Coa Problem Statements

1.Addition of two integers:

Code:-

```
section .bss
    num1   resb 20
    num2   resb 20
    result resb 20

section .data
    msg1    db "Enter first number: ", 0
    msg2    db "Enter second number: ", 0
    msg3    db "Sum = ", 0
    newline db 10, 0

section .text
    global _start

_start:
    ;-----------------------------------
    ; Ask for first number
    ;-----------------------------------
```

```
mov rax, 1          ; sys_write
mov rdi, 1
mov rsi, msg1
mov rdx, 20
syscall

mov rax, 0          ; sys_read
mov rdi, 0
mov rsi, num1
mov rdx, 20
syscall

;-----------------------------------
; Ask for second number
;-----------------------------------
mov rax, 1
mov rdi, 1
mov rsi, msg2
mov rdx, 21
syscall

mov rax, 0
```

```
mov rdi, 0
mov rsi, num2
mov rdx, 20
syscall

;-----------------------------------
; Convert ASCII -> integer
;-----------------------------------
mov rsi, num1
call atoi
mov rbx, rax        ; store first number

mov rsi, num2
call atoi
add rax, rbx        ; ALU ADD
mov rdi, result
call itoa           ; convert result to string
mov rsi, rdi        ; rsi now points to string start

;-----------------------------------
; Print "Sum = "
;-----------------------------------
```

```
mov rax, 1
mov rdi, 1
mov rsi, msg3
mov rdx, 6
syscall

;-----------------------------------
; Print result (until NULL)
;-----------------------------------
mov rax, 1
mov rdi, 1
mov rsi, result
call strlen          ; find length
mov rdx, rax         ; length in rdx
mov rax, 1           ; sys_write
mov rdi, 1
mov rsi, result
syscall

; newline
mov rax, 1
mov rdi, 1
```

```
    mov rsi, newline
    mov rdx, 1
    syscall

    ; exit
    mov rax, 60
    xor rdi, rdi
    syscall



;-----------------------------------
; atoi: ASCII -> integer
;-----------------------------------
atoi:
    xor rax, rax
    xor rcx, rcx
.next:
    movzx rdx, byte [rsi + rcx]
    cmp rdx, 10          ; newline?
    je .done
    cmp rdx, 13
    je .done
```

```
    cmp rdx, 0
    je .done
    sub rdx, '0'
    imul rax, rax, 10
    add rax, rdx
    inc rcx
    jmp .next
.done:
    ret



;-----------------------------------
; itoa: integer -> ASCII
;-----------------------------------
itoa:
    mov rcx, 10
    lea rbx, [rdi + 19]
    mov byte [rbx], 0
.convert:
    xor rdx, rdx
    div rcx
    add dl, '0'
```

```
    dec rbx
    mov [rbx], dl
    test rax, rax
    jnz .convert
    ; Copy string to start of buffer
    mov rsi, rbx
.copy:
    mov al, [rsi]
    mov [rdi], al
    inc rsi
    inc rdi
    test al, al
    jnz .copy
    ret


;-----------------------------------
; strlen: find length of NULL-terminated string
; input: RSI=string
; output: RAX=length
;-----------------------------------
strlen:
```

```
    xor rax, rax
.lenloop:
    cmp byte [rsi + rax], 0
    je .done
    inc rax
    jmp .lenloop
.done:
    Ret
```

Output:

2.Subtraction:

Code:

```
section .bss
    num1   resb 20
    num2   resb 20
```

```
    result resb 20

section .data
    msg1    db "Enter first number: ", 0
    msg2    db "Enter second number: ", 0
    msg3    db "Difference = ", 0
    newline db 10, 0

section .text
    global _start

_start:
    ;-----------------------------------
    ; Ask for first number
    ;-----------------------------------
    mov rax, 1              ; sys_write
    mov rdi, 1
    mov rsi, msg1
    mov rdx, 20
    syscall

    ; read num1
```

```
mov rax, 0
mov rdi, 0
mov rsi, num1
mov rdx, 20
syscall

;-----------------------------------
; Ask for second number
;-----------------------------------
mov rax, 1
mov rdi, 1
mov rsi, msg2
mov rdx, 21
syscall

; read num2
mov rax, 0
mov rdi, 0
mov rsi, num2
mov rdx, 20
syscall
```

```
;-----------------------------------
; Convert both ASCII -> integer
;-----------------------------------
mov rsi, num1
call atoi
mov rbx, rax          ; store first number in RBX

mov rsi, num2
call atoi
sub rbx, rax          ; ALU SUB: RBX = num1 - num2
mov rax, rbx          ; result in RAX

;-----------------------------------
; Convert result integer -> ASCII
;-----------------------------------
mov rdi, result
call itoa

;-----------------------------------
; Print result
;-----------------------------------
mov rax, 1
```

```
mov rdi, 1
mov rsi, msg3
mov rdx, 14          ; length of "Difference = "
syscall

mov rax, 1
mov rdi, 1
mov rsi, result
call strlen
mov rdx, rax
mov rax, 1
mov rdi, 1
mov rsi, result
syscall

; print newline
mov rax, 1
mov rdi, 1
mov rsi, newline
mov rdx, 1
syscall
```

