АВС, ИДЗ 2, Самойлов Павел Павлович.

1. Задание:

ASCII-строка — строка, содержащая символы таблицы кодировки ASCII. (https://ru.wikipedia.org/wiki/ASCII). Размер строки может быть достаточно большим, чтобы вмещать многостраничные тексты, например, главы из книг, если задача связана с использованием файлов или строк, порождаемых генератором случайных чисел. Тексты при этом могут не нести смыслового содержания. Для обработки в программе предлагается использовать данные, содержащие символы только из первой половины таблицы (коды в диапазоне 0–12710), что связано с использованием кодировки UTF-8 в ОС Linux в качестве основной. Символы, содержащие коды выше 12710, должны отсутствовать во входных данных кроме оговоренных специально случаев.

Вариант 7:

Разработать программу, заменяющую все гласные буквы в заданной ASCII-строке их ASCII кодами в шестнадцатиричной системе счисления. Код каждого символа задавать в формате «0xDD», где D — шестнадцатиричная цифра от 0 до F.

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.

7 баллов:

1) Реализация программы на ассемблере в виде двух или более единиц компиляции (программу на языке C разделять допускается, но не обязательно) 2) Использование текстовых файлов для ввода/вывода данных.

Подробное описание проведенной работы:

1) Исходный код программы на языке C:

```c
#include <stdio.h>

// Получение 16 ричного кода символа.
char hex_digit(int code)
{
    if (code < 10)
        code = '0' + code;
    else
        code = 'a' + code - 10;

    return code;
}

// Замена гласных на ASCII код (0xDD).
void changeVowelesToASCII(char *str, char *strASCII16, int n, int *newSize)
{
    int cnt = 0;
    for (int i = 0; i < n; ++i)
    {
        if (str[i] != 'a' && str[i] != 'e' && str[i] != 'i' && str[i] != 'o' && str[i]
!= 'u' && str[i] != 'y'
         && str[i] != 'A' && str[i] != 'E' && str[i] != 'I' && str[i] != 'O' && str[i]
!= 'U' && str[i] != 'Y')
         {
            strASCII16[cnt] = str[i];
            ++cnt;

            continue;
        }

        strASCII16[cnt] = '0';
        strASCII16[cnt + 1] = 'x';
        strASCII16[cnt + 2] = hex_digit(str[i] / 16);
        strASCII16[cnt + 3] = hex_digit(str[i] % 16);

        cnt += 4;
    }

    *newSize = cnt;
}

// Вывод массива в текстовый файл.
static void printArr(FILE *out, char *str, int n)
{
    for (int i = 0; i < n; ++i)
    {
        fprintf(out, "%c", str[i]);
    }
}

// Ввод массива с текстового файла.
static void fillArr(FILE *in, char *str, int *n)
```

```
{
    char curChar;
    int i = 0;
    for (; i < 1000;)
    {
        if (fscanf(in, "%c", &curChar) == -1 || curChar == '\0')
            break;

        if (curChar == '\n')
            continue;

        str[i] = curChar;
        ++i;
    }

    *n = i;
}

int main()
{
    // Создание исходного массива char, ограничение в 10000 символов.
    char str[10000];

    // Создание ASCII массива char, ограничение в 40000 символов(т.к. максимум : 10000
* 4, т.к. символ имеет кодировку 0xDD).
    char strASCII16[40000];
    int n = 0;

    // Файл с входными данными.
    FILE *in = fopen("input.txt", "r");

    // Файл с выходными данными.
    FILE *out = fopen("output.txt", "w");

    // Ввод исходного масива символов.
    fillArr(in, str, &n);

    int newSize = 0;
    changeVowelesToASCII(str, strASCII16, n, &newSize);

    // Вывод полученного результата.
    printArr(out, strASCII16, newSize);

    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 code.c -S -o ./firstAssemblyCodeVersion.s)

```
    .file    "code.c"
    .text
    .globl    hex_digit
    .type    hex_digit, @function
hex_digit:
.LFB0:
    .cfi_startproc
    endbr64
    pushq    %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    movl    %edi, -4(%rbp)
    cmpl    $9, -4(%rbp)
    jg    .L2
    addl    $48, -4(%rbp)
    jmp    .L3
.L2:
    addl    $87, -4(%rbp)
.L3:
    movl    -4(%rbp), %eax
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size    hex_digit, .-hex_digit
    .globl    changeVowelesToASCII
    .type    changeVowelesToASCII, @function
changeVowelesToASCII:
.LFB1:
    .cfi_startproc
    endbr64
    pushq    %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    pushq    %rbx
    subq    $48, %rsp
    .cfi_offset 3, -24
    movq    %rdi, -32(%rbp)
    movq    %rsi, -40(%rbp)
    movl    %edx, -44(%rbp)
    movq    %rcx, -56(%rbp)
    movl    $0, -16(%rbp)
```

