**DISPLAY 2D ARRAY**

INCLUDE Irvine32.inc

.data

comma\_seperator BYTE ", ",0

my\_array WORD 15,20,30,

50,74,82,

60,84,84,

10,54,94

rows\_num DWORD 4

cols\_num DWORD 3

.code

;-------------------------------------------------------------------------------

Display\_Array PROC

mov edx, offset comma\_seperator

mov ecx, rows\_num

mov edi, 0

row\_loop:

push ecx

mov ecx, cols\_num

mov esi, 0

col\_display\_loop:

movzx eax, my\_array [esi+ edi\* TYPE my\_array] ; arr[i][j] \*2 for WORD type

call writeDec

call writeString

add esi,TYPE my\_array

loop col\_display\_loop

call crlf

add edi, cols\_num

pop ecx

loop row\_loop

ret

Display\_Array endp

;-------------------------------------------------------------------------------

main PROC

call Display\_Array

exit

main endp

end main

**DECLARE AND DISPLAY A SINGLE STRUCT ELEMENT**

INCLUDE Irvine32.inc

.data

slash BYTE ' / ',0

Array\_element STRUCT

Numerator DWORD ?

Denominator DWORD ?

Array\_element ENDS

element1 Array\_element <10, 20>

.code

main PROC

mov edx, offset slash

lea ebx, element1 ;loading address of point1 in

mov eax, [ebx]

call WriteDec

call writeString

mov eax, [ebx+4]

call WriteDec

exit

main endp

end main

**DISPLAY A 2D STRUCT**

INCLUDE Irvine32.inc

.data

slash BYTE "/",0

Comma\_seperator BYTE " , ",0

Array\_element STRUCT

Numerator DWORD ?

Denominator DWORD ?

Array\_element ENDS

elements Array\_element <1, 2>, <3, 4>, <5, 6>, <7, 8>, <9, 10>,

<11, 12>, <13, 14>, <15, 16>, <17, 18>, <19, 20>,

<21, 22>, <23, 24>, <25, 26>, <27, 28>, <29, 30>

col\_length DWORD 5

row\_length DWORD 3

.code

;-------------------------------------------------------------------------------

Display\_2D\_Struct PROC

mov ecx, row\_length

lea ebx, elements

mov edi, 0 ;for rows index

row\_display\_loop:

push ecx

mov ecx, col\_length

mov esi, 0 ;for columns index

add esi,edi ; to access next row of elements

col\_display\_loop:

mov eax, [ebx + esi]

call writeDec

mov edx, offset slash

call writeString

mov eax, [ebx+ 4 +esi]

call WriteDec

mov edx, offset Comma\_seperator

call writeString

add esi, sizeof Array\_element

loop col\_display\_loop

call crlf

mov eax, sizeof Array\_element

mul col\_length

add edi, eax

pop ecx

loop row\_display\_loop

ret

Display\_2D\_Struct endp

;-------------------------------------------------------------------------------

main PROC

call Display\_2D\_Struct

exit

main endp

end main

**FLIP A FRACTION GIVEN ITS ADDRESS IN THE STRUCTURE**

INCLUDE Irvine32.inc

.data

slash BYTE "/",0

Comma\_seperator BYTE " , ",0

Fraction\_to\_flip\_index DWORD 0 ;if flipping index 2, it is 2\* SIZEOF Array\_element

Array\_element STRUCT

Numerator DWORD ?

Denominator DWORD ?

Array\_element ENDS

elements Array\_element <1, 2>, <3, 4>, <5, 6>, <7, 8>, <9, 10>

col\_length DWORD 5

.code

;-------------------------------------------------------------------------------

Display\_Struct PROC

mov ecx, col\_length

lea ebx, elements

mov esi, 0

col\_display\_loop:

mov eax, [ebx + esi]

call writeDec

mov edx, offset slash

call writeString

mov eax, [ebx+ 4 +esi]

call WriteDec

mov edx, offset Comma\_seperator

call writeString

add esi, sizeof Array\_element

loop col\_display\_loop

ret

Display\_Struct endp

;-------------------------------------------------------------------------------

;-------------------------------------------------------------------------------

Flip\_Fraction PROC USES esi eax ebx esi

lea ebx, elements

mov esi,Fraction\_to\_flip\_index

mov eax, [ebx + esi]

xchg [ebx+esi+4], eax

xchg eax, [ebx + esi]

ret

Flip\_Fraction endp

;-------------------------------------------------------------------------------

main PROC

call Display\_Struct

call crlf

mov Fraction\_to\_flip\_index,2\* SIZEOF Array\_element

call Flip\_Fraction

call Display\_Struct

exit

main endp

end main

**DIVIDE 2 PARTICULAR ELEMENTS OF THE ARRAY**

INCLUDE Irvine32.inc

.data

slash BYTE "/",0

Comma\_seperator BYTE " , ",0

Array\_element STRUCT

Numerator SDWORD ?

