```
; Breyona Gurosko
; CMPE 310 Spring 2025
; proj2.asm
; This program links with C standard library by default to use C input/output
;   functions in order to open a data file, read the contents of the file
;   which are one integer per line (with the number of total integers in the
;   file on the first line) to output the sum of all integers in the file and
;   the total number of integers in the file.
;   If opening and reading the file was unsuccessful, the output will be the
;   corresponding error statements.
;
; Assemble: nasm -f elf32 -l proj2.lst proj2.asm
; Link: gcc -m32 proj2 proj2.o
;
; C standard library functions linked in by default by gcc linker:
    extern printf
    extern fscanf
    extern fopen
    extern fclose

    section .data


    section .bss

FilePtr:    resb 4     ; File pointer
RunTotal:   resb 4     ; Running total of integers read from file
NumValues:  resb 4     ; Integer variable (2 bytes only needed - max 1000)
                       ;   to store number of values in array
IntValues:  resb 4000  ; Reserve 4000 bytes -- enough for
                       ;    1000 integer (4 byte) values
                       ; Array not needed but directions explicitly state to
                       ;   read into an array

    section .text

FileModeStr:   db "r",0            ; String for fopen() file mode (read only)
ScanFmtStr:    db "%d",0           ; Format string for fscanf()
SumFmtStr:     db "Integers read = %d. Sum of integers read = %d",10,0
    ; Sum output string for printf
FileErrStr:    db "Unable to open file!",10,0
    ; File opening and file reading error outputs
TooManyStr:    db "Too many values in file!  Max = 1000",10,0
NoFileNameStr: db "No file name given!",10,0
ArgErrStr:     db "No file name given!",10,0
ReadErrStr:    db "Error reading file!",10,0
TooManyErrStr: db "Too many values in file!",10,0
TooFewErrStr:  db "Warning! Unable to read all values!",10, 0

    global main
```

```
; C equivalent for main -- int main(int argc, int *argv[])
; argc contains the count of arguments that were on the command line,
;   including the command itself
; argv is a pointer to an array of string pointers, one for each argument.
; For this program:
; argv[0] is "prog2"
; argv[1] is the string name of the integer data file

; On entry, the stack will have arguments pushed to it and
;   the esp (stack pointer) register will be set to the
;   return address for main.
; The offsets are:
; argv                      [esp + 8]   - array point
; argc                      [esp + 4]   - number of arguments
; function return address   [esp]       - return address upon
;                                           completion of main()

main:
    mov     eax, [esp + 4]       ; Get argc -- number of command line arguments
    cmp     eax, 2               ; Need at least 2 arguments
                                 ;   (command name and file name)
    jge     GetArgV              ; Jump to next step if greater than
                                 ;   or equal to 2
    mov     eax, ArgErrStr
    call    PrintErr             ; Print error
    jmp     ProgEnd
GetArgV:
    mov     esi, [esp + 8]       ; Get argv pointer (array of string pointers)
    mov     eax, [esi + 4]       ; Get argv[1] -- each pointer is 4 bytes, so
                                 ;   index 1 is 4 bytes into array
    call    OpenFile
    cmp     eax, 0               ; If Open file was NULL, then failure
    jne     ReadNumberCount
    mov     eax, FileErrStr
    call    PrintErr             ; Print file error string
    jmp     ProgEnd
ReadNumberCount:
    mov     [FilePtr], eax       ; Store file pointer
                                 ; Eax still contains file pointer
    mov     ebx, NumValues       ; Address of NumValues variable
    call    ReadValue
    cmp     eax, -1              ; Check for end of file
    jne     CheckValues
    mov     eax, ReadErrStr
    call    PrintErr             ; Print read error string
    jmp     CloseFile
CheckValues:
    cmp     dword [NumValues], 1000  ; Compare number of file values to 1000
    jle     ReadLoopSetup
    mov     eax, TooManyErrStr
    call    PrintErr             ; Print error for too many values
```

```
    jmp     CloseFile
ReadLoopSetup:
    mov     ecx, 0                  ; Ecx will be our array index
                                    ; Clear all 32 bits (ecx) even though only
                                    ;  lower 16 bits of cx
ReadLoop:
    mov     eax, [FilePtr]
    mov     ebx, ecx                ; Get current index
    shl     ebx, 2                  ; Multiply by 4 - stame as left shift by 2
                                    ;  (each integer is 4 bytes)
    add     ebx, IntValues          ; Address is IntValues address plus 4 * index
    call    ReadValue
    cmp     eax, -1
    je      PrintFinal                ; If -1 assume finished
Add2Sum:
    mov     eax, dword [ebx]
    add     dword [RunTotal], eax   ; Ebx still contains address of this element
Index:
    add     ecx, 1                    ; Add one to index
    cmp     ecx, [NumValues]          ; Compare with number of values to be read
    jl      ReadLoop                  ; Jump back to read loop if less than
CheckRead:
    cmp     ecx, [NumValues]          ; Check to see how many values were
     actually read
    je      PrintFinal                ; If equal, then finished successfully
    push    dword TooFewErrStr        ; Unable to read all values
    call    printf                    ; Print notification
    add     esp, 4
PrintFinal:
    push    dword [RunTotal]         ; Push sum argument onto stack
    push    ecx                      ; Push total number of values read onto stack
                                     ;   (printf expects 32-bit integer even
                                      though
                                     ;        only used 16-bit cx)
    push    SumFmtStr                ; Push format string onto stack
    call    printf                   ; Print final sum
    add     esp, 12                  ; Fixing up stack pointer
CloseFile:
    push    dword [FilePtr]
    call    fclose                   ; Closing file opened
    add     esp, 4                   ; Fixing up stack pointer
ProgEnd:
    ret             ; main() is a function called so code must be returned from
                    ;    when finished. The startup code that called main will
                    ;    clean up and exit.

; OpenFile -- open the file.
; eax must contain the pointer to the file name on entry
; eax will contain the file pointer upon exit returning
OpenFile:
    push dword FileModeStr     ; Push file mode ("r") string pointer on stack
```

```
    push eax                    ; Push file name string pointer on stack
    call fopen                  ; Call C-library fopen() function --
                                ;   eax will contain result NULL (0) for
                                ;   failure, pointer to FILE if success
    add  esp,8                  ; Fix up stack from pushed arguments
    ret

; ReadValue -- read a single integer value from file
; On entry -- eax = file pointer, ebx = address to store value read
; Returns 0 in eax if successful, -1 in eax if end of file
ReadValue:
    push ecx                    ; Save ECX register
    push ebx                    ; Also save EBX register
    push ebx                    ; Push address to store result on stack
    push dword ScanFmtStr       ; Push address of fscanf() format string
    push eax                    ; Push file pointer
    call fscanf                 ; Call C-library fscanf( FILE,Fmt,&storage)
                                ; Result of fscanf() will be in eax (0 or -1)
    add  esp,12                 ; Fix up stack after fscanf() argument push
    pop  ebx                    ; Restore EBX register
    pop  ecx                    ; Restore ECX register

    ret

; PrintError
; Input is pointer to error string in eax
PrintErr:
    push ecx                    ; Save ecx register so it doesn't get altered
    push eax                    ; Push string pointer on stack for printf()
    call printf                 ; Call printf
    add  esp,4                  ; Fix stack from eax push
    pop  ecx                    ; Restore ecx register
    ret
```

