**ОТЧЕТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнили:

студенты группы 23ВВВ4

Гришаев. Д.А.

Фельдман Г.О.

Приняли:

Деев М.В.

Юрова О.В.

**Название**

Оценка времени выполнения программ

**Цель работы**

Осуществить замер времени работы программы библиотекой time.h, содержащую описания типов и прототипы функций для работы с датой и временем.

**Лабораторное задание**

На основе приведенных в лабораторной работе примеров выполнить следующие задания:

**Задание 1:**

1. Вычислить порядок сложности программы (*О*-символику).
2. Оценить время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 100, 200, 400, 1000, 2000, 4000, 10000.
3. Построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.

**Задание 2**:

1. Оценить время работы каждого из реализованных алгоритмов на случайном наборе значений массива.
2. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.
3. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.
4. Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.
5. Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

**Описание метода решения задачи**

На основе описанного программного кода, опираясь на теорию, вычислим порядок сложности программы и оценим время выполнения перемножения матриц с соответствующим графиком зависимости

### Листинг

а)

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define arraysize 10000

void main(void){

clock\_t start, end; // объявляем переменные для определения времени выполнения

int i = 0, j = 0, r;

char\* a = (char\*)malloc(arraysize \* arraysize);

char\* b = (char\*)malloc(arraysize \* arraysize);

int\* c = (int\*)malloc(arraysize \* arraysize\*sizeof(int));

int elem\_c;

//char a[arraysize][arraysize], b[arraysize][arraysize], c[arraysize][arraysize], elem\_c;

srand(time(NULL)); // инициализируем параметры генератора случайных чисел

while (i < arraysize)

{

while (j < arraysize)

{

\*(a + i \* arraysize + j) = rand() % 100 + 1; // заполняем массив случайными числами

j++;

}

i++;

}

srand(time(NULL)); // инициализируем параметры генератора случайных чисел

i = 0; j = 0;

while (i < arraysize)

{

while (j < arraysize)

{

\*(b + i \* arraysize + j) = rand() % 100 + 1; // заполняем массив случайными числами

j++;

}

i++;

}

start = clock();

for (i = 0; i < arraysize; i++)

{

for (j = 0; j < arraysize; j++)

{

elem\_c = 0;

for (r = 0; r < arraysize; r++)

{

elem\_c = (int) elem\_c + \*(a + i \* arraysize + r) \* \*(b + i \* arraysize + j);

\*(c + i \* arraysize + j) = elem\_c;

}

}

}

end = clock();

double time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Time spent:%lf s", time\_spent);

free(a);

free(b);

free(c);

}

b)

void shell(int \*items, int count)

{

int i, j, gap, k;

int x, a[5];

a[0]=9; a[1]=5; a[2]=3; a[3]=2; a[4]=1;

for(k=0; k < 5; k++) {

gap = a[k];

for(i=gap; i < count; ++i) {

x = items[i];

for(j=i-gap; (x < items[j]) && (j >= 0); j=j-gap)

items[j+gap] = items[j];

items[j+gap] = x;

}

}

}

void qs(int \*items, int left, int right) //вызов функции: qs(items, 0, count-1);

{

int i, j;

int x, y;

i = left; j = right;

/\* выбор компаранда \*/

x = items[(left+right)/2];

do {

while((items[i] < x) && (i < right)) i++;

while((x < items[j]) && (j > left)) j--;

if(i <= j) {

y = items[i];

items[i] = items[j];

items[j] = y;

i++; j--;

}

} while(i <= j);

if(left < j) qs(items, left, j);

if(i < right) qs(items, i, right);

}

**Пояснительный текст к программе**

Наибольшая сложность в программе — это сложность умножения матриц, которая является доминирующей. Следовательно, общая сложность программы составляет:

O(arraysize³)

Это означает, что время выполнения программы увеличивается кубически с увеличением размера массивов.

Время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от

100: 0.002 с.

200: 0.023 c.

400: 0.16 c.

