МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ образовательное учреждениевысшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации



**ОТЧЁТ по Лабораторной работе №2**

**по дисциплине «ПРОГРАММИРОВАНИЕ»**

**«Алгоритмы и абстрактные структуры данных»**

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АБс-222  Студент: Гатауллин Д.Р. | Преподаватель: Архипова А.Б. |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

**Цели и задачи работы**: Изучение алгоритмов работы с абстрактными структурами данных.

**Задание к работе**:

1. Обратная польская запись.

Постфиксная (или обратная польская) запись арифметического выражения — это способ записи выражений, в котором знак операции записывается после операндов. Так, операторы больше не являются неоднозначными по отношению к своим операндам. Например, запишем выражение 3+1\*4 в постфиксную форму и произведем вычисление. В постфиксной форме получим выражение 314\*+. Мы умножаем числа 4 и 1, и результат складываем с 3. Необходимо реализовать алгоритм, который вычисляет выражения в обратной польской записи с использованием стека.

1. Схожие подмножества.

Необходимо реализовать алгоритм, который должен разбить множество натуральных чисел на непересекающиеся подмножества, разница между суммами которых была бы минимальна. Вывести получившиеся подмножества и разницу их сумм.

Пример:

Множество S = {5, 8, 1, 14, 7}. Получим два подмножества {5, 14} и {8, 1, 7}. Разница между их суммами - 3.

1. Сопоставление с паттерном.

Необходимо реализовать алгоритм, который сравнивает последовательность символов с шаблоном и выводит сообщение о соответствии. Учтите, что символ "?" может заменять один символ, а "\*" - любую последовательность, в том числе нулевую.

Пример:

Последовательность "meow@stud.nstu.ru" соответствует шаблону "\*@stud.nstu.ru". Последовательность "hello" не соответствует шаблону "h?lo".

1. Удаление узла.

Дано бинарное дерево поиска, в котором хранятся уникальные целые числа. Найдите вершину с заданным значением и удалите её из дерева так, чтобы дерево осталось корректным бинарным деревом поиска.

1. Поиск правого соседа BST.

Задано двоичное дерево. Необходимо для каждого узла вывести узел, который находится "справа" от него, используя очередь. Если соседа нет, вывести null.

Пример:

22 – null, 16 – 51, 51 – null, 7 – 19, 19 – 43, 43 – 57, 57 – null

1. Метод цепочек

Реализовать хеш-таблицу, в которой коллизии будут обрабатываться при помощи метода цепочек.

**Текст программы**

==========================================================

LB2.1.cpp (PS код на СИ, я забыл просто поменять расширение)

==========================================================

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

#define MAX\_SIZE 100

typedef struct NodeHashTable {

char\* key;

char\* element;

struct NodeHashTable\* next;

struct NodeHashTable\* prev;

} NodeHashTable;

typedef struct HashTable {

NodeHashTable\*\* nodes;

int count;

} HashTable;

// Функция для инициализации хеш-таблицы

HashTable\* initHashTable() {

HashTable\* ht = (HashTable\*)malloc(sizeof(HashTable));

ht->nodes = (NodeHashTable\*\*)malloc(MAX\_SIZE \* sizeof(NodeHashTable\*));

ht->count = 0;

for (int i = 0; i < MAX\_SIZE; i++) {

ht->nodes[i] = NULL;

}

return ht;

}

// Функция для вычисления хеша

int calculateHashT(const char\* element) {

int hash = 0;

for (int i = 0; element[i] != '\0'; i++) {

hash = 31 \* hash + element[i];

}

return abs(hash) % MAX\_SIZE;

}

// Функция для добавления элемента в хеш-таблицу

void HSET(HashTable\* ht, const char\* key, const char\* value) {

int hash = calculateHashT(key);

// Создаем новый узел для хранения ключа и значения

NodeHashTable\* newNode = (NodeHashTable\*)malloc(sizeof(NodeHashTable));

newNode->key = \_strdup(key);

newNode->element = \_strdup(value);

newNode->next = NULL; // Устанавливаем указатель на следующий узел как NULL

newNode->prev = NULL; // Устанавливаем указатель на предыдущий узел как NULL

// Обработка коллизий и проверка на дубликаты ключей

NodeHashTable\* current = ht->nodes[hash]; // Получаем узел по хеш-ключу

while (current != NULL) {

if (strcmp(current->key, key) == 0) { // Если ключ уже существует

// Освобождаем память нового узла

free(newNode->key);

free(newNode->element);

free(newNode);

printf("Ключ уже существует.\n");

return;

}

if (current->next == NULL) { // Если достигли конца цепочки

break;

