МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ»
ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Хеш-таблицы с цепочками: вставка и исключение. Демонстрация

Студент гр. 9303 _____ Эйсвальд М.И.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент: Эйсвальд М.И.

Группа: 9303

Тема работы: Хеш-таблицы с цепочками: вставка и исключение. Демонстрация

Исходные данные: вариант 23: требуется реализовать демонстрацию операций вставки и удаления в хеш-таблице с цепочками. Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями).

Содержание пояснительной записки: аннотация, содержание, введение, описание структур данных и их методов, алгоритмов, описание интерфейса пользователя, результаты тестирования, заключение, список использованных источников, приложение.

Предполагаемый объём пояснительной записки: не менее 15 страниц.

Дата выдачи задания: 06.11.2020

Дата сдачи реферата: 23.12.2020

Дата защиты реферата: 25.12.2020

Студент гр. 9303 _____ Эйсвальд М.И.

Преподаватель _____ Филатов Ар.Ю.

АННОТАЦИЯ

В рамках данной работы были изучены операции вставки и удаления для хеш-таблицы с цепочками; самостоятельно реализована хеш-таблица с цепочками на основе лекционного материала. Результатом работы стала программа на языке C++, наглядно демонстрирующая процесс выполнения этих операций с экземпляром структуры данных.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является разработка программы, подробно демонстрирующей операции вставки в хеш-таблицу с цепочками и удаления из неё.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучение хеш-таблицы как структуры данных;
2. Разработка класса хеш-таблицы с цепочками, реализующего весь требуемый функционал;
3. Разработка интерфейса пользователя;
4. Интеграция частей проекта;
5. Сборка программы;
6. Тестирование кода и правка ошибок;
7. Написание отчёта и документации к программе.

1. ОПИСАНИЕ КОДА ПРОГРАММЫ

1.1. Описание структур данных и их методов, алгоритмов.

В программе используются следующие самостоятельно реализованные шаблонные структуры данных: `Symbol<Elem>` и `HashTable<Elem>`, где для типа `Elem` определены операции: приведение к целому числу, вывод в поток, присваивание элементу того же типа, сравнение с элементом того же типа.

`HashTable` — класс хеш-таблицы с цепочками, содержащий основной функционал. Содержит в качестве полей саму хеш-таблицу (массив односвязных списков *символов* — `Symbol`) и таблицу символов — массив *символов*. Таким образом, каждая цепочка представляет собой линейный список символов, который реализован на базе вектора. Вспомогательные поля класса включают в себя индекс первого свободного элемента таблицы символов `free_index` и размер хеш-таблицы `size`.

Класс `Symbol` — структура данных, представляющая элемент таблицы символов (или, что то же самое, элемент цепочки хеш-таблицы). Класс включает в себя поля: `key` — собственно хранимое значение; `count` — количество вхождений ключа; `deleted` — флаг, говорящий, помечена ли ячейка как удалённая; `link` — указатель на следующий элемент цепочки.

Структура `Symbol` не имеет методов, кроме конструктора. Методы класса `HashTable` описаны ниже.

Таблица символов и хеш-таблица выделяются динамически, поэтому в конструкторе `HashTable` по входному параметру — длине поступившей строки, которая, очевидно, равна максимально возможному количеству различных символов — подсчитывается количество требуемой памяти. При этом размер таблицы подбирается так, что количество ячеек больше количества введённых символов, так как чем больше «забита» хеш-таблица, тем менее она эффективна (хотя к таблице с цепочками это относится в меньшей степени).

Используемая хеш-функция стандартная: остаток от деления элемента на размер таблицы. Очевидно, значение хеша конкретного элемента зависит от размера таблицы. Поскольку программа предназначена для демонстрации, специфической хеш-функции и не требуется.

Вспомогательный метод `find()` ищет элемент в таблице. Если элемент найден, возвращается указатель на него; в противном случае возвращается нулевой указатель. Процесс поиска сопровождается пояснениями.

Метод `add()` добавляет элемент в хеш-таблицу, также сопровождая процесс поянениями. Если элемент уже присутствует в таблице (т. е. метод `find()` вернул ненулевой указатель), то увеличивается поле `count` соответствующего элемента. В противном случае производится исследование цепочки, соответствующей значению хеша элемента, и добавление элемента в первую «удалённую» ячейку при наличии таковых или в конец цепочки при их отсутствии.

