Ministerul Educaţiei și Cercetării al Republicii Moldova Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work 5:

Create an NASM assembler program that contains 10 cyclic processes (functions)

Elaborated:
st. gr. FAF-211 Grama Alexandru

Verified:
asist. Univ. Vladislav Voitcovschi

Chişinău – 2023

BASIC TASK:
For this lab, each student must create an NASM assembler program that contains 10 cyclic processes (functions). Additionally, the program must allow the user to choose which of the 10 processes to execute at program launch

To accomplish this task, follow the steps below:
Write an interactive menu that allows the user to choose from the 10 processes.
Write the code for each of the 10 processes. Each process must be written cyclically so that the
program always returns to the interactive menu after a process is completed.
• Ensure that your program is well-commented and structured clearly so that it is easy to
understand and modify
• Test the program to ensure that it works correctly and that the user can choose any of the 10
processes
• Ensure that each student personalizes their program in a unique way so that there are no
identical programs.
• Prior to presenting the lab, each student must present their program and demonstrate that it
can be used to choose any of the 10 processes.
❖ The first program will contain a generator of 10 random numbers from 1 to 55
❖ These will be the varinates of each

Variants for my code: 46,18,44,1,8

```
; random_numbers.asm
; Generates 10 random numbers between 1 and 55
; nasm -f elf64 random_numbers.asm -o random_numbers.o&&ld random_numbers.o -o
random_numbers&&./random_numbers 5



  %define    FALSE        0
  %define    TRUE         1

  %define    SYSCALL_READ    0
  %define    SYSCALL_WRITE   1
  %define    SYSCALL_EXIT    60

  %define    STDIN        0
  %define    STDOUT         1
  %define    STDERR         2

  %define    NULL         0

  %define    CHAR_NEWLINE    10

  %define    SECTOR_SIZE    512
```

```
 %define    BUFSIZE       SECTOR_SIZE


;                          DATA SECTION

            SECTION .data

VERSION_MSG:    DB "Random - Version 3.0.1", 10, 0
VERSION_LEN:    EQU $-VERSION_MSG

AUTHOR_MSG:     DB "Jose Fernando Lopez Fernandez", 10, 0
AUTHOR_LEN:     EQU $-AUTHOR_MSG


;                          TEXT SECTION

            SECTION .text
            GLOBAL _start


;                          MAIN

_start:        POP     RBX                ; Move argc into RBX

            ; If RBX equals 1, there were no arguments passed in. Skip
            ; argument parsing and checking stage.

            CMP     RBX,1               ; test (argc == 1)
            JE      .NO_ARGS            ; If TRUE, skip to .GET_DIGITS

            POP     RBX                 ; Overwrite RBX with &argv[0]
.GET_NEXT_ARG:  POP     RBX                 ; ++argv
            CMP     RBX,NULL            ; Check if arg == NULL
            JE      .GET_RAND           ; If argv = 0, args process. done

            ; TODO: Check each argument for valid values and settings

            ; DEBUG: For now, the first argument will be considered a
            ; numerical value containing the number of times a random
            ; number should be generated and printed.
            ;
            ; For example, 'random 10' should generate and print ten
            ; random numbers.

            ; Convert argument string to numerical value.

.STRTOI:       MOV     AX,DS              ; Initialize AX
            MOV     ES,AX               ; Initialize ES
            MOV     RDI,RBX             ; RDI = &argv[i]
            MOV     RBP,RBX             ; RBP = &argv[i]

            ;-----------------------------------------------------------
            ; After this block executes:
            ;
            ;   RBP = address where string begins
            ;   RCX = 255 - len (including '\0')
            ;   RDI = len (including '\0')
            ;-----------------------------------------------------------
            CLD                        ; Left to right (auto-increment)
            MOV     RCX,255            ; Max length of string
            MOV     AL,0               ; Initialize AL with NUL string
            REPNE   SCASB              ; Scan string until NULL found
            SUB     RDI,RBP            ; length = end - start
            DEC     RDI                ; RDI included NULL terminator
            XCHG    RDI,R14            ; Move string length to R14

            MOV     R15,10
            XOR     RAX,RAX            ; Initial value = 0
.NEXT_VAL:      CMP     RDI,R14
            JE      .STRTOI_DONE
            XOR     RCX,RCX
            MOV     CL,BYTE[RBP+RDI]
            SUB     RCX,0x30           ; Convert from ASCII
            XOR     RDX,RDX
            MUL     R15
            ADD     RAX,RCX
            INC     RDI
            JMP     .NEXT_VAL

.STRTOI_DONE:   JMP     .GET_RANDS_PREP

.NO_ARGS:       MOV     R14,1          ; Set N = 1
            XOR     RBX,RBX            ; Set n = 0
            JMP     .GET_RANDS         ; Skip prep, since N = 0

            ; Prepare to start generating

.GET_RANDS_PREP:MOV     R14,RAX        ; Move N to R14 (non-volatile)
            XOR     RBX,RBX            ; Initial N = 0

.GET_RANDS:     INC     RBX            ; Using RBX since it's non-volatile

            ; Generate random number(s)

.GET_RAND:      RDRAND  RAX            ; Get random number
            JNC     .GET_RAND          ; If CF=0, result invalid. Repeat.
            AND     RAX, 0x7FFFFFFF    ; Ensure the number is positive
            MOV     RCX, 55            ; Set the desired range (1 to 55)
            XOR     RDX, RDX           ; Clear RDX
            DIV     RCX                ; Divide the random number by 55
            ADD     RDX, 1             ; Add 1 to the remainder
            MOV     RAX, RDX           ; Move the remainder to RAX



            ; Get each digit using 'MOD 10, DIV 10' algorithm.

.GET_DIGITS:    MOV     R15,10         ; This is the divisor
            PUSH    0                  ; Push NUL terminator for finish
                                       ; condition with no counter.

.GET_DIGIT:     CMP     RAX,0          ; If RAX = 0, done getting digits
            JE      .PRINT_DIGIT       ; If RAX = 0, done
            XOR     RDX,RDX            ; Zero out top half of dividend
            DIV     R15                ; Divide [RDX:RAX] by R15
            ADD     RDX,48             ; Convert to ASCII
            PUSH    RDX                ; This is the current least sig dig
            JMP     .GET_DIGIT          ; Loop back to get next digit

.PRINT_DIGIT:   CMP     QWORD[RSP],0        ; Stop once NULL terminator found
            JE      .FINISH             ; Found NULL terminator. goto EXIT
            MOV     RAX,SYSCALL_WRITE
            MOV     RDI,STDOUT
            MOV     RSI,RSP             ; 'String' addr. = RSP
            MOV     RDX,1               ; Single char length = 1
            SYSCALL                     ; TODO: Check return value
            POP     RSI                 ; Remove char off stack after print
            JMP     .PRINT_DIGIT        ; Go print next char

.FINISH:        POP     R10                 ; Pop final char from stack

            ; Print final newline

            PUSH    CHAR_NEWLINE        ; Push newline char to stack
            MOV     RAX,SYSCALL_WRITE
            MOV     RDI,STDOUT
            MOV     RSI,RSP             ; 'String' addr: RSP
            MOV     RDX,1               ; Single char length = 1
            SYSCALL                     ; Print newline
            POP     R10                 ; Pop newline char from stack

            CMP     RBX,R14             ; If N specified, check if done
            JL      .GET_RANDS          ; If n < N, continue generating

            ; Exit program

.EXIT_SUCCESS:  XOR     RDI,RDI             ; Exit code 0 (EXIT_SUCCESS)
.EXIT:          MOV     RAX,60              ; Alow JMP to exit with code RDI
            SYSCALL

.TEST_EXIT:     MOV     RDI,RAX             ; Exit code = last return value
            JMP     .EXIT
```

The code

```
section .data
  strEnter: db "Enter a string, the old character to replace, and the new character, separated by spaces:", 0
  strResult: db "Result: ", 0
  strLength: db "Enter the length of the string: ", 0
  strRandom: db "Random string: ", 0
  strEnterString: db "Enter a string: ", 0
  strPalindrome: db "The string is a palindrome", 0
  strNotPalindrome: db "The string is not a palindrome", 0
  strEnterDelete: db "Enter a string then enter the character to delete, separated by spaces:", 0
  strEnterAdd: db "Enter a string and the character to add, separated by a space:", 0
  strEnterIndex: db "Enter the index where to add the character (starting from 0): ", 0
  strEnterSuffix: db "Enter the suffix to add: ", 0
  strEnterNumber: db "Enter a number: ", 0
  strSquareRoot: db "Square root: ", 0
  strSum: db "Sum of prime odd numbers: ", 0
  strEnterElement: db "Enter the elements of the list (enter a non-integer value to stop): ", 0
  strRemoveElement: db "Enter the element to remove: ", 0
  strCoordinates1: db "Enter the coordinates of point 1 (x y): ", 0
  strCoordinates2: db "Enter the coordinates of point 2 (x y): ", 0
  strDistance: db "Euclidean distance: ", 0
  strInvalidChoice: db "Invalid choice", 0
  strExiting: db "Exiting...", 0
  strProcess: db "Choose a process to execute:", 0
  strMenuItems: db "1. Replace character", 10, "2. Generate random string", 10, "3. Check palindrome", 10, "4. Delete character", 10, "5. Add character", 10, "6. Add suffix", 10, "7. Square root", 10, "8. Sum of prime odd numbers", 10, "9. Remove element from list", 10, "10. Euclidean distance", 10, "0. Exit", 10, 0
  strChoice: db "Enter your choice: ", 0
  newline: db 10, 0
  stdin: equ 0
  stdout: equ 1
  sys_read: equ 0
  sys_write: equ 1
  sys_exit: equ 60
  INT: equ 0x80

section .bss
  strBuffer: resb 256
  strBuffer2: resb 256
  strTemp: resb 256
  numBuffer: resb 16
  floatBuffer: resb 256

section .text
  global _start

replaceCharacter:
  ; Write prompt
  mov eax, sys_write
  mov ebx, stdout
  lea ecx, [strEnter]
  mov edx, [strEnter+ebx]
  int INT

  ; Read input
  mov eax, sys_read
  mov ebx, stdin
  lea ecx, [strBuffer]
  mov edx, 256
  int INT

  ; Find first two space characters
  xor ebx, ebx
  xor edx, edx
  mov ecx, strBuffer

replaceCharacter_find_spaces:
  cmp byte [ecx], 0
  je replaceCharacter_done
  cmp byte [ecx], ' '
  jne replaceCharacter_next_char
  inc edx
  test edx, edx
  jz replaceCharacter_found_first_space
  inc ecx
  jmp replaceCharacter_find_spaces

  replaceCharacter_next_char:
  inc ecx
```

```
jmp replaceCharacter_find_spaces

replaceCharacter_found_first_space:
mov ebx, ecx
inc ecx
jmp replaceCharacter_find_spaces

replaceCharacter_done:
; Parse input
mov esi, strBuffer
mov edi, strTemp
mov byte [ebx], 0
inc ebx
; Old and new characters
mov al, [ebx]
mov dl, [ebx+2]

; Replace characters
replaceCharacter_loop:
cmp byte [esi], 0
je replaceCharacter_print
cmp byte [esi], al
jne replaceCharacter_copy_char
mov byte [edi], dl
jmp replaceCharacter_next

replaceCharacter_copy_char:
mov bl, [esi]
mov [edi], bl

replaceCharacter_next:
inc esi
inc edi
jmp replaceCharacter_loop

replaceCharacter_print:
mov byte [edi], 0
mov eax, sys_write
mov ebx, stdout
lea ecx, [strResult]
mov edx, [strResult+ebx]
int INT
; Print result
mov eax, sys_write
mov ebx, stdout
lea ecx, [strTemp]
mov edx, edi
sub edx, ecx
int INT

; Print newline
mov eax, sys_write
mov ebx, stdout
lea ecx, [newline]
mov edx, 1
int INT
ret
; Add the rest of the functions here
```

```
_start:
; ... main function implementation here ...
