汇编语言与逆向工程

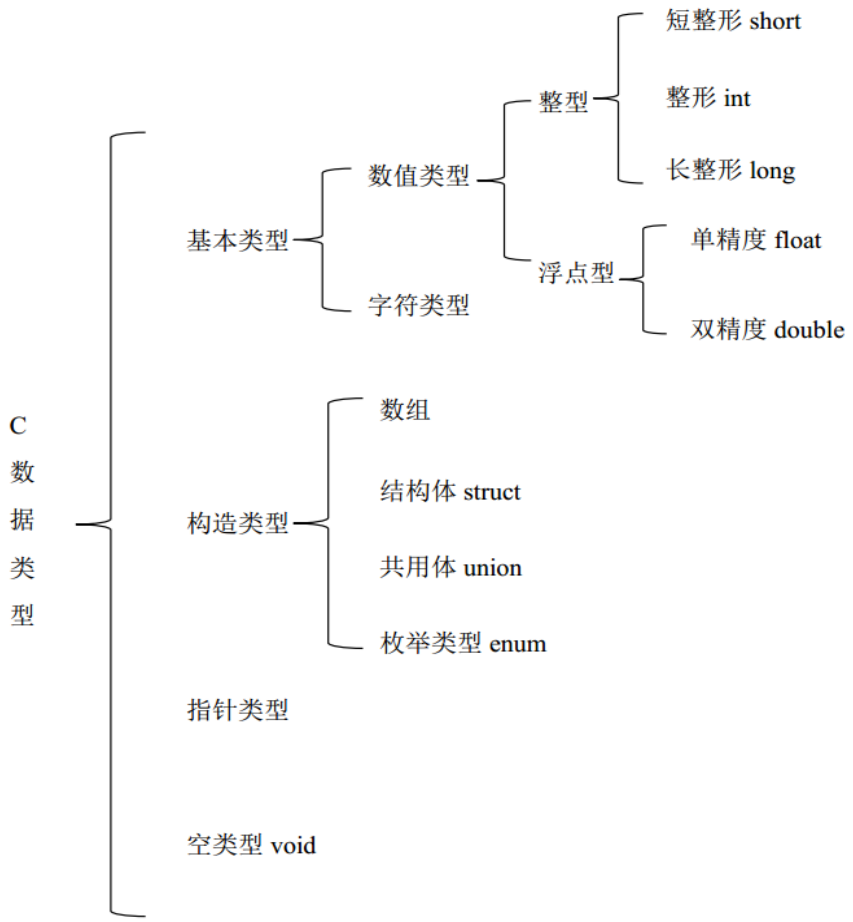
北京邮电大学
崔宝江

第三章从C语言看汇编

- 一. 基本数据类型
- 二. 流程控制语句
- 三. 变量表现形式

一. 基本数据类型

C语言基本的数据类型



一. 基本数据类型

- 本小节介绍基本类型和指针类型在汇编中的表现形式
- 构造类型将在下一章介绍
- 空类型是需要转化为其他形式的类型

一. 基本数据类型

- 内存中任何类型的变量，都以字节的形式存储和运行于内存中
  - windows中，一个字等于两个字节
  - 一个字节由两个十六进制数表示
  - 一个十六进制数可用4个二进制数表示
  - 一个字节也就是由8个二进制数组成
  - 32位程序和64位程序，只需要记住计算机在处理时只对8字节，4字节，2字节和1字节进行处理
    - 计算机的处理“只看字节不看类型”

一. 基本数据类型

- 用一个C语言和汇编对比的例子，从汇编看C语言基本数据类型
  - 定义了基本类型的数组，使用相应的指针指向这些数组
  - 通过指针自加的方式，让大家认识基本类型在汇编中的表现形式

一. 基本数据类型

```
intmain(intargc,char*argv[])
{
        /* define array */
        short shortValue[4]={100,};
        int intValue[4]={200,};
        long longValue[4]={300,};
        float floatValue[4]={400.1,};
        double doubleValue[4]={500.02,};
        char charValue[4]={48,};
        /* define array */
        int i;
        /* define points */
        short *pShortValue=shortValue;
        int *pIntValue=intValue;
        long *pLongValue=longValue;
        float *pFloatValue=floatValue;
        double *pDoubleValue=doubleValue;
        char *pCharValue=charValue;
        /* define points */
```


一. 基本数据类型

```
 /* show point in the memory */
        printf("pShortValue: %p\npIntValue: %p\npLongValue:
%p\npFloatValue: %p\npDoubleValue: %p\npCharValue: %p\n",
pShortValue,pIntValue,pLongValue,pFloatValue,pDoubleValue,pCha
rValue);
        /* show point in the memory */
```


一. 基本数据类型

```
/* show the form of data type in the memory */
        printf("shortValue: \n");
        for(i=0;i<4;i++)
        {
                printf("shortValue%d:
%p\n",i,pShortValue);
                pShortValue++;
        }
        printf("intValue: \n");
        for(i=0;i<4;i++)
        {
                printf("intValue%d: %p\n",i,pIntValue);
                pIntValue++;
        }
        printf("longValue: \n");
        for(i=0;i<4;i++)
        {
                printf("longValue%d:
%p\n",i,pLongValue);
                pLongValue++;
        }
```


一. 基本数据类型

```
printf("floatValue: \n");
for(i=0;i<4;i++)
{
        printf("floatValue%d: %p\n",i,pFloatValue);
        pFloatValue++;
}
printf("doubleValue: \n");
for(i=0;i<4;i++)
{
        printf("doubleValue%d: %p\n",i,pDoubleValue);
        pDoubleValue++;
}
printf("charValue: \n");
for(i=0;i<4;i++)
{
        printf("charValue%d: %p\n",i,pCharValue);
        pCharValue++;
}
/* show data type in the memory */
system("pause");
return0;
}
```


一. 基本数据类型

- 在看相应的汇编代码之前，先想一想上面的代码会输出什么，一会比较一下，这与你想的是否一致
- 自己测试编译时选择debug版本，不选release版本
  - release是经过编译器优化的代码，初学时，看debug版本才能看到原汁原味的源代码编译链接成的程序

一. 基本数据类型

- 先看指针

```
lea     edx, [ebp+var_8]
mov     [ebp+var_64], edx
lea     eax, [ebp+var_18]
mov     [ebp+var_68], eax
lea     ecx, [ebp+var_28]
mov     [ebp+var_6C], ecx
lea     edx, [ebp+var_38]
mov     [ebp+var_70], edx
lea     eax, [ebp+var_58]
mov     [ebp+var_74], eax
lea     ecx, [ebp+var_5C]
mov     [ebp+var_78], ecx
mov     edx, [ebp+var_78]
push    edx
mov     eax, [ebp+var_74]
push    eax
mov     ecx, [ebp+var_70]
push    ecx
mov     edx, [ebp+var_6C]
push    edx
mov     eax, [ebp+var_68]
push    eax
mov     ecx, [ebp+var_64]
push    ecx
push    offset aPshortvaluePPi ; "pShortValue: %p\npIntValue: %p\npLongVa"...
```

pShortValue = shortValue

打印指针pShortValue的值

一. 基本数据类型

- 例
  - pShortValue指针
  - shortValue数组是存储在栈上
  - 上图中[ebp+var_8]，[ebp+var_64]是一个指针，存储的是shortValue数组的地址，指向shortValue数组。

一. 基本数据类型

- 打印的部分：

```
mov     eax, [ebp+var_64]
push    eax
mov     ecx, [ebp+var_60]
push    ecx
push    offset aShortvalueDP ; "shortValue%d: %p\n"
call    _printf
add     esp, 0Ch
mov     edx, [ebp+var_64]
add     edx, 2
mov     [ebp+var_64], edx
jmp     short loc_4010F9
```


一. 基本数据类型

- 取出了指针所指向的地址，然后依次打印出了数组的内容，其中指针的自加，即**add eax,2**，因为是**short**类型，一个数占用两个字节。
- 注意看看**double**指针和**int**指针的自加。

一. 基本数据类型

```
mov     ecx, [ebp+var_74]
push    ecx
mov     edx, [ebp+var_60]
push    edx
push    offset aDoublevalueDP ; "doubleValue%d: %p\n"
call    _printf
add     esp, 0Ch
mov     eax, [ebp+var_74]
add     eax, 8
mov     [ebp+var_74], eax
jmp     short loc_40120D
```

```
mov     ecx, [ebp+var_68]
push    ecx
mov     edx, [ebp+var_60]
push    edx
push    offset aIntvalueDP ; "intValue%d: %p\n"
call    _printf
add     esp, 0Ch
mov     eax, [ebp+var_68]
add     eax, 4
mov     [ebp+var_68], eax
jmp     short loc_40113E
```


一. 基本数据类型

- 同样是指针的自加，不同的数据类型在汇编中的表现形式就十分不同，而且在内存中，均是以字节为单位进行运算。

一. 基本数据类型

- 整型数和浮点数在内存中表示：

```
mov     [ebp+var_18], 0C8h
xor     ecx, ecx
mov     [ebp+var_14], ecx
mov     [ebp+var_10], ecx
mov     [ebp+var_C], ecx
mov     [ebp+var_28], 12Ch
xor     edx, edx
mov     [ebp+var_24], edx
mov     [ebp+var_20], edx
mov     [ebp+var_1C], edx
mov     [ebp+var_38], 43C80CCDh
xor     eax, eax
mov     [ebp+var_34], eax
mov     [ebp+var_30], eax
mov     [ebp+var_2C], eax
mov     [ebp+var_58], 0EB851EB8h
mov     [ebp+var_54], 407F4051h
```

整形数

浮点数

一. 基本数据类型

- 浮点数运算
  - x86使用的是浮点寄存器，Intel提供了8个128位的寄存器，xmm0~xmm7，每一个寄存器可以存放4个(32位)单精度的浮点数。

一. 基本数据类型

- 以浮点数400.1和500.02为例
  - 这两个数在内存中的十六进制表现形式分别为0x43C80CCD和0x407F4051EB851EB8
  - float类型在内存中占4字节，需要经过IEEE编码。编码方式如下：最高位用于表示符号，剩余31位中，8位用于表示指数，其余用于表示尾数，

第三章从C语言看汇编

- 一. 基本数据类型
- 二. 流程控制语句
- 三. 变量表现形式

二. 流程控制语句

- 流程控制语句在汇编中的表现形式
  - C语言基本的选择，循环等流程控制块在汇编中的表现形式

二. 流程控制语句

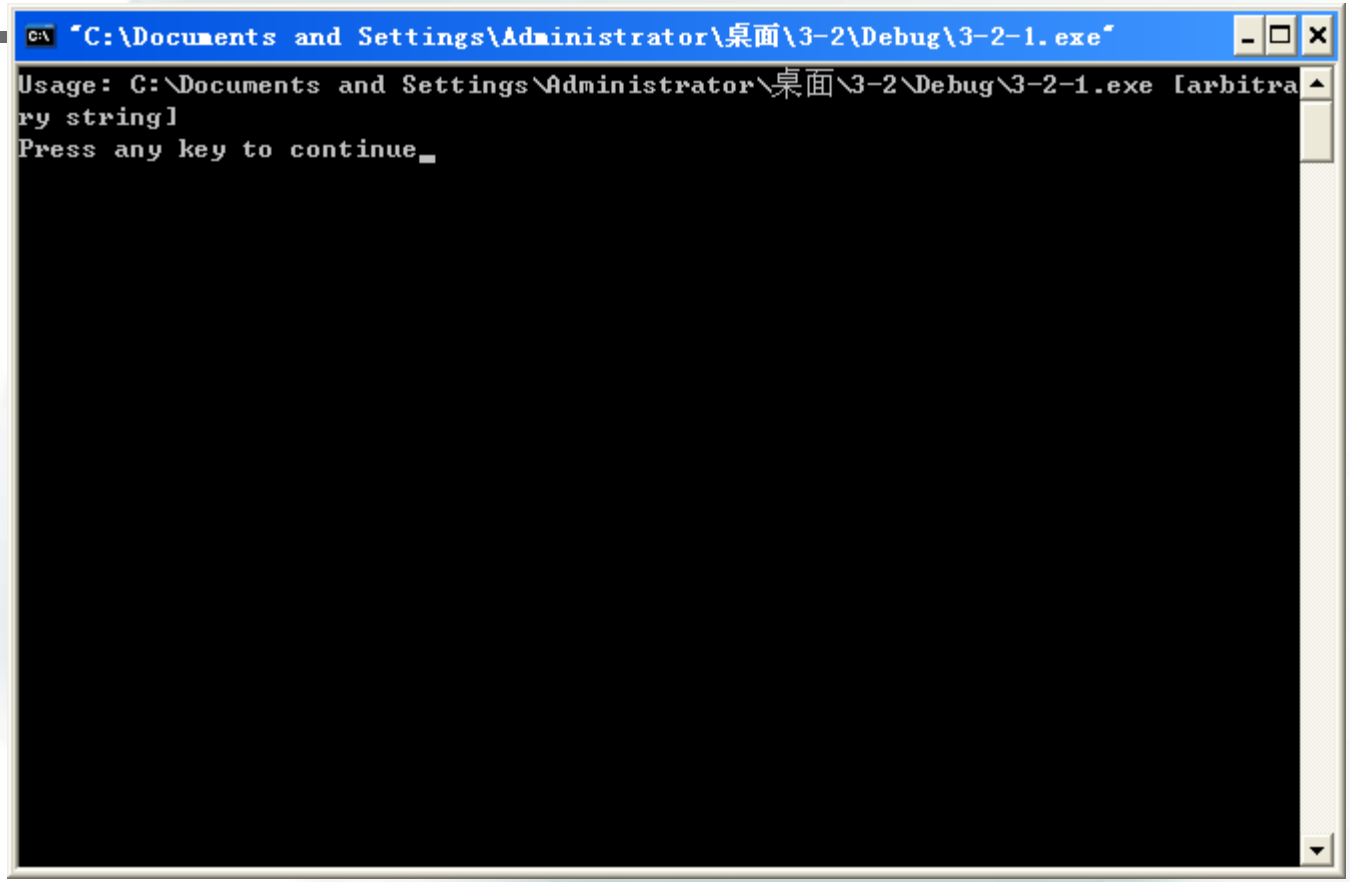
- （1）if,elseif,else选择控制块
- （2）switch case选择控制块
- （3） while/for/do循环控制块

二. 流程控制语句

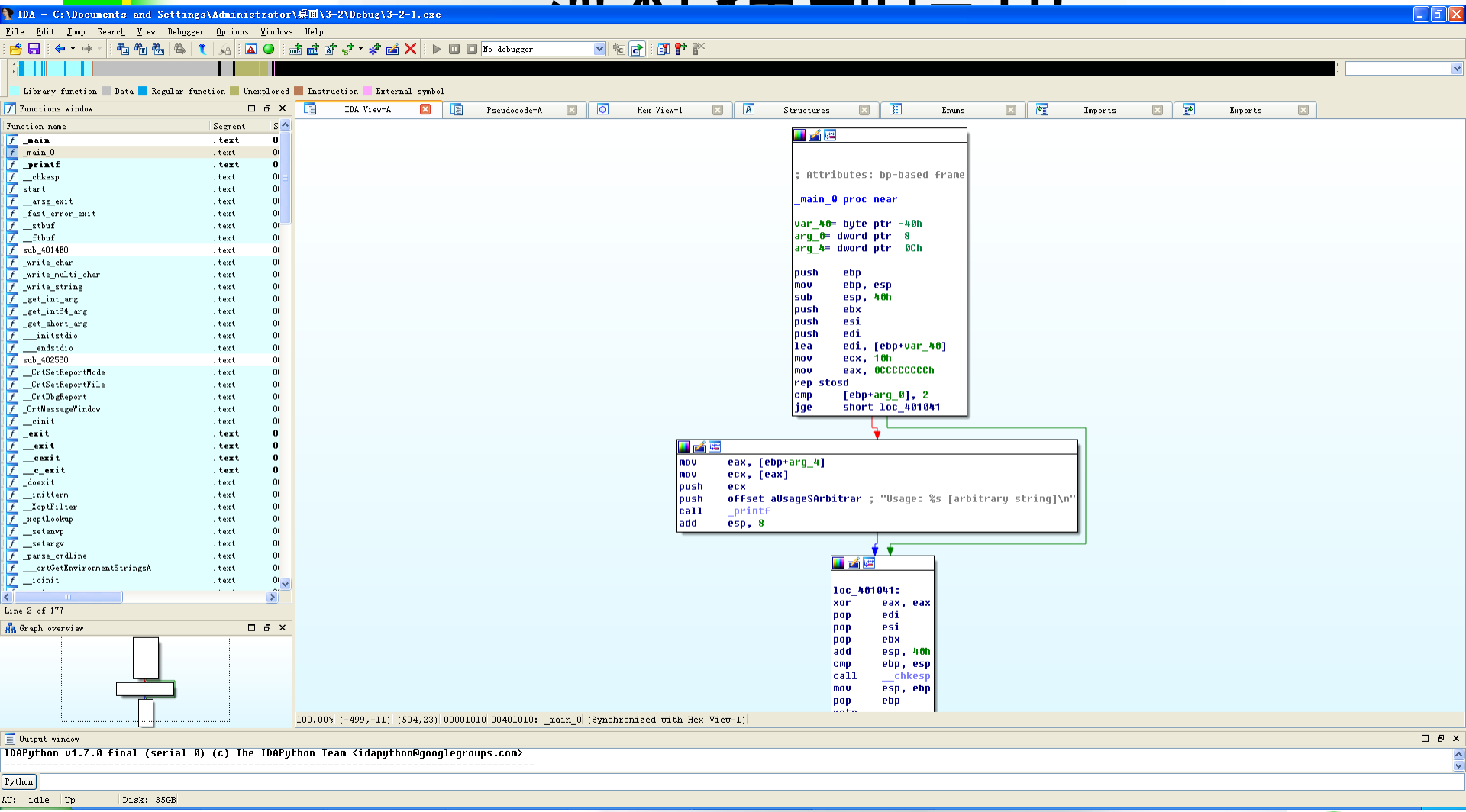
- if语句 (vc++ 6.0 debug版本)

```
Int main(int argc,char *argv[])
{
      if(argc<2)
      {
            printf("Usage: %s [arbitrary
string]\n",argv[0]);
      }
      return 0;
}
```


