

南開大學

## 汇编语言与逆向技术课程实验报告

### 实验九： Reverse Engineering Exercises – Advanced



学 院 网络空间安全学院  
专 业 信息安全、法学双学位  
学 号 2212000  
姓 名 宋奕纬  
班 级 1061

## 一、实验目的

- 1、进一步熟悉静态反汇编工具 IDA Freeware;
- 2、熟悉将反汇编代码进行反编译的过程;
- 3、掌握对于反编译伪代码的逆向分析;
- 4、运用熟悉的编程语言，实现简单的脚本编写

## 二、实验原理

- 1、实验环境：反汇编工具 IDA Freeware

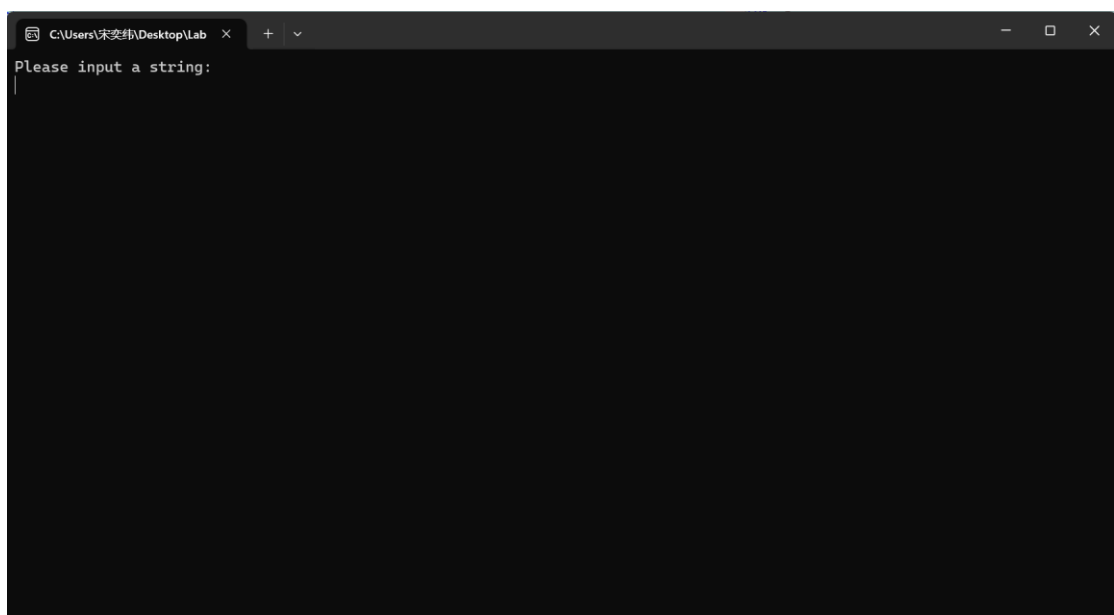
- 2、实验原理：

通过 IDA 得到二进制代码的反汇编代码，利用汇编所学知识对反汇编代码的数学计算、数据结构、条件判断、分支结构进行识别与分析。

## 三、实验过程

### （一）Task3

- 1、使用 IDA Freeware 打开 task3.exe 文件，查看其二进制代码的反汇编代码。（先运行程序大概了解其功能）





## 2、使用 IDA 的反编译功能得到 task3.exe 的伪代码。

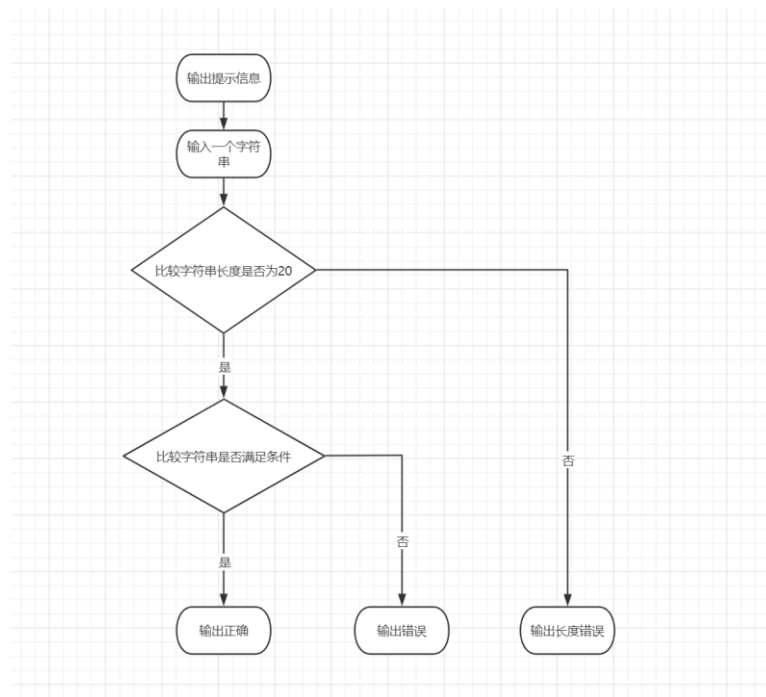
```
1 int __cdecl main_0(int argc, const char **argv, const char **envp)
2 {
3     unsigned int v3; // ecx
4     unsigned int v4; // ecx
5     char *v5; // eax
6     int v6; // edx
7     unsigned int v7; // ecx
8     unsigned int v9; // ecx
9     char v10[8]; // [esp+8h] [ebp-A4h] BYREF
10    char v11; // [esp+8h] [ebp-9Ch] BYREF
11    char v12[8]; // [esp+9h] [ebp-98h] BYREF
12    char v13; // [esp+14h] [ebp-90h] BYREF
13    char v14[12]; // [esp+15h] [ebp-8Fh] BYREF
14    char ArgList[24]; // [esp+24h] [ebp-80h] BYREF
15    char v15[20]; // [esp+3Ch] [ebp-68h]
16    char ArgList[80]; // [esp+50h] [ebp-54h] BYREF
17
18    qmemcpy(ArgList, "B~wsaw2{b9f2s2af{u(", 22);
19    v3 = 0;
20    ArgList[22] = 24;
21    ArgList[23] = 18;
22    do
23    {
24        ArgList[v3++] ^= 0x12u;
25        while ( v3 < 0x18 );
26        sub_7D11E5("%s", (char)ArgList);
27        sub_7D11E5("%s", (char)ArgList);
28        if ( strlen(ArgList) == 20 )
29        {
30            v16[0] = -15;
31            v6 = 0;
32            v16[1] = -55;
33            v16[2] = -31;
34            v16[3] = -1;
35            v16[4] = -25;
36            v16[5] = -109;
37            v16[6] = -12;
38            v16[7] = -17;
39            v16[8] = -44;
40            v16[9] = -24;
41            v16[10] = -17;
42            v16[11] = -64;
43            v16[12] = -50;
44            v16[13] = -4;
45            v16[14] = -30;
46            v16[15] = -47;
47            v16[16] = -3;
48            v16[17] = -64;
49            v16[18] = -15;
50            v16[19] = -4;
51            while ( (ArgList[v6] ^ 0xA5) == (unsigned __int8)v16[v6] )
52            {
53                if ( ++v6 >= 20 )
54                {
55                    v11 = 20;
56                    v7 = 0;
57                    qmemcpy(v12, "8%2d4vW", sizeof(v12));
58                    do
59                    {
60                        v12[v7++] - 1 ^= 0x57u;
61                        while ( v7 < 9 );
62                        sub_7D11E5("%s\n", (char)&v11);
63                        return 0;
64                    }
65                    v10[0] = -57;
66                    v9 = 0;
67                    v10[1] = -30;
68                    v10[2] = -1;
69                    v10[3] = -2;
70                    v10[4] = -9;
71                    v10[5] = -79;
72                    v10[6] = -112;
73                    do
74                    {
75                        v10[v9++] ^= 0x90u;
76                        while ( v9 < 7 );
77                        v5 = v10;
78                    }
79                    else
80                    {
81                        v13 = 4;
82                        v4 = 0;
83                        qmemcpy(v14, "!<4s?6=4'S", sizeof(v14));
84                        do
85                        {
86                            v14[v4++] - 1 ^= 0x53u;
87                            while ( v4 < 0xD );
88                            v5 = &v13;
89                        }
90                    }
91                }
92            }
93        }
94    } while ( v3 < 0x18 );
95}
```

