Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Лабораторная Работа №5**

**Дисциплина:** Низкоуровневое программирование

**Тема:** Программирование на Си

Выполнил

студент гр. 3530901/90003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Иванов Т.Р.

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Алексюк А.О.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург   
2021

**ОГЛАВЛЕНИЕ**

1. [ТЕХНИЧЕСКОЕ ЗАДАНИЕ 3](#_Toc58595498)

2. [МЕТОД РЕШЕНИЯ 3](#_Toc58595499)

3. ЛИСТИНГ ПРОГРАММЫ [4](#_Toc58595499)

*3.1. hashtable\_entry.h* [*4*](#_Toc58595499)

*3.2. hashtable\_struct.h* [*4*](#_Toc58595499)

*3.3. hashtable\_utils.h* [*4*](#_Toc58595499)

*3.4. hashtable\_utils.c* [*5*](#_Toc58595499)

*3.5. hashtable\_insert.h* [*5*](#_Toc58595499)

*3.6. hashtable\_insert.c* [*5*](#_Toc58595499)

*3.7. hashtable\_find.h* [*6*](#_Toc58595499)

*3.8. hashtable\_find.c* [*6*](#_Toc58595499)

*3.9. hashtable\_remove.h* [*6*](#_Toc58595499)

*3.10. hashtable\_remove.c* [*6*](#_Toc58595499)

*3.11. hashtable.h* [*7*](#_Toc58595499)

*3.12. hashtable.c* [*7*](#_Toc58595499)

*3.13. test.h* [*8*](#_Toc58595499)

*3.14. test.c* [*8*](#_Toc58595499)

*3.15. main.c* [*10*](#_Toc58595499)

4. КОДЫ ОШИБОК  [10](#_Toc58595499)

5. ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ  [10](#_Toc58595499)

**1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Разработать программу на C, реализующую структуру данных HashTable (хэш-таблица) с ключами-строками.

**2. МЕТОД РЕШЕНИЯ**

Хеш-таблица — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

Важное свойство хеш-таблиц состоит в том, что, при некоторых разумных допущениях, все три операции (поиск, вставка, удаление элементов) в среднем выполняются за время *O(1)*.

Реализация операций:

1. insert — добавление пары ключ-значение. Вычисляется хэш-функция от ключа, на основе полученного значения рассчитывается индекс в массиве *hashTable*. Если ячейка массива пуста, в неё записывается переданная пара, иначе сравниваются ключи, если они совпадают, то в имеющемся узле значение заменяется на новое. Если ключи не совпадают, но совпадают их хэши, значит произошла коллизия. Один их методов решения данной проблемы это метод цепочек. Каждая ячейка массива *hashTable* является указателем на связный список (цепочку) пар ключ-значение, соответствующих одному и тому же хеш-значению ключа. Коллизии просто приводят к тому, что появляются цепочки длиной более одного элемента. Будем использовать этот метод из-за простоты его реализации.
2. find — поиск пары по ключу. Аналогично insert рассчитывается на основе хэш-функции индекс в массиве *hashTable*. Дальше выполняется проход по всей цепочке, хранящейся в ячейке с данным индексом, сравниваются ключи, если они совпадают, возвращается данная пара. Если пара не найдена, возвращается NULL.
3. remove — удаление пары по ключу. Аналогично find находится пара и удаляется из массива *hashTable.*

**3. ЛИСТИНГ ПРОГРАММЫ**

Исходный код: <https://github.com/iTteruya/C_HashTable>

*3.1. hashtable\_entry.h*

#ifndef HASHTABLE\_HASHTABLE\_ENTRY\_H  
#define HASHTABLE\_HASHTABLE\_ENTRY\_H  
  
#define MAX\_STRING 256  
  
struct Entry{  
 char key[MAX\_STRING];  
 int value;  
 //..can add other types of value  
  
 struct Entry \*next;//next entry in linked list  
};  
  
#endif //HASHTABLE\_HASHTABLE\_ENTRY\_H

*3.2. hashtable\_struct.h*

#ifndef HASHTABLE\_HASHTABLE\_STRUCT\_H  
#define HASHTABLE\_HASHTABLE\_STRUCT\_H  
  
#include "hashtable\_entry.h"  
#include <stdbool.h>  
#include <stdlib.h>  
#include <string.h>  
#include <stdio.h>  
  
#define TABLE\_SIZE 16  
  
struct HashTable{  
 struct Entry \*hashTable[TABLE\_SIZE];  
};  
  