二. 流程控制语句

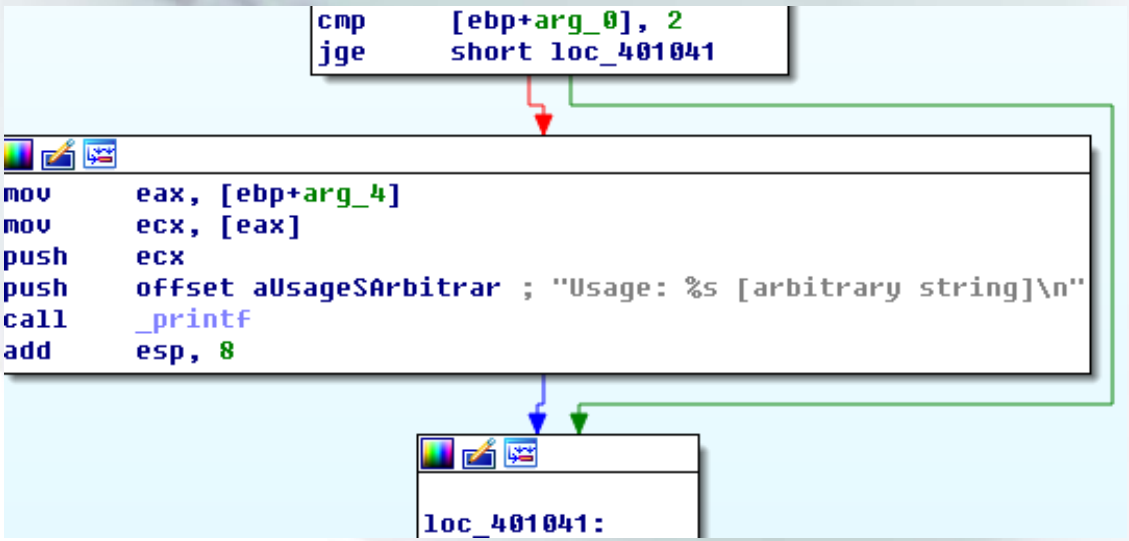

"C:\Documents and Settings\Administrator\桌面\3-2\Debug\3-2-1.exe"
Usage: C:\Documents and Settings\Administrator\桌面\3-2\Debug\3-2-1.exe [arbitrary string]
Press any key to continue

一 流程控制语句


IDA - C:\Documents and Settings\Administrator\桌面\3-2\Debug\3-2-1.exe
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Data Regular function Unexplored Instruction External symbol
Functions window
Function name Segment
_main .text
_main_0 .text
_printf .text
__chkesp .text
start .text
__amsg_exit .text
_fast_error_exit .text
__stbuf .text
__ftbuf .text
sub_4014E0 .text
_write_char .text
_write_multi_char .text
_write_string .text
_get_int_arg .text
_get_int64_arg .text
_get_short_arg .text
___initstdio .text
___endstdio .text
sub_402560 .text
__CrtSetReportMode .text
__CrtSetReportFile .text
__CrtDbgReport .text
_CrtMessageWindow .text
__cinit .text
_exit .text
__exit .text
__cexit .text
__c_exit .text
_doexit .text
__initterm .text
__XcptFilter .text
_xcptlookup .text
__setenvp .text
__setargv .text
_parse_cmdline .text
___crtGetEnvironmentStringsA .text
__ioinit .text
Line 2 of 177
Graph overview
IDA View-A
Pseudocode-A
Hex View-1
Structures
Enums
Imports
Exports
; Attributes: bp-based frame
_main_0 proc near
var_40= byte ptr -40h
arg_0= dword ptr 8
arg_4= dword ptr 0Ch
push ebp
mov ebp, esp
sub esp, 40h
push ebx
push esi
push edi
lea edi, [ebp+var_40]
mov ecx, 10h
mov eax, 0CCCCCCCCh
rep stosd
cmp [ebp+arg_0], 2
jge short loc_401041
mov eax, [ebp+arg_4]
mov ecx, [eax]
push ecx
push offset aUsageSArbitrar ; "Usage: %s [arbitrary string]\n"
call _printf
add esp, 8
loc_401041:
xor eax, eax
pop edi
pop esi
pop ebx
add esp, 40h
cmp ebp, esp
call __chkesp
mov esp, ebp
pop ebp
100.00% (-499,-11) (504,23) 00001010 00401010: _main_0 (Synchronized with Hex View-1)
Output window
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
Python
AU: idle Up Disk: 35GB

二. 流程控制语句

- 可以通过ida清楚地看到if语句控制块的模样，if argc< 2，在汇编中是一句jge short loc_401041代码，如果大于等于2，则跳出，小于，执行printf函数。

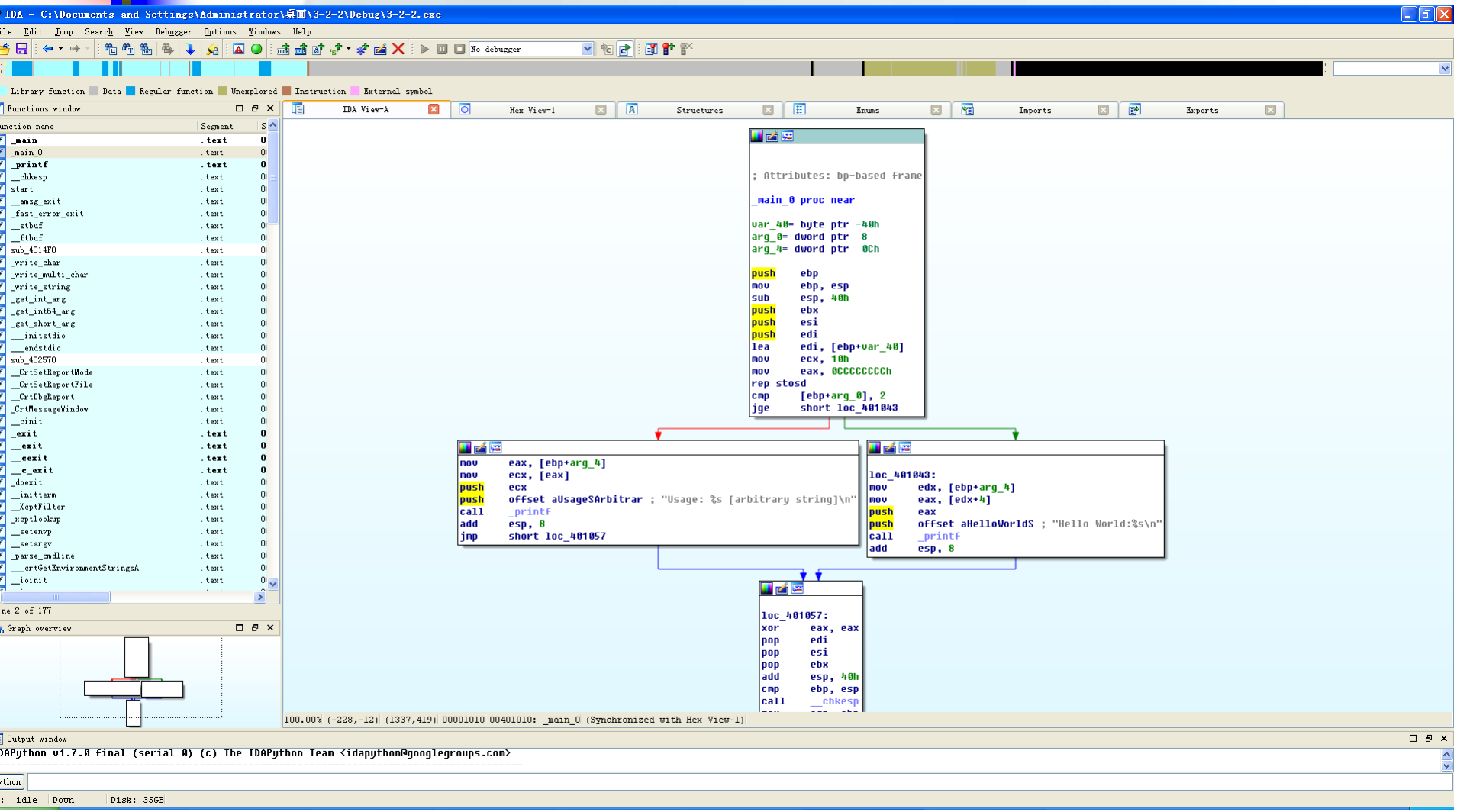


二. 流程控制语句

if else控制块 (vc++ 6.0 debug版本)

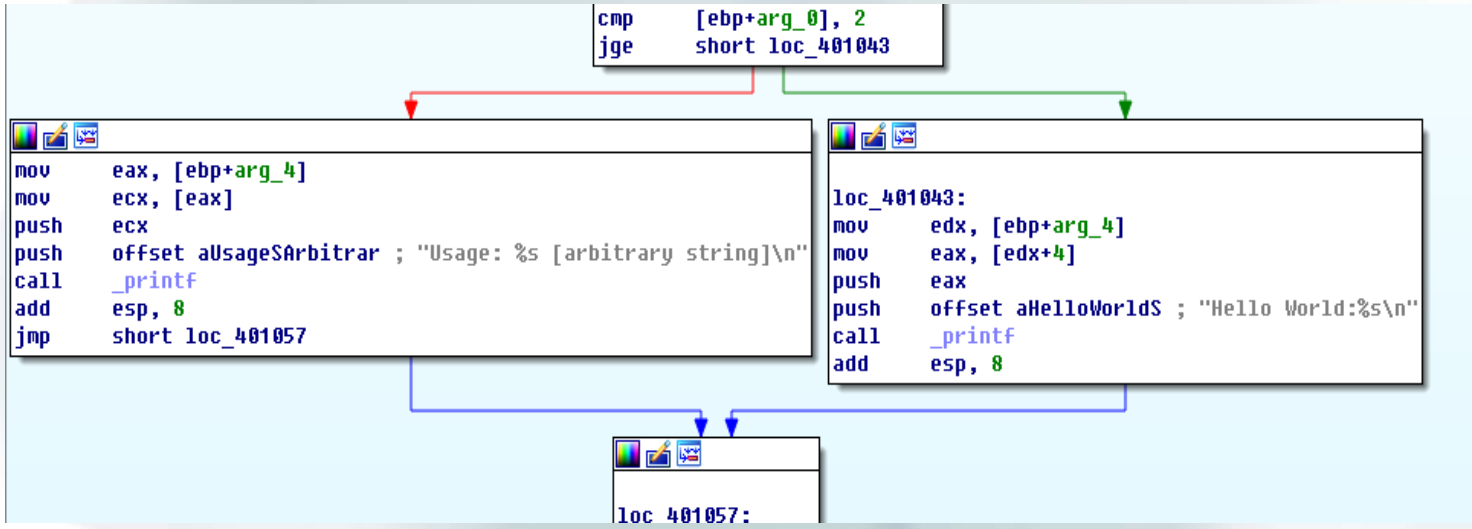
```
Int main(int argc,char *argv[])
{
	if(argc<2)
	{
		printf("Usage: %s [arbitrary string]\n",argv[0]);
	}
	else
	{
		printf("Hello World:%s\n",argv[1]);
	}
	return0;
}
```


二. 流程控制语句


IDA - C:\Documents and Settings\Administrator\桌面\3-2-2\Debug\3-2-2.exe
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Data Regular function Unexplored Instruction External symbol
Functions window
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
Graph overview
; Attributes: bp-based frame
_main_0 proc near
var_40= byte ptr -40h
arg_0= dword ptr 8
arg_4= dword ptr 0Ch
push ebp
mov ebp, esp
sub esp, 40h
push ebx
push esi
push edi
lea edi, [ebp+var_40]
mov ecx, 10h
mov eax, 0CCCCCCCCh
rep stosd
cmp [ebp+arg_0], 2
jge short loc_401043
mov eax, [ebp+arg_4]
mov ecx, [eax]
push ecx
push offset aUsageSArbitrar ; "Usage: %s [arbitrary string]\n"
call _printf
add esp, 8
jmp short loc_401057
loc_401043:
mov edx, [ebp+arg_4]
mov eax, [edx+4]
push eax
push offset aHelloWorldS ; "Hello World:%s\n"
call _printf
add esp, 8
loc_401057:
xor eax, eax
pop edi
pop esi
pop ebx
add esp, 40h
cmp ebp, esp
call __chkesp
Output window
DAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
idle Down Disk: 35GB

二. 流程控制语句

- 用ida打开，进行分析



二. 流程控制语句

- 和刚刚的if控制块相比，if else控制块多出来了一个分支，使得cmp之后，必须选择其中之一进行执行，而且也没有多余的分支可以选择
  - 大于等于2选择右边的基础块
  - 小于2选择左边的基础块

