

МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Государственное бюджетное образовательное учреждение
высшего образования

«ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПУТЕЙ СООБЩЕНИЯ ИМПЕРАТОРА АЛЕКСАНДРА I»

Кафедра «ИНФОРМАЦИОННЫЕ И ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ»

Дисциплина: «Программирование(С)»

О Т Ч Е Т
по лабораторной работе № 3

Вариант 19

Выполнил студент
Факультета *АИТ*
Группы *ИББ-211*

Шефнер А.

Санкт-Петербург
2023

Постановка задачи

Задание 1 по Ханойской башне

а) Вычислить (по формуле !) сколько времени, в воспринимаемом измерении, надо для перемещения 64 элементов Ханойской башни на современном компьютере.

б) Добавить в программу код для подсчета числа произведенных перемещений, сравнить с формулой.

с) Добавить в программу код для отображения номера уровня стека вызовов и аргументов, переданных на этот уровень

д) Написать и отладить программу, в которой надо перенести элементы с А на С, но передвигать можно только с $A \Rightarrow B$, $B \Rightarrow C$ и $C \Rightarrow B$, $B \Rightarrow A$, но не $A \Rightarrow C$ или $C \Rightarrow A$.

е) Написать и отладить программу, в которой надо перенести элементы с А на В, но передвигать можно только с $A \Rightarrow B$, $B \Rightarrow C$ и $C \Rightarrow B$, $B = A$, но не $A \Rightarrow C$ или $C \Rightarrow A$.

Программа должна отображать состояние башен после каждого перемещения.

Задание 2

а) Написать рекурсивную функцию нахождения максимума (минимума) массива, среди элементов, отвечающих какому-либо условию. Функцию условия передавать в виде параметра.

б) Написать рекурсивные функции нахождения суммы и произведения элементов массива, среди элементов, отвечающих какому-либо условию. Функцию условия передавать в виде параметра.

Задание 3

Напишите рекурсивный алгоритм поиска корня уравнения методом последовательных приближений. Функцию для нахождения корня передавать как указатель. Продемонстрируйте работу алгоритма, например, на поиске кубического корня или уравнения, подобного $x^3 = \sin(x)$ (начальное приближение $x_0=1$) или других.

Приведение кубического уравнения, к виду для решения методом последовательных приближений можно сделать такими тождественными преобразованиями:

$$x^3 - x = a$$

$$x = a / (x^2 - x)$$

$$3x^2 = 2x + a / (x^2 - x)$$

$$x = 1/3 * (2x + a / (x^2 - x))$$

Задание 4

Рекурсивная функция соответствует известной вам функции, и приведённая формула также вам известна.

Задание:

а) Написать программу вычисления этой рекурсивной функции на С (для x в диапазоне $[-1.2, 3]$).

б) Найти корень уравнения $rS(x)=0$, при начальном приближении $x=3$.

с) Написать код вычисления этой функции методом итераций.

Примечание: для задач 2, 3 и 4 использовать мемоизацию. Мемоизация (англ. memoization от англ. memory и англ. optimization) — в программировании сохранение результатов выполнения функций для предотвращения повторных вычислений. Это один из способов оптимизации, применяемый для увеличения скорости выполнения компьютерных программ.

Пояснения

Ханойская башня выполнена нерекурсивным алгоритмом. Программа руководствуется таким правилом, что после каждого хода есть только один возможный следующий ход без обращения предыдущего.

