**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №6**

**по дисциплине «Организация ЭВМ и систем»**

**Тема: Организация связи Ассемблера с ЯВУ на примере программы построения частотного распределение попаданий псевдослучайных целых чисел в заданные интервалы.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1304 |  | Чернякова В.А. |
| Преподаватель |  | Кирьянчиков В.А. |

Санкт-Петербург

2022

## Цель работы.

Написать программу построения частотного распределения попаданий псевдослучайных целых чисел в заданные интервалы, используя связь ассемблера с ЯВУ.

## Задание.

На языке С программируется ввод с клавиатуры и контроль исходных данных, а также генерируется массив псевдослучайных целых чисел, изменяющихся в заданном диапазоне и имеющих заданный закон распределения. Необходимые датчики псевдослучайных чисел находятся в каталоге RAND\_GEN (пpи его отсутствии получить у пpеподавателя).

Следует привести числа к целому виду с учетом диапазона изменения.

Далее должны вызываться 1 или 2 ассемблерных процедуры для формирования распределения количества попаданий псевдослучайных целых чисел в заданные интервалы. Ассемблерные процедуры должны вызываться как независимо скомпилированные модули. Передача параметров в процедуру должна выполняться через кадр стека.

Результирующий массив частотного распределения чисел по интервалам, сформированный на ассемблерном уровне, возвращается в программу, реализованную на ЯВУ, и затем сохраняется в файле и выводится на экран средствами ЯВУ.

## Ход работы.

*Вариант 28.*

В начале программы согласно условиям задачи, происходит считывание всех необходимых данных, которые вводит пользователь, с их обязательной проверкой. В случае некорректного ввода программа завершается и выводит ошибку.

Конкретно обрабатываемые ошибки, согласно вариантую Соблюдение требований: Nint >= Dx, Lgi <= Xmin – хотя бы один, ПГпосл > Xmax.

Далее происходит сортировка левых границ пузырьком.

*for (int i = 0; i < n\_int + 1; i++) {*

*for (int j = i; j < n\_int + 1; j++) {*

*if (intervals[i] > intervals[j]) {*

*std::swap(intervals[i], intervals[j]);*

*}*

*}*

*}*

Дальше происходит инициализация массива рандомными числами в пределах [Xmin,Xmax]. Для этого используются функции из папки RAND\_GEN. Сначала происходит начальная инициализация dnk\_randomize(). Затем, согласно варианту, с помощью функции dnk\_normal происходит инициализация каждого элемента нормальным распределением. В качестве мат. ожидания берется среднее арифметическое между Xmin и Xmax, а дисперсия — треть от разности Xmax и Xmin. При таких значениях шанс, что число попадет в нужный диапазон составляет 99.72%. Чтобы исключить ситуацию попадания случайного числа вне интервала, происходит проверка, что число попала в заданный интервал, если число не попало, то происходит дополнительная итерация прохода цикла.

*dnk\_randomize();//начальная иницализация*

*for (int i = 0; i < n; i++) {*

*numbers[i] = (int) round(dnk\_normal((x\_max+x\_min)/2.0, (x\_max-x\_min)/3.0));//вычисление рандомного числа*

*if (numbers[i]>x\_max || numbers[i]<x\_min)*

*i--;*

*std::cout << numbers[i] << " ";*

*}*

*std::cout << std::endl;*

Далее после инициализации нулями 2 массивов (один нужен для записи результата 1 функции, другой второй функции) происходит выполнение ассемблерных функций и вывод результатов как в файл, так и в консоль.

First.asm.

Это функция предназначена для помещения псевдослучайных чисел в единичный интервал. Функция принимает массив псевдослучайных чисел, его длину, результирующий массив, в который будет записан результат и Xmin (минимально возможное случайное число). Все они объявлены как dword, так как являются значениями int(4 байта).Поэтому в программе будут использоваться расширенные регистры. Происходит сохранение в стек расширенных регистров и затем помещение в эти регистры наших переменных.

