDownLink();

EXPECT\_STREQ("str2", tmp.GetLine().c\_str());

}

TEST(TText, can\_go\_next\_link)

{

TTextLink::InitMemSystem(2);

PTTextLink pNext = new TTextLink((char \*)"strNext", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.GoNextLink();

EXPECT\_STREQ("strNext", tmp.GetLine().c\_str());

}

TEST(TText, can\_go\_prev\_link)

{

TTextLink::InitMemSystem(3);

PTTextLink pNext2 = new TTextLink((char \*)"strNext2", NULL, NULL);

PTTextLink pNext = new TTextLink((char \*)"strNext", pNext2, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.GoNextLink();

tmp.GoNextLink();

tmp.GoPrevLink();

EXPECT\_STREQ("strNext", tmp.GetLine().c\_str());

}

/\* Deleting \*/

TEST(TText, can\_delete\_down\_line)

{

TTextLink::InitMemSystem(2);

PTTextLink pDown = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", NULL, pDown);

TText tmp(pLink);

tmp.DelDownLine();

tmp.GoDownLink();

EXPECT\_EQ(TextError, tmp.GetRetCode());

}

TEST(TText, can\_delete\_down\_section)

{

TTextLink::InitMemSystem(2);

PTTextLink pDown = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", NULL, pDown);

TText tmp(pLink);

tmp.DelDownSection();

tmp.GoDownLink();

EXPECT\_EQ(TextError, tmp.GetRetCode());

}

TEST(TText, can\_delete\_next\_line)

{

TTextLink::InitMemSystem(2);

PTTextLink pNext = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.DelNextLine();

tmp.GoNextLink();

EXPECT\_EQ(TextError, tmp.GetRetCode());

}

TEST(TText, can\_delete\_next\_section)

{

TTextLink::InitMemSystem(2);

PTTextLink pNext = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.DelNextSection();

tmp.GoNextLink();

EXPECT\_EQ(TextError, tmp.GetRetCode());

}

/\* Iterator \*/

TEST(TText, can\_detect\_text\_ended)

{

TTextLink::InitMemSystem(1);

PTTextLink pLink = new TTextLink((char \*)"str", NULL, NULL);

TText tmp(pLink);

EXPECT\_TRUE(tmp.IsTextEnded());

}

TEST(TText, can\_reset)

{

TTextLink::InitMemSystem(2);

PTTextLink pNext = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.Reset();

EXPECT\_STREQ("str", tmp.GetLine().c\_str());

}

TEST(TText, can\_go\_next)

{

TTextLink::InitMemSystem(3);

PTTextLink pNext2 = new TTextLink((char \*)"strNext2", NULL, NULL);

PTTextLink pNext = new TTextLink((char \*)"strNext1", pNext2, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.Reset();

tmp.GoNext();

EXPECT\_STREQ("strNext1", tmp.GetLine().c\_str());

}

/\* Work with files \*/

TEST(TText, can\_read\_from\_file)

{

TTextLink::InitMemSystem(10);

PTTextLink pNext = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

ASSERT\_NO\_THROW(tmp.Read((char \*)"file.txt"));

}

TEST(TText, can\_write\_to\_file)

{

TTextLink::InitMemSystem(10);

PTTextLink pNext = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

ASSERT\_NO\_THROW(tmp.Write((char \*)"output.txt"));

}

TEST(TText, can\_print\_text)

{

TTextLink::InitMemSystem(10);

PTTextLink pNext = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

ASSERT\_NO\_THROW(tmp.Print());

}

#include <gtest.h>

#include "TTextLink.h"

TEST(TTextLink, can\_create\_empty\_Link)

{

TTextLink::InitMemSystem(1);

ASSERT\_NO\_THROW(new TTextLink());

}

TEST(TTextLink, can\_create\_link)

{

TTextLink::InitMemSystem(1);

ASSERT\_NO\_THROW(new TTextLink((char \*)"test"));

}

TEST(TTextLink, can\_printFreeLink)

{

TTextLink::InitMemSystem(1);

TTextLink tmp((char \*)"hello");

ASSERT\_NO\_THROW(tmp.PrintFreeLink());

}

TEST(TTextLink, can\_detect\_atom)

{

TTextLink::InitMemSystem(2);

PTTextLink tmp = new TTextLink;

TTextLink notAtom((char\*)"sometext", NULL, tmp);

EXPECT\_EQ(0, notAtom.IsAtom());

}

TEST(TTextLink, can\_get\_next)

{

TTextLink::InitMemSystem(2);

PTTextLink tmp = new TTextLink;

TTextLink testLink((char\*)"sometext", tmp, NULL);

EXPECT\_EQ(tmp, testLink.GetNext());

}

TEST(TTextLink, can\_get\_down)

{

TTextLink::InitMemSystem(2);

PTTextLink tmp = new TTextLink;

TTextLink testLink((char\*)"sometext", NULL, tmp);

EXPECT\_EQ(tmp, testLink.GetDown());

}

TEST(TTextLink, get\_copy\_has\_its\_own\_mem)

{

TTextLink::InitMemSystem(2);

PTTextLink testLink = new TTextLink((char\*)"sometext", NULL, NULL);

PTTextLink tmp = (PTTextLink)testLink->GetCopy();

EXPECT\_NE(tmp, testLink);

}

**Спецификации классов:**

С учетом предложенных к реализации алгоритмов можно сделать следующие объявления классов.

