**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и

автоматизированныхсистем

Лабораторная работа № 8

**Способы вызова ассемблерных подпрограмм**

**в языках высокого уровня**

Выполнил ст. группы КБ-211

Никольская Анастасия Валерьевна

Проверил

Осипов Олег Васильевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Белгород 2023

**Цель работы**: изучение способов вызова подпрограмм, написанных на разных языках программирования посредством dll-библиотек.

**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: подпрограммы в стилях stdcall, cdecl, fastcall, написанные на ассемблере **без** явного перечисления аргументов в заголовке; 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 С++ 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 С++ 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.

Сортировка части массива чисел с индексами от *start* до *end* включительно. Отсортированный массив (элементы *start*…*end*) записать в *res* и возвратить его адрес. Исходный массив *a* оставить без изменений. Под массив *res* зарезервировать память в необходимом размере, но не больше, чем нужно.

Пузырьковая сортировка по не убыванию.

int\* sort(int start, int end, int\* res, int\* a).

**Выполнение**

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.

1) stdcall

sort\_stdcall proc stdcall

    PUSH EBX

    PUSH ECX

    PUSH EDX

    PUSH EDI

    PUSH ESI

    MOV EDX, [ESP+6\*4+4]

    SUB EDX, [ESP+6\*4+0]

    INC EDX                     ; EDX = (end - start + 1)  (размер массива res)

    MOV ESI, [ESP+6\*4+12]       ; ESI = &a

    MOV EAX, [ESP+6\*4+0]        ; EAX = start

    IMUL EAX, 4                 ; EAX = start\*4

    MOV ECX, EDX                ; ECX = EDX  (счетчик)

    ADD ESI, EAX                ; ESI = &a + start\*4

    MOV EDI, [ESP+6\*4+8]        ; EDI = &res

    REP MOVSD                   ; Копирование массива двойных слов

    MOV EDI, [ESP+6\*4+8]        ; EDI = &res

    MOV EBX, 0                  ; EBX = 0  (i)

    loop\_sort\_arr\_outer:

      MOV ECX, 1                ; ECX = 1  (j)

      loop\_sort\_arr\_inner:

        MOV EAX, [EDI+ECX\*4-4]  ; EAX = res[j - 1]

        CMP EAX, [EDI+ECX\*4]

        JLE sort\_arr\_no\_swap    ; Если не (res[j - 1] > res[j]), то пропустить обмен

          MOV ESI, [EDI+ECX\*4]    ; ESI = res[j]

          MOV [EDI+ECX\*4], EAX    ; res[j] = res[j - 1]

          MOV [EDI+ECX\*4-4], ESI  ; res[j - 1] = ESI

        sort\_arr\_no\_swap:

        INC ECX                 ; ECX += 1

        MOV ESI, EDX            ; ESI = size

        SUB ESI, EBX            ; ESI = size - i

        CMP ECX, EDX

        JL loop\_sort\_arr\_inner  ; Цикл пока не дойдем до наибольшего элемента,

                                ; который смещен пузырьком в конец массива res

      loop\_sort\_arr\_inner\_end:

      INC EBX                   ; EBX += 1

      CMP EBX, EDX

      JL loop\_sort\_arr\_outer    ; Цикл пока не пройдем по всем элементам массива res

    loop\_sort\_arr\_outer\_end:

    MOV EAX, [ESP+6\*4+8]        ; EAX = &res

    POP ESI

    POP EDI

    POP EDX

    POP ECX

    POP EBX

    RET 4\*4

  sort\_stdcall endp

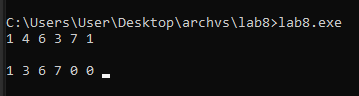
push offset array

    push offset result

    push offset 5

    push offset 2

    call sort\_stdcall



2) cdecl

  sort\_cdecl proc c

    PUSH EBX

    PUSH ECX

    PUSH EDX

    PUSH EDI

    PUSH ESI

    MOV EDX, [ESP+6\*4+4]

    SUB EDX, [ESP+6\*4+0]

    INC EDX                     ; EDX = (end - start + 1)  (размер массива res)

    MOV ESI, [ESP+6\*4+12]       ; ESI = &a

    MOV EAX, [ESP+6\*4+0]        ; EAX = start

    IMUL EAX, 4                 ; EAX = start\*4

    MOV ECX, EDX                ; ECX = EDX  (счетчик)

    ADD ESI, EAX                ; ESI = &a + start\*4

    MOV EDI, [ESP+6\*4+8]        ; EDI = &res

    REP MOVSD                   ; Копирование массива двойных слов

    MOV EDI, [ESP+6\*4+8]        ; EDI = &res

    MOV EBX, 0                  ; EBX = 0  (i)

    loop\_sort\_arr\_outer:

      MOV ECX, 1                ; ECX = 1  (j)

      loop\_sort\_arr\_inner:

        MOV EAX, [EDI+ECX\*4-4]  ; EAX = res[j - 1]

        CMP EAX, [EDI+ECX\*4]

        JLE sort\_arr\_no\_swap    ; Если не (res[j - 1] > res[j]), то пропустить обмен

          MOV ESI, [EDI+ECX\*4]    ; ESI = res[j]

          MOV [EDI+ECX\*4], EAX    ; res[j] = res[j - 1]

          MOV [EDI+ECX\*4-4], ESI  ; res[j - 1] = ESI

        sort\_arr\_no\_swap:

        INC ECX                 ; ECX += 1

        MOV ESI, EDX            ; ESI = size

        SUB ESI, EBX            ; ESI = size - i

        CMP ECX, EDX

        JL loop\_sort\_arr\_inner  ; Цикл пока не дойдем до наибольшего элемента,

                                ; который смещен пузырьком в конец массива res

      loop\_sort\_arr\_inner\_end:

      INC EBX                   ; EBX += 1

      CMP EBX, EDX

      JL loop\_sort\_arr\_outer    ; Цикл пока не пройдем по всем элементам массива res

    loop\_sort\_arr\_outer\_end:

    MOV EAX, [ESP+6\*4+8]        ; EAX = &res

    POP ESI

    POP EDI

    POP EDX

    POP ECX

    POP EBX

    RET

  sort\_cdecl endp

push offset array

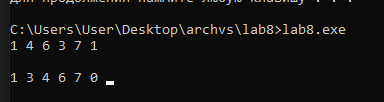
    push offset result

    push offset 4

    push offset 0

    call sort\_cdecl

    add esp, 2\*4



3) fastcall

  sort\_fastcall proc

    PUSH EBX

    PUSH EDI

    PUSH ESI

    ; ECX = start,  EDX = end

    SUB EDX, ECX                ; EDX = end - start

    INC EDX                     ; EDX = (end - start + 1)  (размер массива res)

    MOV ESI, [ESP+4\*4+4]       ; ESI = &a

    IMUL ECX, 4                 ; ECX = start\*4

    ADD ESI, ECX                ; ESI = &a + start\*4

    MOV ECX, EDX                ; ECX = EDX  (счетчик)

    MOV EDI, [ESP+4\*4+0]        ; EDI = &res

    REP MOVSD                   ; Копирование массива двойных слов

    MOV EDI, [ESP+4\*4+0]        ; EDI = &res

    MOV EBX, 0                  ; EBX = 0  (i)

    loop\_sort\_arr\_outer:

      MOV ECX, 1                ; ECX = 1  (j)

      loop\_sort\_arr\_inner:

        MOV EAX, [EDI+ECX\*4-4]  ; EAX = res[j - 1]

        CMP EAX, [EDI+ECX\*4]

        JLE sort\_arr\_no\_swap    ; Если не (res[j - 1] > res[j]), то пропустить обмен

          MOV ESI, [EDI+ECX\*4]    ; ESI = res[j]

          MOV [EDI+ECX\*4], EAX    ; res[j] = res[j - 1]

          MOV [EDI+ECX\*4-4], ESI  ; res[j - 1] = ESI

        sort\_arr\_no\_swap:

        INC ECX                 ; ECX += 1

        MOV ESI, EDX            ; ESI = size

        SUB ESI, EBX            ; ESI = size - i

        CMP ECX, EDX

        JL loop\_sort\_arr\_inner  ; Цикл пока не дойдем до наибольшего элемента,

                                ; который смещен пузырьком в конец массива res

      loop\_sort\_arr\_inner\_end:

      INC EBX                   ; EBX += 1

      CMP EBX, EDX

      JL loop\_sort\_arr\_outer    ; Цикл пока не пройдем по всем элементам массива res

    loop\_sort\_arr\_outer\_end:

    MOV EAX, [ESP+6\*4+8]        ; EAX = &res

    POP ESI

    POP EDI

    POP EBX

    RET 2\*4

  sort\_fastcall endp

    push offset array

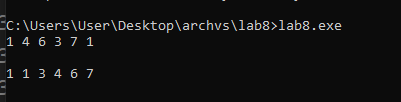
    push offset result

    mov edx, 5

    mov ecx, 0

    call sort\_fastcall

    add esp, 2\*4



2)Подпрограммы собрать и скомпилировать в виде dll-библиотеки. Библиотека должна содержать: подпрограммы в стилях stdcall, cdecl, fastcall, написанные на ассемблере **без** явного перечисления аргументов в заголовке; Подпрограммы в стилях stdcall, cdecl, написанные, наоборот, **с** перечислением аргументов в заголовке подпрограммы.

.386

.model flat, stdcall

option casemap: none

; Компиляция:

;cd C:\Users\User\Desktop\archvs\lab8

;C:\Users\User\Desktop\archvs\masm32\bin\ml /c /coff /I "C:\Users\User\Desktop\archvs\masm32\include" lab8.asm

;C:\Users\User\Desktop\archvs\masm32\bin\link /SUBSYSTEM:WINDOWS /DLL /DEF:lab8.def lab8.obj

.code

  ; Инициализирующая функция

  \_DllMainCRTStartup proc hlnstDLL:DWORD, reason:DWORD, unused:DWORD

    mov EAX, 1

    ret

  \_DllMainCRTStartup endp

  sort\_stdcall proc stdcall

    PUSH EBX

    PUSH ECX

    PUSH EDX

    PUSH EDI

    PUSH ESI

    MOV EDX, [ESP+6\*4+4]

    SUB EDX, [ESP+6\*4+0]

    INC EDX                     ; EDX = (end - start + 1)  (размер массива res)

    MOV ESI, [ESP+6\*4+12]       ; ESI = &a

    MOV EAX, [ESP+6\*4+0]        ; EAX = start

    IMUL EAX, 4                 ; EAX = start\*4

    MOV ECX, EDX                ; ECX = EDX  (счетчик)

    ADD ESI, EAX                ; ESI = &a + start\*4

    MOV EDI, [ESP+6\*4+8]        ; EDI = &res

    REP MOVSD                   ; Копирование массива двойных слов

    MOV EDI, [ESP+6\*4+8]        ; EDI = &res

    MOV EBX, 0                  ; EBX = 0  (i)

    loop\_sort\_arr\_outer:

      MOV ECX, 1                ; ECX = 1  (j)

      loop\_sort\_arr\_inner:

        MOV EAX, [EDI+ECX\*4-4]  ; EAX = res[j - 1]

        CMP EAX, [EDI+ECX\*4]