Метод `remove()` удаляет элемент из хеш-таблицы, также сопровождая процесс поянениями. Алгоритм удаления проще алгоритма добавления: если элемент найден, уменьшается счётчик его значений. Если после этого счётчик обнулился, элемент помечается как удалённый — впоследствии сюда можно будет вставить другой элемент. Если же элемент не найден, удалять просто нечего.

Метод `printChains()` печатает хеш-таблицу, используя перегруженные операторы вывода в поток для элемента таблицы и для цепочки элементов.

Метод `printInfo()` печатает не только хеш-таблицу, но и детали её внутренней реализации — заполненную часть таблицы символов.

1.2. Описание вспомогательных функций.

Фактически в работе используются классы `Symbol<char>` и `HashTable<char>`, в связи с чем вывод в поток элемента был реализован как вывод символа в кавычках, если символ имеет графическое представление, и вывод простой графической интерпретации символа, если символ не имеет графического представления. Вывод в поток всей цепочки — также с помощью оператора << — реализован как вывод ввсех элементов цепочки.

Функция `main()` отвечает за взаимодействие с пользователем: парсит ввод, создаёт и удаляет экземпляры хеш-таблиц, вызывает основные методы хеш-таблицы. Подробнее см. в разделе 2.

Функция `color()`, меняющая цвет выводимого текста, с помощью директив условной компиляции определяется по-разному в зависимости от

платформы: на Windows используется API из заголовочного файла `<windows.h>`, на Linux — в консоль выводится escape-код, отвечающий за определённый цвет.

2. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

2.1. Описание функции main().

Функция `main()` отвечает за взаимодействие с вводом пользователя, создание выходного файла, открытие нужного файла с входными данными. Цикл построчного чтения (из файла или с консоли) и обработки входных данных продолжается, пока не будет встречен символ конца файла `EOF`. Если первый символ считанной строки является специальным, строка интерпретируется как команда. В противном случае выполняется действие по умолчанию — уничтожение старой хеш-таблицы и создание новой, содержащей все символы из переданной строки Во всех случаях полученная C-style строка обрабатывается посимвольно, пока не будет встречен нуль-терминатор.

2.2. Руководство пользователя.

Для сборки программы из исходного кода необходим компилятор языка C++. Для сборки введите в командной строке:

```
g++ -std=c++11 cw/main.cpp cw/elements.cpp,
```

где пути к .cpp файлам заменены фактическими. Возможно использование другого компилятора, но требуется использовать стандарт не старше C++11. Теперь скомпилированная программа готова к запуску.

При запуске программа требует ввести имя файла с входными данными. Если Вы хотите считать данные из файла, введите полный или относительный (относительно Вашего текущего расположения) путь к текстовому файлу. Если файл открыт успешно, его содержимое будет прочитано и интерпретировано программой. По достижении конца файла программа завершится.

Если Вы хотите вводить данные с консоли, нажмите «Ввод» («Return»). На экране появится сообщение о том, что файл не открыт и ожидается ввод с консоли.

Входная строка для программы может быть четырёх типов:

- Строка, начинающаяся с любого символа, кроме `'+'`, `'-'` или `'?'`, считается командой создать новую хеш-таблицу соответствующего размера и поочерёдно добавить в неё все символы строки;
- Строка, начинающаяся символом `'+'`, считается командой добавить все следующие за ним символы в существующую хеш-таблицу;
- Строка, начинающаяся символом `'-'`, считается командой удалить все последующие элементы из хеш-таблицы;
- Строка, начинающаяся символом `'?'`, считается командой распечатать подробную информацию о текущем состоянии хеш-таблицы: саму хеш-таблицу со всеми цепочками и все заполненные ячейки таблицы символов. Обратите внимание, что сама хеш-таблица и так печатается после каждого добавления или удаления элемента, но хеш-таблица — нет. Все символы, следующие за символом команды, игнорируются.

Чтобы корректно завершить ввод, введите символ конца файла. В зависимости от платформы за него товечают разные комбинации клавиш.

Читайте подробный вывод, чтобы понять логику и порядок производимых с экземпляром структуры данных операций. Как при вводе с консоли, так и при вводе из файла весь вывод дублируется в файл «output.txt». Анализируйте выводимый в файл протокол для получения подробной информации.

При получении сообщения об ошибке прочитайте его внимательно: как правило, сообщения об ошибках содержат информацию о способах их решения. Сообщение об ошибке значит, что операция выполнена не была.

