## Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа №13

По дисциплине «Основы алгоритмизации и программирования»

На тему «**Хеш-таблицы с открытой адресацией**»

Выполнила:

Студент 1 курса 8 группы

Рауба Арсений

Преподаватель: асс. Андронова М.В.

Минск, 2024

Вариант 9: Реализовать хеш-таблицу с открытой адресацией для хранения строк. Таблица должна увеличивать свой размер втрое при достижении 70% заполнения.

#include <iostream>

#include <conio.h>

#include "String.h" //доработанный под себя string

using namespace std;

class Hash {

string\* mas;

int size;

int maxSize;

int func;

int pars(string& x) {//перевод строки

int res = 0;

for (int i = 0; i < x.length(); i++) {

res += int(x[i]);

}

return res;

}

bool Empty(int x) { // проверка на пустость

return (mas[x][0] == '\0');

}

void Extend() { //функция расширение массива

string\* temp = new string[maxSize];

for (int i = 0; i < maxSize; i++) temp[i] = mas[i];

maxSize \*= 3;

func \*= 3;

delete[] mas;

mas = new string[maxSize];

for (int i = 0; i < maxSize; i++) mas[i] = "\0";

for (int i = 0; i < maxSize / 3; i++) {// перераспределение элементов в таблице под новый алгоритм размещения

if (temp[i][0] != '\0') this->insert(temp[i]);

}

delete[] temp;

return;

}

public:

Hash(int x) {// функция для создания

maxSize = func = x;

size = 0;

mas = new string[maxSize];

for (int i = 0; i < maxSize; i++) mas[i] = "\0";

}

void insert(string& x) { //функция вставки

int adress = pars(x) % func;

bool added = false;

this->size++;

if ((size \* 100 / maxSize) >= 70) //проверка на заполненность

{

Extend();

cout << "Произошло расширение\n";

}

do {

if (added = Empty(adress)) {//проверка на то, пустой элемент или нет

mas[adress] = x;

continue;

}

else adress++;

if (adress == maxSize) adress = 0;

} while (added == false);

return;

}

const char\* search(string& x) {

int adress = pars(x) % func;

do {

if (mas[adress] == x) return "Элемент найден\n";

else adress++;

if (adress == maxSize) adress = 0;

} while (this->mas[adress][0] != '\0');

}

};

int main() {

RusInput

cout << "Перед началом работы введите размер хэш-таблицы: ";

int size;

cin >> size;

Hash table(size);

char input;

string x;

do {

system("cls");

cout << "Что вы хотите сделать?\n";

cout << "1. Ввести новый элемент\n";

cout << "2. Есть ли элемент?\n";

cout << "3. Выйти (ESC)\n";

input = \_getch();

switch (input) {

case '1': cout << "Введите строку:\n";

cin >> x;

table.insert(x);

cout << "Элемент успешно внесен";

\_getch();

break;

case '2': cout << "Введите строку:\n";

cin >> x;

cout << table.search(x);

\_getch();

break;

case '3': cout << "Хорошего дня!";

exit(0);

break;

}

} while (input != char(27));

}

Содержимое файла string.h:

void out() {

for (int i = 0; this->line[i] != '\0'; i++) cout << this->line[i];

} //

void set(int x, int y) {

COORD c = { x, y };

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), c);

}

int getYcoord() {

CONSOLE\_SCREEN\_BUFFER\_INFO info\_y;

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &info\_y);

return info\_y.dwCursorPosition.Y;

}

int getXcoord() {

CONSOLE\_SCREEN\_BUFFER\_INFO info\_y;

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &info\_y);

return info\_y.dwCursorPosition.X;

}

};

ostream& operator << (ostream& os, str& x) {

x.out();

return os;

}

istream& operator >> (istream& os, str& x) {

x.get();

return os;

}

Содержимое файла string.h:

#pragma once

#include <iostream>

#include <Windows.h>

#include <conio.h>

#include <string>

using namespace std;

#define RusInput setlocale(LC\_ALL, "ru"); SetConsoleCP(1251); SetConsoleOutputCP(1251);

#define string str

class str {

public:

str() { // Конструктор по умолчанию.

line = new char[1]; // Выделяет память под один символ.

}

int pars() { // Метод для преобразования строки в число.

int i = 0;

int res = 0;

do {

if (int(line[i]) >= 48 && int(line[i]) <= 57 || line[i] == '-' || line[i] == '\0') i++;

else return 0;

} while (line[i] != '\0');

i--;

int power = 0;

do {

if (line[i] == '-')

{

res = -res;

}

else {

res += (int(line[i]) - 48) \* pow(10, power);

}

i--;

power++;

} while (i >= 0);

return res;

}

int length() {// Метод для получения длины строки.

return len - 1;

}

str& operator = (str& x) { // Перегрузка оператора присваивания для объектов класса "str"

this->len = x.len;

delete[] this->line;

this->line = new char[this->len];

for (int i = 0; x.line[i] != '\0'; i++) {

this->line[i] = x.line[i];

this->line[i + 1] = '\0';

}

return x;

}

const char\* operator = (const char\* x) {

int i = 0;

for (; x[i] != '\0'; i++);

i++;

this->len = i;

delete[] this->line;

this->line = new char[this->len];

this->line[0] = '\0';

for (int d = 0; x[d] != '\0'; d++) {

this->line[d] = x[d];

this->line[d + 1] = '\0';

