

# **Лабораторная работа №7**

**Команды безусловного и условного переходов в Nasm.  
Программирование ветвлений.**

Чекмарев Александр Дмитриевич | группа: НПИбд 02-23

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
2.1	Реализация переходов в NASM . . . . .	6
2.2	Изучение структуры файлы листинга . . . . .	13
<b>3</b>	<b>Самостоятельная работа</b>	<b>18</b>
<b>4</b>	<b>Выводы</b>	<b>24</b>

## Список иллюстраций

2.1	Рис 2.1.1: Создание каталога и файла .asm . . . . .	6
2.2	Рис 2.1.2: Демонстрация текста программы в файле . . . . .	7
2.3	Рис 2.1.3: Создание файла и его проверка . . . . .	8
2.4	Рис 2.1.4: Демонстрация текста программы в файле . . . . .	9
2.5	Рис 2.1.5: Создание файла и его проверка . . . . .	10
2.6	Рис 2.1.6: Демонстрация измененного текста в файле . . . . .	10
2.7	Рис 2.1.7: Проверка работы программы . . . . .	11
2.8	Рис 2.1.8: Создание файла .asm . . . . .	11
2.9	Рис 2.1.9: Демонстрация текста программы в файле . . . . .	12
2.10	Рис 2.1.10: Создание файла и проверка работы с разными значениями В . . . . .	13
2.11	Рис 2.2.1: Создание файла . . . . .	14
2.12	Рис 2.2.2: Демонстрация ввода команды . . . . .	14
2.13	Рис 2.2.3: Вид файла .lst в текст. редакторе . . . . .	15
2.14	Рис 2.2.4: Демонстрация взятых строк . . . . .	15
2.15	Рис 2.2.5: Удаление операнды . . . . .	16
2.16	Рис 2.2.6: Трансляция файла . . . . .	16
2.17	Рис 2.2.7: Демонстрация ошибки в файле .lst . . . . .	16
3.1	Рис 3.1.1: Создание файла . . . . .	18
3.2	Рис 3.1.2: Демонстрация программы для задания . . . . .	19
3.3	Рис 3.1.3: Проверка программы . . . . .	20
3.4	Рис 3.2.1: Демонстрация 4-го варианта . . . . .	20
3.5	Рис 3.2.2: Демонстрация 4-го варианта . . . . .	20
3.6	Рис 3.2.3: (1)Программа для задачи . . . . .	21
3.7	Рис 3.2.4: (2)Программа для задачи . . . . .	22
3.8	Рис 3.2.5: Проверка программы . . . . .	22
3.9	Рис 3.3.1: Загрузка файлов на github . . . . .	23

## Список таблиц

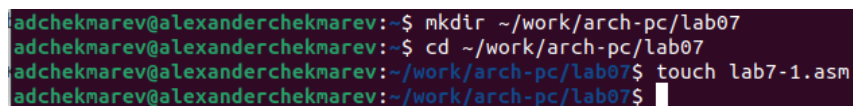
# 1 Цель работы

Изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

## 2 Выполнение лабораторной работы

### 2.1 Реализация переходов в NASM

Создадим каталог для программам лабораторной работы № 7, перейдите в него и создадим файл *lab7-1.asm*:



```
adchekmarev@alexanderchekmarev:~$ mkdir ~/work/arch-pc/lab07
adchekmarev@alexanderchekmarev:~$ cd ~/work/arch-pc/lab07
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ touch lab7-1.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$
```

Рис. 2.1: Рис 2.1.1: Создание каталога и файла .asm

Инструкция *jmp* в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции *jmp*. Введем в файл *lab7-1.asm* текст программы из листинга 7.1

```

1 %include 'in_out.asm' ; подключение внешнего файла
2
3 SECTION .data
4
5 msg1: DB 'Сообщение № 1',0
6 msg2: DB 'Сообщение № 2',0
7 msg3: DB 'Сообщение № 3',0
8
9 SECTION .text
10 GLOBAL _start
11 _start:
12
13 jmp _label2
14
15 _label1:
16     mov eax, msg1 ; Вывод на экран строки
17     call sprintf ; 'Сообщение № 1'
18
19 _label2:
20     mov eax, msg2 ; Вывод на экран строки
21     call sprintf ; 'Сообщение № 2'
22
23 _label3:
24     mov eax, msg3 ; Вывод на экран строки
25     call sprintf ; 'Сообщение № 3'
26
27 _end:
28     call quit ; вызов подпрограммы завершения