## 3、对代码和伪代码进行分析。

### (1) 统观代码：

观察代码后，发现没有出现一开始运行程序时出现的“Please input a String:”字符串，同时也没有我们想看到的提示正确或者提示错误的提示信息。故猜测这个 task 需要先对提示信息进行分析，然后再去寻找可以输出“正确”提示信息的字符串。

对结构进行简单分析后作出以下流程图：



## (2) 解密提示信息:

Ps:

- 1、这里的伪代码直接将需要被处理的字符串给出，也可以跟据反汇编代码中的数字去分析（故写了多套脚本——在之后具体介绍）
- 2、对伪代码中函数和变量的理解：伪代码中的变量、数组其实都是地址，指向这个地址下所存的数据；sub\_7D11E5 这个函数的参数是输出的起始地址，从起始地址的数据开始输出

### ① “Please input a string: ”

```

qmemcpy(ArgList, "B~wsaw2{|bgf2s2af`{|u(", 22);
v3 = 0;
ArgList[22] = 24;
ArgList[23] = 18;
do
    ArgList[v3++] ^= 0x12u;
while ( v3 < 0x18 );
sub_7D11E5("%s", (char)ArgList);
  
```

将这个字符串每一位的 ascII 码与 0x12 进行异或后作为新的 ascII 码，生成一个新的字符串（解出来是 “Please input a string: ”）。

## ② “Correct! ”

```
v11 = 20;
v7 = 0;
qmemcpy(v12, "8%%24#vW", sizeof(v12));
do
    v12[v7++ - 1] ^= 0x57u;
while ( v7 < 9 );
sub_7D11E5("%s\n", (char)&v11);
```

将这个字符串每一位的 ascII 码与 0x54 进行异或后作为新的 ascII 码，生成一个新的字符串（解出来是 “Correct! ”）。

此处有一个细节，这里是将 V12 从-1 开始进行异或和输出的（即 V12 地址前的数据，即 V11 这个被赋值为 20 的数字，20 和 0x57(87)异或，解出了字符“C”）。

## ③ “Wrong! ”

```
}
v10[0] = -57;
v9 = 0;
v10[1] = -30;
v10[2] = -1;
v10[3] = -2;
v10[4] = -9;
v10[5] = -79;
v10[6] = -112;
do
    v10[v9++] ^= 0x90u;
while ( v9 < 7 );
```

将 V10 每一位的 ascII 码与 0x90 进行异或后作为新的 ascII 码，生成一个新的字符串（解出来是 “Wrong! ”）。

## ④ “Wrong length”

```
{
    v13 = 4;
    v4 = 0;
    qmemcpy(v14, "!<=4s?6=4';S", sizeof(v14));
    do
        v14[v4++ - 1] ^= 0x53u;
    while ( v4 < 0xD );
    v5 = &v13;
```

将这个字符串每一位的 ASCII 码与 0x54 进行异或后作为新的 ASCII 码，生成一个新的字符串（解出来是“Wrong length”）

这里与输出“Correct!”信息处细节类似——这里是将 V14 从 -1 开始进行异或和输出的（即 V14 地址前的数据，即 V13 这个被赋值为 4 的数字，4 和 0x57 (87) 异或，解出了字符“W”）。

### （3）解密需要输入的字符串：

```
sub_4E11EF("%s", (char)Arglist);
if ( strlen(Arglist) == 20 )
{
    v16[0] = -15;
    v6 = 0;
    v16[1] = -55;
    v16[2] = -31;
    v16[3] = -1;
    v16[4] = -25;
    v16[5] = -109;
    v16[6] = -12;
    v16[7] = -17;
    v16[8] = -44;
    v16[9] = -24;
    v16[10] = -17;
    v16[11] = -64;
    v16[12] = -50;
    v16[13] = -4;
    v16[14] = -30;
    v16[15] = -47;
    v16[16] = -3;
    v16[17] = -64;
    v16[18] = -15;
    v16[19] = -4;
    while ( (Arglist[v6] ^ 0x45) == (unsigned __int8)v16[v6] )
    {
        if ( ++v6 >= 20 )
        {
            v11 = 20;
            v7 = 0;
            memcpy(v12, "8%24#vW", sizeof(v12));
            do
            {
                v12[v7++ - 1] ^= 0x57u;
                while ( v7 < 9 );
                sub_4E11E5("%s\n", (char)&v11);
                return 0;
            }
        }
    }
}
```

先定义数组 V16

条件：

$\text{Arglist}[i] \oplus 0x45 = V6[i]$ ;

若要解出原始的 Arglist，则利用  $a \oplus b = c$ 、 $a \oplus b \oplus b = a \oplus c \oplus b$ ，得

