«Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ_____
КАФЕДРА _____КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ_____

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

О Т Ч Е Т

по лабораторной работе № 5

Дисциплина: Машинно-зависимые языки и основы компиляции

Название лабораторной работы: Связь разноязыковых модулей

Вариант: 17

Студент гр. ИУ6-43Б **26.03.2022** **М.А. Мяделец**
(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ **М.В. Широкова**
(Подпись, дата) (И.О. Фамилия)

Москва, 2022

Связь разноязыковых модулей

Цель работы: изучение конвенций о способах передачи управления и данных при вызове из программы, написанной на языке высокого уровня, подпрограмм, написанных на ассемблере.

Задание

Рассмотрим задание для лабораторной работы 5 (смотри рисунок 1).

Лабораторная работ №5. Программирование с использованием разноязыковых модулей.
Дан текст не более 255 символов. Слова отделяются друг от друга пробелами.
Поменять местами пары слов с указанными номерами.
Второй модуль должен использовать ковенцию: cdecl.
Третий модуль должен использовать ковенцию: stdcall.

Рисунок 1 – Условие задания

Схемы алгоритма всех частей программы

Основная программа написана на языке высокого уровня C++. Главная функция main (смотри рисунок 2) просит пользователя ввести строчку и номера слов, которые нужно поменять местами. Далее введенные данные проверяются на корректность функцией IsError и подготавливаются для работы: в начале и в конце строки ставится пробел для более компактного и удобного синтаксического анализа строки на ассемблере (смотри рисунок 3). Затем подготовленная строка передается в функцию FuncStr, определение которой находится в файле с ассемблерным кодом. После того, как обработка строки завершится (смотри рисунок 5), в ассемблере происходит вызов функции print, которая определена в основной программе на C++ (смотри рисунок 4).

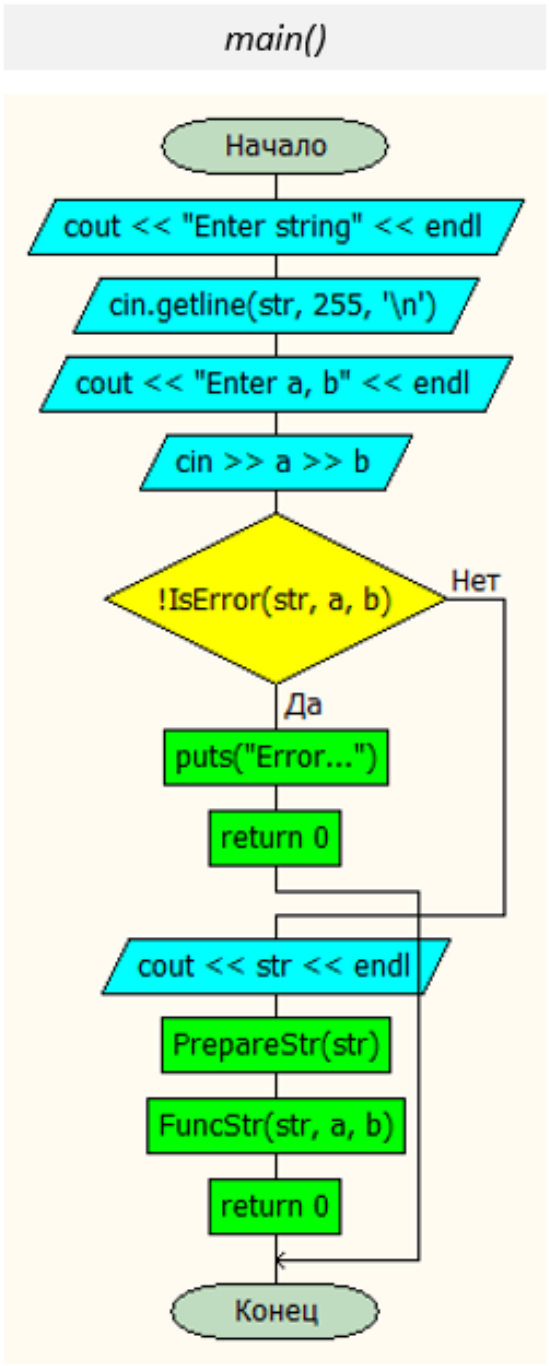

Рисунок 2 – Главная функция main()