*.586*

*.MODEL FLAT, C*

*.CODE*

*PUBLIC C first*

*first PROC C numbers: dword, n: dword, result1: dword, x\_min: dword*

*push esi;сохранение регистров*

*push edi*

*mov esi, numbers;адрес 1 элемента массива случайных чисел*

*mov edi, result1;адрес 1 элемента результата*

*mov ecx, n;длина массива*

Далее идет цикл. В нем обрабатывается очередной элемент, в eax записывается число, из него вычитается Xmin и таким образом получается индекс отрезка, в который должен попасть элемент. Далее в ebx записываем количество чисел, которые попали в этот отрезок до этого, добавляем единицу в ebx и записываем получившийся результат. Далее после окончания цикла восстанавливаются регистры.

*PROCESSING:*

*mov eax, [esi];запись случайного числа*

*sub eax, x\_min;получение индекса для итогового массива*

*mov ebx, [edi+4\*eax];получение элемента итогового массива*

*inc ebx;число попало в ед отрезок=>увеличиваем количесвто на единицу*

*mov [edi+4\*eax], ebx;запись в соот ячейку*

*add esi, 4;переходим к след элементу*

*loop PROCESSING;вычесть ecx и если ecx=0 завершить цикл*

*pop edi;восстановление регистров*

*pop esi*

*ret*

*first\_dist endp*

*end*

Second.asm.

Это функция предназначена для распределения в введённые пользователям границы массива псевдослучайных чисел. Функция принимает массив, полученный после выполнения функции first, а также сами интервалы и массив, в который будет записан результат. Сохраняются расширенные регистры в стек, а также помещаются переменные в регистры.

*.586*

*.MODEL FLAT, C*

*.CODE*

*PUBLIC C second*

*second PROC C result1:dword, intervals: dword, result2: dword, x\_max: dword, x\_min: dword, n\_int: dword*

*push esi*

*push edi*

*mov esi, intervals*

*mov edi, result2*

*mov ecx, n\_int*

Далее идет основной цикл, в eax и ebx записывается левая и правая граница очередного интервала соответственно. Затем происходит проверка, больше ли очередная левая граница минимального возможного случайного числа. Если так, то в метке LEFTGRMOREXMIN проверяется, является ли отрезок нулевым, и если нет, то вычитается из этой границы Xmin и происходит переход к метке PROCESSNOZEROINTERVAL. Если отрезок является нулевым, то происходит переход к метке финальной метке NEXTELEM. Если отрезок меньше Xmin, то отчет будет проводиться от 1 левой границе и произойдет переход к метке PROCESSNOZEROINTERVAL.

*PROCESSING:*

*mov eax, [esi] ; левая граница интервала*

*mov ebx, [esi + 4] ; правая граница*

*cmp eax, x\_min ; если левая граница >= x\_min*

*jge LEFTGRMOREXMIN*

*mov eax, 0 ; иначе, eax = 0, начало массива result1*

*sub ebx, x\_min ; если длина интервала = 0*

*jle NEXTELEM*

*jmp PROCESSNOZEROINTERVAL;иначе перейти к обработке не 0 интервала*

*LEFTGRMOREXMIN:*

*sub ebx, eax ; количество элементов в интервале*

*cmp ebx, 0 ; если длина интервала = 0*

*je NEXTELEM*

*sub eax, x\_min ; индекс первого элемента из текущего интервала в массиве result1*

Затем в метке PROCESSNOZEROINTERVAL происходит основная обработка. Сохраняются нужные регистры, высчитывается сколько единичных отрезков нужно пройти, и во вложенном цикле SUMELEM в интервал, который задал пользователь, добавляется очередное количество попаданий случайного числа в единичном интервале.

*PROCESSNOZEROINTERVAL:*

*push esi ;сохраняем регистры*

*push ecx*

*mov ecx, ebx ; количество элементов из result1 по которым нужно пройти*

*mov esi, result1 ; массив*

*mov ebx, 0 ; считает сумму подходящих элементов*

*SUMELEM: ; цикл, считает сумму элементов, входящих в интервал*

*add ebx, [esi + 4\*eax];к ebx добавляем количество попаданий в очередной еденичный интервал*

*inc eax*

*loop SUMELEM*

Затем восстанавливаются регистры и проверяется, является ли очередной интервал последним, если да, то добавляем еще элемент правой скобки, иначе в результирующий массив в соответствующую ячейку записывается результат.