$\text{Arglist}[i] = V6[i] \oplus 0x45$

即将数组 V6 的每一个元素与 0x45 异或后推出字符串的每一个字符的 ASCII 码，进而推出字符串。

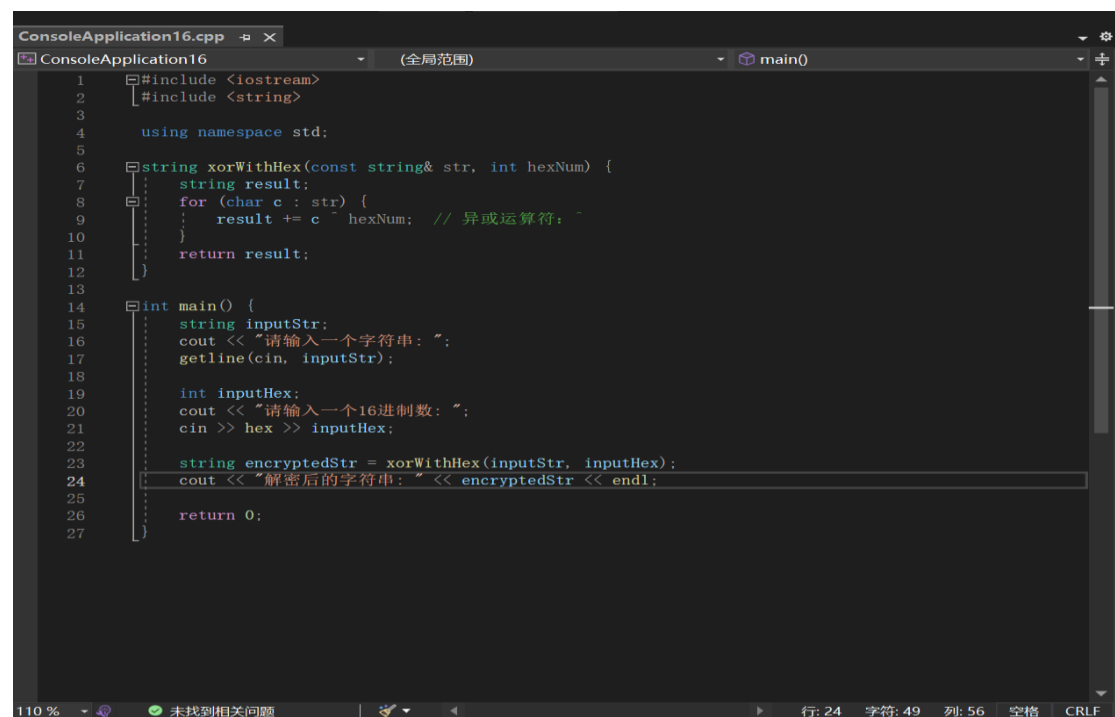
## 4、脚本实现

写了两个脚本：

第一个可以输入一个字符串，输入一个十六进制数，将字符串每个字符的 ASCII 码和十六进制数异或后，生成新的字符串并输出；一个可以输入一个数组和一个数，将数组中的元素和这个数异或后得到对应的字符串。

这里可以只用第二个脚本，根据汇编代码挨着输入数据即可，但效率低。

第一个脚本程序虽然存在一些问题，只能处理字符串（伪代码给出的），但是效率较高，故将两者结合起来进行计算。



```
ConsoleApplication16.cpp  [X]
ConsoleApplication16  (全局范围)  main()
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  string xorWithHex(const string& str, int hexNum) {
7      string result;
8      for (char c : str) {
9          result += c ^ hexNum; // 异或运算符: ^
10     }
11     return result;
12 }
13
14 int main() {
15     string inputStr;
16     cout << "请输入一个字符串: ";
17     getline(cin, inputStr);
18
19     int inputHex;
20     cout << "请输入一个16进制数: ";
21     cin >> hex >> inputHex;
22
23     string encryptedStr = xorWithHex(inputStr, inputHex);
24     cout << "解密后的字符串: " << encryptedStr << endl;
25
26     return 0;
27 }
```

```
ConsoleApplication17.cpp  x
ConsoleApplication17 (全局范围)  main()
1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4
5  int main() {
6      int N=0;
7      cout << "输入数组长度:" << endl;
8      cin >> N;
9      int m;
10     cout << "输入16进制数对应的十进制数:" << endl;
11     cin >> m;
12     int inputArray[100];
13     char a[100];
14
15     // 输入数组
16     cout << "请输入数组元素" << endl;
17     for (int i = 0; i < N; i++) {
18         cin >> inputArray[i];
19     }
20     for (int i = 0; i < N; i++) {
21         inputArray[i] = inputArray[i] * m;
22     }
23     for (int i = 0; i < N; i++) {
24         a[i] = (char)inputArray[i];
25     }
26     cout << "你的字符串: ";
27     for (int i = 0; i < N; i++) {
28         cout << a[i] ;
29     }
30     return 0;
31 }
```

## (1) 解密提示信息

```
qmemcpy(ArgList, "B-wsaw2{|bgf2s2af`{|u{", 22);
v3 = 0;
ArgList[22] = 24;
ArgList[23] = 18;
do
    ArgList[v3++] ^= 0x12u;
while ( v3 < 0x18 );
sub_4E11E5("%s", (char)ArgList);
sub_4E11EF("%s", (char)ArgList);
if ( strlen(ArgList) == 20 )
{
    v16[0] = -15;
    v6 = 0;
    v16[1] = -55;
    v16[2] = -31;
    v16[3] = -1;
    v16[4] = -25;
    v16[5] = -100;
}
```

Microsoft Visual Studio 调试 x + v

请输入一个字符串: B-wsaw2{|bgf2s2af`{|u{  
请输入一个16进制数: 12  
解密后的字符串: Please input a string:

D:\cpp oj\ConsoleApplication16\x64\Debug\ConsoleApplication16.exe (进程 20184)已退出, 代码为 0。  
按任意键关闭此窗口. . .|

解出 “Please input a string: ”

Microsoft Visual Studio 调试 x + v

输入数组长度:  
7  
输入16进制数:  
144  
请输入数组元素  
-57 -30 -1 -2 -9 -79 -112  
Wrong!  
D:\cpp oj\ConsoleApplication17\x64\Debug\ConsoleApplication17.exe (进程 19716)已退出, 代码为 0。  
按任意键关闭此窗口. . .|

解出 “Wrong! ”



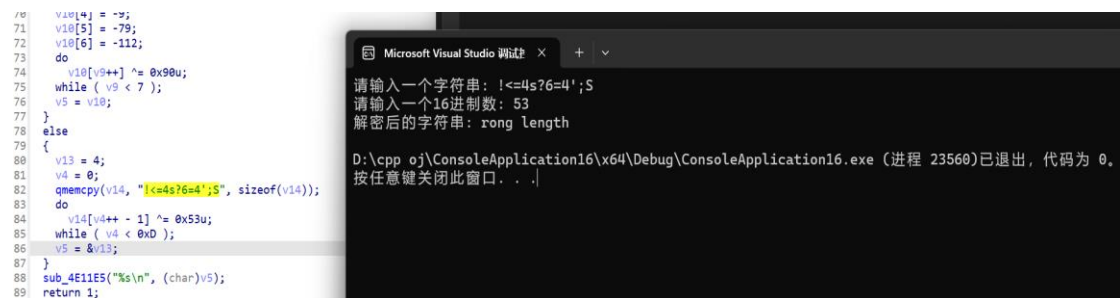
```
Microsoft Visual Studio 调试 x + v
输入数组长度:
1
输入16进制数对应的十进制数:
87
请输入数组元素
20
你的字符串: C
D:\cpp oj\ConsoleApplication17\x64\Debug\ConsoleApplication17.exe (进程 29200)已退出, 代码为 0。
按任意键关闭此窗口. . .|
```

```
Microsoft Visual Studio 调试 x + v
请输入一个字符串: 8%24#vW
请输入一个16进制数: 57
解密后的字符串: orrect!

D:\cpp oj\ConsoleApplication16\x64\Debug\ConsoleApplication16.exe (进程 8980)已退出, 代码为 0。
按任意键关闭此窗口. . .|
```