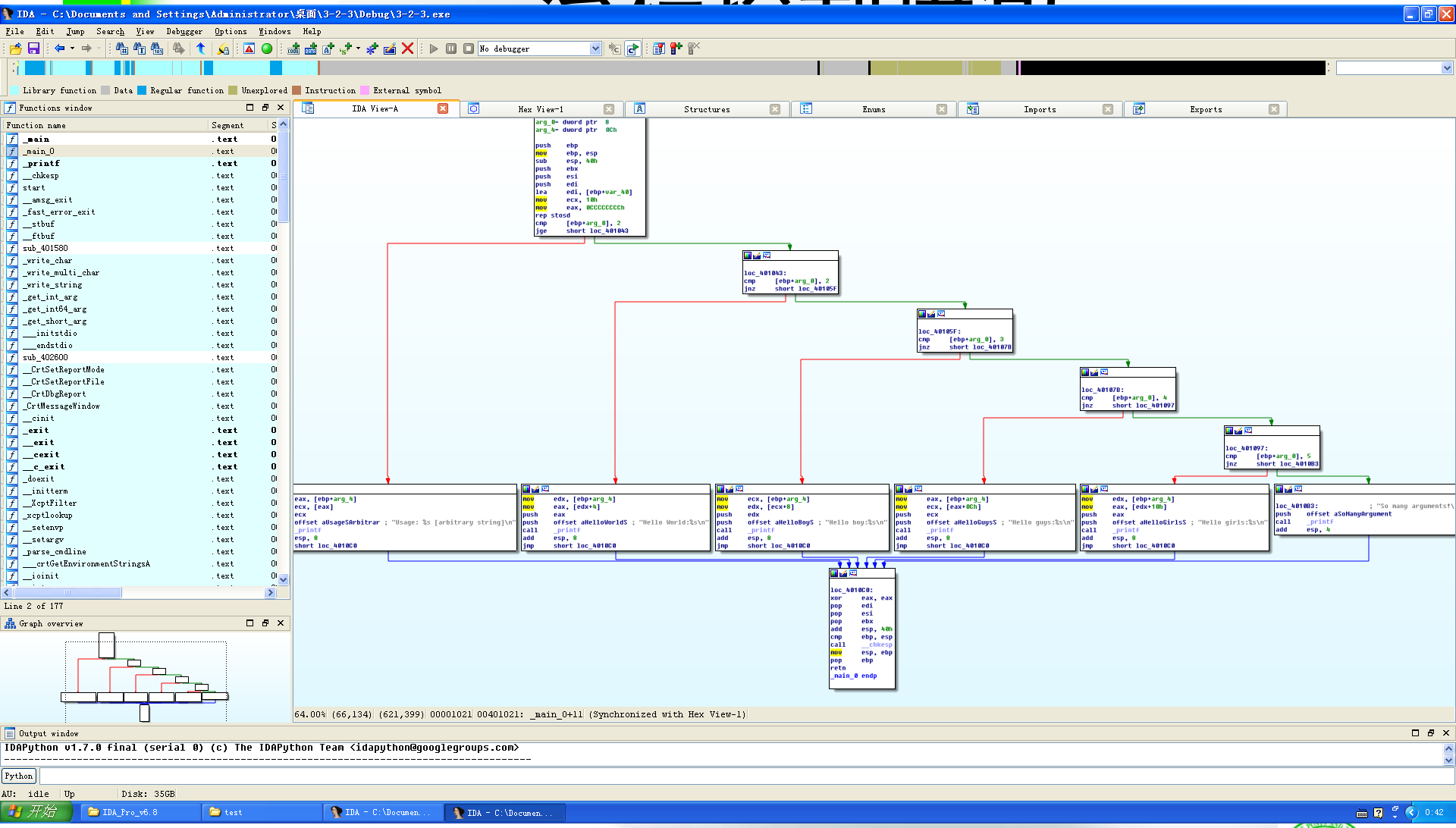
二. 流程控制语句

- if elseif else控制块（vc++ 6.0 debug版本）

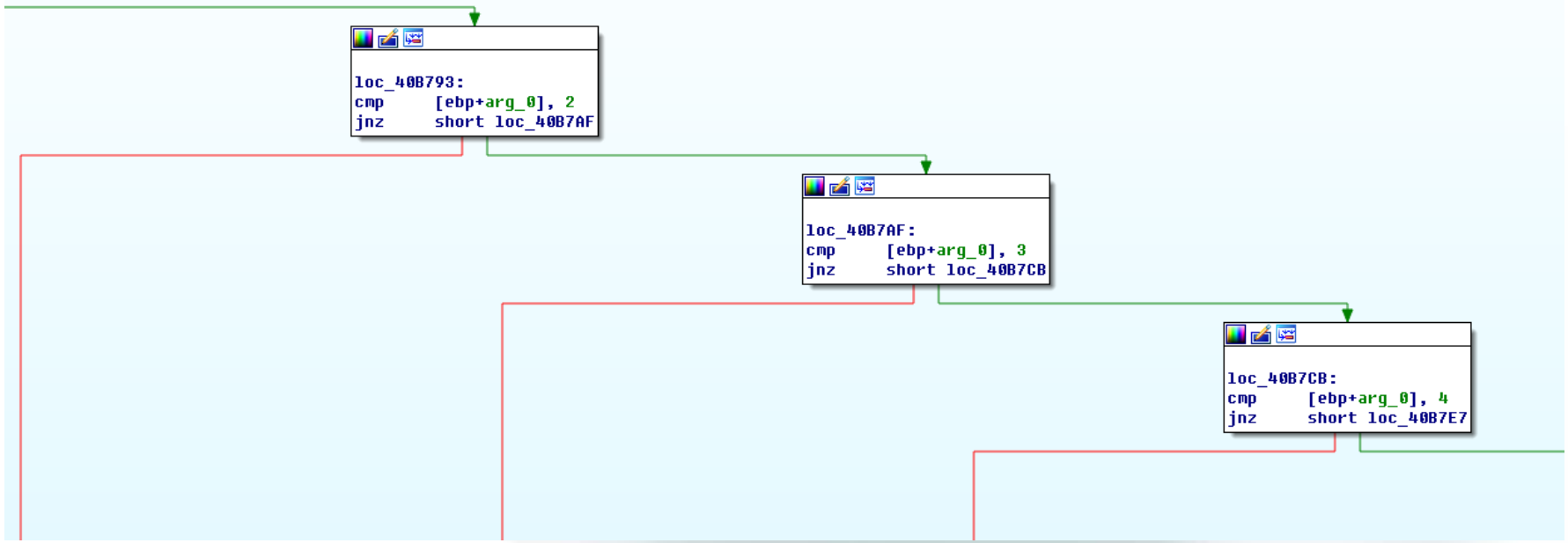
二. 流程控制语句

```
Int main(int argc,char *argv[])
{
        if(argc<2)
        {
                printf("Usage: %s [arbitrary string]\n",argv[0]);
        }
        elseif(argc==2)
        {
                printf("Hello World:%s\n",argv[1]);
        }
        elseif(argc==3)
        {
                printf("Hello boy:%s\n",argv[2]);
        }
        elseif(argc==4)
        {
                printf("Hello guys:%s\n",argv[3]);
        }
        elseif(argc==5)
        {
                printf("Hello girls:%s\n",argv[4]);
        }
        else
        {
                printf("So many arguments!\n");
        }
        return0;
}
```


一、流程控制语句


IDA - C:\Documents and Settings\Administrator\桌面\3-2-3\Debug\3-2-3.exe
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Data Regular function Unexplored Instruction External symbol
Functions window
Function name Segment
_main .text
_main_0 .text
_printf .text
__chkesp .text
start .text
__amsg_exit .text
_fast_error_exit .text
__stbuf .text
__ftbuf .text
sub_401580 .text
_write_char .text
_write_multi_char .text
_write_string .text
_get_int_arg .text
_get_int64_arg .text
_get_short_arg .text
___initstdio .text
___endstdio .text
sub_402600 .text
__CrtSetReportMode .text
__CrtSetReportFile .text
__CrtDbgReport .text
_CrtMessageWindow .text
__cinit .text
_exit .text
__exit .text
__cexit .text
__c_exit .text
_doexit .text
__initterm .text
__XcptFilter .text
_xcptlookup .text
__setenvp .text
__setargv .text
_parse_cmdline .text
___crtGetEnvironmentStringsA .text
__ioinit .text
Line 2 of 177
Graph overview
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
arg_0= dword ptr 8
arg_4= dword ptr 0Ch
push ebp
mov ebp, esp
sub esp, 40h
push ebx
push esi
push edi
lea edi, [ebp+var_40]
mov ecx, 10h
mov eax, 0CCCCCCCCh
rep stosd
cmp [ebp+arg_0], 2
jge short loc_401043
loc_401043:
cmp [ebp+arg_0], 2
jnz short loc_40105F
loc_40105F:
cmp [ebp+arg_0], 3
jnz short loc_401078
loc_401078:
cmp [ebp+arg_0], 4
jnz short loc_401097
loc_401097:
cmp [ebp+arg_0], 5
jnz short loc_4010B3
mov eax, [ebp+arg_4]
mov ecx, [eax]
push ecx
offset aUsageSArbitrar ; "Usage: %s [arbitrary string]\n"
_printf
esp, 8
short loc_4010C0
mov edx, [ebp+arg_4]
mov eax, [edx+4]
push eax
offset aHelloWorldS ; "Hello World:%s\n"
_printf
esp, 8
jmp short loc_4010C0
mov ecx, [ebp+arg_4]
mov edx, [ecx+8]
push edx
offset aHelloBoyS ; "Hello boy:%s\n"
_printf
esp, 8
jmp short loc_4010C0
mov eax, [ebp+arg_4]
mov ecx, [eax+0Ch]
push ecx
offset aHelloGuysS ; "Hello guys:%s\n"
_printf
esp, 8
jmp short loc_4010C0
mov edx, [ebp+arg_4]
mov eax, [edx+10h]
push eax
offset aHelloGirlsS ; "Hello girls:%s\n"
_printf
esp, 8
jmp short loc_4010C0
loc_4010B3: ; "So many arguments!\n"
push offset aSoManyArgument
call _printf
add esp, 4
loc_4010C0:
xor eax, eax
pop edi
pop esi
pop ebx
add esp, 40h
cmp ebp, esp
call __chkesp
mov esp, ebp
pop ebp
retn
_main_0 endp
64.00% (66,134) (621,399) 00001021 00401021: _main_0+11 (Synchronized with Hex View-1)
Output window
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
Python
AU: idle Up Disk: 35GB
开始
IDA_Pro_v6.8
test
IDA - C:\Documen...
IDA - C:\Documen...
0:42

二. 流程控制语句

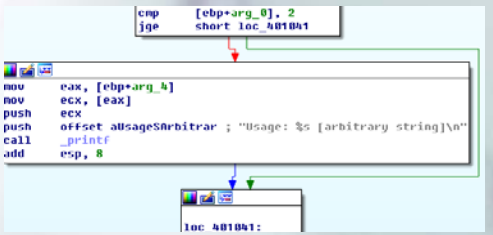

loc_40B793:
cmp [ebp+arg_0], 2
jnz short loc_40B7AF
loc_40B7AF:
cmp [ebp+arg_0], 3
jnz short loc_40B7CB
loc_40B7CB:
cmp [ebp+arg_0], 4
jnz short loc_40B7E7

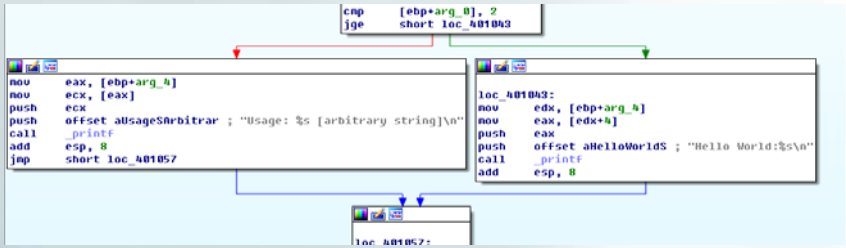
二. 流程控制语句

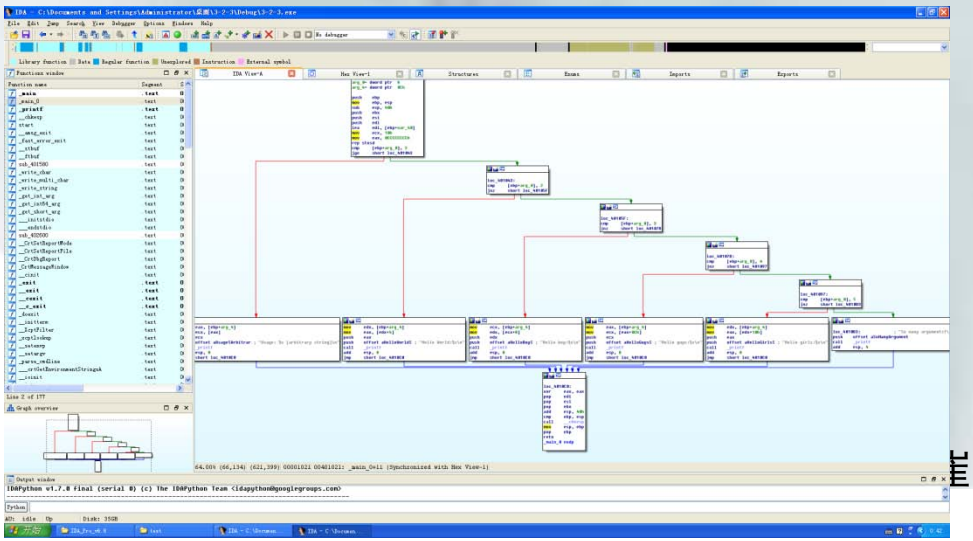
- 通过if else if 控制块全貌图，可以明显地看到，程序是先判断一个分支，然后依次判断下一个else if分支，这样依次下去，判断语句做为其中的一条分支。

二. 流程控制语句

- 通过if，if else，if elseif else控制块，可以从ida的图形上清楚地看到如何去识别一个选择流程控制块，然后转换为高级语言。







崔宝江

二. 流程控制语句

- （1）if,elseif,else选择控制块
- （2）switch case选择控制块
- （3） while/for/do循环控制块

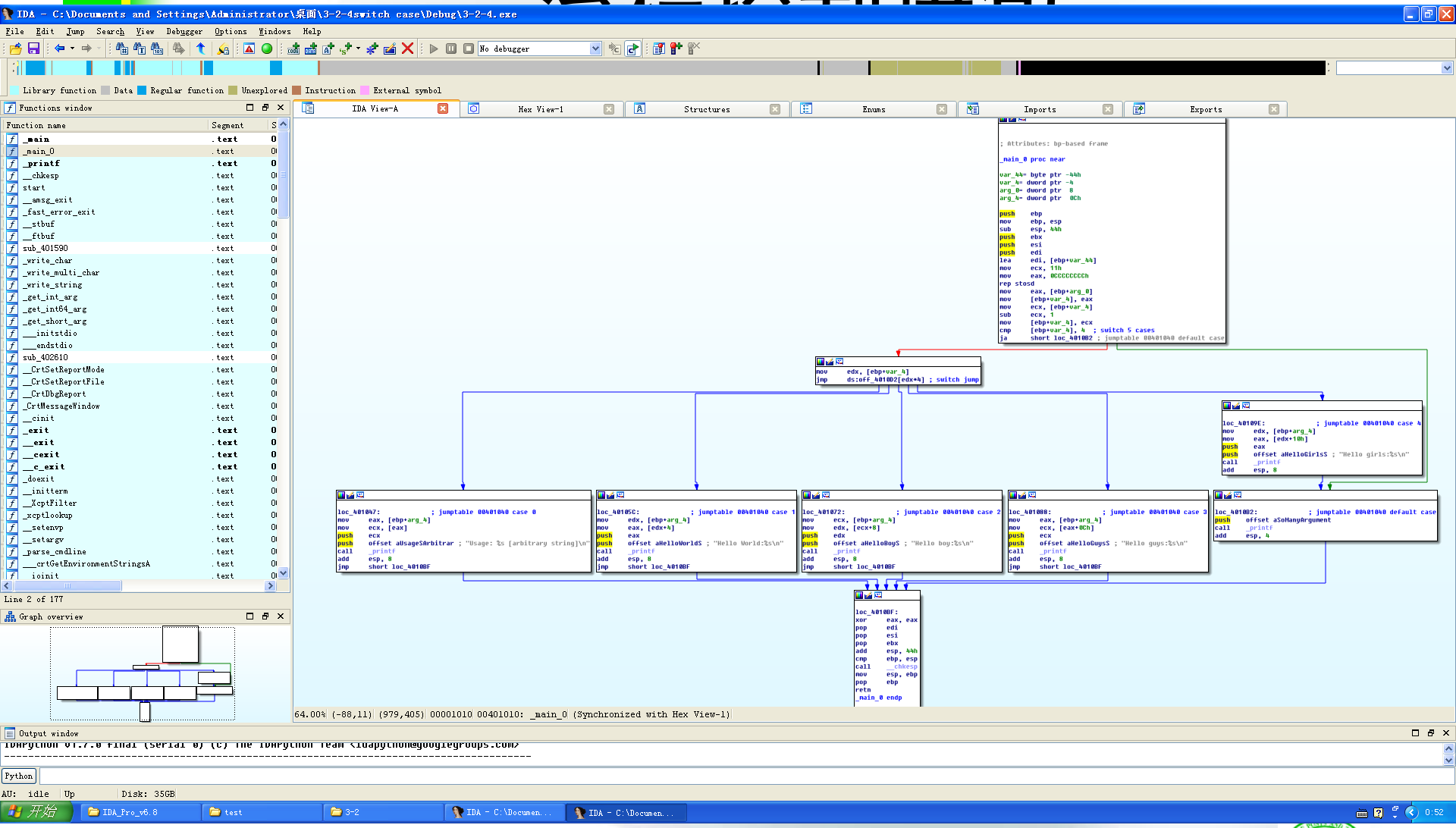
二. 流程控制语句

- switch case控制块 (vc++ 6.0 debug版本)

二. 流程控制语句

```
Int main(int argc,char *argv[])
{
        switch(argc)
        {
                case1:
                        printf("Usage: %s [arbitrary string]\n",argv[0]);
                        break;
                case2:
                        printf("Hello World:%s\n",argv[1]);
                        break;
                case3:
                        printf("Hello boy:%s\n",argv[2]);
                        break;
                case4:
                        printf("Hello guys:%s\n",argv[3]);
                        break;
                case5:
                        printf("Hello girls:%s\n",argv[4]);
                default:
                        printf("So many arguments!\n");
        };
        return0;
}
```