```

Рис. 2.2: Рис 2.1.2: Демонстрация текста программы в файле

Создим исполняемый файл и запустим его. Результат работы данной программы будет следующим:

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$
```

Рис. 2.3: Рис 2.1.3: Создание файла и его проверка

Таким образом, использование инструкции `*jmp _label2*` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `*_label2*`, пропустив вывод первого сообщения.

Инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. Изменим программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого в текст программы после вывода сообщения № 2 добавим инструкцию `jmp` с меткой `*_label1*` (т.е. переход к инструкциям вывода сообщения № 1) и после вывода сообщения № 1 добавим инструкцию `jmp` с меткой `*_end*` (т.е. переход к инструкции `call quit`). Измените текст программы в соответствии с листингом 7.2



```

1 %include 'in_out.asm' ; подключение внешнего файла
2
3 SECTION .data
4
5 msg1: DB 'Сообщение № 1',0
6 msg2: DB 'Сообщение № 2',0
7 msg3: DB 'Сообщение № 3',0
8
9 SECTION .text
10 GLOBAL _start
11 _start:
12
13 jmp _label2
14
15 _label1:
16     mov eax, msg1 ; Вывод на экран строки
17     call sprintf ; 'Сообщение № 1'
18     jmp _end
19
20 _label2:
21     mov eax, msg2 ; Вывод на экран строки
22     call sprintf ; 'Сообщение № 2'
23     jmp _label1
24
25 _label3:
26     mov eax, msg3 ; Вывод на экран строки
27     call sprintf ; 'Сообщение № 3'
28
29 _end:
30     call quit ; вызов подпрограммы завершения

```

Рис. 2.4: Рис 2.1.4: Демонстрация текста программы в файле

Создадим исполняемый файл и проверим его работу.

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
```

Рис. 2.5: Рис 2.1.5: Создание файла и его проверка

Изменим текст программы добавив или изменив инструкции *jmp*.

```
%include 'in_out.asm' ; подключение внешнего файла
```

```
SECTION .data
```

```
msg1: DB 'Сообщение № 1',0
```

```
msg2: DB 'Сообщение № 2',0
```

```
msg3: DB 'Сообщение № 3',0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
    jmp _label3
```

```
_label1:
```

```
    mov eax, msg1 ; Вывод на экран строки
```

```
    call sprintf ; 'Сообщение № 1'
```

```
    jmp _end
```

```
_label2:
```

```
    mov eax, msg2 ; Вывод на экран строки
```

```
    call sprintf ; 'Сообщение № 2'
```

```
    jmp _label1
```

```
_label3:
```

```
    mov eax, msg3 ; Вывод на экран строки
```

```
    call sprintf ; 'Сообщение № 3'
```

```
    jmp _label2
```

```
_end:
```

```
    call quit ; вызов подпрограммы завершения
```

Рис. 2.6: Рис 2.1.6: Демонстрация измененного текста в файле

Проверим работу

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$
```

Рис. 2.7: Рис 2.1.7: Проверка работы программы

Использование инструкции *jmp* приводит к переходу в любом случае. Однако, часто при написании программ необходимо использовать условные переходы, т.е. переход должен происходить если выполнено какое-либо условие. В качестве примера рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А, В и С. Значения для А и С задаются в программе, значение В вводится с клавиатуры

Создадим файл *lab7-2.asm* в каталоге *~/work/arch-pc/lab07*.

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ touch lab7-2.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$
```

Рис. 2.8: Рис 2.1.8: Создание файла .asm

Внимательно изучим текст программы из листинга 7.3 и введем в *lab7-2.asm*.

```

#include 'in_out.asm'
section .data
    msg1 db 'Введите B: ',0h
    msg2 db "Наибольшее число: ",0h
    A dd '20'
    C dd '50'
section .bss
    max resb 10
    B resb 10
section .text
    global _start
_start:
; ----- Вывод сообщения 'Введите B: '
    mov eax,msg1
    call sprint
; ----- Ввод 'B'
    mov ecx,B
    mov edx,10
    call sread
; ----- Преобразование 'B' из символа в число
    mov eax,B
    call atoi ; Вызов подпрограммы перевода символа в число
    mov [B],eax ; запись преобразованного числа в 'B'
; ----- Записываем 'A' в переменную 'max'
    mov ecx,[A] ; 'ecx = A'
    mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символы)
    cmp ecx,[C] ; Сравниваем 'A' и 'C'
    jg check_B ; если 'A>C', то переход на метку 'check_B',
    mov ecx,[C] ; иначе 'ecx = C'
    mov [max],ecx ; 'max = C'
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
    mov eax,max
    call atoi ; Вызов подпрограммы перевода символа в число
    mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
    mov ecx,[max]
    cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
    jg fin ; если 'max(A,C)>B', то переход на 'fin',
    mov ecx,[B] ; иначе 'ecx = B'
    mov [max],ecx
; ----- Вывод результата
fin:
    mov eax, msg2
    call sprint ; Вывод сообщения 'Наибольшее число: '
    mov eax,[max]
    call iprintLF ; Вывод 'max(A,B,C)'
    call quit ; Выход