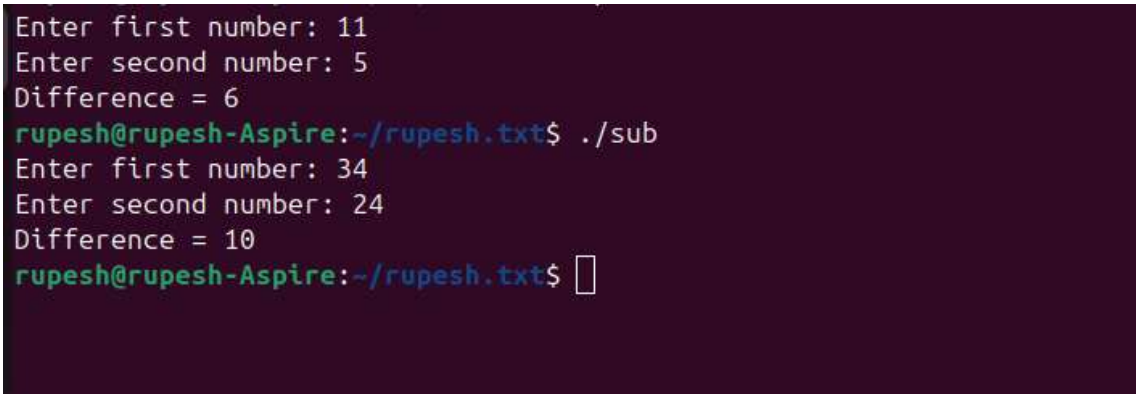
```
    ; exit
    mov rax, 60
    xor rdi, rdi
    syscall


;-----------------------------------
; atoi: ASCII -> integer
;-----------------------------------
atoi:
    xor rax, rax
    xor rcx, rcx
.next:
    movzx rdx, byte [rsi + rcx]
    cmp rdx, 10          ; newline?
    je .done
    cmp rdx, 13
    je .done
    cmp rdx, 0
    je .done
    sub rdx, '0'
    imul rax, rax, 10
```

```
    add rax, rdx
    inc rcx
    jmp .next
.done:
    ret



;-----------------------------------
; itoa: integer -> ASCII
;-----------------------------------
itoa:
    mov rcx, 10
    lea rbx, [rdi + 19]
    mov byte [rbx], 0
.convert:
    xor rdx, rdx
    div rcx
    add dl, '0'
    dec rbx
    mov [rbx], dl
    test rax, rax
    jnz .convert
```

```
    mov rsi, rbx
.copy:
    mov al, [rsi]
    mov [rdi], al
    inc rsi
    inc rdi
    test al, al
    jnz .copy
    ret



;-----------------------------------
; strlen: get string length
;-----------------------------------
strlen:
    xor rax, rax
.lenloop:
    cmp byte [rsi + rax], 0
    je .done
    inc rax
    jmp .lenloop
.done:
```

Ret

Output:



3.Multiplication:

Code:

```
section .bss
   num1   resb 20
   num2   resb 20
   result resb 40         ; bigger buffer for large results

section .data
   msg1    db "Enter first number: ", 0
   msg2    db "Enter second number: ", 0
   msg3    db "Product = ", 0
   newline db 10, 0
```

```
section .text
    global _start

_start:
    ;-----------------------------------
    ; Ask for first number
    ;-----------------------------------
    mov rax, 1              ; sys_write
    mov rdi, 1
    mov rsi, msg1
    mov rdx, 20
    syscall

    ; read num1
    mov rax, 0              ; sys_read
    mov rdi, 0
    mov rsi, num1
    mov rdx, 20
    syscall

    ;-----------------------------------
```

```
; Ask for second number
;-----------------------------------
mov rax, 1
mov rdi, 1
mov rsi, msg2
mov rdx, 21
syscall

; read num2
mov rax, 0
mov rdi, 0
mov rsi, num2
mov rdx, 20
syscall

;-----------------------------------
; Convert ASCII -> integer
;-----------------------------------
mov rsi, num1
call atoi
mov rbx, rax          ; save first number
```

```
mov rsi, num2
call atoi
imul rbx, rax          ; ALU IMUL: rbx = num1 * num2
mov rax, rbx

;-------------------------------------
; Convert result to ASCII
;-------------------------------------
mov rdi, result
call itoa

;-------------------------------------
; Print result
;-------------------------------------
mov rax, 1
mov rdi, 1
mov rsi, msg3
mov rdx, 10
syscall

mov rax, 1
mov rdi, 1
```

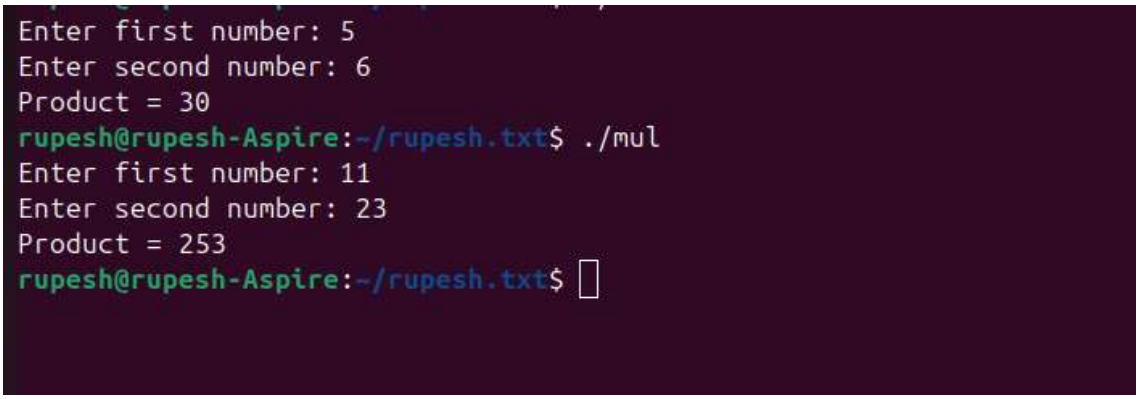
```
    mov rsi, result
    call strlen
    mov rdx, rax
    mov rax, 1
    mov rdi, 1
    mov rsi, result
    syscall

    ; newline
    mov rax, 1
    mov rdi, 1
    mov rsi, newline
    mov rdx, 1
    syscall

    ; exit
    mov rax, 60
    xor rdi, rdi
    syscall


;-----------------------------------
```

```
; atoi: ASCII -> integer
;-----------------------------------
atoi:
    xor rax, rax
    xor rcx, rcx
.next:
    movzx rdx, byte [rsi + rcx]
    cmp rdx, 10          ; newline?
    je .done
    cmp rdx, 13
    je .done
    cmp rdx, 0
    je .done
    sub rdx, '0'
    imul rax, rax, 10
    add rax, rdx
    inc rcx
    jmp .next
.done:
    ret
```

```
;-----------------------------------
; itoa: integer -> ASCII
;-----------------------------------
itoa:
    mov rcx, 10
    lea rbx, [rdi + 39]   ; use larger buffer end
    mov byte [rbx], 0
.convert:
    xor rdx, rdx
    div rcx
    add dl, '0'
    dec rbx
    mov [rbx], dl
    test rax, rax
    jnz .convert
    mov rsi, rbx
.copy:
    mov al, [rsi]
    mov [rdi], al
    inc rsi
    inc rdi
    test al, al
```

```
    jnz .copy
    ret



;-----------------------------------
; strlen: get string length
;-----------------------------------
strlen:
    xor rax, rax
.lenloop:
    cmp byte [rsi + rax], 0
    je .done
    inc rax
    jmp .lenloop
.done:
    Ret
```