        JLE sort\_arr\_no\_swap    ; Если не (res[j - 1] > res[j]), то пропустить обмен

          MOV ESI, [EDI+ECX\*4]    ; ESI = res[j]

          MOV [EDI+ECX\*4], EAX    ; res[j] = res[j - 1]

          MOV [EDI+ECX\*4-4], ESI  ; res[j - 1] = ESI

        sort\_arr\_no\_swap:

        INC ECX                 ; ECX += 1

        MOV ESI, EDX            ; ESI = size

        SUB ESI, EBX            ; ESI = size - i

        CMP ECX, EDX

        JL loop\_sort\_arr\_inner  ; Цикл пока не дойдем до наибольшего элемента,

                                ; который смещен пузырьком в конец массива res

      loop\_sort\_arr\_inner\_end:

      INC EBX                   ; EBX += 1

      CMP EBX, EDX

      JL loop\_sort\_arr\_outer    ; Цикл пока не пройдем по всем элементам массива res

    loop\_sort\_arr\_outer\_end:

    MOV EAX, [ESP+6\*4+8]        ; EAX = &res

    POP ESI

    POP EDI

    POP EDX

    POP ECX

    POP EBX

    RET 4\*4

  sort\_stdcall endp

  sort\_cdecl proc c

    PUSH EBX

    PUSH ECX

    PUSH EDX

    PUSH EDI

    PUSH ESI

    MOV EDX, [ESP+6\*4+4]

    SUB EDX, [ESP+6\*4+0]

    INC EDX                     ; EDX = (end - start + 1)  (размер массива res)

    MOV ESI, [ESP+6\*4+12]       ; ESI = &a

    MOV EAX, [ESP+6\*4+0]        ; EAX = start

    IMUL EAX, 4                 ; EAX = start\*4

    MOV ECX, EDX                ; ECX = EDX  (счетчик)

    ADD ESI, EAX                ; ESI = &a + start\*4

    MOV EDI, [ESP+6\*4+8]        ; EDI = &res

    REP MOVSD                   ; Копирование массива двойных слов

    MOV EDI, [ESP+6\*4+8]        ; EDI = &res

    MOV EBX, 0                  ; EBX = 0  (i)

    loop\_sort\_arr\_outer:

      MOV ECX, 1                ; ECX = 1  (j)

      loop\_sort\_arr\_inner:

        MOV EAX, [EDI+ECX\*4-4]  ; EAX = res[j - 1]

        CMP EAX, [EDI+ECX\*4]

        JLE sort\_arr\_no\_swap    ; Если не (res[j - 1] > res[j]), то пропустить обмен

          MOV ESI, [EDI+ECX\*4]    ; ESI = res[j]

          MOV [EDI+ECX\*4], EAX    ; res[j] = res[j - 1]

          MOV [EDI+ECX\*4-4], ESI  ; res[j - 1] = ESI

        sort\_arr\_no\_swap:

        INC ECX                 ; ECX += 1

        MOV ESI, EDX            ; ESI = size

        SUB ESI, EBX            ; ESI = size - i

        CMP ECX, EDX

        JL loop\_sort\_arr\_inner  ; Цикл пока не дойдем до наибольшего элемента,

                                ; который смещен пузырьком в конец массива res

      loop\_sort\_arr\_inner\_end:

      INC EBX                   ; EBX += 1

      CMP EBX, EDX

      JL loop\_sort\_arr\_outer    ; Цикл пока не пройдем по всем элементам массива res

    loop\_sort\_arr\_outer\_end:

    MOV EAX, [ESP+6\*4+8]        ; EAX = &res

    POP ESI

    POP EDI

    POP EDX

    POP ECX

    POP EBX

    RET

  sort\_cdecl endp

  sort\_fastcall proc

    PUSH EBX

    PUSH EDI

    PUSH ESI

    ; ECX = start,  EDX = end

    SUB EDX, ECX                ; EDX = end - start

    INC EDX                     ; EDX = (end - start + 1)  (размер массива res)

    MOV ESI, [ESP+4\*4+4]       ; ESI = &a

    IMUL ECX, 4                 ; ECX = start\*4

    ADD ESI, ECX                ; ESI = &a + start\*4

    MOV ECX, EDX                ; ECX = EDX  (счетчик)

    MOV EDI, [ESP+4\*4+0]        ; EDI = &res

    REP MOVSD                   ; Копирование массива двойных слов

    MOV EDI, [ESP+4\*4+0]        ; EDI = &res

    MOV EBX, 0                  ; EBX = 0  (i)

    loop\_sort\_arr\_outer:

      MOV ECX, 1                ; ECX = 1  (j)

      loop\_sort\_arr\_inner:

        MOV EAX, [EDI+ECX\*4-4]  ; EAX = res[j - 1]

        CMP EAX, [EDI+ECX\*4]

        JLE sort\_arr\_no\_swap    ; Если не (res[j - 1] > res[j]), то пропустить обмен

          MOV ESI, [EDI+ECX\*4]    ; ESI = res[j]

          MOV [EDI+ECX\*4], EAX    ; res[j] = res[j - 1]

          MOV [EDI+ECX\*4-4], ESI  ; res[j - 1] = ESI

        sort\_arr\_no\_swap:

        INC ECX                 ; ECX += 1

        MOV ESI, EDX            ; ESI = size

        SUB ESI, EBX            ; ESI = size - i

        CMP ECX, EDX

        JL loop\_sort\_arr\_inner  ; Цикл пока не дойдем до наибольшего элемента,

                                ; который смещен пузырьком в конец массива res

      loop\_sort\_arr\_inner\_end:

      INC EBX                   ; EBX += 1

      CMP EBX, EDX

      JL loop\_sort\_arr\_outer    ; Цикл пока не пройдем по всем элементам массива res

    loop\_sort\_arr\_outer\_end:

    MOV EAX, [ESP+6\*4+8]        ; EAX = &res

    POP ESI

    POP EDI

    POP EBX

    RET 2\*4

  sort\_fastcall endp

  sort\_stdcall\_args proc stdcall start\_:DWORD, end\_:DWORD, res\_:DWORD, a\_:DWORD

    PUSH EBX

    PUSH ECX

    PUSH EDX

    PUSH EDI

    PUSH ESI

    MOV EDX, end\_

    SUB EDX, start\_

    INC EDX                     ; EDX = (end - start + 1)  (размер массива res)

    MOV ESI, a\_                 ; ESI = &a

    MOV EAX, start\_             ; EAX = start

    IMUL EAX, 4                 ; EAX = start\*4

    MOV ECX, EDX                ; ECX = EDX  (счетчик)

    ADD ESI, EAX                ; ESI = &a + start\*4

    MOV EDI, res\_               ; EDI = &res

    REP MOVSD                   ; Копирование массива двойных слов

    MOV EDI, res\_               ; EDI = &res

    MOV EBX, 0                  ; EBX = 0  (i)

    loop\_sort\_arr\_outer:

      MOV ECX, 1                ; ECX = 1  (j)

      loop\_sort\_arr\_inner:

        MOV EAX, [EDI+ECX\*4-4]  ; EAX = res[j - 1]

        CMP EAX, [EDI+ECX\*4]

        JLE sort\_arr\_no\_swap    ; Если не (res[j - 1] > res[j]), то пропустить обмен

          MOV ESI, [EDI+ECX\*4]    ; ESI = res[j]

          MOV [EDI+ECX\*4], EAX    ; res[j] = res[j - 1]

          MOV [EDI+ECX\*4-4], ESI  ; res[j - 1] = ESI

        sort\_arr\_no\_swap:

        INC ECX                 ; ECX += 1

        MOV ESI, EDX            ; ESI = size

        SUB ESI, EBX            ; ESI = size - i

        CMP ECX, EDX

        JL loop\_sort\_arr\_inner  ; Цикл пока не дойдем до наибольшего элемента,

                                ; который смещен пузырьком в конец массива res

      loop\_sort\_arr\_inner\_end:

      INC EBX                   ; EBX += 1

      CMP EBX, EDX

      JL loop\_sort\_arr\_outer    ; Цикл пока не пройдем по всем элементам массива res

    loop\_sort\_arr\_outer\_end:

    MOV EAX, res\_               ; EAX = &res

    POP ESI

    POP EDI

    POP EDX

    POP ECX

    POP EBX

    RET 4\*4

  sort\_stdcall\_args endp

  sort\_cdecl\_args proc c start\_:DWORD, end\_:DWORD, res\_:DWORD, a\_:DWORD

    PUSH EBX

    PUSH ECX

    PUSH EDX

    PUSH EDI

    PUSH ESI

    MOV EDX, end\_

    SUB EDX, start\_

    INC EDX                     ; EDX = (end - start + 1)  (размер массива res)

    MOV ESI, a\_                 ; ESI = &a

    MOV EAX, start\_             ; EAX = start

    IMUL EAX, 4                 ; EAX = start\*4

    MOV ECX, EDX                ; ECX = EDX  (счетчик)

    ADD ESI, EAX                ; ESI = &a + start\*4

    MOV EDI, res\_               ; EDI = &res

    REP MOVSD                   ; Копирование массива двойных слов

    MOV EDI, res\_               ; EDI = &res

    MOV EBX, 0                  ; EBX = 0  (i)

    loop\_sort\_arr\_outer:

      MOV ECX, 1                ; ECX = 1  (j)

      loop\_sort\_arr\_inner:

        MOV EAX, [EDI+ECX\*4-4]  ; EAX = res[j - 1]

        CMP EAX, [EDI+ECX\*4]

        JLE sort\_arr\_no\_swap    ; Если не (res[j - 1] > res[j]), то пропустить обмен

          MOV ESI, [EDI+ECX\*4]    ; ESI = res[j]

          MOV [EDI+ECX\*4], EAX    ; res[j] = res[j - 1]

          MOV [EDI+ECX\*4-4], ESI  ; res[j - 1] = ESI

        sort\_arr\_no\_swap:

        INC ECX                 ; ECX += 1

        MOV ESI, EDX            ; ESI = size

        SUB ESI, EBX            ; ESI = size - i

        CMP ECX, EDX

        JL loop\_sort\_arr\_inner  ; Цикл пока не дойдем до наибольшего элемента,

                                ; который смещен пузырьком в конец массива res

      loop\_sort\_arr\_inner\_end:

      INC EBX                   ; EBX += 1

      CMP EBX, EDX

      JL loop\_sort\_arr\_outer    ; Цикл пока не пройдем по всем элементам массива res

    loop\_sort\_arr\_outer\_end:

    MOV EAX, res\_               ; EAX = &res

    POP ESI

    POP EDI

    POP EDX

    POP ECX

    POP EBX

    RET

  sort\_cdecl\_args endp

end

lab8.def

LIBRARY lab8

EXPORTS

\_sort\_stdcall@16 = \_sort\_stdcall@0

sort\_cdecl@16 = sort\_cdecl

@sort\_fastcall@16 = \_sort\_fastcall@0

sort\_stdcall\_args

sort\_cdecl\_args

3)Подключить все подпрограммы из dll-библиотеки к проектам на C# и С++ статическим и динамическим способом. Убедиться в правильности вызова всех подпрограмм.

