АВС, ИДЗ 1, Самойлов Павел Павлович.

1. Задание:

Разработать программу, которая получает одномерный массив AN, после чего формирует из элементов массива A новый массив B по правилам, указанным в варианте, и выводит его. Память под массивы может выделяться статически, на стеке, автоматичеси по выбору разработчика. При решении задачи необходимо использовать подпрограммы для реализации ввода, вывода и формирования нового массива.

Вариант 22:

Сформировать отсортированный по убыванию массив B из элементов массива A.

Отсчет:

4 балла:

1) Приведено решение задачи на C 2) Написана немодифицированная ассемблерная программа с комментариями. 3) Получена оптимизированная ассемблерная программа с коментариями, отдельно откомпилирована и скомпонована без использования опций отладки 4) Написаны тесты(5 штук), и выполнены тестовые прогоны.

5 баллов:

1) Использованы функции с передачей параметров(часть параметров передается по ссылкам). 2) Использованы локальные переменные. 3) Полученная ассемблерная программа сожержит необходимые комменатрии.

6 баллов:

1) Получено решение на ассемблере с рефакторингом программы за счет максимального использования регистров процессора.

Подробное описание проведенной работы:

1) Исходный код программы на языке C:

```c
#include <stdio.h>

void bubbleSort(int *arr, int n)
{
    for (int i = 0; i < (n - 1); ++i)
    {
        for (int j = (n - 1); j > i; --j)
        {
            if (arr[j - 1] < arr[j])
            {
                int temp = arr[j - 1];
                arr[j - 1] = arr[j];
                arr[j] = temp;
            }
```

```
        }
    }
}

static void printArr(FILE *out, int *arr, int n)
{
    for (int i = 0; i < n; ++i)
    {
        fprintf(out, "%d ", arr[i]);
    }
}

static void fillArr(FILE *in, int *arr, int n)
{
    for (int i = 0; i < n; ++i)
    {
        fscanf(in, "%d", &arr[i]);
    }
}

int main()
{
    int A[100], B[100];
    int n, i;

    FILE *in = fopen("input.txt", "r");
    FILE *out = fopen("output.txt", "w");

    fscanf(in, "%d", &n);

    fillArr(in, A, n);
    for (i = 0; i < n; ++i)
        B[i] = A[i];

    bubbleSort(B, n);
    printArr(out, B, n);
    return 0;
}
```

(P.S. Для 5 баллов: В данном коде сразу используется передача параметров в функции по ссылкам, а также есть локальные переменные.)

Также создадим в папке с программой следующие файлы для корректной работы: "input.txt" "output.txt"