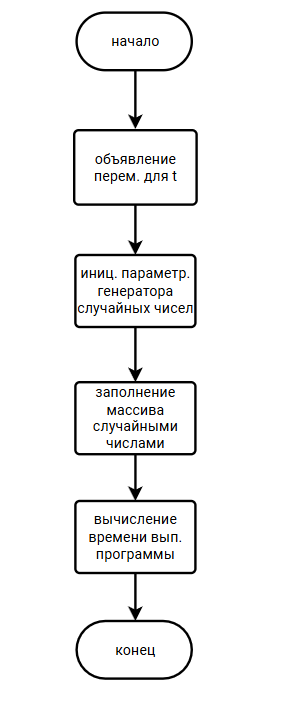
1000: 2.462 c.

2000: 19.516 c.

4000: 158.982 c.

10000: 2496.378 с.

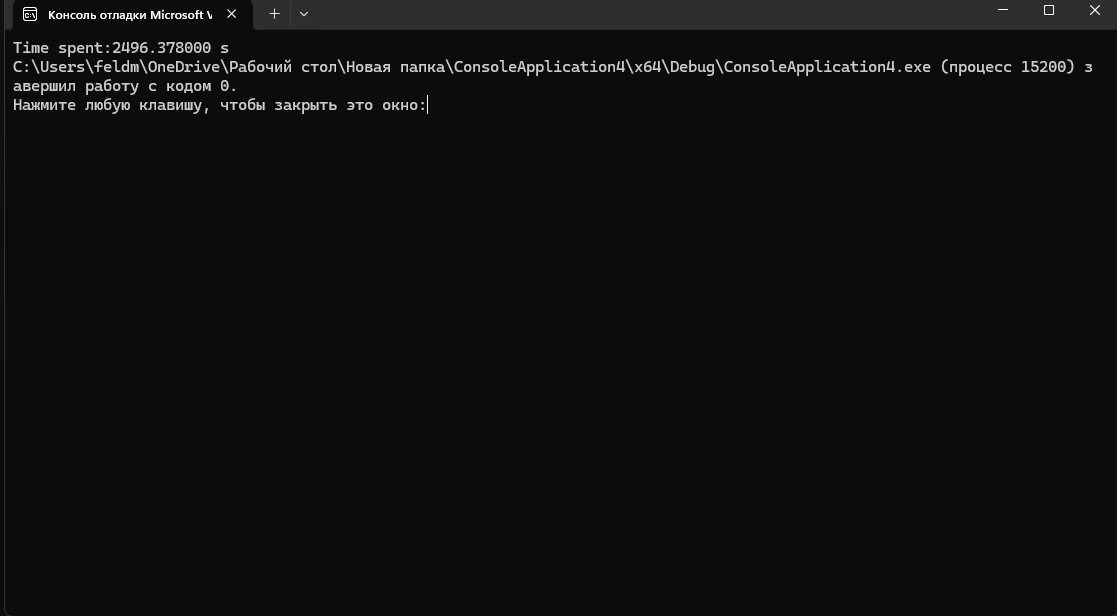
**Схема программы**

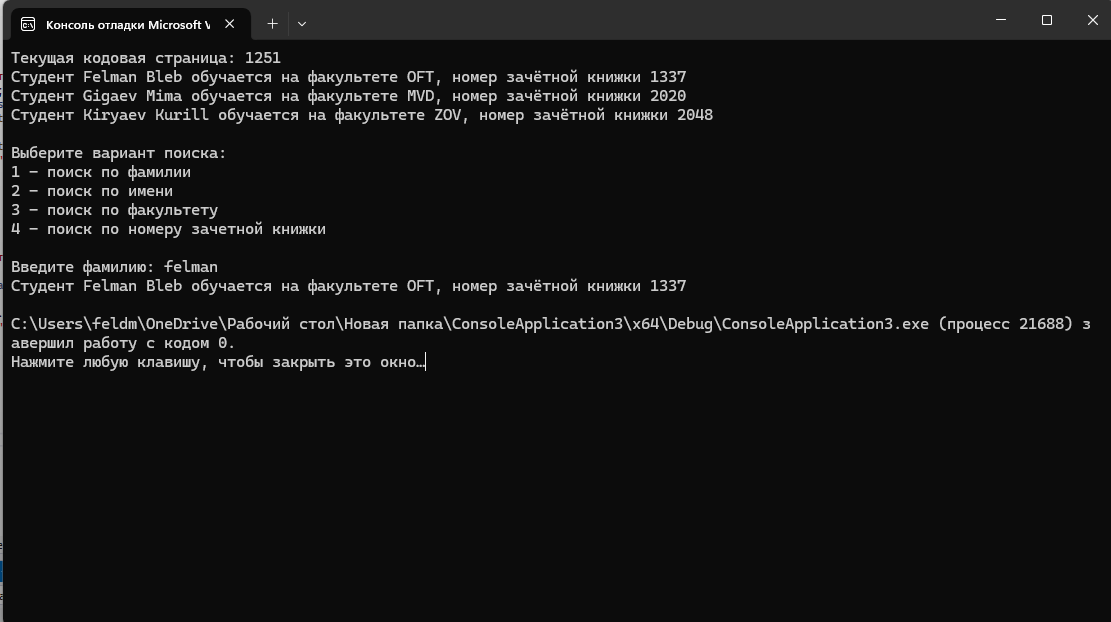
**  
 Рисунок 1 — Схема программы а. c**