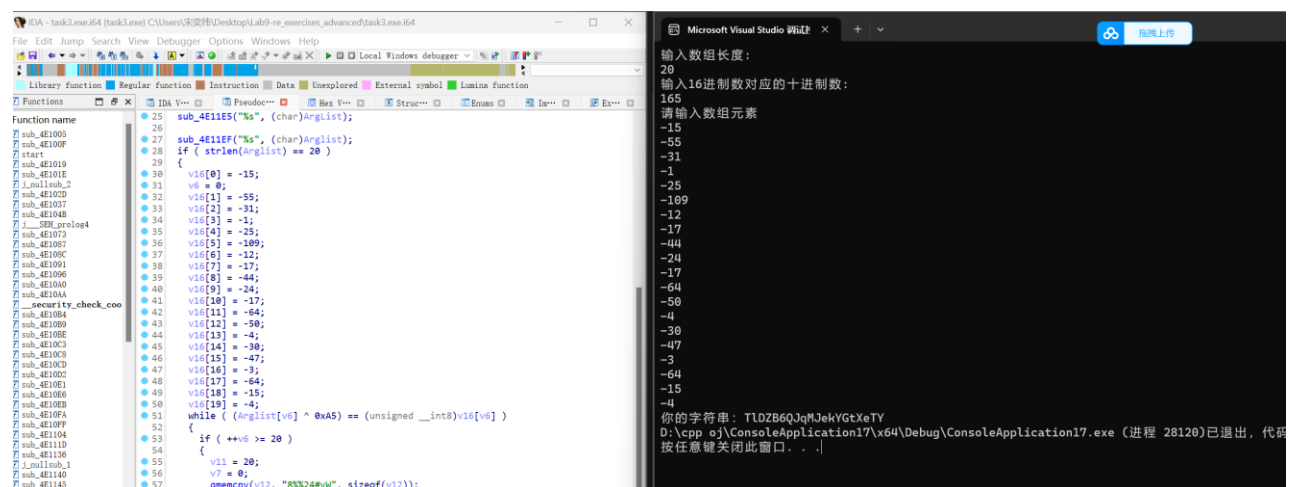
解出“Correct!”

```
Microsoft Visual Studio 调试 x + v - □ x
输入数组长度:
1
输入16进制数对应的十进制数:
83
请输入数组元素
4
你的字符串: W
D:\cpp oj\ConsoleApplication17\x64\Debug\ConsoleApplication17.exe (进程 4724)已退出, 代码为 0。
按任意键关闭此窗口. . .|
```



解出“Wrong length”

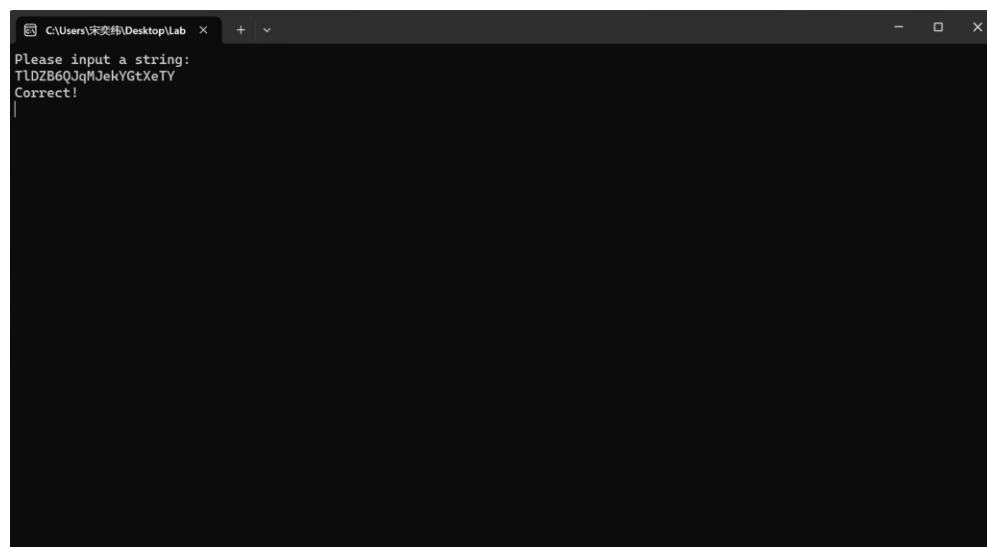
## (2) 解密字符串



解出字符串为“T1DZB6QJqMJekYGtXeTY”