#endif //HASHTABLE\_HASHTABLE\_STRUCT\_H

*3.3. hashtable\_utils.h*

#ifndef HASHTABLE\_HASHTABLE\_UTILS\_H  
#define HASHTABLE\_HASHTABLE\_UTILS\_H  
  
#include "hashtable\_struct.h"  
#include <stdio.h>  
  
  
void print\_table(struct HashTable \*table);  
  
unsigned int hash(char \*key);  
  
  
#endif //HASHTABLE\_HASHTABLE\_UTILS\_H

*3.4. hashtable\_utils.c*

#include "hashtable\_utils.h"  
  
unsigned int hash(char \*key){  
 int length = strnlen(key, MAX\_STRING);  
 unsigned int hash\_value = 0;  
 for (int i = 0; i < length; i++){  
 hash\_value += key[i];  
 hash\_value = (hash\_value \* key[i]) \* 123 % TABLE\_SIZE;  
 }  
 return hash\_value;  
}  
  
void print\_table(struct HashTable \*table){  
 printf("Start\n");  
 for (int i = 0; i < TABLE\_SIZE; i++){  
 if (table->hashTable[i] == NULL) {  
 printf("\t%i\t---\n", i);  
 } else {  
 printf("\t%i\t ", i);  
 struct Entry \*tmp = table->hashTable[i];  
 while (tmp != NULL) {  
 printf("%s - %d; - ", tmp -> key, tmp ->value);  
 tmp = tmp -> next;  
 }  
 printf("\n");  
 }  
 }  
 printf("End\n");  
}

*3.5. hashtable\_insert.h*

#ifndef HASHTABLE\_HASHTABLE\_INSERT\_H  
#define HASHTABLE\_HASHTABLE\_INSERT\_H  
  
#include "hashtable\_utils.h"  
  
bool insert(struct HashTable \*table, struct Entry \*entry);  
  
#endif //HASHTABLE\_HASHTABLE\_INSERT\_H

*3.6. hashtable\_insert.c*

#include "hashtable\_insert.h"  
  
bool insert(struct HashTable \*table, struct Entry \*entry){  
 struct Entry \*elem = (struct Entry \*) malloc(sizeof(\*elem));  
 if (elem == NULL) {  
 printf("Error #1: Memory allocation failure while adding new list element");  
 exit(1);  
 }  
 if (entry == NULL) return false;  
 int index = hash(entry -> key);  
 struct Entry \*tmp = table->hashTable[index];  
 if ((tmp != NULL) && strncmp(entry -> key, tmp->key, MAX\_STRING) == 0){  
 table->hashTable[index] = entry;  
 } else {  
 entry->next = table->hashTable[index];  
 table->hashTable[index] = entry;  
 }  
 return true;  
}

*3.7. hashtable\_find.h*

#ifndef HASHTABLE\_HASHTABLE\_FIND\_H  
#define HASHTABLE\_HASHTABLE\_FIND\_H  
  
#include "hashtable\_utils.h"  
  
struct Entry \*find(struct HashTable \*table, char \*key);  
  
#endif //HASHTABLE\_HASHTABLE\_FIND\_H

*3.8. hashtable\_find.c*

#include "hashtable\_find.h"  
  
struct Entry \*find(struct HashTable \*table, char \*key){  
 int index = hash(key);  
 struct Entry \*tmp = table->hashTable[index];  
 while (tmp != NULL && strncmp(tmp -> key, key, MAX\_STRING) != 0){  
 tmp = tmp -> next;  
 }  
 return tmp;  
}

*3.9. hashtable\_remove.h*

#ifndef HASHTABLE\_HASHTABLE\_REMOVE\_H  
#define HASHTABLE\_HASHTABLE\_REMOVE\_H  
  
#include "hashtable\_utils.h"  
  
struct Entry \*delete(struct HashTable \*table, char \*key);  
  
void deleteAll(struct HashTable \*table);  
  