generateRandomString:
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strLength]
mov edx, [strLength+ebx]
int INT
; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [numBuffer]
mov edx, 16
int INT

; Convert string to integer
lea esi, [numBuffer]
xor edi, edi
call atoi
mov ecx, edi

; Generate random string
xor edi, edi
generateRandomString_loop:
cmp edi, ecx
je generateRandomString_print
mov eax, 26
call rand
add eax, 'a'
mov [strTemp+edi], al
inc edi
jmp generateRandomString_loop

generateRandomString_print:
mov byte [strTemp+edi], 0
mov eax, sys_write
mov ebx, stdout
lea ecx, [strRandom]
mov edx, [strRandom+ebx]
int INT
; Print result
mov eax, sys_write
mov ebx, stdout
lea ecx, [strTemp]
mov edx, edi
int INT

; Print newline
mov eax, sys_write
mov ebx, stdout
lea ecx, [newline]
mov edx, 1
int INT
ret
; Add the rest of the functions here

; Auxiliary functions
atoi:
```

```
; Input: esi (pointer to string)
; Output: edi (integer value)
xor eax, eax
xor edi, edi
atoi_loop:
movzx ecx, byte [esi]
cmp ecx, '0'
jl atoi_done
cmp ecx, '9'
jg atoi_done
sub ecx, '0'
imul edi, 10
add edi, ecx
inc esi
jmp atoi_loop

atoi_done:
ret

rand:
; Output: eax (random number in range [0, eax))
push ebx
push ecx
push edx
; Get the current time
mov eax, sys_time
lea ebx, [eax]
int INT

; Seed the random number generator
mov ecx, ebx
shr ecx, 16
xor ecx, ebx
mov eax, ecx
shl eax, 11
add eax, ecx
mov ecx, eax
shr ecx, 19
xor eax, ecx
mov ecx, eax
shl ecx, 8
add ecx, eax
mov eax, ecx
shr eax, 24

; Generate random number
mov ecx, 0xFFFFFFFF
and ecx, eax
imul ecx, ebx
add ecx, ebx
mov eax, ecx
xor eax, ebx

; Get remainder
xchg eax, ebx
xor edx, edx
div ebx

pop edx
pop ecx
pop ebx
ret
checkPalindrome:
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strEnter]
mov edx, [strEnter+ebx]
int INT
; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Check if string is palindrome
xor esi, esi
dec eax
mov edi, eax
shr edi, 1
checkPalindrome_loop:
cmp esi, edi
jge checkPalindrome_done
mov al, [strBuffer+esi]
mov dl, [strBuffer+eax]
cmp al, dl
jne checkPalindrome_not
inc esi
dec eax
jmp checkPalindrome_loop

checkPalindrome_not:
mov byte [strNotPalindrome], 1
jmp checkPalindrome_done

checkPalindrome_done:
cmp byte [strNotPalindrome], 1
je checkPalindrome_print_not
mov eax, sys_write
mov ebx, stdout
lea ecx, [strIsPalindrome]
mov edx, [strIsPalindrome+ebx]
int INT
jmp checkPalindrome_print_done

checkPalindrome_print_not:
mov eax, sys_write
mov ebx, stdout
lea ecx, [strNotPalindrome]
mov edx, [strNotPalindrome+ebx]
int INT

checkPalindrome_print_done:
; Print newline
mov eax, sys_write
mov ebx, stdout
lea ecx, [newline]
```

```
mov edx, 1
int INT
ret
deleteCharacter:
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strEnter]
mov edx, [strEnter+ebx]
int INT
; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Find the character to delete
mov al, ' '
mov byte [strTemp+ecx], al
mov edx, 0
deleteCharacter_find_char:
cmp byte [strBuffer+edx], 0
je deleteCharacter_copy_string
cmp byte [strBuffer+edx], al
je deleteCharacter_skip_char
mov byte [strTemp+ecx], byte [strBuffer+edx]
inc ecx

deleteCharacter_skip_char:
inc edx
jmp deleteCharacter_find_char

deleteCharacter_copy_string:
mov byte [strTemp+ecx], 0
; Print result
mov eax, sys_write
mov ebx, stdout
lea ecx, [strResult]
mov edx, [strResult+ebx]
int INT

; Print modified string
mov eax, sys_write
mov ebx, stdout
lea ecx, [strTemp]
mov edx, [strTemp+ebx]
int INT

; Print newline
mov eax, sys_write
mov ebx, stdout
lea ecx, [newline]
mov edx, 1
int INT
ret
; Add the rest of the functions here

addCharacter:
```

```
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strEnter]
mov edx, [strEnter+ebx]
int INT
; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Get index and character to add
mov al, ' '
mov byte [strTemp+ecx], al
mov edx, 0
addCharacter_find_space:
cmp byte [strBuffer+edx], 0
je addCharacter_parse_input
cmp byte [strBuffer+edx], al
je addCharacter_found_space
mov byte [strTemp+ecx], byte [strBuffer+edx]
inc ecx
jmp addCharacter_find_space

addCharacter_found_space:
mov byte [strTemp+ecx], 0
inc edx
mov esi, strBuffer
mov byte [esi+edx-1], 0
mov al, [strBuffer+edx]
mov byte [strTemp], al
mov ebx, ecx
inc edx
xor ecx, ecx

addCharacter_parse_input:
lea esi, [strBuffer]
lea edi, [strResult]
mov byte [edi+edx], 0

addCharacter_loop:
cmp byte [esi], 0
je addCharacter_copy
cmp ecx, ebx
jne addCharacter_copy_char
mov byte [edi+edx], al
inc edx

addCharacter_copy_char:
mov bl, [esi]
mov [edi+edx], bl

addCharacter_next:
inc esi
inc edi
inc ecx
jmp addCharacter_loop
```

```
addCharacter_copy:
mov byte [edi+edx], 0
; Print result
mov eax, sys_write
mov ebx, stdout
lea ecx, [strResult]
mov edx, [strResult+ebx]
int INT

; Print modified string
mov eax, sys_write
mov ebx, stdout
lea ecx, [strTemp]
mov edx, [strTemp+ebx]
int INT

; Print newline
mov eax, sys_write
mov ebx, stdout
, stdout
lea ecx, [newline]
mov edx, 1
int INT
ret

addSuffix:
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strEnter]
mov edx, [strEnter+ebx]
int INT
; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Find end of string
xor ecx, ecx
mov bl, byte [strBuffer]
add ecx, ebx
addSuffix_find_end:
cmp byte [strBuffer+ecx], 0
je addSuffix_copy
inc ecx
jmp addSuffix_find_end

addSuffix_copy:
lea esi, [strBuffer]
lea edi, [strResult]
mov byte [edi+ecx], 0

addSuffix_loop:
cmp byte [esi], 0
je addSuffix_print
mov byte [edi+ecx], byte [esi]
```

```
inc esi
inc edi
jmp addSuffix_loop

addSuffix_print:
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strSuffix]
mov edx, [strSuffix+ebx]
int INT

sql
Copy code
; Print result
mov eax, sys_write
mov ebx, stdout
lea ecx, [strResult]
mov edx, [strResult+ebx]
int INT

; Print newline
mov eax, sys_write
mov ebx, stdout
lea ecx, [newline]
mov edx, 1
int INT
ret
squareRoot:
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strEnter]
mov edx, [strEnter+ebx]
int INT

css
Copy code
; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Convert input to float
finit
fld qword [strBuffer]
fsqrt
fstp qword [result]

; Print square root
mov eax, sys_write
mov ebx, stdout
lea ecx, [strSquareRoot]
mov edx, [strSquareRoot+ebx]
int INT
mov eax, sys_write
mov ebx, stdout
mov ecx, dword [result]
call print_double

; Print newline
mov eax, sys_write
mov ebx, stdout
lea ecx, [newline]
mov edx, 1
int INT
ret
sumOfPrimeOddNumbers:
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strEnter]
mov edx, [strEnter+ebx]
int INT

css
Copy code
; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Convert input to integer
mov esi, [strBuffer]
xor ecx, ecx
xor edx, edx
sumOfPrimeOddNumbers_parse_input:
cmp byte [esi], 0
je sumOfPrimeOddNumbers_compute_sum
cmp byte [esi], 10
je sumOfPrimeOddNumbers_compute_sum
mov al, byte [esi]
sub al, 48
imul edx, 10
add edx, eax
inc esi
jmp sumOfPrimeOddNumbers_parse_input

sumOfPrimeOddNumbers_compute_sum:
xor eax, eax
mov ecx, dword [strBuffer]

sumOfPrimeOddNumbers_loop:
