

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

МИКРОПРОЕКТ

“Программа на ассемблере для переворота строки”

Пояснительная записка

Выполнил студент группы БПИ196 (2)

Шестаков Михаил Сергеевич

1. Условие

Разработать программу, которая "переворачивает на месте" заданную ASCII-строку символов

2. Методы решения

2.1. Считывание строки

Для считывания строки используются функция **fgets**. [1] Для получения указателя на handle потока чтения пришлось использовать плохо документированную функцию **__iob_func**.

Так же поскольку размер строки заранее неизвестен используется динамическое увеличение буфера для строки с помощью функции **realloc**. [2]

Для нахождения символа перехода используется инструкция **rep scas**. [3] [4]

2.2. Разворот строки

Для поиска конца строки и нахождения количества элементов используется инструкция **rep scas**. [3] [4]

Сам разворот строки разбит на два этапа:

1) Разворот с использованием SSE3

Для этого алгоритм загружает 16 байт из начала строки и 16 байт из конца строки в регистры **xmm0** и **xmm1** с использованием инструкции **movups**, потом меняет порядок байт в них с помощью инструкции **pshufb**, и складывает обратно в строку с помощью **movups** (см. рис.1). [4][5]

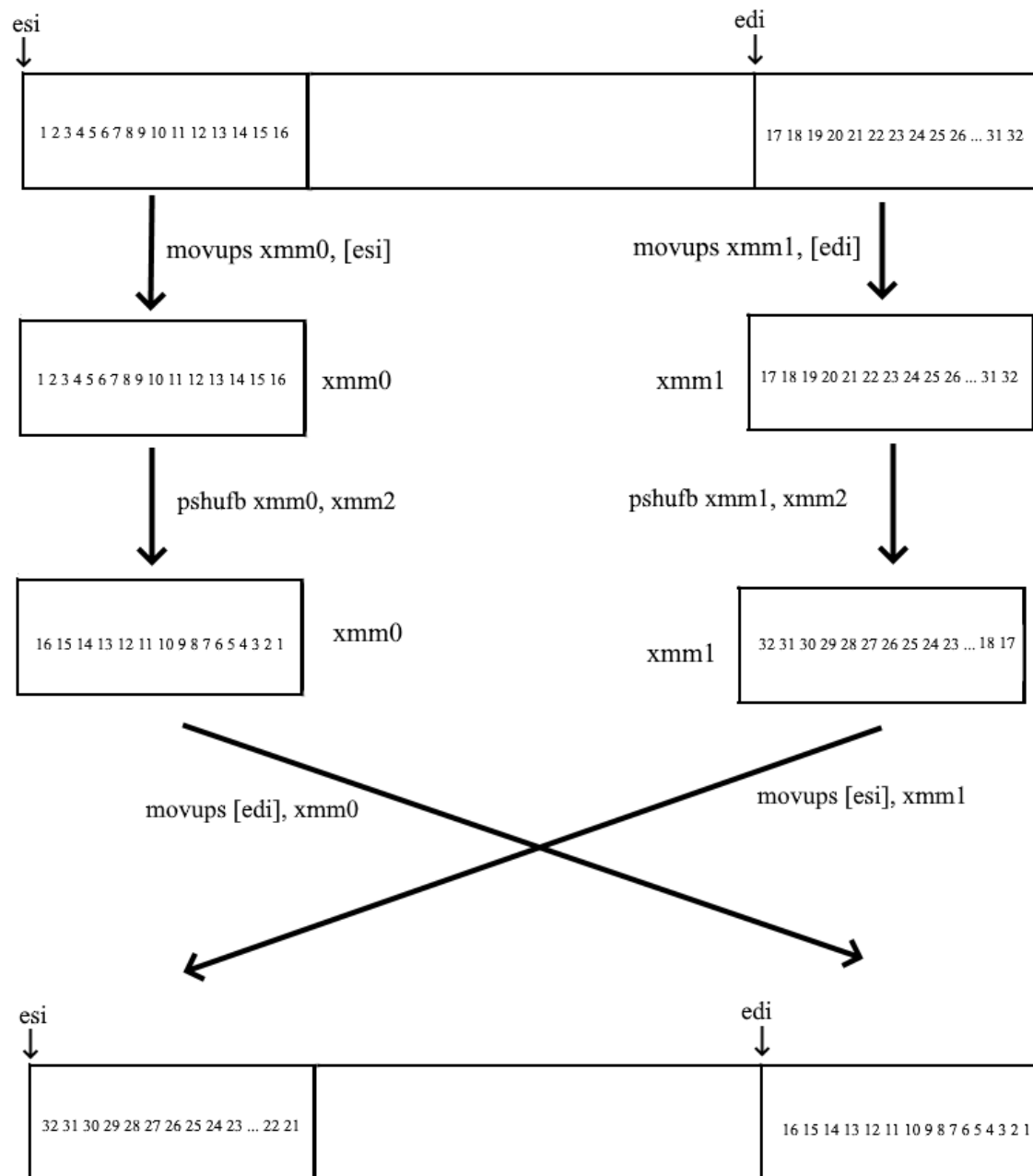
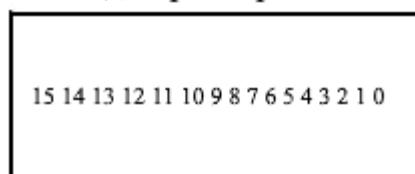