```
    movl    $0, -12(%rbp)
    jmp    .L6
.L9:
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    cmpb    $97, %al
    je    .L7
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    cmpb    $101, %al
    je    .L7
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    cmpb    $105, %al
    je    .L7
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    cmpb    $111, %al
    je    .L7
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    cmpb    $117, %al
    je    .L7
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    cmpb    $121, %al
    je    .L7
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    cmpb    $65, %al
    je    .L7
```

```
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    cmpb    $69, %al
    je    .L7
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    cmpb    $73, %al
    je    .L7
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    cmpb    $79, %al
    je    .L7
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    cmpb    $85, %al
    je    .L7
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    cmpb    $89, %al
    je    .L7
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movl    -16(%rbp), %edx
    movslq    %edx, %rcx
    movq    -40(%rbp), %rdx
    addq    %rcx, %rdx
    movzbl    (%rax), %eax
    movb    %al, (%rdx)
    addl    $1, -16(%rbp)
    jmp    .L8
.L7:
    movl    -16(%rbp), %eax
    movslq    %eax, %rdx
    movq    -40(%rbp), %rax
    addq    %rdx, %rax
```

```
    movb    $48, (%rax)
    movl    -16(%rbp), %eax
    cltq
    leaq    1(%rax), %rdx
    movq    -40(%rbp), %rax
    addq    %rdx, %rax
    movb    $120, (%rax)
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    leal    15(%rax), %edx
    testb    %al, %al
    cmovs    %edx, %eax
    sarb    $4, %al
    movsbl    %al, %eax
    movl    -16(%rbp), %edx
    movslq    %edx, %rdx
    leaq    2(%rdx), %rcx
    movq    -40(%rbp), %rdx
    leaq    (%rcx,%rdx), %rbx
    movl    %eax, %edi
    call    hex_digit
    movb    %al, (%rbx)
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    movl    %eax, %edx
    sarb    $7, %dl
    shrb    $4, %dl
    addl    %edx, %eax
    andl    $15, %eax
    subl    %edx, %eax
    movsbl    %al, %eax
    movl    -16(%rbp), %edx
    movslq    %edx, %rdx
    leaq    3(%rdx), %rcx
    movq    -40(%rbp), %rdx
    leaq    (%rcx,%rdx), %rbx
    movl    %eax, %edi
    call    hex_digit
    movb    %al, (%rbx)
    addl    $4, -16(%rbp)
.L8:
    addl    $1, -12(%rbp)
.L6:
    movl    -12(%rbp), %eax
    cmpl    -44(%rbp), %eax
    jl    .L9
```

```
    movq    -56(%rbp), %rax
    movl    -16(%rbp), %edx
    movl    %edx, (%rax)
    nop
    movq    -8(%rbp), %rbx
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE1:
    .size    changeVowelesToASCII, .-changeVowelesToASCII
    .type    printArr, @function
printArr:
.LFB2:
    .cfi_startproc
    endbr64
    pushq    %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $48, %rsp
    movq    %rdi, -24(%rbp)
    movq    %rsi, -32(%rbp)
    movl    %edx, -36(%rbp)
    movl    $0, -4(%rbp)
    jmp    .L11
.L12:
    movl    -4(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rdx, %rax
    movzbl    (%rax), %eax
    movsbl    %al, %eax
    movq    -24(%rbp), %rdx
    movq    %rdx, %rsi
    movl    %eax, %edi
    call    fputc@PLT
    addl    $1, -4(%rbp)
.L11:
    movl    -4(%rbp), %eax
    cmpl    -36(%rbp), %eax
    jl    .L12
    nop
    nop
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE2:
    .size    printArr, .-printArr
    .section    .rodata
```

