МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №7

по дисциплине: Архитектура вычислительных систем
тема: «Способы вызова ассемблерных подпрограмм
в языках высокого уровня»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
ст. пр. Осипов Олег Васильевич

Белгород 2024 г.

Лабораторная работа №7
Способы вызова ассемблерных подпрограмм
в языках высокого уровня
Вариант 8

Цель работы: изучение команд поразрядной обработки данных.

Задания для выполнения к работе:

1. Написать и отладить подпрограммы на masm32 в разных стилях вызова для решения задачи соответствующего варианта. Глобальные переменные в подпрограммах использовать не разрешается. Если нужна дополнительная память, выделять её в стеке.
2. Подпрограммы собрать и скомпилировать в виде dll-библиотеки. Библиотека должна содержать:
    a. подпрограммы в стилях stdcall, cdecl, fastcall, написанные на ассемблере без явного перечисления аргументов в заголовке;
    b. Подпрограммы в стилях stdcall, cdecl, написанные, наоборот, с перечислением аргументов в заголовке подпрограммы.
3. Подключить все подпрограммы из dll-библиотеки к проектам на C# и C++ статическим и динамическим способом. Убедиться в правильности вызова всех подпрограмм.
4. Написать подпрограмму для решения задачи варианта с использованием ассемблерной вставки на языке C++.
5. Написать подпрограммы для решения задачи варианта с использованием обычного высокоуровнего языка C# и C++ (или любого другого).
6. Сравнить скорость выполнения полученных подпрограмм на одних и тех же тестовых данных. Для сравнения выбрать: подпрограмму на ассемблере в masm32 (какую-нибудь одну из пяти), вызываемую из программы на языке C++ или C#; подпрограмму на C#; подпрограмму на C++; подпрограмму на C++ с использованием ассемблерной вставки. Построить на одной плоскости графики зависимости времени выполнения подпрограмм от длины массивов (не менее 10 точек для каждой подпрограммы). Для замера лучше передавать в подпрограммы массивы большой длины. Время замерять в миллисекундах с помощью API-функции GetTickCount(). Проверить, что подпрограммы при одинаковых тестовых данных выдают одинаковый результат. Для заполнения массивов использовать генератор случайных чисел.
7. В отчёт включить весь исходный код и графики.
8. Сделать выводы по работе.

Задание:

<table>
<tr><td colspan="2">Варианты 8 - 16
Из массива a длиной length скопировать отрицательные числа в массив neg_res, положительные – в массив pos_res. Под массивы neg_res и pos_res в основной программе зарезервировать памяти столько, сколько занимает массив a. Полученные массивы отсортировать. Количество отрицательных чисел записать в выходной параметр neg_count, положительных – в выходной параметр pos_count. Исходный массив a оставить без изменений. Для удобства, можно реализовать в dll-библиотеке отдельную процедуру для сортировки одного массива.
Пример: a = {1, 3, 4, -5, 7, -2, -1, 3, 5, -5}, length = 10;
pos_res = {1, 3, 3, 4, 5, 7} (сортировка по не убыванию);
neg_res = {-1, -2, -5, -5} (сортировка по не возрастанию);
pos_count = 6;
neg_count = 4.</td></tr>
<tr><td>8</td><td>Сортировка методом вставок по не убыванию.
<code>int sort (int* a, int length, int* pos_res,
int* neg_res, int* neg_count).</code>
<code>pos_count</code> возвратить.</td></tr>
</table>