Далее в метке NEXTELEM происходит добавление соответствующим индексным регистрам 4(длины двойного слова). После чего восстанавливаются регистры и функция завершается.

*pop ecx;восстановление регистра*

*cmp ecx, 1 ; если обрабатывали не последний элемент, то записываем сумму в массив результат*

*jne RECORDING*

*add ebx, [esi + 4\*eax] ; иначе добавляем еще элемент последней правой скобки*

*RECORDING:*

*mov [edi], ebx ; записываем результат*

*pop esi;восстановление регистра*

*jmp NEXTELEM*

*NEXTELEM:*

*add edi, 4 ; двигаемся к след. элементам массивов*

*add esi, 4*

*loop PROCESSING*

*pop edi*

*pop esi*

*ret*

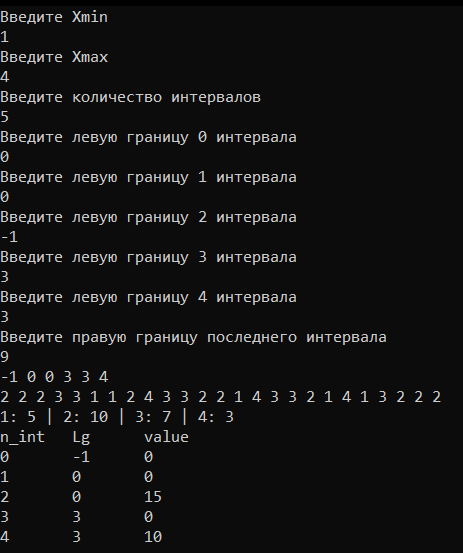
*second\_dist ENDP*

*END*

## Тестирование.

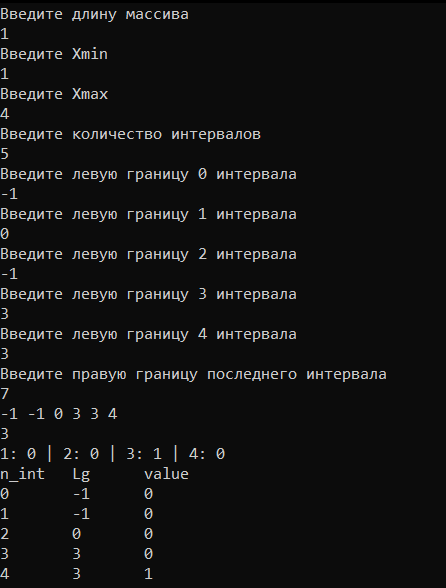
Тест 1.

Проверка базового случая



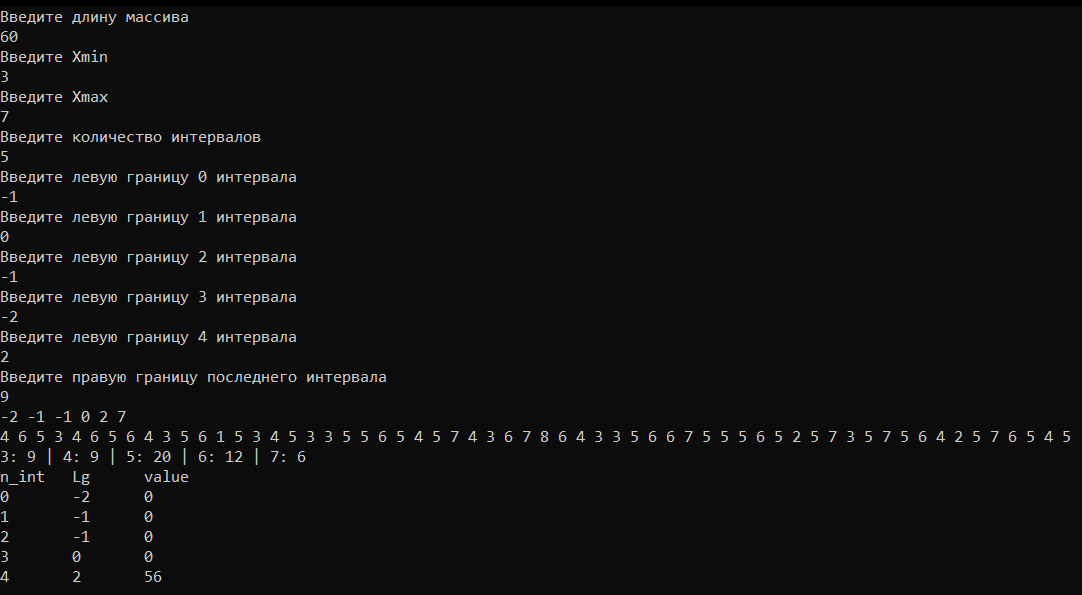
Тест 2.