#endif //HASHTABLE\_HASHTABLE\_REMOVE\_H

*3.10. hashtable\_remove.c*

#include "hashtable\_remove.h"  
  
struct Entry \*delete(struct HashTable \*table, char \*key){  
 int index = hash(key);  
 struct Entry \*tmp = table->hashTable[index];  
 struct Entry \*prev = NULL;  
 while (tmp != NULL && strncmp(tmp -> key, key, MAX\_STRING) != 0){  
 prev = tmp;  
 tmp = tmp ->next;  
 }  
 if (tmp == NULL) return NULL;  
 if (prev == NULL) {  
 table->hashTable[index] = tmp -> next;  
 } else {  
 prev ->next = tmp ->next;  
 }  
 return tmp;  
}  
  
void deleteAll(struct HashTable \*table){  
 for (int i = 0; i < TABLE\_SIZE; i++){  
 if (table->hashTable[i] == NULL) {  
 continue;  
 } else {  
 struct Entry \*tmp = table->hashTable[i];  
 while (tmp != NULL) {  
 struct Entry \*tmp2 = tmp->next;  
 table->hashTable[i] = NULL;  
 tmp = tmp2;  
 }  
 }  
 }  
}

*3.11. hashtable.h*

#ifndef HASHTABLE\_HASHTABLE\_H  
#define HASHTABLE\_HASHTABLE\_H  
  
#include "hashtable\_remove.h"  
#include "hashtable\_find.h"  
#include "hashtable\_insert.h"  
  
struct HashTable \*createHashTable();  
void deleteHashTable(struct HashTable \*table);  
  
  
#endif //HASHTABLE\_HASHTABLE\_H

*3.12. hashtable.c*

#include "hashtable.h"  
  
struct HashTable \*createHashTable(){  
 struct HashTable \*table = (struct HashTable \*) malloc(sizeof(\*table));  
 if (table == NULL) {  
 printf("Error #1: Memory allocation failure while creating new hash table");  
 exit(1);  
 }  
  
 for (int i = 0; i < TABLE\_SIZE; i++){  
 table->hashTable[i] = NULL;  
 }  
 return table;  
}  
  
void deleteHashTable(struct HashTable \*table) {  
 for (int i = 0; i < TABLE\_SIZE; i++){  
 if (table->hashTable[i] == NULL) {  
 continue;  
 } else {  
 struct Entry \*tmp = table->hashTable[i];  
 while (tmp != NULL) {  
 struct Entry \*tmp2 = tmp -> next;  
 free(tmp);  
 tmp = tmp2;  
 }  
 free(tmp);  
 }  
 }  
 free(table);  
}

*3.13.test.h*

#ifndef HASHTABLE\_TEST\_H  
#define HASHTABLE\_TEST\_H  
  
#include "hashtable.h"  
#include <time.h>  
  
void test();  
  
void testInsert(struct HashTable \*table);  
  
void testFind(struct HashTable \*table);  
  
void testRemove(struct HashTable \*table);  
  
#endif //HASHTABLE\_TEST\_H

*3.14. test.c*

#include "test.h"  
  
void test() {  
 struct HashTable \*table = createHashTable();  
  
 testInsert(table);  
}  
  
void testInsert(struct HashTable \*table){  
 printf("\nRunning insert test:\n");  
 printf("Empty table:\n");  
 print\_table(table);  
  
 struct Entry string0 = {.key="John", .value=12};  
 struct Entry string1 = {.key="Sara", .value=21};  
 struct Entry string2 = {.key="Alex", .value=16};  
 struct Entry string3 = {.key="Walter", .value=17};  
 struct Entry string4 = {.key="White", .value=18};  
 struct Entry string5 = {.key="Yellow", .value=14};  
  
 printf("Adding pair John - 12:\n");  
 insert(table, &string0);  
 print\_table(table);  
  
 printf("Adding pair Sara - 21:\n");  
 insert(table, &string1);  
 print\_table(table);  
  
 printf("Adding pair Alex - 16:\n");  
 insert(table, &string2);  
 print\_table(table);  
  
 printf("Adding pair Walter - 17:\n");  
 insert(table, &string3);  
 print\_table(table);  
  
 printf("Adding pair White - 18:\n");  
 insert(table, &string4);  
 print\_table(table);  
  
 printf("Adding pair Yellow - 14:\n");  
 insert(table, &string5);  
 print\_table(table);  
  
  
 printf("Adding pair with already existing key to the table:\n");  
 struct Entry string6 = {.key="John", .value=14};  
 printf("Adding pair John - 14:\n");  
 insert(table, &string6);  
 print\_table(table);  
  
 printf("Showcasing collision:\n");  
 struct Entry string7 = {.key="Program", .value=15};  
 printf("Adding pair Program - 15:\n");  
 insert(table, &string7);  
 print\_table(table);  
  
 testFind(table);  
}  
  
void testFind(struct HashTable \*table){  
 printf("\nRunning find test:\n");  
 print\_table(table);  
  