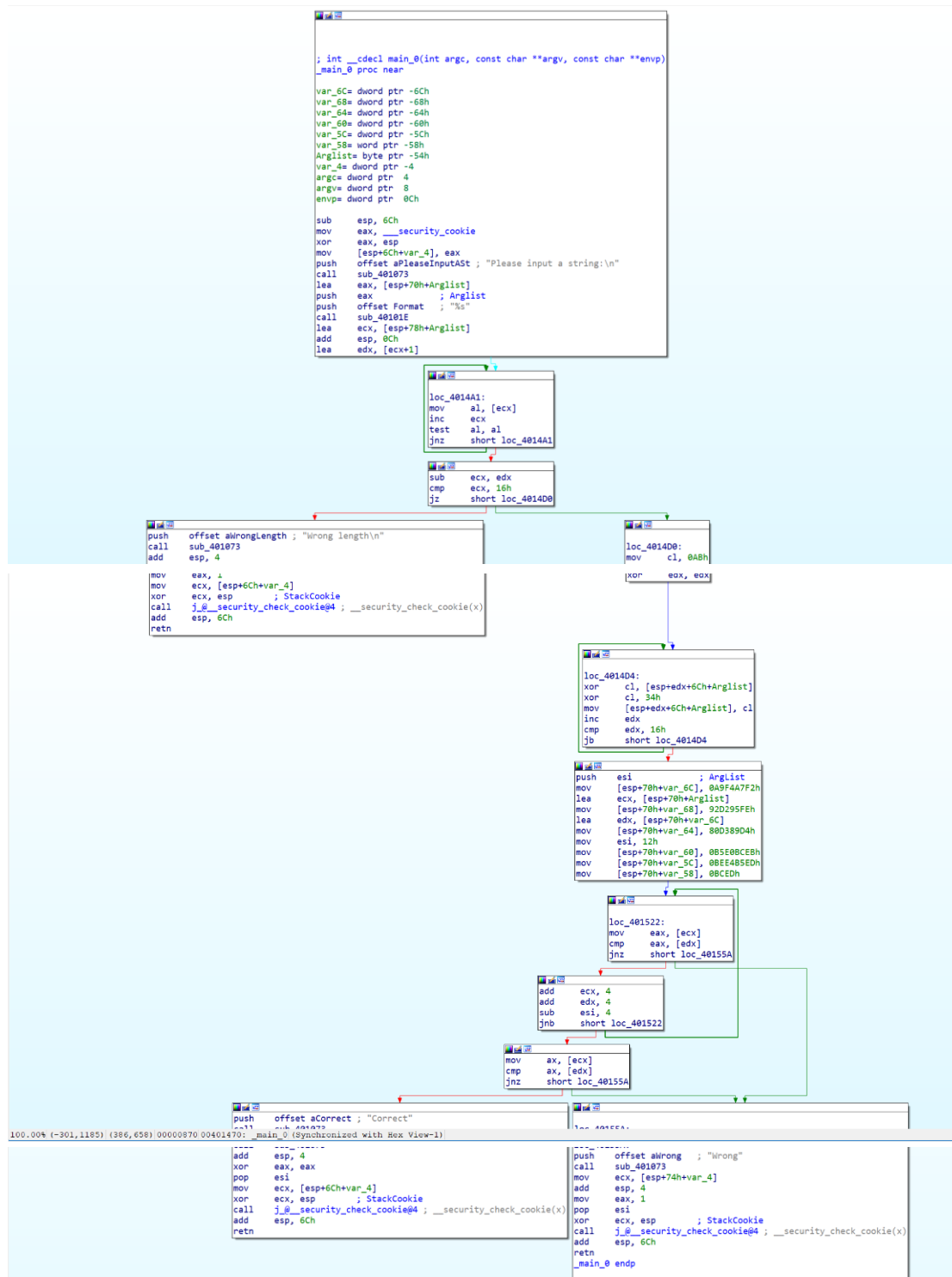
## 5、 测试运行。

将字符串“T1DZB6QJqMJekYGtXeTY”输入程序中,得到正确的结果。



## (二) Task4

- 1、 使用 IDA Freeware 打开 task4.exe 文件，查看其二进制代码的反汇编代码。



- 2、使用 IDA 的反编译功能得到 task4.exe 的伪代码。

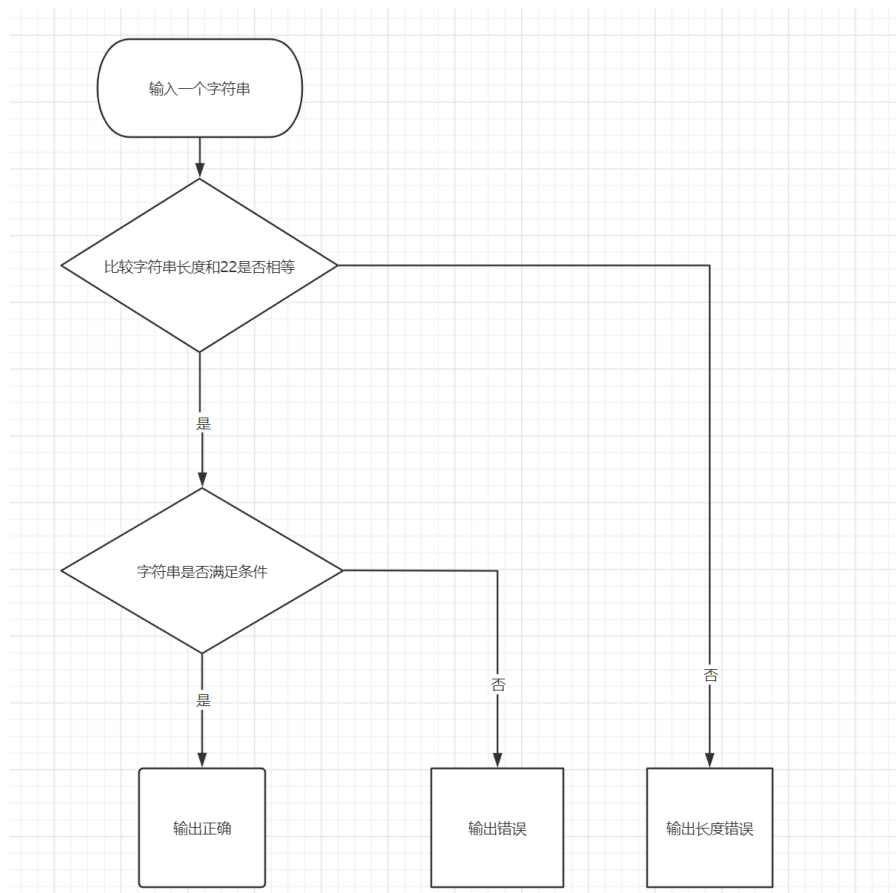
```

1 int __cdecl main_0(int argc, const char **argv, const char **envp)
2 {
3     char v3; // si
4     char v5; // cl
5     unsigned int i; // edx
6     char *v7; // ecx
7     int *v8; // edx
8     unsigned int v9; // esi
9     bool v10; // cf
10    char v11; // [esp-4h] [ebp-70h]
11    int v12[5]; // [esp+0h] [ebp-6Ch] BYREF
12    __int16 v13; // [esp+14h] [ebp-50h]
13    char Arglist[80]; // [esp+18h] [ebp-54h] BYREF
14
15    sub_401073("Please input a string:\n", v12[0]);
16    sub_40101E("%s", (char)Arglist);
17    if ( strlen(Arglist) == 22 )
18    {
19        v5 = -85;
20        for ( i = 0; i < 0x16; ++i )
21        {
22            v5 ^= Arglist[i] ^ 0x34;
23            Arglist[i] = v5;
24        }
25        v11 = v3;
26        v12[0] = -1443584014;
27        v7 = Arglist;
28        v12[1] = -1891692802;
29        v8 = v12;
30        v12[2] = -2133620268;
31        v9 = 18;
32        v12[3] = -1243562773;
33        v12[4] = -1092307475;
34        v13 = -17171;
35        while ( *(_DWORD *)v7 == *v8 )
36        {
37            v7 += 4;
38            ++v8;
39            v10 = v9 < 4;
40            v9 = 4;
41            if ( v10 )
42            {
43                if ( *(_WORD *)v7 == *(_WORD *)v8 )
44                {
45                    sub_401073("Correct", v11);
46                    return 0;
47                }
48                break;
49            }
50        }
51        sub_401073("Wrong", v11);
52        return 1;
53    }
54    return 1;
55 }