2) Полученная ассемблерная программа без удаления лишних макросов и комментариев: (с помощью команды: gcc -S code.S firstAssemblyCodeVersion.s)

```
    .file    "code.c"
    .intel_syntax noprefix
```

```
    .text
    .globl    bubbleSort
    .type    bubbleSort, @function
bubbleSort:
    endbr64
    push    rbp
    mov    rbp, rsp
    mov    QWORD PTR -24[rbp], rdi
    mov    DWORD PTR -28[rbp], esi
    mov    DWORD PTR -4[rbp], 0
    jmp    .L2
.L6:
    mov    eax, DWORD PTR -28[rbp]
    sub    eax, 1
    mov    DWORD PTR -8[rbp], eax
    jmp    .L3
.L5:
    mov    eax, DWORD PTR -8[rbp]
    cdqe
    sal    rax, 2
    lea    rdx, -4[rax]
    mov    rax, QWORD PTR -24[rbp]
    add    rax, rdx
    mov    edx, DWORD PTR [rax]
    mov    eax, DWORD PTR -8[rbp]
    cdqe
    lea    rcx, 0[0+rax*4]
    mov    rax, QWORD PTR -24[rbp]
    add    rax, rcx
    mov    eax, DWORD PTR [rax]
    cmp    edx, eax
    jge    .L4
    mov    eax, DWORD PTR -8[rbp]
    cdqe
    sal    rax, 2
    lea    rdx, -4[rax]
    mov    rax, QWORD PTR -24[rbp]
    add    rax, rdx
    mov    eax, DWORD PTR [rax]
    mov    DWORD PTR -12[rbp], eax
    mov    eax, DWORD PTR -8[rbp]
    cdqe
    lea    rdx, 0[0+rax*4]
    mov    rax, QWORD PTR -24[rbp]
    add    rax, rdx
    mov    edx, DWORD PTR -8[rbp]
    movsx    rdx, edx
    sal    rdx, 2
    lea    rcx, -4[rdx]
    mov    rdx, QWORD PTR -24[rbp]
    add    rdx, rcx
    mov    eax, DWORD PTR [rax]
```

```
    mov    DWORD PTR [rdx], eax
    mov    eax, DWORD PTR -8[rbp]
    cdqe
    lea    rdx, 0[0+rax*4]
    mov    rax, QWORD PTR -24[rbp]
    add    rdx, rax
    mov    eax, DWORD PTR -12[rbp]
    mov    DWORD PTR [rdx], eax
.L4:
    sub    DWORD PTR -8[rbp], 1
.L3:
    mov    eax, DWORD PTR -8[rbp]
    cmp    eax, DWORD PTR -4[rbp]
    jg    .L5
    add    DWORD PTR -4[rbp], 1
.L2:
    mov    eax, DWORD PTR -28[rbp]
    sub    eax, 1
    cmp    DWORD PTR -4[rbp], eax
    jl    .L6
    nop
    nop
    pop    rbp
    ret
    .size    bubbleSort, .-bubbleSort
    .section    .rodata
.LC0:
    .string    "%d "
    .text
    .type    printArr, @function
printArr:
    endbr64
    push    rbp
    mov    rbp, rsp
    sub    rsp, 48
    mov    QWORD PTR -24[rbp], rdi
    mov    QWORD PTR -32[rbp], rsi
    mov    DWORD PTR -36[rbp], edx
    mov    DWORD PTR -4[rbp], 0
    jmp    .L8
.L9:
    mov    eax, DWORD PTR -4[rbp]
    cdqe
    lea    rdx, 0[0+rax*4]
    mov    rax, QWORD PTR -32[rbp]
    add    rax, rdx
    mov    edx, DWORD PTR [rax]
    mov    rax, QWORD PTR -24[rbp]
    lea    rcx, .LC0[rip]
    mov    rsi, rcx
    mov    rdi, rax
    mov    eax, 0
```

```
    call    fprintf@PLT
    add    DWORD PTR -4[rbp], 1
.L8:
    mov    eax, DWORD PTR -4[rbp]
    cmp    eax, DWORD PTR -36[rbp]
    jl    .L9
    nop
    nop
    leave
    ret
    .size    printArr, .-printArr
    .section    .rodata
.LC1:
    .string    "%d"
    .text
    .type    fillArr, @function
fillArr:
    endbr64
    push    rbp
    mov    rbp, rsp
    sub    rsp, 48
    mov    QWORD PTR -24[rbp], rdi
    mov    QWORD PTR -32[rbp], rsi
    mov    DWORD PTR -36[rbp], edx
    mov    DWORD PTR -4[rbp], 0
    jmp    .L11
.L12:
    mov    eax, DWORD PTR -4[rbp]
    cdqe
    lea    rdx, 0[0+rax*4]
    mov    rax, QWORD PTR -32[rbp]
    add    rdx, rax
    mov    rax, QWORD PTR -24[rbp]
    lea    rcx, .LC1[rip]
    mov    rsi, rcx
    mov    rdi, rax
    mov    eax, 0
    call    __isoc99_fscanf@PLT
    add    DWORD PTR -4[rbp], 1
.L11:
    mov    eax, DWORD PTR -4[rbp]
    cmp    eax, DWORD PTR -36[rbp]
    jl    .L12
    nop
    nop
    leave
    ret
    .size    fillArr, .-fillArr
    .section    .rodata
.LC2:
    .string    "r"
.LC3:
```

```
    .string    "input.txt"
.LC4:
    .string    "w"
.LC5:
    .string    "output.txt"
    .text
    .globl    main
    .type    main, @function
main:
    endbr64
    push    rbp
    mov    rbp, rsp
    sub    rsp, 848
    lea    rax, .LC2[rip]
    mov    rsi, rax
    lea    rax, .LC3[rip]
    mov    rdi, rax
    call    fopen@PLT
    mov    QWORD PTR -16[rbp], rax
    lea    rax, .LC4[rip]
    mov    rsi, rax
    lea    rax, .LC5[rip]
    mov    rdi, rax
    call    fopen@PLT
    mov    QWORD PTR -24[rbp], rax
    lea    rdx, -836[rbp]
    mov    rax, QWORD PTR -16[rbp]
    lea    rcx, .LC1[rip]
    mov    rsi, rcx
    mov    rdi, rax
    mov    eax, 0
    call    __isoc99_fscanf@PLT
    mov    edx, DWORD PTR -836[rbp]
    lea    rcx, -432[rbp]
    mov    rax, QWORD PTR -16[rbp]
    mov    rsi, rcx
    mov    rdi, rax
    call    fillArr
    mov    DWORD PTR -4[rbp], 0
    jmp    .L14
.L15:
    mov    eax, DWORD PTR -4[rbp]
    cdqe
    mov    edx, DWORD PTR -432[rbp+rax*4]
    mov    eax, DWORD PTR -4[rbp]
    cdqe
    mov    DWORD PTR -832[rbp+rax*4], edx
    add    DWORD PTR -4[rbp], 1
.L14:
    mov    eax, DWORD PTR -836[rbp]
    cmp    DWORD PTR -4[rbp], eax
    jl    .L15
```

```
    mov    edx, DWORD PTR -836[rbp]
    lea    rax, -832[rbp]
    mov    esi, edx
    mov    rdi, rax
    call    bubbleSort
    mov    edx, DWORD PTR -836[rbp]
    lea    rcx, -832[rbp]
    mov    rax, QWORD PTR -24[rbp]
    mov    rsi, rcx
    mov    rdi, rax
    call    printArr
    mov    eax, 0
    leave
    ret
    .size    main, .-main
    .ident    "GCC: (Ubuntu 11.2.0-19ubuntu1) 11.2.0"
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long    1f - 0f
    .long    4f - 1f
    .long    5
0:
    .string    "GNU"
1:
    .align 8
    .long    0xc0000002
    .long    3f - 2f
2:
    .long    0x3
3:
    .align 8
4:
```