Класс TTextMem для системы управления памятью и класс TTextLink для строк текста (файл TTextLink.h):

class TTextLink : public TDatValue {

protected:

TStr Str; // поле для хранения строки текста

PTTextLink pNext, pDown; // указатели по тек. уровень и на подуровень

static TTextMem MemHeader; // система управления памятью

public:

static void InitMemSystem (int size=MemSize); // инициализация памяти

static void PrintFreeLink (void); // печать свободных звеньев

void \* operator new (size\_t size); // выделение звена

void operator delete (void \*pM); // освобождение звена

static void MemCleaner (TText &txt); // сборка мусора

TTextLink (TStr s = NULL, PTTextLink pn = NULL, PTTextLink pd = NULL){

pNext = pn; pDown = pd;

if (s != NULL) strcpy(Str,s); else Str[0]='\0';

}

~TTextLink() {}

int IsAtom () {return pDown == NULL;} // проверка атомарности звена

PTTextLink GetNext() {return pNext;}

PTTextLink GetDown() {return pDown;}

PTDatValue GetCopy() {return new TTextLink(Str,pNext,pDown);}

protected:

virtual void Print (std::ostream &os) {os << Str;}

friend class TText;

};

class TTextMem {

PTTextLink pFirst; // указатель на первое звено

PTTextLink pLast; // указатель на последнее звено

PTTextLink pFree; // указатель на первое свободное звено

friend class TTextLink;

};

typedef TTextMem \*PTTextMem;

Класс для представления иерархического связного списка (TText.h):

class TText;

typedef TText\* PTText;

class TText : public TDataCom {

protected:

PTTextLink pFirst; // указатель корня дерева

PTTextLink pCurrent; // указатель текущей строки

stack< PTTextLink > Path; // стек траектории движения по тексту

stack< PTTextLink > St; // стек для итератора

PTTextLink GetFirstAtom (PTTextLink pl); // поиск первого атома

void PrintText (PTTextLink ptl); // печать текста со звена ptl

PTTextLink ReadText (ifstream &TxtFile); //чтение текста из файла

void PrintTextFile(PTTextLink pl,ofstream& txtFile);

public:

TText (PTTextLink pl = NULL);

~TText () {pFirst =NULL;}

PTText getCopy();

// навигация

int GoFirstLink (void); // переход к первой строке

int GoDownLink (void); // переход к следующей строке по Down

int GoNextLink (void); // переход к следующей строке по Next

int GoPrevLink (void); // переход к предыдущей позиции в тексте

// доступ

string GetLine(void); // чтение текущей строки

void SetLine ( string s); // замена текущей строки

// модификация

void InsDownLine (string s); // вставка строки в подуровень

void InsDownSection (string s); // вставка раздела в подуровень

void InsNextLine (string s); // вставка строки в том же уровне

void InsNextSection (string s); // вставка раздела в том же уровне

void DelDownLine (void); // удаление строки в подуровне

void DelDownSection (void); // удаление раздела в подуровне

void DelNextLine (void); // удаление строки в том же уровне

void DelNextSection (void); // удаление раздела в том же уровне

// итератор

int Reset (void); // установить на первую звапись

int IsTextEnded (void) const; // текст завершен?

int GoNext (void); // переход к следующей записи

//работа с файлами

void Read (char \*pFileName); // ввод текста из файла

void Write (char \*pFileName); // вывод текста в файл

//печать

void Print (void); // печать текста

};

**Заключение**

В ходе лабораторной работы мы, написали программу реализующую игру «Быки и коровы». Научились использовать массивы и условные операторы, а также циклы и функции ввода-вывода для работы с пользователем.

