**Files in assembly. Text files operations.**

In assembly, a **file** is a sequence of bytes.

The operations on a file in assembly are general operation on a file in different programming language.

1. Create a file
2. Open a file
3. Read from a file
4. Write in a file
5. Close a file

We will execute all these operation using **external function** from **msvcrt.dll library.**

For **opening an existing file or creating a new file**, we use the **fopen** function. The fopen function respects the **cdecl convention.**

**CDECL convention rules:**

* Arguments of an external function are placed on the stack from right to left (**an element from a stack is a doubleword**);
* The result returned by the external functions are stored in EAX register;
* Registers EAX, ECX, EDX are used within the functions (so their values can be overwritten) therefore if we need the values which are stored in EAX, ECX, EDX we have to save them (in auxiliary variables or save them on the stack before using the external functions and redone their values after we execute the external functions) before the function call.
* The external functions do not empty the stack; it is the responsibility of the programmer to take out the arguments out of the stack after the function call to clear the stack it is used a computation related to ESP – extended stack pointer: add esp, 4\*number of doublewords saved on the stack)

**Syntax of the function: fopen in a high level programming language:**

**FILE \* fopen (const char\* file\_name, const char \* access\_mode)**

### Arguments/Parameters of the fopen function meaning:

The first argument - **file\_name -** of the function is the address of a character string containing the name of the file.

The second argument -**access\_mode**- is the address of a character string containing the access mode for opening the file.

|  |  |  |
| --- | --- | --- |
| Access mode | Meaning | Description |
| r | read | * Open file for reading. The file must exist. |
| w | write | * If the file does not exist, it creates a new file with the given name and opens it for writing. * If a file with the given name exists, it opens it for writing. It overwrites the content of the file and starts writing from the beginning of the file. |
| a | append | * If the file does not exist, it creates a new file and opens it for writing. * If a file with the given name exists, it opens it for writing. It does not overwrite the content; it continues writing at the end of the file. |
| r+ | read and write from/into existing file | * Open file for reading and writing. The file must exist. |
| w+ | read and write | * If the file does not exist, it creates a new file and opens it for reading and writing. * If a file with the given name exists, it opens it for reading and writing. It overwrites the content of the file and starts writing at the beginning of the file. |
| a+ | read and append | * If the file does not exist, it creates a new file and opens it for reading and writing. * If a file with the given name exists, it opens it for reading and writing. It does not overwrite the content; it continues writing at the end of the file. |

### *Observations:*

* The name of the file must include the extension (ex: name.txt, example.asm).
* Files are created or opened in the current directory (in the same directory where the .asm source is located).
* Important: in order to open a file using its name, the file must be placed in the same folder as the asm source file, otherwise ***the file will not be found***.
* Writing operations will fail if the file was opened only for reading (ex. access mode "r"). Reading operations will fail for files opened only for writing or appending (ex. access mode "w", "a").
* Both arguments of the fopen function represent character strings that have to be terminated with a 0 (similar to the format for the printf function) – it’s mandatory to be string asciiz.

#### The value returned by the fopen function:

If the file is successfully opened, **EAX will contain the file descriptor (an identifier)** which can be used for working with the file (reading and writing). If an error occurs, fopen will set EAX to the value 0.

It is important to check the value returned by the function in EAX before continuing to work with the file, in order to know whether the file was correctly opened.

If a program opens more files using the fopen function, each value returned by the function must be stored separately, since each file has a unique identifier.

When we are done working with a file, it is important to close the file ( it can be done at the end of the program - before exit).

For closing the file we use the fclose function.

### Syntax of the function****fclose**** in a high level programming language:

int fclose(FILE \* descriptor)

The fclose function respects the cdecl convention and it can be found in the msvcrt.dll library.

### *Argument of the fclose function*

The argument of the fclose function is the file descriptor (identifier) returned by the fopen function call.