Output:



4.Division:

Code:

```
section .bss
   num1   resb 20
   num2   resb 20
   result resb 40        ; bigger buffer for large results

section .data
   msg1    db "Enter first number: ", 0
   msg2    db "Enter second number: ", 0
   msg3    db "Product = ", 0
   newline db 10, 0

section .text
   global _start
```

```
_start:
    ;-----------------------------------
    ; Ask for first number
    ;-----------------------------------
    mov rax, 1              ; sys_write
    mov rdi, 1
    mov rsi, msg1
    mov rdx, 20
    syscall

    ; read num1
    mov rax, 0              ; sys_read
    mov rdi, 0
    mov rsi, num1
    mov rdx, 20
    syscall

    ;-----------------------------------
    ; Ask for second number
    ;-----------------------------------
    mov rax, 1
```

```
mov rdi, 1
mov rsi, msg2
mov rdx, 21
syscall

; read num2
mov rax, 0
mov rdi, 0
mov rsi, num2
mov rdx, 20
syscall

;-----------------------------------
; Convert ASCII -> integer
;-----------------------------------
mov rsi, num1
call atoi
mov rbx, rax          ; save first number

mov rsi, num2
call atoi
imul rbx, rax         ; ALU IMUL: rbx = num1 * num2
```

```
mov rax, rbx

;-----------------------------------
; Convert result to ASCII
;-----------------------------------
mov rdi, result
call itoa

;-----------------------------------
; Print result
;-----------------------------------
mov rax, 1
mov rdi, 1
mov rsi, msg3
mov rdx, 10
syscall

mov rax, 1
mov rdi, 1
mov rsi, result
call strlen
mov rdx, rax
```

```
    mov rax, 1
    mov rdi, 1
    mov rsi, result
    syscall

    ; newline
    mov rax, 1
    mov rdi, 1
    mov rsi, newline
    mov rdx, 1
    syscall

    ; exit
    mov rax, 60
    xor rdi, rdi
    syscall



;-----------------------------------
; atoi: ASCII -> integer
;-----------------------------------
atoi:
```

```
    xor rax, rax
    xor rcx, rcx
.next:
    movzx rdx, byte [rsi + rcx]
    cmp rdx, 10         ; newline?
    je .done
    cmp rdx, 13
    je .done
    cmp rdx, 0
    je .done
    sub rdx, '0'
    imul rax, rax, 10
    add rax, rdx
    inc rcx
    jmp .next
.done:
    ret



;-----------------------------------
; itoa: integer -> ASCII
;-----------------------------------
```

```
itoa:
    mov rcx, 10
    lea rbx, [rdi + 39]   ; use larger buffer end
    mov byte [rbx], 0
.convert:
    xor rdx, rdx
    div rcx
    add dl, '0'
    dec rbx
    mov [rbx], dl
    test rax, rax
    jnz .convert
    mov rsi, rbx
.copy:
    mov al, [rsi]
    mov [rdi], al
    inc rsi
    inc rdi
    test al, al
    jnz .copy
    ret
```

```
;-----------------------------------
; strlen: get string length
;-----------------------------------
strlen:
    xor rax, rax
.lenloop:
    cmp byte [rsi + rax], 0
    je .done
    inc rax
    jmp .lenloop
.done:
    Ret
```

Output:

5.Multiplication of two 64-bit hexadecimal numbers using Successive addition:

Code:

```
%macro write 2
   mov rax, 1
   mov rdi, 1
   mov rsi, %1
   mov rdx, %2
   syscall
%endmacro

%macro read 2
   mov rax, 0
   mov rdi, 0
   mov rsi, %1
   mov rdx, %2
   syscall
%endmacro

section .data
   msg1 db "Enter the first number: ",10
   msg1_len equ $-msg1
```

```
    msg2 db "Enter the second number: ",10
    msg2_len equ $-msg2
    msg3 db "Multiplication Result: ",10
    msg3_len equ $-msg3
    msg_space db 10
    msg_space_len equ $-msg_space

section .bss
    num     resb 17
    buff    resb 16
    ccnt    resq 1
    no1     resq 1
    no2     resq 1

section .text
    global _start
_start:
    write msg1, msg1_len
    read num, 17
    dec rax
    mov [ccnt], rax
    call accept
```

