Question 3

1. Segments

Name	Start	End	R	W	X
.text	00401000	00402000	R	.	X
.idata	00402000	004020B4	R	.	.
.rdata	004020B4	00403000	R	.	.
.data	00403000	00404000	R	W	.

2. Entry

Choose an entry point

Name	Address	Ordinal
start	004014B0	[main entry]

Line 1 of 1

OK Cancel Search Help

3. Main

_main	.text	004011C0

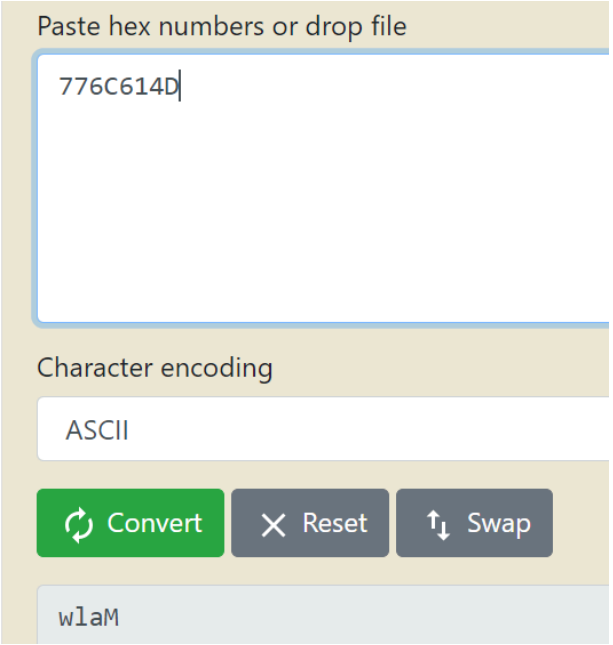
4. The program doesn't take any parameters because there is no argv reference at the code.
   - Before diving into the functions let's see the code in main:

```
push    ebp
mov     ebp, esp
sub     esp, 5Ch
mov     eax, ___security_cookie
xor     eax, ebp
mov     [ebp+var_4], eax
push    esi
push    edi
mov     ecx, 8
mov     esi, offset aAdvancedTopics ; "Advanced Topics In Malware 2024"
lea     edi, [ebp+var_38]
rep movsd
mov     eax, ds:dword_402110
mov     dword ptr [ebp+var_18], eax
mov     ecx, ds:dword_402114
mov     dword ptr [ebp+var_18+4], ecx
mov     edx, ds:dword_402118
mov     dword ptr [ebp+var_18+8], edx
mov     eax, ds:dword_40211C
mov     dword ptr [ebp+var_18+0Ch], eax
mov     cx, ds:word_402120
mov     word ptr [ebp+var_18+10h], cx
mov     ecx, 8
mov     esi, offset aAdvancedPersis ; "Advanced Persistent Threat (APT)"
lea     edi, [ebp+var_5C]
rep movsd
movsb
```

This first part is moving some strings like “Advanced Topics In Malware 2024” and “Advanced Persistent Threat (APT)”. But what is this in the middle? Let’s find out:

```
.rdata:004020F0 aAdvancedTopics db 'Advanced Topics In Malware 2024',0
.rdata:004020F0                                         ; DATA XREF: _main+17↑o
.rdata:00402110 dword_402110    dd 776C614Dh            ; DATA XREF: _main+21↑r
.rdata:00402114 dword_402114    dd 20657261h            ; DATA XREF: _main+29↑r
.rdata:00402118 dword_402118    dd 6C616E41h            ; DATA XREF: _main+32↑r
.rdata:0040211C dword_40211C    dd 69736979h            ; DATA XREF: _main+3B↑r
.rdata:00402120 word_402120     dw 73h                  ; DATA XREF: _main+43↑r
.rdata:00402122                 align 4
.rdata:00402124 aAdvancedPersis db 'Advanced Persistent Threat (APT)',0
```

We see this data which is passed and constructed on the go. Let’s analyze what’s this:



As we can see it’s hex values that represent a string, let’s put it back together (hex to char, and flip the order to little endian):
776C614D = Malw
20657261= are_
6C616E41 = Anal
69736979 = yisi
73 = s
Result -> “Malware Analyisis
Very Cool!

This can be confirmed in the decompiled view:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[36]; // [esp+8h] [ebp-5Ch] BYREF
  char v5[32]; // [esp+2Ch] [ebp-38h] BYREF
  char v6[20]; // [esp+4Ch] [ebp-18h] BYREF

  strcpy(v5, "Advanced Topics In Malware 2024");
  strcpy(v6, "Malware Analyisis");
  strcpy(v4, "Advanced Persistent Threat (APT)");
  sub_401000(v5, 3);
  sub_4010C0(v6);
  sub_401150(v4, 75);
  return 0;
}
```

5-6.
func1:
Ok let's hop to the first function which is called with the following parameters:
sub_401000("Advanced Topics In Malware", 3)