**Приложение**

// Copyright 2024 Vladimir Fomin

//Файлы заголовков

//tdatacom.h

#ifndef \_\_DATACOM\_H\_\_

#define \_\_DATACOM\_H\_\_

#define DataOK 0

#define DataErr -1

// TDataCom является общим базовым классом

class TDataCom

{

protected:

int RetCode; // Код завершения

int SetRetCode(int ret) { return RetCode = ret; }

public:

TDataCom(): RetCode(DataOK) {}

virtual ~TDataCom() {}

int GetRetCode()

{

int temp = RetCode;

RetCode = DataOK;

return temp;

}

};

#endif

//tdataroot.h

typedef TElem\* PTElem;

typedef int TData; // тип значений в СД

enum TMemType { MEM\_HOLDER, MEM\_RENTER };

class TDataRoot: public TDataCom

{

protected:

PTElem pMem; // память для СД

int MemSize; // размер памяти для СД

int DataCount; // количество элементов в СД

TMemType MemType; // режим управления памятью

void SetMem(void \*p, int Size); // задание памяти

public:

virtual ~TDataRoot();

TDataRoot(int Size = DefMemSize);

virtual bool IsEmpty(void) const; // контроль пустоты СД

virtual bool IsFull (void) const; // контроль переполнения СД

virtual void Put (const TData &Val) = 0; // добавить значение

virtual TData Get (void) = 0; // извлечь значение

// служебные методы

virtual int IsValid() = 0; // тестирование структуры

virtual void Print() = 0; // печать значений

// дружественные классы

friend class TMultiStack;

friend class TSuperMultiStack;

friend class TComplexMultiStack;

};

#endif

Tadatavalue.h

#pragma once

#include <iostream>

class TDatValue;

typedef TDatValue \*PTDatValue;

class TDatValue

{

public:

virtual TDatValue \*GetCopy() = 0; //create copy

~TDatValue() {}

};

//Ttext.h

#pragma once

#include <stack>

#include <fstream>

#include "tdatacom.h"

#include "TTextLink.h"

class TText;

typedef TText\* PTText;

class TText : public TDataCom {

protected:

PTTextLink pFirst; // указатель корня дерева

PTTextLink pCurrent; // указатель текущей строки

stack< PTTextLink > Path; // стек траектории движения по тексту

stack< PTTextLink > St; // стек для итератора

PTTextLink GetFirstAtom (PTTextLink pl); // поиск первого атома

void PrintText (PTTextLink ptl); // печать текста со звена ptl

PTTextLink ReadText (ifstream &TxtFile); //чтение текста из файла

void PrintTextFile(PTTextLink pl,ofstream& txtFile);

public:

TText (PTTextLink pl = NULL);

~TText () {pFirst =NULL;}

PTText getCopy();

// навигация

int GoFirstLink (void); // переход к первой строке

int GoDownLink (void); // переход к следующей строке по Down

int GoNextLink (void); // переход к следующей строке по Next

int GoPrevLink (void); // переход к предыдущей позиции в тексте

// доступ

string GetLine(void); // чтение текущей строки

void SetLine ( string s); // замена текущей строки

// модификация

void InsDownLine (string s); // вставка строки в подуровень

void InsDownSection (string s); // вставка раздела в подуровень

void InsNextLine (string s); // вставка строки в том же уровне

void InsNextSection (string s); // вставка раздела в том же уровне

void DelDownLine (void); // удаление строки в подуровне

void DelDownSection (void); // удаление раздела в подуровне

void DelNextLine (void); // удаление строки в том же уровне

void DelNextSection (void); // удаление раздела в том же уровне

// итератор

int Reset (void); // установить на первую звапись

int IsTextEnded (void) const; // текст завершен?

int GoNext (void); // переход к следующей записи

//работа с файлами

void Read (char \*pFileName); // ввод текста из файла

void Write (char \*pFileName); // вывод текста в файл

//печать

void Print (void); // печать текста

};

//ttexters.h

#pragma once

#define TextOK 0 /\* Ошибок нет \*/

/\* коды ситуаций \*/

#define TextNoDown 101 //нет подуровня для текущей позиции

#define TextNoNext 102 //нет след. раздела текущего уровня

#define TextNoPrev 103 // текущая позиция в начале текста

/\* error codes \*/

#define TextError -102 /\* ошибка в тексте \*/

#define TextNoMem -101 /\* нет памяти \*/

//ttextlink.h

#pragma once

#include <string.h>

#include <iostream>

#include "tdatvalue.h"

#include "ttexters.h"

#define TextLineLength 20

#define MemSize 20

using namespace std;

class TTextLink;

class TText;

typedef TTextLink \*PTTextLink;

typedef char TStr[TextLineLength];

class TTextMem {

PTTextLink pFirst; // указатель на первое звено

PTTextLink pLast; // указатель на последнее звено

PTTextLink pFree; // указатель на первое свободное звено

friend class TTextLink;

};

typedef TTextMem \*PTTextMem;

class TTextLink : public TDatValue {

protected:

TStr Str; // поле для хранения строки текста

PTTextLink pNext, pDown; // указатели по тек. уровень и на подуровень

static TTextMem MemHeader; // система управления памятью

public:

static void InitMemSystem (int size=MemSize); // инициализация памяти

static void PrintFreeLink (void); // печать свободных звеньев

void \* operator new (size\_t size); // выделение звена

void operator delete (void \*pM); // освобождение звена

static void MemCleaner (TText &txt); // сборка мусора

TTextLink (TStr s = NULL, PTTextLink pn = NULL, PTTextLink pd = NULL){

pNext = pn; pDown = pd;

if (s != NULL) strcpy(Str,s); else Str[0]='\0';