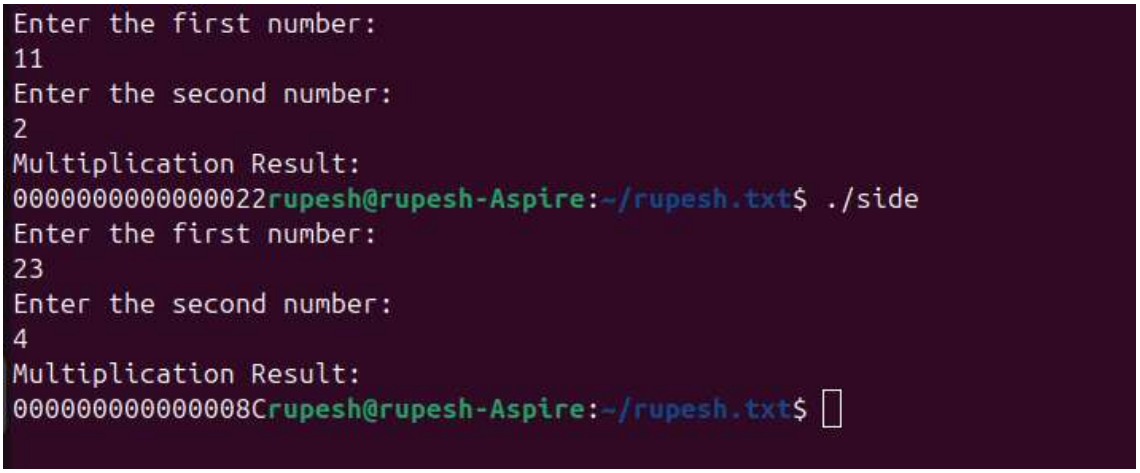
```
    mov [no1], rbx

    write msg2, msg2_len
    read num, 17
    dec rax
    mov [ccnt], rax
    call accept
    mov [no2], rbx

    mov rbx, 0          ; result accumulator

multiply_loop:
    add rbx, [no1]
    dec qword [no2]
    cmp qword [no2], 0
    jne multiply_loop

    write msg3, msg3_len
    call disp

    ; exit
    mov rax, 60
```

```
    xor rdi, rdi
    syscall



;-------------------------------------
; accept: convert ASCII -> integer (hex)
;-------------------------------------
accept:
    mov rbx, 0
    mov rsi, num
up1:
    shl rbx, 4
    mov dl, [rsi]
    cmp dl, 39h
    jbe sub_30
    sub dl, 7
sub_30:
    sub dl, 30h
    add rbx, rdx
    inc rsi
    dec qword [ccnt]
    jnz up1
```

```
    ret

;-------------------------------------
; disp: print rbx in hex (16 chars)
;-------------------------------------
disp:
    mov rsi, buff
    mov rcx, 16
up2:
    rol rbx, 4
    mov dl, bl
    and dl, 0Fh
    cmp dl, 9
    jbe mc
    add dl, 7
mc:
    add dl, 30h
    mov [rsi], dl
    inc rsi
    dec rcx
    jnz up2
    write buff, 16
```

ret

output:



6. Multiplication of two 64-bit hexadecimal numbers using Add and shift method:

Code:

```
%macro write 2
    mov rax, 1
    mov rdi, 1
    mov rsi, %1
    mov rdx, %2
    syscall
%endmacro
```

```
%macro read 2
    mov rax, 0
    mov rdi, 0
    mov rsi, %1
    mov rdx, %2
    syscall
%endmacro

section .data
    msg1 db "Enter the multiplicand: ",10
    msg1_len equ $-msg1
    msg2 db "Enter the multiplier: ",10
    msg2_len equ $-msg2
    msg3 db "Product Result (A then Q):",10
    msg3_len equ $-msg3

section .bss
    num     resb 17
    buff    resb 16
    ccnt    resq 1
    A       resq 1
    B       resq 1
```

```
    Q       resq 1
    n       resq 1

section .text
    global _start

_start:
    write msg1, msg1_len
    read num, 17
    dec rax
    mov [ccnt], rax
    call accept
    mov [B], rbx

    write msg2, msg2_len
    read num, 17
    dec rax
    mov [ccnt], rax
    call accept
    mov [Q], rbx

    mov qword [A], 0
```

```
    mov qword [n], 64      ; 64-bit loop

mul_loop:
    mov rax, [Q]
    and rax, 1
    cmp rax, 1
    jne shift_step

    mov rax, [A]
    mov rbx, [B]
    add rax, rbx
    mov [A], rax

shift_step:
    mov rax, [A]
    mov rbx, [Q]
    shr rbx, 1
    and rax, 1
    cmp rax, 1
    jne shift_a
    mov rdx, 1
    ror rdx, 1
```

```
    or  rbx, rdx
shift_a:
    mov rax, [A]
    shr rax, 1
    mov [A], rax
    mov [Q], rbx

    dec qword [n]
    jnz mul_loop

    write msg3, msg3_len
    mov rbx, [A]
    call disp
    mov rbx, [Q]
    call disp

    ; exit
    mov rax, 60
    xor rdi, rdi
    syscall
```

```
;-------------------------------------
; accept: convert ASCII -> integer (hex)
;-------------------------------------
accept:
   mov rbx, 0
   mov rsi, num
up1:
   shl rbx, 4
   mov dl, [rsi]
   cmp dl, 39h
   jbe sub_30
   sub dl, 7
sub_30:
   sub dl, 30h
   add rbx, rdx
   inc rsi
   dec qword [ccnt]
   jnz up1
   ret


;-------------------------------------
; disp: print rbx in hex (16 chars)
```

```
;-------------------------------------
disp:
   mov rsi, buff
   mov rcx, 16
up2:
   rol rbx, 4
   mov dl, bl
   and dl, 0Fh
   cmp dl, 9
   jbe mc
   add dl, 7
mc:
   add dl, 30h
   mov [rsi], dl
   inc rsi
   dec rcx
   jnz up2
   write buff, 16
   ret
cd

output:
```

```
Enter the multiplicand:
23
Enter the multiplier:
4
Product Result (A then Q):
0000000000000000000000000000008Crupesh@rupesh-Aspire:~/rupesh.txt$ ./add
Enter the multiplicand:
445
Enter the multiplier:
3
Product Result (A then Q):
000000000000000000000000000CCFrupesh@rupesh-Aspire:~/rupesh.txt$
```

7.String operation:

Code:

```
; ==========================================
; STRING OPERATIONS MENU - x86_64 LINUX
; ==========================================


%macro read 2
    mov rax, 0
    mov rdi, 0
    mov rsi, %1
    mov rdx, %2
    syscall
%endmacro

%macro write 2
```

```
    mov rax, 1
    mov rdi, 1
    mov rsi, %1
    mov rdx, %2
    syscall
%endmacro

section .data
Menumsg db 10, "1. String length",10
        db "2. String copy",10
        db "3. String reverse",10
        db "4. String compare",10
        db "5. String concat",10
        db "6. Check palindrome",10
        db "7. String substring",10
        db "8. Exit",10
        db "Enter your choice (1-8): ",10
Menulen equ $ - Menumsg

msg1 db "Enter String1:",10
len1 equ $ - msg1
```

```
msg2 db "Enter String2:",10
len2 equ $ - msg2

msg3 db "The length of string: ",10
len3 equ $ - msg3

msg4 db "The copied string:",10
len4 equ $ - msg4

msg5 db "The reversed string:",10
len5 equ $ - msg5

msg6 db "Strings are equal",10
len6 equ $ - msg6

msg7 db "Strings are not equal",10
len7 equ $ - msg7

msg8 db "The concatenated string:",10
len8 equ $ - msg8

msg9 db "String is palindrome",10
```

```
len9 equ $ - msg9

msg10 db "String is not palindrome",10
len10 equ $ - msg10

msg11 db "Substring found",10
len11 equ $ - msg11

msg12 db "Not a substring",10
len12 equ $ - msg12

msg13 db "Wrong choice",10
len13 equ $ - msg13

section .bss
string1 resb 40
string2 resb 40
string3 resb 80
l1      resq 1
l2      resq 1
l3      resq 1
choice  resb 2
```

```
buff    resb 16

section .text
global _start

_start:
    ; --- Input two strings first ---
    write msg1, len1
    read string1, 40
    dec rax
    mov [l1], rax

    write msg2, len2
    read string2, 40
    dec rax
    mov [l2], rax

printmenu:
    write Menumsg, Menulen
    read choice, 2

    cmp byte [choice], '1'
```

```
    je strlen
    cmp byte [choice], '2'
    je strcpy
    cmp byte [choice], '3'
    je strrev
    cmp byte [choice], '4'
    je strcmp
    cmp byte [choice], '5'
    je strcat
    cmp byte [choice], '6'
    je strpalindrome
    cmp byte [choice], '7'
    je strsub
    cmp byte [choice], '8'
    je exit

    write msg13, len13
    jmp printmenu

; ----------------------------------------
; 1. String Length
; ----------------------------------------
```

```
strlen:
    write msg3, len3
    mov rbx, [l1]
    call display
    jmp printmenu

; ----------------------------------------
; 2. String Copy
; ----------------------------------------
strcpy:
    mov rsi, string1
    mov rdi, string3
    mov rcx, [l1]
    cld
    rep movsb
    write msg4, len4
    write string3, [l1]
    jmp printmenu

; ----------------------------------------
; 3. String Reverse
; ----------------------------------------
```

```
strrev:
    mov rsi, string1
    add rsi, [l1]
    dec rsi
    mov rdi, string3
    mov rcx, [l1]
rev_loop:
    mov bl, [rsi]
    mov [rdi], bl
    dec rsi
    inc rdi
    loop rev_loop
    write msg5, len5
    write string3, [l1]
    jmp printmenu

; ----------------------------------------
; 4. String Compare
; ----------------------------------------
strcmp:
    mov rbx, [l1]
    cmp rbx, [l2]
```

```
    jne nonequal
    mov rsi, string1
    mov rdi, string2
    mov rcx, [l1]
    cld
    repe cmpsb
    jne nonequal
    write msg6, len6
    jmp printmenu
nonequal:
    write msg7, len7
    jmp printmenu

; ------------------------------------------
; 5. String Concat
; ------------------------------------------
strcat:
    mov rsi, string1
    mov rdi, string3
    mov rcx, [l1]
    cld
    rep movsb
```

```
    mov rsi, string2
    mov rcx, [l2]
    rep movsb
    mov rbx, [l1]
    add rbx, [l2]
    mov [l3], rbx
    write msg8, len8
    write string3, [l3]
    jmp printmenu

; ----------------------------------------
; 6. Palindrome Check
; ----------------------------------------
strpalindrome:
    mov rsi, string1
    add rsi, [l1]
    dec rsi
    mov rdi, string3
    mov rcx, [l1]
pal_loop:
    mov dl, [rsi]
    mov [rdi], dl
```

```
    dec rsi
    inc rdi
    loop pal_loop
    mov rsi, string1
    mov rdi, string3
    mov rcx, [l1]
    cld
    repe cmpsb
    jne notpal
    write msg9, len9
    jmp printmenu
notpal:
    write msg10, len10
    jmp printmenu

; ----------------------------------------
; 7. Substring Check (Naive)
; ----------------------------------------
strsub:
    mov rbx, [l1]
    mov rcx, [l2]
    mov rsi, string1
```