Исходный код (asm):

```
.686
.model flat, stdcall
option casemap: none

include windows.inc
include kernel32.inc
include msvcrt.inc
includelib  kernel32.lib
includelib  msvcrt.lib

.code
DllMain proc hlnstDLL:dword, reason: dword, unused: dword
mov eax, 1
ret
DllMain endp

; ARGUMENT_AMOUNT 12

; int sort_stdcall_noarg (int* a, int length, int* pos_res, int* neg_res, int* neg_count)
sort_stdcall_noarg proc
    push ebp
    mov ecx, [esp + 8]       ; ecx = a
    mov edx, [esp + 8 + 4]   ; edx = length

    ; neg_count = 0
    mov edi, dword ptr [esp + 8 + 16]
    mov dword ptr [edi], 0

    ; Используемые аргументы

    ; Выделяем место под локальную переменную current, j, current_comparing
    sub esp, 12
    ; ebx будет нашим счётчиком
    mov ebx, 0
    ; Обнуление eax
    xor eax, eax

sort_stdcall_noarg_loop_a:
        ; ebp = a
        mov ebp, ecx
        ; current = a[ebx]
        mov edi, dword ptr [ebp + ebx * 4]
        mov dword ptr [esp], edi

        ; current > 0?
        cmp dword ptr [esp], 0

        jge sort_stdcall_noarg_current_pos_zero_more
        jmp sort_stdcall_noarg_current_pos_zero_less

sort_stdcall_noarg_current_pos_zero_more:
            ; ebp = pos_res
            mov ebp, [esp + 8 + 12 + 8]
            ; j = pos_count
            mov dword ptr [esp + 4], eax
            push eax
            push edx
            mov eax, dword ptr [esp + 8 + 4]
            mov edx, 4
            mul edx
            mov dword ptr [esp + 8 + 4], eax
            pop edx
            pop eax
```

```
sort_stdcall_noarg_current_pos_zero_more_loop:
                ; j <= 0? Да - выход, нет - проверяем.
                cmp dword ptr [esp + 4], 0
                je sort_stdcall_noarg_current_pos_zero_more_loop_end
                ; current_comparing = pos_res - 1
                lea edi, dword ptr [ebp - 4]
                ; current_comparing = pos_res - 1 + j
                add edi, dword ptr [esp + 4]
                ; current_comparing = *(pos_res - 1 + j)
                mov edi, dword ptr [edi]
                ; current_comparing > current ?
                cmp edi, dword ptr [esp]

                jg sort_stdcall_noarg_current_pos_zero_more_loop_current_more
                jmp sort_stdcall_noarg_current_pos_zero_more_loop_current_less

sort_stdcall_noarg_current_pos_zero_more_loop_current_more:
                    push eax
                    ; eax = pos_res[j - 1]
                    mov eax, edi
                    ; edi = pos_res
                    mov edi, ebp
                    ; edi = pos_res + j
                    add edi, dword ptr [esp + 8]
                    ; pos_res[j] = pos_res[j - 1]
                    mov dword ptr [edi], eax

                    pop eax
                    jmp sort_stdcall_noarg_current_pos_zero_more_loop_current_end

sort_stdcall_noarg_current_pos_zero_more_loop_current_less:
                    jmp sort_stdcall_noarg_current_pos_zero_more_loop_end

sort_stdcall_noarg_current_pos_zero_more_loop_current_end:
                sub dword ptr [esp + 4], 4
                jmp sort_stdcall_noarg_current_pos_zero_more_loop

sort_stdcall_noarg_current_pos_zero_more_loop_end:
            ; ebp = pos_res
            mov ebp, [esp + 8 + 12 + 8]
            ; current_comparing = pos_res
            lea edi, dword ptr [ebp]
            ; current_comparing = pos_res + j
            add edi, dword ptr [esp + 4]
            push eax
            mov eax, dword ptr [esp + 4]
            mov dword ptr [edi], eax
            pop eax

            ; pos_count++
            inc eax
            jmp sort_stdcall_noarg_current_pos_zero_end

sort_stdcall_noarg_current_pos_zero_less:
            ; ebp = neg_res
            ; swap a
            mov edi, [esp + 8 + 12 + 16]
            xor eax, dword ptr [edi]
            xor dword ptr [edi], eax
            xor eax, dword ptr [edi]

            ; ebp = neg_res
            mov ebp, [esp + 8 + 12 + 12]
            ; j = pos_count
            mov dword ptr [esp + 4], eax
            push eax
            push edx
```

```
            mov eax, dword ptr [esp + 8 + 4]
            mov edx, 4
            mul edx
            mov dword ptr [esp + 8 + 4], eax
            pop edx
            pop eax

sort_stdcall_noarg_current_pos_zero_less_loop:
                ; j <= 0? Да - выход, нет - проверяем.
                cmp dword ptr [esp + 4], 0
                je sort_stdcall_noarg_current_pos_zero_less_loop_end
                ; current_comparing = pos_res - 1
                lea edi, dword ptr [ebp - 4]
                ; current_comparing = pos_res - 1 + j
                add edi, dword ptr [esp + 4]
                ; current_comparing = *(pos_res - 1 + j)
                mov edi, dword ptr [edi]
                ; current_comparing > current ?
                cmp edi, dword ptr [esp]

                jg sort_stdcall_noarg_current_pos_zero_less_loop_current_more
                jmp sort_stdcall_noarg_current_pos_zero_less_loop_current_less

sort_stdcall_noarg_current_pos_zero_less_loop_current_more:
                    push eax
                    ; eax = pos_res[j - 1]
                    mov eax, edi
                    ; edi = pos_res
                    mov edi, ebp
                    ; edi = pos_res + j
                    add edi, dword ptr [esp + 8]
                    ; pos_res[j] = pos_res[j - 1]
                    mov dword ptr [edi], eax

                    pop eax
                    jmp sort_stdcall_noarg_current_pos_zero_less_loop_current_end

sort_stdcall_noarg_current_pos_zero_less_loop_current_less:
                    jmp sort_stdcall_noarg_current_pos_zero_less_loop_end

sort_stdcall_noarg_current_pos_zero_less_loop_current_end:
                sub dword ptr [esp + 4], 4
                jmp sort_stdcall_noarg_current_pos_zero_less_loop

sort_stdcall_noarg_current_pos_zero_less_loop_end:
            ; ebp = pos_res
            mov ebp, [esp + 8 + 12 + 12]
            ; current_comparing = pos_res
            lea edi, dword ptr [ebp]
            ; current_comparing = pos_res + j
            add edi, dword ptr [esp + 4]
            push eax
            mov eax, dword ptr [esp + 4]
            mov dword ptr [edi], eax
            pop eax

            inc eax
            ; swap b
            mov edi, [esp + 8 + 12 + 16]
            xor eax, dword ptr [edi]
            xor dword ptr [edi], eax
            xor eax, dword ptr [edi]

            jmp sort_stdcall_noarg_current_pos_zero_end

sort_stdcall_noarg_current_pos_zero_end:
```

```
        ; ebx++
        inc ebx
        ; Если ebx < length, переходим на следующую итерацию цикла
        cmp ebx, edx
        jl sort_stdcall_noarg_loop_a

    add esp, 12
    pop ebp
    ret 5 * 4
sort_stdcall_noarg endp

; int sort_cdecl_noarg (int* a, int length, int* pos_res, int* neg_res, int* neg_count)
sort_cdecl_noarg proc
    push ebp
    mov ecx, [esp + 8]       ; ecx = a
    mov edx, [esp + 8 + 4]   ; edx = length

    ; neg_count = 0
    mov edi, dword ptr [esp + 8 + 16]
    mov dword ptr [edi], 0

    ; Используемые аргументы

    ; Выделяем место под локальную переменную current, j, current_comparing
    sub esp, 12
    ; ebx будет нашим счётчиком
    mov ebx, 0
    ; Обнуление eax
    xor eax, eax

sort_cdecl_noarg_loop_a:
        ; ebp = a
        mov ebp, ecx
        ; current = a[ebx]
        mov edi, dword ptr [ebp + ebx * 4]
        mov dword ptr [esp], edi

        ; current > 0?
        cmp dword ptr [esp], 0

        jge sort_cdecl_noarg_current_pos_zero_more
        jmp sort_cdecl_noarg_current_pos_zero_less

sort_cdecl_noarg_current_pos_zero_more:
            ; ebp = pos_res
            mov ebp, [esp + 8 + 12 + 8]
            ; j = pos_count
            mov dword ptr [esp + 4], eax
            push eax
            push edx
            mov eax, dword ptr [esp + 8 + 4]
            mov edx, 4
            mul edx
            mov dword ptr [esp + 8 + 4], eax
            pop edx
            pop eax

sort_cdecl_noarg_current_pos_zero_more_loop:
                ; j <= 0? Да - выход, нет - проверяем.
                cmp dword ptr [esp + 4], 0
                je sort_cdecl_noarg_current_pos_zero_more_loop_end
                ; current_comparing = pos_res - 1
                lea edi, dword ptr [ebp - 4]
                ; current_comparing = pos_res - 1 + j
                add edi, dword ptr [esp + 4]
                ; current_comparing = *(pos_res - 1 + j)
                mov edi, dword ptr [edi]
```

```
                ; current_comparing > current ?
                cmp edi, dword ptr [esp]

                jg sort_cdecl_noarg_current_pos_zero_more_loop_current_more
                jmp sort_cdecl_noarg_current_pos_zero_more_loop_current_less

sort_cdecl_noarg_current_pos_zero_more_loop_current_more:
                    push eax
                    ; eax = pos_res[j - 1]
                    mov eax, edi
                    ; edi = pos_res
                    mov edi, ebp
                    ; edi = pos_res + j
                    add edi, dword ptr [esp + 8]
                    ; pos_res[j] = pos_res[j - 1]
                    mov dword ptr [edi], eax

                    pop eax
                    jmp sort_cdecl_noarg_current_pos_zero_more_loop_current_end

sort_cdecl_noarg_current_pos_zero_more_loop_current_less:
                    jmp sort_cdecl_noarg_current_pos_zero_more_loop_end

sort_cdecl_noarg_current_pos_zero_more_loop_current_end:
                sub dword ptr [esp + 4], 4
                jmp sort_cdecl_noarg_current_pos_zero_more_loop

sort_cdecl_noarg_current_pos_zero_more_loop_end:
            ; ebp = pos_res
            mov ebp, [esp + 8 + 12 + 8]
            ; current_comparing = pos_res
            lea edi, dword ptr [ebp]
            ; current_comparing = pos_res + j
            add edi, dword ptr [esp + 4]
            push eax
            mov eax, dword ptr [esp + 4]
            mov dword ptr [edi], eax
            pop eax

            ; pos_count++
            inc eax
            jmp sort_cdecl_noarg_current_pos_zero_end

sort_cdecl_noarg_current_pos_zero_less:
            ; ebp = neg_res
            ; swap a
            mov edi, [esp + 8 + 12 + 16]
            xor eax, dword ptr [edi]
            xor dword ptr [edi], eax
            xor eax, dword ptr [edi]

            ; ebp = neg_res
            mov ebp, [esp + 8 + 12 + 12]
            ; j = pos_count
            mov dword ptr [esp + 4], eax
            push eax
            push edx
            mov eax, dword ptr [esp + 8 + 4]
            mov edx, 4
            mul edx
            mov dword ptr [esp + 8 + 4], eax
            pop edx
            pop eax

sort_cdecl_noarg_current_pos_zero_less_loop:
                ; j <= 0? Да - выход, нет - проверяем.
                cmp dword ptr [esp + 4], 0
```

```
                je sort_cdecl_noarg_current_pos_zero_less_loop_end
                ; current_comparing = pos_res - 1
                lea edi, dword ptr [ebp - 4]
                ; current_comparing = pos_res - 1 + j
                add edi, dword ptr [esp + 4]
                ; current_comparing = *(pos_res - 1 + j)
                mov edi, dword ptr [edi]
                ; current_comparing > current ?
                cmp edi, dword ptr [esp]

                jg sort_cdecl_noarg_current_pos_zero_less_loop_current_more
                jmp sort_cdecl_noarg_current_pos_zero_less_loop_current_less

sort_cdecl_noarg_current_pos_zero_less_loop_current_more:
                    push eax
                    ; eax = pos_res[j - 1]
                    mov eax, edi
                    ; edi = pos_res
                    mov edi, ebp
                    ; edi = pos_res + j
                    add edi, dword ptr [esp + 8]
                    ; pos_res[j] = pos_res[j - 1]
                    mov dword ptr [edi], eax

                    pop eax
                    jmp sort_cdecl_noarg_current_pos_zero_less_loop_current_end

sort_cdecl_noarg_current_pos_zero_less_loop_current_less:
                    jmp sort_cdecl_noarg_current_pos_zero_less_loop_end

sort_cdecl_noarg_current_pos_zero_less_loop_current_end:
                sub dword ptr [esp + 4], 4
                jmp sort_cdecl_noarg_current_pos_zero_less_loop

sort_cdecl_noarg_current_pos_zero_less_loop_end:
            ; ebp = pos_res
            mov ebp, [esp + 8 + 12 + 12]
            ; current_comparing = pos_res
            lea edi, dword ptr [ebp]
            ; current_comparing = pos_res + j
            add edi, dword ptr [esp + 4]
            push eax
            mov eax, dword ptr [esp + 4]
            mov dword ptr [edi], eax
            pop eax

            inc eax
            ; swap b
            mov edi, [esp + 8 + 12 + 16]
            xor eax, dword ptr [edi]
            xor dword ptr [edi], eax
            xor eax, dword ptr [edi]

            jmp sort_cdecl_noarg_current_pos_zero_end

sort_cdecl_noarg_current_pos_zero_end:

        ; ebx++
        inc ebx
        ; Если ebx < length, переходим на следующую итерацию цикла
        cmp ebx, edx
        jl sort_cdecl_noarg_loop_a

    add esp, 12
    pop ebp
    ret
sort_cdecl_noarg endp
```

```
; int sort_fastcall_noarg (int* a, int length, int* pos_res, int* neg_res, int* neg_count)
sort_fastcall_noarg proc
    push ebp
    ; ecx = a
    ; edx = length

    ; neg_count = 0
    mov edi, dword ptr [esp + 8 + 8]
    mov dword ptr [edi], 0

    ; Используемые аргументы

    ; Выделяем место под локальную переменную current, j, current_comparing
    sub esp, 12
    ; ebx будет нашим счётчиком
    mov ebx, 0
    ; Обнуление eax
    xor eax, eax

sort_fastcall_noarg_loop_a:
        ; ebp = a
        mov ebp, ecx
        ; current = a[ebx]
        mov edi, dword ptr [ebp + ebx * 4]
        mov dword ptr [esp], edi

        ; current > 0?
        cmp dword ptr [esp], 0

        jge sort_fastcall_noarg_current_pos_zero_more
        jmp sort_fastcall_noarg_current_pos_zero_less

sort_fastcall_noarg_current_pos_zero_more:
            ; ebp = pos_res
            mov ebp, [esp + 8 + 12]
            ; j = pos_count
            mov dword ptr [esp + 4], eax
            push eax
            push edx
            mov eax, dword ptr [esp + 8 + 4]
            mov edx, 4
            mul edx
            mov dword ptr [esp + 8 + 4], eax
            pop edx
            pop eax

sort_fastcall_noarg_current_pos_zero_more_loop:
                ; j <= 0? Да - выход, нет - проверяем.
                cmp dword ptr [esp + 4], 0
                je sort_fastcall_noarg_current_pos_zero_more_loop_end
                ; current_comparing = pos_res - 1
                lea edi, dword ptr [ebp - 4]
                ; current_comparing = pos_res - 1 + j
                add edi, dword ptr [esp + 4]
                ; current_comparing = *(pos_res - 1 + j)
                mov edi, dword ptr [edi]
                ; current_comparing > current ?
                cmp edi, dword ptr [esp]

                jg sort_fastcall_noarg_current_pos_zero_more_loop_current_more
                jmp sort_fastcall_noarg_current_pos_zero_more_loop_current_less

sort_fastcall_noarg_current_pos_zero_more_loop_current_more:
                    push eax
                    ; eax = pos_res[j - 1]
                    mov eax, edi
```

```
                    ; edi = pos_res
                    mov edi, ebp
                    ; edi = pos_res + j
                    add edi, dword ptr [esp + 8]
                    ; pos_res[j] = pos_res[j - 1]
                    mov dword ptr [edi], eax

                    pop eax
                    jmp sort_fastcall_noarg_current_pos_zero_more_loop_current_end

sort_fastcall_noarg_current_pos_zero_more_loop_current_less:
                    jmp sort_fastcall_noarg_current_pos_zero_more_loop_end

sort_fastcall_noarg_current_pos_zero_more_loop_current_end:
                sub dword ptr [esp + 4], 4
                jmp sort_fastcall_noarg_current_pos_zero_more_loop

sort_fastcall_noarg_current_pos_zero_more_loop_end:
            ; ebp = pos_res
            mov ebp, [esp + 8 + 12]
            ; current_comparing = pos_res
            lea edi, dword ptr [ebp]
            ; current_comparing = pos_res + j
            add edi, dword ptr [esp + 4]
            push eax
            mov eax, dword ptr [esp + 4]
            mov dword ptr [edi], eax
            pop eax

            ; pos_count++
            inc eax
            jmp sort_fastcall_noarg_current_pos_zero_end

sort_fastcall_noarg_current_pos_zero_less:
            ; ebp = neg_res
            ; swap a
            mov edi, [esp + 8 + 12 + 8]
            xor eax, dword ptr [edi]
            xor dword ptr [edi], eax
            xor eax, dword ptr [edi]

            ; ebp = neg_res
            mov ebp, [esp + 8 + 12 + 4]
            ; j = pos_count
            mov dword ptr [esp + 4], eax
            push eax
            push edx
            mov eax, dword ptr [esp + 8 + 4]
            mov edx, 4
            mul edx
            mov dword ptr [esp + 8 + 4], eax
            pop edx
            pop eax

sort_fastcall_noarg_current_pos_zero_less_loop:
                ; j <= 0? Да - выход, нет - проверяем.
                cmp dword ptr [esp + 4], 0
                je sort_fastcall_noarg_current_pos_zero_less_loop_end
                ; current_comparing = pos_res - 1
                lea edi, dword ptr [ebp - 4]
                ; current_comparing = pos_res - 1 + j
                add edi, dword ptr [esp + 4]
                ; current_comparing = *(pos_res - 1 + j)
                mov edi, dword ptr [edi]
                ; current_comparing > current ?
                cmp edi, dword ptr [esp]
```

```
                jg sort_fastcall_noarg_current_pos_zero_less_loop_current_more
                jmp sort_fastcall_noarg_current_pos_zero_less_loop_current_less

sort_fastcall_noarg_current_pos_zero_less_loop_current_more:
                    push eax
                    ; eax = pos_res[j - 1]
                    mov eax, edi
                    ; edi = pos_res
                    mov edi, ebp
                    ; edi = pos_res + j
                    add edi, dword ptr [esp + 8]
                    ; pos_res[j] = pos_res[j - 1]
                    mov dword ptr [edi], eax

                    pop eax
                    jmp sort_fastcall_noarg_current_pos_zero_less_loop_current_end

sort_fastcall_noarg_current_pos_zero_less_loop_current_less:
                    jmp sort_fastcall_noarg_current_pos_zero_less_loop_end

sort_fastcall_noarg_current_pos_zero_less_loop_current_end:
                sub dword ptr [esp + 4], 4
                jmp sort_fastcall_noarg_current_pos_zero_less_loop

sort_fastcall_noarg_current_pos_zero_less_loop_end:
            ; ebp = pos_res
            mov ebp, [esp + 8 + 12 + 4]
            ; current_comparing = pos_res
            lea edi, dword ptr [ebp]
            ; current_comparing = pos_res + j
            add edi, dword ptr [esp + 4]
            push eax
            mov eax, dword ptr [esp + 4]
            mov dword ptr [edi], eax
            pop eax

            inc eax
            ; swap b
            mov edi, [esp + 8 + 12 + 8]
            xor eax, dword ptr [edi]
            xor dword ptr [edi], eax
            xor eax, dword ptr [edi]

            jmp sort_fastcall_noarg_current_pos_zero_end

sort_fastcall_noarg_current_pos_zero_end:

        ; ebx++
        inc ebx
        ; Если ebx < length, переходим на следующую итерацию цикла
        cmp ebx, edx
        jl sort_fastcall_noarg_loop_a

    add esp, 12
    pop ebp
    ret 3 * 4
sort_fastcall_noarg endp

; int sort_stdcall (int* a, int length, int* pos_res, int* neg_res, int* neg_count)
sort_stdcall proc stdcall a: DWORD, len: DWORD, pos_res: DWORD, neg_res: DWORD, neg_count: DWORD
    push esi
    mov ecx, a       ; ecx = a
    mov edx, len   ; edx = length

    ; neg_count = 0
    mov edi, dword ptr [neg_count]
    mov dword ptr [edi], 0
```

```
    ; Используемые аргументы

    ; Выделяем место под локальную переменную current, j, current_comparing
    sub esp, 12
    ; ebx будет нашим счётчиком
    mov ebx, 0
    ; Обнуление eax
    xor eax, eax

sort_stdcall_loop_a:
        ; esi = a
        mov esi, ecx
        ; current = a[ebx]
        mov edi, dword ptr [esi + ebx * 4]
        mov dword ptr [esp], edi

        ; current > 0?
        cmp dword ptr [esp], 0

        jge sort_stdcall_current_pos_zero_more
        jmp sort_stdcall_current_pos_zero_less

sort_stdcall_current_pos_zero_more:
            ; esi = pos_res
            mov esi, [pos_res]
            ; j = pos_count
            mov dword ptr [esp + 4], eax
            push eax
            push edx
            mov eax, dword ptr [esp + 8 + 4]
            mov edx, 4
            mul edx
            mov dword ptr [esp + 8 + 4], eax
            pop edx
            pop eax

sort_stdcall_current_pos_zero_more_loop:
                ; j <= 0? Да - выход, нет - проверяем.
                cmp dword ptr [esp + 4], 0
                je sort_stdcall_current_pos_zero_more_loop_end
                ; current_comparing = pos_res - 1
                lea edi, dword ptr [esi - 4]
                ; current_comparing = pos_res - 1 + j
                add edi, dword ptr [esp + 4]
                ; current_comparing = *(pos_res - 1 + j)
                mov edi, dword ptr [edi]
                ; current_comparing > current ?
                cmp edi, dword ptr [esp]

                jg sort_stdcall_current_pos_zero_more_loop_current_more
                jmp sort_stdcall_current_pos_zero_more_loop_current_less

sort_stdcall_current_pos_zero_more_loop_current_more:
                    push eax
                    ; eax = pos_res[j - 1]
                    mov eax, edi
                    ; edi = pos_res
                    mov edi, esi
                    ; edi = pos_res + j
                    add edi, dword ptr [esp + 8]
                    ; pos_res[j] = pos_res[j - 1]
                    mov dword ptr [edi], eax

                    pop eax
                    jmp sort_stdcall_current_pos_zero_more_loop_current_end
```

```
sort_stdcall_current_pos_zero_more_loop_current_less:
                    jmp sort_stdcall_current_pos_zero_more_loop_end

sort_stdcall_current_pos_zero_more_loop_current_end:
                sub dword ptr [esp + 4], 4
                jmp sort_stdcall_current_pos_zero_more_loop

sort_stdcall_current_pos_zero_more_loop_end:
            ; esi = pos_res
            mov esi, [pos_res]
            ; current_comparing = pos_res
            lea edi, dword ptr [esi]
            ; current_comparing = pos_res + j
            add edi, dword ptr [esp + 4]
            push eax
            mov eax, dword ptr [esp + 4]
            mov dword ptr [edi], eax
            pop eax

            ; pos_count++
            inc eax
            jmp sort_stdcall_current_pos_zero_end

sort_stdcall_current_pos_zero_less:
            ; esi = neg_res
            ; swap a
            mov edi, [neg_count]
            xor eax, dword ptr [edi]
            xor dword ptr [edi], eax
            xor eax, dword ptr [edi]

            ; esi = neg_res
            mov esi, [neg_res]
            ; j = pos_count
            mov dword ptr [esp + 4], eax
            push eax
            push edx
            mov eax, dword ptr [esp + 8 + 4]
            mov edx, 4
            mul edx
            mov dword ptr [esp + 8 + 4], eax
            pop edx
            pop eax

sort_stdcall_current_pos_zero_less_loop:
                ; j <= 0? Да - выход, нет - проверяем.
                cmp dword ptr [esp + 4], 0
                je sort_stdcall_current_pos_zero_less_loop_end
                ; current_comparing = pos_res - 1
                lea edi, dword ptr [esi - 4]
                ; current_comparing = pos_res - 1 + j
                add edi, dword ptr [esp + 4]
                ; current_comparing = *(pos_res - 1 + j)
                mov edi, dword ptr [edi]
                ; current_comparing > current ?
                cmp edi, dword ptr [esp]

                jg sort_stdcall_current_pos_zero_less_loop_current_more
                jmp sort_stdcall_current_pos_zero_less_loop_current_less

sort_stdcall_current_pos_zero_less_loop_current_more:
                    push eax
                    ; eax = pos_res[j - 1]
                    mov eax, edi
                    ; edi = pos_res
                    mov edi, esi
                    ; edi = pos_res + j
```

```
                    add edi, dword ptr [esp + 8]
                    ; pos_res[j] = pos_res[j - 1]
                    mov dword ptr [edi], eax

                    pop eax
                    jmp sort_stdcall_current_pos_zero_less_loop_current_end

sort_stdcall_current_pos_zero_less_loop_current_less:
                    jmp sort_stdcall_current_pos_zero_less_loop_end

sort_stdcall_current_pos_zero_less_loop_current_end:
                sub dword ptr [esp + 4], 4
                jmp sort_stdcall_current_pos_zero_less_loop

sort_stdcall_current_pos_zero_less_loop_end:
            ; esi = pos_res
            mov esi, [neg_res]
            ; current_comparing = pos_res
            lea edi, dword ptr [esi]
            ; current_comparing = pos_res + j
            add edi, dword ptr [esp + 4]
            push eax
            mov eax, dword ptr [esp + 4]
            mov dword ptr [edi], eax
            pop eax

            inc eax
            ; swap b
            mov edi, [neg_count]
            xor eax, dword ptr [edi]
            xor dword ptr [edi], eax
            xor eax, dword ptr [edi]

            jmp sort_stdcall_current_pos_zero_end

sort_stdcall_current_pos_zero_end:

        ; ebx++
        inc ebx
        ; Если ebx < length, переходим на следующую итерацию цикла
        cmp ebx, edx
        jl sort_stdcall_loop_a

    add esp, 12
    pop esi
    ret
sort_stdcall endp

; int sort_cdecl (int* a, int length, int* pos_res, int* neg_res, int* neg_count)
sort_cdecl proc c a: DWORD, len: DWORD, pos_res: DWORD, neg_res: DWORD, neg_count: DWORD
push esi
    mov ecx, a       ; ecx = a
    mov edx, len   ; edx = length

    ; neg_count = 0
    mov edi, dword ptr [neg_count]
    mov dword ptr [edi], 0

    ; Используемые аргументы

    ; Выделяем место под локальную переменную current, j, current_comparing
    sub esp, 12
    ; ebx будет нашим счётчиком
    mov ebx, 0
    ; Обнуление eax
    xor eax, eax
```

```
sort_cdecl_loop_a:
        ; esi = a
        mov esi, ecx
        ; current = a[ebx]
        mov edi, dword ptr [esi + ebx * 4]
        mov dword ptr [esp], edi

        ; current > 0?
        cmp dword ptr [esp], 0

        jge sort_cdecl_current_pos_zero_more
        jmp sort_cdecl_current_pos_zero_less

sort_cdecl_current_pos_zero_more:
            ; esi = pos_res
            mov esi, [pos_res]
            ; j = pos_count
            mov dword ptr [esp + 4], eax
            push eax
            push edx
            mov eax, dword ptr [esp + 8 + 4]
            mov edx, 4
            mul edx
            mov dword ptr [esp + 8 + 4], eax
            pop edx
            pop eax

sort_cdecl_current_pos_zero_more_loop:
                ; j <= 0? Да - выход, нет - проверяем.
                cmp dword ptr [esp + 4], 0
                je sort_cdecl_current_pos_zero_more_loop_end
                ; current_comparing = pos_res - 1
                lea edi, dword ptr [esi - 4]
                ; current_comparing = pos_res - 1 + j
                add edi, dword ptr [esp + 4]
                ; current_comparing = *(pos_res - 1 + j)
                mov edi, dword ptr [edi]
                ; current_comparing > current ?
                cmp edi, dword ptr [esp]

                jg sort_cdecl_current_pos_zero_more_loop_current_more
                jmp sort_cdecl_current_pos_zero_more_loop_current_less

sort_cdecl_current_pos_zero_more_loop_current_more:
                    push eax
                    ; eax = pos_res[j - 1]
                    mov eax, edi
                    ; edi = pos_res
                    mov edi, esi
                    ; edi = pos_res + j
                    add edi, dword ptr [esp + 8]
                    ; pos_res[j] = pos_res[j - 1]
                    mov dword ptr [edi], eax

                    pop eax
                    jmp sort_cdecl_current_pos_zero_more_loop_current_end

sort_cdecl_current_pos_zero_more_loop_current_less:
                    jmp sort_cdecl_current_pos_zero_more_loop_end

sort_cdecl_current_pos_zero_more_loop_current_end:
                sub dword ptr [esp + 4], 4
                jmp sort_cdecl_current_pos_zero_more_loop

sort_cdecl_current_pos_zero_more_loop_end:
            ; esi = pos_res
            mov esi, [pos_res]
```

```
            ; current_comparing = pos_res
            lea edi, dword ptr [esi]
            ; current_comparing = pos_res + j
            add edi, dword ptr [esp + 4]
            push eax
            mov eax, dword ptr [esp + 4]
            mov dword ptr [edi], eax
            pop eax

            ; pos_count++
            inc eax
            jmp sort_cdecl_current_pos_zero_end

sort_cdecl_current_pos_zero_less:
            ; esi = neg_res
            ; swap a
            mov edi, [neg_count]
            xor eax, dword ptr [edi]
            xor dword ptr [edi], eax
            xor eax, dword ptr [edi]

            ; esi = neg_res
            mov esi, [neg_res]
            ; j = pos_count
            mov dword ptr [esp + 4], eax
            push eax
            push edx
            mov eax, dword ptr [esp + 8 + 4]
            mov edx, 4
            mul edx
            mov dword ptr [esp + 8 + 4], eax
            pop edx
            pop eax

sort_cdecl_current_pos_zero_less_loop:
                ; j <= 0? Да - выход, нет - проверяем.
                cmp dword ptr [esp + 4], 0
                je sort_cdecl_current_pos_zero_less_loop_end
                ; current_comparing = pos_res - 1
                lea edi, dword ptr [esi - 4]
                ; current_comparing = pos_res - 1 + j
                add edi, dword ptr [esp + 4]
                ; current_comparing = *(pos_res - 1 + j)
                mov edi, dword ptr [edi]
                ; current_comparing > current ?
                cmp edi, dword ptr [esp]

                jg sort_cdecl_current_pos_zero_less_loop_current_more
                jmp sort_cdecl_current_pos_zero_less_loop_current_less

sort_cdecl_current_pos_zero_less_loop_current_more:
                    push eax
                    ; eax = pos_res[j - 1]
                    mov eax, edi
                    ; edi = pos_res
                    mov edi, esi
                    ; edi = pos_res + j
                    add edi, dword ptr [esp + 8]
                    ; pos_res[j] = pos_res[j - 1]
                    mov dword ptr [edi], eax

                    pop eax
                    jmp sort_cdecl_current_pos_zero_less_loop_current_end

sort_cdecl_current_pos_zero_less_loop_current_less:
                    jmp sort_cdecl_current_pos_zero_less_loop_end
```

```
sort_cdecl_current_pos_zero_less_loop_current_end:
                sub dword ptr [esp + 4], 4
                jmp sort_cdecl_current_pos_zero_less_loop

sort_cdecl_current_pos_zero_less_loop_end:
            ; esi = pos_res
            mov esi, [neg_res]
            ; current_comparing = pos_res
            lea edi, dword ptr [esi]
            ; current_comparing = pos_res + j
            add edi, dword ptr [esp + 4]
            push eax
            mov eax, dword ptr [esp + 4]
            mov dword ptr [edi], eax
            pop eax

            inc eax
            ; swap b
            mov edi, [neg_count]
            xor eax, dword ptr [edi]
            xor dword ptr [edi], eax
            xor eax, dword ptr [edi]

            jmp sort_cdecl_current_pos_zero_end

sort_cdecl_current_pos_zero_end:

        ; ebx++
        inc ebx
        ; Если ebx < length, переходим на следующую итерацию цикла
        cmp ebx, edx
        jl sort_cdecl_loop_a

    add esp, 12
    pop esi
    ret
sort_cdecl endp

end DllMain
```