Denominator SDWORD ?

Array\_element ENDS

Temp\_Operand\_Arr1 Array\_element <0,0> ;copies operands and manupulates them for calculations

Temp\_Operand\_Arr2 Array\_element <0,0>

Singular\_Resultant Array\_element <0,0>

First\_Operand\_Address DWORD 0

Second\_Operand\_Address DWORD 0

Array1 Array\_element <1, 2>, <3, 4>, <-5, 6>, <7, 8>, <9, 10>

Array2 Array\_element <5, 9>, <7, 3>, <2, 8>, <4, 6>, <10, 1>

col\_length DWORD 5

Sign\_flag BYTE 0

Smaller\_value DWORD ? ;used in simplification function

temporary\_Simplified\_numerator DWORD ? ;used in simplification function

.code

;Whenever you call this, have address of arrayu to display in ebx, along with col length in col\_length

;-------------------------------------------------------------------------------

Display\_Struct PROC

mov ecx, col\_length

mov esi, 0

col\_display\_loop:

mov eax, [ebx + esi]

call writeInt

mov edx, offset slash

call writeString

mov eax, [ebx+ 4 +esi]

call WriteInt

mov edx, offset Comma\_seperator

call writeString

add esi, sizeof Array\_element

loop col\_display\_loop

ret

Display\_Struct endp

;-------------------------------------------------------------------------------

;Utility function to copy 2 operands into a temporary variable to perform calculation

;-------------------------------------------------------------------------------

Set\_operands\_for\_calculations PROC USES ecx ebx eax

mov ecx,First\_Operand\_Address

mov eax, [ecx]

lea ebx, Temp\_Operand\_Arr1

xchg eax, [ebx]

mov eax, [ecx+4]

lea ebx, Temp\_Operand\_Arr1

xchg eax, [ebx+4]

mov ecx,Second\_Operand\_Address

mov eax, [ecx]

lea ebx, Temp\_Operand\_Arr2

xchg eax, [ebx]

mov eax, [ecx+4]

lea ebx, Temp\_Operand\_Arr2

xchg eax, [ebx+4]

ret

Set\_operands\_for\_calculations endp

;-------------------------------------------------------------------------------

;whenever you call this, have the two operands address in the Operand\_Address variables

;-------------------------------------------------------------------------------

Divide\_given\_2\_Elements PROC USES eax ebx ecx

mov Sign\_flag,0

call Set\_operands\_for\_calculations

lea ebx, Temp\_Operand\_Arr2

call Flip\_Fraction ;convert (a/b)/(c/d) to a/b \* d/c

lea ebx, Temp\_Operand\_Arr1

lea eax, Temp\_Operand\_Arr2

mov ecx, [eax]

mov eax, [ebx]

imul ecx

lea edx, Singular\_Resultant ;move result of multiplication into Singular\_Resultant variable

xchg [edx], eax

lea eax, Temp\_Operand\_Arr2+4

mov ecx, [eax] ;now multiply denominators

mov eax, [ebx+4]

imul ecx

lea edx, Singular\_Resultant ;move result of multiplication into Singular\_Resultant variable

xchg [edx+4], eax

lea ebx,Singular\_Resultant[0] ;here, if resultant is signed, remove sign for simplification

mov eax,[ebx]

cmp eax,0 ; handling case -a/b or a/-b or a\b

jge Not\_signed\_NUM

;#################################### ;case -a/b

mov Sign\_flag,1

mov ecx, -1

imul ecx

xchg [ebx], eax

jmp Not\_signed\_DENOM ; answer can never be -a/-b so dont check denom

;####################################

;####################################

Not\_signed\_NUM:

mov eax, [ebx+4]

cmp eax,0

jge Not\_signed\_DENOM

;case -a/b

mov Sign\_flag,1

imul ecx

lea ebx,Singular\_Resultant[0]

xchg [ebx+4], eax

;####################################

Not\_signed\_DENOM: ;case a/b

call Simplify\_Fraction

movzx ecx, Sign\_flag ;if result was signed before simplification, restore sign.

