Федеральное государственное бюджетное образовательное учреждение высшего образования «Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича»

Кафедра «Программная инженерия и вычислительная техника»

«Машино-зависимые языки программирования»

Отчет

по лабораторной работе №4

«Обработка одномерных массивов»

|  |  |
| --- | --- |
|  | Выполнил |
|  | студент группы ИКПИ-33 |
|  | М.Д. Коньков |
|  |  |
|  | Проверил |
|  | Ст. преподаватель |
|  | И.Л. Савельев |
|  |  |

2024 г.

**1. Задание**

Задав одномерный массив целочисленных данных А в одном из заданных форматов (unsigned char — BYTE, unsigned short int — WORD, char — SHORTEST, short int — INTEGER или long int — LONGINT), реализовать обработку массива, как указано в варианте. Длина массива N. Исходные данные задать самостоятельно, учитывая формат элементов массива А.

Программа должна быть написана целиком **на языке Ассемблера.**

Исходные данные задаются в текстовом файле. Разделитель значений – на усмотрение исполнителя. Размер файла не более 2000 байт. Исходные данные должны вводиться корректно и быть приближенными к максимально возможным для данного типа данных. Тип результата определяется из контекста задачи

**Вариант №5**

Найти сколько элементов массива A={a[i]} удовлетворяет условию:

d <= a[i] <= c

**2. Листинг программы**

**2.1. Модуль program.asm**

section .data

newline db 0xA ; Новая строка

comma db ", " ; Запятая и пробел

comma\_len equ $ - comma

file\_error\_msg db "File open error", 0xA

file\_error\_len equ $ - file\_error\_msg

input\_error\_msg db "Input error", 0xA

input\_error\_len equ $ - input\_error\_msg

array\_full\_msg db "Array is full", 0xA

array\_full\_len equ $ - array\_full\_msg

result\_msg db "Number of negative elements satisfying the condition: ", 0

result\_msg\_len equ $ - result\_msg

total\_count\_msg db "Total count: ", 0

total\_count\_msg\_len equ $ - total\_count\_msg

parse\_error\_msg db "Parse error", 0xA

parse\_error\_msg\_len equ $ - parse\_error\_msg

c dw -32768 ; Нижняя граница

d dw -20000 ; Верхняя граница

buffer\_size equ 4096 ; Размер буфера

section .bss

array resw 2000 ; Массив чисел

n resd 305 ; Количество элементов

buffer resb buffer\_size ; Буфер

fd resq 1 ; Файловый дескриптор

printed\_count resd 1 ; Счетчик выведенных чисел

input\_ended resb 1 ; Флаг конца ввода

num\_counter resd 1 ; Счетчик чисел

; Системные вызовы

SYS\_READ equ 0

SYS\_WRITE equ 1

SYS\_OPEN equ 2

SYS\_CLOSE equ 3

SYS\_EXIT equ 60

O\_RDONLY equ 0 ; Флаг: чтение

section .text

global \_start

\_start:

mov rdi, [rsp+8] ; Количество аргументов

; Проверка аргументов

cmp rdi, 2

jl file\_error

; Имя файла

mov rsi, [rsp + 16]

; Открытие файла

mov rax, SYS\_OPEN

mov rdi, rsi

mov rsi, O\_RDONLY

syscall

js file\_error ; Обработка ошибки

mov [fd], rax ; Сохранение дескриптора

mov dword [n], 305 ; Инициализация n

read\_loop:

; Чтение из файла в буфер

mov rax, SYS\_READ

mov rdi, [fd] ; Файловый дескриптор

lea rsi, [buffer] ; Адрес буфера

mov rdx, buffer\_size ; Размер буфера

syscall

cmp rax, 0 ; Проверка на конец файла (EOF)

jle end\_read\_loop ; Переход, если конец файла или ошибка

lea rdi, [buffer] ; Указатель на начало буфера

mov r15, rax ; Сохраняем количество прочитанных байт

parse\_loop:

cmp r14, [n] ; Проверка на максимальное количество элементов

jge array\_bound\_error

push rdi

push r15 ; Счетчик оставшихся байт

call parse\_int ; Парсим число из буфера

pop r15

pop rdi

jc parse\_error\_handler ; Переход при ошибке парсинга

; Если число спаршено без ошибки

sub r15, r8 ; Уменьшаем счетчик байт

mov dword [array + r14\*4], eax ; Сохраняем число в массив

inc r14 ; Увеличиваем счетчик спаршенных чисел

add rdi, r8 ; Пропускаем обработанные символы

inc rdi ; Обновляем rdi

jmp check\_end\_of\_buffer ; Проверка на необходимость чтения новых данных

; Обработчик ошибок парсинга

parse\_error\_handler:

mov rdi, r10 ; Используем сохраненное значение rdi

; Если конец буфера

cmp r15, 0

jle end\_of\_input\_check

; Если не конец буфера, проверяем на запятую

cmp byte [rdi], ','

jne input\_error ; Если не запятая, то ошибка ввода

; Если запятая, пропускаем её

inc r10 ; Пропускаем запятую

dec r15 ; Уменьшаем количество оставшихся байтов

mov r10, rdi ; Перемещаем указатель

jmp check\_end\_of\_buffer

check\_end\_of\_buffer:

cmp r14, [n]

je process\_loop ; Если n чисел спаршено

cmp r15, 0 ; Проверка на пустой буфер

jle read\_loop ; Если буфер пуст, читаем следующую порцию

jmp skip\_separators ; Иначе продолжаем парсинг

process\_array:

jmp end\_read\_loop ; Переход к обработке массива

skip\_separators:

cmp byte [rdi], ',' ; Проверка на запятую

je next\_char

jmp parse\_loop

next\_char:

inc r10 ; Пропуск запятой

dec r15 ; Счетчик оставшихся байт

mov rdi, r10 ; Обновляем rdi

; Пропуск разделителей

cmp r15, 0 ; Проверка на конец буфера

je read\_loop ; Если конец буфера, читаем следующую порцию

jmp skip\_separators

end\_of\_input\_check:

mov byte [input\_ended], 1 ;

jmp end\_read\_loop ; Завершение чтения

end\_read\_loop:

xor r13w, r13w ; Индекс массива

xor eax, eax ; Счетчик

mov r12w, [n] ; Инициализируем счетчик цикла значением n

xor dword [printed\_count], 0 ; Обнуляем счетчик выведенных чисел

process\_loop:

cmp r13w, r14w ; Сравниваем текущий индекс с количеством чисел

jge output\_result ; Если все числа обработаны

mov r15w, [array + r13\*4] ; Загружаем текущий элемент

cmp r15w, 0

jge next\_element ; Если число неотрицательное, переходим к следующему

; Проверка диапазона [c, d] для отрицательных чисел

mov r8w, [c]

cmp r15w, r8w

jl next\_element

mov r9w, [d]

cmp r15w, r9w

jg next\_element

print\_element:

; Увеличиваем счетчик запятых перед выводом запятой

inc dword [num\_counter]

; Вывод запятой, если это не первый выведенный элемент

cmp dword [printed\_count], 0

je print\_number ; Если первый, пропускаем вывод запятой

;Вывод запятой

mov rsi, comma

mov rdx, comma\_len

call print\_string

print\_number:

movsx rdi, r15w ; Передаем число для вывода

call print\_int ; Выводим число

inc dword [printed\_count] ; Увеличиваем счетчик выведенных чисел

next\_element:

inc r13w ; Увеличиваем индекс для следующего элемента

jmp process\_loop ; Переход к следующему элементу

output\_result: ; Вывод результата

mov rdi, result\_msg ; Сообщение о результате

mov rsi, result\_msg\_len ; Длина сообщения

call print\_string

call print\_newline

; Вывод общего количества чисел

mov rsi, total\_count\_msg

mov rdx, total\_count\_msg\_len

call print\_string

mov rdi, [num\_counter]

call print\_int

call print\_newline

file\_error:

mov rdi, file\_error\_msg

mov rsi, file\_error\_len

call print\_string

jmp exit\_program

input\_error:

mov rdi, input\_error\_msg

mov rsi, input\_error\_len

call print\_string

jmp exit\_program

parse\_error:

mov rdi, parse\_error\_msg

mov rsi, parse\_error\_msg\_len

call print\_string

jmp exit\_program

array\_bound\_error:

mov rdi, array\_full\_msg

mov rsi, array\_full\_len

call print\_string

jmp exit\_program

parse\_int:

push r9

push rbx

push r12

push rdi

push rsi

xor eax, eax ; Результат

xor ebx, ebx ; Флаг знака

xor r12, r12 ; Счетчик цифр

xor r9, r9 ; Спаршенное число

; Проверка на знак

cmp byte [rdi], '-'

jne .positive

inc ebx

inc rdi ; Переход к следующему символу

.positive:

.parse\_digit\_loop:

movzx edx, byte [rdi] ; Читаем символ

cmp edx, '0'

jl .check\_end ; Если не цифра, проверяем конец

cmp edx, '9'

jg .check\_end ; Если не цифра, проверяем конец

; Если цифра:

inc r12d ; Увеличиваем счетчик цифр

imul r9, 10

sub edx, '0'

add r9d, edx ; Добавляем цифру к числу

inc rdi ; Переходим к следующему символу

jmp .parse\_digit\_loop

.check\_end:

; Проверка на запятую (конец числа)

cmp edx, ','

je .parse\_end ; Если запятая, то конец числа

jmp .parse\_error

.parse\_end:

cmp r12d, 0 ; Проверяем, были ли прочитаны цифры

je .parse\_error

; Установка знака

test ebx, ebx ; Проверяем флаг отрицательного числа (ebx = 0, если число положительное)

jnz .negative

jmp .prepare\_return

.negative:

neg r9

.prepare\_return:

mov rax, rdi ; Сохраняем текущую позицию в буфере (rdi) в rax

pop rsi

pop rdi

sub rax, rdi ; Вычисляем длину прочитанного числа (текущая позиция - начальная позиция)

mov r8d, eax ; Сохраняем длину в r8d (возвращаемое значение длины)

pop r12

pop rbx

mov eax, r9d

pop r9

clc ; Сбрасываем флаг переноса CF (нет ошибки)

ret

.parse\_error:

xor eax, eax ; Возвращаем 0 в eax при ошибке

pop r9

pop rsi

pop rdi

pop r12

pop rbx

stc ; Устанавливаем флаг переноса CF (ошибка)

ret

; Функция для вывода строки на экран

; rdi: адрес строки

; rsi: длина строки

print\_string:

mov rax, SYS\_WRITE

mov rdi, 1

syscall

ret

; Функция для вывода целого числа на экран

; rdi: число для печати

print\_int:

push rbp

mov rbp, rsp ; Устанавливаем rbp на вершину стека

sub rsp, 2 ; 2 байта на стеке для буфера строки

mov r8, 10

mov r9, rsp ; Указатель на буфер строки

mov rax, rdi ; Копируем число в rax

test rax, rax ; Проверяем число на знак

jns positive\_loop

neg rax ; Изменяем знак числа на противоположный

positive\_loop:

xor rdx, rdx

div r8

add dl, '0'

mov [r9], dl ; Записываем символ в буфер

inc r9 ; Переходим к следующему символу в буфере

test rax, rax

jnz positive\_loop

; Обработка знака после обработки цифр

cmp rdi, 0 ; Сравниваем исходное число с 0

jge no\_sign

mov byte [r9], '-' ; Добавляем знак минуса в буфер

inc r9

no\_sign:

mov byte [r9], 0 ; Нуль-терминатор строки

; Переворачиваем строку для правильного вывода

mov r10, rsp ; Начало строки

mov r11, r9 ; Конец строки

dec r11 ; Уменьшаем r11 на 1 (исключаем нуль-терминатор)

reverse\_loop:

cmp r10, r11 ; Сравниваем начало и конец строки

jge end\_reverse

mov al, [r10] ; Обмен символов

mov bl, [r11]

mov [r10], bl

mov [r11], al

inc r10 ; Переходим к следующему символу

dec r11

jmp reverse\_loop

end\_reverse:

; Вычисляем длину строки

mov rax, r9

sub rax, rsp

mov rdx, rax ; Длина строки в rdx

; Выводим строку

mov rsi, rsp

mov rax, SYS\_WRITE ;

mov rdi, 1

syscall

leave ; Восстанавливаем rsp и rbp

ret

; Функция для вывода новой строки

print\_newline:

mov rsi, newline

mov rdi, 1

mov rdx, 1

call print\_string

ret

; Функция для завершения программы

exit\_program:

mov rax, SYS\_EXIT

xor rdi, rdi ; Код выхода 0

syscall

**2.2. Модуль makefile**

all:

nasm -f elf64 -F dwarf -g program.asm -o program.o

ld -m elf\_x86\_64 -O0 -o program program.o

./program input.txt #Передача файла в программу

clean:

rm -rf program.o

**3. Сборка проекта**

make

**4. Выполнение программы**

**4.1. Запуск программы**

./program input.txt

**4.2. Входные данные**

**c = -32000**

**d = 32000**

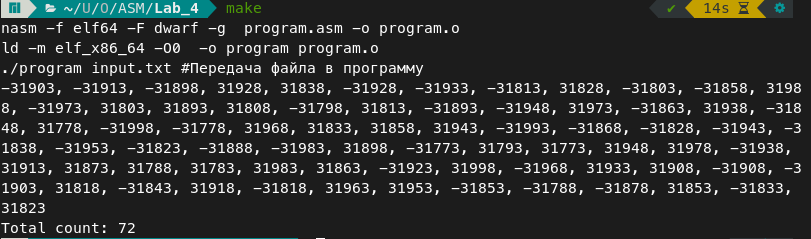
**Файл с исходными числами input.txt**

-31903,32613,32623,-32473,-32423,32298,-32053,32648,32358,-32263,32703,-32068,-32738,-32303,-32288,32678,32378,-31913,32018,-32723,-32408,32323,-32698,-32653,-31898,31928,-32073,31838,-31928,32338,-32298,-32503,-32553,32083,32063,-31933,-31813,32403,-32123,-32628,31828,-31803,-32358,-32268,-32243,-32753,-31858,32073,-32428,31988,-31973,-32603,-32028,-32648,-32483,31803,32698,32488,32228,-32283,32548,32518,32658,31893,31808,32168,32038,-32143,32508,32663,32208,32148,-31798,-32583,32583,-32258,32388,-32488,-32013,-32188,-32728,32078,32688,-32003,32503,32138,-32248,-32683,32273,31813,-31893,-32083,32153,32723,-32623,32348,32333,32373,-32118,-32208,-32403,32633,-32103,-32468,32183,-32638,32248,32268,32733,32553,-31948,-32463,32753,-32153,-32223,31973,-32163,-32348,32438,32413,-31863,31938,-32558,32683,-32458,-32563,-32548,-32433,32628,-31848,32573,-32168,-32618,32543,32443,32528,32113,-32593,31778,-32518,-32513,-32688,-32058,32748,-32203,-31998,32743,-31778,31968,31833,-32278,-32643,32498,32368,32763,31858,-32538,32563,-32218,31943,-32343,-32233,-32033,-32113,32088,-32353,32458,-32608,-32578,-31993,-32293,-31868,-31828,32008,32053,-31943,32738,-32413,-32658,-31838,-32718,-32598,-31953,32023,-31823,-32018,32178,-31888,-32318,-32378,32313,32283,32193,32608,32533,32418,32603,32673,32058,-32238,32203,-31983,32093,-32338,-32308,31898,-32613,-32668,-31773,31793,-32713,-32543,32638,-32633,-32478,31773,31948,31978,-31938,32448,-32568,-32393,32328,31913,32693,31873,32163,32218,32428,-32228,-32193,-32008,32293,32558,31788,32238,31783,32303,32588,31983,32453,32278,-32748,-32063,32653,-32418,31863,-31923,-32523,31998,-32708,-32768,-32183,-31968,32433,32523,32513,-32128,-32328,31933,32593,31908,-32023,32103,32758,32568,-32373,-32253,-32313,32188,32258,-31908,-31903,-32038,31818,32048,-31843,31918,32668,-32588,-31818,32233,31963,32473,31953,32728,32043,32123,-31853,-32048,-32388,-32273,-31788,-31878,31853,-32213,32398,-32763,-31833,-32363,-32148,31823,32128,32408,-32703,

**4.3. Ожидаемый результат выполнения**

Количество подходящих чисел: 72 (а также вывод все этих чисел)

**4.4. Результат выполнения**



**5. Вывод**

Результат выполнения программы соответствует ожидаемому результату. Работа выполнена в полном объеме, дополнительно выводятся значения из файла, которые соответствуют изначальным условиям.