Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра вычислительных методов и программирования

Отчет по лабораторной работе №1
Рекурсивные функции
Вариант 11

Выполнил:
студент 1 курса
группы № 348602
Трошкин Дмитрий Сергеевич

Проверил:
Матюшкин Светослав Иванович

Минск 2023

1 ДИНАМИЧЕСКАЯ СТРУКТУРА СТЕК

Цель работы: изучить алгоритмы работы с динамическими структурами данных в виде стека.

1.1 Условие

Составить алгоритм в виде блок-схемы, написать и отладить поставленную задачу с использованием рекурсивной и обычной функций. Сравнить полученные результаты.

1.2 Исходный код

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

typedef struct Abitur {
        char name[32];
        unsigned long group;
        char addr[64];
        int *marks;
        int marksCount;
        double avgMark;
} abitur;

typedef int (*read_abitur_fun)(FILE *file, abitur *a);
typedef void (*print_abitur_fun)(FILE *file, int id, abitur *a);
typedef int (*cmp_func)(abitur *a, abitur *c);

void swap(abitur* a, abitur* b)
{
    abitur temp = *a;
    *a = *b;
    *b = temp;
}

// Partition function
int partition(abitur *arr, int low, int high, cmp_func cmp)
{

    // initialize pivot to be the first element
    abitur *pivot = arr + low;
    int i = low;
    int j = high;

    while (i < j) {
```

```
        // condition 1: find the first element greater than
        // the pivot (from starting)
        while (cmp(arr + i, pivot) && i <= high - 1) {
            i++;
        }

        // condition 2: find the first element smaller than
        // the pivot (from last)
        while (!cmp(arr + i, pivot) && j >= low + 1) {
            j--;
        }
        if (i < j) {
            swap(arr+i, arr+j);
        }
    }
    swap(arr+low, arr+j);
    return j;
}

// QuickSort function
void quickSort(abitur *arr, int low, int high, cmp_func cmp)
{
    if (low < high) {

        // call Partition function to find Partition Index
        int partitionIndex = partition(arr, low, high, cmp);

        // Recursively call quickSort() for left and right
        // half based on partition Index
        quickSort(arr, low, partitionIndex - 1, cmp);
        quickSort(arr, partitionIndex + 1, high, cmp);
    }
}

int cmp_abitur_by_name(abitur *a, abitur *c)
{
        return strcmp(a->name, c->name);
}


int read_from_term_abitur(FILE *file, abitur *a) {
        printf("Reading abitur's data:\n");
        printf("Enter name: "); if(!scanf("\n%32[^\n]s", a->name)) return -1;
        printf("Enter group: "); if(!scanf("%li", &a->group)) return -1;
        printf("Enter address: "); if(!scanf("\n%64[^\n]s", a->addr)) return -1;
        printf("Enter marks count: "); if(!scanf("%i", &a->marksCount)) return -1;
        a->marks = malloc(a->marksCount * sizeof(int));
        a->avgMark = 0;
        for(int i = 0; i < a->marksCount; i++) {
                printf("Enter mark #%i: ", i+1); if(!scanf("%i", a->marks + i)) return -1;
                a->avgMark += a->marks[i];
```

```
        }
        a->avgMark /= a->marksCount;
        return 0;
}

void print_abitur(FILE *file, int id, abitur *a) {
        fprintf(file, "\nAbitur's data:\nName: %s\nGroup: %li\nAddress: %s\n", a->name, a->group,
        for(int i = 0; i < a->marksCount; i++)
                fprintf(file, "Mark #%i: %i\n", i+1, a->marks[i]);
        fprintf(file, "Average mark: %f\n", a->avgMark);
}

void print_abitur_line(FILE *file, int id, abitur *a)
{
        fprintf(file, "%.*i | %.*s | %.*li | %.*s\n", 7, id, 32, a->name, 6, a->group, 64, a->addr
}

void write_to_file(FILE *file, int id, abitur *a)
{
        fprintf(file, "%li %s\n%s\n%i ", a->group, a->name, a->addr, a->marksCount);
        for(int i = 0; i < a->marksCount; i++) {
                fprintf(file, "%i ", a->marks[i]);
        }
}
int read_from_file(FILE *file, abitur *a)
{
        if((fscanf(file, "%li", &a->group, a->name) + fgets
                fscanf(file, "%64[^\n]s\n%i", a->addr, &a->marksCount)) < 4
        ) return -1;
        a->marks = malloc(a->marksCount * sizeof(int));
        a->avgMark = 0;
        for(int i = 0; i < a->marksCount; i++) {
                if(!fscanf(file, "%i ", a->marks + i)) return -1;
                a->avgMark += a->marks[i];
        }
        a->avgMark /= a->marksCount;
        return 0;
}

void rm_abitur(abitur *a)
{
        free(a->marks);
}

void rm_abiturs(abitur *a, int count)
{
        for(int i = 0; i < count; i++)
                rm_abitur(a+i);
        free(a);
}
```

