МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра прикладной математики и информатики

ОТЧЁТ

по лабораторной работе №5 «Рефакторинг хэш-таблицы»

по дисциплине

Структуры данных

(наименование дисциплины)

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Санников А.Н.

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Зырянов Е.А.

(подпись) (фамилия, и.,о.)

22-ПМ-1

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Задание на лабораторную работу:*** Осуществить рефакторинг лабораторной работы №5 «Хэш-таблица».

**Задание проверяемой лабораторной работы**: h-функция - умножения, колизии — квадратичное опробирование, ключ — строка.

|  |  |  |
| --- | --- | --- |
| Ошибка | Было | Стало |
| Некорректная запись аргументов функций | Имелись функции, аргументы которых были записаны в столбцы | Аргументы записаны в строку |
| Отсутствие единого стиля кода | Неправильный змеиный регистр | Исправленный змеиный регистр |
| Отсутствие комментариев | Нет комментариев на переменные, которые непонятно, что могут значить | Добавления комментариев |
| Отсутствие пробелов в операторах выражения | В операторах отсутствовали пробелы | Добавлены пробелы |

До рефакторинга:

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

struct value {

string key;

value\* next;

string data;

value(string k, string d) : key(k), next(nullptr), data(d) {}

~value() {

value\* ptr = this;

value\* prev;

while (ptr) {

prev = ptr;

ptr = ptr->next;

delete prev;

}

}

void add(string k, string d) {

value\* tmp = this;

while (tmp->next) {

tmp = tmp->next;

}

tmp->next = new value(k, d);

}

void del(string k) {

value\* tmp = this;

value\* prev = nullptr;

while (tmp->next and

tmp→key.compare(k))

{

prev = tmp;

tmp = tmp->next;

}

if (tmp->key.compare(k)) {

return;

}

if (tmp == this and !tmp->next) {

delete this;

return;

}

else if (tmp == this) {

value\* tmp\_next = next;

key = next->key;

data = next->data;

next = next->next;

delete tmp\_next;

return;

}

else if (tmp->next) {

prev->next = tmp->next;

delete tmp;

return;

}

else if (tmp) {

prev->next = nullptr;

delete tmp;

return;

}

}

};

struct hash\_table {

private:

value\*\* map;

size\_t size;

size\_t elem;

int hash(string key) {

return (static\_cast<int>(key[key.length() - 1]) + static\_cast<int>(key[0])) % size;

}

void rehash(int new\_size)

{

int tmp\_size = size;

size = new\_size;

value\*\* new\_map = new value \* [size];

for (size\_t i = 0; i < size; i++)

{

new\_map[i] = nullptr;

}

for (size\_t i = 0; i < tmp\_size; i++)

{

if (map[i]) {

value\* ptr = map[i];

while (ptr) {

int h = hash(ptr->key);

if (new\_map[h]) {

new\_map[h]->add(ptr->key, ptr->data);

}

else {

new\_map[h] = new value(ptr->key, ptr->data);

}

ptr = ptr->next;

}

}

}

map = new\_map;

}

public:

hash\_table() : size(0), elem(0), map(nullptr) {}

void insert(string k, string v) {

if (!size) {

size = 1;

map = new value\* [size];

map[0] = new value(k, v);

}

else {

int tmp = hash(k);

if (!map[tmp])

{

map[tmp] = new value(k, v);

}

else {

map[hash(k)]->add(k, v);

}

}

elem += 1;

if (elem > size)

{

rehash(size \* 2);

}

}

void print() {

for (size\_t i = 0; i < size; i++)

{

printf("%d) ", i);

if (map[i]) {

value\* tmp = map[i];

while (tmp)

{

printf("[%s : %s] -> ", tmp->key.c\_str(), tmp->data.c\_str());

tmp = tmp->next;

}

}

cout << endl;

}

}

string& operator[](string key) {

if (!size) {

rehash(1);

}

if (elem == size) {

rehash(size \* 2);

}

int h = hash(key);

value\* ptr = map[h];

if (map[h])

{

while (ptr) {

if (!ptr->key.compare(key)) {

return ptr->data;

}

ptr = ptr->next;

}

}

insert(key, "");

return (\*this)[key];

}

};

int main() {

hash\_table map;

ifstream f("names.txt");

for (size\_t i = 0; !f.eof(); i++)

{

char line[100];

f.getline(line, 100);

map[line] = i;

}

map.print();

return 0;

}

После рефактора:

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

// Структура, представляющая узел в связанном списке, используемом для метода цепочек

struct Value {

string Key; // Ключ узла

Value\* Next; // Указатель на следующий узел в списке

string Data; // Данные, связанные с ключом

// Конструктор для инициализации узла с ключом и данными

Value( string k, string d ) : Key( k ), Next( nullptr ), Data( d ) {}

// Деструктор для рекурсивного удаления всех узлов в списке

~Value() {

Value\* Ptr = this;

Value\* Prev;

while ( Ptr ) {

Prev = Ptr;

Ptr = Ptr->Next;

delete Prev;

}

}

// Функция для добавления нового узла в конец списка

void Add( string k, string d ) {

Value\* Tmp = this;

while ( Tmp->Next ) {

Tmp = Tmp->Next;