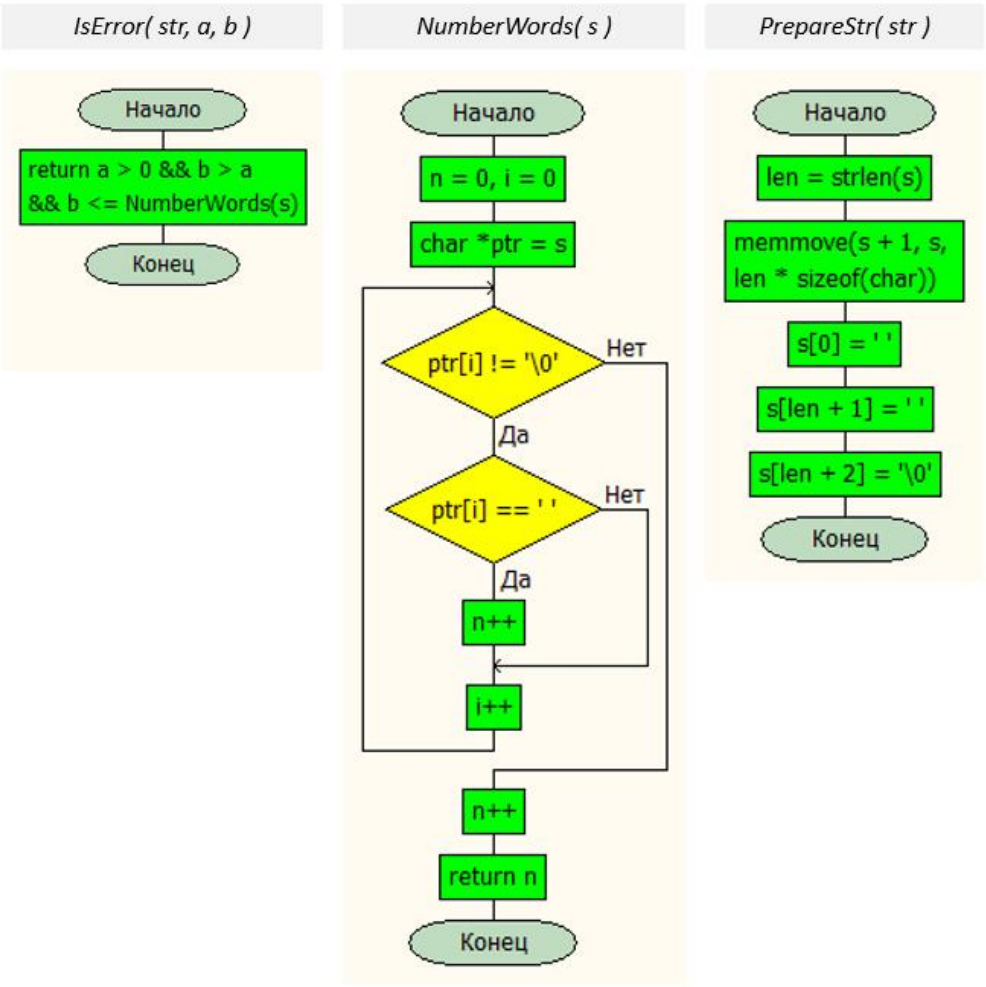

Рисунок 3 – Вспомогательные функции

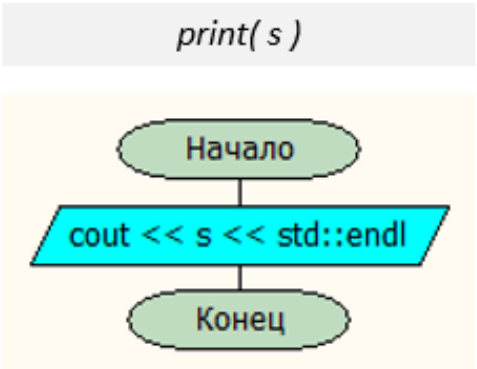

Рисунок 4 – Функция вывода результата

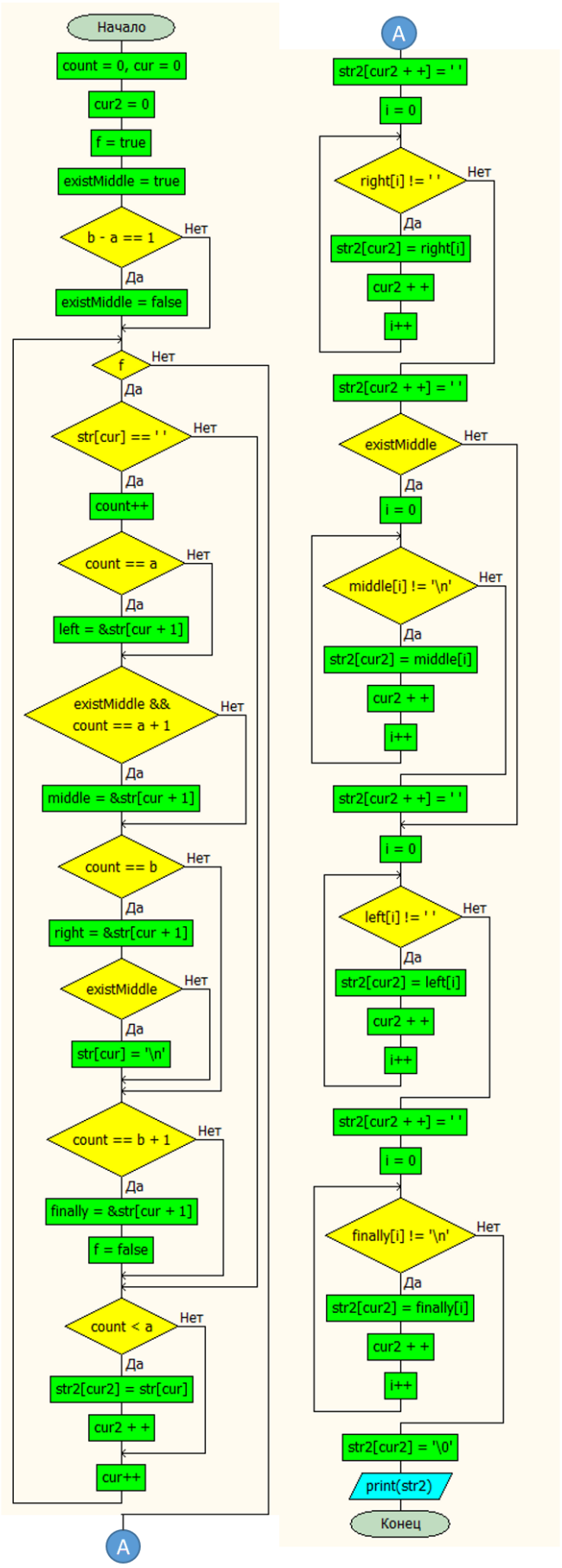

Рисунок 5 – Функция FuncStr на ассемблере

Совместная компоновка, организация передачи и возврата управления

Для того чтобы объединить программу, написанную на языке высокого уровня C++ и программу, написанную на языке ассемблера, нужно отдельно скомпилировать оба файла соответствующими компиляторами и тем самым создать объектные файла, а далее скомпоновать их совместно.

```
nasm -f elf64 text.asm
g++ main.cpp -c
g++ main.o text.o
./a.out
```

Рисунок 6 – Компиляция и совместная компоновка модулей

Либо же можно сократить количество команд, объединив компиляцию main.cpp и линковку с text.o.

```
nasm -f elf64 text.asm
g++ main.cpp text.o
./a.out
```

Рисунок 7 - Компиляция и совместная компоновка модулей (упрощенный вариант)

Для того, чтобы из C++ вызывать функцию, определенную в ассемблере, и наоборот, необходимо явно указать внутренние имена модуля, к которым будет происходить обращение из других модулей, и внешние, которые определены в других модулях, но к которым есть обращение из данного модуля.

```
extern void FuncStr(const char *s, int a, int b);

void print(char *s)
{
    cout << s << std::endl;
}
```

Рисунок 8 – Внешняя и внутренняя функции в C++.

```
global _Z7FuncStrPKcii
extern _Z5printPc
```

Рисунок 9 - Внешняя и внутренняя функции в ассемблере.

В языке C++, в отличие от Си, функции могут быть перегружены, поэтому для однозначного определения, какую в данный момент функцию нужно вызвать, компилятор C++ присваивает функциям идентификаторы в соответствие со следующим шаблоном «_Z<порядковый номер><название функции><типы параметров>». В коде ассемблера необходимо использовать именно эти названия. Их можно узнать, открыв объектный файл main.o в 16-ричном тактовом редакторе (смотри рисунок 10).