Exemples:

Ex1. **Create** a file with the name ‘towrite.txt’, and with access mode w.

bits 32

global start

; declare external functions needed by our program

extern exit**,** fopen**,** fclose

import exit msvcrt.dll

import fopen msvcrt.dll

import fclose msvcrt.dll

; our data is declared here (the variables needed by our program)

segment data use32 class**=**data

file\_name db "towrite.txt"**,** 0 ; filename to be created

access\_mode db "w"**,** 0 ; file access mode:

; w - creates an empty file for writing

file\_descriptor dd **-**1 ; variable to hold the file descriptor

; our code starts here

segment code use32 class**=**code

start**:**

; call fopen() to create the file; fopen() will return a file descriptor in the EAX or 0 in case of error

; eax = fopen(file\_name, access\_mode)

**push** **dword** access\_mode

**push** **dword** file\_name

**call** **[**fopen**]**

**add** **esp,** 4**\***2 ; clean-up the stack

**mov** **[**file\_descriptor**],** **eax** ; store the file descriptor returned by fopen

; check if fopen() has successfully created the file (EAX != 0)

**cmp** **eax,** 0

**je** final

; call fclose() to close the file

; fclose(file\_descriptor)

**push** **dword** **[**file\_descriptor**]**

**call** **[**fclose**]**

**add** **esp,** 4\*1

final**:**

; exit(0)

**push** **dword** 0

**call** **[**exit**]**

Ex2. Open an existing file with the name ‘toread.txt’. The file toread.txt exists in current folder:

bits 32

global start; declare external functions needed by our program

extern exit**,** fopen**,** fclose**,** printf

import exit msvcrt.dll

import fopen msvcrt.dll

import fclose msvcrt.dll

import printf msvcrt.dll

; our data is declared here (the variables needed by our program)

segment data use32 class**=**data

file\_name db 'toread.txt'**,** 0 ; filename to be created

access\_mode db "r"**,** 0 ; file access mode:

; r - creates an empty file for reading

file\_descriptor dd **-**1 ; variable to hold the file descriptor

eroaredeschidere db ' unable to open the file'**,** 0

; our code starts here

segment code use32 class**=**code

start**:**

; call fopen() to create the file

; fopen() will return a file descriptor in the EAX or 0 in case of error

; eax = fopen(file\_name, access\_mode)

**push** **dword** access\_mode

**push** **dword** file\_name

**call** **[**fopen**]**

**add** **esp,** 4**\***2 ; clean-up the stack

**mov** **[**file\_descriptor**],** **eax** ; store the file descriptor returned by fopen

; check if fopen() has successfully created the file (EAX != 0)

**cmp** **eax,** 0

**je** eroareOpen

; call fclose() to close the file

; fclose(file\_descriptor)

**push** **dword** **[**file\_descriptor**]**

**call** **[**fclose**]**

**add** **esp,** 4\*1

**jmp** final

eroareOpen**:**

;print on the screen a message

**push** **dword** eroaredeschidere

**call** **[**printf**]**

**add** **esp,** 4**\***1

final**:**

; exit(0)

**push** **dword** 0

**call** **[**exit**]**

For writing text into a file we use the **fprintf** function from msvcrt.dll library

### *Syntax of the function* ****fprintf**** in a high level programming language:

**int fprintf(FILE \* stream (=file descriptor), const char \* format, <variable\_1>, <constant\_2>, <...>)**

The fprintf function respects the cdecl convention and it can be found in the msvcrt.dll library.

Syntax of the fprintf function is similar to the syntax of the printf function (used for printing on the screen).

The difference is that, in addition to the parameters of the printf function, the fprintf function has the file descriptor as the first argument.

### *Arguments of the fprintf function*

First argument -stream- of the function represents the file descriptor (identifier) returned by the fopen function call.

The next argument of the function -format- is a character string containing the format for printing, followed by the same number of arguments (constant values or variable names) as specified in the format. Similar to the printf function, the character string representing the format can contain certain formatting markups, starting with the character ’%’, which will be replaced by the values given in the following arguments.