Предупреждения можно игнорировать. Предупреждения выводятся при добавлении или удалении пустого множества элементов, а также при обнаружении во входной строке символов не из ASCII (предполагается, что программа работает с однобайтными символами). Экзотические символы, скорее всего, будут обработаны как несколько символов.

Вся прочая информация, которая может потребоваться пользователю, выводится самой программой.

3. ТЕСТИРОВАНИЕ

Результаты тестирования см. в табл. ниже. Многие строки вывода были удалены для удобства чтения.

Таблица 1 – Создание таблицы

Входные данные	Окончательный вывод

`hash`	`hash = 0: nil`
	`hash = 1: nil`
	`hash = 2: nil`
	`hash = 3: 's'(x1) -> nil`
	`hash = 4: nil`
	`hash = 5: nil`
	`hash = 6: 'h'(x2) -> 'a'(x1) -> nil`

Таблица 2 – Проверка обработки операций с несуществующей таблицей

Входные данные	Вывод
`+`	`(!)Rejected: No hash table has been created yet to add, remove, or print detailed info. To create a new hash table, type any string not starting with: '+' '-' '?'`
`-`	`(!)Rejected: No hash table has been created yet to add, remove, or print detailed info. To create a new hash table, type any string not starting with: '+' '-' '?'`

Таблица 3 – Вставка и удаление

Входные данные	Окончательный вывод
+ab	Element is present! Element is not present! hash = 0: 'b'(x1) -> nil hash = 1: nil hash = 2: nil hash = 3: 's'(x1) -> nil hash = 4: nil hash = 5: nil hash = 6: 'h'(x2) -> 'a'(x2) -> nil
-ab	Element is present! Element is present! hash = 0: [deleted](x0) -> nil hash = 1: nil hash = 2: nil hash = 3: 's'(x1) -> nil hash = 4: nil hash = 5: nil hash = 6: 'h'(x2) -> 'a'(x1) -> nil
+i	Element is not present! hash = 0: 'i'(x1) -> nil hash = 1: nil hash = 2: nil hash = 3: 's'(x1) -> nil hash = 4: nil hash = 5: nil hash = 6: 'h'(x2) -> 'a'(x1) -> nil
-1	Element is not present! Nothing to remove. hash = 0: 'i'(x1) -> nil hash = 1: nil hash = 2: nil hash = 3: 's'(x1) -> nil hash = 4: nil hash = 5: nil hash = 6: 'h'(x2) -> 'a'(x1) -> nil
+	Warning: no elements to add.
-	Warning: no elements to remove.

ЗАКЛЮЧЕНИЕ

В процессе работы была разработана и протестирована несложная программа, демонстрирующая операции вставки в хеш-таблицу с цепочками и удаления из неё с текстовыми пояснениями. По результатам тестирования программы можно сказать, что иллюстрации и пояснения получились достаточно наглядными и подробными: в таблицах с результатами тестов даже пришлось приводить лишь конечный результат работы, поскольку ввод даже одного слова приводит к печати лога впечатляющих размеров. Можно утверждать, что поставленные задачи выполнены. Более того, в процессе выполнения работы были решены и задачи, не входящие в изначально сформулированный список: например, реализация выделения цвета текстом для разных платформ с помощью директив условной компиляции (см. файл colorout.h в приложении А). Таким образом, сформулированная цель работы достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] C++ reference: Description of the most important classes, functions and objects of the Standard Language Library, with descriptive fully-functional short programs as examples: http://www.cplusplus.com/reference (дата обращения: 21.12.2020).

[2] Методические указания к лабораторным работам, практическим занятиям и курсовой работе по дисциплине «Алгоритмы и структуры данных»учеб.-метод. пособие / сост.: С.А Ивановский, Т.Г. Фомичева, О.М. Шолохова. СПб. 2017. 88 с.

[3] Викиконспекты: База электронных конспектов по дисциплине «Алгоритмы и структуры данных» университета ИТМО: https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B_%D0%B8_%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85 (дата обращения: 19.12.2020).

[4] Викиконспекты: «разрешение коллизий» на Викиконспектах — базе электронных конспектов университета ИТМО: https://neerc.ifmo.ru/wiki/index.php?title=%D0%A0%D0%B0%D0%B7%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BE%D0%BB%D0%BB%D0%B8%D0%B7%D0%B8%D0%B9 (дата обращения: 18.12.2020).