}

~TTextLink() {}

int IsAtom () {return pDown == NULL;} // проверка атомарности звена

PTTextLink GetNext() {return pNext;}

PTTextLink GetDown() {return pDown;}

PTDatValue GetCopy() {return new TTextLink(Str,pNext,pDown);}

protected:

virtual void Print (std::ostream &os) {os << Str;}

friend class TText;

};

//ttextviewer.h

#pragma once

#include <string>

#include <iostream>

#include <stdlib.h>

#include "TText.h"

#define ESC 27

class TTextViewer

{

public:

static void StartInterface(PTText);

private:

static void Insertion(PTText pText);

static void Navigation(PTText pText);

static void Deleting(PTText pText);

};

//ttext.cpp

#include "TText.h"

#include <TTextViewer.h>

#define BufLength 90

static char StrBuf[BufLength + 1]; //Буфер для ввода строк

static int TextLevel; // номер текущего уровня текста

TText::TText(PTTextLink pl)

{

if (pl == NULL)

pl = new TTextLink();

pFirst = pl;

pCurrent = pl;

}

// Navigation

int TText::GoFirstLink()

{

while (!Path.empty())

Path.pop();

pCurrent = pFirst;

if (pCurrent == NULL)

SetRetCode(TextError);

else

SetRetCode(TextOK);

return RetCode;

}

int TText::GoDownLink() //go to next string on Down

{

SetRetCode(TextError);

if (pCurrent != NULL)

if (pCurrent->pDown != NULL)

{

Path.push(pCurrent);

pCurrent = pCurrent->pDown;

SetRetCode(TextOK);

}

return RetCode;

}

int TText::GoNextLink() // go to next string on Next

{

SetRetCode(TextError);

if (pCurrent != NULL)

if (pCurrent->pNext != NULL)

{

Path.push(pCurrent);

pCurrent = pCurrent->pNext;

SetRetCode(TextOK);

}

return RetCode;

}

int TText::GoPrevLink() // go to previous string

{

if (Path.empty())

SetRetCode(TextNoPrev);

else

{

pCurrent = Path.top();

Path.pop();

SetRetCode(TextOK);

}

return RetCode;

}

//Access

string TText::GetLine() // reading current string

{

if (pCurrent == NULL)

return string("");

else

return string(pCurrent->Str);

}

void TText::SetLine(string s) // replacement current string

{

if (pCurrent == NULL)

SetRetCode(TextError);

else

strncpy(pCurrent->Str, s.c\_str(), TextLineLength);

pCurrent->Str[TextLineLength - 1] = '\0';

}

// Modification

void TText::InsDownLine(string s) //insertion string to Down(SubLevel)

{

if (pCurrent == NULL)

SetRetCode(TextError);

else

{

PTTextLink pd = pCurrent->pDown;

PTTextLink pl = new TTextLink(NULL, pd, NULL); // я думаю при инициализации можно передать s.c\_str()!!!!!

strncpy(pl->Str, s.c\_str(), TextLineLength);

pl->Str[TextLineLength - 1] = '\0'; // set 0 if s.length > Str.length

pCurrent->pDown = pl;

SetRetCode(TextOK);

}

}

void TText::InsDownSection(string s) // insertion section to Down (SubLevel)

{

if (pCurrent == NULL)

SetRetCode(TextError);

else

{

PTTextLink pd = pCurrent->pDown;

PTTextLink pl = new TTextLink(NULL, NULL, pd);

strncpy(pl->Str, s.c\_str(), TextLineLength);

pl->Str[TextLineLength - 1] = '\0'; // set 0 if s.length > Str.length

pCurrent->pDown = pl;

SetRetCode(TextOK);

}

}

void TText::InsNextLine(string s) // insert string to cur level

{

if (pCurrent == NULL)

SetRetCode(TextError);

else

{

strncpy(StrBuf, s.c\_str(), TextLineLength);

pCurrent->pNext = new TTextLink(StrBuf, pCurrent->pNext, nullptr);

SetRetCode(TextOK);

}

}

void TText::InsNextSection(string s)

{

if (pCurrent == NULL)

SetRetCode(TextError);

else

{

strncpy(StrBuf, s.c\_str(), TextLineLength);

pCurrent->pNext = new TTextLink(StrBuf, nullptr, pCurrent->pNext);

SetRetCode(TextOK);

}

}

void TText::DelDownSection() // delete sublevel (down section)

{

if (pCurrent == NULL)

SetRetCode(TextError);

else if (pCurrent->pDown != NULL)