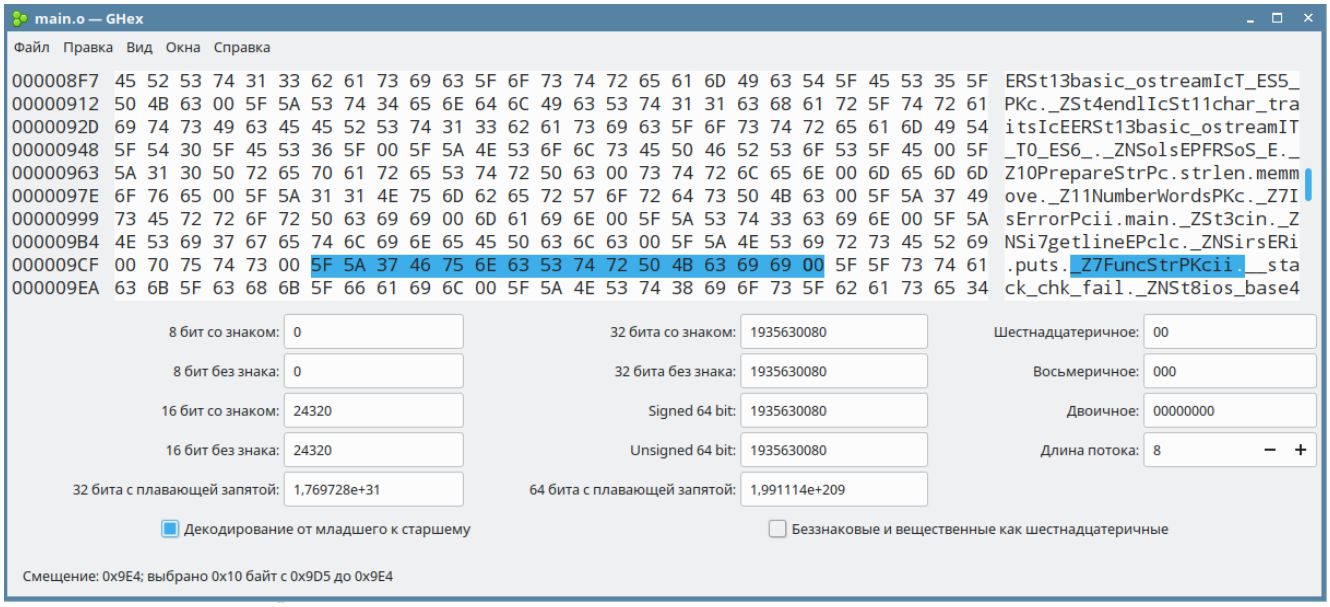

Рисунок 10 – 16-ричный редактор GHex

В 64-х разрядной программе для передачи параметров используют регистры RDI, RSI, RDX, R10, R8, R9. Функции, за исключением функций, результатом которых является строка, возвращают результат в регистре RAX. Если функция возвращает строку, то адрес результата передается в регистре RDI, т.е. в первом регистре, используемом для передачи параметров.

```
int a;
int b;
char str[258];
FuncStr(str, a, b);
```

Рисунок 11 – Передача параметров в C++.

В данном случае строка будет передана в регистре RDI, целочисленные переменные в регистрах RSI и RDX соответственно (смотри рисунки 11, 12 и 13).

```
mov edx, [rbp-0x114]
mov ecx, [rbp-0x118]
lea rax, [rbp-0x110]
mov esi, ecx
mov rdi, rax
call a.out!FuncStr(char const*, int, int)
```

Рисунок 12 – Вызов подпрограммы, определенной в модуле на ассемблере.

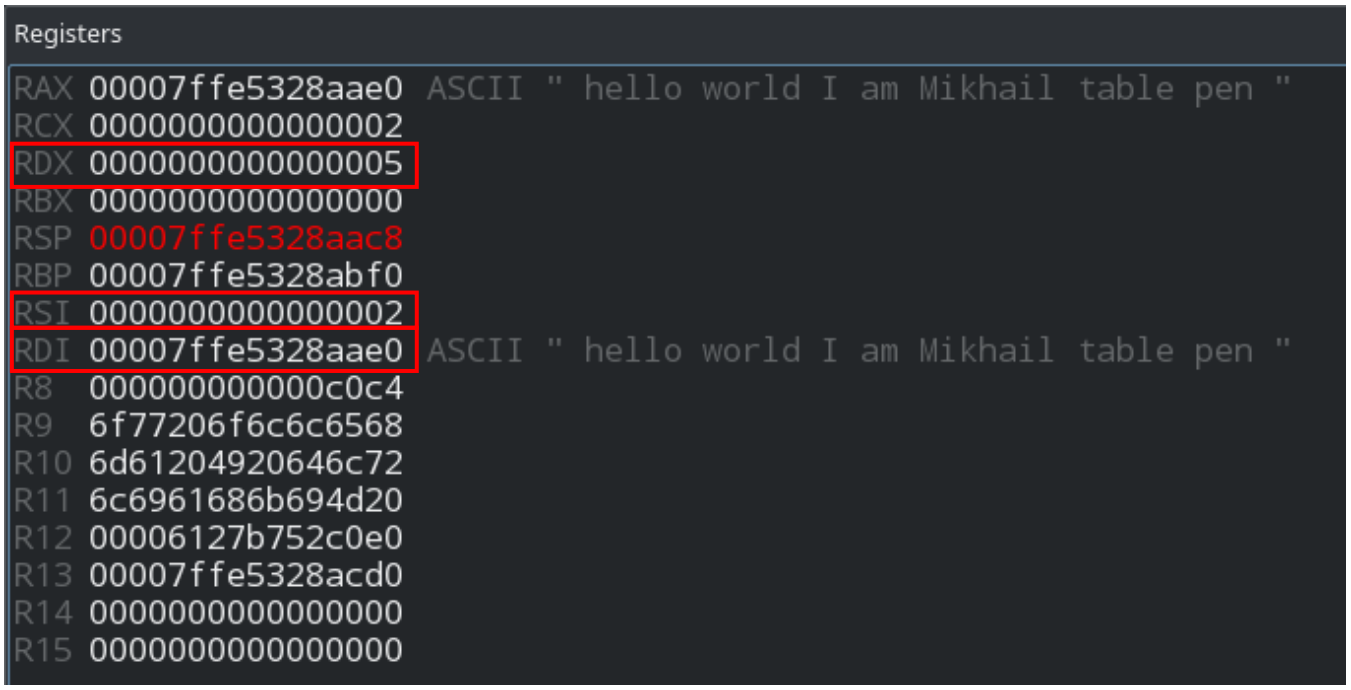

Рисунок 13 – Содержимое регистров после перехода в подпрограмму.