3) Код успешно компилируется, теперь удалим лишние макросы и добавим комментарии. 3.1 Удалим лишние макросы засчет использования соответствующих аргументов командной строки(Флаги для GCC):

```
gcc -masm=intel \
    -fno-asynchronous-unwind-tables \
    -fno-jump-tables \
    -fno-stack-protector \
    -fno-exceptions \
    ./code.c \
    -S -o ./code.s
```

3.2 Далее из полученной ассемблерной программы засчет ручного редактирования удалим следующий мусор:

```
.size printArr, .-printArr
.section .rodata
```

```
    .file    "code.c"
```

```
    .size    main, .-main
    .ident    "GCC: (Ubuntu 11.2.0-19ubuntu1) 11.2.0"
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long    1f - 0f
    .long    4f - 1f
    .long    5
0:
    .string    "GNU"
1:
    .align 8
    .long    0xc0000002
    .long    3f - 2f
2:
    .long    0x3
3:
    .align 8
4:
```

```
    endbr64
```

3.3 И наконец добавим в ассемблерную программу необходимые комментарии и получим следующий код:

```
    .intel_syntax noprefix                # Используем синтаксис в стиле Intel
    .text                                 # Начало секции.
    .globl    bubbleSort                    # Объявляем и экспортируем bubbleSort.
    .type    bubbleSort, @function         # Отмечаем, что это функция.
bubbleSort:
    push    rbp                           # Стандартный пролог функции.
    mov    rbp, rsp                       # Стандартный пролог функции.

    # Загрузка параметров в стек.
    mov    QWORD PTR -24[rbp], rdi        # arr
    mov    DWORD PTR -28[rbp], esi        # n
    mov    DWORD PTR -4[rbp], 0           # i = 0
    jmp    .L2                            # Переход к метке L2 по коду(к сравнению
цикла).
.L6:
        # Зашли внутрь цикла первого цикла.
    mov    eax, DWORD PTR -28[rbp]
    sub    eax, 1                         # Увеличили i.
    mov    DWORD PTR -8[rbp], eax
    jmp    .L3                            # Переходим к условию второго цикла.
.L5:
    mov    eax, DWORD PTR -8[rbp]         # Загрузка из стека.

    # Загрузка arr[j - 1] и arr[j].
    sal    rax, 2
    lea    rdx, -4[rax]
```

```
    mov    rax, QWORD PTR -24[rbp]
    add    rax, rdx
    mov    edx, DWORD PTR [rax]
    mov    eax, DWORD PTR -8[rbp]
    lea    rcx, 0[0+rax*4]
    mov    rax, QWORD PTR -24[rbp]
    add    rax, rcx
    mov    eax, DWORD PTR [rax]
    #

    cmp    edx, eax                      # Сравнение (arr[j - 1] < arr[j])
    jge    .L4
    # Если условие выполнено, то выполняем инструкции внутри блока if.
    # Работа с памятью и выполнение строки кода (int temp = arr[j - 1];).
    mov    eax, DWORD PTR -8[rbp]
    sal    rax, 2
    lea    rdx, -4[rax]
    mov    rax, QWORD PTR -24[rbp]
    add    rax, rdx
    mov    eax, DWORD PTR [rax]
    mov    DWORD PTR -12[rbp], eax
    #

    # Работа с памятью и выполнение строки кода (arr[j - 1] = arr[j];).
    mov    eax, DWORD PTR -8[rbp]    # Присваиваем rax элемент из A.
    lea    rdx, 0[0+rax*4]           # rdx = rax * 4.
    mov    rax, QWORD PTR -24[rbp]
    add    rax, rdx                  # Приравниваем. (не описываю все действия т.к. по
аналогии)
    mov    edx, DWORD PTR -8[rbp]
    movsx    rdx, edx
    sal    rdx, 2
    lea    rcx, -4[rdx]
    mov    rdx, QWORD PTR -24[rbp]
    add    rdx, rcx
    mov    eax, DWORD PTR [rax]
    mov    DWORD PTR [rdx], eax
    #

    # Работа с памятью и выполнение строки кода ( arr[j] = temp;).
    mov    eax, DWORD PTR -8[rbp]  # Присваиваем eax элемент из A.
    lea    rdx, 0[0+rax*4]         # rdx = rax * 4.
    mov    rax, QWORD PTR -24[rbp]
    add    rdx, rax                # Приравниваем.