一、流程控制语句

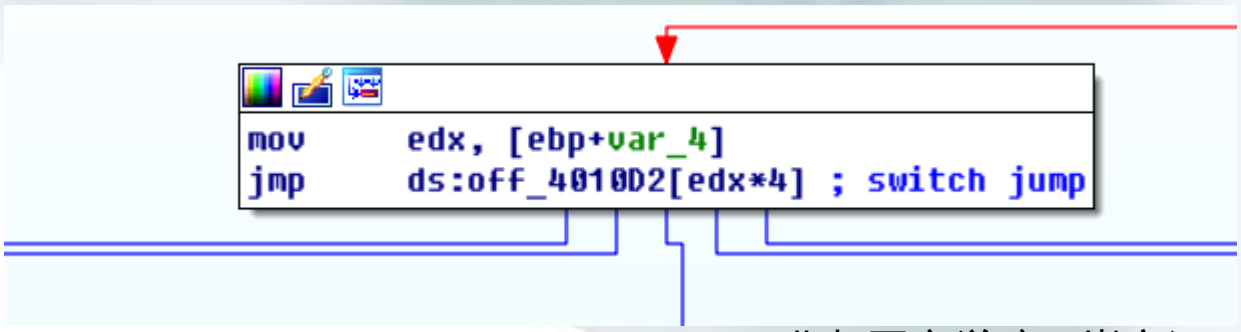

IDA - C:\Documents and Settings\Administrator\桌面\3-2-4switch case\Debug\3-2-4.exe
File Edit Jump Search View Debugger Options Windows Help
Library function Data Regular function Unexplored Instruction External symbol
Functions window
Function name Segment
_main .text
_main_0 .text
_printf .text
__chkesp .text
start .text
__amsg_exit .text
_fast_error_exit .text
__stbuf .text
__ftbuf .text
sub_401590 .text
_write_char .text
_write_multi_char .text
_write_string .text
_get_int_arg .text
_get_int64_arg .text
_get_short_arg .text
___initstdio .text
___endstdio .text
sub_402610 .text
__CrtSetReportMode .text
__CrtSetReportFile .text
__CrtDbgReport .text
_CrtMessageWindow .text
__cinit .text
_exit .text
__exit .text
__cexit .text
__c_exit .text
_doexit .text
__initterm .text
_XcptFilter .text
_xcptlookup .text
__setenvp .text
__setargv .text
_parse_cmdline .text
___crtGetEnvironmentStringsA .text
__ioinit .text
Line 2 of 177
Graph overview
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
; Attributes: bp-based frame
_main_0 proc near
var_44= byte ptr -44h
var_4= dword ptr -4
arg_0= dword ptr 8
arg_4= dword ptr 0Ch
push ebp
mov ebp, esp
sub esp, 44h
push ebx
push esi
push edi
lea edi, [ebp+var_44]
mov ecx, 11h
mov eax, 0CCCCCCCCh
rep stosd
mov eax, [ebp+arg_0]
mov [ebp+var_4], eax
mov ecx, [ebp+var_4]
sub ecx, 1
mov [ebp+var_4], ecx
cmp [ebp+var_4], 4 ; switch 5 cases
ja short loc_4010B2 ; jumptable 004010A0 default case
mov edx, [ebp+var_4]
jmp ds:off_4010D2[edx*4] ; switch jump
loc_401047: ; jumptable 004010A0 case 0
mov eax, [ebp+arg_4]
mov ecx, [eax]
push ecx
push offset aUsageSArbitrar ; "Usage: %s [arbitrary string]\n"
call _printf
add esp, 8
jmp short loc_4010BF
loc_40105C: ; jumptable 004010A0 case 1
mov edx, [ebp+arg_4]
mov eax, [edx+4]
push eax
push offset aHelloWorldS ; "Hello World:%s\n"
call _printf
add esp, 8
jmp short loc_4010BF
loc_401072: ; jumptable 004010A0 case 2
mov ecx, [ebp+arg_4]
mov edx, [ecx+8]
push edx
push offset aHelloBoyS ; "Hello boy:%s\n"
call _printf
add esp, 8
jmp short loc_4010BF
loc_401088: ; jumptable 004010A0 case 3
mov eax, [ebp+arg_4]
mov ecx, [eax+0Ch]
push ecx
push offset aHelloGuysS ; "Hello guys:%s\n"
call _printf
add esp, 8
jmp short loc_4010BF
loc_40109E: ; jumptable 004010A0 case 4
mov edx, [ebp+arg_4]
mov eax, [edx+10h]
push eax
push offset aHelloGirlsS ; "Hello girls:%s\n"
call _printf
add esp, 8
loc_4010B2: ; jumptable 004010A0 default case
push offset aSoManyArgument
call _printf
add esp, 4
loc_4010BF:
xor eax, eax
pop edi
pop esi
pop ebx
add esp, 44h
cmp ebp, esp
call __chkesp
mov esp, ebp
pop ebp
retn
_main_0 endp
64.00% (-88,11) (979,405) 00001010 00401010: _main_0 (Synchronized with Hex View-1)
Output window
Python
AU: idle Up Disk: 35GB
开始
IDA_Pro_v6.8
test
3-2
IDA - C:\Documen...
IDA - C:\Documen...
0:52

二. 流程控制语句

- 从控制流程图上将其与上一小节讲的**if else if**作一下比较
- **switch case**控制块的左边有点类似于分发器，由一个基本块来决定执行哪一块函数功能。

二. 流程控制语句

- 该基本块的汇编代码如下：
  - 将选择的序号值给了edx，根据edx，jmp到off_40B822这个table中的某个地址。
  - 这一处也就是和if else语句明显的不同之处，将要执行的地址存放到了一个数组里面，数组的每一个元素都是一个地址。

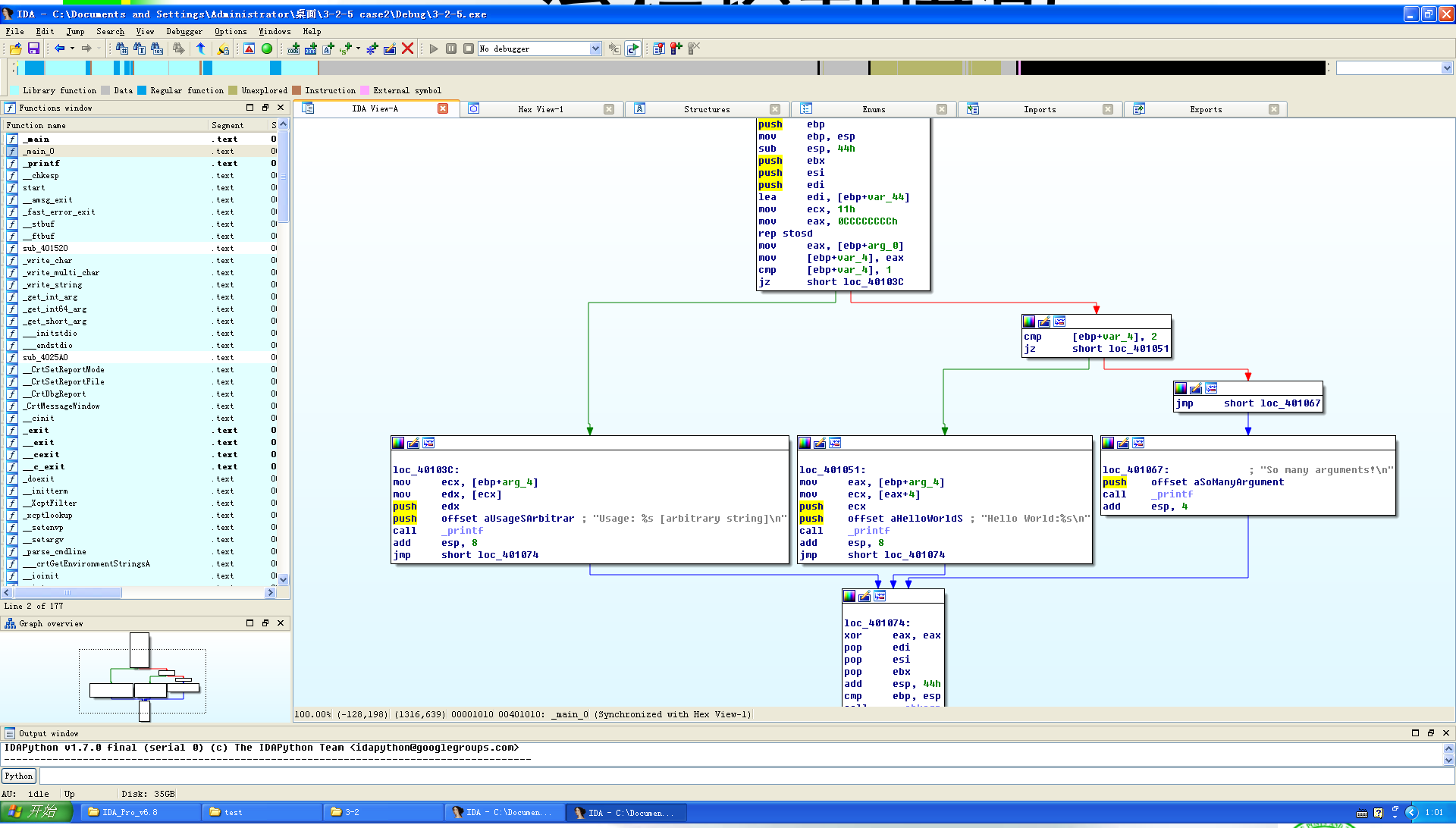


二. 流程控制语句

- 减少case分支，控制块全貌图又会发生什么变化

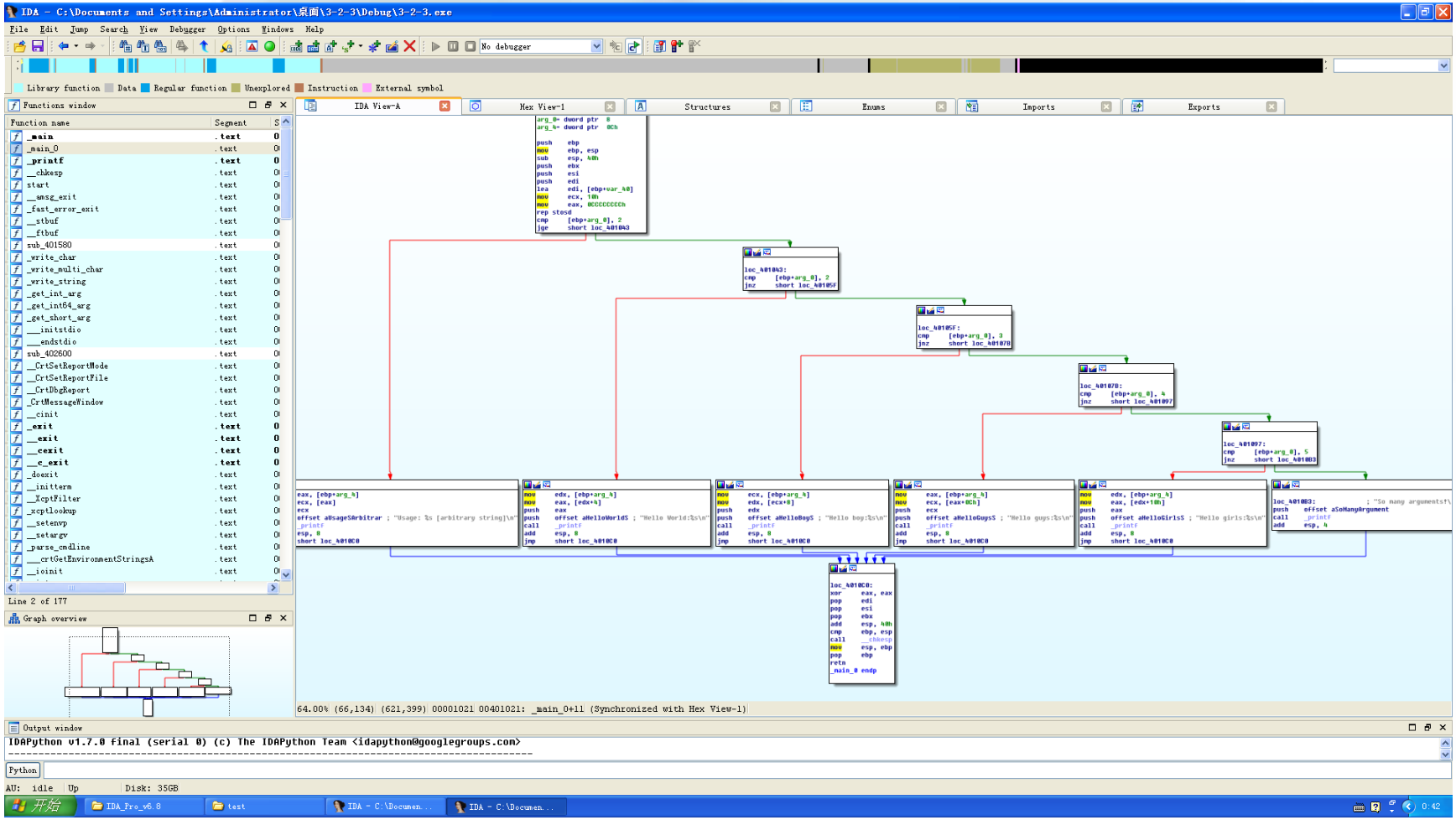
```
Int main(int argc,char *argv[])
{
        switch(argc)
        {
                case1:
                        printf("Usage: %s [arbitrary string]\n",argv[0]);
                        break;
                case2:
                        printf("Hello World:%s\n",argv[1]);
                        break;
                default:
                        printf("So many arguments!\n");
        };
        return0;
}
```


一、流程控制语句


IDA - C:\Documents and Settings\Administrator\桌面\3-2-5 case2\Debug\3-2-5.exe
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Data Regular function Unexplored Instruction External symbol
Functions window
Function name Segment
_main .text
_main_0 .text
_printf .text
__chkesp .text
start .text
__amsg_exit .text
_fast_error_exit .text
__stbuf .text
__ftbuf .text
sub_401520 .text
_write_char .text
_write_multi_char .text
_write_string .text
_get_int_arg .text
_get_int64_arg .text
_get_short_arg .text
___initstdio .text
___endstdio .text
sub_4025A0 .text
__CrtSetReportMode .text
__CrtSetReportFile .text
__CrtDbgReport .text
_CrtMessageWindow .text
__cinit .text
_exit .text
__exit .text
__cexit .text
__c_exit .text
_doexit .text
__initterm .text
__XcptFilter .text
_xcptlookup .text
__setenvp .text
__setargv .text
_parse_cmdline .text
___crtGetEnvironmentStringsA .text
__ioinit .text
Line 2 of 177
Graph overview
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
push ebp
mov ebp, esp
sub esp, 44h
push ebx
push esi
push edi
lea edi, [ebp+var_44]
mov ecx, 11h
mov eax, 0CCCCCCCCh
rep stosd
mov eax, [ebp+arg_0]
mov [ebp+var_4], eax
cmp [ebp+var_4], 1
jz short loc_40103C
cmp [ebp+var_4], 2
jz short loc_401051
jmp short loc_401067
loc_40103C:
mov ecx, [ebp+arg_4]
mov edx, [ecx]
push edx
push offset aUsageSArbitrar ; "Usage: %s [arbitrary string]\n"
call _printf
add esp, 8
jmp short loc_401074
loc_401051:
mov eax, [ebp+arg_4]
mov ecx, [eax+4]
push ecx
push offset aHelloWorldS ; "Hello World:%s\n"
call _printf
add esp, 8
jmp short loc_401074
loc_401067: ; "So many arguments!\n"
push offset aSoManyArgument
call _printf
add esp, 4
loc_401074:
xor eax, eax
pop edi
pop esi
pop ebx
add esp, 44h
cmp ebp, esp
100.00% (-128,198) (1316,639) 00001010 00401010: _main_0 (Synchronized with Hex View-1)
Output window
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
Python
AU: idle Up Disk: 35GB
开始
IDA_Pro_v6.8
test
3-2
IDA - C:\Documen...
IDA - C:\Documen...
1:01

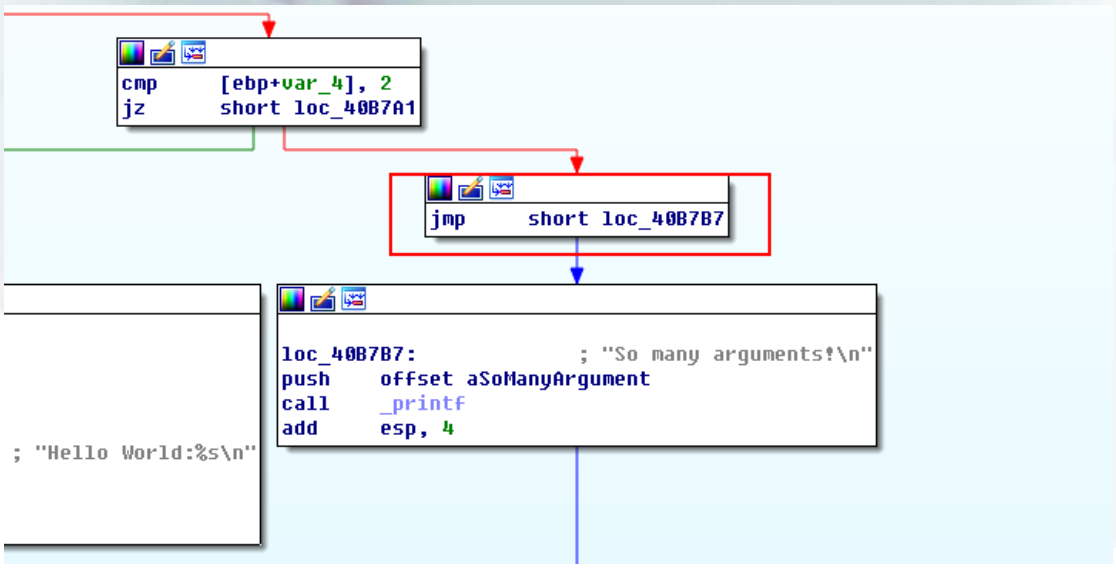
二. 流程控制语句

- 和if else if选择控制块流程相比较，二者十分相似的



二. 流程控制语句

- 除了在最后switch case选择控制块使用了default，使得程序有了一个单入单出的jmp指令
- if else if选择控制块，则没有这个单入单出的jmp指令基础块



