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(C)»

ОТЧЕТ

по лабораторной работе № 3

Вариант *19*

Выполнил студент Шефнер А.

Факультета *АИТ*

Группы *ИВБ-211*

Санкт-Петербург

2023

**Постановка задачи**

**Задание 1 по Ханойской башне**

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

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

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

d) Написать и отладить программу , в которой надо перенести элементы с A на С, но передвигать можно только с А=>B, B=>C и C=>B, B=>A, но не А=>C или C=>A.

e) Написать и отладить программу , в которой надо перенести элементы с A на B, но передвигать можно только с А=>B, B=>C и C=>B, B=A, но не А=>C или C=>A.

Программа должна отображать состояние башен после каждого перемещения.

**Задание 2**

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

b) 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\*x=sin(x) (начальное приближение x0=1) или других.

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

x\*x\*x=a

x=a/(x\*x)

3\*x=2\*x+a/(x\*x)

x=1/3\*(2\*x+a/(x\*x))

**Задание 4**

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

Задание:

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

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

c) 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 - B или 2 - C)\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++) {**

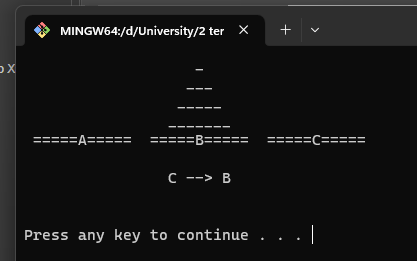
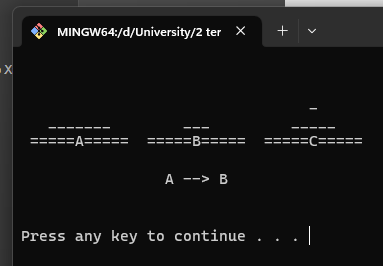
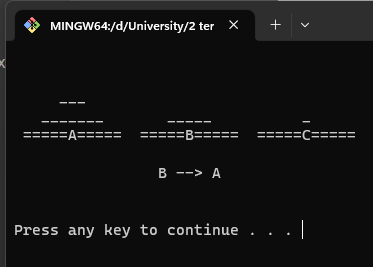
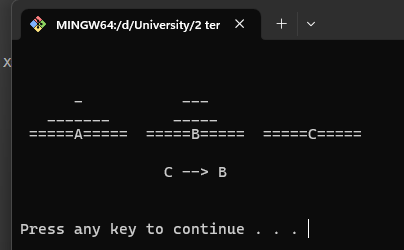
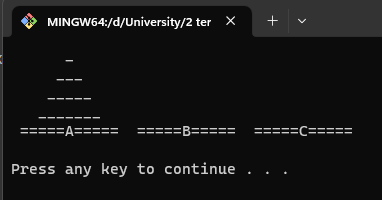
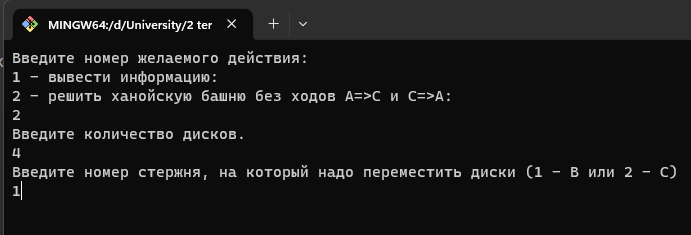
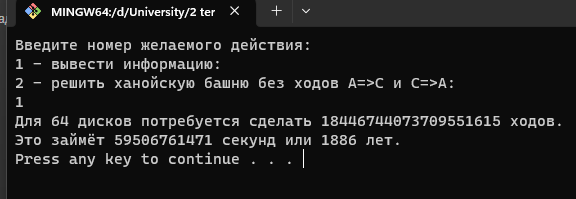
**ans = 2 \* ans \* sqrt(1 - ans \* ans);**

**}**

**return ans;**

**}**

**Отладка приложения (Задание по Ханойской башне):**



**Отладка приложения (Задания 2-4):**