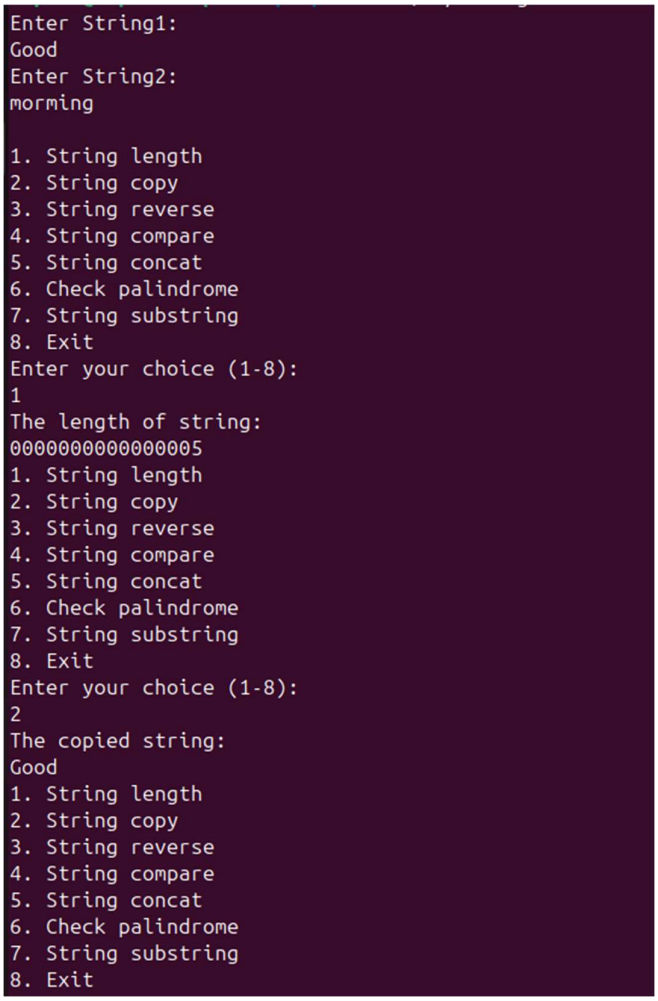
```
outer:
    mov rdi, string2
    mov rdx, rcx
inner:
    mov al, [rsi]
    mov bl, [rdi]
    cmp al, bl
    jne notmatch
    inc rsi
    inc rdi
    dec rdx
    jz found
    jmp inner
notmatch:
    inc rsi
    dec rbx
    jnz outer
    write msg12, len12
    jmp printmenu
found:
    write msg11, len11
    jmp printmenu
```

```
; ------------------------------------------
; Exit
; ------------------------------------------
exit:
    mov rax, 60
    xor rdi, rdi
    syscall

; ------------------------------------------
; Display decimal number in hex
; ------------------------------------------
display:
    mov rsi, buff
    mov rcx, 16
disp_loop:
    rol rbx, 4
    mov dl, bl
    and dl, 0Fh
    cmp dl, 9
    jbe add30
    add dl, 7
```

add30:

```
    add dl, 30h
    mov [rsi], dl
    inc rsi
    loop disp_loop
    write buff, 16
    ret
```

output:



7.Quadratic equation:

Code:

```
%macro write 2
    mov rax, 1
    mov rdi, 1
    mov rsi, %1
    mov rdx, %2
    syscall
%endmacro

%macro read 2
    mov rax, 0
    mov rdi, 0
    mov rsi, %1
    mov rdx, %2
    syscall
%endmacro

section .data
    msg_a     db "Enter coefficient a: ", 0
```

```
    msg_a_len equ $ - msg_a
    msg_b     db "Enter coefficient b: ", 0
    msg_b_len equ $ - msg_b
    msg_c     db "Enter coefficient c: ", 0
    msg_c_len equ $ - msg_c

    msg_r1    db "Root 1: ", 0
    msg_r1_len equ $ - msg_r1
    msg_r2    db "Root 2: ", 0
    msg_r2_len equ $ - msg_r2
    msg_equal db "Equal root: ", 0
    msg_equal_len equ $ - msg_equal
    msg_imag  db "Imaginary roots", 10, 0
    msg_imag_len equ $ - msg_imag

    newline   db 10
    newline_len equ $ - newline

section .bss
    a       resq 1
```

```
    b       resq 1
    c       resq 1
    D       resq 1
    sqrtD   resq 1
    root1   resq 1
    root2   resq 1
    buf     resb 64

section .text
global _start
_start:

    ; ---- Read a ----
    write msg_a, msg_a_len
    read buf, 64
    mov rsi, buf
    call str2int
    mov [a], rax

    ; ---- Read b ----
```

```
write msg_b, msg_b_len
read buf, 64
mov rsi, buf
call str2int
mov [b], rax

; ---- Read c ----
write msg_c, msg_c_len
read buf, 64
mov rsi, buf
call str2int
mov [c], rax

; ---- Compute D = b^2 - 4ac ----
mov rax, [b]
imul rax, rax          ; b^2
mov rcx, rax

mov rax, [a]
imul rax, [c]          ; a*c
```

```
shl rax, 2              ; 4ac
sub rcx, rax            ; D = b^2 - 4ac
mov [D], rcx

; ---- Check D ----
cmp rcx, 0
jl .imaginary
je .equal

; ---- D > 0 ----
mov rax, [D]
call int_sqrt
mov [sqrtD], rax

; root1 = (-b - sqrt(D)) / (2a)
mov rax, [b]
neg rax
sub rax, [sqrtD]
mov rbx, [a]
shl rbx, 1
```

```
cqo
idiv rbx
mov [root1], rax

; root2 = (-b + sqrt(D)) / (2a)
mov rax, [b]
neg rax
add rax, [sqrtD]
mov rbx, [a]
shl rbx, 1
cqo
idiv rbx
mov [root2], rax

; ---- Print roots ----
write msg_r1, msg_r1_len
mov rax, [root1]
call int2str
write rsi, rdx
write newline, newline_len
```

```
    write msg_r2, msg_r2_len
    mov rax, [root2]
    call int2str
    write rsi, rdx
    write newline, newline_len

    jmp .exit

.equal:
    mov rax, [b]
    neg rax
    mov rbx, [a]
    shl rbx, 1
    cqo
    idiv rbx
    mov [root1], rax

    write msg_equal, msg_equal_len
    mov rax, [root1]
```

```
    call int2str
    write rsi, rdx
    write newline, newline_len
    jmp .exit

.imaginary:
    write msg_imag, msg_imag_len

.exit:
    mov rax, 60
    xor rdi, rdi
    syscall

; ---------------------------------
; Integer square root (simple method)
; RAX = input, RAX = sqrt(output)
; ---------------------------------
int_sqrt:
    cmp rax, 0
    jle .done_sqrt
```

```
    mov rcx, rax
    xor rbx, rbx
.sqrt_loop:
    mov rdx, rbx
    imul rdx, rbx
    cmp rdx, rcx
    ja .found
    inc rbx
    jmp .sqrt_loop
.found:
    dec rbx
    mov rax, rbx
.done_sqrt:
    ret

; ---------------------------------
; str2int (supports negative numbers)
; RSI -> string
; Returns RAX = integer value
; ---------------------------------
```

```
str2int:
    xor rax, rax        ; result = 0
    xor rcx, rcx        ; digit holder
    xor r8b, r8b        ; sign flag = 0

    mov bl, [rsi]
    cmp bl, '-'
    jne .parse
    mov r8b, 1          ; negative flag
    inc rsi

.parse:
    mov bl, [rsi]
    cmp bl, 10          ; newline
    je .done
    cmp bl, 0
    je .done
    sub bl, '0'
    movzx rcx, bl       ; rcx = current digit
    imul rax, rax, 10
```

```
    add rax, rcx
    inc rsi
    jmp .parse

.done:
    cmp r8b, 1
    jne .ret
    neg rax
.ret:
    ret

; ---------------------------------
; str2int (supports negative numbers)
; RSI -> string, RAX = result
; ---------------------------------
str2int:
    xor rax, rax
    xor rbx, rbx
    mov bl, [rsi]
    cmp bl, '-'
```

```
    jne .loop
    mov r8b, 1
    inc rsi
    jmp .loop
.loop:
    mov bl, [rsi]
    cmp bl, 10
    je .end
    cmp bl, 0
    je .end
    sub bl, '0'
    imul rax, 10
    add rax, rbx
    inc rsi
    jmp .loop
.end:
    cmp r8b, 1
    jne .done
    neg rax
.done:
```

