|  |  |
| --- | --- |
| Gerb-BMSTU_01 | ***«*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\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по лабораторной работе № 5**

**Дисциплина: Машинно-зависимые языки и основы компиляции**

**Название лабораторной работы: Программирование с использованием разноязыковых модулей**

Вариант 2.21

Студент гр. ИУ6-42Б  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.С. Твердюк**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.С. Данилюк**

(Подпись, дата) (И.О. Фамилия)

Москва, 2023

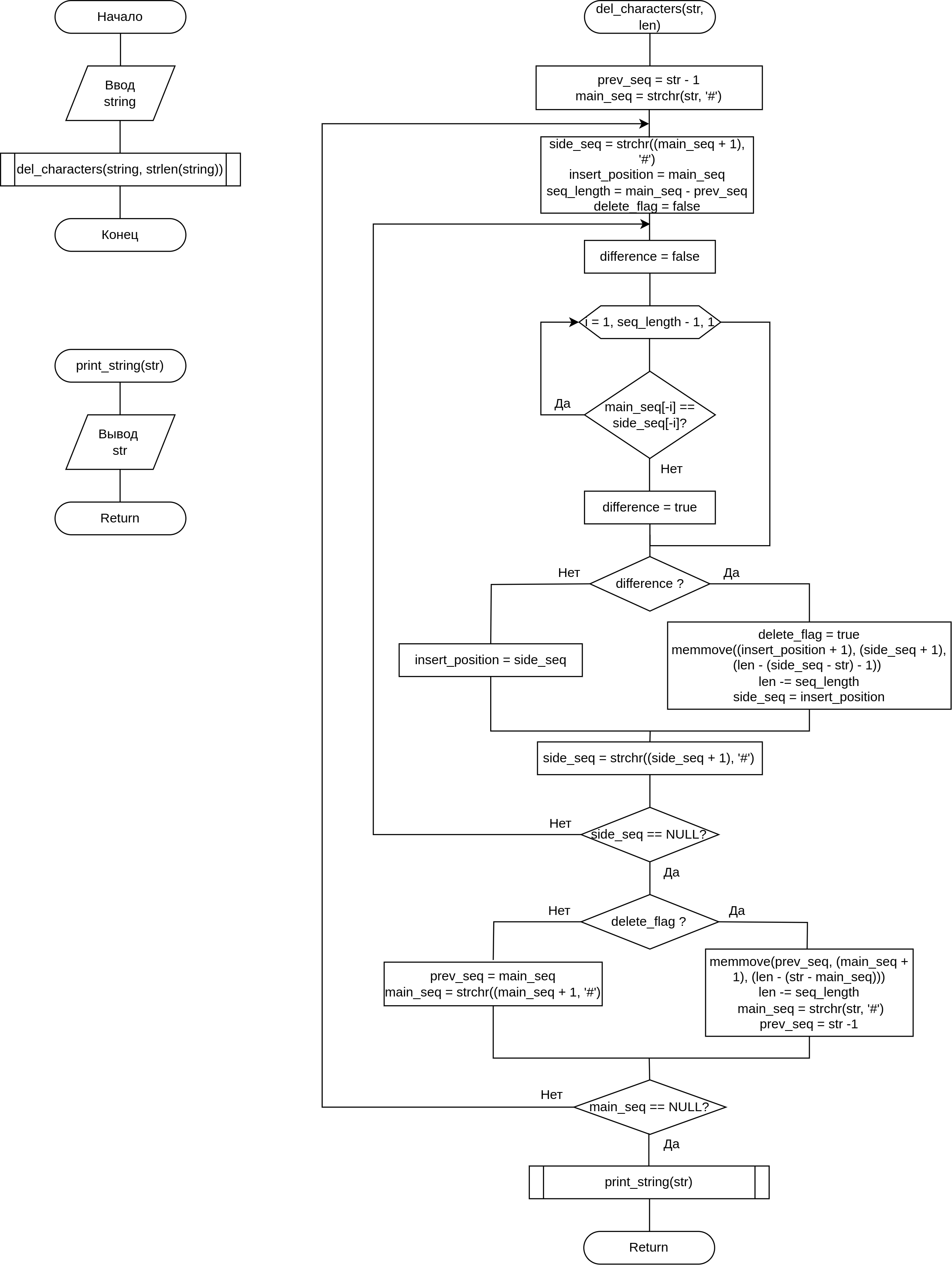
***Цель работы:*** изучение процессов создания, запуска и отладки программ на ассемблере Nasm под управлением операционной системы Linux, а также особенностей описания и внутреннего представления данных.