**Рисунок 2 — Схема программы б.**

**Результат работы программы**

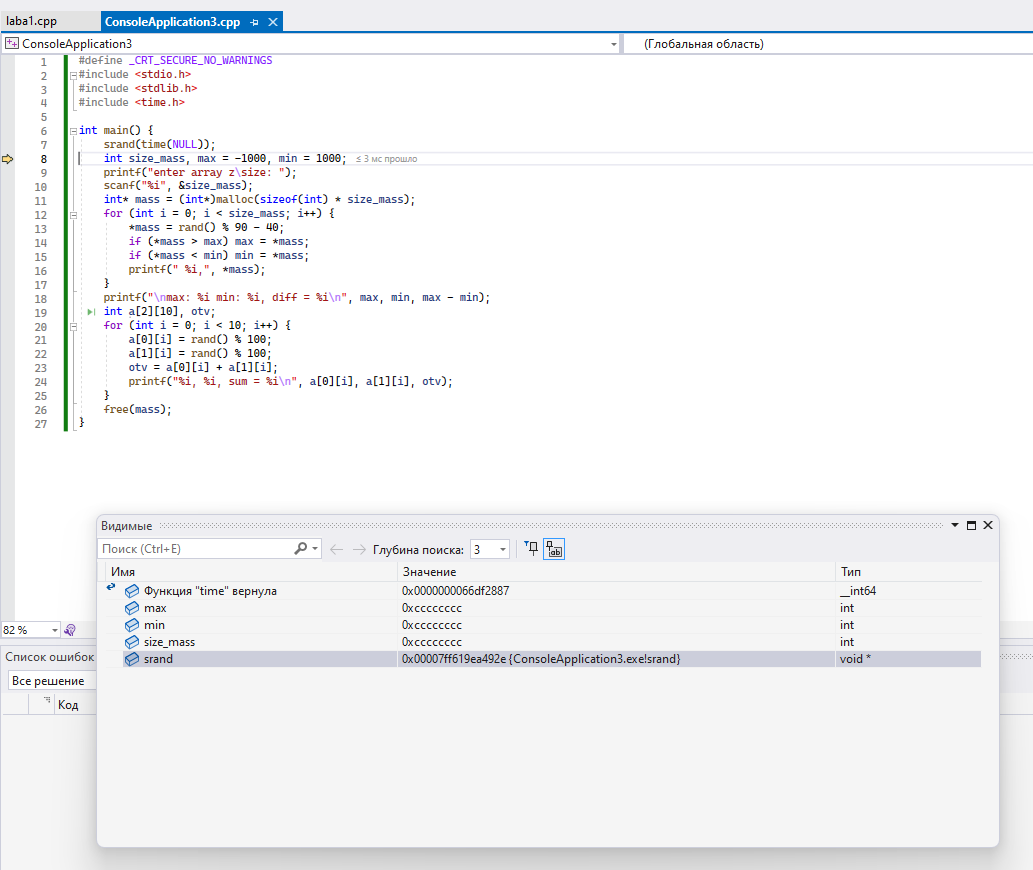
Для массива другой размерности результат аналогичен