二. 流程控制语句

- 前面的case值都是简单而且单增，线性的
- 如果改变其中一个case值为255，整个选择控制块结构又会发生什么变化呢？

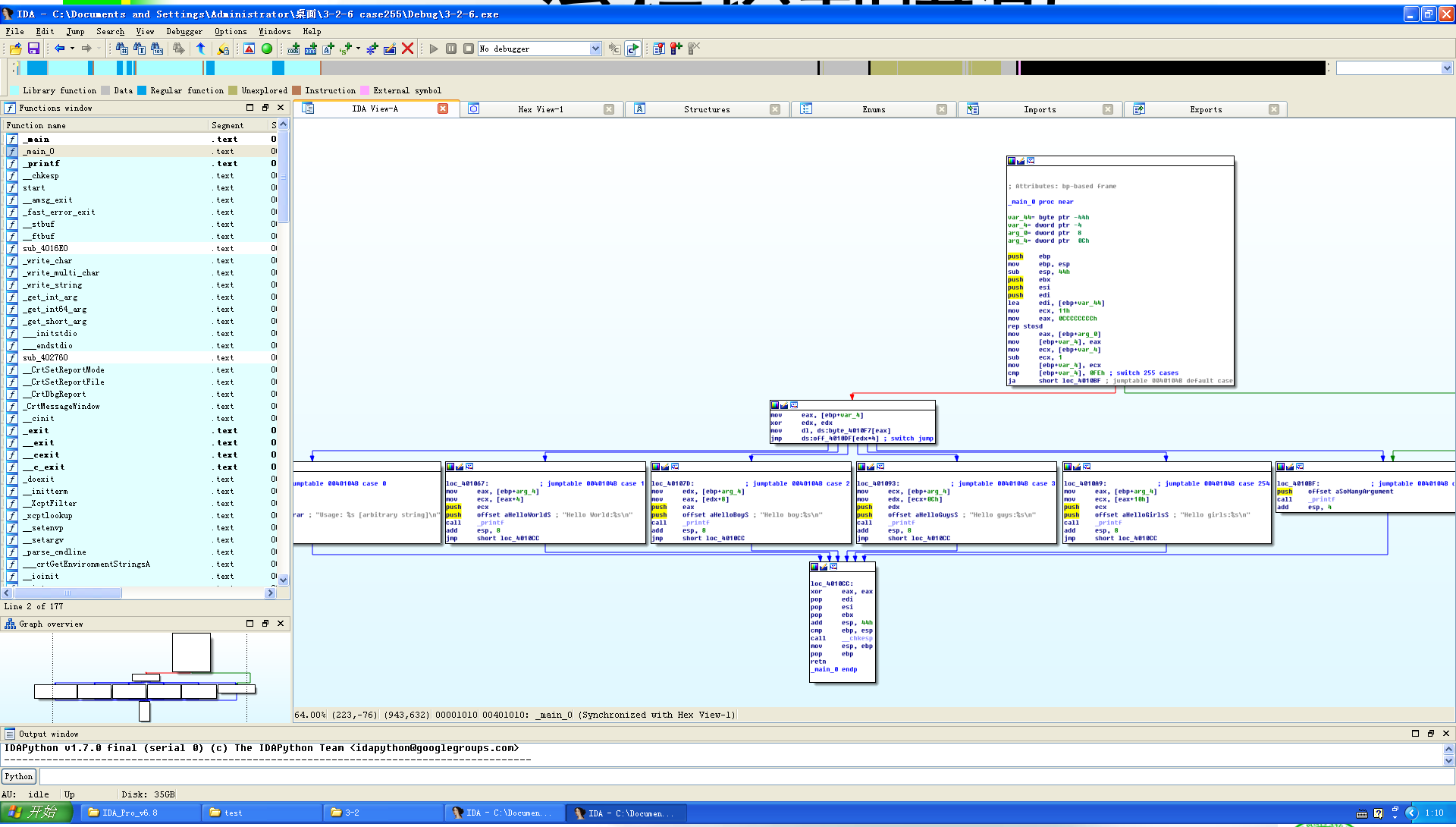
二. 流程控制语句

- 非线性case值 (vc++ 6.0 debug版本)

二. 流程控制语句

```
Int main(int argc,char *argv[])
{
        switch(argc)
        {
                case1:
                        printf("Usage: %s [arbitrary string]\n",argv[0]);
                        break;
                case2:
                        printf("Hello World:%s\n",argv[1]);
                        break;
                case3:
                        printf("Hello boy:%s\n",argv[2]);
                        break;
                case4:
                        printf("Hello guys:%s\n",argv[3]);
                        break;
                case255:
                        printf("Hello girls:%s\n",argv[4]);
                        break;
                default:
                        printf("So many arguments!\n");
        };
        return0;
}
```


一、流程控制语句

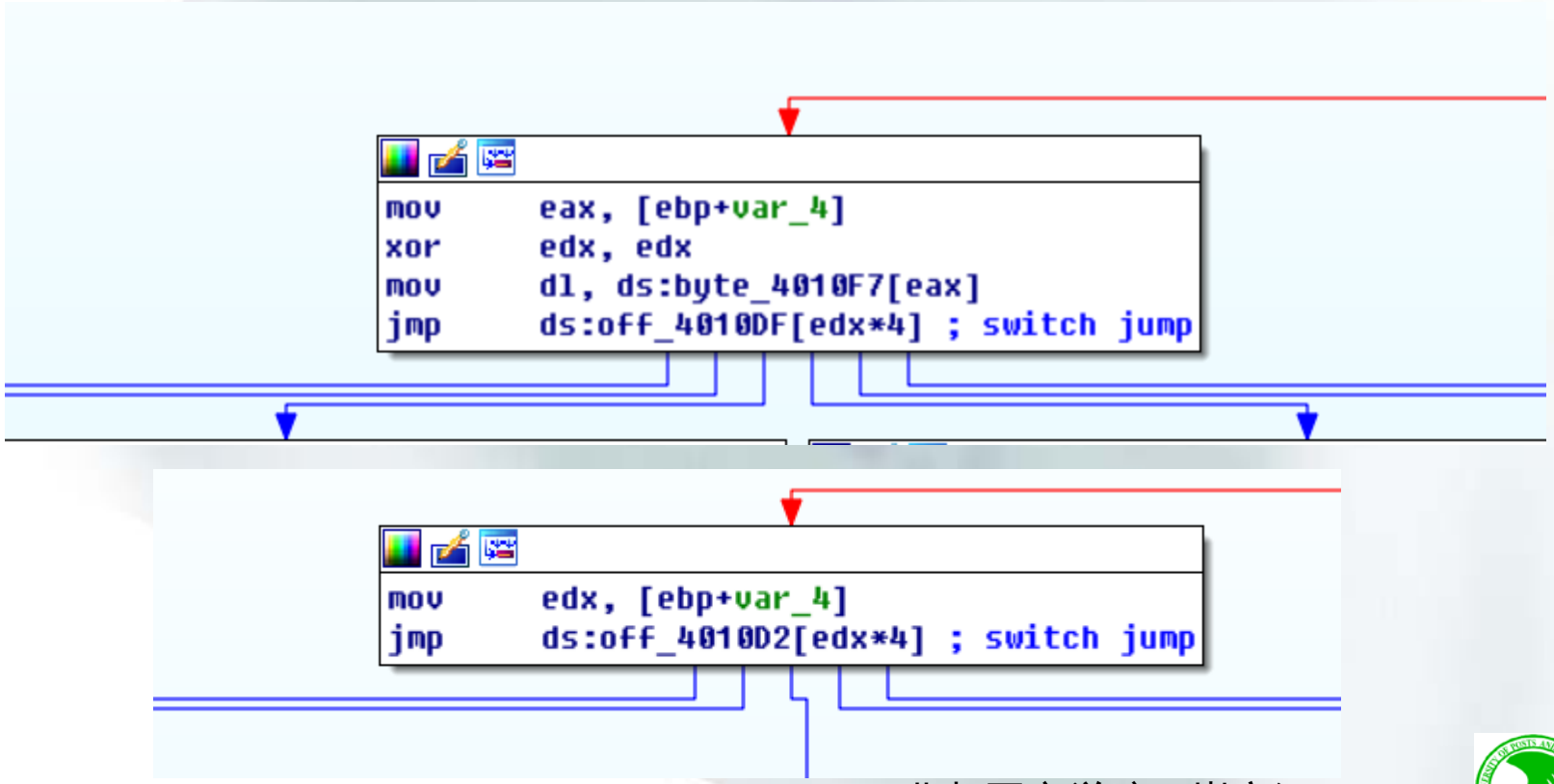

IDA - C:\Documents and Settings\Administrator\桌面\3-2-6 case255\Debug\3-2-6.exe
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Data Regular function Unexplored Instruction External symbol
Functions window
Function name Segment
_main .text
_main_0 .text
_printf .text
__chkesp .text
start .text
__amsg_exit .text
_fast_error_exit .text
__stbuf .text
__ftbuf .text
sub_4016E0 .text
_write_char .text
_write_multi_char .text
_write_string .text
_get_int_arg .text
_get_int64_arg .text
_get_short_arg .text
___initstdio .text
___endstdio .text
sub_402760 .text
__CrtSetReportMode .text
__CrtSetReportFile .text
__CrtDbgReport .text
_CrtMessageWindow .text
__cinit .text
_exit .text
__exit .text
__cexit .text
__c_exit .text
_doexit .text
__initterm .text
__XcptFilter .text
_xcptlookup .text
__setenvp .text
__setargv .text
_parse_cmdline .text
___crtGetEnvironmentStringsA .text
__ioinit .text
Line 2 of 177
Graph overview
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
; Attributes: bp-based frame
_main_0 proc near
var_44= byte ptr -44h
var_4= dword ptr -4
arg_0= dword ptr 8
arg_4= dword ptr 0Ch
push ebp
mov ebp, esp
sub esp, 44h
push ebx
push esi
push edi
lea edi, [ebp+var_44]
mov ecx, 11h
mov eax, 0CCCCCCCCh
rep stosd
mov eax, [ebp+arg_0]
mov [ebp+var_4], eax
mov ecx, [ebp+var_4]
sub ecx, 1
mov [ebp+var_4], ecx
cmp [ebp+var_4], 0FEh ; switch 255 cases
ja short loc_4010BF ; jumptable 004010A8 default case
mov eax, [ebp+var_4]
xor edx, edx
mov dl, ds:byte_4010F7[eax]
jmp ds:off_4010DF[edx*4] ; switch jump
loc_401067: ; jumptable 004010A8 case 1
mov eax, [ebp+arg_4]
mov ecx, [eax+4]
push ecx
push offset aHelloWorldS ; "Hello World:%s\n"
call _printf
add esp, 8
jmp short loc_4010CC
loc_401070: ; jumptable 004010A8 case 2
mov edx, [ebp+arg_4]
mov eax, [edx+8]
push eax
push offset aHelloBoyS ; "Hello boy:%s\n"
call _printf
add esp, 8
jmp short loc_4010CC
loc_401093: ; jumptable 004010A8 case 3
mov ecx, [ebp+arg_4]
mov edx, [ecx+0Ch]
push edx
push offset aHelloGuysS ; "Hello guys:%s\n"
call _printf
add esp, 8
jmp short loc_4010CC
loc_401099: ; jumptable 004010A8 case 254
mov eax, [ebp+arg_4]
mov ecx, [eax+10h]
push ecx
push offset aHelloGirlsS ; "Hello girls:%s\n"
call _printf
add esp, 8
jmp short loc_4010CC
loc_4010BF: ; jumptable 004010A8
push offset aSoManyArgument
call _printf
add esp, 4
loc_4010CC:
xor eax, eax
pop edi
pop esi
pop ebx
add esp, 44h
cmp ebp, esp
call __chkesp
mov esp, ebp
pop ebp
retn
_main_0 endp
64.00% (223,-76) (943,632) 00001010 00401010: _main_0 (Synchronized with Hex View-1)
Output window
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
Python
AU: idle Up Disk: 35GB
开始
IDA_Pro_v6.8
test
3-2
IDA - C:\Documen...
IDA - C:\Documen...
1:10

二. 流程控制语句

- 可以看到非线性的switch结构和之前线性的switch结构相比，从控制流结构上似乎并没有什么不同
- 都是由一个类似分发器的基础块根据我们的输入，jmp到相应的地址。

二. 流程控制语句

- 但是负责分发的那个基础块则有很大的不同



二. 流程控制语句

- 二者相比，多出来了一张表
  - 即一个数组(简称为索引表)

```
byte_4010F7     db 0, 1, 2, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                                        ; DATA XREF: _main_0+35↑r
                db 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                db 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                db 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                db 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                db 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                db 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                db 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                db 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                db 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                db 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                db 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                db 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                db 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
                db 5, 5, 4
```

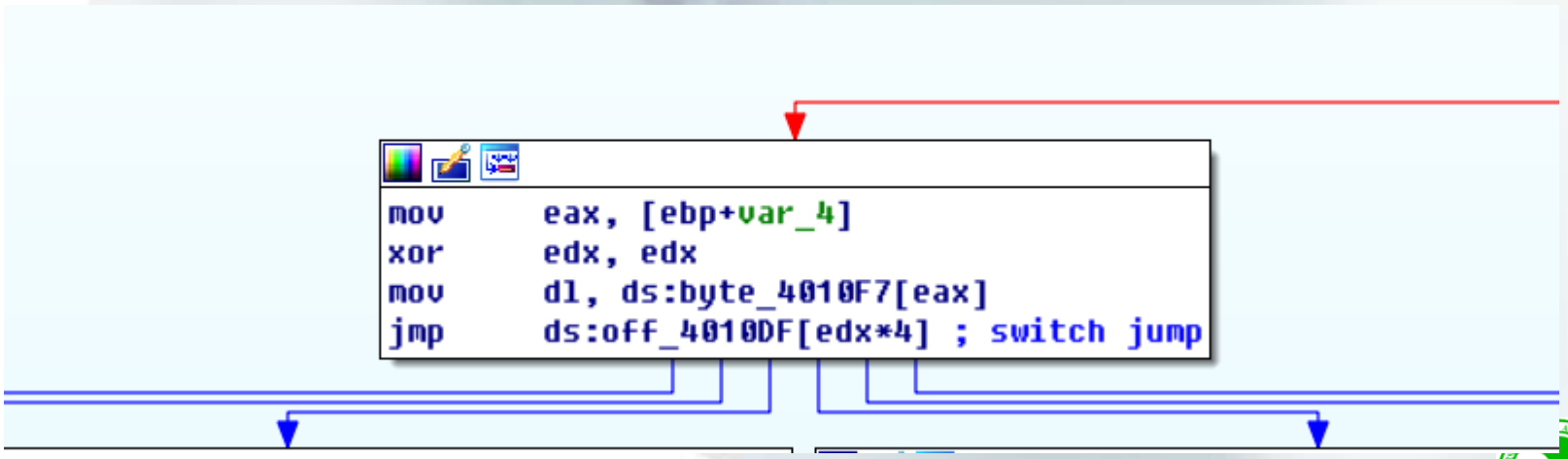
二. 流程控制语句

- 这张表由255个元素组成，数组的大小由我们case值最大的确定即255
- 程序根据输入的case值，在第一张表中取索引值，然后在第二个地址数组中，获取相应的地址进行跳转

```
off_4010DF      dd offset loc_401052, offset loc_401067, offset loc_40107D
                                        ; DATA XREF: _main_0+3B↑r
                dd offset loc_401093, offset loc_4010A9, offset loc_4010BF ; jump table for switch statement
```


二. 流程控制语句

- 将两张表结合起来看，可以得到以下结论
  - 根据case值从索引表里获取索引，根据索引从地址数组里获取地址，并跳转
  - 索引数组中均为5的索引全部都指向的default分支，其余的分支均为case值对应的分支。