    mov    eax, DWORD PTR -12[rbp]
    mov    DWORD PTR [rdx], eax
    #
.L4:
    sub    DWORD PTR -8[rbp], 1          # Изменение значения.
.L3:
        # Проверка условия 2-ого цикла.
    mov    eax, DWORD PTR -8[rbp]        # eax = i
```

```
    cmp    eax, DWORD PTR -4[rbp]        # Если j > i, то переходим к вложенному
циклу.
    jg    .L5
    # Если нарушено условие то выходим из вложенного цикла.
    add    DWORD PTR -4[rbp], 1          # ++i
.L2:
    mov    eax, DWORD PTR -28[rbp]       # Загрузка n из стека в регистр.
    sub    eax, 1
    cmp    DWORD PTR -4[rbp], eax        # Если i < (n - 1), то переходим к циклу.
    jl    .L6
    # Если нарушено условие, то выходим из функции.
    nop
    nop
    pop    rbp
    ret
.LC0:
    .string    "%d "                         # Экспортируем.
    .text                                  # Начало секции.
    .type    printArr, @function           # Отмечаем, что это функция.
printArr:
        # Пролог функции, выделяем 48 байт на стеке.
    push    rbp
    mov    rbp, rsp
    sub    rsp, 48

    # 4 - i
    # 24 - out
    # 32 - arr
    # 36 - n

    # Загрузка параметров в стек.
    mov    QWORD PTR -24[rbp], rdi       # out
    mov    QWORD PTR -32[rbp], rsi       # arr
    mov    DWORD PTR -36[rbp], edx       # n
    mov    DWORD PTR -4[rbp], 0          # i = 0
    jmp    .L8                           # Переход к метке L8 по коду(к условию
цикла).
.L9:
    mov    eax, DWORD PTR -4[rbp]        # eax = i
    lea    rdx, 0[0+rax*4]               # rdx = rax * 4, сдвиг в памяти
    mov    rax, QWORD PTR -32[rbp]       # 3 агрумент arr[i]
    add    rax, rdx
    mov    edx, DWORD PTR [rax]
    mov    rax, QWORD PTR -24[rbp]       # 1 аргумент fprintf, output.txt
    lea    rcx, .LC0[rip]                # 2 аргумент fprtinf, "%d ".
    mov    rsi, rcx
    mov    rdi, rax
    mov    eax, 0                        # Обнуляем eax.
    call    fprintf@PLT                   # Вызываем fprintf.
    add    DWORD PTR -4[rbp], 1          # Увеличение i.
    # Переход к следующей итерации.
.L8:
```

```
    mov    eax, DWORD PTR -4[rbp]        # Загрузка n из стека в регистр.
    cmp    eax, DWORD PTR -36[rbp]       # Сравниваем i и n.
    jl    .L9                           # Если i < n, то переходим к циклу.

    # Если нарушено условие, то выходим из функции.
    nop
    nop
    leave
    ret
.LC1:
    .string    "%d"                          # Экспортируем.
    .text                                  # Начало секции.
    .type    fillArr, @function            # Отмечаем, что это функция.
fillArr:
    # Пролог функции, выделяем 48 байт на стеке.
    push    rbp
    mov    rbp, rsp
    sub    rsp, 48

    # 4 - i
    # 24 - in
    # 32 - arr
    # 36 - n

    mov    QWORD PTR -24[rbp], rdi       # in
    mov    QWORD PTR -32[rbp], rsi       # arr
    mov    DWORD PTR -36[rbp], edx       # n
    mov    DWORD PTR -4[rbp], 0          # i = 0
    jmp    .L11                          # Переход к метке L11 по коду(к условию
цикла).
.L12:
    mov    eax, DWORD PTR -4[rbp]        # eax = i
    lea    rdx, 0[0+rax*4]               # rdx = rax * 4, сдвиг в памяти
    mov    rax, QWORD PTR -32[rbp]       #    3 аргумент arr[i].
    add    rdx, rax
    mov    rax, QWORD PTR -24[rbp]       # 1 аргумент fscanf, "input.txt".
    lea    rcx, .LC1[rip]                # 2 аргумент fscanf, "%d ".
    mov    rsi, rcx
    mov    rdi, rax
    mov    eax, 0                        # Обнуляем eax.
    call    __isoc99_fscanf@PLT           # Вызов fscanf.
    add    DWORD PTR -4[rbp], 1          # увеличение i.