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Type** | **Example** | **Value representation dimension** |
| c | Character | a | byte |
| **d** or i | Signed decimal integer | 392 | dword |
| u | Unsigned decimal integer | 7235 | dword |
| **x** | hexadecimal integer | 7fa | dword |
| s | String (terminated with a 0) | example | string of bytes terminated with 0 |

Ex 3. Print a text defined in data segment into a file called ‘addextratext.txt’. The file exists and has a content.

bits 32

global start

; declare external functions needed by our program

extern exit**,** fopen**,** fprintf**,** fclose

import exit msvcrt.dll

import fopen msvcrt.dll

import fprintf msvcrt.dll

import fclose msvcrt.dll

; our data is declared here (the variables needed by our program)

segment data use32 class**=**data

file\_name db "addextratext.txt"**,** 0 ; filename to be read

access\_mode db "a"**,** 0 ; file access mode:

; a - appends to a file. Append data at

; the end of the file.

file\_descriptor dd **-**1 ; variable to hold the file descriptor

text db "hello, this text is added extra into a file."**,** 0 ; text to append to file

; our code starts here

segment code use32 class**=**code

start**:**

; call fopen() to create the file

; fopen() will return a file descriptor in the EAX or 0 in case of error

; eax = fopen(file\_name, access\_mode)

**push** **dword** access\_mode

**push** **dword** file\_name

**call** **[**fopen**]**

**add** **esp,** 4**\***2 ; clean-up the stack

**mov** **[**file\_descriptor**],** **eax** ; store the file descriptor returned by fopen

; check if fopen() has successfully created the file (EAX != 0)

**cmp** **eax,** 0

**je** final

; append the text to file using fprintf()

; fprintf(file\_descriptor, text)

**push** **dword** text

**push** **dword** **[**file\_descriptor**]**

**call** **[**fprintf**]**

**add** **esp,** 4**\***2

; call fclose() to close the file

; fclose(file\_descriptor)

**push** **dword** **[**file\_descriptor**]**

**call** **[**fclose**]**

**add** **esp,** 4

final**:**

; exit(0)

**push** **dword** 0

**call** **[**exit**]**

For reading from a file we use the **fread** function.

### *Syntax of the function* **fread** in a high level programming language:

int fread(void \* str, int size, int count, FILE \* stream=file descriptor)

The fread function respects the cdecl convention and it can be found in the msvcrt.dll library.

### *Arguments of the fread function*

The first argument of the fread functions -str- represents the address of a string where the data read from file is stored.

The second argument -size- represent the size of one element that will be read from the file.

The third argument -count- represents the maximum number of elements to be read.

The last argument -stream- of the function represents the file descriptor (identifier) returned by the fopen function call.

**When reading from a text file, the first argument of the fread function is a byte string and the second argument is 1 (= dimension of one byte). The third argument is the size of the byte string (number of elements).**

### *The value returned by the fread function:*

The fread function stores in EAX the number of elements read. If this number is below the value of the count argument, it means that either there was an error, or that the function got to the end of the file.

### *Observations:*

If text files have large sizes, we cannot read the whole content of the file with one function call.

In this case multiple fread function calls are necessary, until the whole content of the file is read.

In order to check whether we got to the end of the file, we can check if the value returned by fread is 0.

Ex4. Read the content from a file (the file exists in the folder and has a content on multiple lines - different phrases) and print this text on the screen and then print the content into another file.

bits 32

global start

; declare external functions needed by our program

extern exit**,** fopen**,** fread**,** fclose**,** printf**,** fprintf

import exit msvcrt.dll

import fopen msvcrt.dll

import fread msvcrt.dll

import fclose msvcrt.dll

import printf msvcrt.dll

import fprintf msvcrt.dll

; our data is declared here (the variables needed by our program)

segment data use32 class**=**data

file\_name1 db "filewithlongtext.txt"**,** 0 ; filename to be read

access\_mode1 db "r"**,** 0 ; file access mode:

; r - opens a file for reading. The file must exist.

file\_descriptor1 dd **-**1 ; variable to hold the file descriptor

len equ 300 ; maximum number of characters to read

text times **(**len**+**1**)** db 0 ; string to hold the text which is read from file

;text resb len+1

format db 'We have read %d chars from file'**,** 10 **,** ' The text is: %s'**,** 0

file\_name2 db "savedinlongtext.txt"**,** 0 ; filename to be read

access\_mode2 db "w"**,** 0 ; file access mode:

; r - opens a file for reading. The file must exist.

file\_descriptor2 dd **-**1 ; variable to hold the file descriptor

; our code starts here

segment code use32 class**=**code

start**:**

; call fopen() to create the file

; fopen() will return a file descriptor in the EAX or 0 in case of error

; eax = fopen(file\_name, access\_mode)

