МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Фудеральное государственное автономное образовательное учреждение

Высшего образования

**«Национальный исследовательский**

**Нижегородский государственный университет им.Н.И.Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафелра математического обеспечения и суперкомпьютерных технологий**

Направление подготовки: «Фундаментальная информатика и

информационные технологии»

Профиль подготовки: «Фундаментальная информатика и информационные

технологии (общий профиль)»

**ОТЧЕТ**

по учебной практике

**Выполнил:** студент группы 381606-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.Н. Кутовой

**Проверил:** к.ф.-м.н., доц.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К.А. Баркалов

Нижний Новгород

2018

Оглавление

[Программа для работы с текстом 3](#_Toc514228621)

[Введение 3](#_Toc514228622)

[Постановка задачи 5](#_Toc514228623)

[Описание структуры программы 6](#_Toc514228624)

[Описание модулей программы 7](#_Toc514228625)

[Описание алгоритмов 9](#_Toc514228626)

[Результаты 11](#_Toc514228627)

# Программа для работы с текстом

# Введение

Хранение и редактирование текстов, без преувеличения, очень важная прикладная задача. Для ее реализации удобно использовать иерархические списки. Каждый элемент списка содержит в себе указатель на элемент, находящийся с ним «на одном уровне» и указатель на элемент, находящийся на уровень ниже. Так можно графически представить себе эту структуру:

pDown

pNext

К сожалению, такая структура хранения имеет недостаток – сложность удаления элементов. К примеру, мы решаем удалить элемент, выделенный красным овалом.

Получается так, что мы теряем указатели на элементы, для которого удаленный элемент был отцовским. Эту задачу мы решим с помощью выделения памяти для нашего списка из общего списка свободных звеньев, хранящимся в линейном массиве.

Указатель на начало

Указатель на свободный элемент

Указатель на конец

Так мы легко сможем искать «потерянные» элементы и снова добавлять их к свободным.

# Постановка задачи

Необходимо разработать структуру данных представляющую собой иерархический список с особым видом работы с памятью, который позволял бы хранить и редактировать текст. Должны быть реализованы функции добавления заголовка, подзаголовка, секции и их удаление, чтение текста из файла и его сохранение в файл.

# Описание структуры программы

Список модулей программы

Программа состоит из следующих модулей:

* TLink.h – заголовочный файл класса «список».
* TText.h - заголовочный файл класса «текст».
* mystructures.h – заголовочный файл, содержащий дополнительные функции, необходимые для выполнения программы.
* TLink.cpp – файл, содержащий реализации функций класса «список».
* TText.cpp – файл, содержащий реализации функций класса «текст».
* main.cpp – файл, содержащий тело программы, которая выполняет задачи поставленные ранее в пункте «Постановка задачи».

# Описание модулей программы

**Модуль TLink.h**

**Модуль содержит стуктуру**

struct TMem { TLink\*pFirst, \*pFree, \*pLast;}; - структура для «разметки» памяти.

**Модуль содержит class TLink, который содержит поля:**

char str[80]; - поле для хранения строки,

TLink\*pNext, \*pDown; - указатели на следующие элементы списка,

static TMem mem; - поле, служащее для выделения памяти списку,

bool flag; - служебное поле, использующееся при очистке памяти.

**Класс содержит следующие прототипы функций и некоторые реализации:**

TLink(char\*s = nullptr, TLink\*pN = nullptr, TLink\*pD = nullptr) – конструктор,

{

pNext = pN;

pDown = pD;

if (s == nullptr)

{

str[0] = '\o';

}

else

strcpy(str, s);

}

char\* getStr() – функция получения поля str,

{

return str;

}

TLink\* getPDown() - функция получения поля pDown,

{

return pDown;

}

TLink\* getPNext() - функция получения поля pNext.

{

return pNext;

}

static void InitMem(size\_t s); - функция инициализации памяти.

static void MemClean(TText &t); - функция очистки памяти.

static int PrintFree(TText &t); - функция печати незанятых ячеек памяти.

void\* operator new(size\_t s); - перегрузка оператора new для выделения памяти.

void operator delete(void\*p); - перегрузка оператора delete для удаления памяти.

Модуль TText.h

**Модуль содержит class TText, который содержит поля:**

TLink\*pFirst, \*pCurr; - указатели на первый элемент текста и на текущий элемент,

std::stack<TLink\*> st; - стек,

int lvl; - служебная переменная, использующаяся в алгоритме печати текста.

**Класс содержит следующие прототипы функций и некоторые реализации:**

TText()

int gonxtlnk(); - Сдвиг указателя pNext вниз

int godwnlnk(); - Сдвиг указателя pDown вниз

int goprvlnk(); - Возврат сдвинутого указателя на шаг назад

void insnxtline(char\*str); - Вставить строку по указателю pNext

void insnxtsection(char\*str);

void insdwnline(char\*str); - Вставить строку по указателю pDown

void delnxt(); - Удалить строку по указателю pNext

void deldwn(); - Удалить строку по указателю pDown

void MarkCurr() { pCurr->flag = true; }

TLink\* cpyrec(TLink\*tmp); - Рекурсивное копирование

TLink\* cpy(); - Возвращает результат cpyrec для указателя pFirst

void reset();

bool isend();

void gonext();

TLink\*ReadRec(std::ifstream& file); - Рекурсивное чтение из файла

void Read(char \*fn);

void viewText(TLink \*ptr); - Вывод текста в консоль

void Save(char\* fn);

void SaveText(TLink\* tmp, std::ofstream& f); - Сохранение текста в файл.

# Описание алгоритмов

**Алгоритм очистки памяти TLink::MemClean(TText &t)**

1. Сначала обходим все элементы экземпляра класса TText, передаваемого в функцию и поднимаем у каждого флаг.
2. Обходим все свободные элементы памяти и поднимаем в каждом флаг.
3. Обходим все элементы массива памяти и если флаг не поднят, значит элемент потерян при удалении отцовского, значит добавляем его к свободным элементам.

**Алгоритм рекурсивного чтения из файла \*TText::ReadRec(std::ifstream& file)**

TLink \*tmp, \*first = nullptr;

Пока не конец файла:

1. Если строка равна ‘}’, то выходим из функции.
2. Иначе если строка равна ‘{‘, то tmp-> pDown = ReadRec(file).
3. Иначе если first не указывает никуда, то tmp->pNext = new TLink(buf) (создаем элемент, со считанной строкой), tmp = first.
4. Иначе tmp->pNext = new TLink(buf) (Создаем элемент по указателю pNext со считанной строкой) и сдвигаем указатель по pNext.

Возвращаем указатель first.

**Алгоритм вывода текста в консоль**

void TText::View()

{

lvl = 0;

viewText(pFirst);

}

void TText::viewText(TLink \*ptr)

{

if (ptr == pCurr) {

HANDLE Console = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(Console, (WORD)((0 << 4) | 4));

std::cout << "\_";

Console = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(Console, (WORD)((0 << 4) | 15));

}

if (ptr) {

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

std::cout << "\t";

std::cout << ptr->getStr() << std::endl;

lvl++;

viewText(ptr->getPDown());

lvl--;

viewText(ptr->getPNext());

}

}

Идея алгоритма состоит в работе с переменной lvl, которая показывает сколько отступов необходимо сделать для правильного форматирования. Переменная lvl инкрементируется при рекурсивном вызове ViewText с передачей указателя pDown и дикрементируется при вызове ViewText с передачей указателя pNext.

# Результаты

Мне удалось выполнить все поставленные задачи.