[5] YellowAfterLife: Статьи и записи о программировании: https://ru.yal.cc/cpp-colored-text-via-winapi/ (дата обращения: 23.12.2020).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: elements.h

```
#ifndef ELEMENTS_H
#define ELEMENTS_H

#include <cctype>

#include <iostream>

using namespace std;

template<typename Elem>
struct Symbol{
    Symbol(): link(nullptr), count(1), deleted(false) {}

    Elem key;
    unsigned count;
    Symbol* link;
    bool deleted;
};

ostream& operator <<(ostream& out, Symbol<char>& symb);

ostream& operator <<(ostream& out, Symbol<char>* symb);

#endif // ELEMENTS_H
```

Название файла: elements.cpp

```
#include "elements.h"

ostream& operator <<(ostream& out, Symbol<char>& symb){
    if(symb.deleted){
        out << "[deleted]";
        return out;
    }

    char key = symb.key;

    out << "'";
    if((signed char)(key) < 0){
        out << "~";
        key = ~key;
    }
    if(isgraph(key)) out << key;
    else{
        switch(key){
        case ' ':
            out << "<SPACE>";
            break;
        case '\t':
            out << "<TAB>";
            break;
        default:
            out << "^" << (char)(key + '@');
```

```
        }
    }
    out << "'";
    return out;
}

ostream& operator <<(ostream& out, Symbol<char>* symb){
    while(symb){
        out << *symb << "(x" << symb->count << ") -> ";
        symb = symb->link;
    }
    out << "nil\n";
    return out;
}
```