}

current = current->next; // Переходим к следующему узлу

}

// Добавление нового узла

if (current == NULL) { // Если цепочка пуста

ht->nodes[hash] = newNode; // Устанавливаем новый узел как начало цепочки

}

else {

current->next = newNode; // Добавляем новый узел в конец цепочки

newNode->prev = current; // Устанавливаем предыдущий узел для нового узла

}

ht->count++;

}

// Функция для получения элемента из хеш-таблицы

char\* HGET(HashTable\* ht, const char\* key) {

int hash = calculateHashT(key);

NodeHashTable\* current = ht->nodes[hash]; // Получаем узел по хеш-ключу

while (current != NULL) { // Перебираем узлы в цепочке

if (strcmp(current->key, key) == 0) { // Если ключ совпадает

return current->element;

}

current = current->next; // Переходим к следующему узлу

}

return NULL;

}

// Функция для удаления элемента из хеш-таблицы

void HDEL(HashTable\* ht, const char\* key) {

int hash = calculateHashT(key);

NodeHashTable\* current = ht->nodes[hash]; // Получаем узел по хеш-ключу

NodeHashTable\* nodeToRemove = NULL;

while (current != NULL) { // Перебираем узлы в цепочке

if (strcmp(current->key, key) == 0) { // Если ключ совпадает

nodeToRemove = current; // Устанавливаем узел для удаления

break;

}

current = current->next; // Переходим к следующему узлу

}

if (nodeToRemove != NULL) {

if (nodeToRemove->prev != NULL) { // Если у узла есть предыдущий узел

nodeToRemove->prev->next = nodeToRemove->next; // Удаляем узел из цепочки

}

else {

ht->nodes[hash] = nodeToRemove->next; // Устанавливаем следующий узел как начало цепочки

}

if (nodeToRemove->next != NULL) { // Если у узла есть следующий узел

nodeToRemove->next->prev = nodeToRemove->prev; // Устанавливаем предыдущий узел для следующего узла

}

// Освобождаем память удаляемого узла

free(nodeToRemove->key);

free(nodeToRemove->element);

free(nodeToRemove);

ht->count--;

}

else {

printf("Ключ не найден.\n");

}

}

// Функция для вывода хеш-таблицы

void printHashTable(HashTable\* ht) {

printf("Хеш-таблица:\n");

for (int i = 0; i < MAX\_SIZE; i++) {

NodeHashTable\* current = ht->nodes[i];

if (current != NULL) {

printf("Индекс %d: ", i);

while (current != NULL) {

printf("(%s, %s) ", current->key, current->element);

current = current->next;

}

printf("\n");

}

}

}

// Функция для освобождения памяти хеш-таблицы

void freeHashTable(HashTable\* ht) {

for (int i = 0; i < MAX\_SIZE; i++) {

NodeHashTable\* current = ht->nodes[i];

while (current != NULL) {

NodeHashTable\* temp = current;

current = current->next;

free(temp->key);

free(temp->element);

free(temp);

}

}

free(ht->nodes);

free(ht);

}

int main() {

setlocale(LC\_ALL, "Rus");

// Инициализация хеш-таблицы

HashTable\* ht = initHashTable();

// Добавление элементов в хеш-таблицу

HSET(ht, "AXkLn", "элемент1");

HSET(ht, "kqyzI", "элемент2");

HSET(ht, "ключ3", "элемент3");

// Вывод хеш-таблицы

printHashTable(ht);

// Получение элемента из хеш-таблицы

char\* element = HGET(ht, "kqyzI");

if (element != NULL) {

printf("Элемент с ключом 'ключ2': %s\n", element);

}

else {

printf("Элемент с ключом 'ключ2' не найден.\n");

}

// Удаление элемента из хеш-таблицы

HDEL(ht, "kqyzI");

printHashTable(ht);

// Освобождение памяти хеш-таблицы

freeHashTable(ht);

return 0;

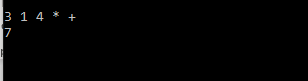
}

==========================================================

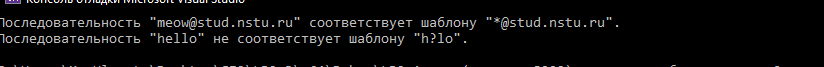
Остальной код в моём репозитории <https://github.com/SpongebobGatik/GataullinLB2>

==========================================================

**Результат работы программы**



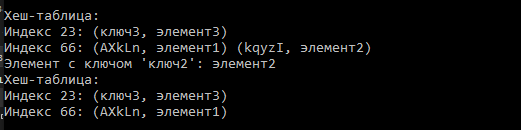




Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание





**Вывод**: Все программы корректно работают: в контрольном примере были взяты, если они были, дано из самих заданий, и ответы моих программных кодов совпадали с заданиями.