```
.LC0:
    .string    "%c"
    .text
    .type    fillArr, @function
fillArr:
.LFB3:
    .cfi_startproc
    endbr64
    pushq    %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $48, %rsp
    movq    %rdi, -24(%rbp)
    movq    %rsi, -32(%rbp)
    movq    %rdx, -40(%rbp)
    movq    %fs:40, %rax
    movq    %rax, -8(%rbp)
    xorl    %eax, %eax
    movl    $0, -12(%rbp)
    jmp    .L14
.L17:
    leaq    -13(%rbp), %rdx
    movq    -24(%rbp), %rax
    leaq    .LC0(%rip), %rcx
    movq    %rcx, %rsi
    movq    %rax, %rdi
    movl    $0, %eax
    call    __isoc99_fscanf@PLT
    cmpl    $-1, %eax
    je    .L15
    movzbl    -13(%rbp), %eax
    testb    %al, %al
    je    .L15
    movzbl    -13(%rbp), %eax
    cmpb    $10, %al
    jne    .L16
    jmp    .L14
.L16:
    movl    -12(%rbp), %eax
    movslq    %eax, %rdx
    movq    -32(%rbp), %rax
    addq    %rax, %rdx
    movzbl    -13(%rbp), %eax
    movb    %al, (%rdx)
    addl    $1, -12(%rbp)
.L14:
    cmpl    $999, -12(%rbp)
    jle    .L17
.L15:
    movq    -40(%rbp), %rax
```

```
    movl    -12(%rbp), %edx
    movl    %edx, (%rax)
    nop
    movq    -8(%rbp), %rax
    subq    %fs:40, %rax
    je    .L18
    call    __stack_chk_fail@PLT
.L18:
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE3:
    .size    fillArr, .-fillArr
    .section    .rodata
.LC1:
    .string    "r"
.LC2:
    .string    "input.txt"
.LC3:
    .string    "w"
.LC4:
    .string    "output.txt"
    .text
    .globl    main
    .type    main, @function
main:
.LFB4:
    .cfi_startproc
    endbr64
    pushq    %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    leaq    -49152(%rsp), %r11
.LPSRL0:
    subq    $4096, %rsp
    orq    $0, (%rsp)
    cmpq    %r11, %rsp
    jne    .LPSRL0
    subq    $896, %rsp
    movq    %fs:40, %rax
    movq    %rax, -8(%rbp)
    xorl    %eax, %eax
    movl    $0, -50040(%rbp)
    leaq    .LC1(%rip), %rax
    movq    %rax, %rsi
    leaq    .LC2(%rip), %rax
    movq    %rax, %rdi
    call    fopen@PLT
    movq    %rax, -50032(%rbp)
```

```
    leaq    .LC3(%rip), %rax
    movq    %rax, %rsi
    leaq    .LC4(%rip), %rax
    movq    %rax, %rdi
    call    fopen@PLT
    movq    %rax, -50024(%rbp)
    leaq    -50040(%rbp), %rdx
    leaq    -50016(%rbp), %rcx
    movq    -50032(%rbp), %rax
    movq    %rcx, %rsi
    movq    %rax, %rdi
    call    fillArr
    movl    $0, -50036(%rbp)
    movl    -50040(%rbp), %edx
    leaq    -50036(%rbp), %rcx
    leaq    -40016(%rbp), %rsi
    leaq    -50016(%rbp), %rax
    movq    %rax, %rdi
    call    changeVowelesToASCII
    movl    -50036(%rbp), %edx
    leaq    -40016(%rbp), %rcx
    movq    -50024(%rbp), %rax
    movq    %rcx, %rsi
    movq    %rax, %rdi
    call    printArr
    movl    $0, %eax
    movq    -8(%rbp), %rdx
    subq    %fs:40, %rdx
    je    .L21
    call    __stack_chk_fail@PLT
.L21:
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE4:
    .size    main, .-main
    .ident    "GCC: (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0"
    .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:

```
.section .rodata
```

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

```
    .size    main, .-main
    .ident    "GCC: (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0"
    .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:
```

```
    .size    printArr, .-printArr
    endbr64
```

