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. 0383 _____ Куртова К. А.

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 (при его отсутствии получить у преподавателя).

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 (<= 16K)
2. Диапазон изменения массива псевдослучайных целых чисел
   [Xmin, Xmax] (м.б. биполярный, например, [-100, 100])
3. Массив псевдослучайных целых чисел $\{X_i\}$.
4. Количество интервалов, на которые разбивается диапазон
   изменения массива псевдослучайных целых чисел - NInt (<=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 Xmax, то часть данных не будет участвовать в формировании распределения.

Вариант — 4.

- Вид распределения — нормальный.
- Число ассемблерных процедур — 2.

- $N_{int} < D_x$
- $L_{gi} \leq X_{min}$
- Правая граница последнего интервала $\leq X_{max}$

Ход работы.

На высокоуровневом языке C++ производится считывание начальных значений и их проверка, а также генерация чисел с нормальным распределением (с использованием библиотеки random).

В реализации программы используется два ассемблерных модуля. Первый модуль содержит функцию *first*, которая считает количество каждого из чисел исходного массива на единичном интервале и записывает это количество в промежуточный массив. Второй модуль содержит функцию *second*, которая на основе промежуточного массива считает количество чисел в промежутках.

Вывод производится на ЯВУ, также результаты дублируются в файл *output.txt*.

Тестирование программы.

Таблица 1 — Результат тестирования программы

№	Входные данные	Выходные данные	Комментарий
1	NumRunDat = 10 Xmin = -10 Xmax = 10 Nint = 3 left_b = {-5, 3, 5}	Сгенерированные числа: -5 6 8 -1 9 9 -1 -9 2 10 # Граница Количество 1 -5 4 2 3 0 3 5 5	Верно
2	NumRunDat = 15 Xmin = 0 Xmax = 30 Nint = 6	Сгенерированные числа: -2 18 13 9 16 13 13 6 12 15 14 30 21 27 15 # Граница Количество 1 1 0 2 2 0	Верно

	left_b = {1, 2, 3, 4, 5, 6}	3 3 0 4 4 0 5 5 0 6 6 15	
3	NumRunDat = 5 Xmin = -20 Xmax = 0 Nint = 1 left_b = {-20}	Сгенерированные числа: -10 -13 -18 -8 -15 # Граница Количество 1 -20 5	Верно

Выводы.

В ходе лабораторной работы была реализована программа, в которой представлено взаимодействие ассемблерных модулей с программой на ЯВУ (C++). Программа выполняет задачу подсчёта количества случайно сгенерированных чисел в заданных интервалах.

ПРОТОКОЛ

Исходный код программы lab6

lab6.cpp

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

const int MAX_N = 16000;
const int MAX_NINT = 24;

extern "C" void first(int* num, int NumRunDat, int* res, int
Xmin);
extern "C" void second(int* first_res, int NumRunDat, int Xmin,
int* borders, int Nint, int* res);


int main() {
     setlocale(LC_ALL, "rus");

     std::ofstream fout;
     fout.open("output.txt", std::ios_base::out);

     int NumRanDat;
     std::cout << "Введите размер массива: ";
     std::cin >> NumRanDat;
     if (NumRanDat > MAX_N) {
          std::cout << "Размер превышает максимальный допустимый
размер " << MAX_N << "\n";
          return 0;
     }

     int Xmin, Xmax;
     std::cout << "Введите диапазон [Xmin, Xmax]: ";
     std::cin >> Xmin >> Xmax;
     int Dx = Xmax - Xmin;

     int Nint;
     std::cout << "Введите количество интервалов: ";
     std::cin >> Nint;
     if (Nint >= Dx || Nint > MAX_NINT || Nint < 0) {
          std::cout << "Количество интервалов должно быть меньше Dx
= " << Dx << "и больше 0\n";
          return 0;
     }

     int* left_borders = new int[Nint];
     int* saved_borders = new int[Nint];
     std::cout << "Введите левые границы: ";
     for (int i = 0; i < Nint; i++)
          std::cin >> left_borders[i];
```