```c
int add_abiturs(read_abitur_fun read_fun, FILE *file, abitur *a, int current_count, int count)
{
        a = realloc(a, (current_count + count) * sizeof(abitur));
        for(int i = 0; i < count; i++)
                if(read_fun(file, a + current_count)) return i+1;
        return count;
}

int get_num_from_range(int min, int max)
{
        int res = min - 1;
        if(max <= min) return min;
        printf("Enter num from %i to %i: ", min, max);
        while(res < min || res > max)
                scanf("%i", &res);
        return res;
}

int main()
{
        int mode = 0, count = 0, num;
        char filename[64] = "test.db";
        char buff;
        FILE *file = NULL;
        abitur *Abiturs = NULL;

        while(mode != 9) {
                printf("Abitur db menu:\n"
                         "1. Add abitur from terminal.\n"
                         "2. Remove abiturs.\n"
                         "3. Print abiturs.\n"
                         "4. Do task.\n"
                         "5. Sort abiturs by name.\n"
                         "\n"
                         "6. Read abiturs from file.(extends current list)\n"
                         "7. Write abiturs to file.(overwrites file)\n"
                         "8. Clean.\n"
                         "9. Exit\n");
                printf("Enter mode: ");
                if(!scanf("%i", &mode)) {
                        printf("Try again!\n");
                        scanf("%c", &buff);
                        continue;
                }
                switch (mode) {
                        case 1:
                                Abiturs = realloc(Abiturs, (++count) * sizeof(abitur));
                                if(read_from_term_abitur(NULL, Abiturs+count-1))
                                {
                                        Abiturs = realloc(Abiturs, (--count) * sizeof(abitur));
                                        printf("Reading failed\n");
```

```
        }
        break;
case 2:
        for(int i = 0; i < count; i++)
                print_abitur_line(stdout, i+1, Abiturs+i);
        printf("\nChoose abitur to delete\n");
        num = get_num_from_range(1, count) - 1;
        swap(Abiturs+num, Abiturs+count-1);
        rm_abitur(Abiturs+count-1);
        Abiturs = realloc(Abiturs, (--count)*sizeof(abitur));
        printf("Successfully deleted %i's abitur\n", num+1);
        break;


case 3:
        if(!count) printf("\nList is empty.\n\n");
        scanf("%c", &buff);
        for(int i = 0; i < count; i++) {
                print_abitur(stdout, i+1, Abiturs+i);
                printf("Press enter to view next or q to exit.\n%i abiturs
                scanf("%c", &buff);
                if(buff == 'q') break;
        }
        break;
case 5:
        quickSort(Abiturs, 0, count-1, cmp_abitur_by_name);
        break;
case 6:
        printf("Enter filename[default = \"%s\"]: ", filename);
        scanf("\n%63[^\n]s", filename);
        file = fopen(filename, "r");
        if(!file) {
                printf("Failed to open file \"%s\"\n", filename);
                break;
        }
        fscanf(file, "%i", &num);
        Abiturs = realloc(Abiturs, (count += num) * sizeof(abitur));
        for (int i = 0; i < num; i++)
        {
                if(read_from_file(file, Abiturs+count-1) == -1) {
                        Abiturs = realloc(Abiturs, (--count) * sizeof(abit
                        printf("Reading failed\n");
                }
        }
        fclose(file);
        break;
case 7:
        printf("Enter filename[previous = \"%s\"]: ", filename);
        scanf("\n%63[^\n]s", filename);
        file = fopen(filename, "w");
        if(!file) {
```

```
                                        printf("Failed to open file \"%s\"\n", filename);
                                        break;
                                }
                                fprintf(file, "%i\n", count);
                                for(int i = 0; i < count; i++)
                                        write_to_file(file, i+1, Abiturs+i);
                                fclose(file);
                                break;
                        case 8:
                                if(!count) {
                                        printf("Already cleaned\n");
                                        break;
                                }
                                free(Abiturs);
                                count = 0;
                                printf("Cleaned\n");
                                break;
                        case 9:
                                printf("Bye!\n");
                                break;
                        default:
                                printf("Wrong number!\n");
                }
        }

}
```

1.3 Пример

```
 $ ./task
Исходный стек:

0: 21.000000
1: 49.000000
2: 92.000000
3: 86.000000
4: 35.000000
5: 93.000000
6: 15.000000
7: 77.000000
8: 86.000000
9: 83.000000


Отсортированный стек:

0: 15.000000
1: 21.000000
2: 35.000000
3: 49.000000
4: 77.000000
```

```
5: 83.000000
6: 86.000000
7: 86.000000
8: 92.000000
9: 93.000000


Исходный стек:

0: 69.000000
1: 58.000000
2: 22.000000
3: 2.000000
4: 29.000000
5: 35.000000
6: 67.000000
7: 23.000000
8: 62.000000
9: 30.000000
10: 82.000000
11: 29.000000
12: 67.000000
13: 68.000000
14: 11.000000
15: 36.000000
16: 72.000000
17: 26.000000
18: 40.000000
19: 26.000000
20: 63.000000
21: 59.000000
22: 90.000000
23: 27.000000
24: 62.000000


Отсортированный стек:

0: 2.000000
1: 11.000000
2: 22.000000
3: 23.000000
4: 26.000000
5: 26.000000
6: 27.000000
7: 29.000000
8: 29.000000
9: 30.000000
10: 35.000000
11: 36.000000
12: 40.000000
```

13: 58.000000
14: 59.000000
15: 62.000000
16: 62.000000
17: 63.000000
18: 67.000000
19: 67.000000
20: 68.000000
21: 69.000000
22: 72.000000
23: 82.000000
24: 90.000000

Задание:

0: 46.200000
1: 11.000000
2: 22.000000
3: 23.000000
4: 26.000000
5: 26.000000
6: 27.000000
7: 29.000000
8: 29.000000
9: 30.000000
10: 35.000000
11: 36.000000
12: 40.000000
13: 58.000000
14: 59.000000
15: 62.000000
16: 62.000000
17: 63.000000
18: 67.000000
19: 67.000000
20: 68.000000
21: 69.000000
22: 72.000000
23: 82.000000
24: 90.000000