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

```
    .intel_syntax noprefix                                          # Используем
синтаксис в стиле Intel.
    .text                                                           # Начало секции.
```

```
    .globl    hex_digit                                          # Объявляем и
экспортируем hex_digit.
    .type    hex_digit, @function                                # Отмечаем, что это
функция.
hex_digit:
    push    rbp                                 # Стандартный пролог
функции(заталкивает rbp на стек).
    mov    rbp, rsp                                      # Стандартный пролог
функции(копирование переданного значения rsp в rbp).

    # Загрузка параметров в стек.
    mov    DWORD PTR -4[rbp], edi                                # (int code)

    # Конструкция if else.
    cmp    DWORD PTR -4[rbp], 9                                  # Сравнение (code <
10).
    jg    .L2                                                   # Переход к метке .L2
(команда: ('a' + code - 10;)).
    add    DWORD PTR -4[rbp], 48                                 # Выполнение сложения
('0' + code), '0' в Dec ASCII равен 48.
    jmp    .L3                                                   # Переход к метке .L3
(там происходит присваивание посчитанного значения в code, возврат функции).
.L2:
    add    DWORD PTR -4[rbp], 87                                 # Выполнение сложения
('a' + code - 10), ('a' - 10) в Dec ASCII равно 87.;
.L3:
    mov    eax, DWORD PTR -4[rbp]                                # Присвоение
посчитанного значения переменной code. (mod копирует данные из операнда-источника в
операнд-получатель).
    pop    rbp                                                   # Выгружает (int
code) из стека(т.к. при выходе из функции происходит удаление локально созданной
переменной).
    ret                                                           # Возврат значения.

    .size    hex_digit, .-hex_digit                               # Загружаем
hex_digit.
    .globl    changeVowelesToASCII                                # Объявляем и
экспортируем changeVowelesToASCII.
    .type    changeVowelesToASCII, @function                      # Отмечаем, что это
функция.
changeVowelesToASCII:
    push    rbp                                 # Стандартный пролог функции
(заталкивает rbp на стек).
    mov    rbp, rsp                          # Стандартный пролог функции (копирование
переданного значения rsp в rbp).
    push    rbx                                                   # Стандартный пролог
функции (заталкивает rbx на стек).
    sub    rsp, 48                                               # Конец пролога
функции(выделяем 48 байт на стеке).

    # 12 - cnt
    # 16 - i
```

```
    # 32 - str
    # 40 - strASCII16
    # 44 - n
    # 56 - newSize

    # Загрузка параметров в стек.
    mov    QWORD PTR -32[rbp], rdi                              # str
    mov    QWORD PTR -40[rbp], rsi                              # strASCII16
    mov    DWORD PTR -44[rbp], edx                              # n
    mov    QWORD PTR -56[rbp], rcx                              # newSize
    mov    DWORD PTR -12[rbp], 0                                # cnt = 0
    mov    DWORD PTR -16[rbp], 0                                # i = 0

    jmp    .L6                                                  # Переход к метке L6
(условию цикла).
.L9:
    mov    eax, DWORD PTR -16[rbp]                              # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                             # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                              # rax := *str.
    add    rax, rdx                                             # Выполнение сложения
(*str + i), получаем str[i].
    movzx    eax, BYTE PTR [rax]                                  # str[i].
    cmp    al, 97                                               # Сравнение str[i] и
'a', код 'a' в Dec ASCII равен 97.
    je    .L7                                                  # Если не выполнено
(str[i] != 'a'), то переходим к метке L7(не попадаем внутрь if).

    mov    eax, DWORD PTR -16[rbp]                              # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                             # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                              # rax := *str.
    add    rax, rdx                                             # Выполнение сложения
(*str + i), получаем str[i].
    movzx    eax, BYTE PTR [rax]                                  # str[i].
    cmp    al, 101                                              # Сравнение str[i] и
'e', код 'e' в Dec ASCII равен 101.
    je    .L7                                                  # Если не выполнено
(str[i] != 'e'), то переходим к метке L7(не попадаем внутрь if).

    mov    eax, DWORD PTR -16[rbp]                              # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                             # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                              # rax := *str.
    add    rax, rdx                                             # Выполнение сложения
(*str + i), получаем str[i].
    movzx    eax, BYTE PTR [rax]                                  # str[i].
    cmp    al, 105                                              # Сравнение str[i] и
'i', код 'i' в Dec ASCII равен 105.
    je    .L7                                                  # Если не выполнено