TestNumber.txt: test data file to be read in to project program to confirm functionality, other than the provided text file.

TestNumber.txt

```
10
1
2
3
4
5
6
7
8
9
10
```

Project 2 outputs with TestNumber.txt and randomInt100.txt.

breyonagurosko — ek19139@linux6:project2 — ssh ek19139@gl.umbc.ed...

```
Passcode or option (1-1): 1172258
Success. Logging you in...
Success. Logging you in...

UMBC Division of Information Technology                    http://doit.umbc.edu/
-------------------------------------------------------------------------------
If you have any questions or problems regarding these systems, please call the
DoIT Technology Support Center at 410-455-3838, or submit your request on the
web by visiting http://my.umbc.edu/help/request

Remember that the Division of Information Technology will never ask for your
password. Do NOT give out this information under any circumstances.
-------------------------------------------------------------------------------

Last login: Sun Apr  6 20:50:06 2025 from 69.143.41.75
[[ek19139@linux6 ~]$ cd CMPE310
[[ek19139@linux6 ~/CMPE310]$ cd project2
[[ek19139@linux6 project2]$ ls
proj2  proj2.asm  proj2.lst  proj2.o  randomInt100.txt  TestNumber.txt
[[ek19139@linux6 project2]$ ./proj2 randomInt100.txt
Integers read = 100. Sum of integers read = 4579
[[ek19139@linux6 project2]$ ./proj2 TestNumber.txt
Integers read = 10. Sum of integers read = 55
[ek19139@linux6 project2]$
```