

**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.**

**Студент гр. 0383**

**Желнин М.Ю.**

**Преподаватель**

**Ефремов М.А.**

**Санкт-Петербург**

**2021**

### **Цель работы.**

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.

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.

Исходные данные:

1. Длина массива псевдослучайных целых чисел - NumRanDat ( $\leq 16K$ )
2. Диапазон изменения массива псевдослучайных целых чисел  $[Xmin, Xmax]$  (м.б. биполярный, например,  $[-100, 100]$ )
3. Массив псевдослучайных целых чисел  $\{Xi\}$ .
4. Количество интервалов, на которые разбивается диапазон изменения массива псевдослучайных целых чисел - NInt ( $\leq 24$ )
5. Массив левых границ интервалов разбиения LGrInt .

В общем случае интервалы разбиения диапазона изменения псевдослучайных чисел могут иметь различную длину, левые границы могут задаваться в произвольном порядке и иметь произвольные значения. Если  $Xmin < LGrInt(1)$ , то часть данных не будет участвовать в

формировании распределения. Каждый интервал, кроме последнего, следует интерпретировать как  $[LGrInt(i), LGrInt(i+1))$ . Если у последнего интервала правая граница меньше  $X_{max}$ , то часть данных не будет участвовать в формировании распределения.

Результаты:

Текстовая таблица, строка которой содержит:

- номер интервала,
- левую границу интервала,
- количество псевдослучайных чисел, попавших в интервал.

Количество строк должно быть равно числу интервалов разбиения.

### **Задание.**

**Вариант №3.** Равномерное распределение, 2 ассемблерных процедуры,  $N_{int} < D_x$ ,  $Lg1 > X_{min}$ ,  $ПГ_{посл} \leq X_{max}$ .

### **Ход работы.**

В ходе работы было реализовано 3 модуля, 1 на языке C++, и 2 на ассемблере. На C++ написан `ASM_LABsix.cpp`, который спрашивает у пользователя данные и проверяет их корректность, выводит результат в консоль и файл, а также вызывает 2 модуля на ассемблере, передавая туда собранные у пользователя данные.

Первый модуль на ассемблере распределяет сгенерированный массив чисел по единичным отрезкам, то есть в цикле `loop` проходится по всем числам, и прибавляет единицу в соответствующий отрезок.

Второй модуль формирует распределение на основе первого, но уже по заданным пользователем интервалам. Сначала мы находим отрезок из первого модуля соответствующий левой границе интервала, а потом проходимся по всем элементам до правой границы и прибавляем 1, если он не равен 0.

Текст исходных файлов представлен в Приложении А.

## Тестирование.

Таблица 1 — тестирование.

№	Входные данные	Выходные данные	Комментарии
1	n_size = 10 xmin = -5 xmax = 5 inter_size = 5 LGrInt = [-3, -1, 1, 3, 4]	Сгенерированные значения 2 3 3 1 5 4 5 1 4 3  Результат: № Граница Количество чисел 1 -3 0 2 -1 0 3 1 3 4 3 3 5 4 4	<b>Верно.</b>
2	n_size = 10 xmin = 0 xmax = 20 inter_size = 10 LGrInt = [1 2 3 4 5 6 7 8 9 15]	Сгенерированные значения 20 13 4 9 2 7 5 4 8 9  Результат: № Граница Количество чисел 1 1 0 2 2 1 3 3 0 4 4 2 5 5 1 6 6 0 7 7 1 8 8 1 9 9 3 10 15 1	<b>Верно.</b>

## Вывод.

В ходе выполнения данной лабораторной работы была изучена организация связи ассемблера с ЯВУ. Была реализована программа частотного распределения случайных чисел по заданным интервалам на языке C++ с использованием ассемблерных модулей.