(str[i] != 'i'), то переходим к метке L7(не попадаем внутрь if).
```

```
    mov    eax, DWORD PTR -16[rbp]                                  # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                                 # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                                  # rax := *str.
    add    rax, rdx                                                 # Выполнение сложения
(*str + i), получаем str[i].
    movzx    eax, BYTE PTR [rax]                                    # str[i].
    cmp    al, 111                                                  # Сравнение str[i] и
'o', код 'o' в Dec ASCII равен 111.
    je    .L7                                                      # Если не выполнено
(str[i] != 'o'), то переходим к метке L7(не попадаем внутрь if).

    mov    eax, DWORD PTR -16[rbp]                                  # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                               # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                                  # rax := *str.
    add    rax, rdx                                                 # Выполнение сложения
(*str + i), получаем str[i].
    movzx    eax, BYTE PTR [rax]                                    # str[i].
    cmp    al, 117                                                  # Сравнение str[i] и
'u', код 'u' в Dec ASCII равен 117.
    je    .L7                                                      # Если не выполнено
(str[i] != 'u'), то переходим к метке L7(не попадаем внутрь if).

    mov    eax, DWORD PTR -16[rbp]                                  # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                               # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                                  # rax := *str.
    add    rax, rdx                                                 # Выполнение сложения
(*str + i), получаем str[i].
    movzx    eax, BYTE PTR [rax]                                    # str[i].
    cmp    al, 121                                                  # Сравнение str[i] и
'y', код 'y' в Dec ASCII равен 121.
    je    .L7                                                      # Если не выполнено
(str[i] != 'y'), то переходим к метке L7(не попадаем внутрь if).

    mov    eax, DWORD PTR -16[rbp]                                  # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                               # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                                  # rax := *str.
    add    rax, rdx                                                 # Выполнение сложения
(*str + i), получаем str[i].
    movzx    eax, BYTE PTR [rax]                                    # str[i].
    cmp    al, 65                                                   # Сравнение str[i] и
'A', код 'A' в Dec ASCII равен 65.
    je    .L7                                                      # Если не выполнено
(str[i] != 'A'), то переходим к метке L7(не попадаем внутрь if).

    mov    eax, DWORD PTR -16[rbp]                                  # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                               # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                                  # rax := *str.
```

```
    add    rax, rdx                                          # Выполнение сложения
(*str + i), получаем str[i].
    movzx    eax, BYTE PTR [rax]                               # str[i].
    cmp    al, 69                                            # Сравнение str[i] и
'E', код 'E' в Dec ASCII равен 69.
    je    .L7                                               # Если не выполнено
(str[i] != 'E'), то переходим к метке L7(не попадаем внутрь if).

    mov    eax, DWORD PTR -16[rbp]                           # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                        # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                           # rax := *str.
    add    rax, rdx                                          # Выполнение сложения
(*str + i), получаем str[i].
    movzx    eax, BYTE PTR [rax]                             # str[i].
    cmp    al, 73                                            # Сравнение str[i] и
'I', код 'I' в Dec ASCII равен 73.
    je    .L7                                               # Если не выполнено
(str[i] != 'I'), то переходим к метке L7(не попадаем внутрь if).

    mov    eax, DWORD PTR -16[rbp]                           # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                        # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                           # rax := *str.
    add    rax, rdx                                          # Выполнение сложения
(*str + i), получаем str[i].
    movzx    eax, BYTE PTR [rax]                             # str[i].
    cmp    al, 79                                            # Сравнение str[i] и
'O', код 'O' в Dec ASCII равен 79.
    je    .L7                                               # Если не выполнено
(str[i] != 'O'), то переходим к метке L7(не попадаем внутрь if).

    mov    eax, DWORD PTR -16[rbp]                           # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                        # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                           # rax := *str.
    add    rax, rdx                                          # Выполнение сложения
(*str + i), получаем str[i].
    movzx    eax, BYTE PTR [rax]                             # str[i].
