**q1:**

noopt.asm:

; the string "A st" , including the null terminator, was previously

; stored in the variable $SG4294967291

; \_str$ is the offset of the beggining of str array (str array in the c program) from ebp

; copy the first 4 bytes of the variable $SG429496729

; which means it copied "A st" to ecx, without the null terminator  
mov ecx, DWORD PTR $SG4294967291

; copy the content of ecx to the beggining of str array

; which means copy "A st" to the beggining of str array

mov DWORD PTR \_str$[ebp], ecx

; copy the null terminator after "A st" to the lowest byte of edx

mov dl, BYTE PTR $SG4294967291+4

; copy the null terminator from dl to the fifth byte of str arry

; which means put a null terminator in str after "A st"

mov BYTE PTR \_str$[ebp+4], dl

opt.asm:

; Same as noopt.asm, except, it uses only eax without using ecx and edx registers.

ex1.s:

/NO\_APP

; notice that 1953701953 = 0x74732041

; in addition, in ASCII representation, 0x41 = 'A', 0x20 = ' ', 0x73 = 's', 0x74 = 't'

; that means that the constans 1953701953 represents the 4 chars - "A st"

lea eax, [ebp-208]

; calculates ebp-208 and store it in eax

mov DWORD PTR [eax], 1953701953

; copy "A st" (as previously explained) to the address stored in eax

; which means copy "A st" to the beggining of str array

mov BYTE PTR [eax+4], 0

; put null terminator at the fifth byte of str array => put null terminator after "A st"

**q2:**

noopt:

; put tv78[ebp] to point to the begging of the long string "A string..." and

; tv76[ebp] to point to the beggining of str array

mov DWORD PTR tv78[ebp], OFFSET $SG4294967289

lea edx, DWORD PTR \_str$[ebp]

mov DWORD PTR tv76[ebp], edx

$LL3@q2:

; compare the one char of the strings

mov eax, DWORD PTR tv76[ebp]

mov cl, BYTE PTR [eax]

mov BYTE PTR tv81[ebp], cl

mov edx, DWORD PTR tv78[ebp]

cmp cl, BYTE PTR [edx]

; different chars, put 1 or -1 according to carry flag

jne SHORT $LN4@q2

; see if we reached the null terminators (of both strings)

cmp BYTE PTR tv81[ebp], 0

; if so, return 0

je SHORT $LN5@q2

; otherwise, compare another char (same logic as the first char)

mov eax, DWORD PTR tv76[ebp]

mov cl, BYTE PTR [eax+1]

mov BYTE PTR tv88[ebp], cl

mov edx, DWORD PTR tv78[ebp]

cmp cl, BYTE PTR [edx+1]

jne SHORT $LN4@q2

add DWORD PTR tv76[ebp], 2

add DWORD PTR tv78[ebp], 2

cmp BYTE PTR tv88[ebp], 0

; if didn't reach null terminator, loop again

jne SHORT $LL3@q2

$LN5@q2:

; put 0 in the return value

mov DWORD PTR tv93[ebp], 0

jmp SHORT $LN6@q2

$LN4@q2:

; if the carry flag in on, the return value will be -1, otherwise 1

sbb eax, eax

or eax, 1

mov DWORD PTR tv93[ebp], eax

$LN6@q2:

; store the return value in the stack at offset \_rc$

mov ecx, DWORD PTR tv93[ebp]

mov DWORD PTR tv71[ebp], ecx

mov edx, DWORD PTR tv71[ebp]

mov DWORD PTR \_rc$[ebp], edx

opt:

; put the address of the variable ??\_C@... (which happens to store the "A string....")

; in eax

mov eax, OFFSET ??\_C@\_0DH@PFMIMKOJ@A?5string?5hjdhkjdhkdjhdkjhdkjhd?5d@

; put the address of str array in acx

lea ecx, DWORD PTR \_str$[ebp]

npad 6

$LL3@q2:

; compare the first byte of the strings

mov dl, BYTE PTR [ecx]

cmp dl, BYTE PTR [eax]

jne SHORT $LN4@q2

; see if we reached the null terminator, if so, return 0

test dl, dl

je SHORT $LN5@q2

; otherwise, compare the next char

mov dl, BYTE PTR [ecx+1]

cmp dl, BYTE PTR [eax+1]

jne SHORT $LN4@q2

; advance both pointers in 2 (becuase we already checked 2 chars)

add ecx, 2

add eax, 2

test dl, dl

jne SHORT $LL3@q2

$LN5@q2:

; put 0 in eax, which tells us the here calling convension uses eax

; to return a value (at least in this function)

xor eax, eax

jmp SHORT $LN6@q2

$LN4@q2:

; same logic as in noopt, put 1 or -1

sbb eax, eax

or eax, 1

$LN6@q2:

ex1.s:

; store the address of the hard coded string in the stack

mov DWORD PTR [esp+4], OFFSET FLAT:LC0

; store the address of str array in the stack

lea eax, [ebp-212]

mov DWORD PTR [esp], eax

; call strcmp

call \_strcmp

; move the return value of strcmp to eax

mov DWORD PTR [ebp-12], eax