{

PTTextLink pl1 = pCurrent->pDown;

PTTextLink pl2 = pl1->pNext;

pCurrent->pDown = pl2;

}

}

void TText::DelDownLine() // delete down line

{

if (pCurrent == NULL)

SetRetCode(TextError);

else if (pCurrent->pDown != NULL)

{

PTTextLink pl1 = pCurrent->pDown;

PTTextLink pl2 = pl1->pNext;

if (pl1->pDown == NULL)

pCurrent->pDown = pl2;

}

}

void TText::DelNextLine()

{

if (pCurrent == NULL)

SetRetCode(TextError);

else if (pCurrent->pNext != NULL)

{

PTTextLink pl1 = pCurrent->pNext;

PTTextLink pl2 = pl1->pNext;

if (pl1->pNext == NULL)

pCurrent->pNext = pl2;

}

}

void TText::DelNextSection()

{

if (pCurrent == NULL)

SetRetCode(TextError);

else if (pCurrent->pNext != NULL)

{

PTTextLink pl1 = pCurrent->pNext;

PTTextLink pl2 = pl1->pNext;

pCurrent->pNext = pl2;

}

}

/\* Iterator \*/

int TText::Reset()

{

while (!St.empty()) // clear stack

St.pop();

pCurrent = pFirst;

if (pCurrent != NULL)

St.push(pCurrent);

if (pCurrent->pNext != NULL)

St.push(pCurrent->pNext);

if (pCurrent->pDown != NULL)

St.push(pCurrent->pDown);

return IsTextEnded();

}

int TText::IsTextEnded() const

{

return (St.size() == 0);

}

int TText::GoNext()

{

if (!IsTextEnded())

{

pCurrent = St.top();

St.pop();

if (pCurrent != pFirst)

{

if (pCurrent->pNext != NULL)

St.push(pCurrent->pNext);

if (pCurrent->pDown != NULL)

St.push(pCurrent->pDown);

}

}

return IsTextEnded();

}

PTTextLink TText::GetFirstAtom(PTTextLink pl)

{

PTTextLink tmp = pl;

while (!tmp->IsAtom())

{

St.push(tmp);

tmp = tmp->GetDown();

}

return tmp;

}

PTText TText::getCopy() //text copying

{

PTTextLink pl1, pl2, pl = pFirst, cpl = NULL;

if (pFirst != NULL)

{

while (!St.empty())

St.pop(); // очистка стека

while (true)

{

if (pl != NULL)

{ // переход к первому атому

pl = GetFirstAtom(pl);

St.push(pl);

pl = pl->GetDown();

}

else if (St.empty())

break;

else

{

pl1 = St.top();

St.pop();

if (strstr(pl1->Str, (char \*)"Copy") == NULL)

{ // первый этап создания копии

// создание копии - pDown на уже скопированный подуровень

pl2 = new TTextLink((char \*)"Copy", pl1, cpl); // pNext на оригинал

St.push(pl2);

pl = pl1->GetNext();

cpl = NULL;

}

else

{ // второй этап создания копии

pl2 = pl1->GetNext();

strncpy(pl1->Str, pl2->Str, TextLineLength);

pl1->pNext = cpl;

cpl = pl1;

}

}

}

}

return new TText(cpl);

}

void TText::Print()

{

TextLevel = 0;

PrintText(pFirst);

Reset();

}

void TText::PrintText(PTTextLink ptl)

{

if (ptl != NULL)

{

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

cout << " ";

cout << " " << ptl->Str << endl;

TextLevel++;

PrintText(ptl->GetDown());

TextLevel--;

PrintText(ptl->GetNext());

}

}

void TText::Read(char \*pFileName)

{

ifstream TxtFile(pFileName);

string buf;

TextLevel = 0;

if (&TxtFile != NULL)

pFirst = ReadText(TxtFile);

}

PTTextLink TText::ReadText(ifstream &TxtFile)

{

bool endoffile = 0;

string buf;

PTTextLink pHead, ptl;

pHead = ptl = new TTextLink();

if (!(TxtFile.is\_open()))

{

SetRetCode(TextError);

throw "error";

}

while (getline(TxtFile, buf))

{

if (buf.front() == '}')

{

endoffile = 1;

TextLevel--;

break;

}

else if (buf.front() == '{')

{

TextLevel++;

ptl->pDown = ReadText(TxtFile);

}

else //присоеднение следующей строки

{

ptl->pNext = new TTextLink((char \*)buf.c\_str());

ptl = ptl->pNext;

}

}

ptl = pHead;

if (pHead->pDown == NULL) //удаление первой строки если нет подуровня

{

pHead = pHead->pNext;

delete ptl;

}

return pHead;

}

void TText::Write(char \*pFileName)

{

TextLevel = 0;

ofstream TxtFile(pFileName);

PrintTextFile(pFirst, TxtFile);