libs.def:

```
LIBRARY libs
EXPORTS

_sort_stdcall_noarg@20 = _sort_stdcall_noarg@0
_sort_cdecl_noarg = _sort_cdecl_noarg@0
@sort_fastcall_noarg@20 = _sort_fastcall_noarg@0
sort_cdecl
sort_stdcall
```

Исходный тестирующий код (C++):

```
#include <iostream>
#include <vector>
#include <assert.h>
#include <chrono>

#pragma comment(lib, "libs.lib")

extern "C" __declspec(dllimport) int _stdcall  sort_stdcall_noarg  (int* a, int length, int*
pos_res, int* neg_res, int* neg_count);
extern "C" __declspec(dllimport) int _cdecl    sort_cdecl_noarg    (int* a, int length, int*
pos_res, int* neg_res, int* neg_count);
extern "C" __declspec(dllimport) int _fastcall sort_fastcall_noarg (int* a, int length, int*
pos_res, int* neg_res, int* neg_count);
```

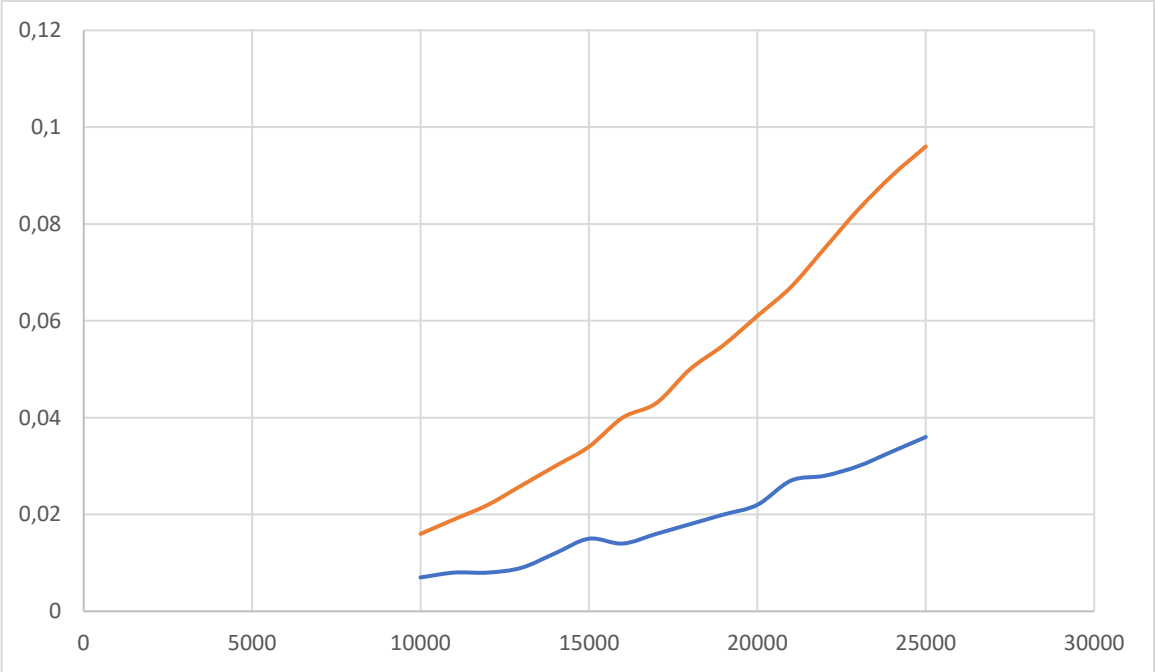
```
extern "C" __declspec(dllimport) int _stdcall  sort_stdcall        (int* a, int length, int*
pos_res, int* neg_res, int* neg_count);
extern "C" __declspec(dllimport) int _cdecl    sort_cdecl          (int* a, int length, int*
pos_res, int* neg_res, int* neg_count);

int sort_native(int* a, int length, int* pos_res, int* neg_res, int* neg_count) {
    int pos_count = 0;
    *neg_count = 0;

    for (int i = 0; i < length; i++) {
        if (a[i] > 0) {
            int j = pos_count;
            while (j > 0 && pos_res[j - 1] > a[i]) {
                pos_res[j] = pos_res[j - 1];
                j--;
            }

            pos_res[j] = a[i];
            pos_count++;
        } else {
            int j = *neg_count;
            while (j > 0 && neg_res[j - 1] > a[i]) {
                neg_res[j] = neg_res[j - 1];
                j--;
            }

            neg_res[j] = a[i];
            (*neg_count)++;
        }
    }

    return pos_count;
}

template <typename TestedFunction>
void test_function1(TestedFunction func_to_test) {
    int a[] = { -1, -2, -3, -4, -5, -6 };
    int pos_res[6] = {};
    int neg_res[6] = {};
    int neg_count;
    auto pos_count = func_to_test(a, 6, pos_res, neg_res, (int*)(&neg_count));

    assert(pos_count == 0);
    assert(neg_count == 6);
    assert(
        neg_res[0] == -6 &&
        neg_res[1] == -5 &&
        neg_res[2] == -4 &&
        neg_res[3] == -3 &&
        neg_res[4] == -2 &&
        neg_res[5] == -1);
}

template <typename TestedFunction>
void test_function2(TestedFunction func_to_test) {
    int a[] = { 6, 5, 4, 3, 2, 1 };
    int pos_res[6] = {};
    int neg_res[6] = {};
    int neg_count;
    auto pos_count = func_to_test(a, 6, pos_res, neg_res, (int*)(&neg_count));

    assert(pos_count == 6);
    assert(neg_count == 0);
    assert(
        pos_res[0] == 1 &&
        pos_res[1] == 2 &&
        pos_res[2] == 3 &&
```

```
        pos_res[3] == 4 &&
        pos_res[4] == 5 &&
        pos_res[5] == 6);
}

template <typename TestedFunction>
void test_function3(TestedFunction func_to_test) {
    int a[] = { -1, 2, -3, 3, 45, -6 };
    int pos_res[6] = {};
    int neg_res[6] = {};
    int neg_count;
    auto pos_count = func_to_test(a, 6, pos_res, neg_res, (int*)(&neg_count));

    assert(pos_count == 3);
    assert(neg_count == 3);
    assert(
        neg_res[0] == -6 &&
        neg_res[1] == -3 &&
        neg_res[2] == -1 &&
        pos_res[0] == 2 &&
        pos_res[1] == 3 &&
        pos_res[2] == 45);
}

template <typename TestedFunction>
void test_function4(TestedFunction func_to_test) {
    int a[] = { -6, -5, -4, -3, -2, -1 };
    int pos_res[6] = {};
    int neg_res[6] = {};
    int neg_count;
    auto pos_count = func_to_test(a, 6, pos_res, neg_res, (int*)(&neg_count));

    assert(pos_count == 0);
    assert(neg_count == 6);
    assert(
        neg_res[0] == -6 &&
        neg_res[1] == -5 &&
        neg_res[2] == -4 &&
        neg_res[3] == -3 &&
        neg_res[4] == -2 &&