Код программы (Задание по Ханойской башне)

main.c

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

#include "hanoisolve.h"
#include "hanoicalcs.h"

void show_info_task();

void solve_hanoi_task();

int main() {
    system("chcp 65001");
    system("cls");
    int task_num;
    printf_s("Введите номер желаемого действия:\n");
    printf_s("1 - вывести информацию:\n");
    printf_s("2 - решить ханойскую башню без ходов A=>C и C=>A:\n");
    scanf_s("%d", &task_num);
    switch (task_num) {
        case 1:
            show_info_task();
            break;
        case 2:
            solve_hanoi_task();
            break;
        default:
            printf_s("Неверный номер действия.\n");
    }

    system("pause");
    return 0;
}

void show_info_task() {
    int disc_count = 64;
    unsigned long long turn_count = calc_turn_count(disc_count);
    printf_s("Для %d дисков потребуется сделать %llu ходов.\n",
disc_count, turn_count);
    unsigned long long seconds = calc_time(disc_count);
    unsigned long long years = seconds / 60 / 60 / 24 / 365;
    printf("Это займёт %llu секунд или %llu лет.\n", seconds,
years );
}

void solve_hanoi_task() {
    int disc_count;
    printf_s("Введите количество дисков.\n");
    scanf_s("%d", &disc_count);
    if(disc_count <=0) {
```

```
        printf_s("Количество дисков должно быть  
положительным.");  
        return;  
    }  
  
    int dest;  
    printf_s("Введите номер стержня, на который надо переместить  
диски (1 - В или 2 - С)\n");  
    scanf_s("%d", &dest);  
    if(dest < 1 || dest > 2) {  
        printf_s("Неверный номер.");  
        return;  
    }  
  
    solve_hanoi(disc_count, dest);  
}
```

hanoicalcs.h

```
#ifndef C_LAB_3_1_HANOICALCS_H
#define C_LAB_3_1_HANOICALCS_H

unsigned long long calc_turn_count(int disc_count);

unsigned long long calc_time(int disc_count);

#endif //C_LAB_3_1_HANOICALCS_H
```

hanoicalcs.c

```
#include "hanoicalcs.h"

unsigned long long calc_turn_count(int disc_count) {
    if(disc_count == 64) return ((unsigned long long)1 << 64) -
1;
    return ((unsigned long long)1 << disc_count) - 1;
}

unsigned long long calc_time(int disc_count) {
    unsigned long long turns_per_second = 309994085;
    return calc_turn_count(disc_count) / turns_per_second;
}
```


hanoisolve.h

```
#ifndef C_LAB_3_1_HANOISOLVE_H
#define C_LAB_3_1_HANOISOLVE_H

typedef unsigned int uint;

typedef struct Rod {
    uint *array;
    uint count;
    uint size;
} Rod;

void solve_hanoi(unsigned int disc_count, unsigned int dest);

#endif //C_LAB_3_1_HANOISOLVE_H
```

hanoisolve.c

```
#include "hanoisolve.h"
#include "malloc.h"
#include "stdio.h"
#include "showhanoi.h"
#include "windows.h"

#define CLEAR 1
#define WAIT_TIME 0

void init_rods(Rod* rods, uint disc_count);
void find_legal_move(Rod* rods_state, int* current_move);
uint check_move(Rod* a, Rod* b);
uint is_empty(Rod* rod);
uint is_full(Rod* rod);
void make_move(Rod* rods, const int* move);
uint check_for_solved(Rod* rod);

void wait() {
    #if WAIT_TIME
        Sleep(WAIT_TIME);
    #else
        system("pause");
    #endif
    #if CLEAR
        system("cls");
    #endif
}

void solve_hanoi(uint disc_count, uint dest) {
    system("cls");
    char* aliases = "ABC";
    Rod rods[3];
    init_rods(rods, disc_count);

    int current_move[2];
    current_move[0] = -1;
    current_move[1] = -1;

    show_rods(rods, aliases);
    do {
        wait();
        find_legal_move(rods, current_move);
        make_move(rods, current_move);
        show_rods(rods, aliases);
        show_move(aliases[current_move[0]],
aliases[current_move[1]], disc_count);
    } while (!check_for_solved(&rods[dest]));
}