```

### 3、 对代码和伪代码进行分析。

还是对代码进行总体分析，分析其逻辑与结构。  
作出一张流程图：

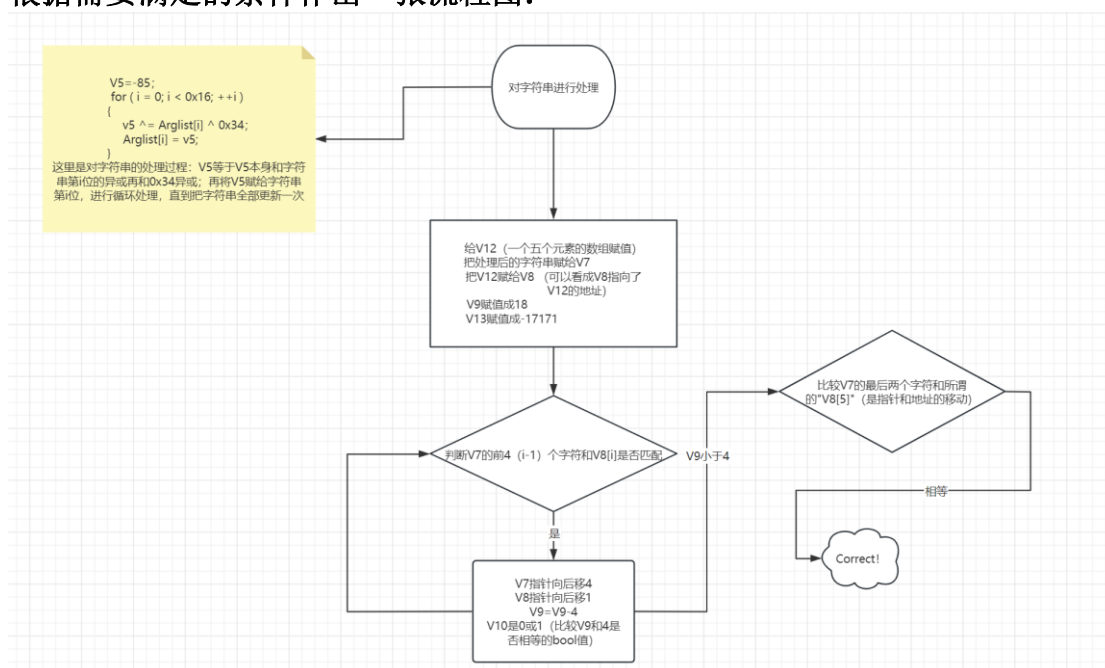


总体逻辑不复杂，主要是对字符串所满足条件的分析。

直接看字符串需要满足条件的伪代码进行分析：

```
v5 = -85;
for ( i = 0; i < 0x16; ++i )
{
    v5 ^= Arglist[i] ^ 0x34;
    Arglist[i] = v5;
}
v11 = v3;
v12[0] = -1443584014;
v7 = Arglist;
v12[1] = -1831692802;
v8 = v12;
v12[2] = -2133620268;
v9 = 18;
v12[3] = -1243562773;
v12[4] = -1092307475;
v13 = -17171;
while ( *(_DWORD *)v7 == *v8 )
{
    v7 += 4;
    ++v8;
    v10 = v9 < 4;
    v9 -= 4;
    if ( v10 )
    {
        if ( *(_WORD *)v7 == *(_WORD *)v8 )
        {
            sub_EE1073("Correct", v11);
            return 0;
        }
    }
}
```

根据需要的条件作出一张流程图：



看上去是双层循环，其实是在比较处理后的字符串和数组之间的对应关系。

若想要输出“Correct!”：

①对字符串进行处理，得到新的字符串。

②新的字符串和从地址 V8 开始的数据进行比较。

\*(\_DWORD \*)v7 是从当前地址强制转化为双字类型(32 位)，一个字符是 8 位，故一个双字可以存四个字符的数据。理解成每次比较 4 个字符——这样的话将 V8 存储的数组用完也只能比较 20 个字符。故用了下一条指令，即在前 20 个字符比较之后均符合的情况下，比较后两个字符是否满足条件：

if ( \*(\_WORD \*)v7 == \*(\_WORD \*)v8 )，从 V7 比较完前 20 个字符后的地址开始，转换成单字，存储两个字符的信息，与 V8 地址之后的 V9 转化成的

单字类型进行比较，如果二者相等，则输出正确的提示语。  
强制类型转化过程中，譬如处理后的字符串中有 4 位是“abcd”，则 a 的信息先进行存储，存储到信息的第 4 位（简单理解为先入后出的栈操作）。

故若想求出原始字符串，就要先求出被处理后的字符串。  
设处理后的是 B[22], 处理前的是 A[22]. 可以通过如下方式进行计算。

```
int v5 = -85;
for (int i = 0; i < 22; ++i){
    A[i] = v5 ^ B[i] ^ 0x34;
    v5 = A[i];
}
```

被处理后的字符串则可以通过 V8 存储的数组及 V9 得出（这里直接观察汇编代码更容易得出）

```
mov     [esp+70h+var_6C], 0A9F4A7F2h
lea     ecx, [esp+70h+Arglist]
mov     [esp+70h+var_68], 92D295FEh
lea     edx, [esp+70h+var_6C]
mov     [esp+70h+var_64], 80D389D4h
mov     esi, 12h
mov     [esp+70h+var_60], 0B5E0BCEBh
mov     [esp+70h+var_5C], 0BEE4B5EDh
mov     [esp+70h+var_58], 0BCEDh
```

V8 中存 8 位 16 进制数（一个双字大小，每两位对应一个字符）

V9 是一个 4 位 16 进制数（每两位对应一个字符）

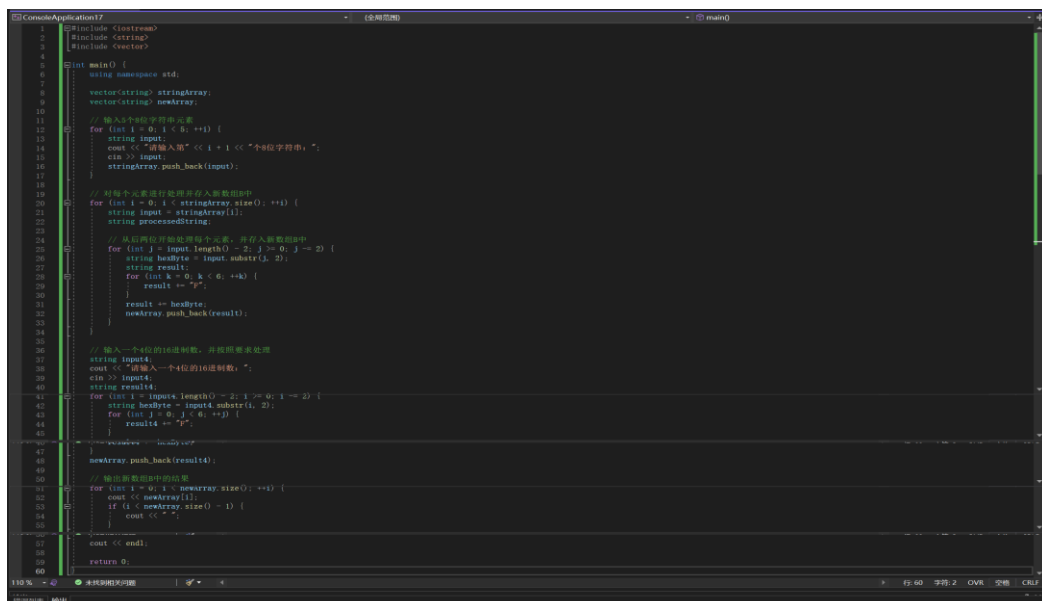
具体实现就是对每个元素从后两位开始处理。

比如 A9F4A7F2, 就是从 F2 开始, 写成 FFFFFFF2 FFFFFFFA7 FFFFFFFF4 FFFFFFFFA9, 每个都对应一个有符号的十进制数, 再对应处理后的字符串的 ascii 码。每一个元素都如上处理。