 **Рисунок 3 — Результаты работы программы а.**

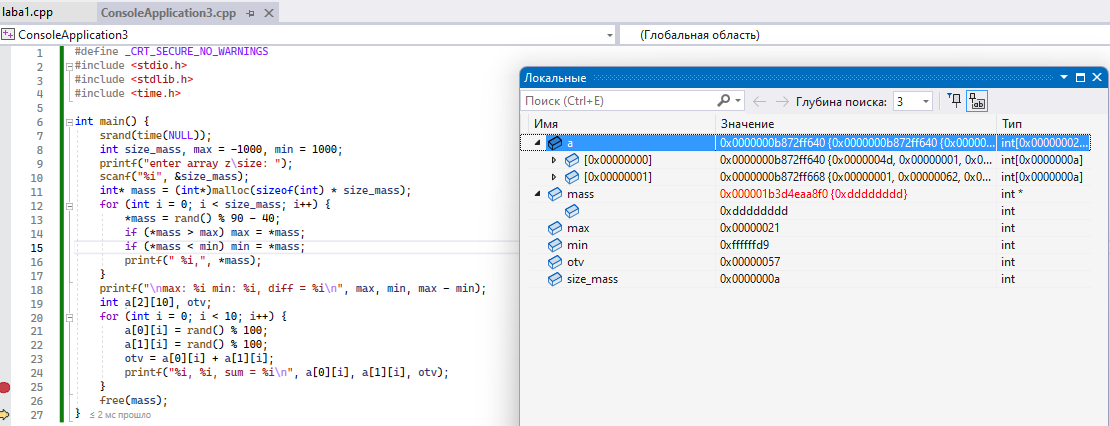
 **Рисунок 4 — Результаты работы программы b.**

### Протокол трассировки программы

На рисунке 5 показан протокол трассировки до заполнения массива.