    # Переход к следующей итерации.
.L11:
    mov    eax, DWORD PTR -4[rbp]        # Загрузка n из стека в регистр.
    cmp    eax, DWORD PTR -36[rbp]       # Сравниваем i и n.
    jl    .L12                          # Если i < n, то переходим к циклу.

    # Если нарушено условие, то выходим из функции.
    nop
    nop
    leave
```

```
    ret
.LC2:
    .string    "r"                          # Загружаем "r".
.LC3:
    .string    "input.txt"                  # Загружаем "input.txt".
.LC4:
    .string    "w"                          # Загружаем "w".
.LC5:
    .string    "output.txt"                 # Загружаем "output.txt".
    .text                                  # Начало секции.
    .globl    main                          # Объявляем и экспортируем main.
    .type    main, @function               # Отмечаем, что это функция.
main:
    # Пролог функции, выделяем 848 байт на стеке.
    push    rbp
    mov    rbp, rsp
    sub    rsp, 848

    lea    rax, .LC2[rip]                # Загружаем LC2("r") для fopen.
    mov    rsi, rax
    lea    rax, .LC3[rip]                # Загружаем LC3("input.txt") для fopen.
    mov    rdi, rax
    call    fopen@PLT                     # Вызываем fopen.
    mov    QWORD PTR -16[rbp], rax       # Заполняем in (in = fopen()).

    lea    rax, .LC4[rip]                # Загружаем LC4("w") для fopen.
    mov    rsi, rax
    lea    rax, .LC5[rip]                # Загружаем LC5("output.txt") для fopen.
    mov    rdi, rax
    call    fopen@PLT                     # Вызываем fopen.
    mov    QWORD PTR -24[rbp], rax       # Заполняем out (out = fopen()).

    lea    rdx, -836[rbp]                # Загружаем n для fscanf.
    mov    rax, QWORD PTR -16[rbp]       # Загружаем in для fscanf.
    lea    rcx, .LC1[rip]                # Загружаем LC1("%d") для fscanf.
    mov    rsi, rcx
    mov    rdi, rax
    mov    eax, 0                        # Обнуляем eax.
    call    __isoc99_fscanf@PLT           # Вызываем fscanf.
    mov    edx, DWORD PTR -836[rbp]      # Загружаем массив A для fillArr.
    lea    rcx, -432[rbp]                # Загружаем n для fillArr.
    mov    rax, QWORD PTR -16[rbp]       # Загружаем in для fillArr.
    mov    rsi, rcx
    mov    rdi, rax
    call    fillArr                       # Вызываем filArr.
    mov    DWORD PTR -4[rbp], 0          # i = 0
    jmp    .L14                          # Переход к метке L14 по коду(к условию
цикла).
.L15:
    mov    eax, DWORD PTR -4[rbp]          # eax = i

    # Операции с индексом и элементами массива.
```

```
    mov    edx, DWORD PTR -432[rbp+rax*4]
    mov    eax, DWORD PTR -4[rbp]
    mov    DWORD PTR -832[rbp+rax*4], edx
    add    DWORD PTR -4[rbp], 1            # Увеличение i
.L14:
    mov    eax, DWORD PTR -836[rbp]      # Загрузка массива из стека в регистр.
    cmp    DWORD PTR -4[rbp], eax        # Сравниваем i и n.
    jl    .L15                          # Если  i < n, то переходим к L15(иначе
выходим).
    mov    edx, DWORD PTR -836[rbp]      # Массив A для bubbleSort(1 аргумент).
    lea    rax, -832[rbp]                # n, для bubbleSort. (2 аргумент).
    mov    esi, edx
    mov    rdi, rax
    call    bubbleSort                    # Вызов функции bubbleSort.

    # Далее 3 строчки загружают out, n, B для printArr.
    mov    edx, DWORD PTR -836[rbp]      # B
    lea    rcx, -832[rbp]                # n
    mov    rax, QWORD PTR -24[rbp]       # in
    #
    mov    rsi, rcx
    mov    rdi, rax
    call    printArr                      # Вызов функции printArr.
    mov    eax, 0                        # Обнуляем eax.
    # Выходим.
    leave
    ret
```