Видно, что регистры RDI, RSI и RDX принимают значения, введенные с консоли (смотри рисунок 14).

Рисунок 14 – Выполнение программы

Для корректной работы функции и выхода из нее нужно выполнять инструкции, соответствующие прологу и эпилогу. Стек после перехода в подпрограмму будет содержать адрес возврата, а после пролога – сохраненный адрес базы стека до захода в подпрограмму (смотри рисунки 15 и 16).

После пересылки адреса вершины стека в регистр RBP область локальных переменных и область, содержащая копии параметров, будет адресоваться относительно нового адреса RBP.

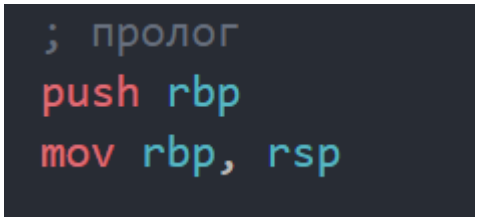

Рисунок 15 – Пролог

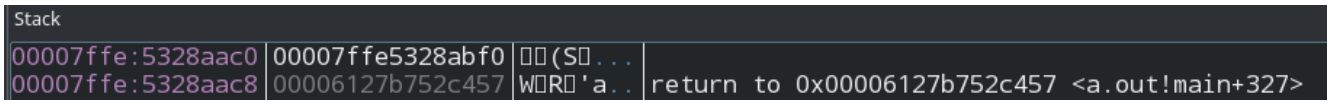

Рисунок 16 – Содержимое стека после пролога.

Эпилог выглядит следующим образом (смотри рисунок 17). Первая команда удаляет все локальные переменные функции, размещенные на стеке, вторая – возвращает исходное состояние базы стека до захода в функцию, третья – способствует переходу в вызывающую подпрограмму по адресу возврата, сохраненному в стеке.

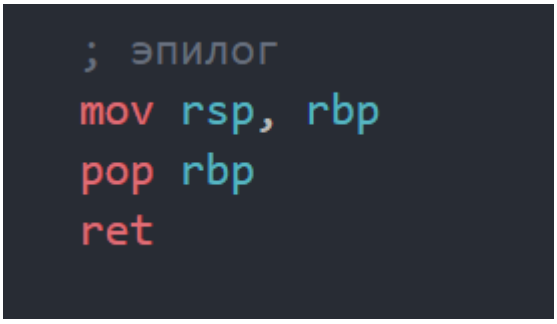

Рисунок 17 – эпилог для корректного возврата управления.

Передача параметров из ассемблера в функцию, определенную в C++, происходит аналогичным образом. В регистр RDI пересылаем адрес обработанной строки и вызываем подпрограмму через идентификатор (смотри рисунок 15).

```
mov rdi, str2          ; возвращаем rdi на первый символ
inc rdi
call _Z5printPc
```

Рисунок 18 – Передача параметров в ассемблере.