Название файла: hashtable.h

```
#ifndef HASHTABLE_H
#define HASHTABLE_H

#include <fstream>
#include <iostream>
#include <utility>

#include "elements.h"
#include "colorout.h"

using namespace std;

#define SCALE 10

template<typename Elem>
class HashTable{
public:

HashTable(int length, ofstream& out): free_index(0), out(out){
        size = length*11/10 + 3;
        symb_table = new Symbol<Elem> [SCALE*size];
        data = new Symbol<Elem>* [size];
        for(int i = 0; i < size; ++i) data[i] = nullptr;
}

~HashTable(){
        delete [] data;
        delete [] symb_table;
}

int hash(Elem elem){
        return abs(elem % size);
}

Symbol<Elem>* find(Elem elem){
        cout << "Now searching for '" << elem << "' in the table.\n";
        out << "Now searching for '" << elem << "' in the table.\n";
        cout << "Hash('" << elem << "') = " << hash(elem) << "\n";
        out << "Hash('" << elem << "') = " << hash(elem) << "\n";
        Symbol<Elem>* symb = data[hash(elem)];
        cout << "Exploring the chain #" << hash(elem) << ":\n" << symb;
        out << "Exploring the chain #" << hash(elem) << ":\n" << symb;
        while(symb){
```

```
            cout << "Found " << *symb << "\n";
            out << "Found " << *symb << "\n";
            if(elem == symb->key && !symb->deleted){
                    cout << "Element is present!\n";
                    out << "Element is present!\n";
                    return symb;
            }
            symb = symb->link;
        }
        cout << "The end of the chain has been reached.\n";
        out << "The end of the chain has been reached.\n";
        cout << "Element is not present!\n";
        out << "Element is not present!\n";
        return nullptr;
}

void add(Elem elem){
        if((signed char)(elem) < 0){
            color(FOREGROUND_YELLOW);
            cout << "Warning: unusual symbol detected. " <<
                "It may be a part of a long (UTF-16 or UTF-32) symbol, " <<
                "but it will be treated as a separate character.\n";
            out << "Warning: unusual symbol detected. " <<
                      "It may be a part of a long (UTF-16 or UTF-32) symbol, " <<
                           "but it will be treated as a separate character.\n";
            color(FOREGROUND_WHITE);
        }
        cout << "Adding '" << elem << "'\n";
        out << "Adding '" << elem << "'\n";


        Symbol<Elem>* symb = this->find(elem);
        if(symb){
            cout << "Increasing the 'count' value.\n";
            out << "Increasing the 'count' value.\n";
            ++symb->count;
            return;
        }

        cout << "Adding a new element to the previously examined chain.\n";
        out << "Adding a new element to the previously examined chain.\n";
        symb = data[hash(elem)];
        if(!symb){
            //! overflow?
            if(free_index >= SCALE*size){
                color(FOREGROUND_RED);
                cout << "(!)Error: overflow in vector-based symbol table.\n";
                out << "(!)Error: overflow in vector-based symbol table.\n";
                cout << "Operation terminated.\n";
                out << "Operation terminated.\n";
                color(FOREGROUND_WHITE);
            cout << "Try to create a larger table by passing a longer string.\n";
             out << "Try to create a larger table by passing a longer string.\n";
                cout << "Chain remains unchanged: " << data[hash(elem)];
                out << "Chain remains unchanged: " << data[hash(elem)];
                return;
            }
            symb_table[free_index].key = elem;
            data[hash(elem)] = &(symb_table[free_index++]);
            cout << "Chain after adding:\n" << data[hash(elem)];
```

```
            out << ”Chain after adding:\n” << data[hash(elem)];
            return;
        }
        while(symb->link && !symb->deleted){
            symb = symb->link;
        }

        if(symb->deleted){
            cout << ”Encountered a ’deleted’ cell. Inserting the new ”;
           cout << ”element here instead of adding to the end of the chain.\n”;
            out << ”Encountered a ’deleted’ cell. Inserting the new ”;
           out << ”element here instead of adding to the end of the chain.\n”;
            symb->key = elem;
            symb->count = 1;
            symb->deleted = false;
        }
        else{//! overflow?
            if(free_index >= SCALE*size){
                color(FOREGROUND_RED);
                cout << ”(!)Error: overflow in vector-based symbol table.\n”;
                cout << ”Operation terminated.\n”;
                out << ”(!)Error: overflow in vector-based symbol table.\n”;
                out << ”Operation terminated.\n”;
                color(FOREGROUND_WHITE);
            cout << ”Try to create a larger table by passing a longer string.\n”;
                cout << ”Chain remains unchanged: ” << data[hash(elem)];
             out << ”Try to create a larger table by passing a longer string.\n”;
                out << ”Chain remains unchanged: ” << data[hash(elem)];
                return;
            }
            symb_table[free_index].key = elem;
            symb->link = &(symb_table[free_index++]);
        }

        cout << ”Chain after adding:\n” << data[hash(elem)];
        out << ”Chain after adding:\n” << data[hash(elem)];
}

void remove(Elem elem){
        Symbol<Elem>* symb = this->find(elem);
        if(!symb){
            cout << ”Nothing to remove.\n”;
            out << ”Nothing to remove.\n”;
            return;
        }
     cout << ”Decreasing ’count’ value: ” << symb->count << ” >> ” << symb->count-1;
     out << ”Decreasing ’count’ value: ” << symb->count << ” >> ” << symb->count-1;
        --symb->count;
        cout << ”\n”;
        out << ”\n”;
        if(symb->count == 0){
            cout << ”Count == 0 => deleting the element.\n”;
            out << ”Count == 0 => deleting the element.\n”;
            symb->deleted = true;
        }
}

void printChains(){
        cout << ”Chains in hash table of size ” << size << ”:\n\n”;
        out << ”Chains in hash table of size ” << size << ”:\n\n”;
```

```
        for(int i = 0; i < size; ++i){
            cout << "hash = " << i << ": ";
            out << "hash = " << i << ": ";
            cout << data[i];
            out << data[i];
        }

        cout << "\n";
        out << "\n";
}

void printInfo(){
    cout << "\n";
    out << "\n";
    cout << "--------------\n";
    out << "--------------\n";
    printChains();
    cout << "Vector-based symbol table:\n";
    out << "Vector-based symbol table:\n";
    for(int i = 0; i < free_index; ++i){
        cout << "index = " << i << ": " << symb_table[i]
        << "(x" << symb_table[i].count << ")\n";
        out << "index = " << i << ": " << symb_table[i]
        << "(x" << symb_table[i].count << ")\n";
    }
    cout << "--------------\n";
    cout << "\n";
    out << "--------------\n";
    out << "\n";
}

private:
    Symbol<Elem>** data;
    Symbol<Elem>* symb_table;
    int free_index;
int size;
    ofstream& out;
};

#endif
```