void init_rods(Rod* rods, uint disc_count) {
```

```

    rods[0].size = disc_count;
    rods[0].count = disc_count;
    rods[0].array = (uint *)malloc(disc_count * sizeof (uint));
    for(int i = 0; i < disc_count; i++) {
        rods[0].array[i] = disc_count - i;
    }
    rods[1].size = rods[2].size = disc_count;
    rods[1].count = rods[2].count = 0;
    rods[1].array = (uint *)malloc(disc_count * sizeof (uint));
    rods[2].array = (uint *)malloc(disc_count * sizeof (uint));
    for(int i = 0; i < disc_count; i++) {
        rods[1].array[i] = rods[2].array[i] = 0;
    }
}

uint check_for_solved(Rod* rod) {
    for(int i = 0; i < rod -> size; i++) {
        if(rod->array[i] != rod -> size - i) return 0;
    }
    return 1;
}

uint check_move_and_change_if_legal(Rod* rods, int
*current_move, int from, int to) {
    if((current_move[0] != to || current_move[1] != from) &&
check_move(&rods[from], &rods[to])) {
        current_move[0] = from;
        current_move[1] = to;
        return 1;
    }
    return 0;
}

void find_legal_move(Rod* rods, int *current_move) {
    if(check_move_and_change_if_legal(rods, current_move, 0, 1))
return;
    if(check_move_and_change_if_legal(rods, current_move, 1, 0))
return;
    if(check_move_and_change_if_legal(rods, current_move, 1, 2))
return;
    if(check_move_and_change_if_legal(rods, current_move, 2, 1))
return;
}

uint check_move(Rod* a, Rod* b) {
    if(is_empty(a)) return 0;
    if(is_full(b)) return 0;
    if(is_empty(b)) return 1;
    return a->array[a->count - 1] < b->array[b->count - 1];
}

uint is_empty(Rod* rod) { return rod->count == 0; }

```

```
uint is_full(Rod* rod) { return rod->count == rod->size; }

void make_move(Rod* rods, const int* move) {
    Rod* a = &rods[move[0]];
    Rod* b = &rods[move[1]];
    b->array[b->count++] = a->array[--a->count];
    a->array[a->count] = 0;
}
```

showhanoi.h

```
#ifndef C_LAB_3_1_SHOWHANOI_H
#define C_LAB_3_1_SHOWHANOI_H

#include "hanoisolve.h"

void show_rods(Rod *rods, char* aliases);

void show_move(char from, char to, uint disc_max);

#endif //C_LAB_3_1_SHOWHANOI_H
```

showhanoi.c

```
#include <stdio.h>
#include "showhanoi.h"
#include "malloc.h"
#define DISC_SYMBOL '_'
#define BASE_SYMBOL '='

char* get_disc_str(uint size, uint max) {
    uint len = max * 2 + 1;
    char* str = (char*)malloc(len + 1);
    for(int i = 0; i < len; i++) {
        str[i] = i < max - size + 1 || i > max + size - 1 ? '
' : DISC_SYMBOL;
    }
    str[len] = '\0';
    return str;
}

char* get_base_str(uint max, char alias) {
    uint len = max * 2 + 3;
    char* str = (char*)malloc(len + 1);
    for(int i = 0; i < len; i++) {
        str[i] = BASE_SYMBOL;
    }
    str[max + 1] = alias;
    str[len] = '\0';
    return str;
}

void print_line(Rod *rods, uint line) {
    uint max = rods->size;
    for(int i = 0; i < 3; i++) {
        char* str = get_disc_str(rods[i].array[max - line - 1],
max);
        printf_s(" %s ", str);
        free(str);
    }
    printf_s("\n");
}

void print_bases(uint max, char* aliases) {
    for(int i = 0; i < 3; i++) {
        char* str = get_base_str(max, aliases[i]);
        printf_s(" %s ", str);
        free(str);
    }
    printf_s("\n\n");
}

void show_rods(Rod *rods, char* aliases) {
    uint disc_count = rods->size;
    for(int i = 0; i < disc_count; i++) {
```