```
unsigned int __cdecl caesarCipher(const char *string_ATIM, int number_3)
{
  unsigned int result; // eax
  unsigned int i; // [esp+Ch] [ebp-8h]
  char char_i; // [esp+13h] [ebp-1h]

  for ( i = 0; ; ++i )
  {
    result = i;
    if ( i >= strlen(string_ATIM) )
      break;
    char_i = string_ATIM[i];
    if ( char_i < 97 || char_i > 122 )          // Chech if char not in a-z
    {
      if ( char_i >= 65 && char_i <= 90 )       // char is A-Z
        string_ATIM[i] = (char_i + number_3 - 65) % 26 + 65;// add +3 and do a wrap around (mod 26)
    }
    else
    {                                           // char is a-z
      string_ATIM[i] = (char_i + number_3 - 97) % 26 + 97;// add +3 and do a wrap around (mod 26)
    }
  }
  return result;
}
```

This function takes the string and a number and encrypt it with caesar cipher. adds +3 to the char value and wraps around - it uses the history accurate key = 3 that Julius Caesar used to encrypt his messages.

Let's go to the second function:
func2:
sub_4010C0("Malware Analyisis")

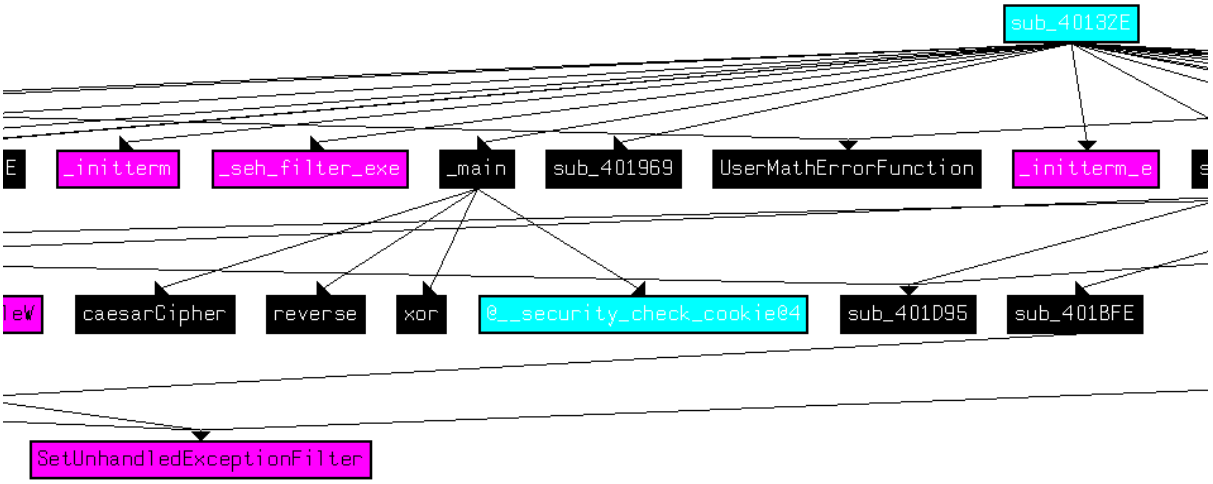
```
int __cdecl reverse(const char *string_MA)
{
  int result; // eax
  signed int string_length_17; // [esp+8h] [ebp-10h]
  int i; // [esp+10h] [ebp-8h]
  char char_i; // [esp+16h] [ebp-2h]

  string_length_17 = strlen(string_MA);
  for ( i = 0; ; ++i )
  {
    result = string_length_17 / 2;
    if ( i >= string_length_17 / 2 )
      break;
    char_i = string_MA[i];
    string_MA[i] = string_MA[string_length_17 - i - 1];
    string_MA[string_length_17 - i - 1] = char_i;
  }
  return result;
}
```

This function reverses the given string in place and returns the middle point of the string.
Nothing more than that…

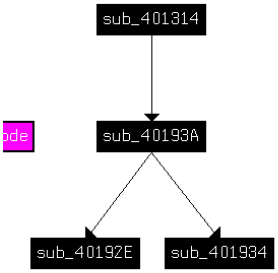
Let's hop on the last func3:
sub_401150("Advanced Persistent Threat (APT)", 75)

```
1  char __cdecl xor(char *string_APT, char number_75)
2  {
3    char result; // al
4    int i; // [esp+Ch] [ebp-Ch]
5    char *string_APT1; // [esp+10h] [ebp-8h]
6
7    string_APT1 = string_APT;
8    do
9      result = *string_APT1;
10   while ( *string_APT1++ );
11   for ( i = 0; i < string_APT1 - (string_APT + 1); ++i )
12   {
13     string_APT[i] ^= number_75;
14     result = i + 1;
15   }
16   return result;
17 }
```

This function implements a xor between the string and the number = 75, thus encrypting it via this xor. returns the number of bits xor-ed +1 (the size of the string+1).
7.
First of all Main is user written and the 3 functions we investigated:



After that we can spot some weird functions that are not called from any point at the code:



Those functions implement some weird code that does not look like

```
_DWORD *sub_40193A()
{
  _DWORD *v0; // eax
  int v1; // ecx
  _DWORD *result; // eax
  int v3; // ecx

  v0 = (_DWORD *)sub_40192E();
  v1 = v0[1];
  *v0 |= 4u;
  v0[1] = v1;
  result = (_DWORD *)sub_401934();
  v3 = result[1];
  *result |= 2u;
  result[1] = v3;
  return result;
}
```

a compiler based:
Those could be investigated further..