Основная идея решения

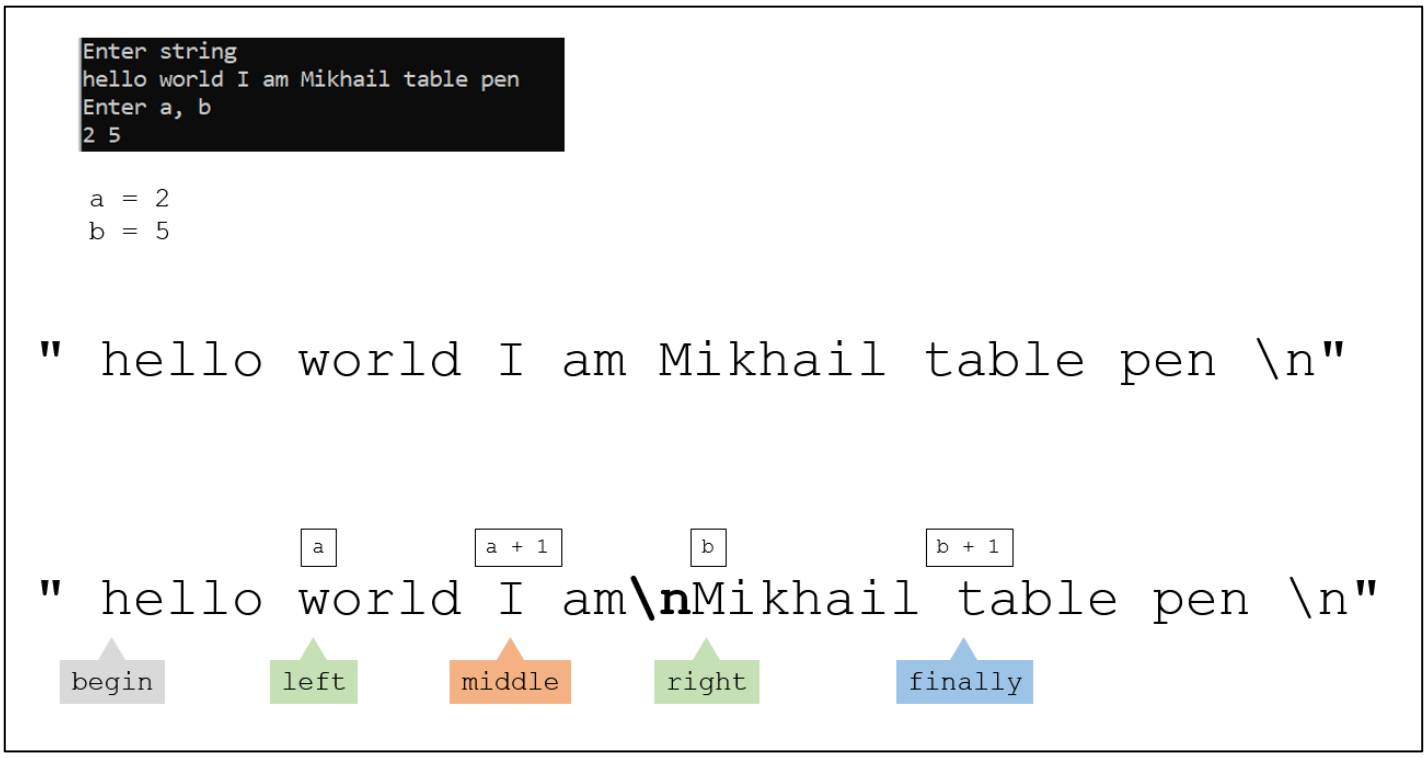

Рисунок 19 – Идея решения задачи.

Для удобной реализации придуманного алгоритма предварительно необходимо обработать строчку, добавив по одному пробелу в начале и в конце. Это делает функция PrepareStr в модуле C++. В ассемблерном модуле реализован основной алгоритм.

1) Резервируем строчку для результата.
2) Циклом проходимся по всей исходной строке и заносим опорные адреса (left, middle, right, finally) в регистры r10 - r13.

- На каждой итерации в результирующую строчку копируется символ до первого слова, который нужно поменять местами со вторым (до left).

3) После цикла мы имеем результирующую строчку с символами “hello” и опорные адреса.
4) Далее копируем в результирующую строку
   - right до пробела;
   - middle до символа конца строки
   - left до пробела
   - finally до символа конца строки.
5) Передаем результирующую строчку в функцию print (смотри рисунок 20).

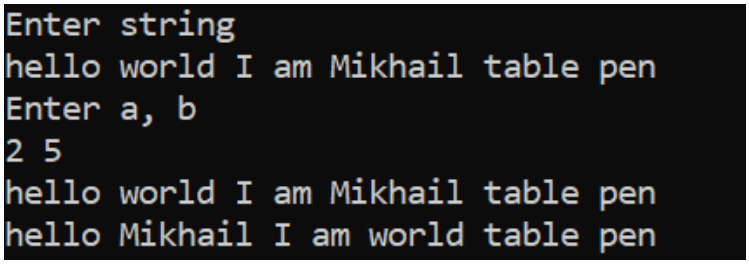

Рисунок 20 – Результат алгоритма.

Для того, чтобы поменять местами два слова при отсутствие каких-либо других компонентов алгоритма (без middle, без finally, без begin) смотри комментарии в коде программы (смотри приложение).

Тестирование

Протестируем разработанную программу, используя граничные условия и некорректно введенные данные (смотри рисунки 19 - 24).