Reset();

}

void TText::PrintTextFile(PTTextLink pl, ofstream &txtFile)

{

SetRetCode(TextError);

if (pl != NULL)

{

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

txtFile << " ";

txtFile << pl->Str << endl;

TextLevel++;

PrintTextFile(pl->GetDown(), txtFile);

TextLevel--;

PrintTextFile(pl->GetNext(), txtFile);

}

}

//ttextlink.cpp

#include "TTextLink.h"

#include "TText.h"

TTextMem TTextLink::MemHeader;

void TTextLink::InitMemSystem(int size)

{

char line[100];

MemHeader.pFirst = (PTTextLink) new char[sizeof(TTextLink) \* size];

MemHeader.pFree = MemHeader.pFirst;

MemHeader.pLast = MemHeader.pFirst + (size - 1);

PTTextLink pLink = MemHeader.pFirst;

for (int i = 0; i < size - 1; i++, pLink++) // размерка памяти

pLink->pNext = pLink + 1;

pLink->pNext = NULL;

}

void TTextLink::PrintFreeLink()

{

PTTextLink pLink = MemHeader.pFree;

cout << "List of free links" << endl;

for (; pLink != NULL; pLink = pLink->pNext)

cout << pLink->Str << endl;

}

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

{

PTTextLink pLink = MemHeader.pFree;

if (MemHeader.pFree != NULL)

pLink = MemHeader.pFree;

else

throw "not initial memory";

if (MemHeader.pFree != NULL)

MemHeader.pFree = pLink->pNext;

return pLink;

}

void TTextLink::operator delete(void \*pM)

{

PTTextLink pLink = (PTTextLink)pM;

pLink->pNext = MemHeader.pFree;

MemHeader.pFree = pLink;

}

void TTextLink::MemCleaner(TText &txt)

{

string str;

// маркировка строк текста - маркер : '&&&'

for (txt.Reset(); !txt.IsTextEnded(); txt.GoNext())

{

if (str.find("&&&") != 0)

txt.SetLine("&&&" + txt.GoNext());

//маркировка свободных звеньев

PTTextLink pLink = MemHeader.pFree;

for (; pLink != NULL; pLink = pLink->pNext)

strcpy(pLink->Str, "&&&");

//сборка мусора

pLink = MemHeader.pFirst;

for(;pLink <= MemHeader.pLast;pLink++)

{

if(strstr(pLink->Str,"&&&") != NULL) // строка текста или свободное звено

strcpy(pLink->Str,pLink->Str + 3); // снятие маркировки

else

delete pLink; // неучтенное звено в список свободных

}

}

}

//ttextviewer.cpp

#include "TTextViewer.h"

void TTextViewer::Insertion(PTText pText)

{

char input = 0;

do

{

cout << "1.SetLine\n2.InsDownLine\n3.InsDownSection\n";

cout << "4.InsNextLine\n5.InsNextSection\nESC.Exit\n";

cin >> input;

system("cls");

} while (input != ESC && input > 5 && input <= 0);

string str;

switch (input)

{

case '1':

cout << "input line:" << endl;

cin >> str;

pText->SetLine(str.c\_str());

break;

case '2':

cout << "input line:" << endl;

cin >> str;

pText->InsDownLine(str);

break;

case '3':

cout << "input line:" << endl;

cin >> str;

pText->InsDownSection(str);

break;

case '4':

cout << "input line:" << endl;

cin >> str;

pText->InsNextLine(str);

break;

case '5':

cout << "input line:" << endl;

cin >> str;

pText->InsNextSection(str);

break;

}

}

void TTextViewer::Navigation(PTText pText)

{

char input=0;

do

{

cout << "1.GoFirstLink\n2.GoDownLink\n3.GoNextLink\n";

cout << "4.GoPrevLink\nESC.Exit\n";

cin >> input;

system("cls");

} while (input != ESC && input > 5 && input <= 0);

string str;

switch (input)

{

case '1':

pText->GoFirstLink();

break;

case '2':

pText->GoDownLink();

break;

case '3':

pText->GoNextLink();

break;

case '4':

pText->GoPrevLink();

break;

}

}

void TTextViewer::Deleting(PTText pText)

{

char input=0;

do

{

cout << "1.DelDownLine\n2.DelDownSection\n3.DelNextLine\n";

cout << "4.DelNextSection\nESC.Exit\n";

cin >> input;

system("cls");

} while (input != ESC && input > 5 && input <= 0);

string str;

switch (input)

{

case '1':

pText->DelDownLine();

break;

case '2':

pText->DelDownSection();

break;

case '3':

pText->DelNextLine();

break;

case '4':

pText->DelNextSection();

break;

}

}

void TTextViewer::StartInterface(PTText pText)

{

char input=0;

while (input != ESC)