Рис. 1 Разворот строки с использованием SSE3 инструкций

маска для разворота в xmm2



2) Разворот с использованием строковых операций

Поскольку строковые операции не поддерживают одновременное перемещение **esi** вправо и **edi** влево, то реализовать разворот с использованием инструкции **rep** невозможно. Поэтому разворот строки выполняется в цикле: с помощью команды **movsb** осуществляется перенос символов из **esi** в **edi**, а затем уже с помощью обычной **mov** копируется символ из **edi** в **esi**. (более подробно см. в коде) [7]

3. Текст программы

```
; Шестаков Михаил Сергеевич
; Группа БПИ196

format PE console
entry start

include 'win32a.inc'

section '.data' data readable writable
    strAskString          db  'Input string: ', 0

    ; маска для разворота 16 байт в обратную сторону
    shuffleOrder          db  15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

    strFormatInputStr     db  'Initial string: "%s"', 10, 0
    strFormatResultStr    db  'Reversed string: "%s"', 10, 0
    strFormatInt          db  '%d', 10, 0
    strPointer            dd  0

section '.code' code readable executable
    start:

        cinvoke printf, strAskString ; просим пользователя ввести строку

        ccall ReadString              ; читаем строку
        mov [strPointer], eax

        cinvoke printf, strFormatInputStr, [strPointer] ; выводим исходную строку

        ccall ReverseString, strPointer ; переворачиваем строку на месте

        cinvoke printf, strFormatResultStr, [strPointer] ; выводим результат (перевёрнутая
строка)

        cinvoke getch
        invoke ExitProcess, 0

; char* ReadString(); -- читает строку (до перевода строки) с использованием WinApi
proc ReadString c uses edi
    locals
        s          dd  0
        s_size      dd  0 ; изначально строка пустая
        handle      dd  0
    endl
```

```

cinvoke __iob_func ; получаем FILE* stdin
mov [handle], eax

cinvoke calloc, [s_size] ; выделяем память под строку
mov [s], eax

mov ecx, -1
.readLoop:
mov eax, [s_size] ; считываем размер
add eax, 1024 ; увеличиваем буффер на 1024 символа
mov [s_size], eax ; сохраняем новый размер

cinvoke realloc, [s], eax ; увличиваем строку
mov [s], eax ; сохраняем указатель на начало строки

mov ecx, [s_size]
lea edi, [eax + ecx - 1024] ; edi -- указатель на символ, начиная с которого мы
считываем новую часть строки

cinvoke fgets, edi, 1025, [handle] ; указываем 1025, чтобы fgets прочитал 1024
символа
cmp eax, 0 ; проверяем на наличие ошибок
je .endRead ; если ничего не удалось прочитать, то выходим

mov ecx, 1024
mov al, 10 ; символ перевода строки
; edi всё ещё указывает на символ, начиная с которого мы читали
repnz SCASB
; если zf = 1, значит мы встретили перевод строки, edi указывает на символ после
него, тогда выходим
LOOPNE .readLoop ; zf = 0

.endLoop: ; edi указывает на \n, s указывает на символ после последнего
dec edi ; возвращаем edi на символ перевода строки
mov ebx, [s]

cmp ebx, edi ; провряем, вдруг пользователь ввёл просто перевод строки (без \r)
je .endRead

dec edi ; смотрим на предыдущий символ
cmp byte [edi], 13 ; проверяем, что это \r
je .endRead ; если да, то обрезаем строку в этом месте
inc edi ; если нет, то возвращаемся обратно на \n

.endRead: ; edi указывает на символ после последнего
mov byte [edi], 0 ; устанавливаем нуль-терминатор
mov eax, [s] ; возвращаем указатель на строку
ret
endp

; void ReverseString(char* str) -- переворачивает строку "на месте"
proc ReverseString c uses esi edi ebx, \
stringPointer: DWord