}

return x;

}

str& operator + (str& x) { // Перегрузка оператора сложения для объектов класса "str".

str temp; // Создание временного объекта класса "str".

temp = \*this; // Присваивание текущего объекта временному.

this->len = this->len + x.len; // Увеличение длины строки на длину входной строки.

delete[] this->line; // Удаление текущей строки.

this->line = new char[this->len]; // Выделение памяти под новую строку.

for (int i = 0; temp[i] != '\0'; i++) this->line[i] = temp.line[i]; // Копирование символов из временного объекта в текущий.

for (int i = this->len - x.len - 1, d = 0; i < this->len; d++, i++) this->line[i] = x.line[d]; // Добавление символов из входной строки в текущий объект.

return \*this; // Возвращение текущего объекта.

}

str& operator + (char\* x) { // Перегрузка оператора сложения для C-строк.

str temp; // Создание временного объекта класса "str".

temp = \*this; // Присваивание текущего объекта временному.

this->len = this->len + strlen(x); // Увеличение длины строки на длину входной C-строки.

delete[] this->line; // Удаление текущей строки.

this->line = new char[this->len]; // Выделение памяти под новую строку.

for (int i = 0; temp[i] != '\0'; i++) this->line[i] = temp.line[i]; // Копирование символов из временного объекта в текущий.

for (int i = this->len - strlen(x), d = 0; i < this->len; d++, i++) this->line[i] = x[d]; // Добавление символов из входной C-строки в текущий объект.

return \*this; // Возвращение текущего объекта.

}

bool operator == (str& x) { // Перегрузка оператора равенства для объектов класса "str".

bool check = true;

for (int i = 0; check && x[i] != '\0' && (\*this)[i] != '\0'; i++) check = x[i] == (\*this)[i];

return check;

}

bool operator == (char\* x) {// Перегрузка оператора равенства для C-строк.

bool check = true;

for (int i = 0; check || x[i] != '\0' || (\*this)[i] != '\0'; i++) check = x[i] == (\*this)[i];

return check;

}

bool operator == (const char\* x) {// Перегрузка оператора равенства для константных C-строк.

bool check = true;

for (int i = 0; check || x[i] != '\0' || (\*this)[i] != '\0'; i++) check = x[i] == (\*this)[i];

return check;

}

bool operator != (const char\* x) {// Перегрузка оператора неравенства для константных C-строк.

return !(\*this == x);

}

bool operator != (char\* x) {// Перегрузка оператора неравенства для C-строк.

return !(\*this == x);

}

bool operator != (str& x) {// Перегрузка оператора неравенства для объектов класса "str".

return !(\*this == x);

}

char operator [] (int x) {

return this->line[x];

}

~str() {

delete[] this->line;

}

friend ostream& operator << (ostream&, str&);

friend istream& operator >> (istream&, str&);

private:

char\* line;

int len;

void get() {

int i = 0;

char input, \* templine;

line = new char[1];

do {//создаем ввод с бесконечной длинной ввода

input = \_getch();//ввод символа

if (input == char(8) && i > 0) {//удаление символа

i--;

set(i, getYcoord());

cout << ' ';

set(i, getYcoord());

}

else {

if (input == char(13)) {//обозначаем конец строки символом \0

line[i] = char(0);

continue;

}

else {//вписываем в строку символ

line[i] = input;

cout << line[i];

}

templine = new char[i + 1];//создаем массив с временной записью того, что у нас есть в основном

for (int d = 0; d < (i + 1); d++) templine[d] = line[d]; //переписываем в него

delete[] line;

i++;

line = new char[i + 1];//создаем по новой основной массив

for (int d = 0; d < i; d++) line[d] = templine[d]; //переписываем обратно то, что было

delete[] templine;

}

} while (input != char(13));

len = i + 1;

cout << endl;

}

void out() {

for (int i = 0; this->line[i] != '\0'; i++) cout << this->line[i];

}

void set(int x, int y) {

COORD c = { x, y };

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), c);

}

int getYcoord() {

CONSOLE\_SCREEN\_BUFFER\_INFO info\_y;

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &info\_y);

return info\_y.dwCursorPosition.Y;

}

int getXcoord() {

CONSOLE\_SCREEN\_BUFFER\_INFO info\_y;

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &info\_y);

return info\_y.dwCursorPosition.X;

}

};

ostream& operator << (ostream& os, str& x) {

x.out();

return os;

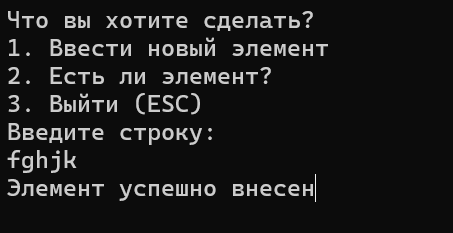
}

istream& operator >> (istream& os, str& x) {

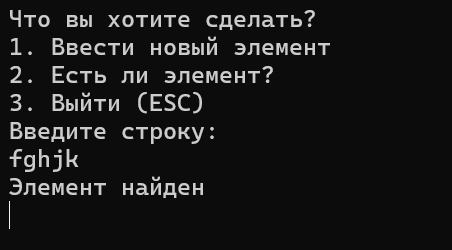
x.get();

return os;

}



Ввод нового элемента



Поиск имеющегося элемента