```
        print_line(rods, i);
    }
    print_bases(disc_count, aliases);
}

void show_move(char from, char to, uint disc_max) {
    uint spaces = disc_max * 3 + 4;
    for(int i = 0; i < spaces; i++) {
        printf_s(" ");
    }
    printf_s("%c --> %c\n\n\n", from, to);
}
```

Код программы (Задания 2-4)

main.c

```
#include <stdio.h>
#include <malloc.h>

#include "num2.h"
#include "num3.h"
#include "num4.h"

int condition(int x);

int main() {
    // Number 1
    printf("Number 2.\n\n");
    int arr[18] = {2, 7, 5, 2, 67,
                  3, 45, 23, 5, 56,
                  23, 45, 566, 4, 5,
                  2, 54, 67};
    for(int i = 0; i < 18; i++) {
        printf("%d ", arr[i]);
    }
    printf("\nMax: %d, sum: %d, product: %d.\n\n",
          find_max(arr, arr + 18, condition),
          sum_func(arr, arr + 18, condition),
          product_func(arr, arr + 18, condition));

    // Number 2
    printf("Number 3.\n\n");
    printf("Root for func 1: %lf.\n", find_root(1, func_1,
0.01));
    printf("Root for func 2: %lf.\n\n", find_root(1, func_2,
0.01));

    // Number 3
    printf("Number 3.\n\n");

    double integ = 0;
    for(double x = -1.2; x <= 3; x += 0.0001)
        integ += func_rs_iterative(x);
    integ *= 0.0001;
    printf("Integral from, -1.2 to 3 of rS: %lf\n", integ);
    printf("rS recursive: %lf\n", find_root(3, func_rs_recursive,
0.01));
    printf("rS iterative: %lf\n\n",
find_root(3, func_rs_iterative, 0.01));

    return 0;
}

int condition(int x) {
    static int *cache = NULL;
    if(cache == NULL) {
        cache = (int *) calloc(__INT16_MAX__, 4);
    }
}
```



```
    }  
    if(x <= __INT16_MAX__ ) {  
        if(cache[x] == 0) {  
            cache[x] = (x % 2 == 1 && x > 5 && x % 3 != 2) + 1;  
        }  
        return cache[x] - 1;  
    }  
    return (x % 2 == 1 && x > 5 && x % 3 != 2);  
}
```

num2.h

```
#ifndef C_LAB_3_2_NUM2_H
#define C_LAB_3_2_NUM2_H

int find_max(int* start, int* end, int (*cond)(int));

int sum_func(int* start, int* end, int (*cond)(int));

int product_func(int* start, int* end, int (*cond)(int));

#endif //C_LAB_3_2_NUM2_H
```

num2.c

```
#include "num2.h"

#define INT_MIN (-2147483648)

void find_max_step(const int *start, const int *end, int
(*cond)(int), int *max_elem) {
    if(start > end) return;
    if(*start > *max_elem && cond(*start)) *max_elem = *start;
    find_max_step(++start, end, cond, max_elem);
}

int find_max(int *start, int *end, int (*cond)(int)) {
    int max_elem = INT_MIN;
    find_max_step(start, end, cond, &max_elem);
    return max_elem;
}

void sum_step(const int *start, const int *end, int(*cond)(int),
int *sum) {
    if(start > end) return;
    if(cond(*start)) *sum += *start;
    sum_step(start + 1, end, cond, sum);
}

int sum_func(int* start, int* end, int (*cond)(int)) {
    int sum = 0;
    sum_step(start, end, cond, &sum);
    return sum;
}

void product_step(int *start, int *end, int(*cond)(int), int
*product) {
    if(start > end) return;
    if(cond(*start)) *product *= *start;
    product_step(start + 1, end, cond, product);
}

int product_func(int* start, int* end, int (*cond)(int)) {
    int product = 1;
    product_step(start, end, cond, &product);
    return product;
}
```

num3.h

