**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

**q3:**

noopt:

; loop that runs 50 times, and copies 4 bytes on each iteration from src to dst.

; it is done by decremeting ecx by 1 each time and repeating the movsd command.

; this command also increases esi and edi by 4 each iteration

mov ecx, 50 ; 00000032H

lea esi, DWORD PTR \_src$[ebp]

lea edi, DWORD PTR \_str$[ebp]

rep movsd

opt:

; same as noopt.asm

ex1.s:

; same as noopt, only the compiler copies adresses to edi and esi via some

; other registers, and intialize ecx via eax

**q4:**

noopt.asm:

; intialize i to 0

mov DWORD PTR \_i$[ebp], 0

jmp SHORT $LN3@q4

$LN2@q4:

; increment i by 1

mov eax, DWORD PTR \_i$[ebp]

add eax, 1

mov DWORD PTR \_i$[ebp], eax

$LN3@q4:

; compare i to 100, if equal or greater, finish iterating

cmp DWORD PTR \_i$[ebp], 100 ; 00000064H

jge SHORT $LN1@q4

; otherwise, add i to rc

mov ecx, DWORD PTR \_rc$[ebp]

add ecx, DWORD PTR \_i$[ebp]

mov DWORD PTR \_rc$[ebp], ecx

jmp SHORT $LN2@q4

$LN1@q4:

Opt.asm:

; here we can see the compiler does some magic

; and calculates 0+1+...+99 in 25 iterations instead of 100

; set eax, edx, esi, edi to 0

xor eax, eax

xor edx, edx

xor esi, esi

xor edi, edi

npad 2

$LL3@q4:

; edi(i) = i + 2\*(i-1)\*i

; esi(i) = 2\*i + 2\*(i-1)\*i

; edx(i) = 3\*i + 2\*(i-1)\*i

; ecx(i) = 2\*(i-1)\*i

inc edi

add esi, 2

add edx, 3

add ecx, eax

add edi, eax

add esi, eax

add edx, eax

add eax, 4

cmp eax, 100 ; 00000064H

jl SHORT $LL3@q4

; ecx = edi(25)+esi(25)+edx(25)+ecx(25)

; ecx = 1225 + 1250 + 1275 + 1200 = 4950 = 0+1+2+...+99

lea eax, DWORD PTR [edx+esi]

add eax, edi

add ecx, eax

ex1.s:

; iterates 100 times, and each iteration, adds the current number

; to a variable on a stack

mov DWORD PTR [ebp-12], 0

jmp L5

L6:

mov eax, DWORD PTR [ebp-12]

add DWORD PTR [ebp-16], eax

inc DWORD PTR [ebp-12]

L5:

cmp DWORD PTR [ebp-12], 99

jle L6

**q5:**

noopt.asm:

; 80 : switch (i)

mov eax, DWORD PTR \_i$[ebp]

mov DWORD PTR tv65[ebp], eax

mov ecx, DWORD PTR tv65[ebp]

sub ecx, 48 ; 00000030H

mov DWORD PTR tv65[ebp], ecx

; case i='u'

cmp DWORD PTR tv65[ebp], 69 ; 00000045H

; if tv65[ebp] is bigger than 69, it means i value is bigger

; than 'u' ascii value => which means none of the options

; are met, and the switch goes to deafult case

ja SHORT $LN1@q5

mov edx, DWORD PTR tv65[ebp]

; $LN18@q5 is a kind map array

; index i in the LN18@q5 array, contains index for the $LN19@q5 array,

; which in its turn, contains the address of

; the instructions handling the case of ascii char represented by i+48

; for example - 'C' char is represented by 67, which means index 67-48=19 in

; $LN18@q5 contains 4, LN19@5 at index 4 contains $LN11@q5

; we can see that the instructions starting at $LN11@q5 are handling case 'C'.

; we find that in all the chars which doesn't match any case we find index 12

; and in $LN19@q5 at index 12 leads to the default case

movzx eax, BYTE PTR $LN18@q5[edx]

jmp DWORD PTR $LN19@q5[eax\*4]

; all the cases

$LN13@q5:

mov DWORD PTR \_rc$[ebp], 1

jmp SHORT $LN14@q5

$LN12@q5:

mov DWORD PTR \_rc$[ebp], 9

jmp SHORT $LN14@q5

$LN11@q5:

mov DWORD PTR \_rc$[ebp], 8

jmp SHORT $LN14@q5

$LN10@q5:

mov DWORD PTR \_rc$[ebp], 7

jmp SHORT $LN14@q5

$LN9@q5:

mov DWORD PTR \_rc$[ebp], 6

jmp SHORT $LN14@q5

$LN8@q5:

mov DWORD PTR \_rc$[ebp], 5

jmp SHORT $LN14@q5

$LN7@q5:

mov DWORD PTR \_rc$[ebp], 4

jmp SHORT $LN14@q5

$LN6@q5:

mov DWORD PTR \_rc$[ebp], 11 ; 0000000bH

jmp SHORT $LN14@q5

$LN5@q5:

mov DWORD PTR \_rc$[ebp], 9

jmp SHORT $LN14@q5

$LN4@q5:

mov DWORD PTR \_rc$[ebp], 9

jmp SHORT $LN14@q5

$LN3@q5:

mov DWORD PTR \_rc$[ebp], 9

jmp SHORT $LN14@q5

$LN2@q5:

mov DWORD PTR \_rc$[ebp], 2

jmp SHORT $LN14@q5

$LN1@q5:

mov DWORD PTR \_rc$[ebp], 30 ; 0000001eH

$LN14@q5:

opt.asm:

; same as noopt.asm, except here the compiler joins the cases where

; the code puts the same value in rc, and some more minor diffrences

; such as the index to the addresses table is saved actually as an index

; and multiplied by 4 the get an offset

ex1.s:

; here we can see similar pattern to noopt.asm, with decreasing 48 from i

; and mapping all the possbilities between '0'-48 to 'u'-48, where chars that are

; not part of the switch leads to the default case