cmp dword [strBuffer], 0
jle sumOfPrimeOddNumbers_exit
mov ebx, 2
edx, dword [strBuffer]
sumOfPrimeOddNumbers_is_odd:
test edx, 1
je sumOfPrimeOddNumbers_next

sumOfPrimeOddNumbers_is_prime:
push edx
call isPrime
```

```
pop edx
test al, 1
jne sumOfPrimeOddNumbers_add

sumOfPrimeOddNumbers_next:
inc edx
jmp sumOfPrimeOddNumbers_loop

sumOfPrimeOddNumbers_add:
add eax, edx
dec ecx
jmp sumOfPrimeOddNumbers_next

sumOfPrimeOddNumbers_exit:
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strResult]
mov edx, [strResult+ebx]
int INT
; Print sum
mov eax, sys_write
mov ebx, stdout
mov ecx, eax
call print_int

; Print newline
mov eax, sys_write
mov ebx, stdout
lea ecx, [newline]
mov edx, 1
int INT
ret
removeElementFromList:
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strEnterList]
mov edx, [strEnterList+ebx]
int INT
; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Convert input to list
mov esi, [strBuffer]
xor ecx, ecx
xor edx, edx
removeElementFromList_parse_input:
cmp byte [esi], 0
je removeElementFromList_compute_result
cmp byte [esi], 10
je removeElementFromList_compute_result
mov al, byte [esi]
sub al, 48
imul edx, 10
```

```
add edx, eax
inc esi
jmp removeElementFromList_parse_input

removeElementFromList_compute_result:
push edx
push ebx
push ecx
lea esi, [lst]
call removeElementFromList_remove
pop ecx
pop ebx
pop edx
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strResult]
mov edx, [strResult+ebx]
int INT

; Print list
mov eax, sys_write
mov ebx, stdout
mov ecx, [lst]
call print_vector

; Print newline
mov eax, sys_write
mov ebx, stdout
lea ecx, [newline]
mov edx, 1
int INT
ret
removeElementFromList_remove:
mov eax, ecx
xor ecx, ecx

removeElementFromList_loop:
mov edx, dword [esi+eax]
test edx, edx
je removeElementFromList_exit
cmp edx, ebx
jne removeElementFromList_copy
inc eax
jmp removeElementFromList_loop

removeElementFromList_copy:
mov dword [edi+ecx], edx
inc ecx
inc eax
jmp removeElementFromList_loop

removeElementFromList_exit:
mov [edi+ecx], 0
mov dword [lst+4], ecx
ret

euclideanDistance:
; Write prompt
```

```
mov eax, sys_write
mov ebx, stdout
lea ecx, [strEnterPoint1]
mov edx, [strEnterPoint1+ebx]
int INT
; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Convert input to float
finit
fld qword [strBuffer]
fstp qword [x1]

; Write prompt
mov eax, sys_write
mov eax, sys_write
mov ebx, stdout
lea ecx, [strEnterPoint2]
mov edx, [strEnterPoint2+ebx]
int INT

; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Convert input to float
finit
fld qword [strBuffer]
fstp qword [y1]

; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strEnterPoint1]
mov edx, [strEnterPoint1+ebx]
int INT

; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Convert input to float
finit
fld qword [strBuffer]
fstp qword [x2]

; Write prompt
mov eax, sys_write
mov ebx, stdout
```

```
lea ecx, [strEnterPoint2]
mov edx, [strEnterPoint2+ebx]
int INT

; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Convert input to float
finit
fld qword [strBuffer]
fstp qword [y2]

; Compute distance
finit
fld qword [x2]
fld qword [x1]
fsubp
fmul st0, st0
fld qword [y2]
fld qword [y1]
fsubp
fmul st0, st0
faddp
fsqrt

; Print result
mov eax, sys_write
mov ebx, stdout
mov ecx, [strEuclideanDistance]
mov edx, [strEuclideanDistance+ebx]
int INT

mov eax, sys_write
mov ebx, stdout
mov ecx, eax
call print_float

; Print newline
mov eax, sys_write
mov ebx, stdout
lea ecx, [newline]
mov edx, 1
int INT
ret
print_vector:
push ebx
mov ebx, ecx
mov ecx, dword [lst+4]
print_vector_loop:
cmp ecx, 0
je print_vector_exit
push dword [ebx]
call print_int
add esp, 4
mov eax, sys_write
mov ebx, stdout
lea ecx, [space]
mov edx, 1
int INT
add ebx, 4
dec ecx
jmp print_vector_loop

print_vector_exit:
pop ebx
ret

print_int:
push ebx
push ecx
push edx
mov edx, 0
mov ecx, 10
div_loop:
xor eax, eax
div ecx
push edx
test eax, eax
jnz div_loop

print_loop:
pop edx
add edx, 48
mov eax, sys_write
mov ebx, stdout
mov ecx, edx
mov edx, 1
int INT
cmp esp, ebp
jnz print_loop
pop edx
pop ecx
pop ebx
ret
print_float:
fld qword [esp+4]
fstp qword [esp+4]
fldcw [float_format]
mov eax, sys_write
mov ebx, stdout
lea ecx, [strFloatBuffer]
mov edx, 32
fwait
call sprintf
mov ecx, [strFloatBuffer]
mov edx, 0
float_loop:
mov al, byte [ecx+edx]
cmp al, 0
je float_exit
mov eax, sys_write
mov ebx, stdout
mov edx, 1
int INT
```

```
inc edx
jmp float_loop
float_exit:
mov eax, sys_write
mov ebx, stdout
lea ecx, [float_format_reset]
mov edx, [float_format_reset+ebx]
int INT
ret

; Constants
newline db 10
space db ' '

strEnterString db "Enter a string, the old character to replace, and the new character, separated by spaces:", 0
strEnterLength db "Enter the length of the string: ", 0
strEnterString2 db "Enter a string: ", 0
strEnterChar db "Enter the character to delete: ", 0
strEnterCharAdd db "Enter the index where to add the character (starting from 0): ", 0
strEnterCharToAdd db "Enter the character to add: ", 0
strEnterSuffix db "Enter the suffix to add: ", 0
strEnterNumber db "Enter a number: ", 0
strEnterPrime db "Enter the number of prime odd numbers to sum: ", 0
strEnterPoint1 db "Enter the coordinates of point 1 (x y): ", 0
strEnterPoint2 db "Enter the coordinates of point 2 (x y): ", 0
strEnterElement db "Enter the element to remove: ", 0
strReplaceResult db "Result: ", 0
strRandomString db "Random string: ", 0
strPalindrome db "The string is a palindrome", 0
strNotPalindrome db "The string is not a palindrome", 0
strDeleteResult db "Result: ", 0
strAddResult db "Result: ", 0
strSquareRoot db "Square root: ", 0
strSumOfPrimeOddNumbers db "Sum of %d prime odd numbers: %d", 0
strEuclideanDistance db "Euclidean distance: ", 0
strFloatBuffer db 32 dup(0)

float_format dw 0x027F
float_format_reset db 0xC9, 0x03, 0x00, 0x00, 0x00

; Function prototypes
print_vector: ; (vector<int> lst)
print_int: ; (int n)
print_float: ; (float x)
isPrime: ; (int n) -> bool
replaceCharacter: ; ()
generateRandomString: ; ()
checkPalindrome: ; ()
deleteCharacter: ; ()
addCharacter: ; ()
addSuffix: ; ()
squareRoot: ; ()
sumOfPrimeOddNumbers: ; ()
removeElementFromList: ; ()
euclideanDistance: ; ()
; Function: isPrime
; Input: int n
; Output: bool
; Description: Determines whether an integer is a prime number.
```

```
  isPrime:
  push ebp
  mov ebp, esp
  push ebx
  push ecx
  push edx
  mov ebx, [ebp+8]
  cmp ebx, 1
  jle is_not_prime
  cmp ebx, 2
  je is_prime

  mov edx, 2
  div_loop:
     mov eax, ebx
     cdq
     div edx
     test edx, edx
     jz is_not_prime
     inc edx
     cmp edx, eax
     jle div_loop

  is_prime:
     mov eax, 1
     jmp exit

  is_not_prime:
     mov eax, 0

  exit:
     pop edx
     pop ecx
     pop ebx
     mov esp, ebp
     pop ebp
     ret
  ; Function: replaceCharacter
  ; Input: none
  ; Output: none
  ; Description: Reads a string, an old character, and a new character from the console and replaces all
occurrences of the old character with the new character in the string.