 char \*key = "John";  
 printf("Searching for the key: %s; found value for the key: %d\n", key, find(table, key)->value);  
  
 key = "White";  
 printf("Searching for the key: %s; found value for the key: %d\n", key, find(table, key)->value);  
  
 key = "Alex";  
 printf("Searching for the key: %s; found value for the key: %d\n", key, find(table, key)->value);  
  
 key = "Sky";  
 struct Entry \*pair = find(table, key);  
 printf("Searching for the key: %s; found value for the key: %s\n", key, pair == NULL ? "NULL" : "");  
  
 testRemove(table);  
}  
  
void testRemove(struct HashTable \*table){  
 printf("\nRunning remove test:\n");  
 print\_table(table);  
  
 char \*key = "John";  
 printf("Removing pair with key: %s\n", key);  
 delete(table, key);  
 print\_table(table);  
  
 key = "White";  
 printf("Removing pair with key: %s\n", key);  
 delete(table, key);  
 print\_table(table);  
  
 key = "Alex";  
 printf("Removing pair with key: %s\n", key);  
 delete(table, key);  
 print\_table(table);  
  
 key = "Sky";  
 printf("Removing pair with key: %s\n", key);  
 delete(table, key);  
 print\_table(table);  
  
 printf("Removing all entries: \n");  
 deleteAll(table);  
 print\_table(table);  
  
 deleteHashTable(table);  
}

*3.15. main.c*

#include "test.h"  
  
int main() {  
 test();  
 return 0;  
}

**4. КОДЫ ОШИБОК**

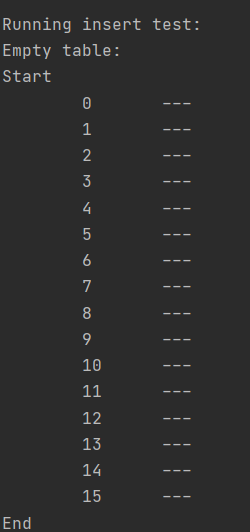
Программа может сгенерировать только один тип ошибок: ошибки выделения памяти (при создании HashTable либо при добавлении нового элемента):

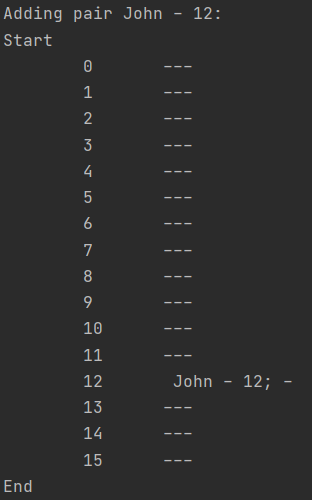
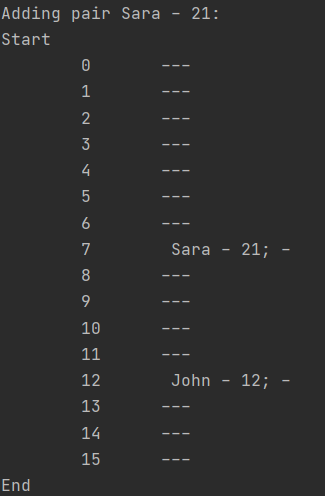
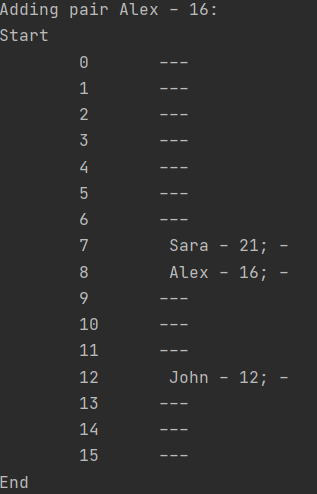
|  |
| --- |
| Error #1: Memory allocation failure while [adding list element|creating hashTable] |

5. Программа и методика испытаний

Испытания созданной структуры данных проводятся с помощью функций, реализованных в файле *test.c*. Сначала создаётся новый HashTable, затем последовательно проходят три группы тестов: на добавление элементов, на получение элементов и на удаление элементов.

В тестах первой группы последовательно добавляется 6 пар с уникальными ключами, затем добавляется ещё одна пара с уже имеющимся ключом. Затем добавляется еще одна пара, хэш ключа, который совпадает с уже имеющимся в хэш-таблице. Элементы выводятся на экран в порядке добавления. Результаты представлены на рис. 1–9.