Статический способ:

#pragma once

#pragma comment(lib,"lab8.lib")

extern "C" \_\_declspec(dllimport) int\* \_stdcall sort\_stdcall(int start, int end, int\* res, const int\* a);

extern "C" \_\_declspec(dllimport) int\* \_cdecl sort\_cdecl(int start, int end, int\* res, const int\* a);

extern "C" \_\_declspec(dllimport) int\* \_fastcall sort\_fastcall(int start, int end, int\* res, const int\* a);

extern "C" \_\_declspec(dllimport) int\* \_stdcall sort\_stdcall\_args(int start, int end, int\* res, const int\* a);

extern "C" \_\_declspec(dllimport) int\* \_cdecl sort\_cdecl\_args(int start, int end, int\* res, const int\* a);

Динамический способ:

#include <iostream>

#include <cstdio>

#include <windows.h>

typedef void(\_cdecl\* func)(int start, int end, int\* res, const int\* a);

int main()

{

setlocale(LC\_ALL, "rus");

char dll\_name[] = "lab8.dll"; // Полный путь к библиотеке

HMODULE hModule = LoadLibraryA(dll\_name);

// Если dll-файл найден

if (hModule != NULL)

{

char func\_name[] = "sort\_cdecl\_args";

// Получить указатель на подпрограмму

func f = (func)GetProcAddress(hModule, func\_name);

if (f == NULL)

printf("Подпрограмма %s не найдена в библиотеке", func\_name);

else

{

int a[] = { 1, 2, 3, 4, 5, 5, 4, 3, 2, 1 }, res[10];

// Вызов подпрограммы

f(2,5,res,a);

}

// Освобождение библиотеки

FreeLibrary(hModule);

}

else

printf("dll-файл %s не найден", dll\_name);

}

4)Написать подпрограмму для решения задачи варианта с использованием ассемблерной вставки на языке C++.

/\* Запись в массив res отсортированнного по не убыванию массива a на отрезке [start..end] \*/

int\* sort\_asm(int start\_, int end\_, int\* res\_, const int\* a\_)

{

\_\_asm

{

PUSH EBX

PUSH ECX

PUSH EDX

PUSH EDI

PUSH ESI

MOV EDX, end\_

SUB EDX, start\_

INC EDX; EDX = (end - start + 1) (размер массива res)

MOV ESI, a\_; ESI = &a

MOV EAX, start\_; EAX = start

IMUL EAX, 4; EAX = start \* 4

MOV ECX, EDX; ECX = EDX(счетчик)

ADD ESI, EAX; ESI = &a + start \* 4

MOV EDI, res\_; EDI = &res

REP MOVSD; Копирование массива двойных слов

MOV EDI, res\_; EDI = &res

MOV EBX, 0; EBX = 0 (i)

loop\_sort\_arr\_outer:

MOV ECX, 1; ECX = 1 (j)

loop\_sort\_arr\_inner:

MOV EAX, [EDI + ECX \* 4 - 4]; EAX = res[j - 1]

CMP EAX, [EDI + ECX \* 4]

JLE sort\_arr\_no\_swap; Если не(res[j - 1] > res[j]), то пропустить обмен

MOV ESI, [EDI + ECX \* 4]; ESI = res[j]

MOV[EDI + ECX \* 4], EAX; res[j] = res[j - 1]

MOV[EDI + ECX \* 4 - 4], ESI; res[j - 1] = ESI

sort\_arr\_no\_swap :

INC ECX; ECX += 1

MOV ESI, EDX; ESI = size

SUB ESI, EBX; ESI = size - i

CMP ECX, EDX

JL loop\_sort\_arr\_inner; Цикл пока не дойдем до наибольшего элемента,

; который смещен пузырьком в конец массива res

loop\_sort\_arr\_inner\_end :

INC EBX; EBX += 1

CMP EBX, EDX

JL loop\_sort\_arr\_outer; Цикл пока не пройдем по всем элементам массива res

loop\_sort\_arr\_outer\_end :

MOV EAX, res\_; EAX = &res

POP ESI

POP EDI

POP EDX

POP ECX

POP EBX

}

return res\_;

}

5)Написать подпрограммы для решения задачи варианта с использованием обычного высокоуровнего языка C# и C++ (или любого другого).

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <windows.h>

#define COUNT\_ATTEMPTS 1

#define SIZE\_ARR 10

#define SORT\_START 1

#define SORT\_END 8

/\* Заполняет массив a размера n случайными числами на отрезке [min..max] \*/

void set\_rand\_arr(int\* a, int n, int min, int max) {

for (int i = 0; i < n; i++) {

a[i] = rand() % (max - min + 1) + min;

}

}

/\* Вывод массива целых чисел a размера n \*/

void output(int\* a, int n) {

for (int i = 0; i < n; i++) {

std::cout << a[i] << ' ';

}

std::cout << '\n';

}

/\* Запись в массив res отсортированнного по не убыванию массива a на отрезке [start..end] \*/

int\* sort(int start, int end, int\* res, const int\* a)

{

int size = end - start + 1;

for (int i = 0; i < size; i++) {

res[i] = a[i + start];

}

for (int i = 0; i < size; i++) {

for (int j = 1; j < size - i; j++) {

if (res[j - 1] > res[j]) {

int t = res[j];

res[j] = res[j - 1];

res[j - 1] = t;

}

}

}

return res;

}

int main()

{

setlocale(LC\_ALL, "rus");

int a[SIZE\_ARR];

int res[SIZE\_ARR];

memset(res, 0, SIZE\_ARR \* sizeof(int));

for (int i = 0; i < COUNT\_ATTEMPTS; i++) {

set\_rand\_arr(a, SIZE\_ARR, 1, 100);

sort(SORT\_START, SORT\_END, res, a);

output(a, SIZE\_ARR);

output(res, SORT\_END - SORT\_START);