Скомпонуем и откомпилируем ассемблерную программу при помощи следующих комманд:

```
gcc ./code.s -o ./code.exe
/.code.exe
```

Все прошло успешно, программа выпоняется без ошибок. 4. Для тестирования создал отдельную папку Tests, и в ней сохранил 5 наборов тестов соотвественно.

Проведенные тесты и результаты вывода:

```
Входные данные:
n = 5
A: 1 2 3 4
Результат:
Одидаемые результат значений массива B(output.txt): 5 4 3 2 1
Полученный результат значений массива B скомпилированной программы на C: 5 4 3 2 1
Полученный результат значений массива B скомпилированной ассемблер программы: 5 4 3 2
1
```

```
Входные данные:
n = 6
A: 9 8 7 11 5 2
Результат:
Одидаемые результат значений массива B(output.txt): 11 9 8 7 5 2
```

```
Полученный результат значений массива B скомпилированной программы на C: 11 9 8 7 5 2
Полученный результат значений массива B скомпилированной ассемблер программы: 11 9 8 7 5 2
```

```
Входные данные:
n = 4
A: 2 2 9 9
Результат:
Одидаемые результат значений массива B(output.txt): 9 9 2 2
Полученный результат значений массива B скомпилированной программы на C: 9 9 2 2
Полученный результат значений массива B скомпилированной ассемблер программы: 9 9 2 2
```

```
Входные данные:
n = 5
A: 1 9 8 2 3
Результат:
Одидаемые результат значений массива B(output.txt): 9 8 3 2 1
Полученный результат значений массива B скомпилированной программы на C: 9 8 3 2 1
Полученный результат значений массива B скомпилированной ассемблер программы: 9 8 3 2 1
```

```
Входные данные:
n = 4
A: 2 3 3 2
Результат:
Одидаемые результат значений массива B(output.txt): 3 3 2 2
Полученный результат значений массива B скомпилированной программы на C: 3 3 2 2
Полученный результат значений массива B скомпилированной ассемблер программы: 3 3 2 2
```

Итог тестирования: Представлено полное тестовое покрытие, получен одинаковый результат на обоих программах, из написанного выше видна эквивалентность функционирования.

5. Изменения для пункта 5 не были проведены, посколько код уже соответствует всем критериям.

6. Оптимизация:

6. 1 Удалены следующие строки: cdqe
7. 2 Заменено:

```
DWORD PTR -8[rbp] -> r12 (Были заменены не все, а часть, посколько замена всех не возвожна(происхоидт некомпил))
```