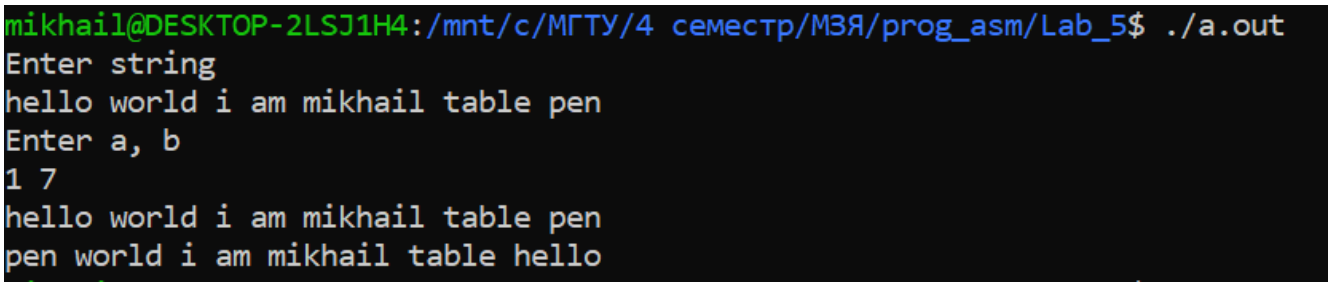

Рисунок 21 – Тест 1

```
mikhail@DESKTOP-2LSJ1H4:/mnt/c/МГТУ/4 семестр/МЗЯ/prog_asm/Lab_5$ ./a.out
Enter string
hello world i am mikhail table pen
Enter a, b
5 6
hello world i am mikhail table pen
hello world i am table mikhail pen
```

Рисунок 22 – Тест 2

```
mikhail@DESKTOP-2LSJ1H4:/mnt/c/МГТУ/4 семестр/МЗЯ/prog_asm/Lab_5$ ./a.out
Enter string
hello world i am mikhail table pen
Enter a, b
2 9
Error...
```

Рисунок 23 – Тест 3

```
mikhail@DESKTOP-2LSJ1H4:/mnt/c/МГТУ/4 семестр/МЗЯ/prog_asm/Lab_5$ ./a.out
Enter string
Pen Table Bottle
Enter a, b
2 2
Error...
```

Рисунок 24 - Тест 4

```
mikhail@DESKTOP-2LSJ1H4:/mnt/c/МГТУ/4 семестр/МЗЯ/prog_asm/Lab_5$ ./a.out
Enter string
Hello world
Enter a, b
1 2
Hello world
world Hello
```

Рисунок 25 - Тест 5

```
mikhail@DESKTOP-2LSJ1H4:/mnt/c/МГТУ/4 семестр/МЗЯ/prog_asm/Lab_5$ ./a.out
Enter string
A B C D E F G H I J K L M N O P Q R S T
Enter a, b
1 20
A B C D E F G H I J K L M N O P Q R S T
T B C D E F G H I J K L M N O P Q R S A
```

Рисунок 26 – Тест 6

Вывод: была сделана лабораторная работа 5, которая была направлена на реализацию совместного выполнения разноязыковых модулей. В ходе выполнения задания была осуществлена совместная компоновка модулей, организация передачи параметров в подпрограмму и возврата управления в вызывающую программу. Также был придуман алгоритм, позволяющий менять местами произвольные слова в предложении. Программа работает успешно, результаты тестирования корректны.

Контрольные вопросы

1) *Что такое «конвенции о связи»?*

   Операционные системы Linux, как и другие Unix-системы, использует единственную конвенцию, которая включена в стандартизованный двоичный интерфейс приложения (англ. System V Application Binary Interface или сокр. System V ABI). Указанная конвенция предполагает, что до 6-ти параметров передается в регистрах, остальные – в стеке в обратном порядке. Порядок занесения параметров в регистры прямой, т.е. первым записывается в регистр первый параметр и т.д.

2) *Что такое «пролог» и «эпилог»? Где располагается область локальных данных?*

   Ответ смотри на странице 9.

3) *Как связана структура данных стека в момент передачи управления и текст программы и подпрограмм?*

   Ответ смотри на страницах 8-9.

4) *С какой целью применяют разноязыкорвые модули в одном проекте?*

   У каждого языка свои преимущества и недостатки в связи в разным уровнем абстракции. Языки высокого уровня выполняют задачу компактно, ограничены в возможностях и не могут напрямую взаимодействовать с памятью и процессором, с чем как раз-таки прекрасно справляется assembler.