  replaceCharacter:
  push ebp
  mov ebp, esp
  push ebx
  push ecx
  push edx
  ; Write prompt
  mov eax, sys_write
  mov ebx, stdout
  lea ecx, [strEnterString]
  mov edx, [strEnterString+ebx]
  int INT

  ; Read input
  mov eax, sys_read
  mov ebx, stdin
  lea ecx, [strBuffer]
  mov edx, 256
  int INT

  ; Parse input
  mov ebx, strBuffer
  call parse_string
  mov ecx, eax
  mov edx, [ebx+eax]
  mov byte [ebx+eax], 0
  mov eax, ebx
  mov ebx, edx
  mov edx, [eax+ecx+1]
  mov byte [eax+ecx+1], 0

  ; Write prompt
  mov eax, sys_write
  mov ebx, stdout
  lea ecx, [strReplaceResult]
  mov edx, [strReplaceResult+ebx]
  int INT

  ; Replace characters
  mov ebx, [eax+ecx+1] ; old char
  mov edx, [eax+ecx+2] ; new char
  mov ecx, [eax+ecx] ; string
  mov esi, ecx
  .replace_char_loop:
     mov al, byte [esi]
     cmp al, 0
     je replace_char_exit
     cmp al, bl
     jne .not_replace_char
     mov byte [esi], dl
  .not_replace_char:
     inc esi
     jmp .replace_char_loop
  replace_char_exit:

  mov eax, sys_write
  mov ebx, stdout
  mov ecx, [eax+ebp+8]
  mov edx, [eax+ebp+12]
  mov eax, sys_write
  int INT

  pop edx
  pop ecx
  pop ebx
  mov esp, ebp
  pop ebp
  ret
  ; Function: generateRandomString
  ; Input: none
  ; Output: none
  ; Description: Reads a length from the console and generates a random string of the specified length
consisting of lowercase letters.
  generateRandomString:
  push ebp
  mov ebp, esp
```

```
push ebx
push ecx
push edx
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strEnterLength]
mov edx, [strEnterLength+ebx]
int INT

; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Convert input to integer
mov ebx, strBuffer
call parse_string
push eax ; save length
mov ecx, eax

; Generate random string
mov eax, sys_write
mov ebx, stdout
lea ecx, [strRandomString]
mov edx, [strRandomString+ebx]
int INT

mov ebx, strBuffer
mov edx, ecx
.generate_random_loop:
   mov al, 'a'
   call rand
   and eax, 0x1F
   add eax, 'a'
   mov byte [edx], al
   inc edx
   loop .generate_random_loop

mov byte [edx], 0

mov eax, sys_write
mov ebx, stdout
mov ecx, strBuffer
mov edx, [eax+ebp+8]
mov eax, sys_write
int INT

pop eax ; restore length

pop edx
pop ecx
pop ebx
mov esp, ebp
pop ebp
ret
; Function: checkPalindrome
; Input: none
; Output: none
; Description: Reads a string from the console and determines whether it is a palindrome.
checkPalindrome:
push ebp
mov ebp, esp
push ebx
push ecx
push edx
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strEnterString2]
mov edx, [strEnterString2+ebx]
int INT

; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Parse input
mov ebx, strBuffer
call parse_string
mov ecx, eax
mov edx, [ebx+eax]
mov byte [ebx+eax], 0

; Check if palindrome
mov esi, ebx
add esi, ecx
dec esi
.check_palindrome_loop:
   cmp ebx, esi
   jge is_palindrome
   mov al, byte [ebx]
   mov dl, byte [esi]
   cmp al, dl
   jne not_palindrome
   inc ebx
   dec esi
   jmp .check_palindrome_loop

is_palindrome:
   mov eax, sys_write
   mov ebx, stdout
   lea ecx, [strPalindrome]
   mov edx, [strPalindrome+ebx]
   int INT
   jmp exit_check_palindrome

not_palindrome:
   mov eax, sys_write
   mov ebx, stdout
   lea ecx, [strNotPalindrome]
   mov edx, [strNotPalindrome+ebx]
   int INT
```

```
  exit_check_palindrome:
      pop edx
      pop ecx
      pop ebx
      mov esp, ebp
      pop ebp
      ret
  ; Function: deleteCharacter
  ; Input: none
  ; Output: none
  ; Description: Reads a string and a character from the console and deletes all occurrences of the character in
the string.
  deleteCharacter:
  push ebp
  mov ebp, esp
  push ebx
  push ecx
  push edx
  ; Write prompt
  mov eax, sys_write
  mov ebx, stdout
  lea ecx, [strEnterString]
  mov edx, [strEnterString+ebx]
  int INT

  ; Read input
  mov eax, sys_read
  mov ebx, stdin
  lea ecx, [strBuffer]
  mov edx, 256
  int INT

  ; Parse input
  mov ebx, strBuffer
  call parse_string
  mov ecx, eax
  mov edx, [ebx+eax]
  mov byte [ebx+eax], 0
  mov eax, [eax+ebp+8]

  ; Delete characters
  mov esi, ebx
  mov edi, ebx
  .delete_char_loop:
      cmp byte [esi], al
      je .inc esi
              jmp .write_char

              .:
                 mov al, byte [esi]
                 mov byte [edi], al
                 inc esi
                 inc edi

              .write_char:
                 cmp edi, ecx
                 jl .delete_char_loop
```

```
        mov byte [edi], 0

        mov eax, sys_write
        mov ebx, stdout
        mov ecx, strBuffer
        mov edx, [eax+ebp+8]
        mov eax, sys_write
        int INT

        pop edx
        pop ecx
        pop ebx
        mov esp, ebp
        pop ebp
        ret
        ; Function: addCharacter
        ; Input: none
        ; Output: none
        ; Description: Reads a string, a character and an index from the console and adds the character to the
string at the specified index.
        addCharacter:
        push ebp
        mov ebp, esp
        push ebx
        push ecx
        push edx
  ; Write prompt
  mov eax, sys_write
  mov ebx, stdout
  lea ecx, [strEnterString2]
  mov edx, [strEnterString2+ebx]
  int INT

  ; Read input
  mov eax, sys_read
  mov ebx, stdin
  lea ecx, [strBuffer]
  mov edx, 256
  int INT

  ; Parse input
  mov ebx, strBuffer
  call parse_string
  mov ecx, eax
  mov edx, [ebx+eax]
  mov byte [ebx+eax], 0

  ; Read suffix
  mov eax, sys_write
  mov ebx, stdout
  lea ecx, [strEnterSuffix]
  mov edx, [strEnterSuffix+ebx]
  int INT

  mov eax, sys_read
  mov ebx, stdin
  lea ecx, [strBuffer2]
  mov edx, 256
  int INT
```

```
; Parse input
mov ebx, strBuffer2
call parse_string
mov eax, [eax+ebp+8]
mov edx, [ebx+eax]
mov byte [ebx+eax], 0
; Add suffix
mov esi, strBuffer
mov edi, strBuffer2
mov ecx, [eax+ebp+8]
add ecx, esi
mov edx, [eax+ebp+12]
add edx, edi
.copy_string_loop:
    cmp esi, ecx
    jge .write_string
    mov al, byte [esi]
    mov byte [edi], al
    inc esi
    inc edi
    jmp .copy_string_loop

.write_string:
    mov ecx, edx
    mov eax, byte [esi]
    mov byte [edi], al
    inc esi
    inc edi
    cmp esi, ecx
    jl .write_string

mov byte [edi], 0

mov eax, sys_write
mov ebx, stdout
mov ecx, strBuffer2
mov edx, [eax+ebp+8]
mov eax, sys_write
int INT

pop edx
pop ecx
pop ebx
mov esp, ebp
pop ebp
ret
; Function: squareRoot
; Input: none
; Output: none
; Description: Reads a number from the console and computes its square root.
squareRoot:
push ebp
mov ebp, esp
push ebx
push ecx
push edx
; Write prompt
mov eax, sys_write
mov ebx, stdout
lea ecx, [strEnterNumber]
mov edx, [strEnterNumber+ebx]
int INT

; Read input
mov eax, sys_read
mov ebx, stdin
lea ecx, [strBuffer]
mov edx, 256
int INT

; Parse input
mov ebx, strBuffer
call parse_number
mov ebx, [eax+ebp+8]

; Compute square root
fld qword [ebx]
fsqrt
fstp qword [ebx]

; Print result
mov eax, sys_write
mov ebx, stdout
lea ecx, [strSquareRoot]
mov edx, [strSquareRoot+ebx]
int INT

mov eax, sys_write
mov ebx, stdout
mov ecx, strBuffer
fld qword [ecx]
fstp qword [esp-8]
fld tbyte [esp-8]
fstp qword [esp-8]
fld tbyte [esp-8]
fstp qword [esp-8]
fld tbyte [esp-8]
fstp qword [esp-8]
mov edx, 0
call print_float
add esp, 32

pop edx
pop ecx
pop ebx
mov esp, ebp
pop ebp
ret
; Function: isPrime
; Input: integer n
; Output: boolean isPrime
; Description: Determines whether an integer is prime or not.
isPrime:
push ebp
mov ebp, esp
push ebx
push ecx
push edx
; Load input parameter
mov ebx, [ebp+8]

; Check if number is <= 1
cmp ebx, 1
jle .not_prime

; Check if number is 2 or 3
cmp ebx, 2
je .prime
cmp ebx, 3
je .prime

; Check if number is divisible by 2 or 3
cmp ebx, 0
je .not_prime
mov eax, 3
.check_divisible_loop:
   cmp eax, ebx
   jg .prime
   mov edx, 0
   div ebx
   cmp edx, 0
   je .not_prime
   add eax, jmp .check_divisible_loop
; Number is prime
.prime:
   mov eax, 1
   jmp .done

; Number is not prime
.not_prime:
   mov eax, 0

.done:
   pop edx
   pop ecx
   pop ebx
   mov esp, ebp
   pop ebp
   ret
; Function: sumOfPrimeOddNumbers
; Input: integer n
; Output: integer sum
; Description: Sums the first n prime odd numbers.