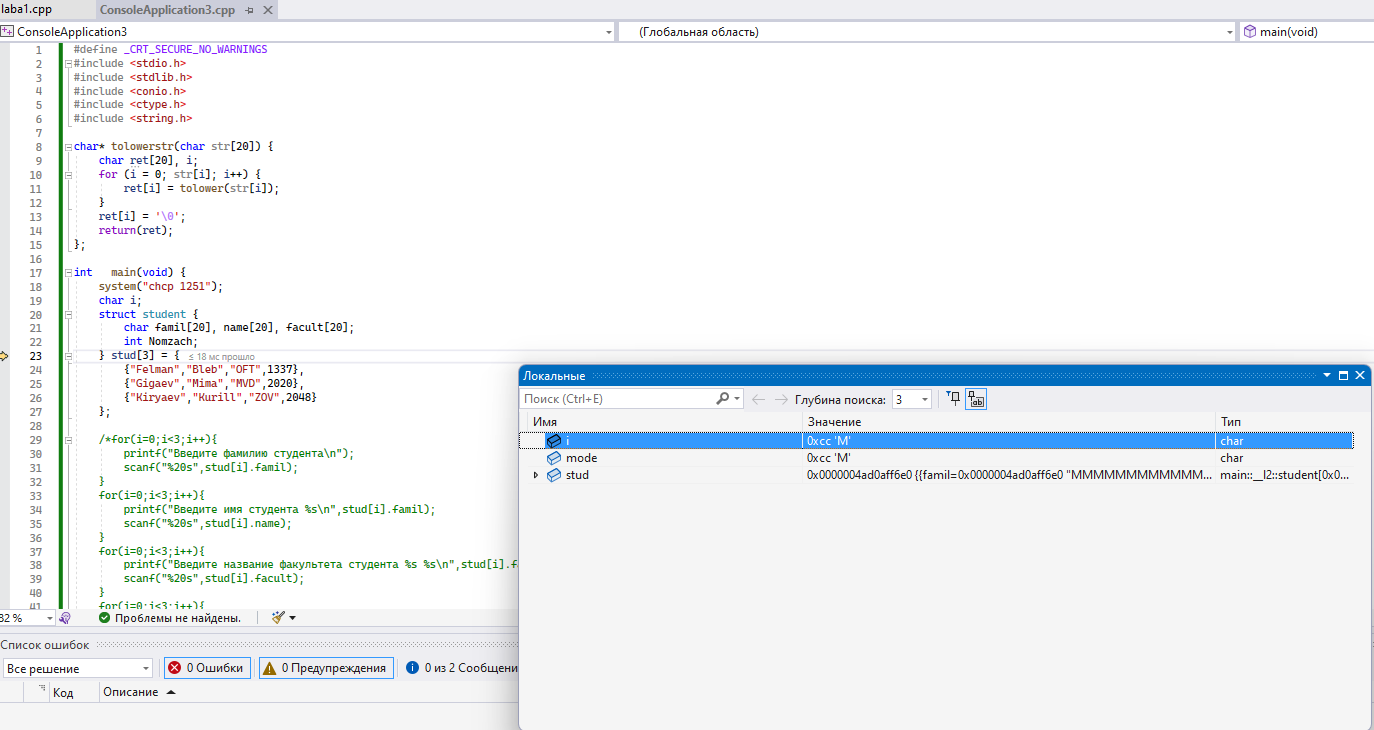
  
**Рисунок 5 — Протокол трассировки прогр. а до зап.**

На рисунке 6 показан протокол трассировки после заполнения массива

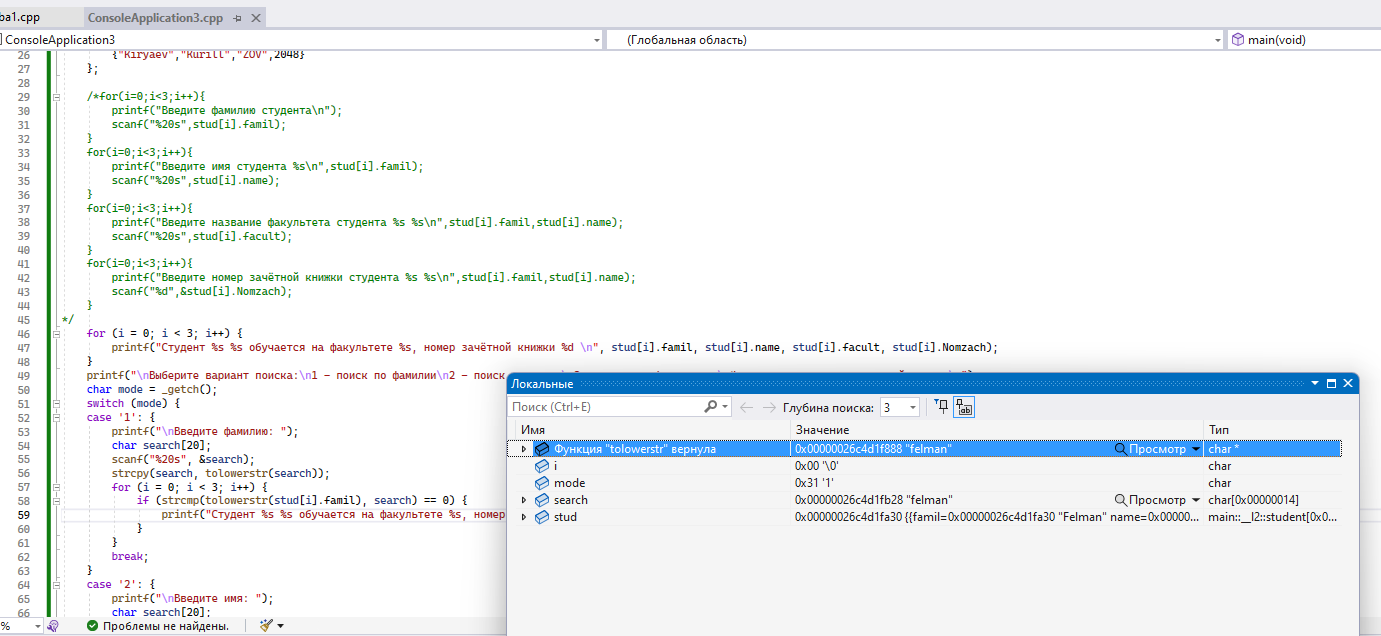


**Рисунок 6 — Протокол трассировки прогр. а после зап.**

На рисунке 7 показан протокол трассировки до заполнения структур.

  
**Рисунок 7 — Протокол трассировки программы b до зап.**

На рисунке 8 показан протокол трассировки после заполнения структур и поиска.

  
**Рисунок 8 — Протокол трассировки b после зап.**

Результат работы программы, показанный на рисунке 1, совпал с результатами трассировки.

### Выводы

Повторили принципы использования и применения простых структур данных языка Си (массивов, строк) для решения поставленных задач.