**push** **dword** access\_mode1

**push** **dword** file\_name1

**call** **[**fopen**]**

**add** **esp,** 4**\***2 ; clean-up the stack

**mov** **[**file\_descriptor1**],** **eax** ; store the file descriptor returned by fopen

; check if fopen() has successfully created the file (EAX != 0)

**cmp** **eax,** 0

**je** final

; read the text from file using fread()

; after the fread() call, EAX will contain the number of chars we've read

; eax = fread(text, 1, len, file\_descriptor)

**push** **dword** **[**file\_descriptor1**]**

**push** **dword** len

**push** **dword** 1

**push** **dword** text

**call** **[**fread**]**

**add** **esp,** 4**\***4

; display the number of chars we've read and the text

; printf(format, eax, text)

**push** **dword** text

**push** **dword** **EAX**

**push** **dword** format

**call** **[**printf**]**

**add** **esp,** 4**\***3

; print into second file, in savedinlongtext.txt

;before printing, we create the file: accesmode w

; fopen() will return a file descriptor in the EAX or 0 in case of error

; eax = fopen(file\_name, access\_mode)

**push** **dword** access\_mode2

**push** **dword** file\_name2

**call** **[**fopen**]**

**add** **esp,** 4**\***2 ; clean-up the stack

**mov** **[**file\_descriptor2**],** **eax** ; store the file descriptor returned by fopen

;now printing:

; display the number of chars we've read and the text

; fprintf(descriptor,format, eax, text)

**push** **dword** text

**push** **dword** **EAX**

**push** **dword** format

**push** **dword** **[**file\_descriptor2**]**

**call** **[**fprintf**]**

**add** **esp,** 4**\***4

; call fclose() to close the file

; fclose(file\_descriptor)

**push** **dword** **[**file\_descriptor1**]**

**call** **[**fclose**]**

**add** **esp,** 4

; call fclose() to close the file

; fclose(file\_descriptor)

**push** **dword** **[**file\_descriptor2**]**

**call** **[**fclose**]**

**add** **esp,** 4

final**:**

; exit(0)

**push** **dword** 0

**call** **[**exit**]**

Text files in assembly are sequences of byes, meaning are sequences of characters. The fread function read sequences of bytes, so sequences of characters.

**If we want to read numbers from files**, the function fscanf is used.

**Syntax of fscanf** in a high level programming language:

int fscanf (FILE \* stream=file descriptor, const char \* format, <address\_variable\_1>, <address\_variable\_2>, <...>)

### *Arguments of the fscanf function:*

First argument -stream- of the function represents the file descriptor (identifier) returned by the fopen function call.

The next argument -format- of the function is a character string containing the format for reading, followed by the same number of arguments (variables) as specified in the format.

Similar to the scanf function, the character string representing the format can contain certain formatting markups, starting with the character ’%’, which will be replaced by the values given in the arguments.

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Type** | **Example** | **Value representation dimension** |
| c | Character | a | byte |
| d or i | Signed decimal integer | 392 | dword |
| u | Unsigned decimal integer | 7235 | dword |
| x | hexadecimal integer | 7fa | dword |
| s | String (terminated with a 0) | example | string of bytes terminated with 0 |

**Ex 5: Read a number from a file: ‘nr.txt’**

**Add constant 10 at this number and then print the result in file nr.txt, on newline.**

bits 32

global start

; declare external functions needed by our program

extern exit**,** fopen**,** fprintf**,** fclose**,** fscanf

import exit msvcrt.dll

import fopen msvcrt.dll

import fprintf msvcrt.dll

import fclose msvcrt.dll

import fscanf msvcrt.dll

; our data is declared here (the variables needed by our program)

segment data use32 class**=**data

file\_name db "nr.txt"**,** 0 ; filename to be created

access\_mode db "a+"**,** 0 ; file access mode:

; w - creates an empty file for writing

file\_descriptor dd **-**1 ; variable to hold the file descriptor

format\_read db '%d'**,** 0

a resd 1

newline db 10**,** 0

rez resd 1

; our code starts here

segment code use32 class**=**code

start**:**

; call fopen() to create the file

; fopen() will return a file descriptor in the EAX or 0 in case of error

; eax = fopen(file\_name, access\_mode)

