

**МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ**

ОТЧЁТ

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. 1383

Ковалев П. А.

Преподаватель

Ефремов М. А.

Санкт-Петербург

2022

Цель работы

Написание программы реализующего алгоритм построения частотного распределения попаданий псевдослучайных целых чисел в заданные интервалы.

Задание

Вариант 2.

На языке высокого уровня (Pascal или C) генерируется массив псевдослучайных целых чисел, изменяющихся в заданном диапазоне и имеющих равномерное распределение. Необходимые датчики псевдослучайных чисел находятся в каталоге Tasks\RAND_GEN (при его отсутствии программу датчика получить у преподавателя).

Далее должен вызываться ассемблерный модуль(модули) для формирования распределения количества попаданий псевдослучайных целых чисел в заданные интервалы. В общем случае интервалы разбиения диапазона изменения псевдослучайных чисел могут иметь различную длину.

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.

В зависимости от номера бригады формирование частотного распределения должно производиться по одному из двух вариантов:

1. Для бригад с нечетным номером: подпрограмма формирования распределения количества попаданий псевдослучайных целых чисел в заданные интервалы реализуется в виде одного ассемблерного модуля, сразу формирующего требуемое распределение и возвращающего его в главную программу, написанную на ЯВУ;
2. Для бригад с четным номером: подпрограмма формирования распределения количества попаданий псевдослучайных целых чисел в заданные интервалы реализуется в виде двух ассемблерных модулей, первый из которых формирует распределение исходных чисел по интервалам

единичной длины и возвращает его в вызывающую программу на ЯВУ как промежуточный результат. Это распределение должно выводиться в текстовом виде для контроля. Затем вызывается второй ассемблерный модуль, который по этому промежуточному распределению формирует окончательное распределение псевдослучайных целых чисел по интервалам произвольной длины (с заданными границами). Это распределение возвращается в главную программу и выдается как основной результат в виде текстового файла и, возможно, графика.

Выполнение работы

```
[pavel@MyPowerLaptop][~/Documents/Latex/orgEvm_lab6/src]%
Latex/orgEvm_lab6/src via C v12.2.0-gcc
→ echo 16000 -20 20 5 -20 -10 0 15 20 | ./main && cat result.txt
439 402 405 433 433 388 399 398 416 429 420 395 371 399 420 399 404 375 385 375 417 370 363 434 412 432
396 391 361 423 397 376 385 422 392 368 393 392 398 393
-----
1 |      -20 |      3703
-----
2 |      -10 |      3523
-----
3 |         0 |      5554
-----
4 |        15 |      1576
-----
5 |        20 |         0
-----
Latex/orgEvm_lab6/src via C v12.2.0-gcc
→
```

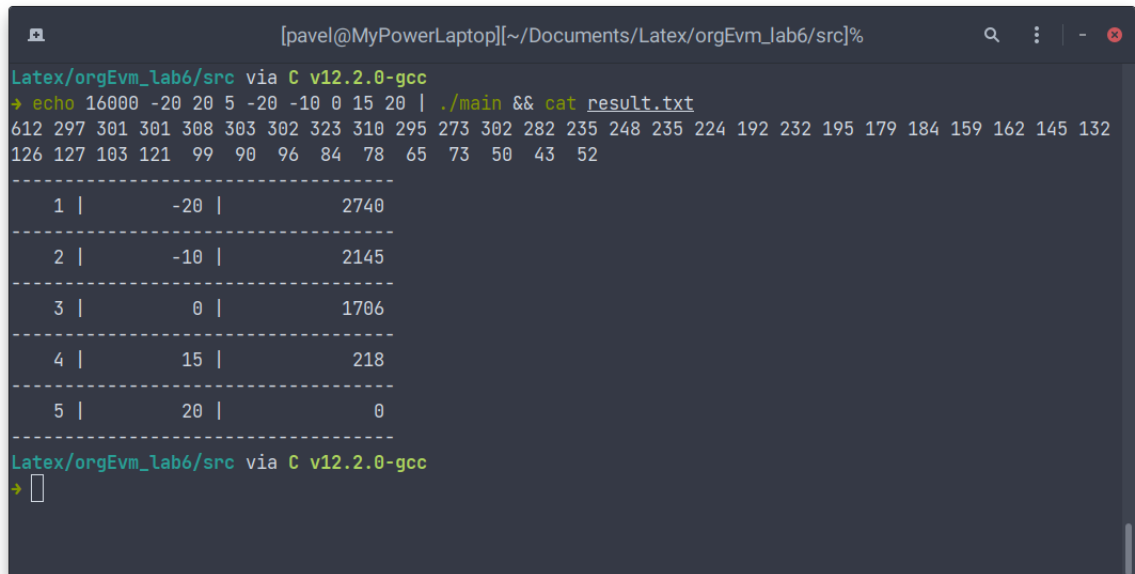
Рис. 1: Пример запуска программы с равномерным распределением

В начале выполнения программы с помощью ЯВУ считываются NumRandat, Xmin, Xmax, Nint и границы интервалов и производится инициализация массива случайных значений и сортировка массива границ по возрастанию. При этом происходит проверка правильности введенных данных в соответствии с условием. Если условия не выполнены, программа завершает свою работу. Иначе, происходит выполнение функций funcAsm1 и funcAsm2, далее запись результатов в файл.