```
    for (int i = 0; i < Nint - 1; i++) {
        for (int j = i + 1; j < Nint; j++) {
            if (left_borders[j] < left_borders[i])
                std::swap(left_borders[j], left_borders[i]);
        }
    }

    for (int i = 0; i < Nint; i++)
        saved_borders[i] = left_borders[i];

    std::random_device rd;
    std::mt19937 gen(rd());
    std::normal_distribution<> distribution((Xmin + Xmax) / 2,
std::abs(Xmax - Xmin) / 4);
    int* num = new int[NumRanDat];
    for (int i = 0; i < NumRanDat; i++)
        num[i] = std::round(distribution(gen));

    std::cout << "Сгенерированные числа: ";
    fout << "Сгенерированные числа: ";
    for (int i = 0; i < NumRanDat; i++) {
        std::cout << num[i] << " ";
        fout << num[i] << " ";
    }
    std::cout << "\n";
    fout << "\n";

    int len1 = abs(Xmax - Xmin) + 1;
    int* first_res = new int[len1];
    for (int i = 0; i < len1; i++)
        first_res[i] = 0;

    int len2 = Nint + 1;
    int* final_res = new int[len2];
    for (int i = 0; i < Nint + 1; i++)
        final_res[i] = 0;

    first(num, NumRanDat, first_res, Xmin);

    std::cout << "Промежуточные результаты: ";
    for (int i = 0; i < len1; i++)
        std::cout << first_res[i] << " ";
    std::cout << "\n";

    second(first_res, NumRanDat, Xmin, left_borders, Nint,
final_res);

    std::cout << "#\tГраница\tКоличество\n";
    fout << "#\tГраница\tКоличество\n";
    for (int i = 1; i < Nint + 1; i++) {
        std::cout << i << "\t" << saved_borders[i - 1] << "\t" <<
final_res[i] << "\n";
```

```
          fout << i << "\t" << saved_borders[i - 1] << "\t" <<
final_res[i] << "\n";
     }

     delete[] first_res;
     delete[] final_res;
     delete[] left_borders;
     delete[] num;
     fout.close();
}
```

first.asm

```
.586p
.MODEL FLAT, C
.CODE

PUBLIC C first
first PROC C num: dword, N: dword, res: dword, xmin: dword

push esi
push edi

mov edi, num
mov ecx, N                          ; Необходимо для работы loop -
итерация по массиву
mov esi, res                    ; Промежуточный массив

for_additional_res:
     mov eax, [edi]             ; Берем очередной элемент из num
     sub eax, xmin              ; Найдём индекс ячейки,
соответствующей числу в промежуточном массиве
     mov ebx, [esi + 4*eax]     ; Находим ячейку с этим индексом в
промежуточном массиве и помещаем её в ebx
     inc ebx                        ; Увеличиваем значение на 1
     mov [esi + 4*eax], ebx     ; Кладём новое значение обратно в
промежуточный массив
     add edi, 4                     ; Переходим к следующему
элементу в массиве num
     loop for_additional_res

pop edi
pop esi
ret
first ENDP
END
```

second.asm

```
.586p
.MODEL FLAT, C
.CODE
PUBLIC C second
```

```
second PROC C first_res: dword, N: dword, xmin: dword, borders:
dword, Nint: dword, res: dword

push esi
push edi
push ebp

mov edi, first_res
mov esi, borders
mov ecx, Nint

for_borders:
     mov eax, [esi]                ; Достаём очередную границу
интервала
     sub eax, xmin                 ; Находим индекс границы в
промежуточном массиве
     mov [esi], eax                ; Кладём обратно
     add esi, 4                        ; Переходим к следующему
элементу
     loop for_borders

mov esi, borders
mov ecx, Nint
mov ebx, 0                             ; Счётчик
mov eax, [esi]                     ; Первый элемент обновленного
borders

for_start:
     push ecx                      ; Сохраним значение ecx
     mov ecx, eax                  ; Поместим в ecx значение
индекса границы
     push esi                      ; Сохраним esi
     mov esi, res                  ; Будем работать с массивом res

    for_array:
          cmp ecx, 0
          je for_end                    ; Если достигли конца
границы, выходим из цикла
        mov eax, [edi]             ; Берём лежащее в промежуточном
массиве количество элементов
        add [esi + 4*ebx], eax     ; Добавляем к результату для
данного интервала
        add edi, 4                 ; Переходим к следующему
        loop for_array

for_end:
    pop esi                            ; Возвращаемся к массиву
borders
    inc ebx                            ; Увеличиваем счётчик
     mov eax, [esi]
     add esi, 4
```

```
     sub eax, [esi]                ; Из предыдущего значения
borders вычитаем следующий - получаем длину очередного интервала *
-1
     neg eax                       ; Делаем значение
положительным
     pop ecx                       ; Возвращаемся к итератору
Nint
     loop for_start

mov esi, res
mov ecx, Nint
mov eax, 0

final_for:                         ; Считаем количество чисел,
которые не были обработаны
     add eax, [esi]
     add esi, 4
     loop final_for

mov esi, res
sub eax, N
neg eax

add [esi + 4*ebx], eax             ; Помещаем это количество в
последнюю ячейку результата

pop ebp
pop edi
pop esi

ret
second ENDP
END
```