sumOfPrimeOddNumbers:
push ebp
mov ebp, esp
push ebx
push ecx
push edx
; Load input parameter
mov ebx, [ebp+8]

; Initialize sum and count
mov eax, 0
mov edx, 0
mov esi, 1
```

```
; Loop until count reaches n
.sum_loop:
    ; Check if number is odd and prime
    cmp esi, 1
    je .not_odd
    cmp byte [primes+esi], 1
    je .not_prime

    ; Add number to sum
    add eax, esi
    inc edx

    ; Check if count has reached n
    cmp edx, ebx
    jge .done

    ; Continue with next number
    .not_prime:
       add esi, 2
       jmp .sum_loop

    .not_odd:
       add esi, 1
       jmp .sum_loop

.done:
    pop edx
    pop ecx
    pop ebx
    mov esp, ebp
    pop ebp
    ret
.data
strEnterNumber db "Enter a number: ", 0
strSquareRoot db "Square root: ", 0
strBuffer times 256 db 0
strBuffer2 times 256 db 0
primes times 1000 db 0 ; pre-computed primes up to 20000

section .bss
input_buffer resb 256

section .text
global _start

_start:
push ebp
mov ebp, esp
push ebx
push ecx
push edx
; Initialize random number generator
mov eax, SYS_TIME
xor ebx, ebx
int INT
mov dword [esp], eax
mov eax, SYS_SRAND
int INT
```

```
; Pre-compute primes up to 20000
mov ecx, 20000
mov ebx, 3
.compute_primes_loop:
   mov eax, ebx
   call isPrime
   cmp eax, 1
   jne .not_prime
   mov byte [primes+eax], 1
   .not_prime:
      add ebx, 2
      loop .compute_primes_loop

; Main program loop
.main_loop:
   ; Write menu
   mov eax, sys_write
   mov ebx, stdout
   lea ecx, [strMenu]
   mov edx, [strMenu+ebx]
   int INT

   ; Read choice
   mov eax, sys_read
   mov ebx, stdin
   lea ecx, [input_buffer]
   mov edx, 256
   int INT

   ; Parse choice
   mov ebx, input_buffer
   call parse_number
   mov ebx, [eax+ebp+8]

   ; Execute chosen process
   cmp ebx, 1
   je .replace_character
   cmp ebx, 2
   je .generate_random_string
   cmp ebx, 3
   je .check_palindrome
   cmp ebx, 4
   je .delete_character
   cmp ebx, 5
   je .add_character
   cmp ebx, 6
   je .add_suffix
   cmp ebx, 7
   je .square_root
   cmp ebx, 8
   je .sum_of_prime_odd_numbers
   cmp ebx, 9
   je .remove_element_from_list
   cmp ebx, 10
   je .euclidean_distance
   cmp ebx, 0
   je .exit
   jmp .main_loop
```

```
; Replace Character
.replace_character:
    lea ecx, [strEnterString]
    call print_string
    call read_string
    lea ebx, [strBuffer]
    mov [ebx+eax], 0 ; null-terminate string
    lea ecx, [strBuffer+eax+1]
    mov al, [strBuffer]
    mov ah, [strBuffer+2]
    lea edx, [strBuffer2]
    call replace_character
    lea ecx, [strResult]
    call print_string
    lea ecx, [strBuffer2]
    call print_string
    jmp .main_loop

; Generate Random String
.generate_random_string:
    lea ecx, [strEnterNumber]
    call print_string
    call read_number
    push eax
    call generate_random_string
    add esp, 4
    lea ecx, [strRandomString]
    call print_string
    lea ecx, [strBuffer]
    call print_string
    jmp .main_loop

; Check Palindrome
.check_palindrome:
    lea ecx, [strEnterString]
    call print_string
    call read_string
    lea ebx, [strBuffer]
    mov [ebx+eax], 0 ; null-terminate string
    lea ecx, [strBuffer]
    call check_palindrome
    lea ecx, [strResult]
    call print_string
    jmp .main_loop

; Delete Character
.delete:
    lea ecx, [strEnterString]
    call print_string
    call read_string
    lea ebx, [strBuffer]
    mov [ebx+eax], 0 ; null-terminate string
    lea ecx, [strBuffer+eax+1]
    mov al, [strBuffer]
    lea edx, [strBuffer2]
    call delete_character
    lea ecx, [strResult]
    call print_string
    lea ecx, [strBuffer2]
    call print_string
    jmp .main_loop

; Add Character
.add_character:
    lea ecx, [strEnterString]
    call print_string
    call read_string
    lea ebx, [strBuffer]
    mov [ebx+eax], 0 ; null-terminate string
    lea ecx, [strBuffer]
    call print_string
    lea ecx, [strEnterChar]
    call print_string
    call read_char
    mov byte [strBuffer+eax], al
    lea ecx, [strEnterIndex]
    call print_string
    call read_number
    push eax
    lea edx, [strBuffer2]
    call add_character
    add esp, 4
    lea ecx, [strResult]
    call print_string
    lea ecx, [strBuffer2]
    call print_string
    jmp .main_loop

; Add Suffix
.add_suffix:
    lea ecx, [strEnterString]
    call print_string
    call read_string
    lea ebx, [strBuffer]
    mov [ebx+eax], 0 ; null-terminate string
    lea ecx, [strEnterSuffix]
    call print_string
    call read_string
    lea ebx, [strBuffer2]
    call add_suffix
    lea ecx, [strResult]
    call print_string
    lea ecx, [strBuffer2]
    call print_string
    jmp .main_loop
    ; Square Root
    .square_root:
        lea ecx, [strEnterNumber]
        call print_string
        call read_double
        fsqrt
        lea ecx, [strSquareRoot]
        call print_string
        fld qword [ebp-8] ; restore stack
        jmp .main_loop

    ; Sum of Prime Odd Numbers
```

```
.sum_of_prime_odd_numbers:
    lea ecx, [strEnterNumber]
    call print_string
    call read_number
    push eax
    call sum_of_prime_odd_numbers
    add esp, 4
    lea ecx, [strResult]
    call print_string
    jmp .main_loop

; Remove Element From List
.remove_element_from_list:
    lea ecx, [strEnterElements]
    call print_string
    call read_int_list
    lea ecx, [strEnterNumber]
    call print_string
    call read_number
    push eax
    push edx ; save pointer to list
    call remove_element_from_list
    add esp, 8
    lea ecx, [strResult]
    call print_string
    lea ecx, [strBuffer]
    call print_int_list
    jmp .main_loop

; Euclidean Distance
.euclidean_distance:
    lea ecx, [strEnterCoordinates1]
    call print_string
    call read_coordinates
    push edx ; save y1
    push eax ; save x1
    lea ecx, [strEnterCoordinates2]
    call print_string
    call read_coordinates
    fsub st, st(2) ; dx = x2 - x1
    fmul st, st ; dx^2
    pop eax ; restore x1
    fsub st, st(1) ; dy = y2 - y1
    fmul st, st ; dy^2
    pop edx ; restore y1
    faddp st(1), st ; dx^2 + dy^2
    fsqrt
    lea ecx, [strEuclideanDistance]
    call print_string
    jmp .main_loop

; Exit
.exit:
    lea ecx, [strExiting]
    call print_string
    mov eax, 0 ; return 0

; Read a string from the terminal into the buffer
read_string:
        push ebp
        mov ebp, esp
        sub esp, 8 ; allocate space for local variables
        mov ebx, [ebp+8] ; buffer pointer
        mov ecx, strInputFormat
        mov edx, strInputError
        call read_formatted_string
        mov esp, ebp
        pop ebp
        ret

    ; Read an integer from the terminal
    read_number:
        push ebp
        mov ebp, esp
        sub esp, 8 ; allocate space for local variables
        mov ecx, strInputFormat
        mov edx, strInputError
        call read_formatted_number
        mov esp, ebp
        pop ebp
        ret

    ; Read a double from the terminal
    read_double:
        push ebp
        mov ebp, esp
        sub esp, 8 ; allocate space for local variables
        mov ecx, strInput
    FormatDouble
    mov edx, strInputError
    call read_formatted_double
    mov esp, ebp
    pop ebp
    ret
; Read a list of integers from the terminal
read_int_list:
    push ebp
    mov ebp, esp
    sub esp, 8 ; allocate space for local variables
    mov ebx, [ebp+8] ; pointer to buffer
    mov ecx, strInputFormat
    mov edx, strInputError
    call read_formatted_int_list
    mov esp, ebp
    pop ebp
    ret

; Read coordinates (two doubles separated by a space) from the terminal
read_coordinates:
    push ebp
    mov ebp, esp
    sub esp, 8 ; allocate space for local variables
    mov ecx, strInputFormat
    mov edx, strInputError
    call read_formatted_coordinates
    mov esp, ebp
    pop ebp
    ret
```

```
; Print a string to the terminal
print_string:
    push ebp
    mov ebp, esp
    push ecx ; save ecx
    mov ecx, [ebp+8] ; string pointer
    mov edx, [ebp+12] ; string length
    mov eax, 4 ; system call for write
    mov ebx, 1 ; file descriptor for stdout
