МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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 автоматизированных систем

Лабораторная работа №8

по дисциплине: Архитектура вычислительных систем

тема: «Способы вызова ассемблерных подпрограмм в языках высокого уровня»

Выполнил: ст. группы ПВ-31 Донцов А.А.

Проверил: Осипов О.В.

Белгород 2019 г.

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

**Задание:**

⦁ 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 ее в стеке.

⦁ 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.

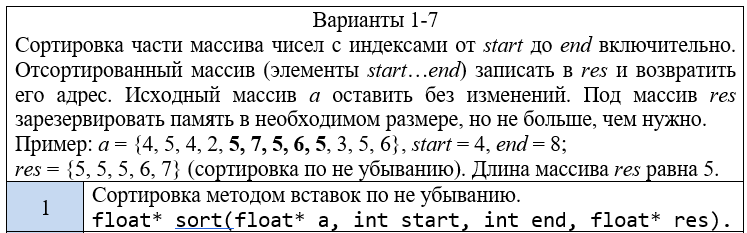
⦁ Подключить все подпрограммы из dll-библиотеки к проектам на C# и С++ статическим и динамическим способом. Убедиться в правильности вызова всех подпрограмм.

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

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

⦁ 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.

⦁ В отчет включить весь исходный код и графики.



**Выполнение**

.486

.model flat, stdcall

option casemap: none

include c:\masm32\include\kernel32.inc

include c:\masm32\include\msvcrt.inc

includelib c:\masm32\lib\kernel32.lib

includelib c:\masm32\lib\msvcrt.lib

.data

str1 dd 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

res dd 100 dup(?)

first dd 0

fend dd 19

format db "%d ", 0

.code

;DllMain proc hlnstDLL:DWORD, reason:DWORD, unused:DWORD

; mov EAX, 1

; ret

;DllMain Endp

sort1 proc stdcall ; float\* sort(float \*a, int start, int end, float \*res);

PUSH EAX

PUSH EDX

PUSH ESI

PUSH ECX

PUSH EDI

PUSH EBX

MOV EAX, [ESP + 36] ; EAX = end

SUB EAX, [ESP + 32] ; EAX = end - start

INC EAX ; EAX += 1

MOV ECX, EAX ; ECX = end - start + 1

MOV EBX, ECX

MOV ESI, [ESP + 32] ; ESI = start

MOV EAX, 4

MUL ESI

MOV ESI, EAX

MOV EAX, 4

MUL ECX

PUSH EAX

MOV EAX, [ESP + 32] ; EAX = a

ADD EAX, ESI

PUSH EAX ; push a

PUSH [ESP + 48] ; push res

call memcopy32

MOV EAX, [ESP + 40] ; EAX = начало массива

sort1j1: ; первый цикл

jmp sort1j2 ; Установка EAX на нужное место

sort1j3: ; возврат из sortj2

ADD EAX, 4 ; подготовка к итерации

loop sort1j1 ; итерация закончена уменьшение ECX

jmp sort1end ; прыжок на выход

sort1j2: ; второй счетчик

MOV EDX, EBX ; счетчик итераций для второго цикла

SUB EDX, ECX ; подготовка 2 счетчика

CMP EDX, 0

JE sort1j3

MOV ESI, EAX ; сохранение индекса для первого цикла

sort1j4:

MOV EDI, ESI ; создание второй переменной, для сравнения

SUB EDI, 4 ; EDI = вторая переменная

PUSH EAX

PUSH EBX

MOV EAX, dword ptr[ESI]

MOV EBX, dword ptr[EDI]

CMP EAX, EBX ; сравнение по указателю

POP EBX

POP EAX

JB sort1j5 ; прыжок на swap

jmp sort1j3 ; выход из итерации 2го счетчика

sort1j5:

PUSH ESI

PUSH EDI

call swap

DEC EDX

SUB ESI, 4

CMP EDX, 0

je sort1j3 ; выход и итерации 2го счетчика

jmp sort1j4 ; начать итерацию

sort1end:

POP EBX

POP EDI

POP ECX

POP ESI

POP EDX

POP EAX

MOV EAX, [ESP + 16]

ret 16

sort1 endp

sort2 proc cdecl ; float\* sort(float \*a, int start, int end, float \*res);

PUSH EAX

PUSH EDX

PUSH ESI

PUSH ECX

PUSH EDI

PUSH EBX

MOV EAX, [ESP + 40] ; EAX = end

SUB EAX, [ESP + 36] ; EAX = end - start

INC EAX ; EAX += 1

MOV ECX, EAX ; ECX = end - start + 1

MOV EBX, EAX

MOV ESI, [ESP + 36] ; ESI = start

MOV EAX, 4

MUL ESI

MOV ESI, EAX

MOV EAX, 4

MUL ECX

PUSH EAX

MOV EAX, [ESP + 36] ; EAX = a

ADD EAX, ESI

PUSH EAX ; push a

PUSH [ESP + 52] ; push res

call memcopy32

MOV EAX, [ESP + 44] ; EAX = начало массива

sort2j1: ; первый цикл

jmp sort2j2 ; Установка EAX на нужное место

sort2j3: ; возврат из sortj2

ADD EAX, 4 ; подготовка к итерации

loop sort2j1 ; итерация закончена уменьшение ECX

jmp sort2end ; прыжок на выход

sort2j2: ; второй счетчик

MOV EDX, EBX ; счетчик итераций для второго цикла

SUB EDX, ECX ; подготовка 2 счетчика

CMP EDX, 0

JE sort2j3

MOV ESI, EAX ; сохранение индекса для первого цикла

sort2j4:

MOV EDI, ESI ; создание второй переменной, для сравнения

SUB EDI, 4 ; EDI = вторая переменная

PUSH EAX

PUSH EBX

MOV EAX, dword ptr[ESI]

MOV EBX, dword ptr[EDI]

CMP EAX, EBX ; сравнение по указателю

POP EBX

POP EAX

JB sort2j5 ; прыжок на swap

jmp sort2j3 ; выход из итерации 2го счетчика

sort2j5:

PUSH ESI

PUSH EDI

call swap

DEC EDX

SUB ESI, 4

CMP EDX, 0

je sort2j3 ; выход и итерации 2го счетчика

jmp sort2j4 ; начать итерацию

sort2end:

POP EBX

POP EDI

POP ECX

POP ESI

POP EDX

POP EAX

MOV EAX, [ESP + 20]

ret

sort2 endp

sort3 proc fastcall ; float\* sort(float \*a, int start, int end, float \*res); ECX, EDX, стек

PUSH EAX

PUSH EDX

PUSH ESI

PUSH ECX

PUSH EDI

PUSH EBP

PUSH EBX

SUB ESP, 16

MOV EBP, ESP

MOV dword ptr[EBP], ECX ; [EBP] = \*a

MOV dword ptr[EBP+4], EDX ; [EBP + 4] = start

MOV ECX, dword ptr[ESP + 13\*4]

MOV dword ptr[EBP + 8], ECX ; [EBP + 8] = end

MOV ECX, dword ptr[ESP + 14\*4]

MOV dword ptr[EBP + 12], ECX ; [EBP + 12] = \*res

MOV EAX, [EBP + 8] ; EAX = end

SUB EAX, [EBP + 4] ; EAX = end - start

INC EAX ; EAX += 1

MOV ECX, EAX ; ECX = end - start + 1

MOV EBX, EAX

MOV ESI, [EBP + 4] ; ESI = start

MOV EAX, 4

MUL ESI

MOV ESI, EAX

MOV EAX, 4

MUL ECX

PUSH EAX

MOV EAX, [EBP] ; EAX = a

ADD EAX, ESI

PUSH EAX ; push a

PUSH [EBP + 12] ; push res

call memcopy32

MOV EAX, [EBP + 12] ; EAX = начало массива

sort3j1: ; первый цикл

jmp sort3j2 ; Установка EAX на нужное место

sort3j3: ; возврат из sortj2

ADD EAX, 4 ; подготовка к итерации

loop sort3j1 ; итерация закончена уменьшение ECX

jmp sort3end ; прыжок на выход

sort3j2: ; второй счетчик

MOV EDX, EBX ; счетчик итераций для второго цикла

SUB EDX, ECX ; подготовка 2 счетчика

CMP EDX, 0

JE sort3j3

MOV ESI, EAX ; сохранение индекса для первого цикла

sort3j4:

MOV EDI, ESI ; создание второй переменной, для сравнения

SUB EDI, 4 ; EDI = вторая переменная

PUSH EAX

PUSH EBX

MOV EAX, dword ptr[ESI]

MOV EBX, dword ptr[EDI]

CMP EAX, EBX ; сравнение по указателю

POP EBX

POP EAX

JB sort3j5 ; прыжок на swap

jmp sort3j3 ; выход из итерации 2го счетчика

sort3j5:

PUSH ESI

PUSH EDI

call swap

DEC EDX

SUB ESI, 4

CMP EDX, 0

je sort3j3 ; выход и итерации 2го счетчика

jmp sort3j4 ; начать итерацию

sort3end:

ADD ESP, 16

POP EBX

POP EBP

POP EDI

POP ECX

POP ESI

POP EDX

POP EAX

MOV EAX, [ESP + 12]

ret 8

sort3 endp

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

sort4 proc stdcall a: DWORD, start: DWORD, endt: DWORD, rest: DWORD

PUSH EAX

PUSH EDX

PUSH ESI

PUSH ECX

PUSH EDI

PUSH EBX

MOV EAX, endt ; EAX = end

SUB EAX, start ; EAX = end - start

INC EAX ; EAX += 1

MOV ECX, EAX ; ECX = end - start + 1

MOV EBX, EAX

MOV ESI, start ; ESI = start

MOV EAX, 4

MUL ESI

MOV ESI, EAX

MOV EAX, 4

MUL ECX

PUSH EAX

MOV EAX, a ; EAX = a

ADD EAX, ESI

PUSH EAX ; push a

PUSH rest ; push res

call memcopy32

MOV EAX, rest ; EAX = начало массива

sort4j1: ; первый цикл

jmp sort4j2 ; Установка EAX на нужное место

sort4j3: ; возврат из sortj2

ADD EAX, 4 ; подготовка к итерации

loop sort4j1 ; итерация закончена уменьшение ECX

jmp sort4end ; прыжок на выход

sort4j2: ; второй счетчик

MOV EDX, EBX ; счетчик итераций для второго цикла

SUB EDX, ECX ; подготовка 2 счетчика

CMP EDX, 0

JE sort4j3

MOV ESI, EAX ; сохранение индекса для первого цикла

sort4j4:

MOV EDI, ESI ; создание второй переменной, для сравнения

SUB EDI, 4 ; EDI = вторая переменная

PUSH EAX

PUSH EBX

MOV EAX, dword ptr[ESI]

MOV EBX, dword ptr[EDI]

CMP EAX, EBX ; сравнение по указателю

POP EBX

POP EAX

JB sort4j5 ; прыжок на swap

jmp sort4j3 ; выход из итерации 2го счетчика

sort4j5:

PUSH ESI

PUSH EDI

call swap

DEC EDX

SUB ESI, 4

CMP EDX, 0

je sort4j3 ; выход и итерации 2го счетчика

jmp sort4j4 ; начать итерацию

sort4end:

POP EBX

POP EDI

POP ECX

POP ESI

POP EDX

POP EAX

MOV EAX, [ESP + 16]

ret 16

sort4 endp

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

sort5 proc c a: DWORD, start: DWORD, endt: DWORD, rest: DWORD

PUSH EAX

PUSH EDX

PUSH ESI

PUSH ECX

PUSH EDI

PUSH EBX

MOV EAX, endt ; EAX = end

SUB EAX, start ; EAX = end - start

INC EAX ; EAX += 1

MOV ECX, EAX ; ECX = end - start + 1

MOV EBX, EAX

MOV ESI, start ; ESI = start

MOV EAX, 4

MUL ESI

MOV ESI, EAX

MOV EAX, 4

MUL ECX

PUSH EAX

MOV EAX, a ; EAX = a

ADD EAX, ESI

PUSH EAX ; push a

PUSH rest ; push res

call memcopy32

MOV EAX, rest ; EAX = начало массива

sort5j1: ; первый цикл

jmp sort5j2 ; Установка EAX на нужное место

sort5j3: ; возврат из sortj2

ADD EAX, 4 ; подготовка к итерации

loop sort5j1 ; итерация закончена уменьшение ECX

jmp sort5end ; прыжок на выход

sort5j2: ; второй счетчик

MOV EDX, EBX ; счетчик итераций для второго цикла

SUB EDX, ECX ; подготовка 2 счетчика

CMP EDX, 0

JE sort5j3

MOV ESI, EAX ; сохранение индекса для первого цикла

sort5j4:

MOV EDI, ESI ; создание второй переменной, для сравнения

SUB EDI, 4 ; EDI = вторая переменная

PUSH EAX

PUSH EBX

MOV EAX, dword ptr[ESI]

MOV EBX, dword ptr[EDI]

CMP EAX, EBX ; сравнение по указателю

POP EBX

POP EAX

JB sort5j5 ; прыжок на swap

jmp sort5j3 ; выход из итерации 2го счетчика

sort5j5:

PUSH ESI

PUSH EDI

call swap

DEC EDX

SUB ESI, 4

CMP EDX, 0

je sort5j3 ; выход и итерации 2го счетчика

jmp sort5j4 ; начать итерацию

sort5end:

POP EBX

POP EDI

POP ECX

POP ESI

POP EDX

POP EAX

MOV EAX, rest

ret

sort5 endp

swap proc stdcall ; void swap(float\* a, float\* b)

PUSH EAX

PUSH EBX

PUSH EBP

MOV EAX, [ESP + 16] ; EAX = a

MOV EBX, [ESP + 20] ; EBX = b

SUB ESP, 4

MOV EBP, ESP

PUSH 4

PUSH EAX

PUSH EBP

call memcopy32

PUSH 4

PUSH EBX

PUSH EAX

call memcopy32

PUSH 4

PUSH EBP

PUSH EBX

call memcopy32

ADD ESP, 4

POP EBP

POP EBX

POP EAX

ret 8

swap endp

memcopy32 proc stdcall ; void memcopy(void\* a, void\* b, int size) из b в a

; Сохранение используемых регистров в стеке

push ESI

push EDI

push ECX

mov ECX, [ESP+24] ; ECX = mem\_size

shr ECX, 2 ; ECX = ECX / 4 (mem\_size = mem\_size >> 2)

cld ; Очистка флага направления

mov esi, [ESP+20] ; ESI = source

mov edi, [ESP+16] ; EDI = dest

rep movsd ; Копирование блока

; Восстановление используемых регистров

pop ECX

pop EDI

pop ESI

ret 12

memcopy32 endp

start:

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

;PUSH offset [res]

;PUSH fend

;PUSH first

;PUSH offset [str1]

;call sort1

;ADD ESP, 16

MOV ECX, offset [str1]

MOV EDX, first

PUSH offset [res]

PUSH fend

call sort3

MOV EBX, EAX

MOV ECX, 20

j1:

PUSH ECX

PUSH [EBX]