Название файла: main.cpp

```
#include <cstring>

#include <iostream>
#include <fstream>
#include <string>

using namespace std;

#include "hashtable.h"
#include "colorout.h"

#define OUTFILE "output.txt"
#define ADD '+'
#define REMOVE '-'
#define INFO '?'
#define SPECIAL "+-?"
```

```
int main(){
    HashTable<char>* ht = nullptr;
const char* source;

std::ofstream out(OUTFILE);
if(!out.is_open()){
std::cout << "Failed to create output file. Exiting...\n";
return 0;
}
std::string str;
std::cout << "Specify input file name: ";
getline(std::cin, str);
std::ifstream infile(str.c_str());
if(!infile.is_open()){
std::cout << "Failed to open file. Expecting input from console...\n";
        std::cout << "Enter EOF (e.g. Ctrl+D on Linux) to finish.\n";
}
do{
getline( (infile.is_open() ? infile : std::cin) , str);
if( (infile.is_open() && infile.eof()) || (std::cin.eof()) ) break;
        std::cout << "Read line: \"" << str << "\"\n";
out << "Read line: \"" << str << "\"\n";
        if(!str.length()) continue;
source = str.c_str();

        if(strchr(SPECIAL, source[0])){
            //received a command
            if(!ht){
                color(FOREGROUND_RED);
             cout << "(!)Rejected: No hash table has been created yet to add, " <<
                        "remove, or print detailed info.\n";
              out << "(!)Rejected: No hash table has been created yet to add, " <<
                        "remove, or print detailed info.\n";
                cout << "To create a new hash table, type any " <<
                        "string not starting with: ";
                out << "To create a new hash table, type any " <<
                        "string not starting with: ";
                for(const char* ptr = SPECIAL; *ptr; ++ptr){
                    cout << "'" << *ptr << "' ";
                    out << "'" << *ptr << "' ";
                }
                cout << "\n";
                out << "\n";
                color(FOREGROUND_WHITE);
                continue;
            }
            switch(source[0]){
            case ADD:
                ++source;
                if(!(*source)){
                    color(FOREGROUND_YELLOW);
                    cout << "Warning: no elements to add.\n";
                    out << "Warning: no elements to add.\n";
                    color(FOREGROUND_WHITE);
                }
                while(*source){
                    ht->add(*source);
                    ht->printChains();
                    ++source;
```

```
                }
                continue;
            case REMOVE:
                ++source;
                if(!(*source)){
                    color(FOREGROUND_YELLOW);
                    cout << ”Warning: no elements to remove.\n”;
                    out << ”Warning: no elements to remove.\n”;
                    color(FOREGROUND_WHITE);
                }
                while(*source){
                    ht->remove(*source);
                    ht->printChains();
                    ++source;
                }
                continue;
            case INFO:
                cout << ”Printing detailed info...\n”;
                out << ”Printing detailed info...\n”;
                ht->printInfo();
                continue;
            }
        }

        delete ht;
        ht = new HashTable<char>(str.length(),out);
        ht->printChains();

while(*source){
            ht->add(*source);
            ht->printChains();
++source;
}
std::cout << ”\n”;
out << ”\n”;
}while((infile.is_open() && !infile.eof()) ||
 (!(infile.is_open()) && !std::cin.eof()) );

    cout << ”[Quit]\n”;
    delete ht;
return 0;
}
```

Название файла: colorout.h

```
#ifndef COLOROUT_H
#define COLOROUT_H

#ifdef _WIN32
#define ALLOW_COLOR
#elif __linux__
#define ALLOW_COLOR
#endif

#ifdef ALLOW_COLOR
#ifdef _WIN32
    #include <windows.h>
    #define FOREGROUND_YELLOW (FOREGROUND_RED|FOREGROUND_GREEN|
    FOREGROUND_INTENSITY)
    #define FOREGROUND_WHITE (FOREGROUND_RED|FOREGROUND_GREEN|
```

```
    FOREGROUND_BLUE|FOREGROUND_INTENSITY)
    void color(int color){
        HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
        SetConsoleTextAttribute(handle, color);
    }
#else
    #define FOREGROUND_WHITE "\033[0m"
    #define FOREGROUND_RED "\033[0;31m"
    #define FOREGROUND_YELLOW "\033[0;33m"
    void color(const char* color){
        cout << color;
    }
#endif
#else
    #define FOREGROUND_WHITE 1
    #define FOREGROUND_RED 1
    #define FOREGROUND_YELLOW 1
    void color(int){}
#endif
#endif // COLOROUT_H
```