{

cout << "hello it's interface manager" << endl;

cout << "0.Print" << endl;

cout << "1.Insertion" << endl;

cout << "2.Navigation" << endl;

cout << "3.Deleting" << endl;

cout << "4.WriteToFile" << endl;

cout << "5.ReadFromFile" << endl;

cout << "ESC.Exit" << endl;

cin >> input;

system("cls");

if ((input - '0') == 0)

pText->Print();

else if ((input - '0') == 1)

Insertion(pText);

else if ((input - '0') == 2)

Navigation(pText);

else if ((input - '0') == 3)

{

Deleting(pText);

}

else if ((input - '0') == 4)

{

pText->Write((char\*)"..\\output.txt");

}

else if ((input - '0') == 5)

{

pText->Read((char\*)"..\\file.txt");

}

}

}

//Text test modul

#include <gtest.h>

int main(int argc, char \*\*argv)

{

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

#include <gtest.h>

#include "TText.h"

TEST(TText, can\_create\_empty\_text)

{

TTextLink::InitMemSystem(1);

ASSERT\_NO\_THROW(TText tmp);

}

TEST(TText, copy\_has\_its\_own\_memory)

{

TTextLink::InitMemSystem(3);

PTTextLink pl = new TTextLink((char \*)"copy", NULL, NULL);

TText t1(pl);

PTText cpy = t1.getCopy();

EXPECT\_NE(&t1, cpy);

}

TEST(TText, can\_getcopy)

{

TTextLink::InitMemSystem(3);

PTTextLink pl = new TTextLink((char \*)"copy", NULL, NULL);

TText t1(pl);

PTText cpy = t1.getCopy();

string s = t1.GetLine();

s = cpy->GetLine();

EXPECT\_EQ(t1.GetLine(), cpy->GetLine());

}

TEST(TText, new\_text\_ret\_code\_OK)

{

TTextLink::InitMemSystem(1);

PTTextLink pLink = new TTextLink((char \*)"str");

TText tmp(pLink);

EXPECT\_EQ(TextOK, tmp.GetRetCode());

}

/\* Access \*/

TEST(TText, can\_get\_set\_line)

{

TTextLink::InitMemSystem(1);

PTTextLink pLink = new TTextLink((char \*)"str");

TText tmp(pLink);

tmp.SetLine((char \*)"fire");

EXPECT\_EQ("fire", tmp.GetLine());

}

/\* Insertion \*/

TEST(TText, can\_ins\_next\_line)

{

TTextLink::InitMemSystem(2);

PTTextLink pLink = new TTextLink((char \*)"str");

TText tmp(pLink);

tmp.InsNextLine((char \*)"str11");

EXPECT\_EQ(TextOK, tmp.GetRetCode());

}

TEST(TText, can\_ins\_next\_section)

{

TTextLink::InitMemSystem(2);

PTTextLink pLink = new TTextLink((char \*)"str");

TText tmp(pLink);

tmp.InsNextSection((char \*)"str11");

EXPECT\_EQ(TextOK, tmp.GetRetCode());

}

TEST(TText, can\_ins\_down\_line)

{

TTextLink::InitMemSystem(2);

PTTextLink pLink = new TTextLink((char \*)"str");

TText tmp(pLink);

tmp.InsDownLine((char \*)"str11");

EXPECT\_EQ(TextOK, tmp.GetRetCode());

}

TEST(TText, can\_ins\_down\_section)

{

TTextLink::InitMemSystem(2);

PTTextLink pLink = new TTextLink((char \*)"str");

TText tmp(pLink);

tmp.InsDownSection((char \*)"str11");

EXPECT\_EQ(TextOK, tmp.GetRetCode());

}

/\*Navigation\*/

TEST(TText, can\_go\_frist\_link)

{

TTextLink::InitMemSystem(2);

PTTextLink pNext = new TTextLink((char \*)"strNext", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.GoFirstLink();

EXPECT\_STREQ("str", tmp.GetLine().c\_str());

}

TEST(TText, can\_go\_down\_link)

{

TTextLink::InitMemSystem(2);

PTTextLink pDown = new TTextLink((char \*)"str2", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", NULL, pDown);

TText tmp(pLink);

tmp.GoDownLink();

EXPECT\_STREQ("str2", tmp.GetLine().c\_str());

}

TEST(TText, can\_go\_next\_link)

{

TTextLink::InitMemSystem(2);

PTTextLink pNext = new TTextLink((char \*)"strNext", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.GoNextLink();

EXPECT\_STREQ("strNext", tmp.GetLine().c\_str());

}

TEST(TText, can\_go\_prev\_link)

{

TTextLink::InitMemSystem(3);

PTTextLink pNext2 = new TTextLink((char \*)"strNext2", NULL, NULL);

PTTextLink pNext = new TTextLink((char \*)"strNext", pNext2, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.GoNextLink();

tmp.GoNextLink();

tmp.GoPrevLink();