```

```

mov esi, [stringPointer]
mov esi, [esi] ; помещаем в esi и edi указатель на начало строки
mov edi, esi

```

```

mov ecx, -1 ; мы не знаем сколько символов в строке
xor al, al ; al = 0
cld
repne SCASB
sub edi, 2
neg ecx
sub ecx, 2
je .ret ; если строка оказалась пустая, то выходим

```

; в итоге в edi лежит указатель на последний символ строки, а в ecx количество символов (без нуля терминатора)

```

shr ecx, 5 ; делим на 32, потому что мы будем с использованием SSE3 поворачивать
; по 16 символов за раз в начале и в конце, то есть в сумме за итерацию 32

```

СИМВОЛА

```

je .simpleswap ; если ecx равен 0, то переходим к повороту строки без использования

```

SSE

; загружаем в xmm2 нужный нам порядок замены байт

```

mov eax, shuffleOrder
movups xmm2, [eax]

```

```

sub edi, 15 ; сдвигаемся на 15, потому что мы двигаем по 16 за раз

```

.loop:

; загружаем в xmm0 начало и конец

```

movups xmm0, [esi]
movups xmm1, [edi]

```

; поворачиваем начало и конец строки

; pshufb -- sse3 инструкция, которая меняет порядок байт в 128й битном
; слове src на основе порядка, указанного в dst

```

pshufb xmm0, xmm2
pshufb xmm1, xmm2

```

; вставляем конец в начало, а начало в конец

```

movups [esi], xmm1
movups [edi], xmm0

```

; сдвигаем начало и конец

```

add esi, 16
sub edi, 16

```

LOOP .loop

```

add edi, 15 ; возвращаем лишний сдвиг

```

.simpleswap: ; простой разворот строки по байту за раз

```

cld ; на всякий случай очищаем флаг направления

```

mov ecx, edi ; считаем число операций ecx = (edi - esi + 1) / 2

```

inc ecx
sub ecx, esi
shr ecx, 1

```

.loop2:

mov bl, [edi] ; в bl сохраняем значение из edi

movsb ; перемещаем символ из esi в edi, после этого esi++, edi++

```

        sub edi, 2 ; сдвигаем edi в обратную сторону
        mov [esi - 1], bl ; записываем в esi ранее сохранённый символ

    LOOP .loop2
.ret:
    ret
endp