二. 流程控制语句

- 这些并不是switch控制流结构的全部，讲解的这些东西仅仅是作为入门，抛砖引玉，希望能提高大家自我学习的兴趣，查阅资料进行深入的学习

二. 流程控制语句

- （1）if,elseif,else选择控制块
- （2）switch case选择控制块
- （3） while/for/do循环控制块

二. 流程控制语句

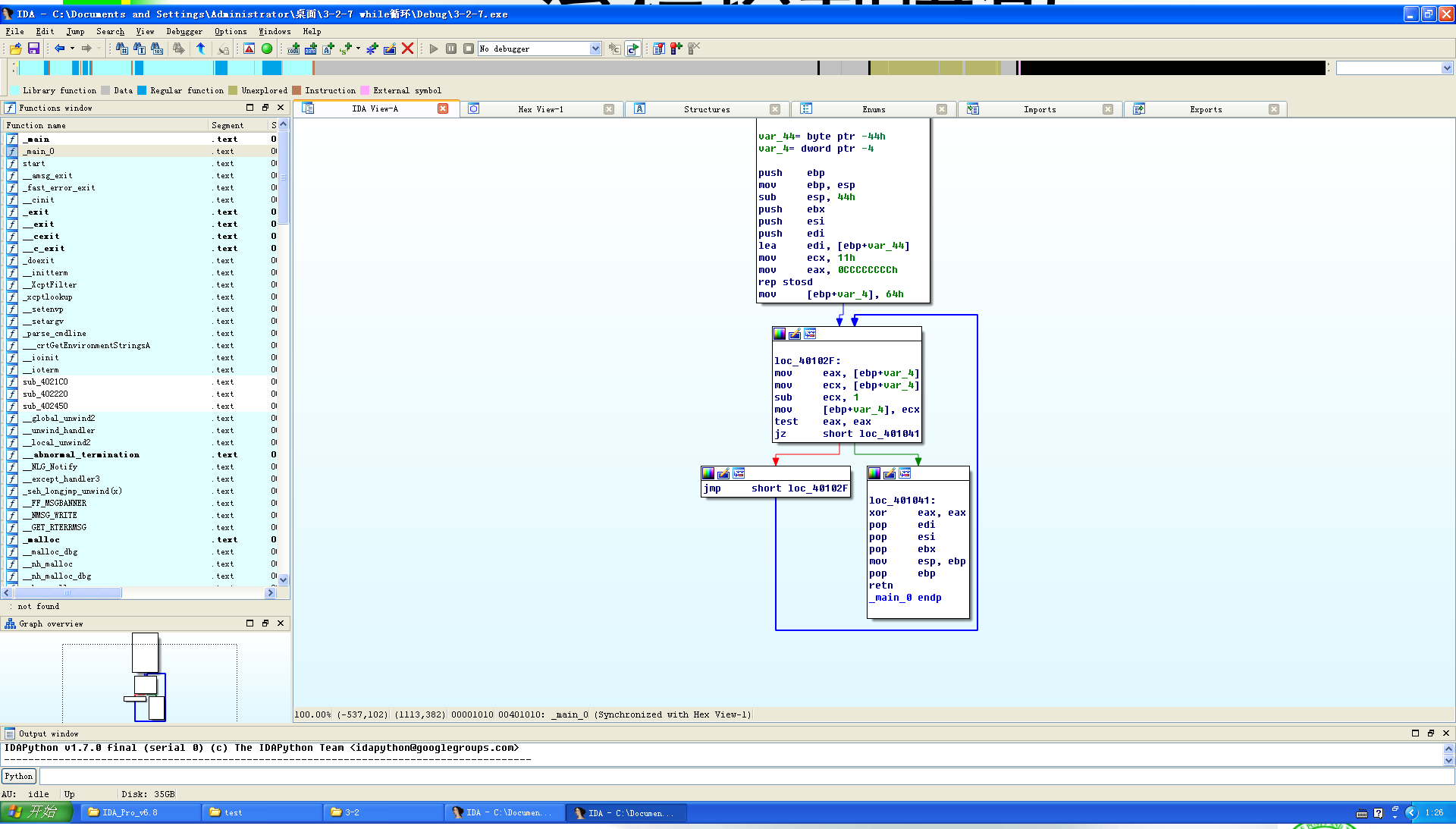
- （3） while/for/do循环控制块
  - while循环控制块
  - for循环控制块
  - do循环控制块

二. 流程控制语句

- while循环 (vc++ 6.0 debug版本)

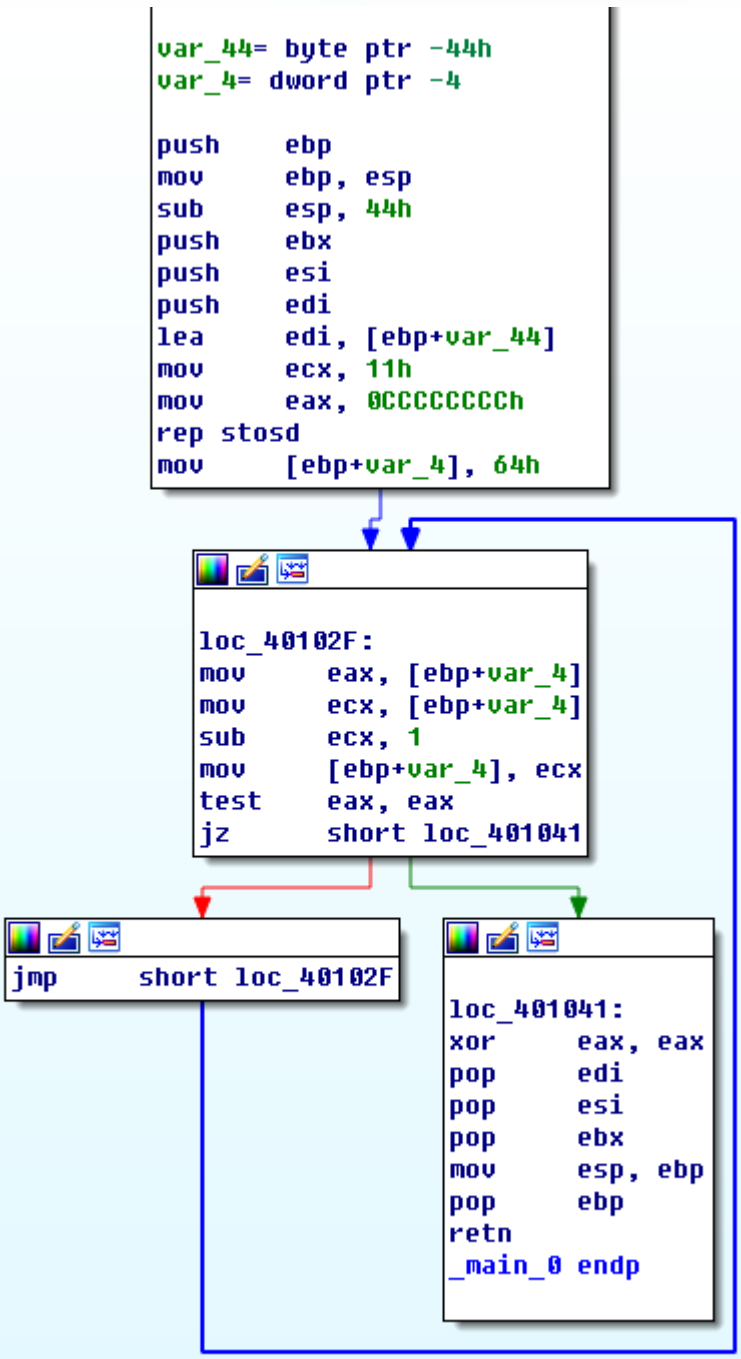
```
Int main()
{
        int i=100;
        while(i--)
        {

        }
        return0;
}
```


一、流程控制语句


IDA - C:\Documents and Settings\Administrator\桌面\3-2-7 while循环\Debug\3-2-7.exe
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Data Regular function Unexplored Instruction External symbol
Functions window
Function name Segment
_main .text
_main_0 .text
start .text
__amsg_exit .text
_fast_error_exit .text
__cinit .text
_exit .text
__exit .text
__cexit .text
__c_exit .text
_doexit .text
__initterm .text
__XcptFilter .text
_xcptlookup .text
__setenvp .text
__setargv .text
_parse_cmdline .text
___crtGetEnvironmentStringsA .text
__ioinit .text
__ioterm .text
sub_4021C0 .text
sub_402220 .text
sub_402450 .text
__global_unwind2 .text
__unwind_handler .text
__local_unwind2 .text
__abnormal_termination .text
__NLG_Notify .text
__except_handler3 .text
_seh_longjmp_unwind(x) .text
__FF_MSGBANNER .text
__NMSG_WRITE .text
__GET_RTERRMSG .text
_malloc .text
__malloc_dbg .text
__nh_malloc .text
__nh_malloc_dbg .text
: not found
Graph overview
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
var_44= byte ptr -44h
var_4= dword ptr -4
push ebp
mov ebp, esp
sub esp, 44h
push ebx
push esi
push edi
lea edi, [ebp+var_44]
mov ecx, 11h
mov eax, 0CCCCCCCCh
rep stosd
mov [ebp+var_4], 64h
loc_40102F:
mov eax, [ebp+var_4]
mov ecx, [ebp+var_4]
sub ecx, 1
mov [ebp+var_4], ecx
test eax, eax
jz short loc_401041
jmp short loc_40102F
loc_401041:
xor eax, eax
pop edi
pop esi
pop ebx
mov esp, ebp
pop ebp
retn
_main_0 endp
100.00% (-537,102) (1113,382) 00001010 00401010: _main_0 (Synchronized with Hex View-1)
Output window
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
Python
AU: idle Up Disk: 35GB
开始
IDA_Pro_v6.8
test
3-2
IDA - C:\Documen...
IDA - C:\Documen...
1:26

二. 流程控制语句

- 我们可以看到，while循环有两次跳转，所在的循环控制块一定是由闭合的基本块组成，循环，顾名思义，一定是闭合的。

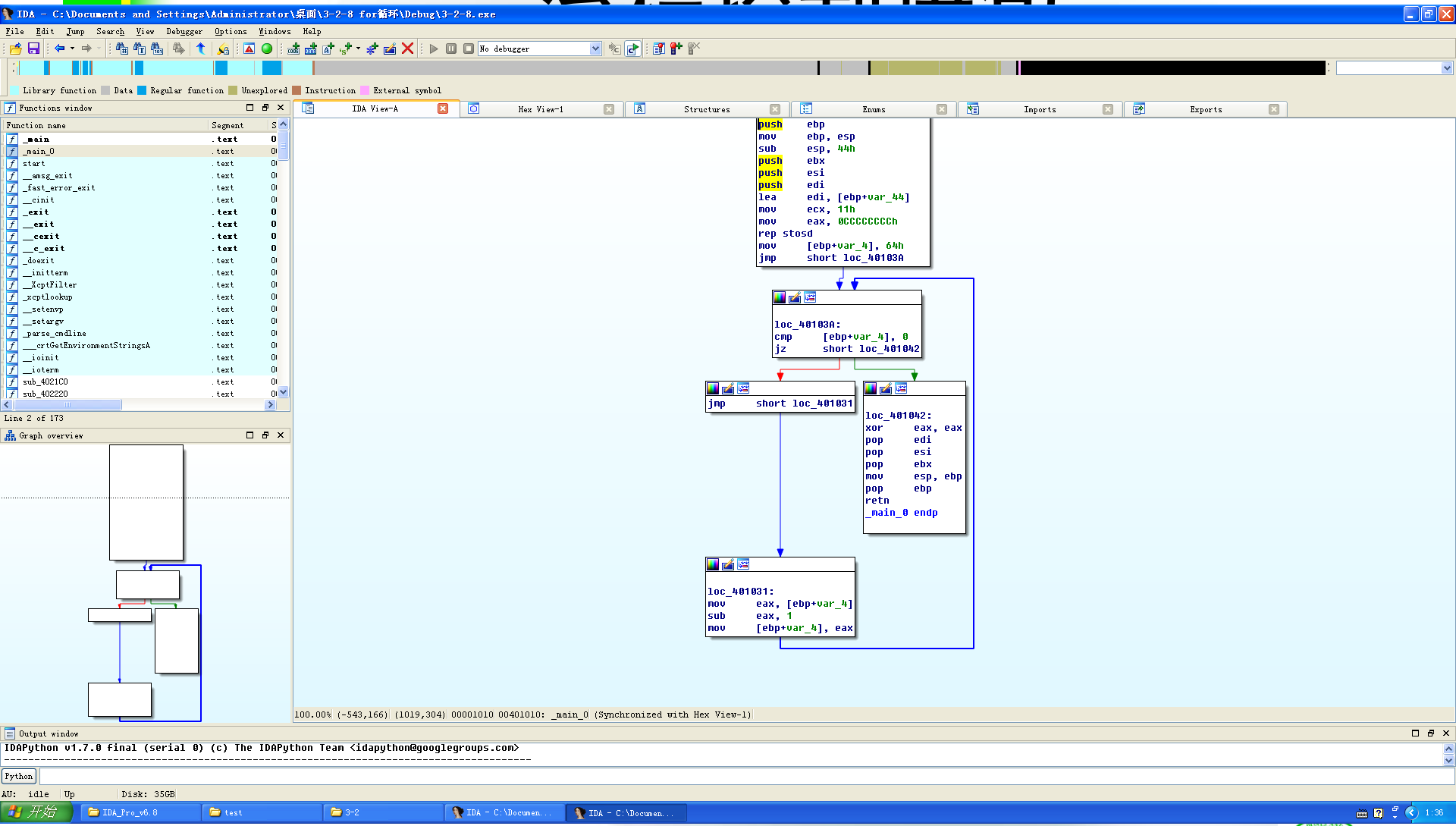

var_44= byte ptr -44h
var_4= dword ptr -4
push ebp
mov ebp, esp
sub esp, 44h
push ebx
push esi
push edi
lea edi, [ebp+var_44]
mov ecx, 11h
mov eax, 0CCCCCCCCh
rep stosd
mov [ebp+var_4], 64h
loc_40102F:
mov eax, [ebp+var_4]
mov ecx, [ebp+var_4]
sub ecx, 1
mov [ebp+var_4], ecx
test eax, eax
jz short loc_401041
jmp short loc_40102F
loc_401041:
xor eax, eax
pop edi
pop esi
pop ebx
mov esp, ebp
pop ebp
retn
_main_0 endp

二. 流程控制语句

- for循环 (vc++ 6.0 debug版本)

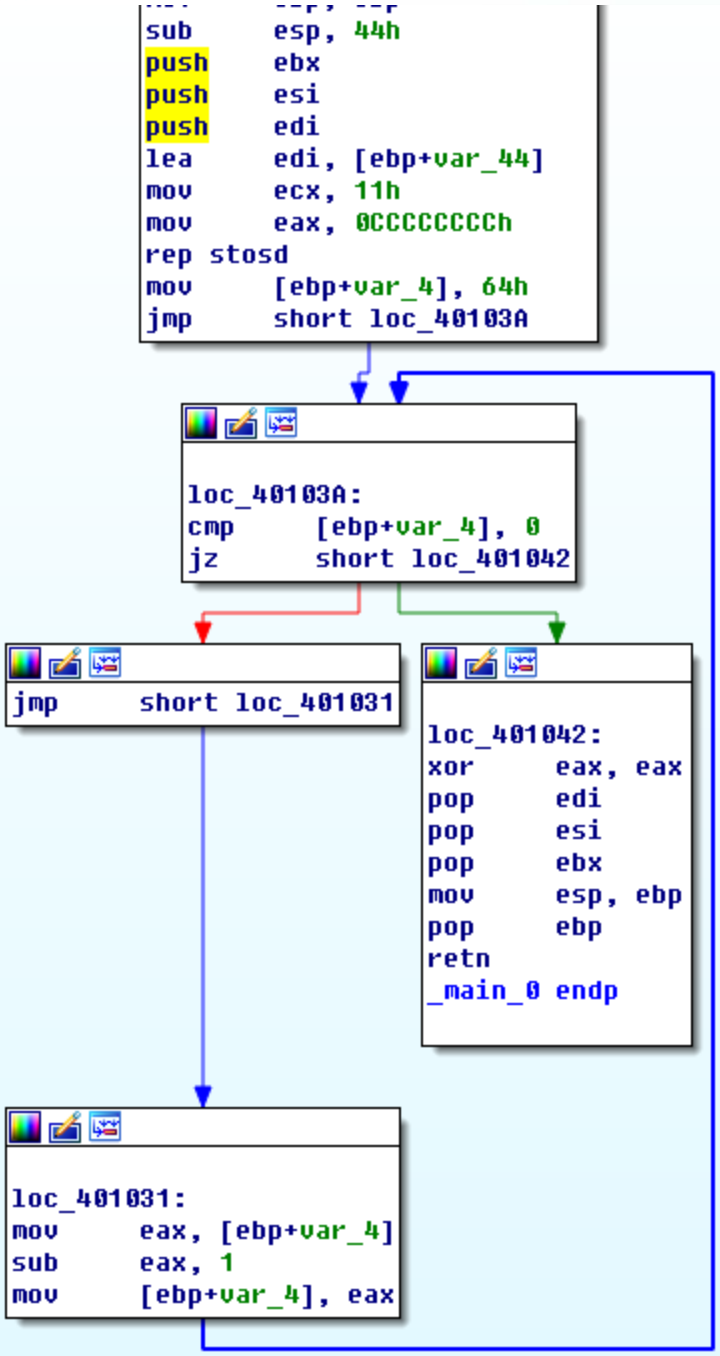
```
Int main()
{
        inti=100;
        for(;i;i--)
        {

        }
        return0;
}
```