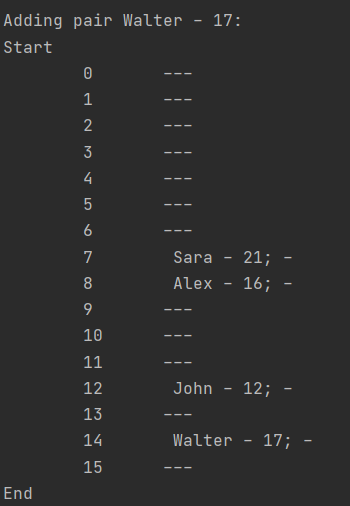
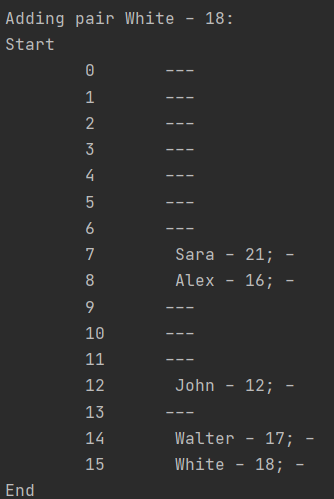
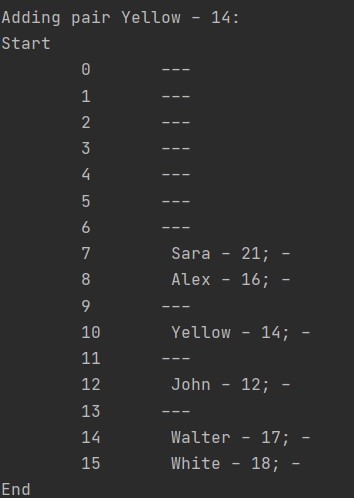
  

Рис. 1-7. Добавления элементов

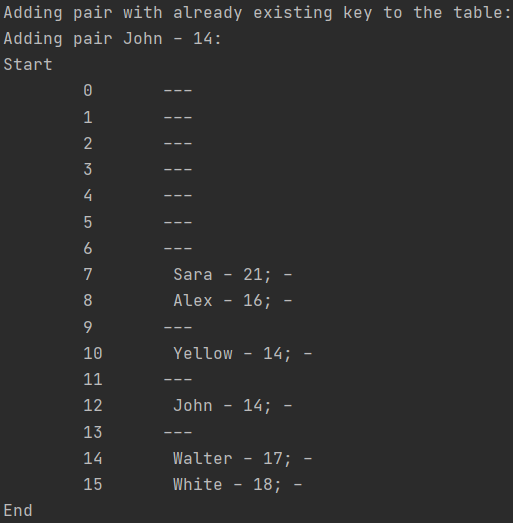
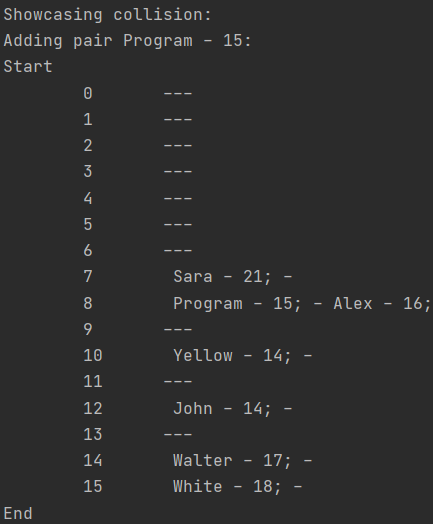
 

Рис.8-9. Добавления элементов ( замена и коллизия)

В тестах второй группы извлекается 3 значения с имеющимися в HashTable ключами и одно с ключом, не представленным в HashTable. Последний вызов возвращает NULL (результаты на рис. 10).