EXPECT\_STREQ("strNext", tmp.GetLine().c\_str());

}

/\* Deleting \*/

TEST(TText, can\_delete\_down\_line)

{

TTextLink::InitMemSystem(2);

PTTextLink pDown = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", NULL, pDown);

TText tmp(pLink);

tmp.DelDownLine();

tmp.GoDownLink();

EXPECT\_EQ(TextError, tmp.GetRetCode());

}

TEST(TText, can\_delete\_down\_section)

{

TTextLink::InitMemSystem(2);

PTTextLink pDown = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", NULL, pDown);

TText tmp(pLink);

tmp.DelDownSection();

tmp.GoDownLink();

EXPECT\_EQ(TextError, tmp.GetRetCode());

}

TEST(TText, can\_delete\_next\_line)

{

TTextLink::InitMemSystem(2);

PTTextLink pNext = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.DelNextLine();

tmp.GoNextLink();

EXPECT\_EQ(TextError, tmp.GetRetCode());

}

TEST(TText, can\_delete\_next\_section)

{

TTextLink::InitMemSystem(2);

PTTextLink pNext = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.DelNextSection();

tmp.GoNextLink();

EXPECT\_EQ(TextError, tmp.GetRetCode());

}

/\* Iterator \*/

TEST(TText, can\_detect\_text\_ended)

{

TTextLink::InitMemSystem(1);

PTTextLink pLink = new TTextLink((char \*)"str", NULL, NULL);

TText tmp(pLink);

EXPECT\_TRUE(tmp.IsTextEnded());

}

TEST(TText, can\_reset)

{

TTextLink::InitMemSystem(2);

PTTextLink pNext = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.Reset();

EXPECT\_STREQ("str", tmp.GetLine().c\_str());

}

TEST(TText, can\_go\_next)

{

TTextLink::InitMemSystem(3);

PTTextLink pNext2 = new TTextLink((char \*)"strNext2", NULL, NULL);

PTTextLink pNext = new TTextLink((char \*)"strNext1", pNext2, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

tmp.Reset();

tmp.GoNext();

EXPECT\_STREQ("strNext1", tmp.GetLine().c\_str());

}

/\* Work with files \*/

TEST(TText, can\_read\_from\_file)

{

TTextLink::InitMemSystem(10);

PTTextLink pNext = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

ASSERT\_NO\_THROW(tmp.Read((char \*)"file.txt"));

}

TEST(TText, can\_write\_to\_file)

{

TTextLink::InitMemSystem(10);

PTTextLink pNext = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

ASSERT\_NO\_THROW(tmp.Write((char \*)"output.txt"));

}

TEST(TText, can\_print\_text)

{

TTextLink::InitMemSystem(10);

PTTextLink pNext = new TTextLink((char \*)"strDown", NULL, NULL);

PTTextLink pLink = new TTextLink((char \*)"str", pNext, NULL);

TText tmp(pLink);

ASSERT\_NO\_THROW(tmp.Print());

}

#include <gtest.h>

#include "TTextLink.h"

TEST(TTextLink, can\_create\_empty\_Link)

{

TTextLink::InitMemSystem(1);

ASSERT\_NO\_THROW(new TTextLink());

}

TEST(TTextLink, can\_create\_link)

{

TTextLink::InitMemSystem(1);

ASSERT\_NO\_THROW(new TTextLink((char \*)"test"));

}

TEST(TTextLink, can\_printFreeLink)

{

TTextLink::InitMemSystem(1);

TTextLink tmp((char \*)"hello");

ASSERT\_NO\_THROW(tmp.PrintFreeLink());

}

TEST(TTextLink, can\_detect\_atom)

{

TTextLink::InitMemSystem(2);

PTTextLink tmp = new TTextLink;

TTextLink notAtom((char\*)"sometext", NULL, tmp);

EXPECT\_EQ(0, notAtom.IsAtom());

}

TEST(TTextLink, can\_get\_next)

{

TTextLink::InitMemSystem(2);

PTTextLink tmp = new TTextLink;

TTextLink testLink((char\*)"sometext", tmp, NULL);

EXPECT\_EQ(tmp, testLink.GetNext());

}

TEST(TTextLink, can\_get\_down)

{

TTextLink::InitMemSystem(2);

PTTextLink tmp = new TTextLink;

TTextLink testLink((char\*)"sometext", NULL, tmp);

EXPECT\_EQ(tmp, testLink.GetDown());

}

TEST(TTextLink, get\_copy\_has\_its\_own\_mem)

{

TTextLink::InitMemSystem(2);

PTTextLink testLink = new TTextLink((char\*)"sometext", NULL, NULL);

PTTextLink tmp = (PTTextLink)testLink->GetCopy();

EXPECT\_NE(tmp, testLink);

}