

EECS 444 HW2(3)

Name: Mingan Huang

Network ID: mxh805

Q1: (Comments marked in red)

```
push ebp
mov ebp,esp
and esp,0FFFFFFF0h
sub esp,20h
call __main
mov dword ptr[esp+1Ch],3 // Initialize 3 at address esp+1Ch
mov dword ptr[esp+18h],5 // Initialize 5 at address esp+18h
mov dword ptr[esp+14h],0 // Initialize 0 at address esp+14h
mov eax,[esp+1Ch] // eax = 3
imul eax,[esp+18h] // eax = 3 * 5
mov edx,eax // edx = 15
mov eax,[esp+1Ch] // Move 3 to eax
mov ecx,eax // Move 3 to ecx
shr ecx,1Fh // Isolating sign bit
add eax,ecx // eax = eax + ecx
sar eax,1 // divide eax by 2
// (above four steps have the same functionalities as shr eax 1)
sub edx,eax // edx = edx - eax
mov eax,edx
mov [esp+14h],eax // Copy the value in edx to [esp+14h]
mov eax,[esp+14h]
mov [esp+4],eax
mov dword ptr[esp], offset aD; "%d"
call _printf // Use printf to print out the result
mov eax,0
leave
retn
__main endp
```

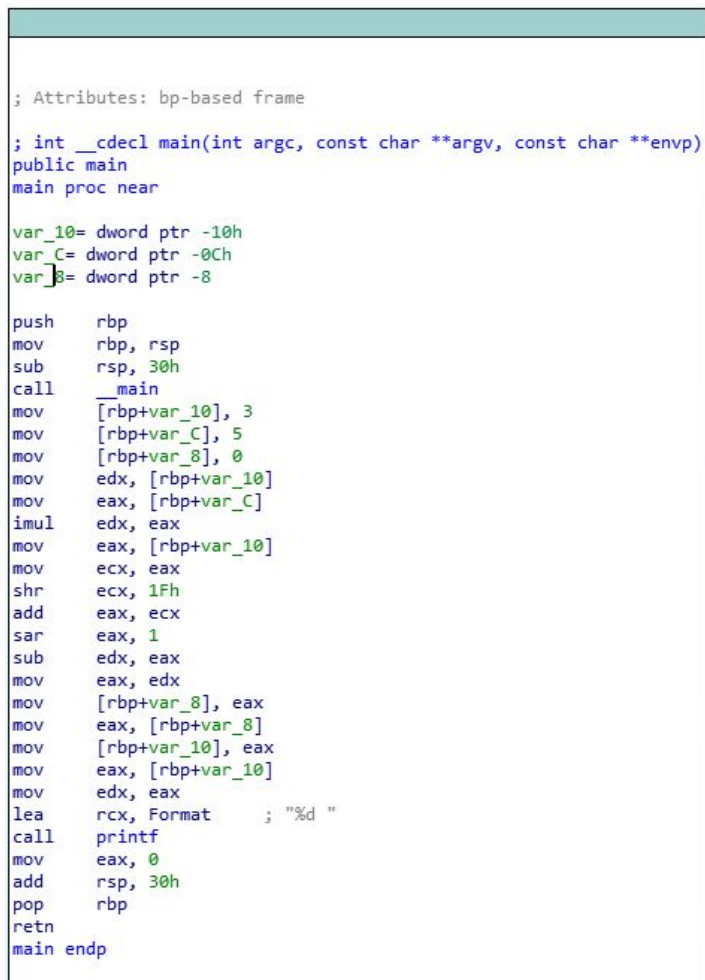
The functionality of this assembly code is to initialize three variables, calculate $(3*5 - 3/2)$ and store the result in the last variable. The result is 14.

My C code for Q1:

```
#include <stdio.h>
```

```
int main() {  
    int x = 3;  
    int y = 5;  
    int z = 0;  
    z = (x*y - x/2);  
    printf("%d ", z);  
    return 0;  
}
```

IDA Analysis for Q1.c:



The screenshot shows the assembly code for the `main` function in IDA Pro. The code is color-coded: comments in grey, C-style declarations in blue, variable declarations in green, and instructions in black. The assembly code corresponds to the C code provided in the previous block.

```
; Attributes: bp-based frame  
; int __cdecl main(int argc, const char **argv, const char **envp)  
public main  
main proc near  
  
var_10= dword ptr -10h  
var_C= dword ptr -0Ch  
var_8= dword ptr -8  
  
push    rbp  
mov     rbp, rsp  
sub     rsp, 30h  
call    _main  
mov     [rbp+var_10], 3  
mov     [rbp+var_C], 5  
mov     [rbp+var_8], 0  
mov     edx, [rbp+var_10]  
mov     eax, [rbp+var_C]  
imul    edx, eax  
mov     eax, [rbp+var_10]  
mov     ecx, eax  
shr     ecx, 1Fh  
add     eax, ecx  
sar     eax, 1  
sub     edx, eax  
mov     eax, edx  
mov     [rbp+var_8], eax  
mov     eax, [rbp+var_8]  
mov     [rbp+var_10], eax  
mov     eax, [rbp+var_10]  
mov     edx, eax  
lea     rcx, Format      ; "%d "  
call    printf  
mov     eax, 0  
add     rsp, 30h  
pop     rbp  
retn  
main endp
```

Q2: (Comments marked in red)

```
push ebp
mov ebp,esp
and esp,0FFFFFFF0h
sub esp,40h
call __main
mov dword ptr[esp+18h], 0Ch // Initialize 12
mov dword ptr[esp+1Ch],0Fh // Initialize 15
mov dword ptr[esp+20h],0DDh // Initialize 221
mov dword ptr[esp+24h],3 // Initialize 3
mov dword ptr[esp+28h],1B0h // Initialize 432
mov dword ptr[esp+2Ch],36h // Initialize 54
mov dword ptr[esp+30h],10h, // Initialize 16
mov dword ptr[esp+34h], 43h // Initialize 67
mov dword ptr[esp+3Ch],0 // Initialize 0
mov dword ptr[esp+38h],0 // Initialize 0
(Above steps initialized 10 elements)
jmp short loc_40157F // Jump to the loc_40157F
```

```
loc_401560:                ;CODE XREF: __main+84j
mov eax,[esp+38h]
mov eax,[esp+eax*4+18h]
cmp eax,[esp+3Ch] // Compare the 9th element with the ith element
jle short loc_40157A // If the ith element is less or equal than the 9th element, jump to
loc_40157A
mov eax,[esp+38h]
mov eax,[esp+eax*4+18h]
mov [esp+3Ch],eax // If the ith element is greater than 9th element, let the 9th element
equals to the value of the ith element
```

```
loc_40157A:                ;CODE XREF: __main+6Cj
add dword ptr[esp+38h],1 // Add one to the last element
```

```
loc_40157F:                ;CODE XREF: __main+5Ej
cmp dword ptr[esp+38h],7 // Compare the last element with value 7
jle short loc_401560 // Jump to loc_401560 if when the last element is less or equal
than 7; if it is not, the loop is over
```

```

mov eax,[esp+3Ch]
mov [esp+4],eax
mov dword ptr[esp], offset aD; "%d"
call _printf // Print out the 9th element, which is the greatest element among all
elements
mov eax,0
leave
retn
endp

```

The functionality of this assembly code is using a loop to compare the first eight elements and store the greatest value in the ninth element, and the tenth element is a counter of this loop. The result would be 432, which is 1B0h in hexadecimal.

My C code for Q2:

```
#include <stdio.h>
```

```

int main() {
    int elements[8] = {12, 15, 221, 3, 432, 54, 16, 67};
    int greatest = 0;
    int count = 0;

    while (count <= 7) {
        if (elements[count] > greatest)
            greatest = elements[count];
        count++;
    }
    printf("%d", greatest);
    return 0;
}

```

IDA Analysis for Q2.c:

```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

    var_30= dword ptr -30h
    var_2C= dword ptr -2Ch
    var_28= dword ptr -28h
    var_24= dword ptr -24h
    var_20= dword ptr -20h
    var_1C= dword ptr -1Ch
    var_18= dword ptr -18h
    var_14= dword ptr -14h
    var_8= dword ptr -8
    var_4= dword ptr -4

    push    rbp
    mov     rbp, rsp
    sub     rsp, 50h
    call    __main
    mov     [rbp+var_30], 0Ch
    mov     [rbp+var_2C], 0Fh
    mov     [rbp+var_28], 00Dh
    mov     [rbp+var_24], 3
    mov     [rbp+var_20], 180h
    mov     [rbp+var_1C], 36h
    mov     [rbp+var_18], 10h
    mov     [rbp+var_14], 43h
    mov     [rbp+var_4], 0
    mov     [rbp+var_8], 0
    jmp     short loc_4015A3
```

```
loc_4015A3:
    cmp     [rbp+var_8], 7
    jle     short loc_401585
```

```
loc_401585:
    mov     eax, [rbp+var_8]
    cdqe
    mov     eax, [rbp+rax*4+var_30]
    cmp     eax, [rbp+var_4]
    jle     short loc_40159F
```

```
    mov     eax, [rbp+var_4]
    mov     edx, eax
    lea     rcx, Format      ; "%d"
    call    printf
    mov     eax, 0
    add     rsp, 50h
    pop     rbp
    retn
main endp
```

```
    mov     eax, [rbp+var_8]
    cdqe
    mov     eax, [rbp+rax*4+var_30]
    mov     [rbp+var_4], eax
```

```
loc_40159F:
    add     [rbp+var_8], 1
```

Q3: (Comments marked in red)

```
push ebp
mov ebp,esp
and esp,0FFFFFFF0h
sub esp,20h
call __main
mov dword ptr[esp+1Ch],64h // Initialize x with value 100
jmp loc_4015D6 // Jump to the loc_4015D6
```

```
loc_40151B:                                ;CODE XREF: _main+DEj
mov ecx,[esp+1Ch]
mov edx,51EB851Fh // Initialize 1374389535 which is a magic number for dividing
mov eax,ecx
imul edx
sar edx,5 // edx = quotient
mov eax, ecx
sar eax,1Fh
sub edx,eax
(Above steps divide the x by 100 and store in edx)
mov eax,edx // Copy x/100 to eax
mov [esp+18h],eax // Initialize a value i with value x divided by 100
mov eax,[esp+18h] // Copy x/100 to eax
imul edx,eax,-64h // edx = x/100 * -100
mov eax,[esp+1Ch] // Copy x to eax
lea ecx,[edx+eax] // ecx = edx + eax = (x/100 * -100) + x
mov edx,66666667h // A magic number for dividing
mov eax,ecx
imul edx
sar edx,2
mov eax,ecx
sar eax,1Fh
sub edx,eax
(Above steps divide the (x/100 * -100) + x by 10 and store in edx)
mov eax,edx // Copy ((x/100 * -100) + x)/10 to eax
mov [esp+14h],eax // Initialize a value j with value ((x/100 * -100) + x)/10
mov ecx,[esp+1Ch]
mov edx,66666667h // A magic number for dividing
```

```

mov eax,ecx
imul edx
sar edx,2
mov eax,ecx
sar eax,1Fh
sub edx,eax
(Above steps divide value x by 10)
mov eax,edx // Copy x/10 to eax
shl eax,2 // Multiply by 4
add eax,edx //  $eax = x/10 + x/10*4$ 
add eax,eax //  $eax = (x/10 + x/10*4) * 2$ 
sub ecx,eax //  $ecx = x - (x/10 + x/10*4) * 2$ 
mov eax,ecx
mov [esp+10h],eax // Initialize a value k with value  $x - (x/10 + x/10*4) * 2$ 
mov eax,[esp+18h] // Copy i to eax
imul eax,[esp+18h] //  $eax = i^2$ 
imul eax,[esp+18h] //  $eax = i^3$ 
mov edx,eax //  $edx = i^3$ 
mov eax,[esp+14h] // Copy j to eax
imul eax,[esp+14h] //  $eax = j^2$ 
imul eax,[esp+14h] //  $eax = j^3$ 
add edx,eax //  $edx = i^3 + j^3$ 
mov eax,[esp+10h] // Copy k to eax
imul eax,[esp+10h] //  $eax = k^2$ 
mul eax,[esp+10h] //  $eax = k^3$ 
add eax,edx //  $eax = k^3 + i^3 + j^3$ 
cmp eax,[esp+1Ch] // Compare  $i^3 + j^3 + k^3$  with x
jnz short loc_4015D1 // Jump to loc_4015D1 if  $i^3 + j^3 + k^3$  is not equal to x
mov eax,[esp+1Ch] // If x equals  $i^3 + j^3 + k^3$ 
mov [esp+4],eax
mov dword ptr [esp], offset aD; "%d "
call _printf // We print out x

loc_4015D1                                     ;CODE XREF: _main+BBj
add dword ptr[esp+1Ch],1 // Add 1 to x

loc_4015D6                                     ;CODE XREF: _main+16j
cmp dword ptr[esp+1Ch],3E7h // Compare the x with 999

```

```

jle loc_40151B // Jump to the loc_40151B if x is less or equal than 999, otherwise the
loop is over
mov eax,0
leave
retn
endp

```

The functionality of this assembly code is to find values ranging from 100 to 999, which satisfies the condition $x = (x/100)^3 + (-100x/10)^3 + (x - (x/10 + x/10^4) * 2)^3$. My c program returns 4 results, which are 153, 370, 371, and 407.

My C code for Q3:

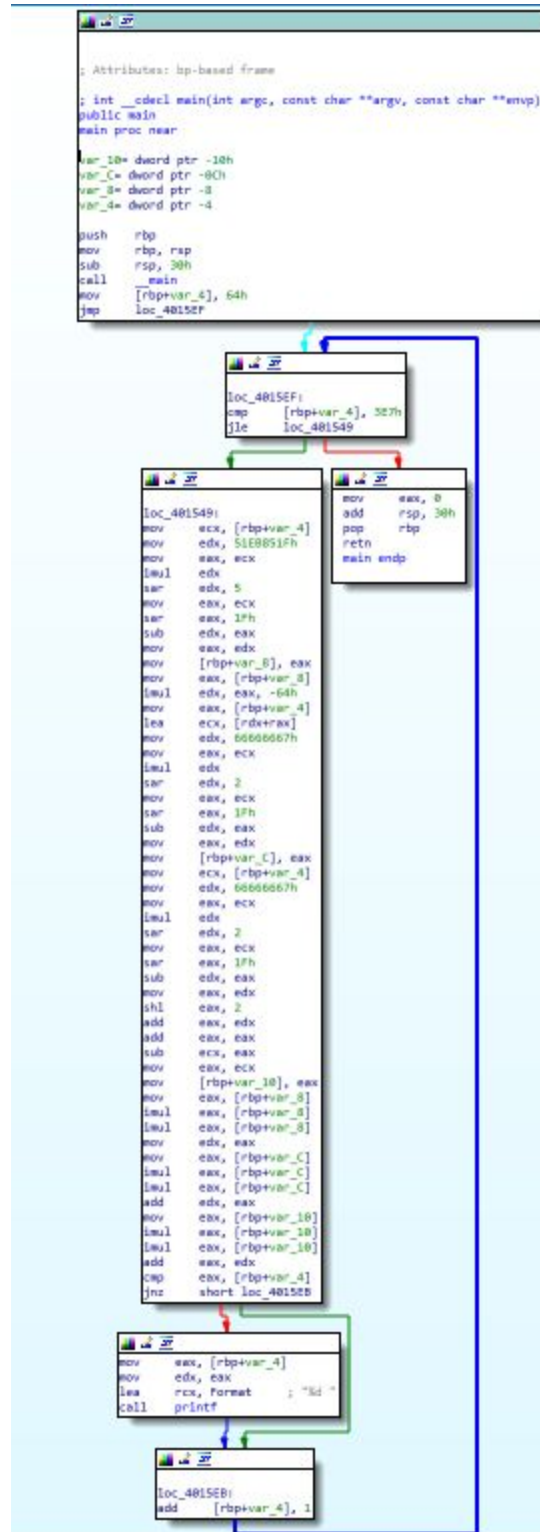
```

#include <stdio.h>

int main() {
    int x = 100;
    int i, j, k;
    while (x <= 999) {
        i = (int)(x/100);
        j = (int)((i*(-100) + x) / 10);
        k = x - ((int)(x/10) * 4 + (int)(x/10)) * 2;
        if ((i*i*i + j*j*j + k*k*k) == x)
            printf("%d ", x);
        x++;
    }
    return 0;
}

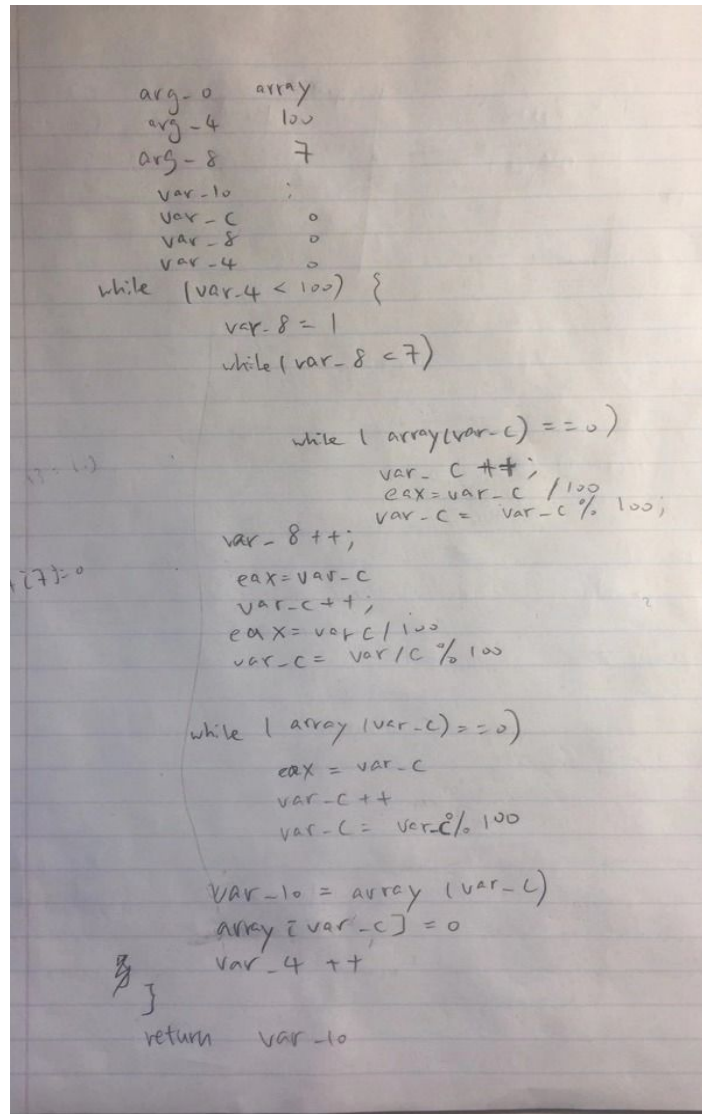
```


IDA Analysis for Q3.c:



Q4:

This assembly code first initializes three value $x = 7$, $y = 100$, and $z = 0$. Then there is a loop while $z < 100$, we do $\text{array}[z] = z + 1$ (since there is a line of code “`mov [esp+eax*4+14h],edx`”, I recognize there could exist an iterator in an array), then add z by 1. Once z is equal to 100, we should have an array consists of 100 integers from 1 to 100. Then we call another function, which includes three parameters x, y , and z , and print out the return value of this function. Then I analyze the assembly code of this function as following:



The image shows handwritten assembly code analysis on lined paper. The code is written in a mix of assembly-like syntax and pseudocode. It starts with variable declarations: `arg-0 array`, `arg-4 100`, and `arg-8 7`. Then it declares local variables: `var-10`, `var-c`, `var-8`, and `var-4`, all initialized to 0. A `while` loop is shown with the condition `(var-4 < 100)`. Inside this loop, `var-8` is incremented by 1, and another `while` loop is entered with the condition `(var-8 < 7)`. Inside this nested loop, there is a `while` loop with the condition `(array(var-c) == 0)`. Inside this loop, `var-c` is incremented, `eax` is set to `var-c`, and `var-c` is updated to `var-c % 100`. After the nested loops, `var-10` is set to `array(var-c)`, `array(var-c)` is set to 0, and `var-4` is incremented. The function ends with a `return var-10` statement.

```
arg-0 array
arg-4 100
arg-8 7
var-10
var-c 0
var-8 0
var-4 0
while (var-4 < 100) {
    var-8 = 1
    while (var-8 < 7)
        while (array(var-c) == 0)
            var-c ++;
            eax = var-c / 100;
            var-c = var-c % 100;
            var-8 ++;
            eax = var-c;
            var-c ++;
            eax = var-c / 100;
            var-c = var-c % 100;
        while (array(var-c) == 0)
            eax = var-c;
            var-c ++;
            var-c = var-c % 100;
        var-10 = array(var-c)
        array(var-c) = 0
        var-4 ++
    }
    return var-10
```

Based on my analysis, this function has 100 loops, and each loop sets 1 of 7 elements equals to 0. It will returns 50 in the end, which is the position of the last element set to 0.

My C Code for Q4:

```
#include <stdio.h>
```

```
int fun(int *array, int y, int x) {
    int count = 0;
    int pos = 0;
    int res;
    int i = 0;
    while (i < y) {
        count = 1;
        while (count < x) {
            while (array[pos] == 0)
                pos = (pos + 1) % y;
            count++;
            pos = (pos + 1) % y;
        }
        while (array[pos] == 0)
            pos = (pos + 1) % y;
        res = array[pos];
        array[pos] = 0;
        i++;
    }
    return res;
}

int main() {
    int array[100];
    int x = 7;
    int y = 100;
    int z = 0;
    int res;

    while (z < y) {
        array[z] = z + 1;
        z++;
    }
    res = fun(array, y, x);
    printf("%d\n", res);
    return 0;
}
```

IDA Analysis for Q4.c:

```
; Attributes: bp-based frame fpd=140h

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_1A0= dword ptr -1A0h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

push    rbp
sub     rsp, 1C0h
lea     rbp, [rsp+80h]
call    __main
mov     [rbp+140h+var_8], 7
mov     [rbp+140h+var_C], 64h
mov     [rbp+140h+var_4], 0
jmp     short loc_40166E
```

```
loc_40166E:
mov     eax, [rbp+140h+var_4]
cmp     eax, [rbp+140h+var_C]
jnl     short loc_401652
```

```
loc_401652:
mov     eax, [rbp+140h+var_4]
lea     edx, [rax+1]
mov     eax, [rbp+140h+var_4]
cdqe
mov     [rbp+rax*4+140h+var_1A0], edx
add     [rbp+140h+var_4], 1
```

```
mov     ecx, [rbp+140h+var_8]
mov     edx, [rbp+140h+var_C]
lea     rax, [rbp+140h+var_1A0]
mov     r8d, ecx
mov     rcx, rax
call    fun
mov     [rbp+140h+var_10], eax
mov     eax, [rbp+140h+var_10]
mov     edx, eax
lea     rcx, Format      ; "%d\n"
call    printf
mov     eax, 0
add     rsp, 1C0h
pop     rbp
retn
main endp
```