## Приложение А.

### Исходный код программы.

#### ASM\_LABsix.cpp

```
#include <iostream>
#include <fstream>
#include <random>

using namespace std;

extern "C" void one(int* numbers, int n_size, int* result, int xmin);
extern "C" void two(int* array, int array_size, int xmin, int* intervals, int inter_size, int* result);

int main() {
    setlocale(0, "Russian");
    srand(time(NULL));
    ofstream result("result.txt");

    int n_size;
    int* numbers;
    int xmin, xmax;
    int inter_size;
    int* intervals;
    int* intervals2;
    int* result1;
    int* result2;

    cout << "Введите количество чисел:\n";
    cin >> n_size;
    if (n_size > 16 * 1024) {
        cout << "Количество чисел должно быть меньше или равно, чем 16*1024\n";
        return 0;
    }
    cout << "Введите xmin и xmax:\n";
    cin >> xmin >> xmax;
    int Dx = xmax - xmin;
    if (Dx > 24) {
        cout << "Xmax и Xmin не должны отличаться больше чем на 24\n";
        return 0;
    }
}
```

```

}

cout << "Введите число границ:\n";
cin >> inter_size;

if (inter_size > 24) {
    cout << "Число интервалов должно быть меньше или равно 24\n";
    return 0;
}
if (inter_size >= Dx) {
    cout << "Число интервалов должно быть меньше dx\n";
    return 0;
}
numbers = new int[n_size];
intervals = new int[inter_size];
intervals2 = new int[inter_size];

int lenmod1 = abs(xmax - xmin) + 1;
result1 = new int[lenmod1];
for (int i = 0; i < lenmod1; i++)
    result1[i] = 0;

result2 = new int[inter_size + 1];
for (int i = 0; i < inter_size + 1; i++)
    result2[i] = 0;

cout << "Введите все границы:\n";
for (int i = 0; i < inter_size; i++) {
    cin >> intervals[i];
    if (intervals[i] < xmin) {
        cout << "Левая граница должна быть больше либо равна Xmin\n";
        return 0;
    }
    intervals2[i] = intervals[i];
}
}

```

```
std::random_device rd;
std::mt19937 gen(rd());
std::uniform_int_distribution<> dis(xmin, xmax);
for (int i = 0; i < n_size; i++) numbers[i] = dis(gen);
```

```
cout << "Сгенерированные значения\n";
result << "Сгенерированные значения\n";
for (int i = 0; i < n_size; i++) {
    cout << numbers[i] << ' ';
    result << numbers[i] << ' ';
}
cout << '\n';
cout << '\n';
result << '\n';
result << '\n';
```

```
one(numbers, n_size, result1, xmin);
```

```
two(result1, n_size, xmin, intervals, inter_size, result2);
```

```
cout << "Результат:\n";
result << "Результат:\n";
cout << "№\tГраница\tКоличество чисел" << endl;
result << "№\tГраница\tКоличество чисел" << endl;
```

```
for (int i = 1; i < inter_size + 1; i++) {
    cout << i << "\t" << intervals2[i - 1] << '\t' << result2[i] << endl;
    result << i << "\t" << intervals2[i - 1] << '\t' << result2[i] << endl;
}
```

```
delete[] numbers;
delete[] intervals;
delete[] intervals2;
delete[] result1;
delete[] result2;
```

```
        return 0;
    }
}
```

## **one.asm**

```
.586p
.MODEL FLAT, C
.CODE
PUBLIC C one
one PROC C array: dword, arraysize: dword, res: dword, xmin: dword
```

```
    push esi
    push edi
```

```
    mov edi, array
    mov ecx, arraysize
    mov esi, res
```

```
for_numbers:
    mov eax, [edi]
    sub eax, xmin
    mov ebx, [esi + 4*eax]
    inc ebx
    mov [esi + 4*eax], ebx
    add edi, 4
    loop for_numbers
```

```
    pop edi
    pop esi
```

```
    ret
one ENDP
END
```

## **two.asm**

```
.586p
.MODEL FLAT, C
.CODE
PUBLIC C two
two PROC C array: dword, array_size: dword, xmin: dword, borders: dword, intN: dword, result: dword
```



```
push esi
push edi
push ebp
```

```
mov edi, array
mov esi, borders
mov ecx, intN
```

```
for_borders:
    mov eax, [esi]
    sub eax, xmin
    mov [esi], eax
    add esi, 4
    loop for_borders
```

```
mov esi, borders
mov ecx, intN
mov ebx, 0
mov eax, [esi]
```

```
for_1:
    push ecx
    mov ecx, eax
    push esi
    mov esi, result
```

```
for_arr:
    cmp ecx, 0
    je end_for
    mov eax, [edi]
    add [esi + 4*ebx], eax
    add edi, 4
    loop for_arr
```

```
end_for:
    pop esi
    inc ebx
    mov eax, [esi]
```

```
    add esi, 4
    sub eax, [esi]
    neg eax
    pop ecx
    loop for_1
```

```
mov esi, result
mov ecx, intN
mov eax, 0
```

```
fin_for:
    add eax, [esi]
    add esi, 4
    loop fin_for
```

```
mov esi, result
sub eax, array_size
neg eax
```

```
add [esi + 4*ebx], eax
```

```
pop ebp
pop edi
pop esi
```

```
ret
two ENDP
END
```