Функция funcAsm1 формирует распределение чисел в массиве случайных чисел по единичным отрезкам в промежутке $[X_{\min}; X_{\max}]$.

Функция funcAsm2 формирует распределение по пользовательским промежуткам из массива borders $[i; i + 1)$. В первом цикле функция форми-

рует массив префиксов. Затем во втором цикле формирует требуемое распределение по промежуткам. Массив с результатом вычислений возвращается в С программу.



```
[pavel@MyPowerLaptop][~/Documents/Latex/orgEvm_lab6/src]%  
Latex/orgEvm_lab6/src via C v12.2.0-gcc  
→ echo 16000 -20 20 5 -20 -10 0 15 20 | ./main && cat result.txt  
612 297 301 301 308 303 302 323 310 295 273 302 282 235 248 235 224 192 232 195 179 184 159 162 145 132  
126 127 103 121 99 90 96 84 78 65 73 50 43 52  
-----  
1 | -20 | 2740  
-----  
2 | -10 | 2145  
-----  
3 | 0 | 1706  
-----  
4 | 15 | 218  
-----  
5 | 20 | 0  
-----  
Latex/orgEvm_lab6/src via C v12.2.0-gcc  
→
```

Рис. 2: Пример запуска программы с нормальным распределением

Вывод

Была написана программа реализующая алгоритм построения частотного распределения попаданий псевдослучайных целых чисел в заданные интервалы.

Приложение А

Исходный код программы

Название файла: main.c

```
#include "random.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void funcAsm1(int *, int64_t *, int, int, int);
void funcAsm2(int64_t *, int *, int *, int, int, int, int);

static int cmp(const void *first, const void *second) {
    int *left = (int *)first;
    int *right = (int *)second;

    if (*left > *right) {
        return 1;
    }
    if (*left < *right) {
        return -1;
    }
    return 0;
}

int main(void) {
    int NumRanDat, Xmin, Xmax, NInt;

    if (isatty(0)) {
        printfВведите(" NumRanDat, Xmin, Xmax, NInt: ");
    }
    scanf("%d %d %d %d", &NumRanDat, &Xmin, &Xmax, &NInt);

    if (NumRanDat > 16384 || NumRanDat < 0) {
        fprintf(stderr, "Error: wrong NumRanDat");
        exit(EXIT_FAILURE);
    }

    if (Xmin >= Xmax) {
        fprintf(stderr, "Error: Xmin > Xmax");
        exit(EXIT_FAILURE);
    }
}
```

```

}

if (NInt > 24 || NInt < 0) {
    fprintf(stderr, "Error: wrong NInt");
    exit(EXIT_FAILURE);
}

if (isatty(0)) {
    printfВведите(" массив границ: ");
}
int *borders = (int *)calloc(NInt, sizeof(int));
for (int i = 0; i < NInt; ++i) {
    scanf("%d", &borders[i]);
}
qsort(borders, NInt, sizeof(int), cmp);

if (borders[0] < Xmin || borders[NInt - 1] > Xmax + 1) {
    fprintf(stderr,
        "Error: wrong borders, Xmin = %d, borders[0] = %d,
Xmax = %d, "
        "borders[NInt] = %d\n",
        Xmin, borders[0], Xmax, borders[NInt - 1]);

    free(borders);

    exit(EXIT_FAILURE);
}

int *arr = (int *)calloc(NumRanDat, sizeof(int));
dnk_randomize();
for (int i = 0; i < NumRanDat; ++i) {
    arr[i] = Xmin + (int)(dnk_random() * (Xmax - Xmin + 1));
    /* arr[i] = Xmin + (int)(dnk_normal(0.0, 1.0) * (Xmax - Xmin)
    / 2); */
}

size_t tmpLen = Xmax - Xmin + 1;
int64_t *tmp = (int64_t *)calloc(tmpLen, sizeof(int64_t));

funcAsm1(arr, tmp, tmpLen, NumRanDat, Xmin);

```

```

free(arr);

for (size_t i = 0; i < tmpLen; ++i) {
    printf("%3ld ", tmp[i]);
}
printf("\n");

int *result = (int *)calloc(NInt, sizeof(int));

funcAsm2(tmp, borders, result, tmpLen, NInt, Xmin, Xmax);

free(tmp);

FILE *f = fopen("result.txt", "w");

fprintf(f, "-----\n");
for (int i = 0; i < NInt; ++i) {
    fprintf(f, " %4d | %10d | %14d \n", i + 1, borders[i], result
[i]);
    fprintf(f, "-----\n");
}

fclose(f);

free(borders);
free(result);
return EXIT_SUCCESS;
}