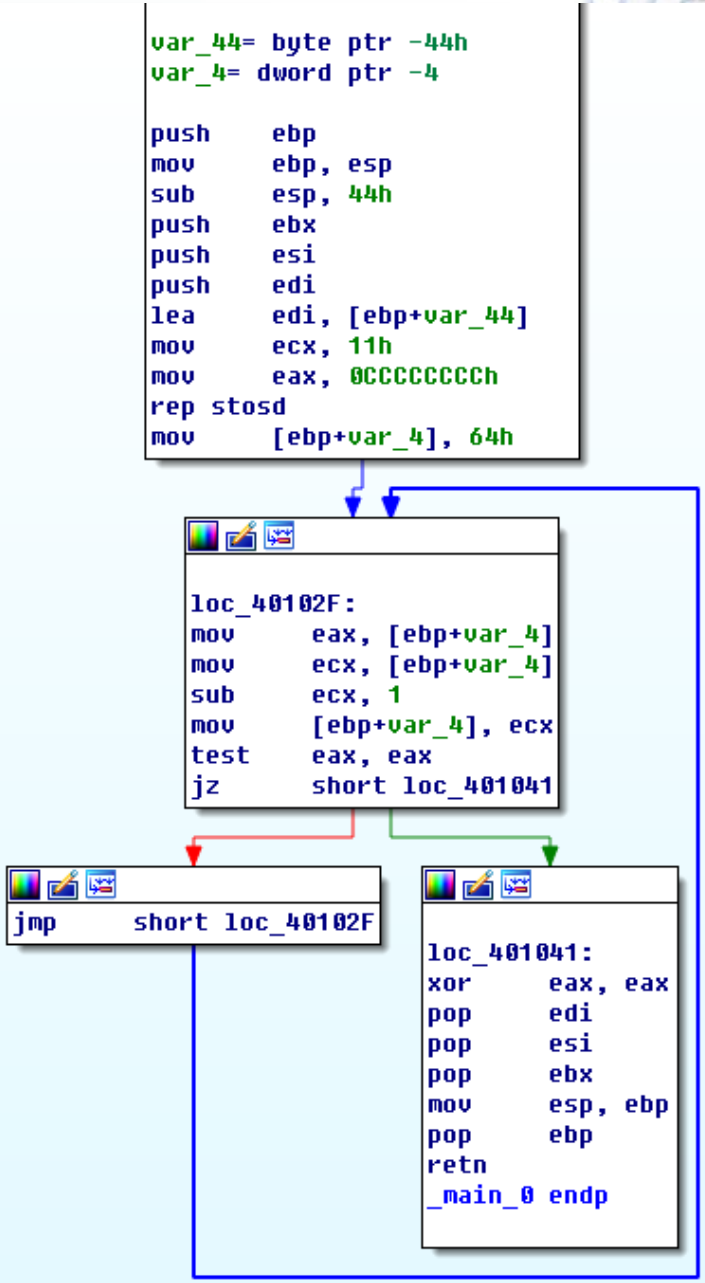

一、流程控制语句


IDA - C:\Documents and Settings\Administrator\桌面\3-2-8 for循环\Debug\3-2-8.exe
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Data Regular function Unexplored Instruction External symbol
Functions window
Function name Segment
_main .text
_main_0 .text
start .text
__amsg_exit .text
_fast_error_exit .text
__cinit .text
_exit .text
__exit .text
__cexit .text
__c_exit .text
_doexit .text
__initterm .text
__XcptFilter .text
_xcptlookup .text
__setenvp .text
__setargv .text
_parse_cmdline .text
___crtGetEnvironmentStringsA .text
__ioinit .text
__ioterm .text
sub_4021C0 .text
sub_402220 .text
Line 2 of 173
Graph overview
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
push ebp
mov ebp, esp
sub esp, 44h
push ebx
push esi
push edi
lea edi, [ebp+var_44]
mov ecx, 11h
mov eax, 0CCCCCCCCh
rep stosd
mov [ebp+var_4], 64h
jmp short loc_40103A
loc_40103A:
cmp [ebp+var_4], 0
jz short loc_401042
jmp short loc_401031
loc_401042:
xor eax, eax
pop edi
pop esi
pop ebx
mov esp, ebp
pop ebp
retn
_main_0 endp
loc_401031:
mov eax, [ebp+var_4]
sub eax, 1
mov [ebp+var_4], eax
100.00% (-543,166) (1019,304) 00001010 00401010: _main_0 (Synchronized with Hex View-1)
Output window
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
Python
AU: idle Up Disk: 35GB
开始
IDA_Pro_v6.8
test
3-2
IDA - C:\Documen...
IDA - C:\Documen...
1:36

二. 流程控制语句

- 对于for循环同样有两次跳转，整个循环的判断逻辑也和while表示式的运算顺序一模一样，在汇编中，能清楚地看到代码逻辑的执行。


sub esp, 44h
push ebx
push esi
push edi
lea edi, [ebp+var_44]
mov ecx, 11h
mov eax, 0CCCCCCCCh
rep stosd
mov [ebp+var_4], 64h
jmp short loc_40103A
loc_40103A:
cmp [ebp+var_4], 0
jz short loc_401042
jmp short loc_401031
loc_401042:
xor eax, eax
pop edi
pop esi
pop ebx
mov esp, ebp
pop ebp
retn
_main_0 endp
loc_401031:
mov eax, [ebp+var_4]
sub eax, 1
mov [ebp+var_4], eax

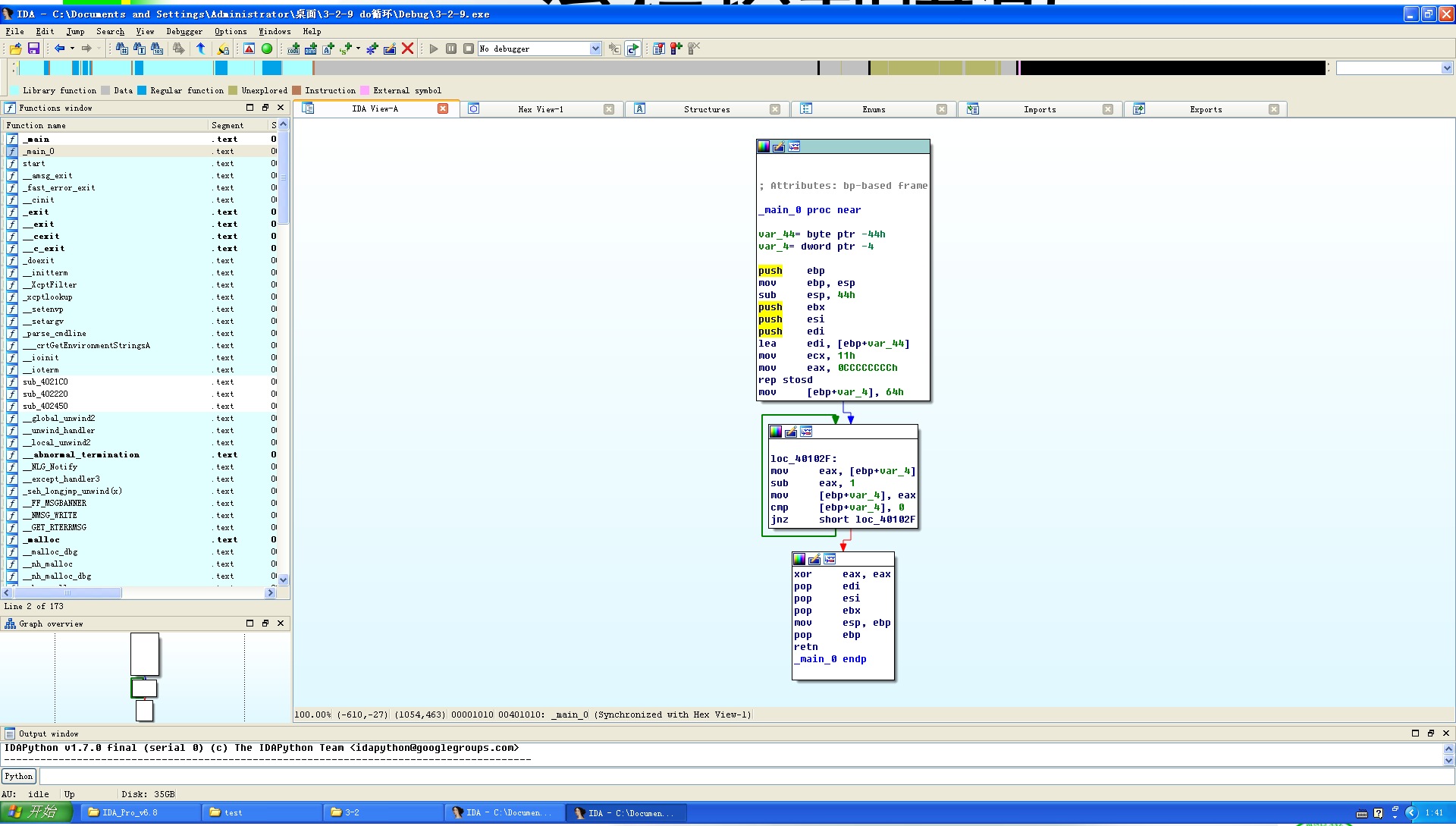

var_44= byte ptr -44h
var_4= dword ptr -4
push ebp
mov ebp, esp
sub esp, 44h
push ebx
push esi
push edi
lea edi, [ebp+var_44]
mov ecx, 11h
mov eax, 0CCCCCCCCh
rep stosd
mov [ebp+var_4], 64h
loc_40102F:
mov eax, [ebp+var_4]
mov ecx, [ebp+var_4]
sub ecx, 1
mov [ebp+var_4], ecx
test eax, eax
jz short loc_401041
jmp short loc_40102F
loc_401041:
xor eax, eax
pop edi
pop esi
pop ebx
mov esp, ebp
pop ebp
retn
_main_0 endp

二. 流程控制语句

- do循环 (vc++ 6.0 debug版本)

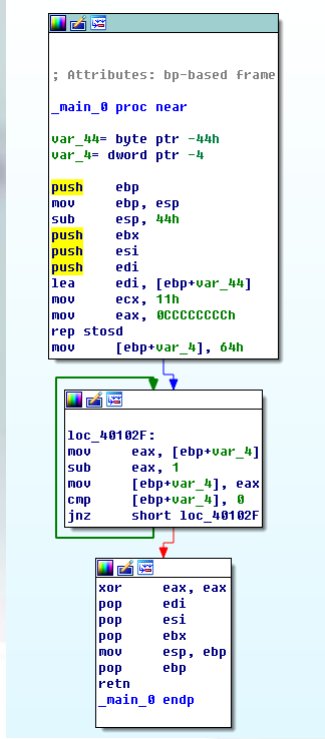
```
Int main()
{
        inti=100;
        do
        {
                i--;
        }while(i);
        return0;
}
```


一、流程控制语句


IDA - C:\Documents and Settings\Administrator\桌面\3-2-9 do循环\Debug\3-2-9.exe
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Data Regular function Unexplored Instruction External symbol
Functions window
Function name Segment
_main .text
_main_0 .text
start .text
__amsg_exit .text
_fast_error_exit .text
__cinit .text
_exit .text
__exit .text
__cexit .text
__c_exit .text
_doexit .text
__initterm .text
__XcptFilter .text
_xcptlookup .text
__setenvp .text
__setargv .text
_parse_cmdline .text
___crtGetEnvironmentStringsA .text
__ioinit .text
__ioterm .text
sub_4021C0 .text
sub_402220 .text
sub_402450 .text
__global_unwind2 .text
__unwind_handler .text
__local_unwind2 .text
__abnormal_termination .text
__NLG_Notify .text
__except_handler3 .text
_seh_longjmp_unwind(x) .text
__FF_MSGBANNER .text
__NMSG_WRITE .text
__GET_RTERRMSG .text
_malloc .text
__malloc_dbg .text
__nh_malloc .text
__nh_malloc_dbg .text
Line 2 of 173
Graph overview
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
; Attributes: bp-based frame
_main_0 proc near
var_44= byte ptr -44h
var_4= dword ptr -4
push ebp
mov ebp, esp
sub esp, 44h
push ebx
push esi
push edi
lea edi, [ebp+var_44]
mov ecx, 11h
mov eax, 0CCCCCCCCh
rep stosd
mov [ebp+var_4], 64h
loc_40102F:
mov eax, [ebp+var_4]
sub eax, 1
mov [ebp+var_4], eax
cmp [ebp+var_4], 0
jnz short loc_40102F
xor eax, eax
pop edi
pop esi
pop ebx
mov esp, ebp
pop ebp
retn
_main_0 endp
100.00% (-610,-27) (1054,463) 00001010 00401010: _main_0 (Synchronized with Hex View-1)
Output window
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
Python
AU: idle Up Disk: 35GB
开始
IDA_Pro_v6.8
test
3-2
IDA - C:\Documen...
IDA - C:\Documen...
1:41

二. 流程控制语句

- 可以看到do循环只有一次跳转，这也是为什么do循环的效率要更高一些的原因。



二. 流程控制语句

- 以上就是三种循环的简单对比，在逆向分析过程中，循环代码都是很好辨认的代码

第三章从C语言看汇编

- 一. 基本数据类型
- 二. 流程控制语句
- 三. 变量表现形式

三. 变量表现形式

- 在之前的学习中，大家已经基本了解了栈结构，接触到了存储在栈中的局部变量
- 除了局部变量，还需要了解一下
  - 全局变量
  - 静态变量

三. 变量表现形式

- 局部变量的作用域属于函数作用域，在”{}”语句内定义的变量，只能在定义其的”{}”语句内才能访问到
- 静态变量属于文件作用域
- 全局变量则属于进程作用域

三. 变量表现形式

- PE(Windows下可执行程序)文件包括以下节区:
  - 1.textbss/BSS
    - BSS段通常是用来存放程序中未初始化的全局变量的一块内存区域。属于静态内存分配。
  - 2.text/CODE
    - 代码段通常是指用来存放程序执行代码的一块内存区域。
    - 这部分区域的大小在程序运行前就已经确定，并且内存区域属于只读。
    - 在代码段中，也有可能包含一些只读的常量
  - 3.rdata
    - 只读数据段

三. 变量表现形式

- 4.data
  - 数据段通常是指用来存放程序中以初始化的全局变量的一块内存区域。属于静态内存分配。
- 5.idata
  - 导入段。包含程序需要的所有DLL文件信息。
- 6.edata
  - 导出段。包含所有提供给其他程序使用的函数和数据。
- 7.rsrc
  - 资源数据段，程序需要用到的资源数据。
- 8.reloc
  - 重定位段。如果加载PE文件失败，将基于此段进行重新调整。

三. 变量表现形式

- （1）栈中的局部变量
- （2）全局变量
- （3）全局静态变量和局部静态变量

三. 变量表现形式

（1）栈中的局部变量

- 局部变量在栈中的分布 (vc++ 6.0 debug版本)

```
Int main(int argc,char *argv[])
{
        char str1[20]={0,};
        char *p1 = str1;
        int intValue=80;
        printf("%d\n",argc);
        return0;
}
```


三. 变量表现形式

- 1. 20个长度的空字符串
  - 00401028-0040103E，可看到程序将栈中刚好20个字节的长度置0，

```
.text:00401028        mov     [ebp+var_14], 0
.text:0040102C        xor     eax, eax
.text:0040102E        mov     [ebp+var_13], eax
.text:00401031        mov     [ebp+var_F], eax
.text:00401034        mov     [ebp+var_B], eax
.text:00401037        mov     [ebp+var_7], eax
.text:0040103A        mov     [ebp+var_3], ax
.text:0040103E        mov     [ebp+var_1], al
.text:00401041        lea     ecx, [ebp+var_14]
.text:00401044        mov     [ebp+var_18], ecx
.text:00401047        mov     [ebp+var_1C], 50h
.text:0040104E        mov     edx, [ebp+arg_0]
.text:00401051        push    edx
.text:00401052        push    offset aD       ; "%d\n"
.text:00401057        call    _printf
```