故整个流程就是利用 V8 和 V9 求出转换后的字符串的 ascii 码，然后逆推出原来字符串的 ascii 码，最后根据 ascii 码推出对应的字符串。

#### 4、脚本计算。

- （1）对 V8 和 V9 进行处理，整合出一个数组，存放 22 个转化后字符串中字符的 8 位 16 进制数。



```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

vector<string> stringArray;
vector<string> newArray;

// 输入4个字符并处理
for (int i = 0; i < k; ++i) {
    string input;
    cout << "请输入第" << i + 1 << "个4位字符串: ";
    cin >> input;
    stringArray.push_back(input);
}

// 对每个元素进行处理并存储在数组中
for (int i = 0; i < stringArray.size(); ++i) {
    string input = stringArray[i];
    string processedString;

    // 从后两位开始处理每个元素，并存储在数组中
    for (int j = input.length() - 2; j >= 0; j -= 2) {
        string hexByte = input.substr(j, 2);
        string result;
        for (int k = 0; k < 4; ++k) {
            result += "F";
        }
        result += hexByte;
        newArray.push_back(result);
    }

    // 输入一个4位的16进制数，并处理
    string input4;
    cout << "请输入一个4位的16进制数: ";
    cin >> input4;
    string result4;
    for (int i = input4.length() - 2; i >= 0; i -= 2) {
        string hexByte = input4.substr(i, 2);
        for (int j = 0; j < 4; ++j) {
            result4 += "F";
        }
        result4 += hexByte;
    }
    newArray.push_back(result4);
}

// 输出数组中的结果
for (int i = 0; i < newArray.size(); ++i) {
    cout << newArray[i];
    if (i < newArray.size() - 1) {
        cout << " ";
    }
}

cout << endl;
return 0;
```

```
Microsoft Visual Studio 调试
请输入第1个8位字符串: A9F4A7F2
请输入第2个8位字符串: 92D295FE
请输入第3个8位字符串: 80D389D4
请输入第4个8位字符串: B5E0BCEB
请输入第5个8位字符串: BEE4B5ED
请输入一个4位的16进制数: BCED
FFFFFFFF2 FFFFFFFA7 FFFFFFFF4 FFFFFFFA9 FFFFFFFFE FFFFFFF95 FFFFFFFD2 FFFFFFF92 FFFFFFFD4 FFFFFFF89 FFFFFFFD3 FFFFFFF80 FFFFFFFEB FFF
FFBFC FFFFFFFE0 FFFFFFFB5 FFFFFFFED FFFFFFFB5 FFFFFFFE4 FFFFFFFBE FFFFFFFEDFFFFFFBFC

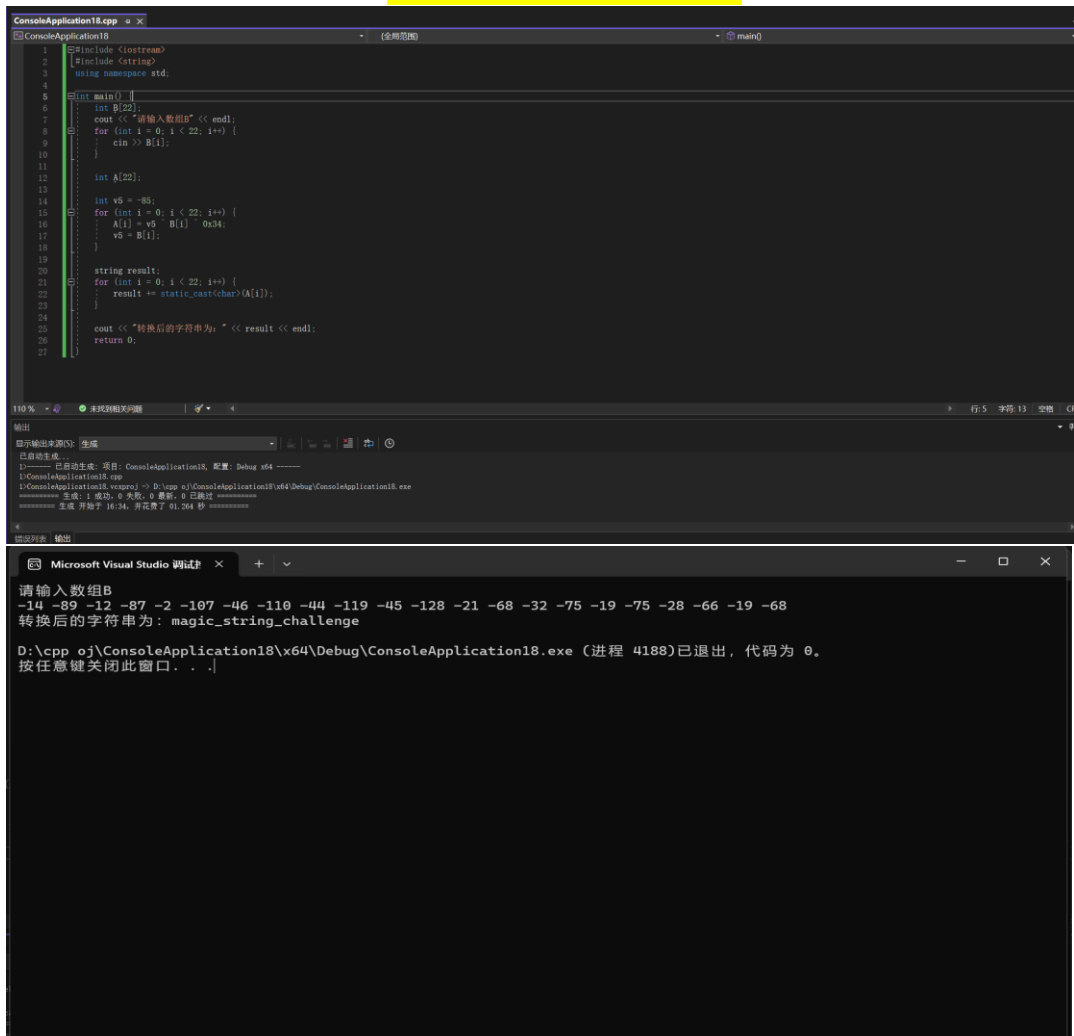
D:\cpp oj\ConsoleApplication17\x64\Debug\ConsoleApplication17.exe (进程 13012)已退出, 代码为 0。
按任意键关闭此窗口. . .
```