Проверка вырожденного случая, когда 1 элемент у псевдослучайного массива



Тест 3.

Случай, когда все левые границы меньше Xmin.



Числа не должны находиться вне диапазона [Xmin; Xmax]

# ПРИЛОЖЕНИЕ.

Исходный код программы.

Файл lb6.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <fstream>

#include <string>

#include <random>

#include "Randomer.h"

#include <locale>

extern "C" void first(int\* numbers, int n, int\* result1, int x\_min);

extern "C" void second(int\* result1, int\* intervals, int\* result2, int x\_max, int x\_min, int n\_int);

int main(){

setlocale(LC\_ALL, "Russian");

int n, x\_min, x\_max, r\_last, n\_int;

std::cout << "Введите длину массива" << std::endl;

std::cin >> n;

if (n <= 0 || n > 16384) {//проверка на корретный ввод массива

std::cout << "0 < n <= 16384!!" << std::endl;

return 0;

}

std::cout << "Введите Xmin" << std::endl;

std::cin >> x\_min;

std::cout << "Введите Xmax" << std::endl;

std::cin >> x\_max;

if (x\_max <= x\_min) {

std::cout << "Xmax > Xmin!" << std::endl;

return 0;

}

std::cout << "Введите количество интервалов" << std::endl;

std::cin >> n\_int;

int Dx = x\_max - x\_min;

if (n\_int <= 0 || n\_int > 24 || n\_int < Dx) {

std::cout << "0 < n\_int <= 24 и n\_int > Dx, где Dx = Xmax - X min = " << x\_max << " - " << x\_min << " = " << Dx << std::endl;

system("Pause");

return 0;

}

int cnt\_lg = 0;

int\* intervals = new int[n\_int + 1];

for (int i = 0; i < n\_int; i++) {

std::cout << "Введите левую границу " << i << " интервала" << std::endl;

std::cin >> intervals[i];

// Такого условия в задании нет

if (intervals[i] > x\_max) {

std::cout << "Введена некорректная левая граница! Она больше x\_max";

return 0;

}

if (intervals[i] <= x\_min) {

cnt\_lg++;

}

}

if (cnt\_lg == 0) {

std::cout << "Ни одна левая граница не удовлетворяет условию: Lgi <= Xmin !" << std::endl;

return 0;

}

std::cout << "Введите правую границу последнего интервала" << std::endl;

std::cin >> r\_last;

if (r\_last <= x\_max) {

std::cout << "Не выполнено условие Rg > Xmax" << std::endl;

return 0;

}

// Если я правильно понимаю – intervals[n\_int] - правая граница последнего интервала. В таком случае не понятно, почему ей присваивается значение x\_max. Значение r\_last в программе не используется вообще

intervals[n\_int] = x\_max;

for (int i = 0; i < n\_int+1; i++) {

for (int j = i; j < n\_int+1; j++) {

if (intervals[i] > intervals[j]) {

std::swap(intervals[i], intervals[j]);

}

}

}

for (int i = 0; i < n\_int + 1; i++) {

std::cout << intervals[i] << " ";

}

std::cout << std::endl;

int\* numbers = new int[n];

dnk\_randomize();//начальная иницализация

for (int i = 0; i < n; i++) {

numbers[i] = (int)round(dnk\_normal((x\_max + x\_min) / 2.0, (x\_max - x\_min) / 3.0));//вычисление рандомного числа

std::cout << numbers[i] << " ";

}

std::cout << std::endl;

int\* result1 = new int[abs(x\_max - x\_min) + 1];

int\* result2 = new int[n\_int];

for (int i = 0; i < abs(x\_max - x\_min) + 1; i++) {

result1[i] = 0;

}

for (int i = 0; i < n\_int; i++) {

result2[i] = 0;

}

first(numbers, n, result1, x\_min);

//for (int i = 0; i < n; i++)

//printf("%d ", numbers[i]);

for (int i = 0; i < abs(x\_max - x\_min); i++) {

std::cout << i + x\_min << ": " << result1[i] << " | ";

}

std::cout << std::to\_string(abs(x\_max - x\_min) + x\_min) << ": " << result1[abs(x\_max - x\_min)] << std::endl;

second(result1, intervals, result2, x\_max, x\_min, n\_int);

std::ofstream file("table.txt");

auto head = "n\_int\tLg\tvalue";

file << head << std::endl;

std::cout << head << std::endl;

for (int i = 0; i < n\_int; i++) {

auto line = std::to\_string(i) + "\t" + std::to\_string(intervals[i]) + "\t" + std::to\_string(result2[i]) + "\n";

file << line;

std::cout << line;

}

return 0;

}

Файл first.asm

.586

.MODEL FLAT, C

.CODE

PUBLIC C first

first PROC C numbers: dword, n: dword, result1: dword, x\_min: dword

push esi;сохранение регистров

push edi

mov esi, numbers;адрес 1 элемента массива случайных чисел

mov edi, result1;адрес 1 элемента результата

mov ecx, n;длина массива

PROCESSING:

mov eax, [esi];запись случайного числа

sub eax, x\_min;получение индекса для итогового массива

mov ebx, [edi+4\*eax];получение элемента итогового массива

inc ebx;число попало в ед отрезок=>увеличиваем количесвто на единицу

mov [edi+4\*eax], ebx;запись в соот ячейку

add esi, 4;переходим к след элементу

loop PROCESSING;вычесть ecx и если ecx=0 завершить цикл

pop edi;восстановление регистров

pop esi

ret

first endp

end

Файл second.asm

.586

.MODEL FLAT, C

.CODE

PUBLIC C second

second PROC C result1:dword, intervals: dword, result2: dword, x\_max: dword, x\_min: dword, n\_int: dword

push esi

push edi

mov esi, intervals

mov edi, result2

mov ecx, n\_int

PROCESSING:

mov eax, [esi] ; левая граница интервала

mov ebx, [esi + 4] ; правая граница

cmp eax, x\_min ; если левая граница >= x\_min

jge LEFTGRMOREXMIN

mov eax, 0 ; иначе, eax = 0, начало массива result1

sub ebx, x\_min ; если длина интервала = 0

jle NEXTELEM

jmp PROCESSNOZEROINTERVAL;иначе перейти к обработке не 0 интервала

LEFTGRMOREXMIN:

sub ebx, eax ; количество элементов в интервале

cmp ebx, 0 ; если длина интервала = 0

je NEXTELEM

sub eax, x\_min ; индекс первого элемента из текущего интервала в массиве result1

PROCESSNOZEROINTERVAL:

push esi ;сохраняем регистры

push ecx

mov ecx, ebx ; количество элементов из result1 по которым нужно пройти

mov esi, result1 ; массив

mov ebx, 0 ; считает сумму подходящих элементов

SUMELEM: ; цикл, считает сумму элементов, входящих в интервал

add ebx, [esi + 4\*eax];к ebx добавляем количество попаданий в очередной еденичный интервал

inc eax

loop SUMELEM

pop ecx;восстановление регистра

cmp ecx, 1 ; если обрабатывали не последний элемент, то записываем сумму в массив результат

jne RECORDING

add ebx, [esi + 4\*eax] ; иначе добавляем еще элемент последней правой скобки

RECORDING:

mov [edi], ebx ; записываем результат

pop esi;восстановление регистра

jmp NEXTELEM

NEXTELEM:

add edi, 4 ; двигаемся к след. элементам массивов

add esi, 4

loop PROCESSING

pop edi

pop esi

ret

second ENDP

END