```

Рис. 2.9: Рис 2.1.9: Демонстрация текста программы в файле

Создадим исполняемый файл и проверим его работу для разных значений *B*.

```
adchekmarev@alexandercchekmarev:~/work/arch-pc/lab07$ touch lab7-2.asm
adchekmarev@alexandercchekmarev:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
adchekmarev@alexandercchekmarev:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
adchekmarev@alexandercchekmarev:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 42
Наибольшее число: 50
adchekmarev@alexandercchekmarev:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 199
Наибольшее число: 199
adchekmarev@alexandercchekmarev:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 14
Наибольшее число: 50
adchekmarev@alexandercchekmarev:~/work/arch-pc/lab07$ ./lab7-2
Введите B: -34
Наибольшее число: 50
adchekmarev@alexandercchekmarev:~/work/arch-pc/lab07$ ./lab7-2
Введите B: -51
Наибольшее число: 50
adchekmarev@alexandercchekmarev:~/work/arch-pc/lab07$
```

Рис. 2.10: Рис 2.1.10: Создание файла и проверка работы с разными значениями *B*

Обратим внимание, в данном примере переменные *A* и *C* сравниваются как символы, а переменная *B* и максимум из *A* и *C* как числа (для этого используется функция *atoi* преобразования символа в число). Это сделано для демонстрации того, как сравниваются данные. Данную программу можно упростить и сравнивать все 3 переменные как символы (т.е. не использовать функцию *atoi*). Однако если переменные преобразовать из символов числа, над ними можно корректно проводить арифметические операции.

## 2.2 Изучение структуры файлы листинга

Обычно *nasm* создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ *-l* и задав имя файла листинга в командной строке. Создадим файл листинга для программы из файла *lab7-2.asm*

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
```

Рис. 2.11: Рис 2.2.1: Создание файла

Откроем файл листинга lab7-2.lst с помощью любого текстового редактора, например mcedit:

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ mcedit lab7-2.lst
```

Рис. 2.12: Рис 2.2.2: Демонстрация ввода команды

```

/home/adchekmarev/work/arch-pc/lab07/lab7-2.lst [----] 0 L: [ 1+ 0 1/225] *(0 /14560b) 0032 0
1      %include 'in_out.asm'
2      <1> ;----- slen -----
3      <1> ; Функция вычисления длины сообщения
4      <1> slen:.....
5 00000000 53      <1>      push     ebx.....
6 00000001 89C3    <1>      mov      ebx, eax.....
7
8      <1>.....
9      <1> nextchar:.....
10 00000003 803800 <1>      cmp      byte [eax], 0...
11 00000006 7403   <1>      jz       finished.....
12 00000008 40     <1>      inc      eax.....
13 00000009 EBF8   <1>      jmp      nextchar.....
14      <1>.....
15      <1> finished:
16 0000000B 29D8   <1>      sub      eax, ebx
17 0000000D 5B     <1>      pop      ebx.....
18 0000000E C3     <1>      ret.....
19      <1>.
20      <1>.
21      <1> ;----- sprint -----
22      <1> ; Функция печати сообщения
23      <1> ; входные данные: mov eax,<message>
24      <1> sprint:
25 0000000F 52     <1>      push     edx
26 00000010 51     <1>      push     ecx
27 00000011 53     <1>      push     ebx
28 00000012 50     <1>      push     eax
29 00000013 E8E8FFFF <1>      call     slen
30      <1>.....
31 00000018 89C2   <1>      mov      edx, eax
32 0000001A 58     <1>      pop      eax
33      <1>.....
34 0000001B 89C1   <1>      mov      ecx, eax
35 0000001D B801000000 <1>      mov      ebx, 1
36 00000022 B804000000 <1>      mov      eax, 4
37 00000027 CD80   <1>      int      80h
38      <1>.
39 00000029 5B     <1>      pop      ebx
40 0000002A 59     <1>      pop      ecx
41 0000002B 5A     <1>      pop      edx
42 0000002C C3     <1>      ret
43      <1>.
44      <1>.
45      <1> ;----- sprintf -----
46      <1> ; Функция печати сообщения с переводом строки
47      <1> ; входные данные: mov eax,<message>
48      <1> sprintf:

```

Рис. 2.13: Рис 2.2.3: Вид файла .lst в текст. редакторе

Внимательно ознакомимся с его форматом и содержанием. **Подробно объяснить содержимое трёх строк файла листинга по выбору.**

Возьмем первые 3 строки файла листинга, начиная с 3-ей\*

```

3      <1> ; Функция вычисления длины сообщения
4      <1> slen:.....
5 00000000 53      <1>      push     ebx.....

```

Рис. 2.14: Рис 2.2.4: Демонстрация взятых строк

- 3 - номер строки кода, “; Функция вычисления длины сообщения” - оно не имеет отношения к работе кода, это пояснение.
- 4 - номер строки кода, “slen:.....” - название функции, не имеет адреса и машинного кода.
- 5 - номер строки кода, “00000000” - адрес строки, “53” - машинный код, “push ebx” - исходный текст программы, инструкция “push” помещает операнд “ebx” в стек.

Откроем файл с программой *lab7-2.asm* и в любой инструкции с двумя операндами удалим один операнд. Выполним трансляцию с получением файла листинга:

```

6   mov [max],ecx ; 'max = A'
7 ; ----- Сравниваем 'A' и 'C' (как символы)
8   cmp ecx,[C] ; Сравниваем 'A' и 'C'
9   jg check_B ; если 'A>C', то переход на метку 'check_B',
0   mov ecx,[C] ; иначе 'ecx = C'
1   mov [max],ecx ; 'max = C'
2 ; ----- Преобразование 'max(A,C)' из символа в число

```

Рис. 2.15: Рис 2.2.5: Удаление операнды

**Какие выходные файлы создаются в этом случае? Что добавляется в листинге?**

Выполним трансляцию с измененной программой

```

adchekmarev@alexandercchekmarev:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
adchekmarev@alexandercchekmarev:~/work/arch-pc/lab07$

```

Рис. 2.16: Рис 2.2.6: Трансляция файла

```

27   ; ----- Сравниваем 'A' и 'C'
28   cmp ecx, ; Сравниваем 'A' и 'C'
29   0000011C 7F0C          error: invalid combination of opcode and operands
                           jg check_B ; если 'A>C', то переход на метку 'check_B',

```

Рис. 2.17: Рис 2.2.7: Демонстрация ошибки в файле .lst

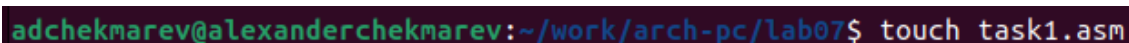


На выходе мы не получаем файла из-за ошибки. **mov** не может работать, имея только один операнд, из-за чего нарушается работа кода

### 3 Самостоятельная работа

***Задание №1 Напишите программу нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу.***

Создадим новый файл *task1.asm* и напомним программу нахождения наименьшей из 3 целочисленных переменных a, b и c для варианта 4 (8, 88, 68).



```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ touch task1.asm
```

Рис. 3.1: Рис 3.1.1: Создание файла

Возьмем за основу код из *lab7-2.asm* и переделаем его

```

1 %include 'in_out.asm'
2 section .data
3     msg db "Наименьшее число: ",0h
4     A dd '8'
5     B dd '88'
6     C dd '68'
7 section .bss
8     min resb 0
9 section .text
10    global _start
11 _start:
12    mov ecx,[A] ; 'ecx = A'
13    mov [min],ecx ; 'min = A'
14
15    cmp ecx,[C] ; Сравниваем 'A' и 'C'
16    jl check_B
17    mov ecx,[C] ; иначе 'ecx = C'
18    mov [min],ecx ; 'min = C'
19
20 check_B:
21    mov eax,min
22    call atoi ; Вызов подпрограммы перевода символа в число
23    mov [min],eax ; запись преобразованного числа в `min`
24
25    mov ecx,[min]
26    cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
27    jl fin ; если 'min(A,C)<B', то переход на 'fin',
28    mov ecx,[B] ; иначе 'ecx = B'
29    mov [min],ecx
30
31 fin:
32    mov eax,msg
33    call sprint ; Вывод сообщения 'Наименьшее число: '
34    mov eax,[min]
35    call iprintLF ; Вывод 'min(A,B,C)'
36    call quit ; Выход

```

Рис. 3.2: Рис 3.1.2: Демонстрация программы для задания

Проверим программу

```

adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ nasm -f elf task1.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ ld -m elf_i386 -o task1 task1.o
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ ./task1
Наименьшее число: 8

```

Рис. 3.3: Рис 3.1.3: Проверка программы

Программа работает как нужно для задания.

