Домашнє завдання №11

Скласти програму (C/C++), яка виконує імплементацію хеш-таблиці*(геш-таблиці, англ. Hash table)* без застосування готових реалізацій.

Вибір варіанту

(Nж + Nг + 1) % 6 + 1

де: Nж – порядковий номер студента в групі, а Nг – номер групи(1,2,3,4,5,6,7,8 або 9)

Варіанти завдань

Варіант	Розмір масиву для реалізації хеш-таблиці
1	30
2	40
3	50
4	60
5	70
6	80

Приклад коду

Лістинг

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define SIZE 20

#define EMPTY_ITEM_KEY_MARKER -1

struct Item {
	int key;
	int value;
};

typedef struct HashArray_{
	int(*hashCode)(int key);
	struct Item *(*get)(struct HashArray_ * hashArray, int key);
	void (*add)(struct HashArray_ * hashArray, int key, int value);
	void (*remove)(struct HashArray_ * hashArray, struct Item* item);
	void (*print)(struct HashArray_ * hashArray);

	struct Item dummyItem;

	struct Item* hashArray[SIZE];
} HashArray;

int hashCode_(int key) {
	return key % SIZE;
}

struct Item *get_(HashArray * hashArray, int key) {
```

```
	int hashIndex;
	for (hashIndex = hashArray->hashCode(key); hashArray->hashArray[hashIndex];
hashIndex = hashArray->hashCode(++hashIndex)) {
		if (hashArray->hashArray[hashIndex]->key == key){
			return hashArray->hashArray[hashIndex];
		}
	}

	return NULL;
}

void add_(HashArray * hashArray, int key, int value) {
	int hashIndex;

	struct Item *item = (struct Item*) malloc(sizeof(struct Item));
	if (item){
		item->value = value;
		item->key = key;

		for (hashIndex = hashArray->hashCode(key); hashArray->hashArray[hashIndex]
&& hashArray->hashArray[hashIndex]->key != EMPTY_ITEM_KEY_MARKER; hashIndex = hashArray-
>hashCode(++hashIndex));

		hashArray->hashArray[hashIndex] = item;
	}
}

void delete_(HashArray * hashArray, struct Item* item) {
	int hashIndex, key;

	if (item){
		key = item->key;
		for (hashIndex = hashArray->hashCode(key); hashArray->hashArray[hashIndex];
hashIndex = hashArray->hashCode(++hashIndex)) {
			if (hashArray->hashArray[hashIndex]->key == key) {
				free(hashArray->hashArray[hashIndex]);
				hashArray->hashArray[hashIndex] = &hashArray->dummyItem;
			}
		}
	}
}

void print_(HashArray * hashArray) {
	int index = 0;

	printf("\r\nHashArray: \r\n");
	for (index = 0; index < SIZE; ++index) {
		if (hashArray->hashArray[index]){
			if (hashArray->hashArray[index]->key != EMPTY_ITEM_KEY_MARKER){
				printf("(%d,%d)\r\n", hashArray->hashArray[index]->key,
hashArray->hashArray[index]->value);
			}
			else{
				printf(" ~~ \r\n");
			}
		}
		else{
			printf(" ~~ \r\n");
		}
	}

	printf("\n");
}

int main() {
```

```
	HashArray hashArray = { hashCode_, get_, add_, delete_, print_, {
EMPTY_ITEM_KEY_MARKER /*, -1*/ } };

	struct Item * tempItem;

	hashArray.add(&hashArray, 2,  200);
	hashArray.add(&hashArray, 22, 300);
	hashArray.add(&hashArray, 3,  300);
	hashArray.add(&hashArray, 23, 200);
	hashArray.add(&hashArray, 4,  300);
	hashArray.add(&hashArray, 24, 300);

	hashArray.print(&hashArray);
	tempItem = hashArray.get(&hashArray, 23);

	if (tempItem) {
		printf("Element found: %d\n", tempItem->value);
	}
	else {
		printf("Element not found\n");
	}

	hashArray.remove(&hashArray, hashArray.get(&hashArray, 23));

	hashArray.print(&hashArray);
	tempItem = hashArray.get(&hashArray, 23);

	if (tempItem) {
		printf("Element found: %d\n", tempItem->value);
	}
	else {
		printf("Element not found\n");
	}

#ifdef __linux__
	(void)getchar();
#elif defined(_WIN32)
	system("pause");
#else
#endif

	return 0;
}
```