```

-----third act - including HeapApi-----

```

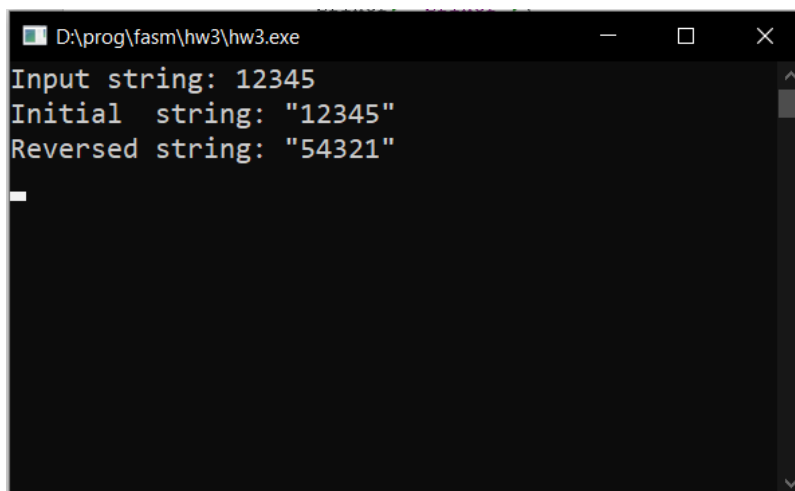
section '.idata' import data readable
    library kernel, 'kernel32.dll', \
        msvcrt, 'msvcrt.dll', \
        user32, 'USER32.DLL'

include 'api\user32.inc'
include 'api\kernel32.inc'
import kernel, \
    ExitProcess, 'ExitProcess', \
    HeapCreate, 'HeapCreate', \
    ReadConsoleA, 'ReadConsoleA', \
    GetStdHandle, 'GetStdHandle', \
    HeapAlloc, 'HeapAlloc'
include 'api\kernel32.inc'
import msvcrt, \
    fgets, 'fgets', \
    malloc, 'malloc', \
    calloc, 'calloc', \
    realloc, 'realloc', \
    strlen, 'strlen', \
    printf, 'printf', \
    scanf, 'scanf', \
    __iob_func, '__iob_func', \
    getch, '_getch'

```

4. Тестирование

4.1. Простой тест

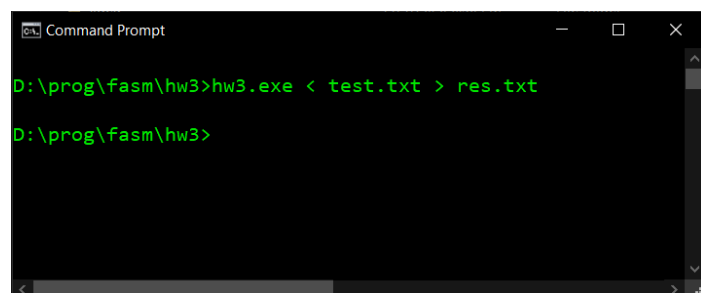


```

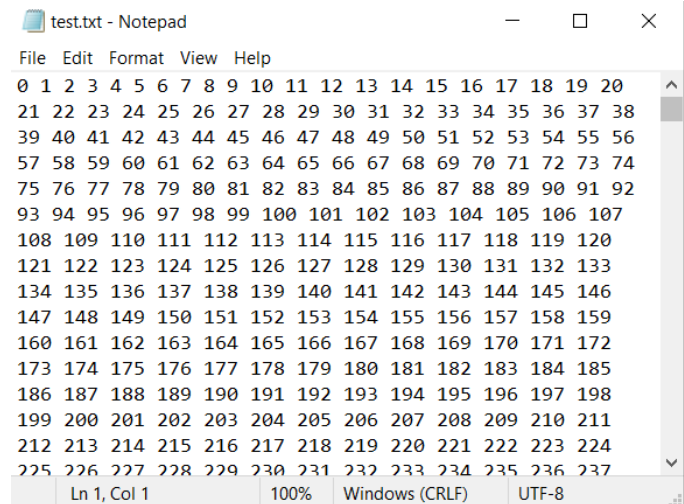
D:\prog\iasm\hw3\hw3.exe
Input string: 12345
Initial string: "12345"
Reversed string: "54321"

```

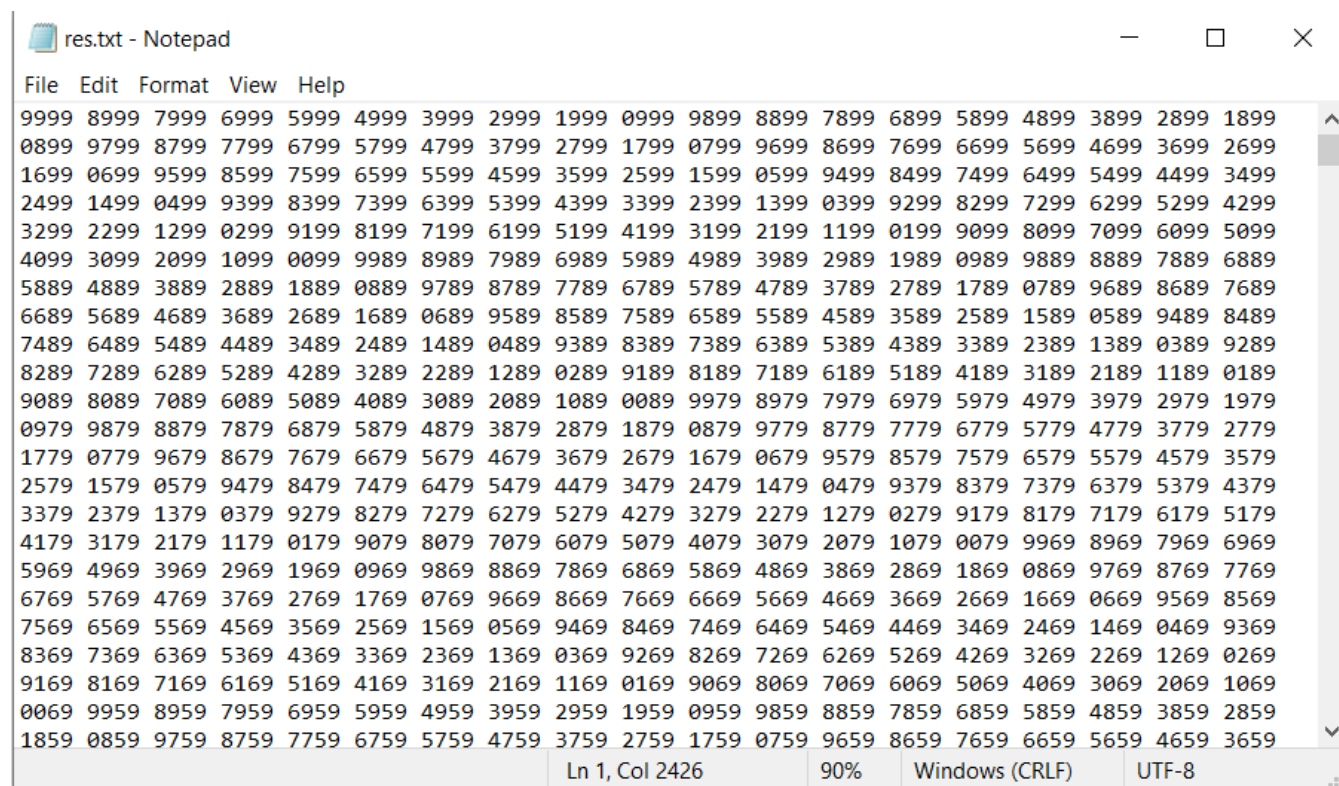

4.6. Очень длинная строка (все числа от 0 до 9999)



```
Command Prompt
D:\prog\fasm\hw3>hw3.exe < test.txt > res.txt
D:\prog\fasm\hw3>
```



```
test.txt - Notepad
File Edit Format View Help
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92
93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120
121 122 123 124 125 126 127 128 129 130 131 132 133
134 135 136 137 138 139 140 141 142 143 144 145 146
147 148 149 150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169 170 171 172
173 174 175 176 177 178 179 180 181 182 183 184 185
186 187 188 189 190 191 192 193 194 195 196 197 198
199 200 201 202 203 204 205 206 207 208 209 210 211
212 213 214 215 216 217 218 219 220 221 222 223 224
225 226 227 228 229 230 231 232 233 234 235 236 237
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```



```
res.txt - Notepad
File Edit Format View Help
9999 8999 7999 6999 5999 4999 3999 2999 1999 0999 9899 8899 7899 6899 5899 4899 3899 2899 1899
0899 9799 8799 7799 6799 5799 4799 3799 2799 1799 0799 9699 8699 7699 6699 5699 4699 3699 2699
1699 0699 9599 8599 7599 6599 5599 4599 3599 2599 1599 0599 9499 8499 7499 6499 5499 4499 3499
2499 1499 0499 9399 8399 7399 6399 5399 4399 3399 2399 1399 0399 9299 8299 7299 6299 5299 4299
3299 2299 1299 0299 9199 8199 7199 6199 5199 4199 3199 2199 1199 0199 9099 8099 7099 6099 5099
4099 3099 2099 1099 0099 9989 8989 7989 6989 5989 4989 3989 2989 1989 0989 9889 8889 7889 6889
5889 4889 3889 2889 1889 0889 9789 8789 7789 6789 5789 4789 3789 2789 1789 0789 9689 8689 7689
6689 5689 4689 3689 2689 1689 0689 9589 8589 7589 6589 5589 4589 3589 2589 1589 0589 9489 8489
7489 6489 5489 4489 3489 2489 1489 0489 9389 8389 7389 6389 5389 4389 3389 2389 1389 0389 9289
8289 7289 6289 5289 4289 3289 2289 1289 0289 9189 8189 7189 6189 5189 4189 3189 2189 1189 0189
9089 8089 7089 6089 5089 4089 3089 2089 1089 0089 9979 8979 7979 6979 5979 4979 3979 2979 1979
0979 9879 8879 7879 6879 5879 4879 3879 2879 1879 0879 9779 8779 7779 6779 5779 4779 3779 2779
1779 0779 9679 8679 7679 6679 5679 4679 3679 2679 1679 0679 9579 8579 7579 6579 5579 4579 3579
2579 1579 0579 9479 8479 7479 6479 5479 4479 3479 2479 1479 0479 9379 8379 7379 6379 5379 4379
3379 2379 1379 0379 9279 8279 7279 6279 5279 4279 3279 2279 1279 0279 9179 8179 7179 6179 5179
4179 3179 2179 1179 0179 9079 8079 7079 6079 5079 4079 3079 2079 1079 0079 9969 8969 7969 6969
5969 4969 3969 2969 1969 0969 9869 8869 7869 6869 5869 4869 3869 2869 1869 0869 9769 8769 7769
6769 5769 4769 3769 2769 1769 0769 9669 8669 7669 6669 5669 4669 3669 2669 1669 0669 9569 8569
7569 6569 5569 4569 3569 2569 1569 0569 9469 8469 7469 6469 5469 4469 3469 2469 1469 0469 9369
8369 7369 6369 5369 4369 3369 2369 1369 0369 9269 8269 7269 6269 5269 4269 3269 2269 1269 0269
9169 8169 7169 6169 5169 4169 3169 2169 1169 0169 9069 8069 7069 6069 5069 4069 3069 2069 1069
0069 9959 8959 7959 6959 5959 4959 3959 2959 1959 0959 9859 8859 7859 6859 5859 4859 3859 2859
1859 0859 9759 8759 7759 6759 5759 4759 3759 2759 1759 0759 9659 8659 7659 6659 5659 4659 3659
Ln 1, Col 2426 90% Windows (CRLF) UTF-8
```

Источники

- [1] <https://en.cppreference.com/w/c/io/fgets>
- [2] <https://en.cppreference.com/w/c/memory/realloc>
- [3] <http://www.club155.ru/x86cmd/SCAS>
- [4] <http://www.club155.ru/x86cmd/REP>
- [5] <https://www.felixcloutier.com/x86/pshufb>
- [6] <http://www.club155.ru/x86cmdsimd/MOVUPS>
- [7] <http://www.club155.ru/x86cmd/MOVSb>