cmp ecx,1

jne Was\_not\_signed\_before

mov ecx,-1

lea ebx,Singular\_Resultant[0]

mov eax,[ebx]

imul ecx

xchg [ebx], eax

Was\_not\_signed\_before:

ret

Divide\_given\_2\_Elements endp

;-------------------------------------------------------------------------------

;whenever you call this, have the address of element to flip in ebx

;-------------------------------------------------------------------------------

Flip\_Fraction PROC USES eax

mov eax, [ebx]

xchg [ebx+4], eax

xchg eax, [ebx]

ret

Flip\_Fraction endp

;-------------------------------------------------------------------------------

;-------------------------------------------------------------------------------

Get\_smaller\_value PROC USES eax

lea ebx, Singular\_Resultant

mov eax, [ebx]

cmp eax, [ebx+4] ;equal case to be handled

je Set\_to\_1

jg Denom\_is\_smaller

jmp skip\_rest\_of\_cases ; eax already has numerator

Denom\_is\_smaller:

mov eax, [ebx+4]

jmp skip\_rest\_of\_cases

Set\_to\_1:

mov eax, 1

xchg [ebx], eax

mov eax,1

xchg [ebx+4], eax

ret

skip\_rest\_of\_cases:

mov Smaller\_value, eax

ret

Get\_smaller\_value endp

;-------------------------------------------------------------------------------

;before calling this, make sure the fraction to simplify is in 'Singular\_Resultant' variable

;-------------------------------------------------------------------------------

Simplify\_Fraction PROC USES ebx edi eax edx

lea ebx, Singular\_Resultant

mov edi,2

Simplification\_comparision\_loop:

call Get\_smaller\_value

mov edx,0

cmp edi, Smaller\_value

jg end\_simplification

mov eax, [ebx]

idiv edi

cmp edx,0

jne Cant\_divide\_here

mov temporary\_Simplified\_numerator,eax ;save numerator

mov edx,0

mov eax, [ebx+4]

idiv edi

cmp edx,0

jne Cant\_divide\_here

xchg [ebx+4],eax

mov eax, temporary\_Simplified\_numerator

xchg [ebx], eax

mov edi,1 ; reset edi if division happened

Cant\_divide\_here:

inc edi

jmp Simplification\_comparision\_loop

end\_simplification:

ret

Simplify\_Fraction endp

;-------------------------------------------------------------------------------

main PROC

lea ebx, Array1

call Display\_Struct

call crlf

add ebx, 16

mov First\_Operand\_Address, ebx ;4th index struct1 is divisor

lea ebx, Array2

call Display\_Struct

call crlf

add ebx, 16

mov Second\_Operand\_Address, ebx ;4th index struct1 is dividant

call Divide\_given\_2\_Elements

lea ebx, Singular\_Resultant

mov eax, [ebx]

call WriteInt

mov edx, offset slash

call WriteString

mov eax, [ebx+4]

call WriteInt

exit

main endp

end main

INPUT

INCLUDE irvine32.inc

.data

Comma\_seperator BYTE " , ",0

matrix\_input\_bracket BYTE "): ",0

spaces\_to\_print DWORD 10

Counter\_row BYTE 1

Counter\_col BYTE 1

entr\_size\_msg1 BYTE "Enter the rows in the Matrix 1: ",0

entr\_size\_msg2 BYTE "Enter the cols in the Matrix 1: ",0

entr\_size\_msg3 BYTE "Enter the rows in the Matrix 2: ",0

entr\_size\_msg4 BYTE "Enter the cols in the Matrix 2: ",0

display\_msg1 BYTE "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Matrix 1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_",10,10,0

display\_msg2 BYTE "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Matrix 2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_",10,10,0

display\_msg3 BYTE "\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Resultant \_\_\_\_\_\_\_\_\_\_\_\_\_\_",10,0

element\_counter\_heading1 BYTE " (For Matrix 1) ------> Enter Element (",0

element\_counter\_heading2 BYTE " (For Matrix 2) ------> Enter Element (",0

Matrix1 SDWORD 100 DUP (0)

Matrix2 SDWORD 100 DUP (0)