}

Tmp->Next = new Value( k, d );

}

// Функция для удаления узла с заданным ключом

void Del( string k ) {

Value\* Tmp = this;

Value\* Prev = nullptr;

while ( Tmp->Next and Tmp->Key.compare( k ) ) {

Prev = Tmp;

Tmp = Tmp->Next;

}

// Проверка, найден ли узел с ключом k

if ( Tmp->Key.compare( k ) ) {

return;

}

// Удаление узла с учетом его положения в списке

if ( Tmp == this and !Tmp->Next ) {

delete this;

return;

}

else if ( Tmp == this ) {

Value\* Tmp\_next = Next;

Key = Next->Key;

Data = Next->Data;

Next = Next->Next;

delete Tmp\_next;

return;

}

else if ( Tmp->Next ) {

Prev->Next = Tmp->Next;

delete Tmp;

return;

}

else if ( Tmp ) {

Prev->Next = nullptr;

delete Tmp;

return;

}

}

};

// Структура, представляющая хеш-таблицу

struct Hash\_Table {

private:

Value\*\* Map; // Указатель на массив указателей на узлы Value ( связанные списки )

size\_t Size; // Размер хеш-таблицы

size\_t Elem; // Количество элементов в хеш-таблице

// Хеш-функция для вычисления индекса по заданному ключу

int Hash( string Key ) {

return ( static\_cast<int>( Key[ Key.length() - 1 ] ) + static\_cast<int>( Key[ 0 ] ) ) % Size;

}

// Функция для изменения размера и перехеширования хеш-таблицы

void Rehash( int New\_Size ) {

int Tmp\_Size = Size;

Size = New\_Size;

Value\*\* New\_Map = new Value \* [ Size ];

for ( size\_t i = 0; i < Size; i++ ) {

New\_Map[ i ] = nullptr;

}

// Перехеширование элементов из старой таблицы в новую

for ( size\_t i = 0; i < Tmp\_Size; i++ ) {

if ( Map[ i ] ) {

Value\* Ptr = Map[ i ];

while ( Ptr ) {

int h = Hash( Ptr->Key );

if ( New\_Map[ h ] ) {

New\_Map[ h ]->Add( Ptr->Key, Ptr->Data );

}

else {

New\_Map[ h ] = new Value( Ptr->Key, Ptr->Data );

}

Ptr = Ptr->Next;

}

}

}

Map = New\_Map;

}

public:

// Конструктор для инициализации хеш-таблицы

Hash\_Table() : Size( 0 ), Elem( 0 ), Map( nullptr ) {}

// Функция для вставки пары ключ-значение в хеш-таблицу

void Insert( string k, string v ) {

// Если таблица пуста, создается новая с одним элементом

if ( !Size ) {

Size = 1;

Map = new Value \* [ Size ];

Map[ 0 ] = new Value( k, v );

}

else {

// Вычисление хеша для ключа

int Tmp = Hash( k );

// Если в позиции Tmp уже есть элемент, добавляем новый элемент в связанный список

if ( !Map[ Tmp ] ) {

Map[ Tmp ] = new Value( k, v );

}

else {

Map[ Hash( k ) ]->Add( k, v );

}

}

Elem += 1;

// Проверка на переполнение таблицы и перехеширование при необходимости

if ( Elem > Size ) {

Rehash( Size \* 2 );

}

}

// Функция для вывода содержимого хеш-таблицы

void Print() {

for ( size\_t i = 0; i < Size; i++ ) {

printf( "%d ) ", i );

if ( Map[ i ] ) {

Value\* Tmp = Map[ i ];

while ( Tmp ) {

printf( "[ %s : %s ] -> ", Tmp->Key.c\_str(), Tmp->Data.c\_str() );

Tmp = Tmp->Next;

}

}

cout << endl;

}

}

// Перегруженный оператор[], позволяющий получить и изменить элементы в хеш-таблице

string& operator {

// Если таблица пуста, создается новая с одним элементом

if ( !Size ) {

Rehash( 1 );

}

// Проверка на переполнение таблицы и перехеширование при необходимости

if ( Elem == Size ) {

Rehash( Size \* 2 );

}

// Получение хеша для ключа

int h = Hash( Key );

Value\* Ptr = Map[ h ];

if ( Map[ h ] ) {

// Поиск элемента с заданным ключом в связанном списке

while ( Ptr ) {

if ( !Ptr->Key.compare( Key ) ) {

return Ptr->Data;

}

Ptr = Ptr->Next;

}

}

// Если элемент не найден, он добавляется в таблицу

Insert( Key, "" ); // Вставка элемента с пустым значением

}

// Возврат ссылки на данные по ключу

return ( \*this )[ Key ];

}

};

// Главная функция

int main() {

Hash\_Table Map; // Создание хеш-таблицы

ifstream f( "names.txt" ); // Открытие файла для чтения

// Чтение строк из файла и добавление их в хеш-таблицу с индексом i

for ( size\_t i = 0; !f.eof(); i++ ) {

char line[ 100 ];

f.getline( line, 100 );

Map[ line ] = i;

}

Map.Print(); // Вывод содержимого хеш-таблицы

return 0;

}