```

Название файла: lr6asm.s

```

.global funcAsm1
funcAsm1:
    push rbp
    mov rbp, rsp

    push rdi
    push rax
arrLoop:
    mov eax, DWORD PTR [rdi]
    sub eax, r8d
    cmp eax, 0

```

```

# rdi -- int* arr
# rsi -- int64* tmp
# edx -- int (Xmax - Xmin)
# ecx -- int NumRanDat
# r8d -- int Xmin

```

```

    jl skip
    cmp eax, edx
    jg skip
    inc QWORD PTR [rsi + rax * 8]
skip:
    add rdi, 4
    loop arrLoop

    pop rax
    pop rdi
    mov rsp, rbp
    pop rbp
    ret

.global funcAsm2
funcAsm2:                                # rdi -- int64* tmp
    push rbp                            # rsi -- int* borders
    mov rbp, rsp                        # rdx -- int* results
    push r11                            # ecx -- int (Xmax - Xmin)
    push rbx                            # r8d -- int NInt
                                         # r9d -- int Xmin
                                         # r10d -- int Xmax
    mov r10, QWORD PTR [rbp + 2 * 8]

    push rax
    push rcx
    push rdi

    xor rax, rax
suffixLoop:
    add rax, QWORD PTR [rdi]
    mov QWORD PTR [rdi], rax
    add rdi, 8
    loop suffixLoop

    pop rdi
    push rdi

    push rdx
    push rsi

```



```

    mov eax, DWORD PTR [rsi]
    sub eax, r9d
    mov rax, QWORD PTR [rdi + rax * 8]
    mov DWORD PTR [rdx], eax
    add rdx, 4

    mov ecx, r8d
    sub rcx, 2
resultsLoop:
    mov eax, DWORD PTR [rsi]
    sub eax, r9d
    mov rbx, QWORD PTR [rdi + rax * 8]

    mov eax, DWORD PTR [rsi + 4]
    cmp eax, r10d
    jle skip2
    dec eax
skip2:
    sub eax, r9d
    mov r11, QWORD PTR [rdi + rax * 8]

    sub r11, rbx
    mov DWORD PTR [rdx], r11d

    add rdx, 4
    add rsi, 4
    loop resultsLoop

    pop rsi
    pop rdx

    pop rdi
    pop rcx
    pop rax

    pop rbx
    pop r11
    mov rsp, rbp
    pop rbp
    ret

```

Название файла: Makefile

```

##
# LR6
#
# @file
# @version 0.1

CC = gcc
CFLAGS ?= -Wall -Wextra -ggdb -masm=intel

all: main

main: main.o lr6asm.o random.o
    $(CC) $(CFLAGS) -o $@ $^

%.s: %.c
    $(CC) $(CFLAGS) -S -fverbose-asm -o $@ $^

%.o: %.c
    $(CC) $(CFLAGS) -c -o $@ $^

%.o: %.s
    as -msyntax=intel -mnaked-reg -mmnemonic=intel --gstabs -o $@ $^

# %.o: %.asm
#     nasm -f elf64 -o $@ $^

clean:
    rm -f *.o main

.PHONY: all clean

# end

```

Название файла: random.h

/* random.h

Copyright (c) 1995 by D.N.Kirshin, LEEI Reseach Center
All Rights Reserved.

*/
void dnk_randomize(void);

```
double dnk_random(void);  
double dnk_normal(double, double);
```

Название файла: random.c

```
#include "random.h"  
#include <stdlib.h>  
#include <time.h>
```

```
/* random.c
```

```
    Copyright (c) 1995 by D.N.Kirshin, LEEI Reseach Center  
    All Rights Reserved.
```

```
*/
```

```
#define RAND_A 7.74597E-1  
#define RAND_M 1.37438953472E+11
```

```
static double xrand;
```

```
double dnk_random(void) {  
    xrand *= 3125.0;  
    xrand = xrand - ((long)(xrand / RAND_M)) * RAND_M;  
    return xrand / RAND_M;  
}
```

```
double dnk_normal(double M, double S) {  
    int i;  
    double sum = 0.0, xn;  
  
    for (i = 0; i < 5; i++) {  
        xn = (2.0 * dnk_random() - 1.0) * RAND_A;  
        sum += xn;  
    }  
  
    return M + S * sum * (1.0 + 0.01 * (sum * sum - 3.0));  
}
```

```
void dnk_randomize(void) {  
    char k;  
    srand(time(NULL));  
    xrand = 1.0 * rand() / RAND_MAX;
```

```
    for (k = 0; k < 10; k++)  
        dnk_random();  
}
```