**push** **dword** access\_mode

**push** **dword** file\_name

**call** **[**fopen**]**

**add** **esp,** 4**\***2 ; clean-up the stack

**mov** **[**file\_descriptor**],** **eax** ; store the file descriptor returned by fopen

; check if fopen() has successfully created the file (EAX != 0)

**cmp** **eax,** 0

**je** final

**push** **dword** a

**push** **dword** format\_read ; definit in ds

**push** **dword** **[**file\_descriptor**]** ; identificatorul de fisier din care citim

**call** **[**fscanf**]**

**add** **esp,** 4**\***3

; write the text to file using fprintf()

; fprintf(file\_descriptor, text)

**mov** **eax,** **[**a**]**

**add** **eax,** 10

**mov** **[**rez**],** **eax**

;print on new line the value from file add value 10

;first print new line

**push** **dword** newline

**push** **dword** **[**file\_descriptor**]**

**call** **[**fprintf**]**

**add** **esp,** 4**\***2

;second, print the result

**push** **dword** **[**rez**]**

**push** **dword** format\_read

**push** **dword** **[**file\_descriptor**]**

**call** **[**fprintf**]**

**add** **esp,** 4**\***3

; call fclose() to close the file

; fclose(file\_descriptor)

**push** **dword** **[**file\_descriptor**]**

**call** **[**fclose**]**

**add** **esp,** 4

final**:**

; exit(0)

**push** **dword** 0

**call** **[**exit**]**

ex6. Problem: Read a string from a file ’input.txt’.

Extract in a string s all small letters and print this string on the screen

Extract in a string c all capital letters and print this string c in file ’out.txt’

bits 32

global start

; declare external functions needed by our program

extern exit**,** fopen**,** fread**,** fclose**,** printf**,** fprintf

import exit msvcrt.dll

import fopen msvcrt.dll

import fread msvcrt.dll

import fclose msvcrt.dll

import printf msvcrt.dll

import fprintf msvcrt.dll

; our data is declared here (the variables needed by our program)

segment data use32 class**=**data

file\_name1 db "input.txt"**,** 0 ; filename to be read

access\_mode1 db "r"**,** 0 ; file access mode:

; r - opens a file for reading. The file must exist.

file\_descriptor1 dd **-**1 ; variable to hold the file descriptor

len equ 300 ; maximum number of characters to read

text times **(**len**+**1**)** db 0 ; string to hold the text which is read from file

format db 'We have read %d chars from file'**,** 10 **,** ' The text is: %s'**,** 0

file\_name2 db "output.txt"**,** 0 ; filename to be read

access\_mode2 db "w"**,** 0 ; file access mode:

; r - opens a file for reading. The file must exist.

file\_descriptor2 dd **-**1 ; variable to hold the file descriptor

nr\_caractere resd 1

s resb 300

**c** resb 300

formatrez db '%s'**,** 0

; our code starts here

segment code use32 class**=**code

start**:**

; call fopen() to create the file

; fopen() will return a file descriptor in the EAX or 0 in case of error

; eax = fopen(file\_name, access\_mode)

**push** **dword** access\_mode1

**push** **dword** file\_name1

**call** **[**fopen**]**

**add** **esp,** 4**\***2 ; clean-up the stack

**mov** **[**file\_descriptor1**],** **eax** ; store the file descriptor returned by fopen

; check if fopen() has successfully created the file (EAX != 0)

**cmp** **eax,** 0

**je** final

; read the text from file using fread()

; after the fread() call, EAX will contain the number of chars we've read

; eax = fread(text, 1, len, file\_descriptor)