***Ход работы:***

***Задание.*** Дан текст не более 255 символов. Удалить последовательности одинаковых символов, завершающиеся символом «#».

Разработанная схема алгоритма приведена на рисунке 1.

а

Рисунок 1 - Схема алгоритма

**Программа main.c**

#include <stdio.h>

#include <string.h>

extern void del\_characters(char\* string, int lenghth);

int main() {

char string[256] = {};

puts("Введите строку:");

fgets(string, 255, stdin);

string[strlen(string) - 1] = '\0';

del\_characters(string, strlen(string));

return 0;

}

**Программа output.c**

#include <stdio.h>

void print\_string(char\* string) {

puts(string);

}

**Программа libasm.asm**

section .text

extern print\_string

global del\_characters

del\_characters:

push rbp

mov rbp, rsp

push rax

push rbx

push rcx

push rdx

push r8

push r9

push r10

push r11

push r12

push r13

push r14

push r15

xor rbx, rbx ; адрес начала строки

xor rdx, rdx ; длина последовательности символов

xor r8, r8 ; адрес ПРЕДШЕСТВУЮЩЕЙ последовательности символов

xor r9, r9 ; адрес ОСНОВНОЙ последовательности символов

xor r10, r10 ; адрес ПОБОЧНОЙ последовательности

xor r11, r11 ; адрес начала усечения строки

lea rbx, [rdi]

mov r8, rbx

dec r8

mov rcx, rsi ; длина строки

mov rax, '#'

repne scasb

mov r9, rdi

dec r9

next\_main\_seq:

mov rcx, rsi

mov r14, r9

sub r14, rbx

sub rcx, r14

dec rcx

cmp rcx, 0

je return

repne scasb

mov r10, rdi

dec r10

mov r11, r9

mov rdx, r9

sub rdx, r8

xor r12, r12 ; флаг наличия удаления последовательности

next\_side\_seq:

xor r13, r13 ; флаг наличия различий между последовательностями символов

xor rcx, rcx ; смещение внутри последовательностей символов

sequence\_compare:

dec rcx

mov rax, rdx

add rax, rcx

cmp rax, 0

je compare\_end

mov r14b, byte[r9 + rcx]

mov r15b, byte[r10 + rcx]

cmp r14b, r15b

je sequence\_compare

inc r13

compare\_end:

mov rax, '#'

cmp r13, 1

jne truncate\_string

mov r11, r10

jmp skip\_truncate

truncate\_string:

inc r12

call \_memmove

sub rsi, rdx

mov r10, r11

skip\_truncate:

mov rdi, r10

inc rdi

mov rcx, rsi

mov r14, r11

sub r14, rbx

sub rcx, r14

dec rcx

cmp rcx, 0

je main\_seq\_cycle

repne scasb

mov r10, rdi

dec r10

jmp next\_side\_seq

main\_seq\_cycle:

cmp r12, 0

jg delete\_main\_seq

mov r8, r9

mov rdi, r9

inc rdi

mov rcx, rsi

mov r14, r9

sub r14, rbx

sub rcx, r14

dec rcx

cmp rcx, 0

je return

repne scasb

mov r9, rdi

dec r9

jmp next\_main\_seq

delete\_main\_seq:

call \_delete\_seq

sub rsi, rdx

mov rdi, rbx

mov rcx, rsi

repne scasb

jne return

mov r9, rdi

dec r9

mov r8, rbx

dec r8

jmp next\_main\_seq

return:

lea rdi, [rbx]

pop rax

pop rbx

pop rcx

pop rdx

pop r8

pop r9

pop r10

pop r11

pop r12

pop r13

call print\_string

mov rsp, rbp

pop rbp

ret

\_memmove:

mov rcx, rsi

mov r14, r10

sub r14, rbx

sub rcx, r14

dec rcx

cmp rcx, 0

je last\_seq1

mov r14, 1

swap\_data1:

mov r15b, byte[r10 + r14]

mov byte[r10 + r14], 0

mov [r11 + r14], r15b

inc r14

loop swap\_data1

mov byte[r11 + r14], 0

jmp return\_memmove

last\_seq1:

mov rcx, rdx

zeroing1:

mov byte[r11 + rcx], 0

loop zeroing1

return\_memmove

ret

\_delete\_seq:

mov rcx, rsi

sub rcx, rdx

mov r14, 1

cmp rcx, 0

jne swap\_data2

mov byte[r8 + 1], 0

jmp return\_delete\_seq

swap\_data2:

mov r15b, byte[r9 + r14]

mov byte[r9 + r14], 0

mov [r8 + r14], r15b

inc r14

loop swap\_data2

mov byte[r8 + r14], 0

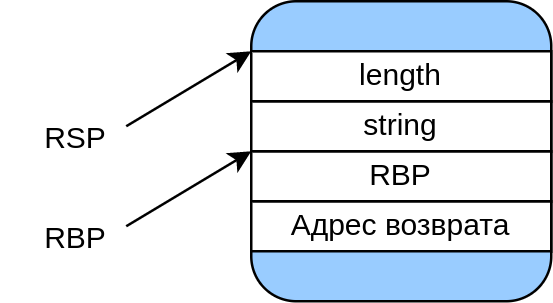
jmp return\_delete\_seq

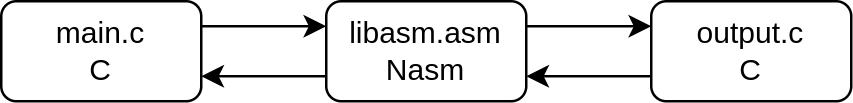
return\_delete\_seq:

ret

Структура стека приведена на рисунке 2.

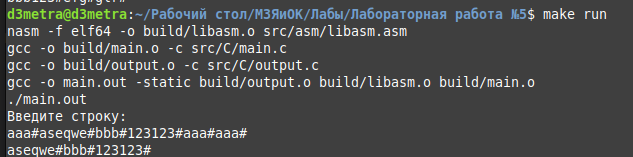
Структурную декомпозицию можно наблюдать на рисунке 3.

Рисунок 2 - Структура стека

Рисунок 3 - Схема структурной декомпозиции

Запустим программу. Результат выполнения программы представлен на рисунке 4.

Проведём тестирование программы. Результаты представлены в таблице 1.

Рисунок 4 - Выполнение программы

*Таблица 1 — Результаты тестирования программы*

|  |  |  |
| --- | --- | --- |
| **Исходные данные** | **Ожидаемый результат** | **Полученный результат** |
| aaa#bbb#bbb#aaa# | — | — |
| abc#bcd#erg# | abc#bcd#erg# | abc#bcd#erg# |
| abc#aaa#aaa#bbb#grw# | abc#grw# | abc#grw# |
| aaa#aaa#twe# | twe# | twe# |

***Контрольные вопросы***

1. Что такое «конвенции о связи»? В чем заключается конвенция register?

Конвенции о связи это правила передачи параметров при организации связи модулей на различных языках. Конвенция register заключается в том, что первые три параметра передаются через регистры EAX, EDX и ECX, а остальные передаются через стек.

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

Пролог - это специальные команды, которые необходимо написать в начале подпрограммы для сохранения состояния регистров до вызова функции и правильной передачи управления. Эпилог - это специальные команды, которые необходимо написать в конце подпрограммы для возврата состояния регистров до вызова функции и правильной передачи управления. Область локальных данных располагается в стеке.

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

В момент передачи управления структура стека зависит от конвенций о связи разноязыковых модулей. Обычно в стек помещаются передаваемые параметры, и данные необходимые для возврата управления.

4. С какой целью применяют разноязыковые модули в одном проекте?

С помощью вызова модуля, написанного на низкоуровневом языке программирования можно повысить скорость работы необходимой части программы, написанной на языке программирования более высокого уровня.

***Вывод:*** изучены процессы создания, запуска и отладки программ на ассемблере Nasm под управлением операционной системы Linux, а также особенности описания и внутреннего представления данных.