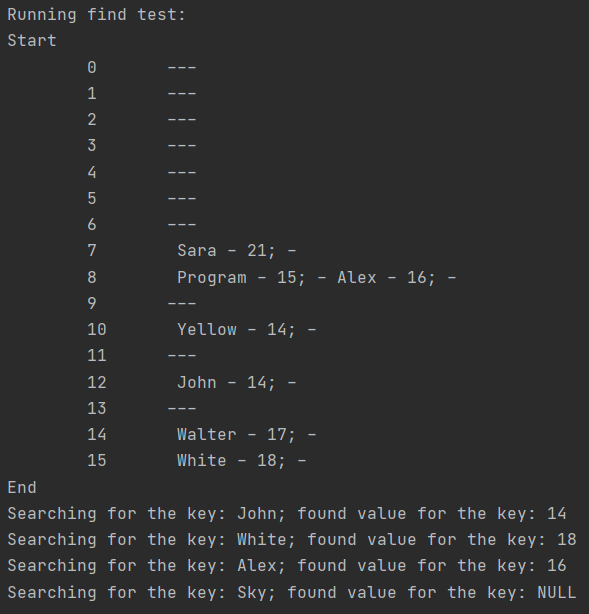
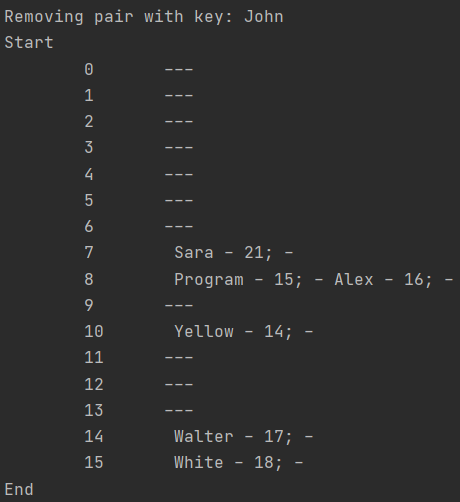
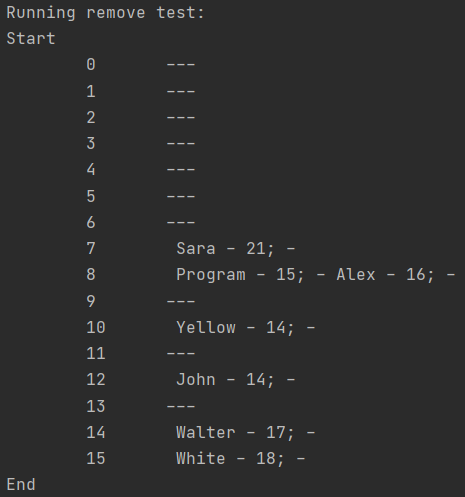
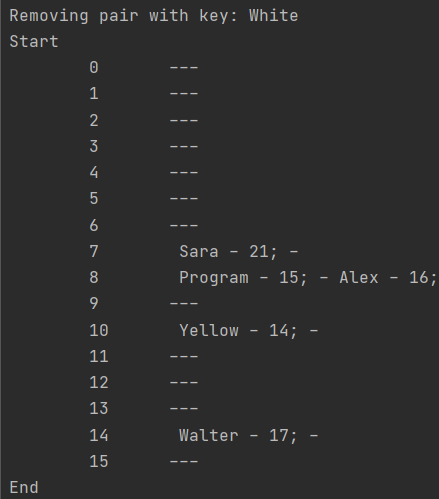
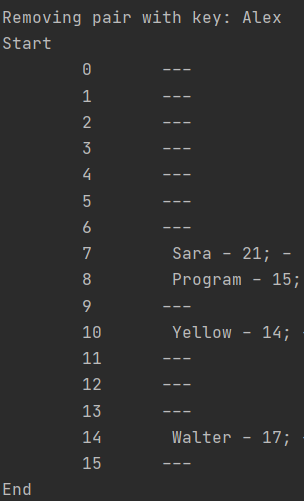
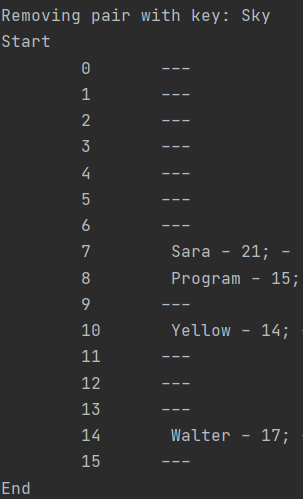


Рис.10. Получение элементов по ключу

В тестах третьей группы удаляется 3 значения с имеющимися в HashTable ключами и одно с ключом, не представленным в Hashtable. Затем удаляются все остальные элементы (результаты на рис. 11–16).



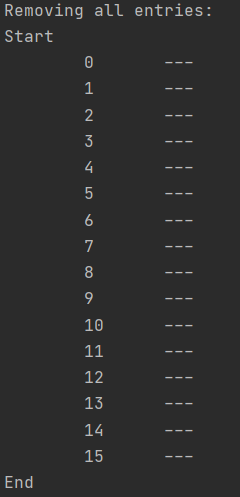


Рис.11-16. Удаление элементов по ключу