**push** **dword** **[**file\_descriptor1**]**

**push** **dword** len

**push** **dword** 1

**push** **dword** text

**call** **[**fread**]**

**add** **esp,** 4**\***4

**mov** **[**nr\_caractere**],** **eax**

; now text contains the string from our input file

**mov** **esi,** 0; text

**mov** **edi,** 0 ; s - small letters

**mov** **ebp,** 0 ; c - capital letters

; we first create the string of capital letters

**mov** **ecx,** **[**nr\_caractere**]**

repeta**:**

**mov** **al,** **[**text**+esi]**

**cmp** **al,** 'A'

**JAE** verificaZM

**JB** mergimaideparte

verificaZM**:**

**cmp** **al,** 'Z'

**JBE** adaugainc

**JA** mergimaideparte

adaugainc**:**

**mov** **[c+ebp],** **al**

**inc** **ebp**

**inc** **esi**

**jmp** end\_repeta

mergimaideparte**:**

**inc** **esi**

end\_repeta**:**

**loop** repeta

;second, we create the string of small letters

**mov** **ecx,** **[**nr\_caractere**]**

**mov** **esi,** 0

**mov** **edi,** 0

repeta2**:**

**mov** **al,** **[**text**+esi]**

**cmp** **al,** 'a'

**JAE** verificaz

**JB** mergimaideparte2

verificaz**:**

**cmp** **al,** 'z'

**JBE** adaugains

**JA** mergimaideparte2

adaugains**:**

**mov** **[**s**+edi],** **al**

**inc** **edi**

**inc** **esi**

**jmp** end\_repeta2

mergimaideparte2**:**

**inc** **esi**

end\_repeta2**:**

**loop** repeta2

;print string with small letters in the screen

**push** **dword** s

**push** **dword** formatrez

**call** **[**printf**]**

**add** **esp,** 4**\***2

; print into second file the string of capital letters

;before printing, we create the file: accesmode w

; fopen() will return a file descriptor in the EAX or 0 in case of error

; eax = fopen(file\_name, access\_mode)

**push** **dword** access\_mode2

**push** **dword** file\_name2

**call** **[**fopen**]**

**add** **esp,** 4**\***2 ; clean-up the stack

**mov** **[**file\_descriptor2**],** **eax** ; store the file descriptor returned by fopen

;now printing:

; display the number of chars we've read and the text

; printf(format, eax, text)

**push** **dword** **c**

**push** **dword** formatrez

**push** **dword** **[**file\_descriptor2**]**

**call** **[**fprintf**]**

**add** **esp,** 4**\***3

; call fclose() to close the file

; fclose(file\_descriptor)

**push** **dword** **[**file\_descriptor1**]**

**call** **[**fclose**]**

**add** **esp,** 4

; call fclose() to close the file

; fclose(file\_descriptor)

**push** **dword** **[**file\_descriptor2**]**

**call** **[**fclose**]**

**add** **esp,** 4

final**:**

; exit(0)

**push** **dword** 0

**call** **[**exit**]**

Ex 7

;A text consisting letters, digits and special symbols are given in data segment.

;Compute the sum of all digits from text and print this number in a file (the file name is given also in data segment)

bits 32

global start

import fopen msvcrt.dll

import fprintf msvcrt.dll

import fclose msvcrt.dll

import exit msvcrt.dll

extern fopen**,** fprintf**,** fclose**,** exit

segment data use32 class**=**data

file\_name db 'file.txt'**,** 0

text db 'ab12cd#@%3ef4g5h\_6+7!i8$j9^g10^h11\*i12`~13='**,** 0 ; 1+2+3+4+5+6+7+8+9+1+0+1+1+1+2+1+3 = 55

text\_len equ **$-**text

write\_mode db 'w'**,** 0

write\_text db 'Sum of all ditis from text is = %d.'**,** 0

file\_descriptor resd 1

sum resd 0

segment code use32 class**=**code

start**:**

**cld** ; DF = 0, parcurgem de la stanga la dreapta

**mov** **ecx,** text\_len

**mov** **esi,** text

whole\_text**:**

**lodsb** ; al <- cate un caracter

**cmp** **al,** '0'

**jl** next

**cmp** **al,** '9'

**jg** next

**sub** **al,** '0' ; char to int

**add** **[**sum**],** **al**

next**:**

**loop** whole\_text

; fopen("file.txt", "w")

**push** **dword** write\_mode

**push** **dword** file\_name

**call** **[**fopen**]**

**add** **esp,** 4**\***2

**cmp** **eax,** 0

**je** final

**mov** **[**file\_descriptor**],** **eax**

; fprintf(file, "Sum of all ditis from text is = %d.", sum)

**push** **dword** **[**sum**]**

**push** **dword** write\_text

**push** **dword** **[**file\_descriptor**]**

**call** **[**fprintf**]**

**add** **esp,** 4**\***3

; fclose(file)

**push** **dword** **[**file\_descriptor**]**

**call** **[**fclose**]**

**add** **esp,** 4**\***1

final**:**

; exit(0)

**push** **dword** 0

**call** **[**exit**]**

Ex\_8:

;A text is given in data segment. Text contains all types of special symbols, small ;and capital letters and digits. Replace all special character with character X.