    cmp    al, 85                                            # Сравнение str[i] и
'U', код 'U' в Dec ASCII равен 85.
    je    .L7                                               # Если не выполнено
(str[i] != 'U'), то переходим к метке L7(не попадаем внутрь if).

    mov    eax, DWORD PTR -16[rbp]                           # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                        # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                           # rax := *str.
    add    rax, rdx                                          # Выполнение сложения
(*str + i), получаем str[i].
    movzx    eax, BYTE PTR [rax]                             # str[i].
    cmp    al, 89                                            # Сравнение str[i] и
```

```
'Y', код 'Y' в Dec ASCII равен 89.
    je    .L7                                                 # Если не выполнено
(str[i] != 'Y'), то переходим к метке L7(не попадаем внутрь if).

    mov    eax, DWORD PTR -16[rbp]                            # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                           # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                            # rax := *str.
    add    rax, rdx                                           # Выполнение сложения
(*str + i), получаем str[i].
    mov    edx, DWORD PTR -12[rbp]                            # str[i].

    movsx    rcx, edx                                           # rcx := edx.

    mov    rdx, QWORD PTR -40[rbp]                            # Выполнение сложения
(*strASCII + i), получаем str[i].
    add    rdx, rcx                                           # strASCII[i].

    # Выполнение (strASCII16[cnt] = str[i]).
    movzx    eax, BYTE PTR [rax]
    mov    BYTE PTR [rdx], al

    add    DWORD PTR -12[rbp], 1                              # Увеличение cnt на 1
(++cnt).
    jmp    .L8                                                # Переход к метке L8
(continue).
.L7:
    # Выполнение строчки кода (strASCII[cnt] = '0');.
    mov    eax, DWORD PTR -12[rbp]                            # Перемещаем rbp - 12
в eax (i).
    movsx    rdx, eax                                           # rdx := eax (i).
    mov    rax, QWORD PTR -40[rbp]                            # rax := *strASCII16.
    add    rax, rdx                                           # Выполнение сложения
(*strASCII + i), получаем strASCII[i].
    mov    BYTE PTR [rax], 48                                 # Выполняем
(strASCII[cnt] = '0').

    # Выполнение строчки кода (strASCII[cnt + 1] = 'x').
    mov    eax, DWORD PTR -12[rbp]                            # Перемещаем rbp - 12
в eax (i).
    lea    rdx, 1[rax]                                        # задаем (i).
    mov    rax, QWORD PTR -40[rbp]                            # rax := *strASCII.
    add    rax, rdx                                           # Выполнение сложения
(*strASCII + i), получаем strASCII[i].
    mov    BYTE PTR [rax], 120                                # Выполняем
(strASCII[cnt] = 'x').

    # Выполнение строчки кода (strASCII[cnt + 2] = hex_digit(str[i] / 16)).
    mov    eax, DWORD PTR -16[rbp]                            # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                           # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]
```

```
    add    rax, rdx
    movzx    eax, BYTE PTR [rax]                                  # Складываем,
получая strASCII[cnt + 3].
    lea    edx, 15[rax]                                          # Для последующего
деления на 16.
    test    al, al
    cmovs    eax, edx                                           # Складываем,
получая str[i].
    sar    al, 4
    movsx    eax, al
    mov    edx, DWORD PTR -12[rbp]
    movsx    rdx, edx
    lea    rcx, 2[rdx]
    mov    rdx, QWORD PTR -40[rbp]
    lea    rbx, [rcx+rdx]
    mov    edi, eax                                              # edi := eax.
    call    hex_digit                                             # Вызов hex_digit.
    mov    BYTE PTR [rbx], al                                    # Выполняем
(strASCII[cnt + 2] = hex_digit(str[i] / 16)).

    # Выполнение строчки кода (strASCII[cnt + 3] = hex_digit(str[i] % 16)).
    mov    eax, DWORD PTR -16[rbp]                               # Перемещаем rbp - 16
в eax (i).
    movsx    rdx, eax                                            # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]
    add    rax, rdx                                              # Складываем, получая
strASCII[cnt + 3].
    movzx    eax, BYTE PTR [rax]
    mov    edx, eax
    sar    dl, 7
    shr    dl, 4
    add    eax, edx                                              # Складываем, получая
str[i].
    and    eax, 15                                               # str[i] % 16.
    sub    eax, edx
    movsx    eax, al
    mov    edx, DWORD PTR -12[rbp]
    movsx    rdx, edx
    lea    rcx, 3[rdx]
    mov    rdx, QWORD PTR -40[rbp]
    lea    rbx, [rcx+rdx]
    mov    edi, eax                                              # edi := eax.
    call    hex_digit                                             # Вызов hex_digit.
    mov    BYTE PTR [rbx], al                                    # Выполняем
(strASCII[cnt + 2] = hex_digit(str[i] / 16)).

    add    DWORD PTR -12[rbp], 4                                 # Увеличение cnt на 4
(cnt += 4).
.L8:
    add    DWORD PTR -16[rbp], 1                                 # Увеличение i (++i).
.L6:
    mov    eax, DWORD PTR -16[rbp]                               # Загрузка n из стека
```

```
в регистр.
    cmp    eax, DWORD PTR -44[rbp]                                  # Сравнение i и n.
    jl    .L9                                                      # Если выполнено
условие (i < n), то переходим в метке L9(циклу).

    mov    rax, QWORD PTR -56[rbp]                                  # rax = rbp - 56.
    mov    edx, DWORD PTR -12[rbp]                                  # edx = rbp - 12.
    mov    DWORD PTR [rax], edx                                     # (*newSize = cnt).

    # Выход из функции.
    nop
    mov    rbx, QWORD PTR -8[rbp]                                   # rbx = rbp - 8.
    leave
    ret

    .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 - str
    # 36 - n

    # Загрузка параметров в стек.
    mov    QWORD PTR -24[rbp], rdi                                  # out
    mov    QWORD PTR -32[rbp], rsi                                  # str
    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.
    movsx    rdx, eax                                                 # rdx := eax. (i)
    mov    rax, QWORD PTR -32[rbp]                                  # 3 аргумент str[i]
(для fprintf).
    add    rax, rdx                                                 # (*str + i), то есть
для получения str[i].
    movzx    eax, BYTE PTR [rax]                                      # str[i]
    movsx    eax, al                                                  # копирует все
(str[i]) в eax.
    mov    rdx, QWORD PTR -24[rbp]                                  # 1 аргумент out
"output.txt" (для fprintf).
    mov    rsi, rdx                                                 # rsi := rdx.
    mov    edi, eax                                                 # edi := eax.
    call    fputc@PLT                                                # Вызов функции
fprintf.
    add    DWORD PTR -4[rbp], 1                                     # Увеличение i (++i).
```

```
.L11:
    mov    eax, DWORD PTR -4[rbp]                                  # Загрузка n из стека
в регистр.
    cmp    eax, DWORD PTR -36[rbp]                                 # Сравниваем i и n (i
< n).
    jl    .L12                                                    # Если i < n, то
переходим к циклу.

    # Если нарушено условие, то выходим из функции.
    nop
    nop
    leave
    ret
.LC0:
    .string    "%c"                                                  # Экспортируем
"%c".
    .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 - str
    # 40 - n

    # Загрузка параметров в стек.
    mov    QWORD PTR -24[rbp], rdi                                 # in
    mov    QWORD PTR -32[rbp], rsi                                 # str
    mov    QWORD PTR -40[rbp], rdx                                 # n
    mov    DWORD PTR -4[rbp], 0                                    # i
    jmp    .L14                                                    # Переход к условию
цикла.
.L17:
    lea    rdx, -5[rbp]                                            # Перемещение rbp -
5[rbp] в rdx.
    mov    rax, QWORD PTR -24[rbp]                                 # 1 аргумент in
"input.txt" (для fscanf).
    lea    rcx, .LC0[rip]                                          # 2 аргумент "%c" в
rcx (для fscanf).
    mov    rsi, rcx                                                # rsi := rcs.
    mov    rdi, rax                                                # rdi := rax.
    mov    eax, 0                                                  # eax := 0.
    call    __isoc99_fscanf@PLT                                    # Вызов функции
fscanf.
    cmp    eax, -1                                                 # Сравнение
(fscanf(in, "%c", &curChar) и -1).
    je    .L15                                                    # Если (fscanf(in,
```

```
"%c", &curChar) == -1), то переход к L15 (break).
    movzx    eax, BYTE PTR -5[rbp]                                  # str[i].
    test    al, al                                                 # Сравниваем
(curChar и '\0').
    je    .L15                                                    # Если (curChar ==
'\0'), то переход к L15 (break).
    movzx    eax, BYTE PTR -5[rbp]                                  # str[i].
    cmp    al, 10                                                  # Сравниваем (curChar
и '\n').
    jne    .L16                                                    # Если (curChar !=
'\n'), то переход к L16.
    jmp    .L14
.L16:
    mov    eax, DWORD PTR -4[rbp]                                  # Загрузка eax из
стека в регистр.
    movsx    rdx, eax                                              # rdx := eax (i).
    mov    rax, QWORD PTR -32[rbp]                                 # rax := str[i].
    add    rdx, rax                                                # rdx := rax.
    movzx    eax, BYTE PTR -5[rbp]                                 # str[i].
    mov    BYTE PTR [rdx], al                                      # Сдвиг.
    add    DWORD PTR -4[rbp], 1                                    # ++i.
.L14:
    cmp    DWORD PTR -4[rbp], 999                                  # Сравнение i и 1000,
(i < 1000).
    jle    .L17                                                    # Если i < 1000, то
переходим к циклу.
.L15:
    mov    rax, QWORD PTR -40[rbp]                                 # rax := str[i].
    mov    edx, DWORD PTR -4[rbp]                                  # Загрузка edx из
стека в регистр.
    mov    DWORD PTR [rax], edx                                    # Сдвиг.

    # Выходим из функции.
    nop
    leave
    ret
.LC1:
    .string    "r"                                                   # Загружаем "r".
.LC2:
    .string    "input.txt"                                           # Загружаем
"input.txt".
.LC3:
    .string    "w"                                                   # Загружаем "w".
.LC4:
    .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 память на стеке.
    push    rbp
    mov    rbp, rsp
    lea    r11, -49152[rsp]
.LPSRL0:
    # Процессы по выделению памяти.
    sub    rsp, 4096
    or    DWORD PTR [rsp], 0
    cmp    rsp, r11
    jne    .LPSRL0
    sub    rsp, 880
    mov    DWORD PTR -50020[rbp], 0                                # (int n = 0;).

    lea    rax, .LC1[rip]                                          # Загружаем LC1("r")
для fopen.
    mov    rsi, rax                                                # rsi := rax.
    lea    rax, .LC2[rip]                                          # Загружаем
LC2("input.txt") для fopen.
    mov    rdi, rax                                                # rdi := rax.
    call    fopen@PLT                                               # Вызов fopen.
    mov    QWORD PTR -8[rbp], rax                                  # Заполняем in (in =
fopen()).

    lea    rax, .LC3[rip]                                          # Загружаем LC3("w")
для fopen.
    mov    rsi, rax                                                # rsi := rax.
    lea    rax, .LC4[rip]                                          # Загружаем
LC4("output.txt") для fopen.
    mov    rdi, rax                                                # rdi := rax.
    call    fopen@PLT                                               # Вызов fopen.
    mov    QWORD PTR -16[rbp], rax                                 # Заполняем out (out
= fopen()).

    lea    rdx, -50020[rbp]                                        # Выгружаем
strASCII16 в rdx.
    lea    rcx, -10016[rbp]                                        # Выгружаем str в
rcx.
    mov    rax, QWORD PTR -8[rbp]                                  # Выгружаем n в rax.
    mov    rsi, rcx                                                # rsi := rcx.
    mov    rdi, rax                                                # rdi := rax.
    call    fillArr                                                 # Вызов fillArr.

    mov    DWORD PTR -50024[rbp], 0                                # (int newSize = 0;).
    mov    edx, DWORD PTR -50020[rbp]

    # Выгружаем: (str, strASCII16, n) для функции changeVowelesToASCII.
    lea    rcx, -50024[rbp]
    lea    rsi, -50016[rbp]
    lea    rax, -10016[rbp]
    mov    rdi, rax                                                # rdi := rax.
    call    changeVowelesToASCII                                    # Вызов
changeVowelesToASCII.
```

```
    mov    edx, DWORD PTR -50024[rbp]                              # newSize
    lea    rcx, -50016[rbp]                                        # Выгружаем
strASCII16.
    mov    rax, QWORD PTR -16[rbp]                                 # out.
    mov    rsi, rcx                                                # rsi := rcx.
    mov    rdi, rax                                                # rdi := rax;
    call    printArr                                                # Вызов printArr.

    # Завершение программы.
    mov    eax, 0                                                  # return 0.
    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:
wqd123
Результат:
Одидаемые результат (output.txt): wqd123
Полученный результат скомпилированной программы на C: wqd123
Полученный результат скомпилированной ассемблер программы: wqd123
```

```
Входные данные:
wakA666
Результат:
Одидаемые результат (output.txt): w0x61k0x41666
Полученный результат скомпилированной программы на C: w0x61k0x41666
Полученный результат скомпилированной ассемблер программы: w0x61k0x41666
```

```
Входные данные:
123o
Результат:
Одидаемые результат (output.txt): 1230x6f
Полученный результат скомпилированной программы на C: 1230x6f
Полученный результат скомпилированной ассемблер программы: 1230x6f
```

```
Входные данные:
aoeuy6q
Результат:
Одидаемые результат (output.txt): 0x610x6f0x650x750x796q
Полученный результат скомпилированной программы на C: 0x610x6f0x650x750x796q
Полученный результат скомпилированной ассемблер программы: 0x610x6f0x650x750x796q
```

```
Входные данные:
kuruzawa
Результат:
Одидаемые результат (output.txt): k0x75r0x75z0x61w0x61
Полученный результат скомпилированной программы на C: k0x75r0x75z0x61w0x61
Полученный результат скомпилированной ассемблер программы: k0x75r0x75z0x61w0x61
```

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 -4[rbp] -> r12 (Были заменены не все, а часть, посколько замена всех невозвожна(происходит некомпил)).
```

7. Разбиение программы на несколько файлов и их компоновка.

1) Разобьем программу на следующие файлы:

```
main.c hex_digit.c changeVowelesToASCII.c fillArr.c printArr.c
```

2) Теперь слинкуем эти файлы с помощью следующий команд:

```
gcc ./main.c -c -o main.o
gcc ./hex_digit.c -c -o hex_digit.o
gcc ./changeVowelesToASCII.c -c -o changeVowelesToASCII.o
gcc ./printArr.c -c -o printArr.o
gcc ./fillArr.c -c -o fillArr.o
```

3) Соберем итоговый файл с помощью команды:

```
gcc ./main.o hex_digit.o changeVowelesToASCII.o printArr.o fillArr.o -o ./foo.exe
```

4) Использованы файлы "output.txt" и "input.txt" для ввода/вывода данных.