Answer\_matrix SDWORD 100 DUP (0)

col\_length1 DWORD 10

row\_length1 DWORD 10

col\_length2 DWORD 10

row\_length2 DWORD 10

.code

;----------------------------------------

Take\_input\_Dimentions PROC

mov eax,0

mov edx , offset entr\_size\_msg1

Call WriteString

call readDec

mov col\_length1,eax

mov edx , offset entr\_size\_msg2

Call WriteString

call readDec

mov row\_length1,eax

mov edx , offset entr\_size\_msg3

Call WriteString

call readDec

mov col\_length2,eax

mov edx , offset entr\_size\_msg4

Call WriteString

call readDec

mov row\_length2,eax

call crlf

ret

Take\_input\_Dimentions endp

;----------------------------------------

;----------------------------------------

print\_nice\_heading PROC heading\_ptr:DWORD

movzx eax, Counter\_row

mov edx, heading\_ptr

call WriteString

call writeDec

mov edx, offset Comma\_seperator

call WriteString

movzx eax, Counter\_col

call writeDec

mov edx, offset matrix\_input\_bracket

call WriteString

call ReadInt

call crlf

ret

print\_nice\_heading endp

;----------------------------------------

;----------------------------------------

Take\_elements\_matrix\_1 PROC

mov edi,0

mov esi,0

mov Counter\_row,1

mov Counter\_col,1

mov ecx, row\_length1

row\_input\_loop:

push ecx

mov ecx, col\_length1

col\_input\_loop:

invoke print\_nice\_heading, OFFSET element\_counter\_heading1

inc Counter\_col

mov Matrix1[esi\*4],eax

inc esi

loop col\_input\_loop

pop ecx

inc Counter\_row

mov counter\_col,1

mov esi,0

add edi, col\_length1

add esi, edi

loop row\_input\_loop

call crlf

ret

Take\_elements\_matrix\_1 endp

;----------------------------------------

;----------------------------------------

Take\_elements\_matrix\_2 PROC

mov edi,0

mov esi,0

mov Counter\_row,1

mov Counter\_col,1

mov ecx, row\_length2

row\_input\_loop:

push ecx

mov ecx, col\_length2

col\_input\_loop:

invoke print\_nice\_heading, OFFSET element\_counter\_heading2

inc Counter\_col

mov Matrix2[esi\*4],eax

inc esi

loop col\_input\_loop

pop ecx

inc Counter\_row

mov counter\_col,1

mov esi,0

add edi, col\_length2

add esi, edi

loop row\_input\_loop

call crlf

ret

Take\_elements\_matrix\_2 endp

;----------------------------------------

;counting the number of digits in a number for clean output display

;------------------------------------------

CountDigits PROC USES eax ecx ; ebx contains number to count letters in

mov spaces\_to\_print, 10

mov eax, ebx

mov ecx,1

mov esi,10

cmp eax,10

jl countFinal

countLoop:

cdq

idiv esi ; Divide by 10

cmp eax, 0 ; Check if quotient is zero

je countFinal ; If not zero, continue the loop

inc ecx ; Increment the digit counter

jmp countLoop

countFinal:

sub spaces\_to\_print,ecx

ret

CountDigits ENDP

;------------------------------------------

;----------------------------------------

printMatrix PROC USES eax ebx ecx edx esi,

matrix\_ptr:PTR SDWORD,

printRows:SDWORD,

printCols:SDWORD

LOCAL row\_index:SDWORD, col\_index:SDWORD

mov row\_index, 0

row\_loop:

mov eax, row\_index

cmp eax, printRows

jae done

mov col\_index, 0

col\_loop:

mov eax, col\_index

cmp eax, printCols

jae next\_row

; Calculate offset: row\_index \* cols + col\_index

mov eax, row\_index

imul eax, printCols

add eax, col\_index

shl eax, 2 ; Multiply by 4 for SDWORD

; Get matrix element and print it

mov esi, matrix\_ptr

mov eax, [esi + eax]

call WriteInt ; Use WriteInt for signed integers

push ecx

mov ebx,eax

mov eax, ' '

call CountDigits

mov ecx, spaces\_to\_print ;padding for nicer output

Padding\_loop:

call WriteChar

loop Padding\_loop

pop ecx

inc col\_index

jmp col\_loop

next\_row:

call Crlf

inc row\_index

jmp row\_loop

done:

ret

printMatrix ENDP

;----------------------------------------

;-------------------------------------------------------------------------------

main PROC

call Take\_input\_Dimentions

call Take\_elements\_matrix\_1

call Take\_elements\_matrix\_2

mov edx, offset display\_msg1

call WriteString

INVOKE printMatrix, OFFSET Matrix1, row\_length1, col\_length1

call crlf

mov edx, offset display\_msg2

call WriteString

INVOKE printMatrix, OFFSET Matrix2, row\_length2, col\_length2

exit

main endp

end main