;Create a file with a given name in the data segment and print the achieved text ;in file.

bits 32 ; assembling for the 32 bits architecture

; declare the EntryPoint (a label defining the very first instruction of the program)

global start

; declare external functions needed by our program

extern exit**,** fopen**,** fclose**,** fprintf

import exit msvcrt.dll

import fopen msvcrt.dll

import fclose msvcrt.dll

import fprintf msvcrt.dll

; our data is declared here (the variables needed by our program)

segment data use32 class**=**data

nume db "fisierscriere.txt"**,** 0

text db "\_Sir cu ?cifre, caractere speciale /si\_, si 099, \_LITERE-; si altele?=<>aSDasdfF!%$ CUVANT][\]litere}}{~aaa"

; "XSirXcuXXcifreXXcaractereXspecialeXXsiXXXsiX099XXXLITEREXXXsiXalteleXXXXaSDasdfFXXXXCUVANTXXXXlitereXXXXaaa"

text\_len EQU **$-**text

mod\_acces db "w"**,** 0

descriptor dd 0

format db "%c"**,** 0

; our code starts here

segment code use32 class**=**code

start**:**

**push** **dword** mod\_acces

**push** **dword** nume

**call** **[**fopen**]**

**add** **esp,** 4**\***2

; verificam daca functia fopen a creat cu succes fisierul (daca eax != 0)

**CMP** **eax,** 0

**JE** final

**mov** **[**descriptor**],** **eax**

**mov** **ecx,** text\_len ; ecx = lungimea textului (numarul de caractere din text)

**mov** **esi,** text

**CLD**

repeta**:**

**LODSB** ; incarcam urmatorul caracter din text in al

;CMP al, ' ' ; daca e spatiu sarim peste

;JE next

**CMP** **al,** '0' ; daca al < '0' at nu e nici cifra, nici litera mare sau mica deci inlocuim

**JB** inlocuieste

**CMP** **al,** '9' ; daca al >= '0' si al <= '9' at e cifra si sarim peste

**JBE** next

**CMP** **al,** 'A' ; daca al > '9' si al < 'A' at nu e nici cifra, nici litera mare sau mica deci inlocuim

**JB** inlocuieste

**CMP** **al,** 'Z' ; daca al >= 'A' si al <= 'Z' at e litera mare deci sarim peste

**JBE** next

**CMP** **al,** 'a' ; daca al > 'Z' si al < 'a' at nu e nici cifra, nici litera mare sau mica deci inlocuim

**JB** inlocuieste

**CMP** **al,** 'z' ; daca al >= 'a' si al <= 'z' at e litera mica deci sarim peste

**JBE** next

; daca e mai mare decat 'z' inlocuim oricum

inlocuieste**:**

**mov** **al,** 'X'

next**:**

**mov** **bl,** **al**

**mov** **eax,** 0

**mov** **al,** **bl**

**PUSHAD**

; scriem in fisier caracterul

**push** **dword** **eax**

**push** **dword** format

**push** **dword** **[**descriptor**]**

**call** **[**fprintf**]**

**add** **esp,** 4**\***3 ; eliberam stiva

**POPAD**

**LOOP** repeta

; inchidem fisierul

**push** **dword** **[**descriptor**]**

**call** **[**fclose**]**

**add** **esp,** 4 ; eliberam stiva

final**:**

; exit(0)

**push** **dword** 0 ; push the parameter for exit onto the stack

**call** **[**exit**]** ; call exit to terminate the program

Extra bonus points requirements from Lecture 10:

Working with files is not a requirement for laboratory.

However, for an extra bonus point, you can try to solve one of the next problems:

Prob1:

A file name try.txt is given in data segment.

The file try exists and contains 10 numbers in hexadecimal.

Extract the maximum number and print this maximum into another file try2. The file try2 doesn’t exist (it is necessary to be created first).

Prob2:

A string of doublewords is given in data segment.

Extract only the doublewords with an odd numbers of bits equal to 1 and print this string in file try3. The file try3 doesn’t exist (it is necessary to be created first).

If you are solving 1 and/or 2, just send the asm file to me on Teams chat before 22.01.2023