        neg_res[5] == -1);
}

template <typename TestedFunction>
void test_function5(TestedFunction func_to_test) {
    int a[] = { 1, 2, 3, 4, 5, 6 };
    int pos_res[6] = {};
    int neg_res[6] = {};
    int neg_count;
    auto pos_count = func_to_test(a, 6, pos_res, neg_res, (int*)(&neg_count));

    assert(pos_count == 6);
    assert(neg_count == 0);
    assert(
        pos_res[0] == 1 &&
        pos_res[1] == 2 &&
        pos_res[2] == 3 &&
        pos_res[3] == 4 &&
        pos_res[4] == 5 &&
        pos_res[5] == 6);
}

template <typename TestedFunction>
void test_function6(TestedFunction func_to_test) {
    int a[] = { 6, 3, 4, 2, 1, 5 };
    int pos_res[6] = {};
    int neg_res[6] = {};
```

```
    int neg_count;
    auto pos_count = func_to_test(a, 6, pos_res, neg_res, (int*)(&neg_count));

    assert(pos_count == 6);
    assert(neg_count == 0);
    assert(
        pos_res[0] == 1 &&
        pos_res[1] == 2 &&
        pos_res[2] == 3 &&
        pos_res[3] == 4 &&
        pos_res[4] == 5 &&
        pos_res[5] == 6);
}

template <typename TestedFunction>
void test_function7(TestedFunction func_to_test) {
    int a[] = { -4, -2, -6, -1, -5, -3 };
    int pos_res[6] = {};
    int neg_res[6] = {};
    int neg_count;
    auto pos_count = func_to_test(a, 6, pos_res, neg_res, (int*)(&neg_count));

    assert(pos_count == 0);
    assert(neg_count == 6);
    assert(
        neg_res[0] == -6 &&
        neg_res[1] == -5 &&
        neg_res[2] == -4 &&
        neg_res[3] == -3 &&
        neg_res[4] == -2 &&
        neg_res[5] == -1);
}

template <typename TestedFunction>
void stress_test(TestedFunction func_to_test, int amount) {
    srand(0);
    int *a = (int*)malloc(sizeof(int) * amount);
    int *pos_res = (int*)malloc(sizeof(int) * amount);
    int *neg_res = (int*)malloc(sizeof(int) * amount);
    int neg_count;
    for (int i = 0; i < amount; i++) {
        a[i] = rand() % 1000;
    }

    std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
    auto pos_count = func_to_test(a, amount, pos_res, neg_res, (int*)(&neg_count));
    std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
    auto delta = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count();

    std::cout << "Working time: " << delta / 1000.0 << std::endl;

    free(a);
    free(pos_res);
    free(neg_res);
}

template <typename TestedFunction>
void test_function(TestedFunction func_to_test) {
    test_function1(func_to_test);
    test_function2(func_to_test);
    test_function3(func_to_test);
    test_function4(func_to_test);
    test_function5(func_to_test);
    test_function6(func_to_test);
    test_function7(func_to_test);
    for (int i = 10000; i <= 25000; i += 1000) {
        stress_test(func_to_test, i);
```

```
    }
}

int main() {
    std::cout << "Native function:" << std::endl;
    test_function(sort_native);
    std::cout << "__cdecl auto parameters:" << std::endl;
    test_function(sort_cdecl);
    std::cout << "__stdcall auto parameters:" << std::endl;
    test_function(sort_stdcall);
    std::cout << "__fastcall manual parameters:" << std::endl;
    test_function(sort_fastcall_noarg);
    std::cout << "__stdcall manual parameters" << std::endl;
    test_function(sort_stdcall_noarg);
    std::cout << "__cdecl manual parameters" << std::endl;
    test_function(sort_cdecl_noarg);

    return 0;
}
```

Графики времени выполнения:



Оранжевый – время выполнения для вручную написанного кода, синий – релизная версия функции сортировки, скомпилированная средствами Visual Studio 2022.

Вывод: в ходе лабораторной изучили способы вызова подпрограмм, написанных на разных языках программирования посредством dll-библиотек. В большинстве случаев скомпилированный код будет быстрее и надёжнее кода, написанного вручную.