三. 变量表现形式

- 从[ebp+var_14](ebp-20)到[ebp+var_1](ebp-1)之间的栈空间均被初始化。

```
.text:00401028                  mov     [ebp+var_14], 0
.text:0040102C                  xor     eax, eax
.text:0040102E                  mov     [ebp+var_13], eax
.text:00401031                  mov     [ebp+var_F], eax
.text:00401034                  mov     [ebp+var_B], eax
.text:00401037                  mov     [ebp+var_7], eax
.text:0040103A                  mov     [ebp+var_3], ax
.text:0040103E                  mov     [ebp+var_1], al
.text:00401041                  lea     ecx, [ebp+var_14]
.text:00401044                  mov     [ebp+var_18], ecx
.text:00401047                  mov     [ebp+var_1C], 50h
.text:0040104E                  mov     edx, [ebp+arg_0]
.text:00401051                  push    edx
.text:00401052                  push    offset aD       ; "%d\n"
.text:00401057                  call    _printf
```

```
.text:00401010 var_5C           = byte ptr -5Ch
.text:00401010 var_1C           = dword ptr -1Ch
.text:00401010 var_18           = dword ptr -18h
.text:00401010 var_14           = byte ptr -14h
.text:00401010 var_13           = dword ptr -13h
.text:00401010 var_F            = dword ptr -0Fh
.text:00401010 var_B            = dword ptr -0Bh
.text:00401010 var_7            = dword ptr -7
.text:00401010 var_3            = word ptr -3
.text:00401010 var_1            = byte ptr -1
.text:00401010 arg_0            = dword ptr  8
```


三. 变量表现形式

2. 指针指向数组

- 紧接下来的两句指令，[ebp+var_18](ebp-24)处存储了20个字节数组的起始地址。

```
char *p1 = str1;
```

```
.text:00401028          mov     [ebp+var_14], 0
.text:0040102C          xor     eax, eax
.text:0040102E          mov     [ebp+var_13], eax
.text:00401031          mov     [ebp+var_F], eax
.text:00401034          mov     [ebp+var_B], eax
.text:00401037          mov     [ebp+var_7], eax
.text:0040103A          mov     [ebp+var_3], ax
.text:0040103E          mov     [ebp+var_1], al
.text:00401041          lea     ecx, [ebp+var_14]
.text:00401044          mov     [ebp+var_18], ecx
.text:00401047          mov     [ebp+var_1C], 50h
.text:0040104E          mov     edx, [ebp+arg_0]
.text:00401051          push    edx
.text:00401052          push    offset aD       ; "%d\n"
.text:00401057          call    _printf
```


三. 变量表现形式

3. 整型局部变量

- 程序的整型局部变量则存储在了[ebp+var_1C](ebp-28)处
- 而[ebp+arg_0]则是参数argc的值

```
int intValue=80;
```

```
.text:00401028                  mov     [ebp+var_14], 0
.text:0040102C                  xor     eax, eax
.text:0040102E                  mov     [ebp+var_13], eax
.text:00401031                  mov     [ebp+var_F], eax
.text:00401034                  mov     [ebp+var_B], eax
.text:00401037                  mov     [ebp+var_7], eax
.text:0040103A                  mov     [ebp+var_3], ax
.text:0040103E                  mov     [ebp+var_1], al
.text:00401041                  lea     ecx, [ebp+var_14]
.text:00401044                  mov     [ebp+var_18], ecx
.text:00401047                  mov     [ebp+var_1C], 50h
.text:0040104E                  mov     edx, [ebp+arg_0]
.text:00401051                  push    edx
.text:00401052                  push    offset aD       ; "%d\n"
.text:00401057                  call    _printf
```


三. 变量表现形式

- （1）栈中的局部变量
- （2）全局变量
- （3）全局静态变量和局部静态变量

三. 变量表现形式

- 全局变量的表现形式 (vc++ 6.0 debug版本)

```
int global_var=0x66666666;
int main(int argc,char *argv[])
{
      printf("%d\n",global_var);
      return0;
}
```


三. 变量表现形式

- 输出全局变量的值，主要看看全局变量 dword_424A30

```
mov     eax, dword_424A30
push    eax
push    offset aD          ; "%d\n"
call    _printf
```

- 已经初始化的全局变量所在区段是.data段

```
.data:00424A30 dword_424A30     dd 66666666h                    ;
```


三. 变量表现形式

- 通过ida->View->Open subviews->Segments，可以看到程序中的所有区段

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	es	ss	ds	fs	gs
.text	00401000	00422000	R	.	X	.	L	para	0001	public	CODE	32	0000	0000	0003	FFFF…	FFFF…
.rdata	00422000	00424000	R	.	.	.	L	para	0002	public	DATA	32	0000	0000	0003	FFFF…	FFFF…
.data	00424000	0042A000	R	W	.	.	L	para	0003	public	DATA	32	0000	0000	0003	FFFF…	FFFF…
.idata	0042A148	0042A268	R	W	.	.	L	para	0004	public	DATA	32	0000	0000	0003	FFFF…	FFFF…

三. 变量表现形式

- （1）栈中的局部变量
- （2）全局变量
- （3）全局静态变量和局部静态变量

三. 变量表现形式

- 静态变量分为全局静态变量和局部静态变量
- 全局静态变量
  - 全局静态变量和全局变量类似，只是全局静态变量只能在本文件内使用
  - 全局静态变量等价于编译器限制外部源码文件访问的全局变量
- 局部静态变量
  - 局部静态变量则比较特殊
  - C语言中局部静态变量的赋值只进行一次，而且它不会随作用域的结束而消失，并且在未进入作用域之前就已经存在

三. 变量表现形式

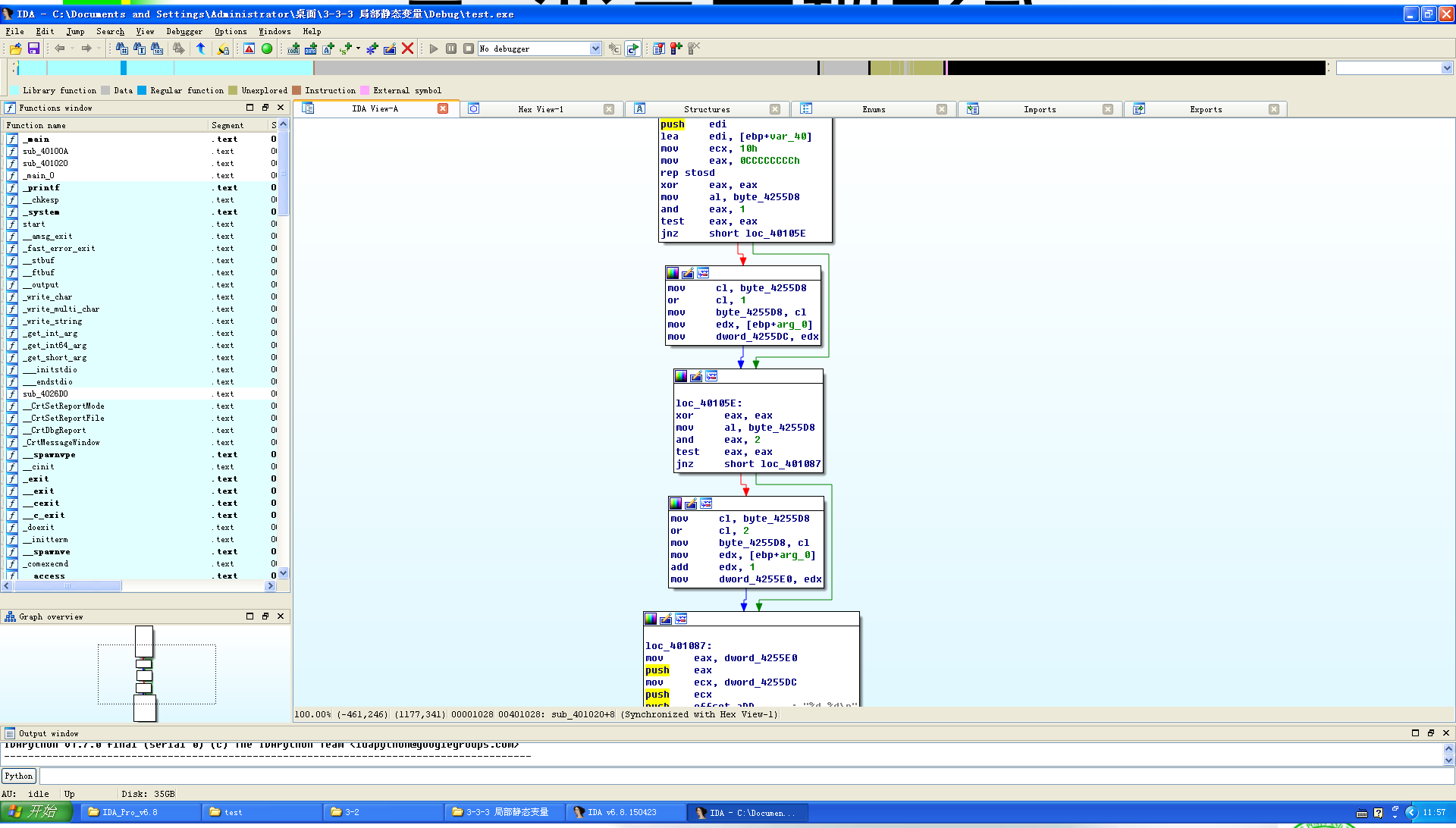
- 局部静态变量 (vc++ 6.0 debug版本)

```
void testStaticVar(inti)
{
      staticint staticVar=i;
      staticint staticVar2 =i+1;
      printf("%d %d\n", staticVar,staticVar2);
}
int main(int argc,char *argv[])
{
      for(int i=1;i<=5;i++)
      {
            testStaticVar(i);
      }
      system("pause");
      return0;
}
```

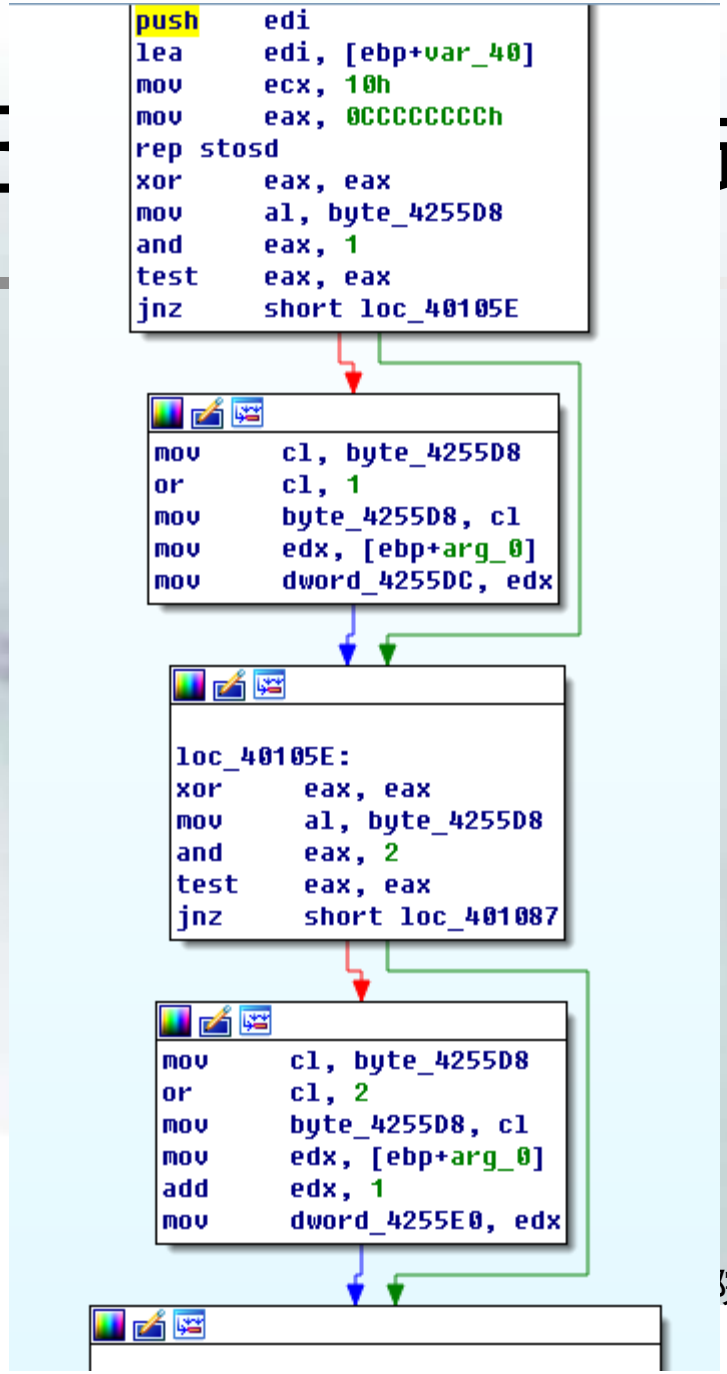

三. 变量表现形式

- 从汇编看局部静态变量 (vc++ 6.0 debug版本)

三、变量表现形式


IDA - C:\Documents and Settings\Administrator\桌面\3-3-3 局部静态变量\Debug\test.exe
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Data Regular function Unexplored Instruction External symbol
Functions window
Function name Segment
_main .text
sub_40100A .text
sub_401020 .text
_main_0 .text
_printf .text
__chkesp .text
_system .text
start .text
__amsg_exit .text
_fast_error_exit .text
__stbuf .text
__ftbuf .text
__output .text
_write_char .text
_write_multi_char .text
_write_string .text
_get_int_arg .text
_get_int64_arg .text
_get_short_arg .text
___initstdio .text
___endstdio .text
sub_4026D0 .text
__CrtSetReportMode .text
__CrtSetReportFile .text
__CrtDbgReport .text
_CrtMessageWindow .text
__spawnvpe .text
__cinit .text
_exit .text
__exit .text
__cexit .text
__c_exit .text
_doexit .text
__initterm .text
__spawnve .text
_comexecmd .text
_access .text
Graph overview
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
push edi
lea edi, [ebp+var_40]
mov ecx, 10h
mov eax, 0CCCCCCCCh
rep stosd
xor eax, eax
mov al, byte_4255D8
and eax, 1
test eax, eax
jnz short loc_40105E
mov cl, byte_4255D8
or cl, 1
mov byte_4255D8, cl
mov edx, [ebp+arg_0]
mov dword_4255DC, edx
loc_40105E:
xor eax, eax
mov al, byte_4255D8
and eax, 2
test eax, eax
jnz short loc_401087
mov cl, byte_4255D8
or cl, 2
mov byte_4255D8, cl
mov edx, [ebp+arg_0]
add edx, 1
mov dword_4255E0, edx
loc_401087:
mov eax, dword_4255E0
push eax
mov ecx, dword_4255DC
push ecx
100.00% (-461,246) (1177,341) 00001028 00401028: sub_401020+8 (Synchronized with Hex View-1)
Output window
Python
AU: idle Up Disk: 35GB
开始
IDA_Pro_v6.8
test
3-2
3-3-3 局部静态变量
IDA v6.8.150423
IDA - C:\Documen...
11:57

三 式


push edi
lea edi, [ebp+var_40]
mov ecx, 10h
mov eax, 0CCCCCCCCh
rep stosd
xor eax, eax
mov al, byte_4255D8
and eax, 1
test eax, eax
jnz short loc_40105E
mov cl, byte_4255D8
or cl, 1
mov byte_4255D8, cl
mov edx, [ebp+arg_0]
mov dword_4255DC, edx
loc_40105E:
xor eax, eax
mov al, byte_4255D8
and eax, 2
test eax, eax
jnz short loc_401087
mov cl, byte_4255D8
or cl, 2
mov byte_4255D8, cl
mov edx, [ebp+arg_0]
add edx, 1
mov dword_4255E0, edx

三. 变量表现形式

- 对上述算法进行一下分析，即如果byte_4255D8地址处的字节每次相与的最低bit位为0，则没有赋值，将其置1，然后对局部静态变量赋值;
- 反之， byte_4255D8地址处相与的最低bit位为1，则说明已经对局部静态变量赋过值，不再对其进行赋值。

三. 变量表现形式

- 当然并不是所有编译器的局部静态变量赋值算法都是这样，程序因为使用的是**VC6.0++debug**版本，如果选择**VS2015**等更高版本或者是其他编译器，赋值的算法则不想同。
- 总之，分析方法还是一样，具体情况要根据编译器的编译结果具体情况具体分析。

三. 变量表现形式

- 局部静态变量存储在.data段
- 当局部静态变量被初始化为一个常量值时，由于其在初始化过程中不会产生任何代码，这样无需再做初始化标志，编译器采用了直接以全局变量的方式处理，优化了代码，提升了效率
- 虽然转换为了全局变量，但仍然不可以超出其作用域

```
.data:004255D8 byte_4255D8      db 0                    ; DATA XREF: sub_401020+1A↑r
```


小结

- 通过本节的学习，我们基本了解和掌握了C语言中常见的基本数据类型，基本变量在汇编层次中的表示方法，以及顺序，选择，循环在汇编中的流程控制块表现形式，为之后的学习奠定了一定的基础。

Q & A