PUSH offset format

call crt\_printf

ADD ESP, 8

POP ECX

ADD EBX, 4

loop j1

call crt\_\_getch ; Задержка ввода с клавиатуры

push 0

call ExitProcess ; Выход из программы

end start

;END DllMain

**Статический**

Def

LIBRARY num1

EXPORTS

\_sort1@16 = \_sort1@0

sort2

@sort3@16 = \_sort3@4

sort4

sort5

C++

#include <iostream>

extern "C" int\* \_stdcall sort1(int\* a, int start, int end, int\* res); //++

extern "C" int\* \_stdcall sort4(int\* a, int start, int end, int\* res); //++

extern "C" int\* \_cdecl sort2(int\* a, int start, int end, int\* res);

extern "C" int\* \_cdecl sort5(int\* a, int start, int end, int\* res); //++

extern "C" int\* \_fastcall sort3(int\* a, int start, int end, int\* res); //++

int main() {

int a[] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };

int res[10];

sort2(a, 0, 9, res);

for (int i = 0; i < 10; i++)

std::cout << res[i] << " ";

}

**Динамический**

#include <iostream>

#include <Windows.h>

int main() {

typedef int(\_stdcall\* myFunc) (int\* a, int start, int end, int\* res);

char dllName[] = "num1.dll";

HMODULE hModule = LoadLibraryA(dllName);

if (hModule != NULL) {

char funcName[] = "sort4";

myFunc func = (myFunc)GetProcAddress(hModule, funcName);

if (func == NULL) {

std::cout << "Function not found";

}

else {

int a[] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };

int res[10];

func(a, 0, 9, res);

for (int i = 0; i < 10; i++)

std::cout << res[i] << " ";

}

}

else {

std::cout << "DLL file not found";

}

}

**Алг. с вставками**

**Static inline void sort() {**

**\_asm {**

PUSH EAX

PUSH EDX

PUSH ESI

PUSH ECX

PUSH EDI

PUSH EBX

MOV EAX, endt ; EAX = end

SUB EAX, start ; EAX = end - start

INC EAX ; EAX += 1

MOV ECX, EAX ; ECX = end - start + 1

MOV EBX, EAX

MOV ESI, start ; ESI = start

MOV EAX, 4

MUL ESI

MOV ESI, EAX

MOV EAX, 4

MUL ECX

PUSH EAX

MOV EAX, a ; EAX = a

ADD EAX, ESI

PUSH EAX ; push a

PUSH rest ; push res

call memcopy32

MOV EAX, rest ; EAX = начало массива

sort5j1: ; первый цикл

jmp sort5j2 ; Установка EAX на нужное место

sort5j3: ; возврат из sortj2

ADD EAX, 4 ; подготовка к итерации

loop sort5j1 ; итерация закончена уменьшение ECX

jmp sort5end ; прыжок на выход

sort5j2: ; второй счетчик

MOV EDX, EBX ; счетчик итераций для второго цикла

SUB EDX, ECX ; подготовка 2 счетчика

CMP EDX, 0

JE sort5j3

MOV ESI, EAX ; сохранение индекса для первого цикла

sort5j4:

MOV EDI, ESI ; создание второй переменной, для сравнения

SUB EDI, 4 ; EDI = вторая переменная

PUSH EAX

PUSH EBX

MOV EAX, dword ptr[ESI]

MOV EBX, dword ptr[EDI]

CMP EAX, EBX ; сравнение по указателю

POP EBX

POP EAX

JB sort5j5 ; прыжок на swap

jmp sort5j3 ; выход из итерации 2го счетчика

sort5j5:

PUSH ESI

PUSH EDI

call swap

DEC EDX

SUB ESI, 4

CMP EDX, 0

je sort5j3 ; выход и итерации 2го счетчика

jmp sort5j4 ; начать итерацию

sort5end:

POP EBX

POP EDI

POP ECX

POP ESI

POP EDX

POP EAX

MOV EAX, rest

ret

**}**

**}**

**Подпрограммы на C++:**

#include <iostream>

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

void swap(int\* a, int index);

int main() {

int a[10] = { 10,9,8,7,6,5,4,3,2,1 };

int res[10];

sort(a, 0, 9, res);

for (int i = 0; i < 10; i++)

std::cout << res[i] << " ";

}

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

int index = 0;

for (int i = start; i <= end; i++) {

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

}

bool flag;

for (int i = 1; i < (end - start + 1); i++) {

flag = true;

index = i;

while (index > 0 && flag) {

if (res[index] < res[index - 1])

swap(res, index--);

else

flag = !flag;

}

}

return res;

}

void swap(int \*a, int index) {

int path = a[index - 1];

a[index - 1] = a[index];

a[index] = path;

}

**Написали программу для измерения быстродействия подпрограмм:**

void init\_random\_arr(float \*arr, int n) { srand(time(NULL));

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

{float random = ((float)rand()) / (float)RAND\_MAX; float number = (random \* 200)-100; // Float Numbers between 0 and 100 arr[i] = number;

}

}

void arrcpy(float \*arr1, float \*arr2, int n) {

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

arr1[i] = arr2[i];

}

}

int main()

{

clock\_t begin\_time;

float \*arrAsm = (float\*)std::malloc(SIZE \* sizeof(float));

float \*arrAsmInsert = (float\*)std::malloc(SIZE \* sizeof(float));

float \*arrCPlusPlus = (float\*)std::malloc(SIZE \* sizeof(float));

for (int i = 1; i <= 10; i++) {

// Array Init int n = i \* 10000; init\_random\_arr(arrAsm, n); arrcpy(arrAsmInsert, arrAsm, n); arrcpy(arrCPlusPlus, arrAsm, n);

float posarr[SIZE];

float negarr[SIZE];

int poscount = 0;

int negcount = 0;

// Clean Asm begin\_time = clock();

negcount = mainfunc(arrAsm, n, posarr, &poscount, negarr);

float timeAsm = float(clock() - begin\_time) / CLOCKS\_PER\_SEC;

poscount = 0; negcount = 0;

// Asm Insert begin\_time = clock();

negcount = mainfuncAsm(arrAsmInsert, n, posarr, &poscount, negarr);

float timeAsmInsert = float(clock() - begin\_time) / CLOCKS\_PER\_SEC;

poscount = 0; negcount = 0;

// Clean C++ begin\_time = clock();

negcount = mainfuncCPlus(arrCPlusPlus, n, posarr, &poscount, negarr);

float timeCPlus = float(clock() - begin\_time) / CLOCKS\_PER\_SEC;

std::cout << "N = " << n << std::endl; std::cout << "Assembler: " << timeAsm << " Sec" << std::endl; std::cout << "Assembler Insert: " < < timeAsmInsert << " Sec" << std::endl;

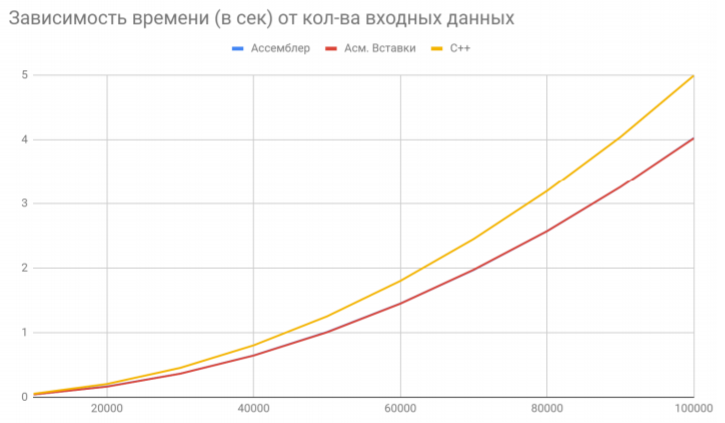
std::cout << "C++: " << timeCPlus < < " Sec" << std::endl; std::cout << std::endl << std::endl;

}

return 0;

}

**График:**

****

**Вывод:** в результате выполнения работы были изучены способы вызова подпрограмм, написанных на разных языках программирования посредством dll-библиотек. Написали одни и те же подпрограммы на ассемблере и C++ и сравнили их быстродействие. По результатам можно сказать, что подпрограммы, написанные на ассемблере, работают быстрее, чем на языках более высокого уровня.