## (2) 对得到的数组进行处理,全部转化成对应的 10 进制数(作为数组 B)

```
ConsoleApplication16.cpp
1 #include <iostream>
2 #include <string>
3 #include <sstream>
4
5 using namespace std;
6
7 int main() {
8     string hexStrings[22];
9     int decimalValues[22];
10
11     // 从命令行读入22个字符串
12     for (int i = 0; i < 22; ++i) {
13         cin >> hexStrings[i];
14     }
15
16     // 将每个字符串转换为符号十进制数
17     for (int i = 0; i < 22; ++i) {
18         stringstream ss
19             ss << hex << hexStrings[i];
20         unsigned int value;
21         ss >> value;
22         if (value > 0x7FFFFFFF) { // 如果大于最大有符号整数, 将其作为负数处理
23             value -= 0x100000000;
24         }
25         decimalValues[i] = static_cast<int>(value);
26     }
27
28     // 输出数组内容
29     for (int i = 0; i < 22; ++i) {
30         cout << decimalValues[i] << " ";
31     }
32
33     return 0;
34 }
```

```
Microsoft Visual Studio 调试
FFFFFFFF2 FFFFFFFA7 FFFFFFFF4 FFFFFFFA9 FFFFFFFFE FFFFFFF95 FFFFFFFD2 FFFFFFF92 FFFFFFFD4 FFFFFFF89 FFFFFFFD3 FFFFFFF80 FFFFFFFEB FFF
FFBFC FFFFFFFE0 FFFFFFFB5 FFFFFFFED FFFFFFFB5 FFFFFFFE4 FFFFFFFBE FFFFFFFEDFFFFFFBFC
-14 -89 -12 -87 -2 -107 -46 -110 -44 -119 -45 -128 -21 -68 -32 -75 -19 -75 -28 -66 -19 -68
D:\cpp oj\ConsoleApplication16\x64\Debug\ConsoleApplication16.exe (进程 30264)已退出, 代码为 0。
按任意键关闭此窗口. . .
```

- (3) 对得到的数组 B 进行处理,得到存放正确字符串 ASCII 码的数组 A, 根据 A 得到字符串 **magic\_string\_challenge**



The screenshot displays the Visual Studio IDE with a C++ project named 'ConsoleApplication18'. The code in 'main.cpp' defines an array B of 22 integers, processes them into array A, and then constructs a string result from A. The output window shows the execution results, including the input array B, the resulting string 'magic\_string\_challenge', and the program's exit status.

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main() {
6     int B[22];
7     cout << "请输入数组B" << endl;
8     for (int i = 0; i < 22; i++) {
9         cin >> B[i];
10    }
11
12    int A[22];
13
14    int v5 = -85;
15    for (int i = 0; i < 22; i++) {
16        A[i] = v5 * B[i] % 31;
17        v5 = B[i];
18    }
19
20    string result;
21    for (int i = 0; i < 22; i++) {
22        result += static_cast<char>(A[i]);
23    }
24
25    cout << "转换后的字符串为: " << result << endl;
26    return 0;
27 }
```

输出

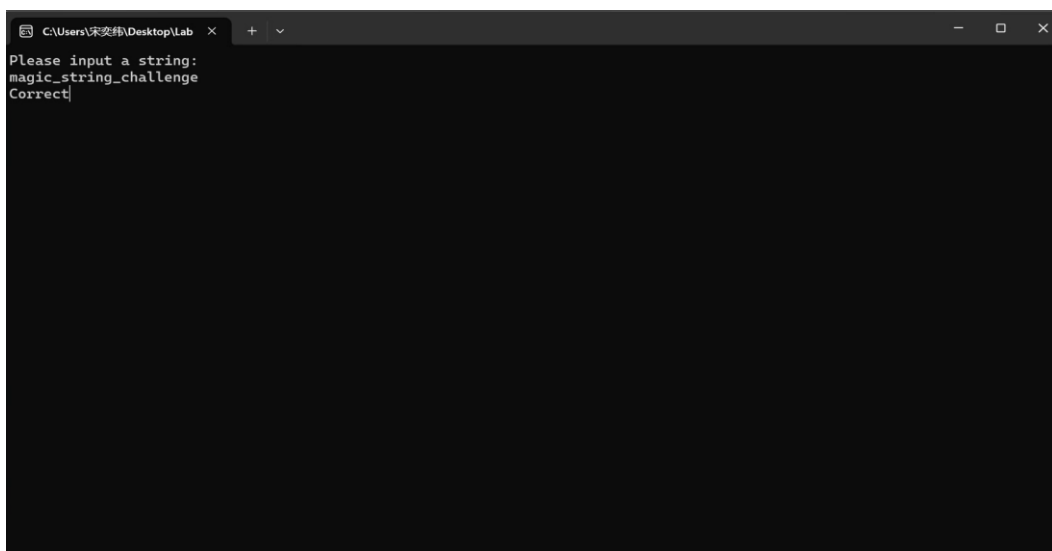
```
已启动生成...
D:\... 已启动生成: 项目: ConsoleApplication18, 配置: Debug x64
D:\ConsoleApplication18\...
D:\ConsoleApplication18\vcproj\... D:\cpp oj\ConsoleApplication18\Debug\ConsoleApplication18.exe
***** 生成: 1 成功: 0 失败: 0 耗时: 0 已编译 *****
***** 生成 开始于 16:34, 并花费了 01.264 秒 *****
```

Microsoft Visual Studio 调试

```
请输入数组B
-14 -89 -12 -87 -2 -107 -46 -110 -44 -119 -45 -128 -21 -68 -32 -75 -19 -75 -28 -66 -19 -68
转换后的字符串为: magic_string_challenge
D:\cpp oj\ConsoleApplication18\x64\Debug\ConsoleApplication18.exe (进程 4188)已退出, 代码为 0。
按任意键关闭此窗口。 . . .
```

## 5、测试运行。

将字符串 “**magic\_string\_challenge**” 输入, 得到正确答案。



The screenshot shows a terminal window titled 'C:\Users\宋奕伟\Desktop\Lab'. It displays the program's prompt 'Please input a string:', the user input 'magic\_string\_challenge', and the program's response 'Correct'.

```
C:\Users\宋奕伟\Desktop\Lab
Please input a string:
magic_string_challenge
Correct
```



## 四、实验结论及心得体会

- 1、熟悉静态反汇编工具 IDA Freeware。
- 2、熟悉反汇编代码的逆向分析过程。
- 3、掌握反汇编语言中的数学计算、数据结构、条件判断、分支结构的识别和逆向分析。
- 4、对地址、变量、指针都有了更深入的理解
- 5、对汇编语言有了更深入的了解。
- 6、提升了解决问题的能力，学会多种编程语言综合使用。