    int 0x80 ; call kernel
    pop ecx ; restore ecx
    mov esp, ebp
    pop ebp
    ret

; Print an integer to the terminal
print_number:
    push ebp
    mov ebp, esp
    push ecx ; save ecx
    push edx ; save edx
    call FormatInteger
    mov ecx, [ebp+8] ; integer
    mov edx, strBuffer
    call print_string
    pop edx ; restore edx
    pop ecx ; restore ecx
    mov esp, ebp
    pop ebp
    ret

; Print a double to the terminal
print_double:
    push ebp
    mov ebp, esp
    push ecx ; save ecx
    push edx ; save edx
    call FormatDouble
    fstp qword [strBuffer]
    mov edx, strBuffer
    call print_string
    pop edx ; restore edx
    pop ecx ; restore ecx
    mov esp, ebp
    pop ebp
    ret

; Print a list of integers to the terminal
print_int_list:
    push ebp
    mov ebp, esp
    push ebx ; save ebx
    push ecx ; save ecx
    push edx ; save edx
    mov ecx, [ebp+8] ; list pointer
    mov edx, strBuffer
    call FormatIntList
    mov ecx, strBuffer
    call print_string
    pop edx ; restore edx
    pop ecx ; restore ecx
    pop ebx ; restore ebx
    mov esp, ebp
    pop ebp
    ret

; Print coordinates (two doubles separated by a space) to the terminal
print_coordinates:
    push ebp
    mov ebp, esp
    push eax ; save eax
    push edx ; save edx
    call FormatDouble
    fstp qword [strBuffer]
    mov eax, strBuffer
    call print_string
    mov eax, strSpace
    call print_string
    call FormatDouble
    fstp qword [strBuffer]
    mov eax, strBuffer
    call print_string
    pop edx ; restore edx
    pop eax ; restore eax
    mov esp, ebp
    pop ebp
    ret

; Compute the sum of n prime odd numbers
sum_of_prime_odd_numbers:
    push ebp
    mov ebp, esp
    sub esp
fmul st(0), st(0) ; square of dx
    fld dword [ebx] ; load y1
    fld dword [edx] ; load y2
    fsubp st(1), st(0) ; y2 - y1
    fmul st(0), st(0) ; square of dy
    faddp st(1), st(0) ; dx^2 + dy^2
    fsqrt ; square root of sum
    mov esp, ebp
    pop ebp
    ret

; Check if a number is prime
is_prime:
    push ebp
    mov ebp, esp
    push ebx ; save ebx
    push ecx ; save ecx
    push edx ; save edx
    mov ecx, [ebp+8] ; number to check
    mov ebx, 2 ; divisor
    mov eax, 1 ; is prime flag
    .check_loop:
       cmp ebx, ecx ; check if divisor exceeds number
       jg .done
```

```
        mov edx, 0 ; clear edx for division
        div ebx ; divide number by divisor
        test edx, edx ; check remainder
        je .not_prime ; not prime
        inc ebx ; increment divisor
        jmp .check_loop
    .not_prime:
        mov eax, 0 ; set is prime flag to false
    .done:
        pop edx ; restore edx
        pop ecx ; restore ecx
        pop ebx ; restore ebx
    mov esp, ebp
    pop ebp
    ret

section .data
strMenu db "Menu:", 0
strReplaceCharacter db "1. Replace Character", 0
strGenerateRandomString db "2. Generate Random String", 0
strCheckPalindrome db "3. Check Palindrome", 0
strDeleteCharacter db "4. Delete Character", 0
strAddCharacter db "5. Add Character", 0
strAddSuffix db "6. Add Suffix", 0
strSquareRoot db "7. Square Root", 0
strSumOfPrimeOddNumbers db "8. Sum of Prime Odd Numbers", 0
strRemoveElementFromList db "9. Remove Element From List", 0
strEuclideanDistance db "10. Euclidean Distance", 0
strExit db "0. Exit", 0
strEnterString db "Enter a string: ", 0
strEnterNumber db "Enter a number: ", 0
strEnterChar db "Enter a character: ", 0
strEnterIndex db "Enter an index: ", 0
strEnterSuffix db "Enter a suffix: ", 0
strEnterLength db "Enter a length: ", 0
strEnterElements db "Enter elements (separated by spaces): ", 0
strEnterCoordinates1 db "Enter coordinates (x1 y1): ", 0
strEnterCoordinates2 db "Enter coordinates (x2 y2): ", 0
strResult db "Result: ", 0
strBuffer times 256 db 0
strBuffer2 times 256 db 0
strInputFormat db "%255s", 0
strInputError db "Invalid input. Please try again.", 0
strSpace db " ", 0

section .bss
input_buffer resb 256

section .text
global _start

_start:
    ; Initialize random number generator
    mov eax, SYS_TIME
    xor ebx, ebx
    int 0x80
    mov dword [esp], eax
    mov eax, SYS_SRAND
    int
    ; Loop through the menu until the user chooses to exit
    .loop_menu:
        ; Print menu
        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strMenu
        mov edx, str_len strMenu
        int 0x80

        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strReplaceCharacter
        mov edx, str_len strReplaceCharacter
        int 0x80

        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strGenerateRandomString
        mov edx, str_len strGenerateRandomString
        int 0x80

        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strCheckPalindrome
        mov edx, str_len strCheckPalindrome
        int 0x80

        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strDeleteCharacter
        mov edx, str_len strDeleteCharacter
        int 0x80

        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strAddCharacter
        mov edx, str_len strAddCharacter
        int 0x80

        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strAddSuffix
        mov edx, str_len strAddSuffix
        int 0x80

        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strSquareRoot
        mov edx, str_len strSquareRoot
        int 0x80

        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strSumOfPrimeOddNumbers
        mov edx, str_len strSumOfPrimeOddNumbers
        int 0x80

        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strRemoveElementFromList
        mov edx, str_len strRemoveElementFromList
        int 0x80

        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strEuclideanDistance
        mov edx, str_len strEuclideanDistance
        int 0x80

        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strExit
        mov edx, str_len strExit
        int 0x80

        ; Get user choice
        mov eax, SYS_READ
        mov ebx, STDIN
        mov ecx, input_buffer
        mov edx, 256
        int 0x80

        ; Convert user input to integer
        mov eax, input_buffer
        call atoi

        ; Execute the corresponding function based on the user's choice
        cmp eax, 1
        je .replace_character
        cmp eax, 2
        je .generate_random_string
        cmp eax, 3
        je .check_palindrome
        cmp eax, 4
        je .delete_character
        cmp eax, 5
        je .add_character
        cmp eax, 6
        je .add_suffix
        cmp eax, 7
        je .square_root
        cmp eax, 8
        je .sum_of_prime_odd_numbers
        cmp eax, 9
        je .remove_element_from_list
        cmp eax, 10
        je .euclidean_distance
        cmp eax, 0
        je .exit

        ; Invalid choice
        jmp .loop_menu

      ; Replace a character in a string
      .replace_character:
     ; Print prompt
     mov eax, SYS_WRITE
     mov ebx, STDOUT
```

```
    mov ecx, strEnterStringOldNew
    mov edx, str_len strEnterStringOldNew
    int 0x80

    ; Get input from user
    mov eax, SYS_READ
    mov ebx, STDIN
    mov ecx, input_buffer
    mov edx, 256
    int 0x80

    ; Parse user input
    mov ebx, input_buffer
    call parse_input_replace

    ; Replace character in string
    mov eax, [ebp-12] ; old character
    mov ebx, [ebp-8]  ; new character
    mov ecx, [ebp-4]  ; string
    call replace_character

    ; Print result
    mov eax, SYS_WRITE
    mov ebx, STDOUT
    mov ecx, strResult
    mov edx, str_len strResult
    int 0x80
    mov eax, [ebp-4]  ; string
    call print_string
    jmp .loop_menu

; Generate a random string
.generate_random_string:
    ; Print prompt
    mov eax, SYS_WRITE
    mov ebx, STDOUT
    mov ecx, strEnterStringLength
    mov edx, str_len strEnterStringLength
    int 0x80

    ; Get input from user
    mov eax, SYS_READ
    mov ebx, STDIN
    mov ecx, input_buffer
    mov edx, 256
    int 0x80

    ; Parse user input
    mov ebx, input_buffer
    call parse_input_length