```
#ifndef C_LAB_3_2_NUM3_H
#define C_LAB_3_2_NUM3_H

double func_1(double x);

double func_2(double x);

double find_root(double start_x, double (*func) (double x),
double delta);

#endif //C_LAB_3_2_NUM3_H
```

num3.c

```
#include "num3.h"

#include <math.h>

double func_1(double x) {
    return x * x - sin(x);
}

double func_2(double x) {
    return x * x * x - 1.728;
}

double find_root(double start_x, double (*func) (double x),
double delta) {
    if(round(func(start_x) * 100) == 0) {
        return start_x;
    }

    double f1 = round(func(start_x - delta) * 100);
    double f2 = round(func(start_x + delta) * 100);

    if(fabs(f1) <= fabs(f2)) {
        return find_root(start_x - delta, func, delta);
    }
    return find_root(start_x + delta, func, delta);
}
```

num4.h

```
#ifndef C_LAB_3_2_NUM4_H
#define C_LAB_3_2_NUM4_H

double func_rs_recursive(double x);

double func_rs_iterative(double x);

#endif //C_LAB_3_2_NUM4_H
```

num4.c

```
#include "num4.h"

#include <math.h>

double func_rs_recursive(double x) {
    if(x < 0.01) {
        return x;
    }

    double r = func_rs_recursive(x / 2);
    return 2 * r * sqrt(1 - r * r);
}

double func_rs_iterative(double x) {
    int depth = 0;
    double ans = x;

    while(ans >= 0.01) {
        depth++;
        ans /= 2;
    }

    for(int i = 0; i < depth; i++) {
        ans = 2 * ans * sqrt(1 - ans * ans);
    }

    return ans;
}
```

Отладка приложения (Задание по Ханойской башне):

```
MINGW64:/d/University/2 ter X + v
Введите номер желаемого действия:
1 - вывести информацию:
2 - решить ханойскую башню без ходов A=>C и C=>A:
1
Для 64 дисков потребуется сделать 18446744073709551615 ходов.
Это займёт 59506761471 секунд или 1886 лет.
Press any key to continue . . . |
```

```
MINGW64:/d/University/2 ter X + v
Введите номер желаемого действия:
1 - вывести информацию:
2 - решить ханойскую башню без ходов A=>C и C=>A:
2
Введите количество дисков.
4
Введите номер стержня, на который надо переместить диски (1 - В или 2 - С)
1|
```

```
MINGW64:/d/University/2 ter X + v
      -
     ---
    -----
   -----
  -----A-----  =====B=====  =====C=====
Press any key to continue . . .
```

```
MINGW64:/d/University/2 ter X + v
      -          ---
     -----    -----
    =====A=====  =====B=====  =====C=====
                    C --> B
Press any key to continue . . . |
```



```
MINGW64:/d/University/2 ter X + v

      ---
    -----
  =====A=====  =====B=====  =====C=====

      B --> A

Press any key to continue . . . |
```

```
MINGW64:/d/University/2 ter X + v

      -----
  =====A=====  -----
  =====B=====  -----
  =====C=====

      A --> B

Press any key to continue . . . |
```

```
MINGW64:/d/University/2 ter X + v

      -
      ---
      ----
      -----
  =====A=====  =====B=====  =====C=====

      C --> B

Press any key to continue . . . |
```

Отладка приложения (Задания 2-4):

```
"D:\University\2 term\Programming C\Lab_3\c_lab_
Number 2.

2 7 5 2 67 3 45 23 5 56 23 45 566 4 5 2 54 67
Max: 67, sum: 231, product: 63631575.

Number 3.

Root for func 1: 0.880000.
Root for func 2: 1.200000.

Number 3.

Integral from, -1.2 to 3 of rS: 1.269912
rS recursive: 3.140000
rS iterative: 3.140000

Process finished with exit code 0
```