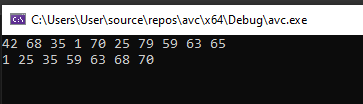
}

char ch;

std::cin >> ch;

return 0;

}



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 С++ 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.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <cstdio>

#include <ctime>

#include <windows.h>

#include "Header.h"

#define COUNT\_ATTEMPTS 100000

#define SIZE\_ARR\_MIN 10

#define SIZE\_ARR\_MAX 100

#define COUNT\_POINTS 10

#define STEP\_SIZE ((SIZE\_ARR\_MAX - SIZE\_ARR\_MIN) / (COUNT\_POINTS - 1))

/\* Заполняет массив a размера n случайными числами на отрезке [min..max] \*/

void set\_rand\_arr(int\* a, int n, int min, int max) {

for (int i = 0; i < n; i++) {

a[i] = rand() % (max - min + 1) + min;

}

}

/\* Вывод массива целых чисел a размера n \*/

void output(int\* a, int n) {

for (int i = 0; i < n; i++) {

printf("%d ", a[i]);

}

putchar('\n');

}

/\* Запись в массив res отсортированнного по не убыванию массива a на отрезке [start..end] \*/

int\* sort(int start, int end, int\* res, const int\* a)

{

int size = end - start + 1;

for (int i = 0; i < size; i++) {

res[i] = a[i + start];

}

for (int i = 0; i < size; i++) {

for (int j = 1; j < size - i; j++) {

if (res[j - 1] > res[j]) {

int t = res[j];

res[j] = res[j - 1];

res[j - 1] = t;

}

}

}

return res;

}

/\* Запись в массив res отсортированнного по не убыванию массива a на отрезке [start..end] \*/

int\* sort\_asm(int start\_, int end\_, int\* res\_, const int\* a\_)

{

\_\_asm

{

PUSH EBX

PUSH ECX

PUSH EDX

PUSH EDI

PUSH ESI

MOV EDX, end\_

SUB EDX, start\_

INC EDX; EDX = (end - start + 1) (размер массива res)

MOV ESI, a\_; ESI = &a

MOV EAX, start\_; EAX = start

IMUL EAX, 4; EAX = start \* 4

MOV ECX, EDX; ECX = EDX(счетчик)

ADD ESI, EAX; ESI = &a + start \* 4

MOV EDI, res\_; EDI = &res

REP MOVSD; Копирование массива двойных слов

MOV EDI, res\_; EDI = &res

MOV EBX, 0; EBX = 0 (i)

loop\_sort\_arr\_outer:

MOV ECX, 1; ECX = 1 (j)

loop\_sort\_arr\_inner:

MOV EAX, [EDI + ECX \* 4 - 4]; EAX = res[j - 1]

CMP EAX, [EDI + ECX \* 4]

JLE sort\_arr\_no\_swap; Если не(res[j - 1] > res[j]), то пропустить обмен

MOV ESI, [EDI + ECX \* 4]; ESI = res[j]

MOV[EDI + ECX \* 4], EAX; res[j] = res[j - 1]

MOV[EDI + ECX \* 4 - 4], ESI; res[j - 1] = ESI

sort\_arr\_no\_swap :

INC ECX; ECX += 1

MOV ESI, EDX; ESI = size

SUB ESI, EBX; ESI = size - i

CMP ECX, EDX

JL loop\_sort\_arr\_inner; Цикл пока не дойдем до наибольшего элемента,

; который смещен пузырьком в конец массива res

loop\_sort\_arr\_inner\_end :

INC EBX; EBX += 1

CMP EBX, EDX

JL loop\_sort\_arr\_outer; Цикл пока не пройдем по всем элементам массива res

loop\_sort\_arr\_outer\_end :

MOV EAX, res\_; EAX = &res

POP ESI

POP EDI

POP EDX

POP ECX

POP EBX

}

return res\_;

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(NULL));

DWORD timeStart, timeEnd;

int a[SIZE\_ARR\_MAX];

int res[SIZE\_ARR\_MAX];

char strSpace[25];

printf("Сортировка на языке C++:\n");

printf("Количество сортировок: \t%d\n", COUNT\_ATTEMPTS);

printf("Размер массива: \tот %d до %d\n", SIZE\_ARR\_MIN, SIZE\_ARR\_MAX);

printf("Количество точек: \t%d\n", COUNT\_POINTS);

for (int size = SIZE\_ARR\_MIN; size <= SIZE\_ARR\_MAX; size += (int)STEP\_SIZE) {

printf("Размер массива: %d\n", size);

timeStart = GetTickCount64();

for (int i = 0; i < COUNT\_ATTEMPTS; i++) {

set\_rand\_arr(a, size, 1, 100);

sort(0, size-1, res, a);

}

timeEnd = GetTickCount64();

printf("{time = %d ms}\n",timeEnd - timeStart);

putchar('\n');

}

printf("\nСортировка с ассемблерной вставкой:\n");

printf("Количество сортировок: \t%d\n", COUNT\_ATTEMPTS);

printf("Размер массива: \tот %d до %d\n", SIZE\_ARR\_MIN, SIZE\_ARR\_MAX);

printf("Количество точек: \t%d\n", COUNT\_POINTS);

for (int size = SIZE\_ARR\_MIN; size <= SIZE\_ARR\_MAX; size += (int)STEP\_SIZE) {

printf("Размер массива: %d\n", size);

timeStart = GetTickCount64();

for (int i = 0; i < COUNT\_ATTEMPTS; i++) {

set\_rand\_arr(a, size, 1, 100);

sort\_asm(0, size - 1, res, a);

}

timeEnd = GetTickCount64();

printf("{time = %d ms}\n", timeEnd - timeStart);

putchar('\n');