**Задание №2** Напишите программу, которая для введенных с клавиатуры значений  $x$  и  $a$  вычисляет значение заданной функции  $f(x)$  и выводит результат вычислений. Вид функции  $f(x)$  выбрать из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу для значений  $x$  и  $a$  из 7.6

Создадим файл *task2.asm* для 2-го задания

```

adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ touch task2.asm

```

Рис. 3.4: Рис 3.2.1: Демонстрация 4-го варианта

Напишем код к решению 4-го варианта

$$4 \quad \begin{cases} 2x + a, & a \neq 0 \\ 2x + 1, & a = 0 \end{cases} \quad (3;0) \quad (3;2)$$

Рис. 3.5: Рис 3.2.2: Демонстрация 4-го варианта

```

1 %include 'in_out.asm'
2 section .data
3
4 msg1 DB 'Введите x: ',0h
5 msg2 DB "Введите a: ",0h
6 otv: DB 'F(x)=',0h
7
8 section .bss
9 x: RESB 10
10 a: RESB 10
11 res: RESB 10
12
13 section .text
14 global _start
15 _start:
16
17 mov eax, msg1
18 call sprint
19 mov ecx,x
20 mov edx,10
21 call sread
22 mov eax,x
23 call atoi
24 mov [x],eax
25
26 mov eax,msg2
27 call sprint
28 mov ecx,a
29 mov edx,10
30 call sread
31 mov eax,a
32 call atoi
33 mov [a],eax
34
35 mov ecx, [a]
36 cmp ecx, 0
37 je x_is_0
38 mov eax, [x]
39 mov ebx,2
40 mul ebx
41 add eax, ecx
42 jmp calc_res
43 x_is_0:
44 mov ebx,2
45 mov eax, [x]
46 mul ebx
47 inc eax
48 calc_res:
49 mov [res],eax
50 fin:
51 mov eax,otv
52 call sprint

```

Рис. 3.6: Рис 3.2.3: (1)Программа для задачи

```
53 mov eax,[res]
54 call iprintLF
55 call quit
```

Рис. 3.7: Рис 3.2.4: (2)Программа для задачи

Проверим работу программы

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ nasm -f elf task2.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ ld -m elf_i386 -o task2 task2.o
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ ./task2
Введите x: 3
Введите a: 0
F(x)=7
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$ ./task2
Введите x: 3
Введите a: 2
F(x)=8
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab07$
```

Рис. 3.8: Рис 3.2.5: Проверка программы

Как видно по (рис 3.2.5) программа работает корректно и я выполнил задание  
Загрузим все файлы на github по окончании лаб. работы

```

adchekmarev@galexanderchekmarev:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git add .
adchekmarev@galexanderchekmarev:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git commit -am 'feat(main): add files lab-7'
[master 256d4d5] feat(main): add files lab-7
29 files changed, 208 insertions(+), 119 deletions(-)
delete mode 100644 labs/lab07/report/image/placeimg_800_600_tech.jpg
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.1.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.10.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.2.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.3.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.4.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.5.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.6.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.7.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.8.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.9.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.2.1.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.2.2.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.2.3.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.2.4.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.2.5.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.2.6.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.2.7.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 3.1.1.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 3.1.2.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 3.1.3.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 3.2.1.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 3.2.2.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 3.2.3.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 3.2.4.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 3.2.5.png"
create mode 100644 labs/lab07/report/report.docx
rewrite labs/lab07/report/report.md (71%)
create mode 100644 labs/lab07/report/report.pdf
adchekmarev@galexanderchekmarev:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git push
Перечисление объектов: 40, готово.
Подсчет объектов: 100% (40/40), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (34/34), готово.
Запись объектов: 100% (34/34), 2.63 МиБ | 2.66 МиБ/с, готово.
Всего 34 (изменения 2), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:nenokixd/study_2023-2024_arh-pc.git
29f8b96..256d4d5 master -> master
adchekmarev@galexanderchekmarev:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$

```

Рис. 3.9: Рис 3.3.1: Загрузка файлов на github

## 4 Выводы

Я изучил команды условного и безусловного переходов. Приобрел навыки написания программ с использованием переходов. Познакомился с назначением и структурой файла листинга.