Ret

Output:

```
rupesh@rupesh-Aspire:~$ cd rupesh.txt
rupesh@rupesh-Aspire:~/rupesh.txt$ nasm -f elf64 quad1.asm -o quad1.o
rupesh@rupesh-Aspire:~/rupesh.txt$ ld quad1.o -o quad1
rupesh@rupesh-Aspire:~/rupesh.txt$ ./quad1
Enter coefficient a: 1
Enter coefficient b: 5
Enter coefficient c: 6
Root 1: -3
Root 2: -2
rupesh@rupesh-Aspire:~/rupesh.txt$ ./quad1

Enter coefficient a: 1
Enter coefficient b: 2
Enter coefficient c: 1
Equal root: -1
rupesh@rupesh-Aspire:~/rupesh.txt$ ./quad1
Enter coefficient a: 3
Enter coefficient b: 2
Enter coefficient c: 1
Imaginary roots
rupesh@rupesh-Aspire:~/rupesh.txt$ ./quad1
Enter coefficient a: 1
Enter coefficient b: -3
Enter coefficient c: 2
Root 1: 0
Root 2: -3
rupesh@rupesh-Aspire:~/rupesh.txt$ ./quad1
Enter coefficient a: 1
Enter coefficient b: -3
Enter coefficient c: 2
Root 1: 0
Root 2: -3
rupesh@rupesh-Aspire:~/rupesh.txt$ ./quad1
Enter coefficient a: 1
Enter coefficient b: 4
Enter coefficient c: 5
Imaginary roots
rupesh@rupesh-Aspire:~/rupesh.txt$
```

9.String length:

Code:

```
%macro read 2
   mov rax, 0
   mov rdi, 0
   mov rsi, %1
   mov rdx, %2
   syscall
%endmacro

%macro write 2
   mov rax, 1
   mov rdi, 1
   mov rsi, %1
   mov rdx, %2
   syscall
%endmacro
; --- String Length Program ---
section .data
msg1 db "Enter a string:",10
len1 equ $ - msg1
msg2 db "Length of string: ",10
```

```
len2 equ $ - msg2

section .bss
string resb 40
strlen_val resq 1
buff resb 16

section .text
global _start


_start:
   write msg1, len1
   read string, 40
   dec rax              ; exclude newline
   mov [strlen_val], rax

   write msg2, len2
   mov rbx, [strlen_val]
   call display
   call exit
```

```
display:
    mov rsi, buff
    mov rcx, 16
disp_loop:
    rol rbx, 4
    mov dl, bl
    and dl, 0Fh
    cmp dl, 9
    jbe add30
    add dl, 7
add30:
    add dl, 30h
    mov [rsi], dl
    inc rsi
    loop disp_loop
    write buff, 16
    ret

exit:
    mov rax, 60
    xor rdi, rdi
    syscall
```

output:

10.String Copy:

Code:

```
%macro read 2
   mov rax, 0
   mov rdi, 0
   mov rsi, %1
   mov rdx, %2
   syscall
%endmacro

%macro write 2
   mov rax, 1
   mov rdi, 1
   mov rsi, %1
   mov rdx, %2
   syscall
%endmacro
```

```
section .data
msg1 db "Enter a string:",10
len1 equ $ - msg1
msg2 db "Copied string:",10
len2 equ $ - msg2

section .bss
src resb 40
dest resb 40
len resq 1

section .text
global _start


_start:
    write msg1, len1
    read src, 40
    dec rax
    mov [len], rax

    mov rsi, src
```

```
    mov rdi, dest
    mov rcx, [len]
    cld
    rep movsb

    write msg2, len2
    write dest, [len]
    call exit

exit:
    mov rax, 60
    xor rdi, rdi
    syscall
```

Output:

11.String Concatenation:

Cod:

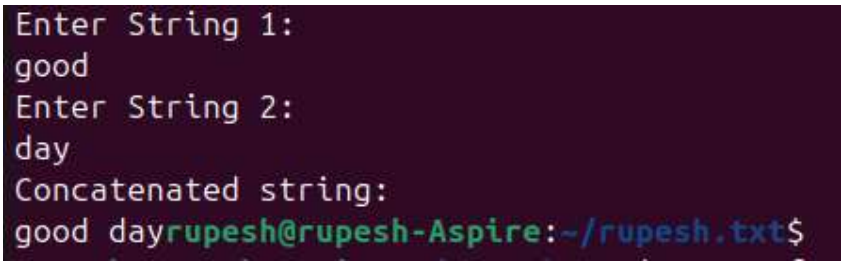
```
%macro read 2
```

```
    mov rax, 0
    mov rdi, 0
    mov rsi, %1
    mov rdx, %2
    syscall
%endmacro

%macro write 2
    mov rax, 1
    mov rdi, 1
    mov rsi, %1
    mov rdx, %2
    syscall
%endmacro
section .data
msg1 db "Enter String 1:",10
len1 equ $ - msg1
msg2 db "Enter String 2:",10
len2 equ $ - msg2
msg3 db "Concatenated string:",10
len3 equ $ - msg3
```

```
section .bss
s1 resb 40
s2 resb 40
dest resb 80
l1 resq 1
l2 resq 1


section .text
global _start


_start:
    write msg1, len1
    read s1, 40
    dec rax
    mov [l1], rax

    write msg2, len2
    read s2, 40
    dec rax
    mov [l2], rax
```

```
    mov rsi, s1
    mov rdi, dest
    mov rcx, [l1]
    cld
    rep movsb

    mov rsi, s2
    mov rcx, [l2]
    rep movsb

    mov rbx, [l1]
    add rbx, [l2]

    write msg3, len3
    write dest, rbx
    call exit

exit:
    mov rax, 60
    xor rdi, rdi
    syscall
```

Output:



12.String comparison:

Code:

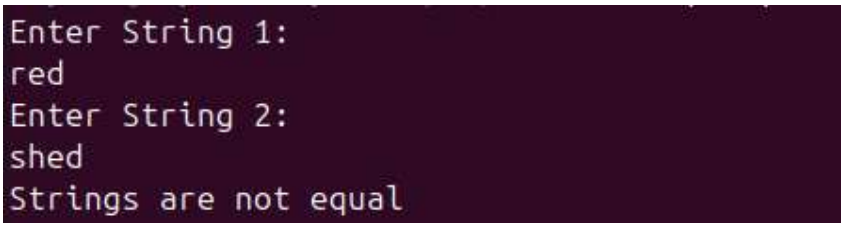
```
%macro read 2
    mov rax, 0
    mov rdi, 0
    mov rsi, %1
    mov rdx, %2
    syscall
%endmacro

%macro write 2
    mov rax, 1
    mov rdi, 1
    mov rsi, %1
    mov rdx, %2
    syscall
```

```
%endmacro
section .data
msg1 db "Enter String 1:",10
len1 equ $ - msg1
msg2 db "Enter String 2:",10
len2 equ $ - msg2
eqmsg db "Strings are equal",10
neqmsg db "Strings are not equal",10
len_eq equ $ - eqmsg
len_neq equ $ - neqmsg

section .bss
s1 resb 40
s2 resb 40
l1 resq 1
l2 resq 1


section .text
global _start
```

```
_start:
    write msg1, len1
    read s1, 40
    dec rax
    mov [l1], rax

    write msg2, len2
    read s2, 40
    dec rax
    mov [l2], rax

    mov rbx, [l1]
    cmp rbx, [l2]
    jne not_equal

    mov rsi, s1
    mov rdi, s2
    mov rcx, [l1]
    cld
    repe cmpsb
    jne not_equal
```

```
    write eqmsg, len_eq
    jmp exit

not_equal:
    write neqmsg, len_neq

exit:
    mov rax, 60
    xor rdi, rdi
    syscall
```

Output:



13.string reverse:

Code:

```
%macro read 2
    mov rax, 0
    mov rdi, 0
```

```
    mov rsi, %1
    mov rdx, %2
    syscall
%endmacro

%macro write 2
    mov rax, 1
    mov rdi, 1
    mov rsi, %1
    mov rdx, %2
    syscall
%endmacro
section .data
msg1 db "Enter a string:",10
len1 equ $ - msg1
msg2 db "Reversed string:",10
len2 equ $ - msg2

section .bss
src resb 40
dest resb 40
len resq 1
```

```
section .text
global _start


_start:
    write msg1, len1
    read src, 40
    dec rax
    mov [len], rax

    mov rsi, src
    add rsi, [len]
    dec rsi
    mov rdi, dest
    mov rcx, [len]

rev_loop:
    mov al, [rsi]
    mov [rdi], al
    dec rsi
    inc rdi
```

```
    loop rev_loop

    write msg2, len2
    write dest, [len]
    call exit

exit:
    mov rax, 60
    xor rdi, rdi
    syscall
```

output:

14.String Palindrome:

Code:

```
%macro read 2
    mov rax, 0
    mov rdi, 0
    mov rsi, %1
```

```
    mov rdx, %2
    syscall
%endmacro

%macro write 2
    mov rax, 1
    mov rdi, 1
    mov rsi, %1
    mov rdx, %2
    syscall
%endmacro
; --- String Length Program ---
section .data
msg1 db "Enter a string:",10
len1 equ $ - msg1
msg2 db "Length of string: ",10
len2 equ $ - msg2

section .bss
string resb 40
strlen_val resq 1
buff resb 16
```

```
section .text
global _start


_start:
    write msg1, len1
    read string, 40
    dec rax             ; exclude newline
    mov [strlen_val], rax

    write msg2, len2
    mov rbx, [strlen_val]
    call display
    call exit

display:
    mov rsi, buff
    mov rcx, 16
disp_loop:
    rol rbx, 4
    mov dl, bl
```

```
    and dl, 0Fh
    cmp dl, 9
    jbe add30
    add dl, 7
add30:
    add dl, 30h
    mov [rsi], dl
    inc rsi
    loop disp_loop
    write buff, 16
    ret

exit:
    mov rax, 60
    xor rdi, rdi
    syscall
```

output: