Бюджетное учреждение высшего образования   
Ханты-Мансийского автономного округа   
«Сургутский государственный университет»

Политехнический институт

Кафедра автоматики и компьютерных систем

**Отчет**

по лабораторной работе № 1 «Формирование массивов экспериментальных данных»

по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы 609-01  
Кадырин В. Ю

Проверил: старший преподаватель кафедры АиКС   
Назаров Е. В.

Сургут

2022 г.

# Цель работы:

Освоить принципы формирования монотонных (упорядоченных и упорядоченных в обратном порядке), частично упорядоченных и случайных последовательностей данных; изучить функции, позволяющие производить оценку длительности времени выполнения алгоритмов.

# Задание:

1. Разработать функции, формирующие упорядоченные (в том числе и в обратном порядке), частично упорядоченные и случайные последовательности целых чисел и чисел с плавающей запятой. При реализации функций считать, что выделение памяти под последовательности происходит вне этих функций, а в качестве формальных параметров функции получают указатель на массив, его размер и, возможно, диапазон изменения величин и длину интервалов (для частично упорядоченных последовательностей). Рекомендуется продумать единый прототип для функций, формирующих последовательности, определить его и использовать массив указателей на функции для автоматизации сбора статистической информации.
2. Оценить длительность формирования последовательностей всех типов для нескольких значений размеров последовательностей (5⋅105 , 10⋅105 , …, 50⋅105 ), и на основе полученных значений построить графики зависимостей длительностей формирования от размера последовательностей.
3. Составить отчет, в котором привести блок-схемы функций (возможно, на примере одной функции), реализующих формирование частично упорядоченных последовательностей, и главной функции, полученные графики зависимостей и выводы по полученным результатам.

# блок-схемы алгоритмов функций формирования частично упорядоченных последовательностей

Изображение выглядит как текст

Автоматически созданное описание

Рисунок . Блок схема функции пилообразного заполнения последовательности

# блок-схема главной функции

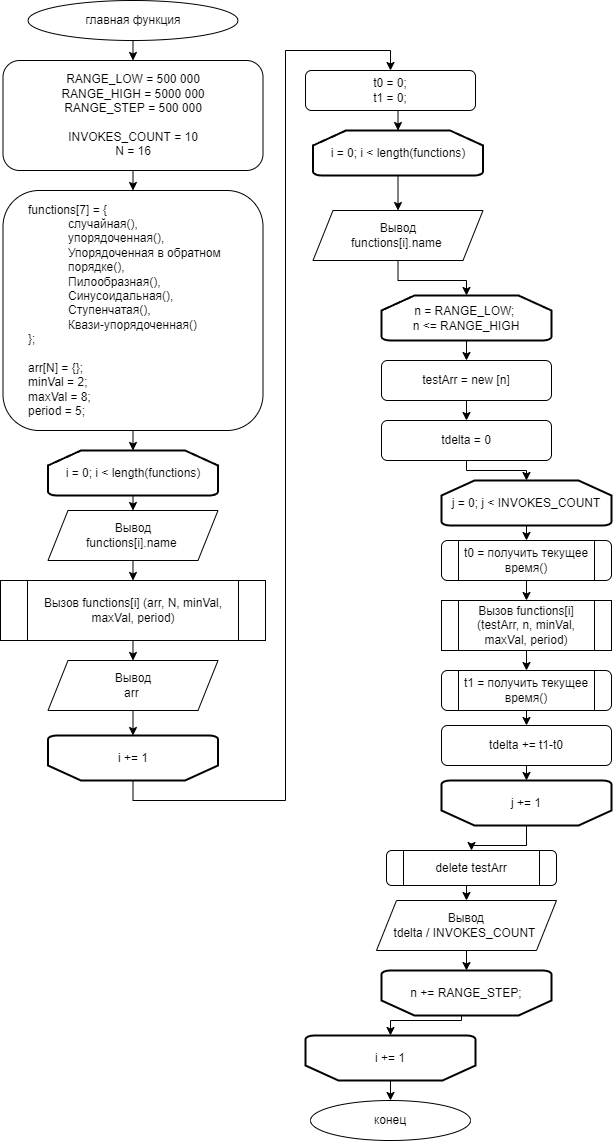


Рисунок . Блок схема главной функции

# результаты измерений длительности формирования последовательностей

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод | Размер последовательности и время заполнения в мс | | | | | | | | | |
|  | 500 000 | 1000 000 | 1500 000 | 2000 000 | 2500 000 | 3000 000 | 3500 000 | 4000 000 | 4500 000 | 5000 000 |
| Случайная | 11 | 18 | 29 | 39 | 48 | 57 | 70 | 76 | 85 | 95 |
| Упорядоченная | 7 | 17 | 23 | 32 | 40 | 48 | 56 | 65 | 73 | 79 |
| Упорядоченная в обратном порядке | 12 | 23 | 35 | 45 | 57 | 70 | 79 | 90 | 103 | 114 |
| Пилообразная | 1 | 4 | 6 | 7 | 10 | 10 | 12 | 15 | 17 | 18 |
| Синусоидальная | 37 | 70 | 107 | 143 | 178 | 215 | 250 | 285 | 323 | 359 |
| Ступенчатая | 18 | 39 | 57 | 76 | 95 | 114 | 132 | 153 | 171 | 189 |
| Квази-упорядоченная | 14 | 28 | 43 | 59 | 73 | 87 | 101 | 117 | 128 | 143 |

Таблица . Зависимость размера длительности формирования от размера последовательностей для типа **double**

Рисунок 3. График зависимости размера длительностей формирования от размера последовательностей для типа **double**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод | Размер последовательности и время заполнения в мс | | | | | | | | | |
|  | 500 000 | 1000 000 | 1500 000 | 2000 000 | 2500 000 | 3000 000 | 3500 000 | 4000 000 | 4500 000 | 5000 000 |
| Случайная | 6 | 14 | 18 | 28 | 32 | 39 | 45 | 53 | 59 | 65 |
| Упорядоченная | 3 | 6 | 9 | 12 | 15 | 18 | 23 | 25 | 28 | 31 |
| Упорядоченная в обратном порядке | 3 | 6 | 9 | 12 | 15 | 20 | 21 | 25 | 28 | 31 |
| Пилообразная | 3 | 6 | 9 | 12 | 15 | 18 | 21 | 25 | 28 | 31 |
| Синусоидальная | 21 | 42 | 64 | 84 | 107 | 126 | 150 | 170 | 192 | 214 |
| Ступенчатая | 9 | 18 | 29 | 39 | 48 | 57 | 68 | 76 | 87 | 96 |
| Квази-упорядоченная | 7 | 17 | 25 | 32 | 40 | 48 | 57 | 65 | 73 | 82 |

Таблица 2 Зависимость размера длительности формирования от размера последовательностей для типа **int**

Рисунок . График зависимости размера длительностей формирования от размера последовательностей для типа **int**

# Листинг кода

#define \_USE\_MATH\_DEFINES

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <math.h>

#include <Windows.h>

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

#define VAR\_TYPE        double      // тип данных для массива

#define MODIFIER        "0.2lf"     // модификатор для вывода массива на экран

#define RANGE\_LOW       500000      // нижняя граница количества эл-тов массива

#define RANGE\_HIGH      5000000     // верхняя граница количества эл-тов массива

#define RANGE\_STEP      500000      // шаг изменения элементов массива

#define INVOKES\_COUNT   10          // кол-во вызовов функции заполнения

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

#define N               16          // кол-во эдлементов для контроля ф-ций

#define FUNCTIONS\_COUNT 7           // кол-во функций заполнения

//////////////////////////////////////////////////////////////////////////////

typedef VAR\_TYPE key\_t;

typedef void (\*fillArrFun\_t)(key\_t\*, size\_t, key\_t, key\_t, int);

typedef struct ftableCell {

    char name[62];

    fillArrFun\_t invoke;

} ftableCell\_t;

// utils

int     random\_int(int min, int max);

double  random\_double(double min, double max);

// buisness logic

void    fill\_arr\_random(key\_t\* arr, size\_t n, key\_t min, key\_t max, int r);

void    fill\_arr\_linear\_upwards(key\_t\* arr, size\_t n, key\_t min, key\_t max, int r);

void    fill\_arr\_linear\_downwards(key\_t\* arr, size\_t n, key\_t min, key\_t max, int r);

void    fill\_arr\_sin(key\_t\* arr, size\_t n, key\_t min, key\_t max, int r);

void    fill\_arr\_sawtooth(key\_t\* arr, size\_t n, key\_t min, key\_t max, int r);

void    fill\_arr\_stepped(key\_t\* arr, size\_t n, key\_t min, key\_t max, int r);

void    fill\_arr\_quasi\_ordered(key\_t\* arr, size\_t n, key\_t min, key\_t max, int r);

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

int main() {

    srand(time(NULL));

    system("chcp 1251 > nul");

    key\_t arr[N];

    const ftableCell\_t functionsTable[FUNCTIONS\_COUNT] = {

        { "Случайная",                          fill\_arr\_random             },

        { "Упорядоченная",                      fill\_arr\_linear\_upwards     },

        { "Упорядоченная в обратном порядке",   fill\_arr\_linear\_downwards   },

        { "Пилообразная",                       fill\_arr\_sawtooth           },

        { "Синусоидальная",                     fill\_arr\_sin                },

        { "Ступенчатая",                        fill\_arr\_stepped            },

        { "Квази-упорядоченная",                fill\_arr\_quasi\_ordered      }

    };

    for (int i = 0; i < FUNCTIONS\_COUNT; ++i) {

        printf("Function: %s\n", functionsTable[i].name);

        functionsTable[i].invoke(arr, N, 2, 8, 5);

        for (int j = 0; j < FUNCTIONS\_COUNT; j++) printf("%"MODIFIER" ", arr[j]);

        printf("\n");

    }

    printf("\n");

    ///////////////////////////////////////////////////////////////////////////

    DWORD t0, t1, tdelta;

    // вывод в csv файл, для перставления данных в виде таблицы

    FILE\* csv\_file = fopen("out.csv", "w");

    if (csv\_file == NULL) printf("\a[ERR] Cannot open a file!\n");

    for (int n = RANGE\_LOW; n <= RANGE\_HIGH; n += RANGE\_STEP)

        fprintf(csv\_file, ";%d", n);

    fprintf(csv\_file, "\n");

    for (int i = 0; i < FUNCTIONS\_COUNT; i++) {

        printf("Function: %s\n"

               "Time delta: ", functionsTable[i].name);

        fprintf(csv\_file, "%s;", functionsTable[i].name);

        for (size\_t n = RANGE\_LOW; n <= RANGE\_HIGH; n += RANGE\_STEP) {

            key\_t\* test\_arr = (key\_t\*)malloc(sizeof(key\_t) \* n);

            tdelta = 0;

            for (int j = 0; j < INVOKES\_COUNT; j++) {

                t0 = GetTickCount();

                functionsTable[i].invoke(test\_arr, n, 2, 8, 5);

                t1 = GetTickCount();

                tdelta += t1 - t0;

            }

            free(test\_arr);

            fprintf(csv\_file, "%d;", tdelta / INVOKES\_COUNT);

            printf("%d ", tdelta / INVOKES\_COUNT);

        }

        fprintf(csv\_file, "\n");

        printf("\n");

    }

    fclose(csv\_file);

    printf("end");

    return 0;

}

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

int random\_int(int min, int max) {

    return rand() % (max - min) + min;

}

double random\_double(double min, double max) {

    return ((double)rand() / RAND\_MAX) \* (max - min) + min;

}

void fill\_arr\_random(key\_t \* arr, size\_t n, key\_t min, key\_t max, int r) {

    if (!arr) return;

    for (int i = 0; i < n; ++i) {

        arr[i] = (key\_t)random\_double(min, max);

    }

}

void fill\_arr\_linear\_upwards(key\_t \* arr, size\_t n, key\_t min, key\_t max, int r) {

    if (!arr) return;

    key\_t delta = (max - min);

    for (int i = 0; i < n; ++i) {

        arr[i] = delta \* (i / (double)(n-1)) + min;

    }

}

void fill\_arr\_linear\_downwards(key\_t \* arr, size\_t n, key\_t min, key\_t max, int r) {

    if (!arr) return;

    key\_t delta = (max - min);

    for (int i = 0; i < n; i++) {

        arr[i] = delta \* ((double)(n - 1 - i) / (n-1)) + min;

    }

}

void fill\_arr\_sin(key\_t\* arr, size\_t n, key\_t min, key\_t max, int r) {

    if (!arr) return;

    if (r == 0) return;

    double delta = max - min;

    double amplitude = delta / 2.0;

    for (int i = 0; i < n; ++i) {

        arr[i] = cos((i \* M\_PI \* 2) / r) \* amplitude + min + amplitude;

    }

}

void fill\_arr\_sawtooth(key\_t\* arr, size\_t n, key\_t min, key\_t max, int r) {

    if (!arr) return;

    if (r == 0) return;

    double delta = max - min;

    for (int i = 0; i < n; ++i) {

        arr[i] = ((double)(i % (r+1)) / r) \* delta + min;

    }

}

void fill\_arr\_stepped(key\_t\* arr, size\_t n, key\_t min, key\_t max, int r) {

    if (!arr) return;

    if (r == 0) return;

    double delta = max - min;

    double period = n / r;

    for (int i = 0; i < n; ++i) {

        arr[i] = delta \* ((int)( (i / (double)(n-1)) \* period) / period) +

                 random\_double(0.0, period / 4)

                 + min;

    }

}

void fill\_arr\_quasi\_ordered(key\_t\* arr, size\_t n, key\_t min, key\_t max, int r) {

    if (!arr) return;

    key\_t delta = (max - min);

    for (int i = 0; i < n; ++i) {

        arr[i] = delta \* (i / (double)(n-1) + random\_double(0.0, 0.4) - 0.2 ) + min;

    }

}

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

# снимки экрана окна

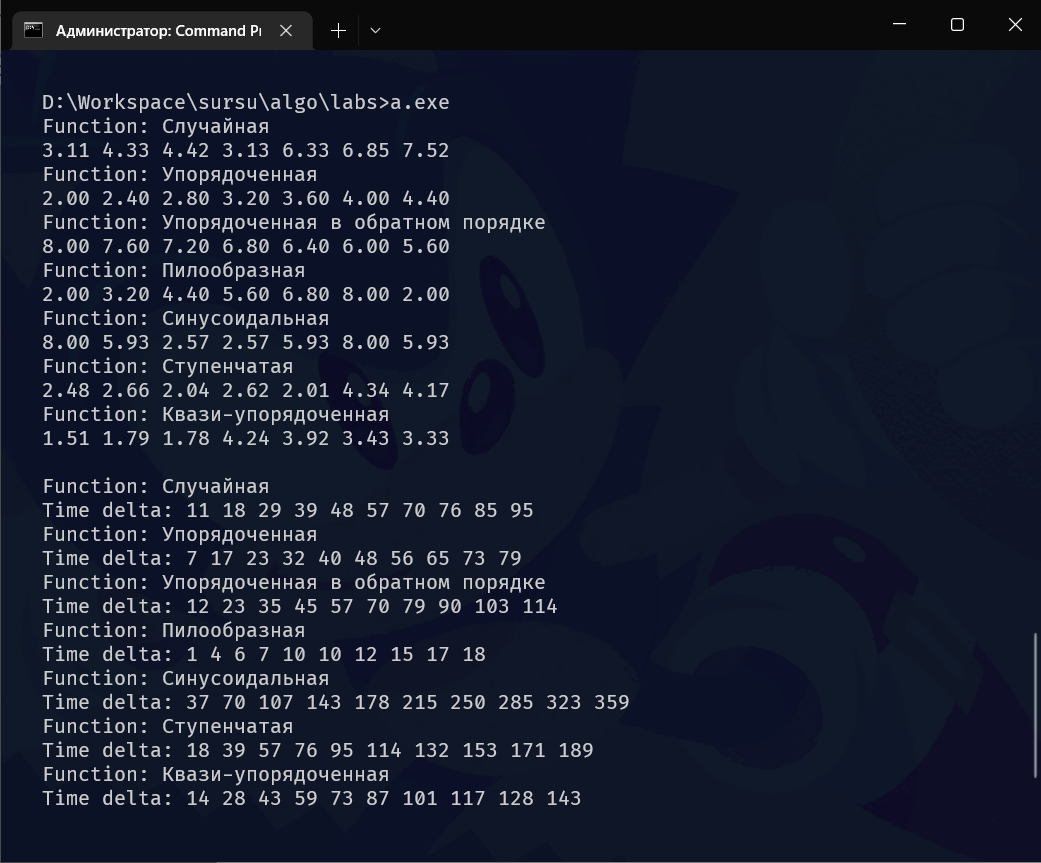


Рисунок . Демонстрация работы программы

# Вывод

В результате выполнения лабораторной работы были освоены принципы формирования монотонных, частично упорядоченных и случайных последовательностей данных, а также изучены функции, позволяющие проводить оценку длительности времени выполнения алгоритмов.

На основе полученных данных, было выявлено, что самый быстрый вариант заполнения последовательности – ступенчатый, самый медленный и значительно зависящий от размера последовательности – синусоидальный.