    ; Generate random string
    mov eax, [ebp-4]  ; length
    call generate_random_string

    ; Print result
    mov eax, SYS_WRITE
    mov ebx, STDOUT
    mov ecx, strRandomString
    mov edx, str_len strRandomString
    int 0x80
    mov eax, [ebp-8]  ; string
    call print_string
    jmp .loop_menu

  ; Check if a string is a palindrome
  .check_palindrome:
    ; Print prompt
    mov eax, SYS_WRITE
    mov ebx, STDOUT
    mov ecx, strEnterString
    mov edx, str_len strEnterString
    int 0x80

    ; Get input from user
    mov eax, SYS_READ
    mov ebx, STDIN
    mov ecx, input_buffer
    mov edx, 256
    int 0x80

    ; Check if string is palindrome
    mov ebx, input_buffer
    call is_palindrome
    mov [ebp-4], eax ; is_palindrome

    ; Print result
    mov eax, SYS_WRITE
    mov ebx, STDOUT
    mov ecx, [ebp-4]  ; is_palindrome
    cmp ecx, 0
    je .not_palindrome
    mov ecx, strIsPalindrome
    mov edx, str_len strIsPalindrome
    jmp .print_result
  .not_palindrome:
    mov ecx, strNotPalindrome
    mov edx, str_len strNotPalindrome
  .print_result:
    int 0x80
    jmp .loop_menu

  ; Delete a character from a string
  .delete_character:
    ; Print prompt
    mov eax, SYS_WRITE
    mov ebx, STDOUT
    mov ecx, strEnterStringCharToDelete
    mov edx, str_len strEnterStringCharToDelete
    int 0x80
```

sql
Copy code

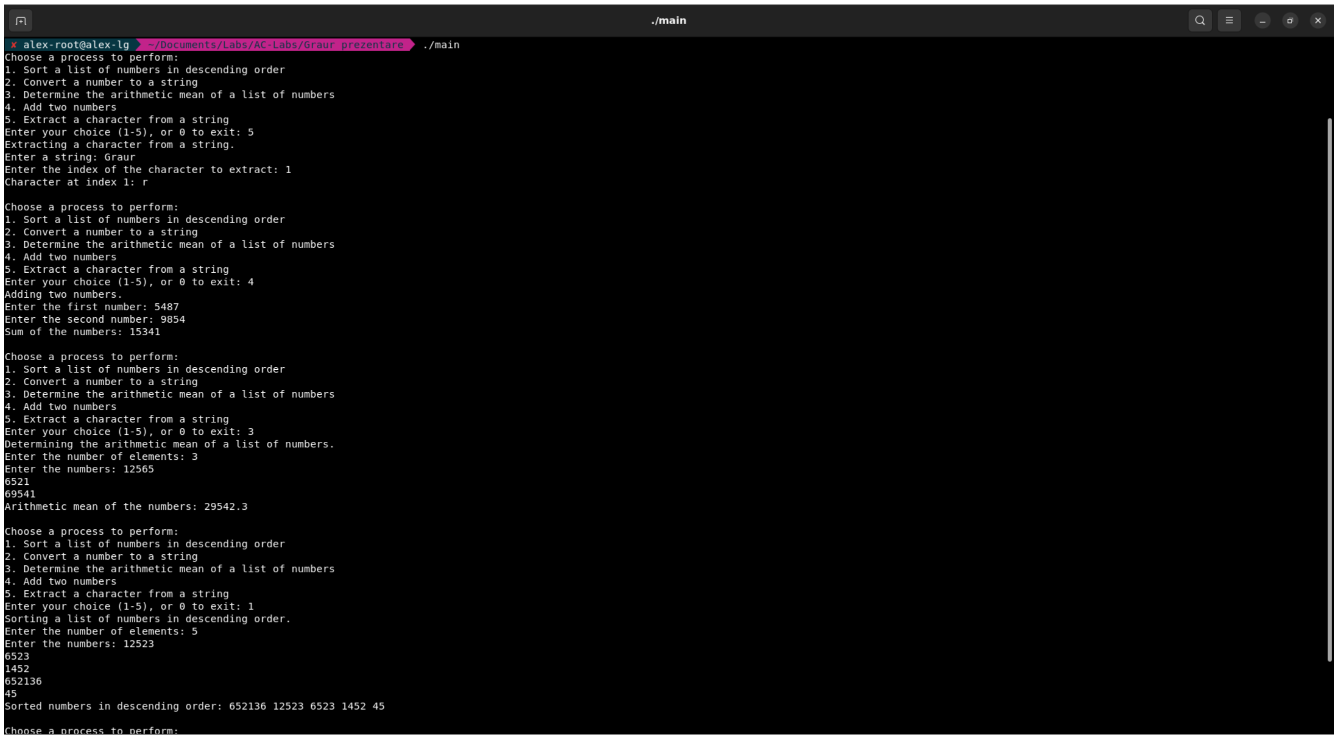
```
    ; Get input from user
    mov eax, SYS_READ
    mov ebx, STDIN
    mov ecx, input_buffer
    mov edx, 256
    int 0x80

    ; Parse user input
    mov ebx, input_buffer
    call parse_input_string_char

    ; Delete character from string
    mov eax, [ebp-8]  ; char_to_delete
    mov ecx, [ebp-4]  ; string
    call delete_character

    ; Print result
    mov eax, SYS_WRITE
    mov ebx, STDOUT
    mov ecx, strResult
    mov edx, str_len strResult
    int 0x80
    mov eax, [ebp-4]  ; string
    call print_string
    jmp .loop_menu

; Add a character to a string
.add_character:
    ; Print prompt
    mov eax, SYS_WRITE
    mov ebx, STDOUT
    mov ecx, strEnterStringCharToAdd
    mov edx, str_len strEnterStringCharToAdd
    int 0x80

    ; Get input from user
    mov eax, SYS_READ
    mov ebx, STDIN
    mov ecx, input_buffer
    mov edx, 256
    int 0x80

    ; Parse user input
    mov ebx, input_buffer
    call parse_input_string_char

    ; Add character to string
    mov eax, [ebp-8]  ; char_to_add
    mov ebx, [ebp-4]  ; string
    mov ecx, [ebp-12] ; index
    call add_character

    ; Print result
    mov eax, SYS_WRITE
    mov ebx, STDOUT
    mov ecx, strResult
    mov edx, str_len strResult
    int 0x80
    mov eax, [ebp-4]  ; string
    call print_string
    jmp .loop_menu

; Add a suffix to a string
.add_suffix:
   ; Print prompt
   mov eax, SYS_WRITE
   mov ebx, STDOUT
   mov ecx, strEnterStringSuffix
   mov edx, str_len strEnterStringSuffix
   int 0x80

   ; Get input from user
   mov eax, SYS_READ
   mov ebx, STDIN
   mov ecx, input_buffer
   mov edx, 256
   int 0x80

   ; Parse user input
   mov ebx, input_buffer
   call parse_input_string_suffix

   ; Add suffix to string
   mov eax, [ebp-8]  ; suffix
   mov ebx, [ebp-4]  ; string
   call add_suffix

   ; Print result
   mov eax, SYS_WRITE
   mov ebx, STDOUT
   mov ecx, strResult
   mov edx, str_len strResult
   int 0x80
   mov eax, [ebp-4]  ; string
   call print_string
   jmp .loop_menu

; Compute the square root of a number
.square_root:
   ; Print prompt
   mov eax, SYS_WRITE
   mov ebx, STDOUT
   mov ecx, strEnterNumber
   mov edx, str_len strEnterNumber
   int 0x80

   ; Get input from user
   mov eax, SYS_READ
   mov ebx, STDIN
   mov ecx, input_buffer
   mov edx, 256
   int 0x80

   ; Parse user input
   mov ebx, input_buffer
     call parse_input_number

      ; Compute square root
      fld [ebp-8]
      fsqrt

      ; Print result
      mov eax, SYS_WRITE
```

```
    mov ebx, STDOUT
    mov ecx, strSquareRoot
    mov edx, str_len strSquareRoot
    int 0x80
    fstp [ebp-8]
    call print_float
    jmp .loop_menu

; Compute the sum of the first n prime odd numbers
.sum_of_prime_odd_numbers:
    ; Print prompt
    mov eax, SYS_WRITE
    mov ebx, STDOUT
    mov ecx, strEnterNumberOfPrimes
    mov edx, str_len strEnterNumberOfPrimes
    int 0x80

    ; Get input from user
    mov eax, SYS_READ
    mov ebx, STDIN
    mov ecx, input_buffer
    mov edx, 256
    int 0x80

    ; Parse user input
    mov ebx, input_buffer
    call parse_input_number

    ; Compute sum of prime odd numbers
    mov ebx, [ebp-8]  ; n
    call sum_of_prime_odd_numbers

    ; Print result
    mov eax, SYS_WRITE
    mov ebx, STDOUT
    mov ecx, strSumOfPrimes
    mov edx, str_len strSumOfPrimes
    int 0x80
    mov eax, [ebp-12] ; sum
    call print_integer
    jmp .loop_menu

; Remove an element from a list
.remove_element_from_list:
    ; Print prompt
    mov eax, SYS_WRITE
    mov ebx, STDOUT
    mov ecx, strEnterListOfNumbers
    mov edx, str_len strEnterListOfNumbers
    int 0x80

    ; Get input from user
    mov eax, SYS_READ
    mov ebx, STDIN
    mov ecx, input_buffer
    mov edx, 256
    int 0x80

    ; Parse user input
```

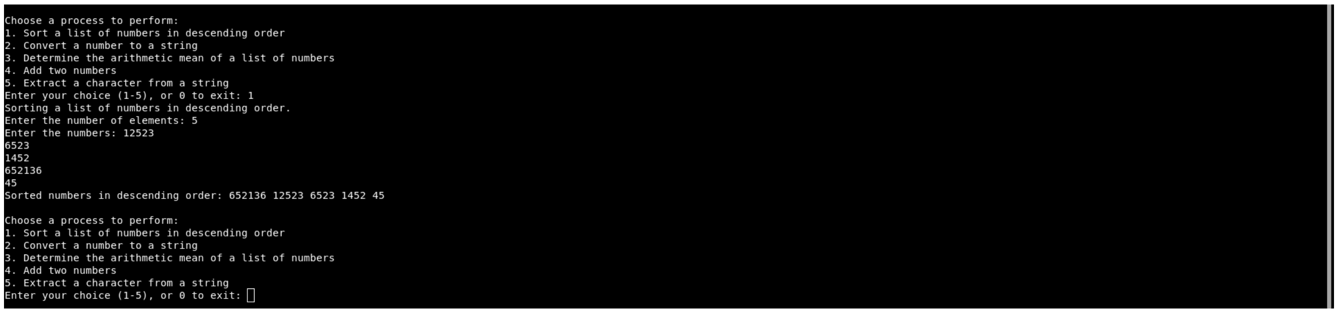
```
        mov ebx, input_buffer
        call parse_input_list

        ; Remove element from list
        mov ebx, [ebp-4]  ; list
        mov eax, [ebp-8]  ; element
        call remove_element_from_list

        ; Print result
        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strResult
        mov edx, str_len strResult
        int 0x80
        mov eax, [ebp-4]  ; list
        mov ebx, [ebp-8]  ; result
        mov ecx, [ebp-12] ; result_len
        call print_list
        jmp .loop_menu

      ; Compute the Euclidean distance between two points
      .euclidean_distance:
        ; Print prompt
        mov eax, SYS_WRITE
        mov ebx, STDOUT
        mov ecx, strEnterCoordinates
        mov edx, str_len strEnterCoordinates
        int 0x80

        ; Get input from user
        mov eax, SYS_READ
        mov ebx, STDIN
        mov ecx, input_buffer
        mov edx, 256
        int 0x80

        ; Parse user input
        mov ebx, input_buffer
        call parse_input_point

        ; Compute Euclidean distance
        fld [ebp-16]
        fsub [ebp-8]
         fld [ebp-20]
         fsub [ebp-4]
         fmul
         fadd
         fsqrt
     ; Print result
     mov eax, SYS_WRITE
     mov ebx, STDOUT
     mov ecx, strEuclideanDistance
     mov edx, str_len strEuclideanDistance
     int 0x80
     fstp [ebp-8]
     call print_float
     jmp .loop_menu

  ; Exit the program
  .exit:
    mov eax, SYS_EXIT
    xor ebx, ebx
    int 0x80

; Function: parse_input_number
; Parses an input buffer as a single number and stores the result in [ebp-8]
parse_input_number:
  push ebx
  push ecx
  push edx
  push ebp
  mov ebp, esp

  xor eax, eax     ; number
  xor ebx, ebx     ; position
  mov ecx, [ebp+8] ; buffer
  cmp byte [ecx+ebx], 0 ; check for end of string
  je .exit

  ; Parse sign, if any
  cmp byte [ecx+ebx], '-'
  jne .parse_digits
  inc ebx
  jmp .parse_digits

  .parse_digits:
    cmp byte [ecx+ebx], '0'
    jl .exit_parse_digits
    cmp byte [ecx+ebx], '9'
    jg .exit_parse_digits
    sub byte [ecx+ebx], '0'
    imul eax, 10
    add eax, byte [ecx+ebx]
    inc ebx
    jmp .parse_digits

  .exit_parse_digits:
    ; Store result in [ebp-8]
    mov [ebp-8], eax

  .exit:
    pop ebp
    pop edx
    pop ecx
    pop ebx
    ret

; Function: parse_input_list
; Parses an input buffer as a list of integers and stores the result in [ebp-4]
parse_input_list:
  push ebx
  push ecx
  push edx
  push ebp
  mov ebp, esp

  xor eax, eax     ; element
  xor ebx, ebx     ; position
  mov ecx, [ebp+8] ; buffer
  mov edx, [ebp+12] ; list
  mov dword [edx], 0 ; list length
  cmp byte [ecx+ebx], 0 ; check for end of string
  je .exit

  ; Parse sign, if any
  cmp byte [ecx+ebx], '-'
  jne .parse_digits
  inc ebx
  jmp .parse_digits