ПРИЛОЖЕНИЕ

Текст программы

text.asm

```
      global _Z7FuncStrPKcii
      extern _Z5printPc

      section .bss

str2  resb  255

      section .text
_Z7FuncStrPKcii:
      ; пролог
      push rbp
      mov rbp, rsp

         ; параметры функции: rdi - строка, rsi - a, rdx - b

      ; подготовка регистров

      mov rcx, 0       ; count - счетчик слов
      mov r8, rsi      ; a - позиция первого слова
      mov r9, rdx      ; b - позиция второго слова
      mov rsi, rdi     ; str - исходная строка
      mov rdi, str2    ; str2 - измененная строка
      mov bx, 1  ; f = true;
         mov dx, 1        ; existMiddle = true;

      ; r10 - left
      ; r11 - middle
      ; r12 - right
      ; r13 - finally

         ; установка existMiddle
         push r9          ; stack >
         sub r9, r8
         cmp r9, 1
         pop r9           ; stack <
         jne cycle
         mov dx, 0
cycle:
      cmp bx, 1
      jne fill_full
            cmp byte[rsi], 20h
            jne fill_begin

                  ; обрабатываем <пробел>

            inc rcx
            cmp rcx, r8      ; if (count == a)
                  jl fill_begin
            jg middle
            inc rsi             ; переходим на <символ>
            mov r10, rsi     ; r10 - left
            jmp next_sym
middle:
```

```
                inc rsi             ; переходим на <символ>
                cmp dx, 1       ; if (existMiddle && count == a + 1)
                jne right

           push r8          ; stack <
           inc r8
           cmp rcx, r8
           pop r8           ; stack >
           jne right
           mov r11, rsi     ; r11 - middle
           jmp next_sym

right:
           cmp rcx, r9      ; if (count == b)
           jne finally

           mov r12, rsi     ; r12 - right

                cmp dx, 1       ; if (existMiddle)
                jne next_sym
           dec rsi          ; перед right ставим \n
           mov byte[rsi], 10
           inc rsi

           jmp next_sym

finally:
           push r9          ; stack <
           inc r9
           cmp rcx, r9      ; if (count == b + 1)
           pop r9           ; stack <
           jne next_sym
           mov r13, rsi     ; r13 = finally
           mov bx, 0        ; f = false;

                ; заполняем начало (до слова left)
fill_begin:
           cmp rcx, r8
                jge next_sym
           movsb            ; из rsi в rdi пересылаем символ
              jmp cycle
next_sym:
                inc rsi
                jmp cycle

fill_full:

        ; перемещаемся по rsi через r10 - r13
        ; заполняем rdi

        ; "<begin> <right>"
      mov byte[rdi], 20h      ; str2[cur2++] = ' ';
      inc rdi
      mov rsi, r12            ; r12 - right
r_cycle:
      cmp byte[rsi], 20h      ; while(right[i] != ' ')
      je end_r_cycle
      movsb
      jmp r_cycle
end_r_cycle:
```

```
        ; "<begin> <right> <middle>"
        cmp dx, 1               ; if (existMiddle)
        jne not_exist_middle
      mov byte[rdi], 20h
      inc rdi
      mov rsi, r11            ; r11 - middle
m_cycle:
      cmp byte[rsi], 10       ; while(middle[i] != '\n')
      je end_m_cycle
      movsb
      jmp m_cycle
end_m_cycle:

not_exist_middle:

        ; "<begin> <right> <middle> <left>"
      mov byte[rdi], 20h
      inc rdi
      mov rsi, r10            ; r10 - left
l_cycle:
      cmp byte[rsi], 20h      ; while(left[i] != ' ')
      je end_l_cycle
      movsb
      jmp l_cycle
end_l_cycle:

        ; "<begin> <right> <middle> <left> <finally>"
      mov byte[rdi], 20h
      inc rdi
      mov rsi, r13            ; r10 - finally
f_cycle:
      cmp byte[rsi], 10       ; while(finally[i] != '\n')
      je end_f_cycle
      movsb
      jmp f_cycle
end_f_cycle:

      mov byte[rdi], 10       ; последний символ "\n"
        mov rdi, str2           ; возвращаем rdi на первый символ
        inc rdi
      call _Z5printPc

      ; эпилог
      mov rsp, rbp
      pop rbp
ret
```

main.cpp

```
#include <iostream>
#include <cstring>
using std::cin;
using std::cout;
using std::endl;
```

```
	extern void FuncStr(const char *s, int a, int b);

	void print(char *s)
	{
		cout << s << std::endl;
	}

	void PrepareStr(char *s)
	{
		int len = strlen(s);
		memmove(s + 1, s, len * sizeof(char));
		s[0] = ' ';
		s[len + 1] = ' ';
		s[len + 2] = '\0';
	}

	int NumberWords(const char *s)
	{
		int n = 0, i = 0;
		const char *ptr = s;
		while (ptr[i] != '\0')
		{
			if (ptr[i] == ' ')
			{
				n++;
			}
			i++;
		}
		n++;
		return n;
	}

	bool IsError(char *s, int a, int b)
	{
			return a > 0 && b > a && b <= NumberWords(s);

	}

	int main()
	{
		// int a = 2, b = 7;
		int a;
		int b;
		char str[258];
		cout << "Enter string" << endl;
		cin.getline(str, 255, '\n');
		cout << "Enter a, b" << endl;
		cin >> a >> b;
		if (!IsError(str, a, b))
		{
			puts("Error...");
			return 0;
		}
		cout << str << endl;
		PrepareStr(str);
		FuncStr(str, a, b);
		return 0;
}
```