}

printf("\nПодпрограмма на asembler, вызываемая C++:\n");

printf("Количество сортировок: \t%d\n", COUNT\_ATTEMPTS);

printf("Размер массива: \tот %d до %d\n", SIZE\_ARR\_MIN, SIZE\_ARR\_MAX);

printf("Количество точек: \t%d\n", COUNT\_POINTS);

for (int size = SIZE\_ARR\_MIN; size <= SIZE\_ARR\_MAX; size += (int)STEP\_SIZE) {

printf("Размер массива: %d\n", size);

timeStart = GetTickCount64();

for (int i = 0; i < COUNT\_ATTEMPTS; i++) {

set\_rand\_arr(a, size, 1, 100);

sort\_cdecl\_args(0, size - 1, res, a);

}

timeEnd = GetTickCount64();

printf("{time = %d ms}\n", timeEnd - timeStart);

putchar('\n');

}

return 0;

}

Header.h

#pragma once

#pragma comment(lib,"lab8.lib")

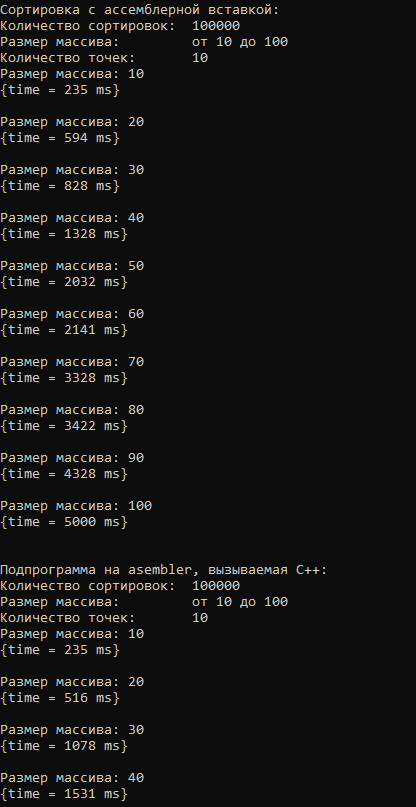
extern "C" \_\_declspec(dllimport) int\* \_stdcall sort\_stdcall(int start, int end, int\* res, const int\* a);

extern "C" \_\_declspec(dllimport) int\* \_cdecl sort\_cdecl(int start, int end, int\* res, const int\* a);

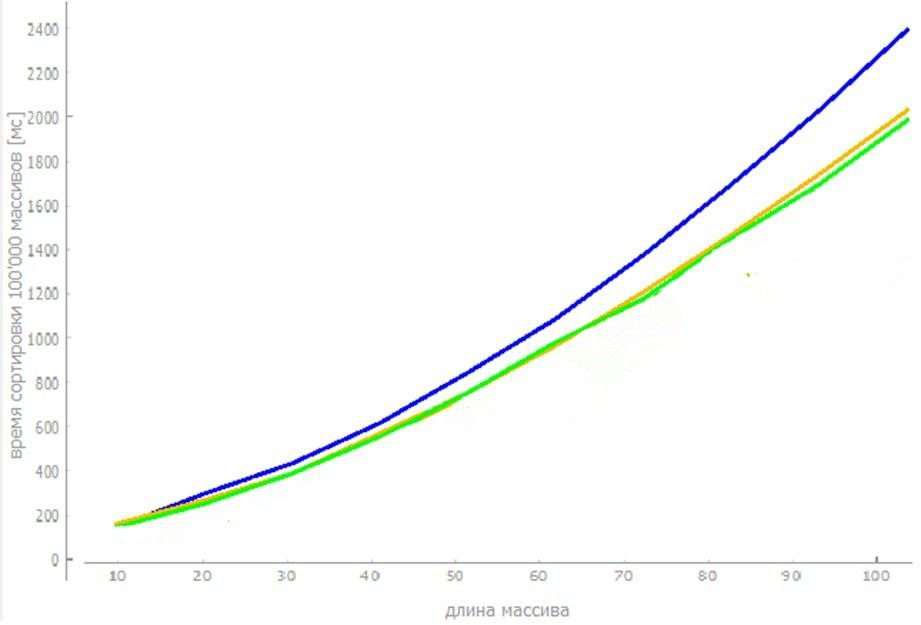
extern "C" \_\_declspec(dllimport) int\* \_fastcall sort\_fastcall(int start, int end, int\* res, const int\* a);

extern "C" \_\_declspec(dllimport) int\* \_stdcall sort\_stdcall\_args(int start, int end, int\* res, const int\* a);

extern "C" \_\_declspec(dllimport) int\* \_cdecl sort\_cdecl\_args(int start, int end, int\* res, const int\* a);



7)В отчёт включить весь исходный код и графики.



* Синий – подпрограмма С++
* Оранжевый – подпрограмма С++ на ассемблерных вставках
* Зеленый – подпрограмма assembler (стиль вызова cdecl)

8)Сделать выводы по работе.

В ходе лабораторной работы были изучены способы вызова подпрограмм, написанных на разных языках программирования посредством dll-библиотек. Программы без использования кода на ассемблере работают медленнее, чем с его использованием при больших объемах вычислений. Использование стиля вызова cdecl по сравнению с другими работает быстрее.

## Реализация на x64 архитектуре

Код программы на языке С++:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <ctime>

#include <windows.h>

#define PRINT\_ARRAYS

#define COUNT\_ATTEMPTS 1

#define SIZE\_ARR\_MIN 10

#define SIZE\_ARR\_MAX 20

#define COUNT\_POINTS 3

#define STEP\_SIZE ((SIZE\_ARR\_MAX - SIZE\_ARR\_MIN) / (COUNT\_POINTS - 1))

extern int printf(char const\* const \_Format, ...);

// Стиль вызова - fastcall

extern "C" long long\* sort(int start, int end, long long\* res, const long long\* a);

// Стиль вызова - fastcall

extern "C" void printArray(long long\* elements, int count);

/\* Заполняет массив a размера n случайными числами на отрезке [min..max] \*/

void setRandArr(long long\* a, int n, int min, int max) {

for (int i = 0; i < n; i++) {

a[i] = rand() % (max - min + 1) + min;

}

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(NULL));

DWORD timeStart, timeEnd;

long long a[SIZE\_ARR\_MAX];

long long res[SIZE\_ARR\_MAX];

for (int size = SIZE\_ARR\_MIN; size <= SIZE\_ARR\_MAX; size += (int)STEP\_SIZE) {

printf("Размер массива: %d\n", size);

timeStart = GetTickCount64();

for (int i = 0; i < COUNT\_ATTEMPTS; i++) {

setRandArr(a, size, 1, 100);

printf("Массив ДО сортровки:\n");

printArray(a, size);

sort(0, size - 1, res, a);

printf("Массив ПОСЛЕ сортровки:\n");

printArray(res, size);

}

timeEnd = GetTickCount64();

printf("{time = %d ms}\n", timeEnd - timeStart);

}

return 0;

}

Код библиотеки на языке assembler:

EXTERN printf: PROC

.DATA

fmt\_int db "%d ", 0

fmt\_new\_line db 13, 10, 0

.CODE

; long long\* sort(int start, int end, long long\* res, const long long\* a)

; Запись в массив res отсортированнного по не убыванию массива a на отрезке [start..end]

sort proc

PUSH RBX

PUSH RDI

PUSH RSI

; RCX = start, RDX = end, R8 = res, R9 = a

SUB RDX, RCX ; RDX = end - start

INC RDX ; RDX = (end - start + 1) (размер массива res)

MOV RSI, R9 ; RSI = &a

IMUL RCX, 8 ; RCX = start\*8

ADD RSI, RCX ; RSI = &a + start\*8

MOV RCX, RDX ; RCX = RDX (счетчик)

IMUL RCX, 2 ; RCX \*= 2 (так как базовый тип массив учетверенное слово, а пересылка для двойного)

MOV RDI, R8 ; RDI = &res

REP MOVSD ; Копирование массива двойных слов

MOV RDI, R8 ; RDI = &res

MOV RBX, 0 ; RBX = 0 (i)

loop\_sort\_arr\_outer:

MOV RCX, 1 ; RCX = 1 (j)

loop\_sort\_arr\_inner:

MOV RAX, [RDI+RCX\*8-8] ; RAX = res[j - 1]

CMP RAX, [RDI+RCX\*8]

JLE sort\_arr\_no\_swap ; Если не (res[j - 1] > res[j]), то пропустить обмен

MOV RSI, [RDI+RCX\*8] ; RSI = res[j]

MOV [RDI+RCX\*8], RAX ; res[j] = res[j - 1]

MOV [RDI+RCX\*8-8], RSI ; res[j - 1] = RSI

sort\_arr\_no\_swap:

INC RCX ; RCX += 1

MOV RSI, RDX ; RSI = size

SUB RSI, RBX ; RSI = size - i

CMP RCX, RDX

JL loop\_sort\_arr\_inner ; Цикл пока не дойдем до наибольшего элемента,

; который смещен пузырьком в конец массива res

loop\_sort\_arr\_inner\_end:

INC RBX ; RBX += 1

CMP RBX, RDX

JL loop\_sort\_arr\_outer ; Цикл пока не пройдем по всем элементам массива res

loop\_sort\_arr\_outer\_end:

MOV RAX, R8 ; RAX = &res

POP RSI

POP RDI

POP RBX

RET

sort endp

; void printArray(long long\* a, int size)

printArray PROC

; В RDX хранится количество элементов в массиве

; В RCX - указатель на массив

PUSH RSI

MOV RSI, RCX ; RSI = &a

MOV ECX, EDX ; ECX = size

print\_loop:

JECXZ exit\_print ; Если ECX = 0, то конец алгоритма

PUSH RCX ; Сохраним RCX в стеке, потому что функция printf изменяет его

SUB RSP, 5\*8 ; Выделение памяти для теневого хранилища

MOV RCX, offset fmt\_int

MOV RDX, [RSI]

call printf ; printf("%d ", RDX)

ADD RSP, 5\*8

POP RCX

ADD RSI, 8 ; Переход к следующему элементу в массиве

DEC ECX ; Уменьшение количества оставшихся элементов

JMP print\_loop

exit\_print:

SUB RSP, 5\*8 ; Выделение памяти для теневого хранилища

MOV RCX, offset fmt\_new\_line

call printf ; printf("\n")

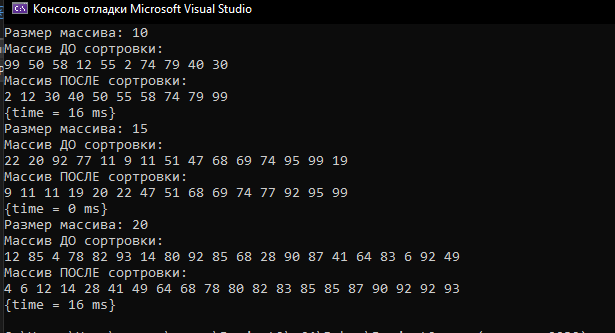
ADD RSP, 5\*8

POP RSI

RET

printArray ENDP

END

****

Динамический способ в проекте ConsoleApplication3

6 задание в проекте ConsoleApplication2

64 разрядный в проекте Project2