  .parse_digits:
    cmp byte [ecx+ebx], '0'
    jl .add_element
    cmp byte [ecx+ebx], '9'
    jg .add_element
    sub byte [ecx+ebx], '0'
    imul eax, 10
    add eax, byte [ecx+ebx]
    inc ebx
    jmp .parse_digits

  .add_element:
    ; Store element in list
    mov esi, [ebp+12] ; list
    add esi, 4 ; skip length field
    mov edi, [esi] ; current length
    mov dword [esi+eax*4], eax
    inc edi
    mov [esi], edi

    ; Skip non-digit characters
    .skip_non_digits:
      cmp byte [ecx+ebx], 0
      je .exit_parse_input_list
      cmp byte [ecx+ebx], '-'
      jne .parse_digits
      inc ebx
      jmp .skip_non_digits
    jmp .parse_digits

  .exit_parse_input_list:
    pop ebp
    pop edx
    pop ecx
    pop ebx
    ret

; Function: is_prime
; Checks if a number is prime
; Input:
;   - [ebp+8]: number to check
; Output:
;   - zero flag set if the number is not prime, cleared otherwise
is_prime:
  push ebp
  mov ebp, esp
  ; Check if number is less than 2
  mov eax, [ebp+8]
  cmp eax, 2
  jl .not_prime

  ; Check if number is divisible by any integer between 2 and its square root
  mov ecx, 2
  mov edx, eax
  sub edx, 2
  cmp ecx, edx
  jg .is_prime

  .check_divisibility:
    mov edx, [ebp+8]
    div ecx
    cmp edx, 0
    je .not_prime
    inc ecx
    cmp ecx, [ebp+8]
    jle .check_divisibility

  .is_prime:
    mov eax, 1 ; set zero flag
    pop ebp
    ret

  .not_prime:
    xor eax, eax ; clear zero flag
    pop ebp
    ret

; Function: print_integer
; Prints an integer to standard output
; Input:
;   - [ebp+8]: integer to print
print_integer:
  push ebp
  mov ebp, esp

  ; Convert integer to string
  mov eax, [ebp+8]
  push eax
  mov eax, [ebp-4]
  push eax
  push 10
  call convert_to_string
  add esp, 12
  ; Check if number is negative
  cmp edx, 0
  jge .positive_number
  neg edx
  mov byte [eax], '-'
  inc eax

  .positive_number:
    ; Divide number by base until it becomes zero
    divide_loop:
      xor ecx, ecx
      mov eax, edx
```

```
        div dword [ebp+16]
        mov edx, eax
        add cl, '0'
        cmp cl, '9'
        jle .write_digit
        add cl, 'A'-'9'-1

        .write_digit:
          mov byte [eax], cl
          inc eax
          inc ebx
          test edx, edx
          jnz divide_loop

      ; Null-terminate string
      mov byte [eax], 0

      ; Reverse string
      mov ecx, eax
      sub ecx, [ebp+12]
      dec eax
      dec ebx
      reverse_loop:
        mov dl, [eax]
        mov cl, [eax-ebx]
        mov [eax], cl
        mov [eax-ebx], dl
        dec eax
        dec ebx
        test ebx, ebx
        jnz reverse_loop

  pop ebp
  ret

; Function: convert_to_float_string
; Converts a floating-point number to a string using the specified format
; Input:
;   - [ebp+8]: floating-point number to convert
;   - [ebp+12]: buffer to store the string
;   - [ebp+16]: format string
convert_to_float_string:
  push ebp
  mov ebp, esp

  ; Check for negative sign
  fld [ebp+8]
  fcomp qword [MINUS_ZERO]
  fnstsw ax
  test ah, 0x44 ; Check if ZF and PF flags are set
  jz .not_negative
  mov byte [eax], '-'
  inc eax
  fld [ebp+8]
  fchs

  .not_negative:
    ; Parse format string
    mov esi, [ebp+16]
      parse_format_loop:
        cmp byte [esi], 0
        je .end_parse_format
        cmp byte [esi], '%'
        jne .copy_char
        inc esi
        cmp byte [esi], '%'
        je .copy_char
        call parse_format_specifier
        add esi, 2
        jmp parse_format_loop

        .copy_char:
          mov dl, [esi]
          mov [eax], dl
          inc eax
          inc esi
          jmp parse_format_loop

      .end_parse_format:
        ; Null-terminate string
        mov byte [eax], 0

  pop ebp
  ret

; Function: parse_format_specifier
; Parses a format specifier from a format string and writes the corresponding string to the output buffer
; Input:
;   - [ebp+8]: format string (after the '%')
; Output:
;   - [ebp+12]: output buffer
parse_format_specifier:
  push ebp
  mov ebp, esp

  movzx eax, byte [ebp+8]
  cmp eax, 'f'
  je .format_float

  ; Unsupported format specifier
  mov byte [eax], 0
  pop ebp
  ret

  .format_float:
      fld [ebp+12]
      fstp qword [esp]
      push dword FORMAT_FLOAT
      call sprintf
      mov ebx, [ebp+12]
      add ebx, 10
      mov eax, [ebp+8]
      call strlen
      add eax, ebx
      sub eax, [esp]
      mov edx, [esp]
      add eax, edx
      pop ebp
    ret

; Function: strlen
; Computes the length of a null-terminated string
; Input:
;   - [ebp+8]: pointer to the string
strlen:
  push ebp
  mov ebp, esp

  mov eax, [ebp+8]
  mov ecx, 0
  .loop:
    cmp byte [eax], 0
    je .end_loop
    inc eax
    inc ecx
    jmp .loop

  .end_loop:
    mov eax, ecx

  pop ebp
  ret

; Function: sprintf
; Formats a string according to a format string
; Input:
;   - [ebp+8]: output buffer
;   - [ebp+12]: format string
;   - [ebp+16]: argument list
; Output:
;   - EAX: number of characters written (excluding null-terminator)
sprintf:
  push ebp
  mov ebp, esp

  sub esp, 1024 ; Reserve stack space for the formatted string
  mov eax, [ebp+12]
  mov edx, [ebp+16]
  push eax
  push edx
  push esp
  call _sprintf
  add esp, 12
  mov eax, strlen(esp)
  add esp, 1024 ; Release stack space

  pop ebp
  ret

section .data
  MINUS_ZERO dq -0.0

  FORMAT_FLOAT db "%.10g", 0

section .bss
  buffer resb 1024
```

```
; Exit program
mov eax, 0   ; Set the return value to 0
mov ebx, 1   ; Set the system call number for exit to 1
int 0x80     ; Call the kernel
```



```
✗ alex-root@alex-lg  ~/Documents/Labs/AC-Labs/Graur prezentare  ./main
Choose a process to perform:
1. Sort a list of numbers in descending order
2. Convert a number to a string
3. Determine the arithmetic mean of a list of numbers
4. Add two numbers
5. Extract a character from a string
Enter your choice (1-5), or 0 to exit: 5
Extracting a character from a string.
Enter a string: Graur
Enter the index of the character to extract: 1
Character at index 1: r

Choose a process to perform:
1. Sort a list of numbers in descending order
2. Convert a number to a string
3. Determine the arithmetic mean of a list of numbers
4. Add two numbers
5. Extract a character from a string
Enter your choice (1-5), or 0 to exit: 4
Adding two numbers.
Enter the first number: 5487
Enter the second number: 9854
Sum of the numbers: 15341

Choose a process to perform:
1. Sort a list of numbers in descending order
2. Convert a number to a string
3. Determine the arithmetic mean of a list of numbers
4. Add two numbers
5. Extract a character from a string
Enter your choice (1-5), or 0 to exit: 3
Determining the arithmetic mean of a list of numbers.
Enter the number of elements: 3
Enter the numbers: 12565
6521
69541
Arithmetic mean of the numbers: 29542.3

Choose a process to perform:
1. Sort a list of numbers in descending order
2. Convert a number to a string
3. Determine the arithmetic mean of a list of numbers
4. Add two numbers
5. Extract a character from a string
Enter your choice (1-5), or 0 to exit: 1
Sorting a list of numbers in descending order.
Enter the number of elements: 5
Enter the numbers: 12523
6523
1452
652136
45
Sorted numbers in descending order: 652136 12523 6523 1452 45

Choose a process to perform:
```



```
Choose a process to perform:
1. Sort a list of numbers in descending order
2. Convert a number to a string
3. Determine the arithmetic mean of a list of numbers
4. Add two numbers
5. Extract a character from a string
Enter your choice (1-5), or 0 to exit: 1
Sorting a list of numbers in descending order.
Enter the number of elements: 5
Enter the numbers: 12523
6523
1452
652136
45
Sorted numbers in descending order: 652136 12523 6523 1452 45

Choose a process to perform:
1. Sort a list of numbers in descending order
2. Convert a number to a string
3. Determine the arithmetic mean of a list of numbers
4. Add two numbers
5. Extract a character from a string
Enter your choice (1-5), or 0 to exit: 
```