



汇编语言与逆向工程

北京邮电大学
2019年3月

北邮网安学院 崔宝江



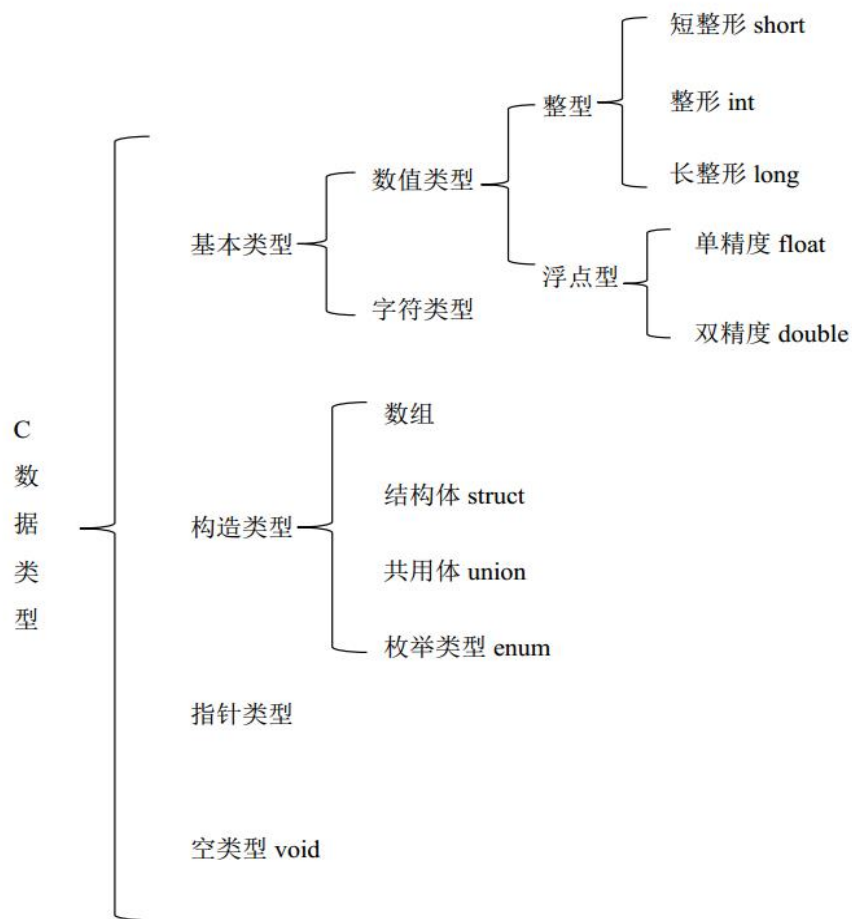
第三章从C语言看汇编

- ① 一. 基本数据类型
- ② 二. 流程控制语句
- ③ 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
 - 一个字节也就是由8个二进制数组成
 - 一个十六进制数可用4个二进制数表示
 - 一个字节由两个十六进制数表示
 - \windows中，一个字等于两个字节
 - 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

```
Int main(int argc,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, pCharValue);  
/* 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");
return 0;
```

}



C:\ "C:\Documents and Settings\Administrator\桌面\3-1 backup\Debug\3-1-1.exe" - _ X

pShortValue: 0012FF78
pIntValue: 0012FF68
pLongValue: 0012FF58
pFloatValue: 0012FF48
pDoubleValue: 0012FF28
pCharValue: 0012FF24
shortValue:
shortValue0: 0012FF78
shortValue1: 0012FF7A
shortValue2: 0012FF7C
shortValue3: 0012FF7E
intValue:
intValue0: 0012FF68
intValue1: 0012FF6C
intValue2: 0012FF70
intValue3: 0012FF74
longValue:
longValue0: 0012FF58
longValue1: 0012FF5C
longValue2: 0012FF60
longValue3: 0012FF64
floatValue:
floatValue0: 0012FF48
floatValue1: 0012FF4C
floatValue2: 0012FF50
floatValue3: 0012FF54
doubleValue:
doubleValue0: 0012FF28
doubleValue1: 0012FF30
doubleValue2: 0012FF38
doubleValue3: 0012FF40
charValue:
charValue0: 0012FF24
charValue1: 0012FF25
charValue2: 0012FF26
charValue3: 0012FF27
请按任意键继续. . .



一. 基本数据类型

④ 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

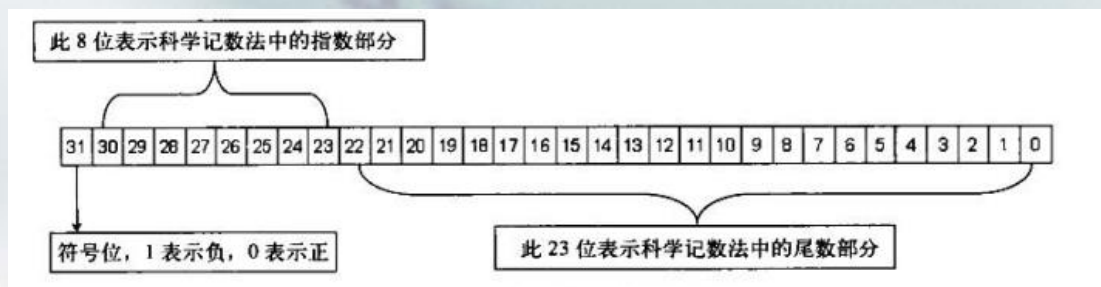
④ 以浮点数**400.1**和**500.02**为例

- ❑ 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



一. 基本数据类型

□ 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



一. 基本数据类型

@ 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

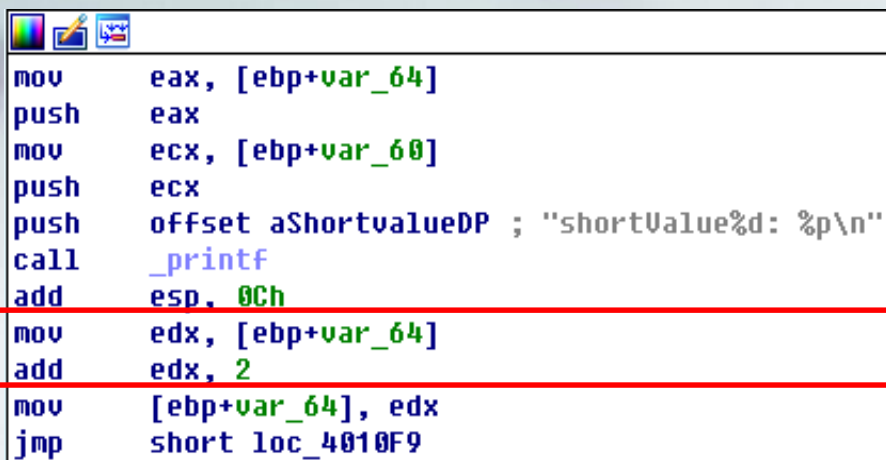
□ 例

- 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
```

The assembly code snippet is displayed in a window. The lines `mov edx, [ebp+var_64]` and `add edx, 2` are highlighted with a red rectangular box.



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**int**指针和**double**指针的自加。



一. 基本数据类型

```
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
```

```
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
```



一. 基本数据类型

- ④ 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。



第三章从C语言看汇编

- ① 一. 基本数据类型
- ② 二. 流程控制语句
- ③ 三. 变量表现形式



二. 流程控制语句

④ 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, else if, 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

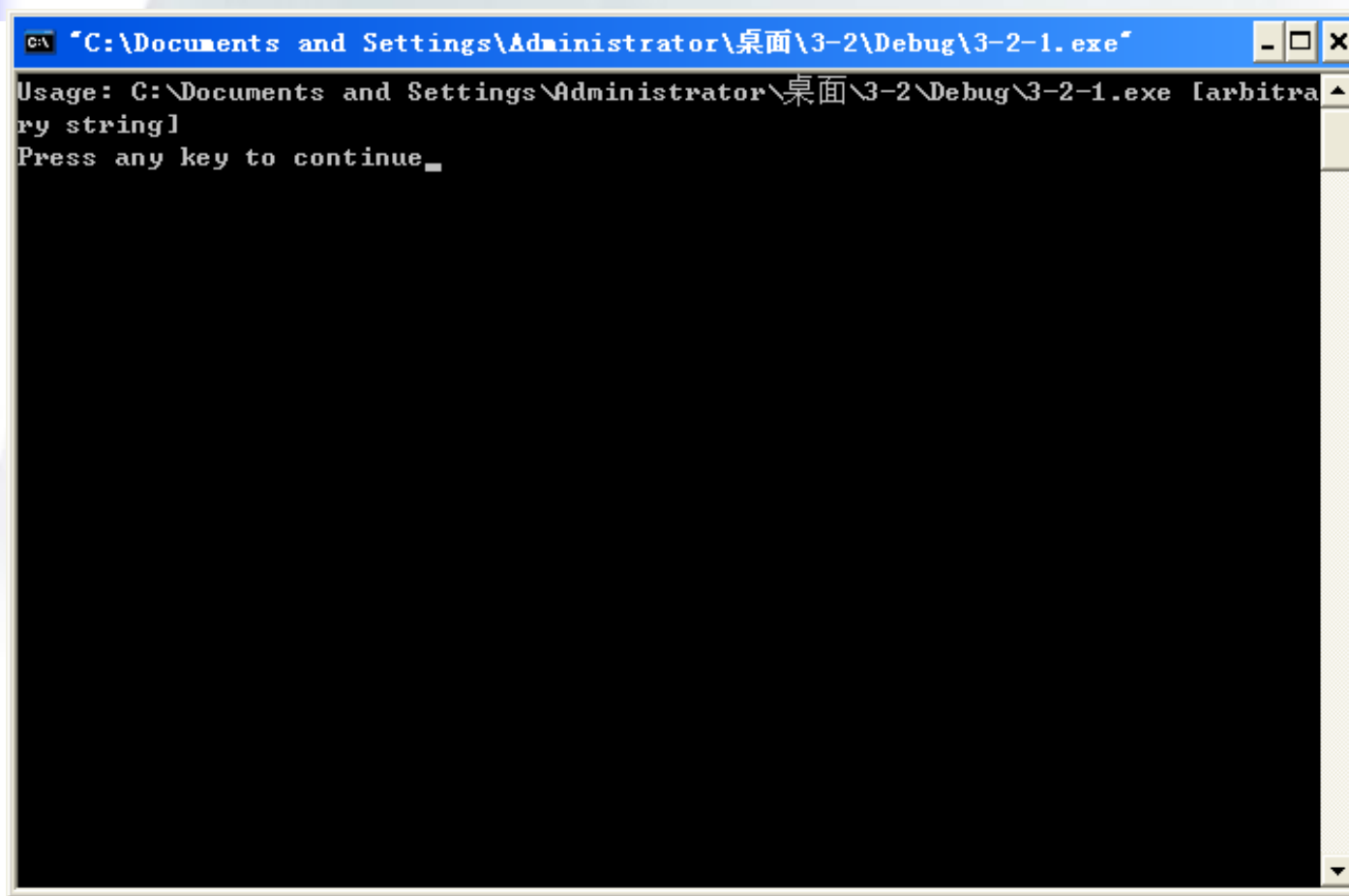
@if语句 (vc++ 6.0 debug版本)

3-2-1

```
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.
```



二. 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

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
_main	.text	0
_main_0	.text	0
_printf	.text	0
_chkesp	.text	0
start	.text	0
_smg_exit	.text	0
_fast_error_exit	.text	0
_stbuf	.text	0
_ftbuf	.text	0
sub_4014E0	.text	0
_write_char	.text	0
_write_multi_char	.text	0
_write_string	.text	0
_get_int_arg	.text	0
_get_int64_arg	.text	0
_get_short_arg	.text	0
_initstdio	.text	0
_endstdio	.text	0
sub_402560	.text	0
_CrtSetReportMode	.text	0
_CrtSetReportFile	.text	0
_CrtDbgReport	.text	0
_CrtMessageWindow	.text	0
_cinit	.text	0
_exit	.text	0
_cexit	.text	0
_doexit	.text	0
_initterm	.text	0
_XcptFilter	.text	0
_xceptlookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse_cmdline	.text	0
_crtGetEnvironmentStringsA	.text	0
_iointit	.text	0

Line 2 of 177

Graph overview

100.00% (-499,-11) (504,23) 00001010 00401010: _main_0 (Synchronized with Hex View-1)

Output window

IDA Python v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

Python

AU: idle Up Disk: 35GB

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

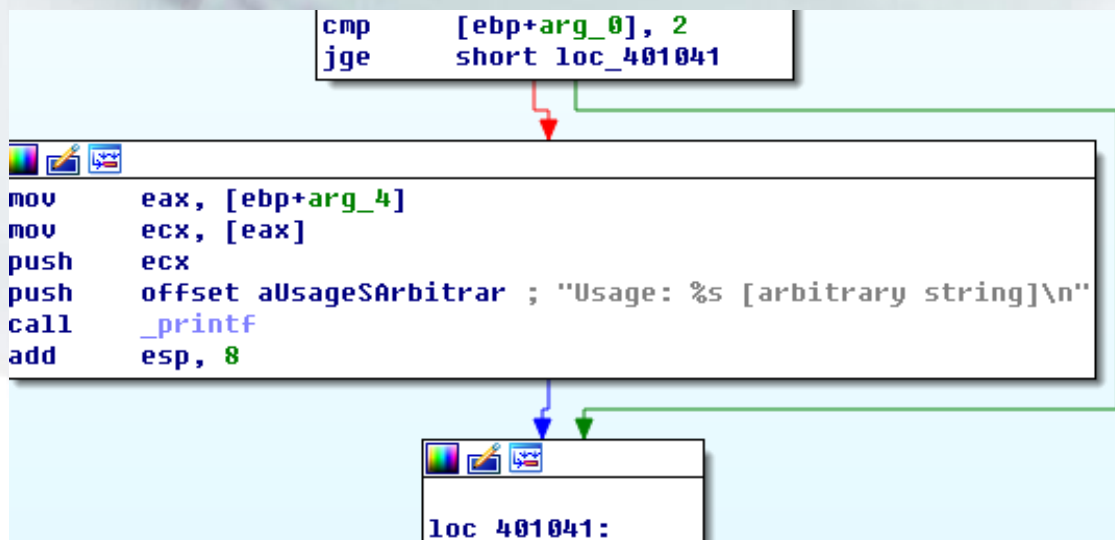
call _chkesp

mov esp, ebp

pop ebp

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)

3-2-2

```
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]);
    }
    return 0;
}
```



二. 流程控制语句

The screenshot displays the IDA Pro interface for a function named `_main_0`. The left pane shows the function list, and the right pane shows the assembly code. A control flow graph (CFG) is overlaid on the assembly, illustrating the execution flow.

Assembly Code:

```
; 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 stsd
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 aHelloWorld$ ; "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
```

Control Flow Graph (CFG):

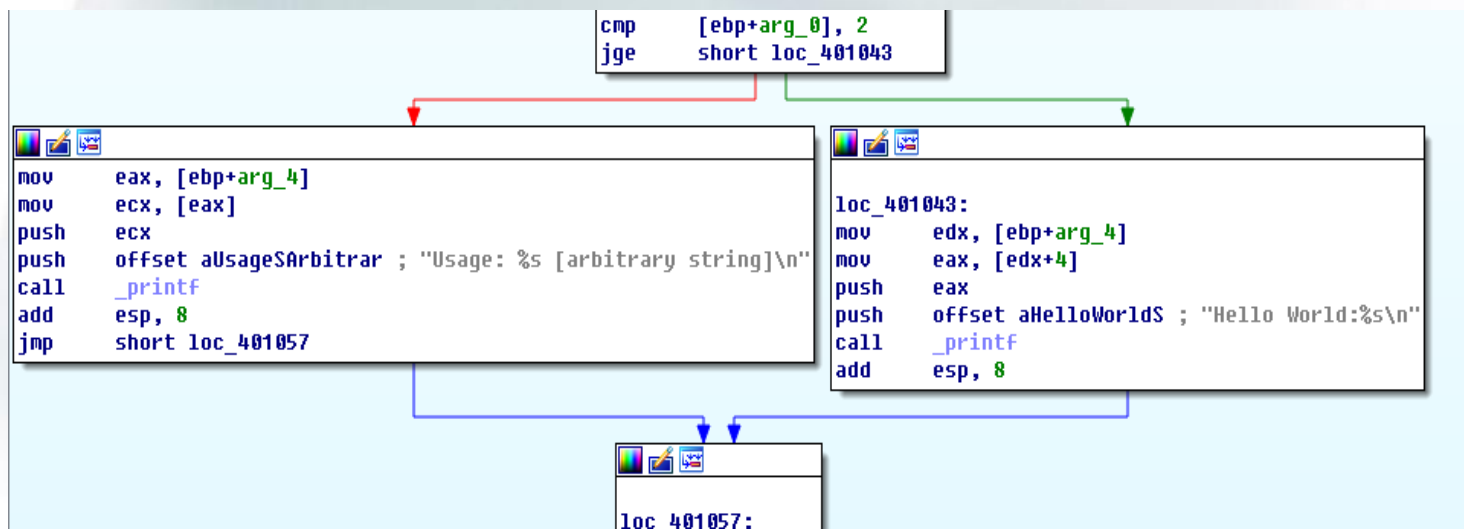
- The graph starts at the entry point of `_main_0`.
- It branches to `loc_401043` if `jge` (jump if greater or equal) is true.
- From the entry point, it goes to a block that pushes `ebp`, sets up the stack, and pushes `ebx`, `esi`, and `edi`.
- This block then calls `_printf` with arguments from `arg_4` and jumps to `loc_401057`.
- `loc_401043` is a block that pushes `eax` and calls `_printf` with arguments from `arg_4`.
- `loc_401057` is a block that performs cleanup (xor `eax`, `pop` registers, `add` `esp`, `cmp` `ebp`, `call` `_chkesp`) and returns.

Output window:

Output window shows the IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>.

二. 流程控制语句

□ 用ida打开，进行分析



○ JGE/JNL 大于或等于转移



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 else if else 控制块（vc++ 6.0 debug版本）



二. 流程控制语句

3-2-3

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



二. 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	S
_main	.text	0
_main_0	.text	0
_printf	.text	0
_chkexp	.text	0
start	.text	0
_smg_exit	.text	0
_fast_error_exit	.text	0
_stbuf	.text	0
_ftbuf	.text	0
sub_401500	.text	0
_write_char	.text	0
_write_multi_char	.text	0
_write_string	.text	0
_get_int_arg	.text	0
_get_int64_arg	.text	0
_get_short_arg	.text	0
_initstdio	.text	0
_endstdio	.text	0
sub_402800	.text	0
_CrSetReportMode	.text	0
_CrSetReportFile	.text	0
_CrDbgReport	.text	0
_CrMessageWindow	.text	0
_cinit	.text	0
_exit	.text	0
_cexit	.text	0
_c_exit	.text	0
_doexit	.text	0
_initterm	.text	0
_xcpFilter	.text	0
_xcpLookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse cmdline	.text	0
_crGetEnvironmentStringsA	.text	0
_jinit	.text	0

Hex View-1

```
arg_0= dword ptr 8
arg_4= dword ptr 0Ch
push ebp
mov esp, esp
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_401079
```

loc_401079:

```
cmp [ebp+arg_0], 4
jnz short loc_401097
```

loc_401097:

```
cmp [ebp+arg_0], 5
jnz short loc_4010B3
```

loc_4010B3:

```
cmp [ebp+arg_0], 6
jnz short loc_4010D9
```

loc_4010D9:

```
cmp [ebp+arg_0], 7
jnz short loc_4010F7
```

loc_4010F7:

```
cmp [ebp+arg_0], 8
jnz short loc_401115
```

loc_401115:

```
cmp [ebp+arg_0], 9
jnz short loc_401133
```

loc_401133:

```
cmp [ebp+arg_0], 10
jnz short loc_401151
```

loc_401151:

```
cmp [ebp+arg_0], 11
jnz short loc_40116F
```

loc_40116F:

```
cmp [ebp+arg_0], 12
jnz short loc_40118D
```

loc_40118D:

```
cmp [ebp+arg_0], 13
jnz short loc_4011AB
```

loc_4011AB:

```
cmp [ebp+arg_0], 14
jnz short loc_4011C9
```

loc_4011C9:

```
cmp [ebp+arg_0], 15
jnz short loc_4011E7
```

loc_4011E7:

```
cmp [ebp+arg_0], 16
jnz short loc_401205
```

loc_401205:

```
cmp [ebp+arg_0], 17
jnz short loc_401223
```

loc_401223:

```
cmp [ebp+arg_0], 18
jnz short loc_401241
```

loc_401241:

```
cmp [ebp+arg_0], 19
jnz short loc_401259
```

loc_401259:

```
cmp [ebp+arg_0], 20
jnz short loc_401277
```

loc_401277:

```
cmp [ebp+arg_0], 21
jnz short loc_401295
```

loc_401295:

```
cmp [ebp+arg_0], 22
jnz short loc_4012B3
```

loc_4012B3:

```
cmp [ebp+arg_0], 23
jnz short loc_4012D1
```

loc_4012D1:

```
cmp [ebp+arg_0], 24
jnz short loc_4012E9
```

loc_4012E9:

```
cmp [ebp+arg_0], 25
jnz short loc_401307
```

loc_401307:

```
cmp [ebp+arg_0], 26
jnz short loc_401325
```

loc_401325:

```
cmp [ebp+arg_0], 27
jnz short loc_401343
```

loc_401343:

```
cmp [ebp+arg_0], 28
jnz short loc_401361
```

loc_401361:

```
cmp [ebp+arg_0], 29
jnz short loc_401379
```

loc_401379:

```
cmp [ebp+arg_0], 30
jnz short loc_401397
```

loc_401397:

```
cmp [ebp+arg_0], 31
jnz short loc_4013B5
```

loc_4013B5:

```
cmp [ebp+arg_0], 32
jnz short loc_4013D3
```

loc_4013D3:

```
cmp [ebp+arg_0], 33
jnz short loc_4013F1
```

loc_4013F1:

```
cmp [ebp+arg_0], 34
jnz short loc_401409
```

loc_401409:

```
cmp [ebp+arg_0], 35
jnz short loc_401427
```

loc_401427:

```
cmp [ebp+arg_0], 36
jnz short loc_401445
```

loc_401445:

```
cmp [ebp+arg_0], 37
jnz short loc_401463
```

loc_401463:

```
cmp [ebp+arg_0], 38
jnz short loc_401481
```

loc_401481:

```
cmp [ebp+arg_0], 39
jnz short loc_401499
```

loc_401499:

```
cmp [ebp+arg_0], 40
jnz short loc_4014B7
```

loc_4014B7:

```
cmp [ebp+arg_0], 41
jnz short loc_4014D5
```

loc_4014D5:

```
cmp [ebp+arg_0], 42
jnz short loc_4014F3
```

loc_4014F3:

```
cmp [ebp+arg_0], 43
jnz short loc_401511
```

loc_401511:

```
cmp [ebp+arg_0], 44
jnz short loc_401529
```

loc_401529:

```
cmp [ebp+arg_0], 45
jnz short loc_401547
```

loc_401547:

```
cmp [ebp+arg_0], 46
jnz short loc_401565
```

loc_401565:

```
cmp [ebp+arg_0], 47
jnz short loc_401583
```

loc_401583:

```
cmp [ebp+arg_0], 48
jnz short loc_4015A1
```

loc_4015A1:

```
cmp [ebp+arg_0], 49
jnz short loc_4015B9
```

loc_4015B9:

```
cmp [ebp+arg_0], 50
jnz short loc_4015D7
```

loc_4015D7:

```
cmp [ebp+arg_0], 51
jnz short loc_4015F5
```

loc_4015F5:

```
cmp [ebp+arg_0], 52
jnz short loc_401613
```

loc_401613:

```
cmp [ebp+arg_0], 53
jnz short loc_401631
```

loc_401631:

```
cmp [ebp+arg_0], 54
jnz short loc_401649
```

loc_401649:

```
cmp [ebp+arg_0], 55
jnz short loc_401667
```

loc_401667:

```
cmp [ebp+arg_0], 56
jnz short loc_401685
```

loc_401685:

```
cmp [ebp+arg_0], 57
jnz short loc_4016A3
```

loc_4016A3:

```
cmp [ebp+arg_0], 58
jnz short loc_4016C1
```

loc_4016C1:

```
cmp [ebp+arg_0], 59
jnz short loc_4016D9
```

loc_4016D9:

```
cmp [ebp+arg_0], 60
jnz short loc_4016F7
```

loc_4016F7:

```
cmp [ebp+arg_0], 61
jnz short loc_401715
```

loc_401715:

```
cmp [ebp+arg_0], 62
jnz short loc_401733
```

loc_401733:

```
cmp [ebp+arg_0], 63
jnz short loc_401751
```

loc_401751:

```
cmp [ebp+arg_0], 64
jnz short loc_401769
```

loc_401769:

```
cmp [ebp+arg_0], 65
jnz short loc_401787
```

loc_401787:

```
cmp [ebp+arg_0], 66
jnz short loc_4017A5
```

loc_4017A5:

```
cmp [ebp+arg_0], 67
jnz short loc_4017C3
```

loc_4017C3:

```
cmp [ebp+arg_0], 68
jnz short loc_4017E1
```

loc_4017E1:

```
cmp [ebp+arg_0], 69
jnz short loc_4017F9
```

loc_4017F9:

```
cmp [ebp+arg_0], 70
jnz short loc_401817
```

loc_401817:

```
cmp [ebp+arg_0], 71
jnz short loc_401835
```

loc_401835:

```
cmp [ebp+arg_0], 72
jnz short loc_401853
```

loc_401853:

```
cmp [ebp+arg_0], 73
jnz short loc_401871
```

loc_401871:

```
cmp [ebp+arg_0], 74
jnz short loc_401889
```

loc_401889:

```
cmp [ebp+arg_0], 75
jnz short loc_4018A7
```

loc_4018A7:

```
cmp [ebp+arg_0], 76
jnz short loc_4018C5
```

loc_4018C5:

```
cmp [ebp+arg_0], 77
jnz short loc_4018E3
```

loc_4018E3:

```
cmp [ebp+arg_0], 78
jnz short loc_4018F1
```

loc_4018F1:

```
cmp [ebp+arg_0], 79
jnz short loc_401909
```

loc_401909:

```
cmp [ebp+arg_0], 80
jnz short loc_401927
```

loc_401927:

```
cmp [ebp+arg_0], 81
jnz short loc_401945
```

loc_401945:

```
cmp [ebp+arg_0], 82
jnz short loc_401963
```

loc_401963:

```
cmp [ebp+arg_0], 83
jnz short loc_401981
```

loc_401981:

```
cmp [ebp+arg_0], 84
jnz short loc_401999
```

loc_401999:

```
cmp [ebp+arg_0], 85
jnz short loc_4019B7
```

loc_4019B7:

```
cmp [ebp+arg_0], 86
jnz short loc_4019D5
```

loc_4019D5:

```
cmp [ebp+arg_0], 87
jnz short loc_4019F3
```

loc_4019F3:

```
cmp [ebp+arg_0], 88
jnz short loc_401A11
```

loc_401A11:

```
cmp [ebp+arg_0], 89
jnz short loc_401A29
```

loc_401A29:

```
cmp [ebp+arg_0], 90
jnz short loc_401A47
```

loc_401A47:

```
cmp [ebp+arg_0], 91
jnz short loc_401A65
```

loc_401A65:

```
cmp [ebp+arg_0], 92
jnz short loc_401A83
```

loc_401A83:

```
cmp [ebp+arg_0], 93
jnz short loc_401AA1
```

loc_401AA1:

```
cmp [ebp+arg_0], 94
jnz short loc_401AB9
```

loc_401AB9:

```
cmp [ebp+arg_0], 95
jnz short loc_401AD7
```

loc_401AD7:

```
cmp [ebp+arg_0], 96
jnz short loc_401AE5
```

loc_401AE5:

```
cmp [ebp+arg_0], 97
jnz short loc_401AF3
```

loc_401AF3:

```
cmp [ebp+arg_0], 98
jnz short loc_401B11
```

loc_401B11:

```
cmp [ebp+arg_0], 99
jnz short loc_401B29
```

loc_401B29:

```
cmp [ebp+arg_0], 100
jnz short loc_401B47
```

loc_401B47:

```
cmp [ebp+arg_0], 101
jnz short loc_401B65
```

loc_401B65:

```
cmp [ebp+arg_0], 102
jnz short loc_401B83
```

loc_401B83:

```
cmp [ebp+arg_0], 103
jnz short loc_401BA1
```

loc_401BA1:

```
cmp [ebp+arg_0], 104
jnz short loc_401BB9
```

loc_401BB9:

```
cmp [ebp+arg_0], 105
jnz short loc_401BD7
```

loc_401BD7:

```
cmp [ebp+arg_0], 106
jnz short loc_401BF5
```

loc_401BF5:

```
cmp [ebp+arg_0], 107
jnz short loc_401C13
```

loc_401C13:

```
cmp [ebp+arg_0], 108
jnz short loc_401C31
```

loc_401C31:

```
cmp [ebp+arg_0], 109
jnz short loc_401C49
```

loc_401C49:

```
cmp [ebp+arg_0], 110
jnz short loc_401C67
```

loc_401C67:

```
cmp [ebp+arg_0], 111
jnz short loc_401C85
```

loc_401C85:

```
cmp [ebp+arg_0], 112
jnz short loc_401CA3
```

loc_401CA3:

```
cmp [ebp+arg_0], 113
jnz short loc_401CC1
```

loc_401CC1:

```
cmp [ebp+arg_0], 114
jnz short loc_401CE1
```

loc_401CE1:

```
cmp [ebp+arg_0], 115
jnz short loc_401CF9
```

loc_401CF9:

```
cmp [ebp+arg_0], 116
jnz short loc_401D17
```

loc_401D17:

```
cmp [ebp+arg_0], 117
jnz short loc_401D35
```

loc_401D35:

```
cmp [ebp+arg_0], 118
jnz short loc_401D53
```

loc_401D53:

```
cmp [ebp+arg_0], 119
jnz short loc_401D71
```

loc_401D71:

```
cmp [ebp+arg_0], 120
jnz short loc_401D89
```

loc_401D89:

```
cmp [ebp+arg_0], 121
jnz short loc_401DA7
```

loc_401DA7:

```
cmp [ebp+arg_0], 122
jnz short loc_401DC5
```

loc_401DC5:

```
cmp [ebp+arg_0], 123
jnz short loc_401DE3
```

loc_401DE3:

```
cmp [ebp+arg_0], 124
jnz short loc_401E01
```

loc_401E01:

```
cmp [ebp+arg_0], 125
jnz short loc_401E19
```

loc_401E19:

```
cmp [ebp+arg_0], 126
jnz short loc_401E37
```

loc_401E37:

```
cmp [ebp+arg_0], 127
jnz short loc_401E55
```

loc_401E55:

```
cmp [ebp+arg_0], 128
jnz short loc_401E73
```

loc_401E73:

```
cmp [ebp+arg_0], 129
jnz short loc_401E91
```

loc_401E91:

```
cmp [ebp+arg_0], 130
jnz short loc_401EAF
```

loc_401EAF:

```
cmp [ebp+arg_0], 131
jnz short loc_401ED5
```

loc_401ED5:

```
cmp [ebp+arg_0], 132
jnz short loc_401EF3
```

loc_401EF3:

```
cmp [ebp+arg_0], 133
jnz short loc_401F11
```

loc_401F11:

```
cmp [ebp+arg_0], 134
jnz short loc_401F29
```

loc_401F29:

```
cmp [ebp+arg_0], 135
jnz short loc_401F47
```

loc_401F47:

```
cmp [ebp+arg_0], 136
jnz short loc_401F65
```

loc_401F65:

```
cmp [ebp+arg_0], 137
jnz short loc_401F83
```

loc_401F83:

```
cmp [ebp+arg_0], 138
jnz short loc_401FA1
```

loc_401FA1:

```
cmp [ebp+arg_0], 139
jnz short loc_401FB9
```

loc_401FB9:

```
cmp [ebp+arg_0], 140
jnz short loc_401FD7
```

loc_401FD7:

```
cmp [ebp+arg_0], 141
jnz short loc_401FF5
```

loc_401FF5:

```
cmp [ebp+arg_0], 142
jnz short loc_402013
```

loc_402013:

```
cmp [ebp+arg_0], 143
jnz short loc_402031
```

loc_402031:

```
cmp [ebp+arg_0], 144
jnz short loc_402049
```

loc_402049:

```
cmp [ebp+arg_0], 145
jnz short loc_402067
```

loc_402067:

```
cmp [ebp+arg_0], 146
jnz short loc_402085
```

loc_402085:

```
cmp [ebp+arg_0], 147
jnz short loc_4020A3
```

loc_4020A3:

```
cmp [ebp+arg_0], 148
jnz short loc_4020C1
```

loc_4020C1:

```
cmp [ebp+arg_0], 149
jnz short loc_4020E1
```

loc_4020E1:

```
cmp [ebp+arg_0], 150
jnz short loc_4020F9
```

loc_4020F9:

```
cmp [ebp+arg_0], 151
jnz short loc_402117
```

loc_402117:

```
cmp [ebp+arg_0], 152
jnz short loc_402135
```

loc_402135:

```
cmp [ebp+arg_0], 153
jnz short loc_402153
```

loc_402153:

```
cmp [ebp+arg_0], 154
jnz short loc_402171
```

loc_402171:

```
cmp [ebp+arg_0], 155
jnz short loc_402189
```

loc_402189:

```
cmp [ebp+arg_0], 156
jnz short loc_4021A7
```

loc_4021A7:

```
cmp [ebp+arg_0], 157
jnz short loc_4021C5
```

loc_4021C5:

```
cmp [ebp+arg_0], 158
jnz short loc_4021E3
```

loc_4021E3:

```
cmp [ebp+arg_0], 159
jnz short loc_4021F1
```

loc_4021F1:

```
cmp [ebp+arg_0], 160
jnz short loc_402209
```

loc_402209:

```
cmp [ebp+arg_0], 161
jnz short loc_402227
```

loc_402227:

```
cmp [ebp+arg_0], 162
jnz short loc_402245
```

loc_402245:

```
cmp [ebp+arg_0], 163
jnz short loc_402263
```

loc_402263:

```
cmp [ebp+arg_0], 164
jnz short loc_402281
```

loc_402281:

```
cmp [ebp+arg_0], 165
jnz short loc_402299
```

loc_402299:

```
cmp [ebp+arg_0], 166
jnz short loc_4022B7
```

loc_4022B7:

```
cmp [ebp+arg_0], 167
jnz short loc_4022D5
```

loc_4022D5:

```
cmp [ebp+arg_0], 168
jnz short loc_4022F3
```

loc_4022F3:

```
cmp [ebp+arg_0], 169
jnz short loc_402311
```

loc_402311:

```
cmp [ebp+arg_0], 170
jnz short loc_402329
```

loc_402329:

```
cmp [ebp+arg_0], 171
jnz short loc_402347
```

loc_402347:

```
cmp [ebp+arg_0], 172
jnz short loc_402365
```

loc_402365:

```
cmp [ebp+arg_0], 173
jnz short loc_402383
```

loc_402383:

```
cmp [ebp+arg_0], 174
jnz short loc_4023A1
```

loc_4023A1:

```
cmp [ebp+arg_0], 175
jnz short loc_4023C1
```

loc_4023C1:

```
cmp [ebp+arg_0], 176
jnz short loc_4023E1
```

loc_4023E1:

```
cmp [ebp+arg_0], 177
jnz short loc_4023F9
```

loc_4023F9:

```
cmp [ebp+arg_0], 178
jnz short loc_402417
```

loc_402417:

```
cmp [ebp+arg_0], 179
jnz short loc_402435
```

loc_402435:

```
cmp [ebp+arg_0], 180
jnz short loc_402453
```

loc_402453:

```
cmp [ebp+arg_0], 181
jnz short loc_402471
```

loc_402471:

```
cmp [ebp+arg_0], 182
jnz short loc_402489
```

loc_402489:

```
cmp [ebp+arg_0], 183
jnz short loc_4024A7
```

loc_4024A7:

```
cmp [ebp+arg_0], 184
jnz short loc_4024C5
```

loc_4024C5:

```
cmp [ebp+arg_0], 185
jnz short loc_4024E3
```

loc_4024E3:

```
cmp [ebp+arg_0], 186
jnz short loc_4024F1
```

loc_4024F1:

```
cmp [ebp+arg_0], 187
jnz short loc_402509
```

loc_402509:

```
cmp [ebp+arg_0], 188
jnz short loc_402527
```

loc_402527:

```
cmp [ebp+arg_0], 189
jnz short loc_402545
```

loc_402545:

```
cmp [ebp+arg_0], 190
jnz short loc_402563
```

loc_402563:

```
cmp [ebp+arg_0], 191
jnz short loc_402581
```

loc_402581:

```
cmp [ebp+arg_0], 192
jnz short loc_402599
```

loc_402599:

```
cmp [ebp+arg_0], 193
jnz short loc_4025B7
```

loc_4025B7:

```
cmp [ebp+arg_0], 194
jnz short loc_4025D5
```

loc_4025D5:

```
cmp [ebp+arg_0], 195
jnz short loc_4025F3
```

loc_4025F3:

```
cmp [ebp+arg_0], 196
jnz short loc_402611
```

loc_402611:

```
cmp [ebp+arg_0], 197
jnz short loc_402629
```

loc_402629:

```
cmp [ebp+arg_0], 198
jnz short loc_402647
```

loc_402647:

```
cmp [ebp+arg_0], 199
jnz short loc_402665
```

loc_402665:

```
cmp [ebp+arg_0], 200
jnz short loc_402683
```

loc_402683:

```
cmp [ebp+arg_0], 201
jnz short loc_4026A1
```

loc_4026A1:

```
cmp [ebp+arg_0], 202
jnz short loc_4026C1
```

loc_4026C1:

```
cmp [ebp+arg_0], 203
jnz short loc_4026E1
```

loc_4026E1:

```
cmp [ebp+arg_0], 204
jnz short loc_4026F9
```

loc_4026F9:

```
cmp [ebp+arg_0], 205
jnz short loc_402717
```

loc_402717:

```
cmp [ebp+arg_0], 206
jnz short loc_402735
```

loc_402735:

```
cmp [ebp+arg_0], 207
jnz short loc_402753
```

loc_402753:

```
cmp [ebp+arg_0], 208
jnz short loc_402771
```

loc_402771:

```
cmp [ebp+arg_0], 209
jnz short loc_402789
```

loc_402789:

```
cmp [ebp+arg_0], 210
jnz short loc_4027A7
```

loc_4027A7:

```
cmp [ebp+arg_0], 211
jnz short loc_4027C5
```

loc_4027C5:

```
cmp [ebp+arg_0], 212
jnz short loc_4027E3
```

loc_4027E3:

```
cmp [ebp+arg_0], 213
jnz short loc_4027F1
```

loc_4027F1:

```
cmp [ebp+arg_0], 214
jnz short loc_402809
```

loc_402809:

```
cmp [ebp+arg_0], 215
jnz short loc_402827
```

loc_402827:

```
cmp [ebp+arg_0], 216
jnz short loc_402845
```

loc_402845:

```
cmp [ebp+arg_0], 217
jnz short loc_402863
```

loc_402863:

```
cmp [ebp+arg_0], 218
jnz short loc_402881
```

loc_402881:

```
cmp [ebp+arg_0], 219
jnz short loc_402899
```

loc_402899:

```
cmp [ebp+arg_0], 220
jnz short loc_4028B7
```

loc_4028B7:

```
cmp [ebp+arg_0], 221
jnz short loc_4028D5
```

loc_4028D5:

```
cmp [ebp+arg_0], 222
jnz short loc_4028F3
```

loc_4028F3:

```
cmp [ebp+arg_0], 223
jnz short loc_402911
```

loc_402911:

```
cmp [ebp+arg_0], 224
jnz short loc_402929
```

loc_402929:

```
cmp [ebp+arg_0], 225
jnz short loc_402947
```

loc_402947:

```
cmp [ebp+arg_0], 226
jnz short loc_402965
```

loc_402965:

```
cmp [ebp+arg_0], 227
jnz short loc_402983
```

loc_402983:

```
cmp [ebp+arg_0], 228
jnz short loc_4029A1
```

loc_4029A1:

```
cmp [ebp+arg_0], 229
jnz short loc_4029C1
```

loc_4029C1:

```
cmp [ebp+arg_0], 230
jnz short loc_4029E1
```

loc_4029E1:

```
cmp [ebp+arg_0], 231
jnz short loc_4029F9
```

loc_4029F9:

```
cmp [ebp+arg_0], 232
jnz short loc_402A17
```

loc_402A17:

```
cmp [ebp+arg_0], 233
jnz short loc_402A35
```

loc_402A35:

```
cmp [ebp+arg_0], 234
jnz short loc_402A53
```

loc_402A53:

```
cmp [ebp+arg_0], 235
jnz short loc_402A71
```

loc_402A71:

```
cmp [ebp+arg_0], 236
jnz short loc_402A89
```

loc_402A89:

```
cmp [ebp+arg_0], 237
jnz short loc_402AA7
```

loc_402AA7:

```
cmp [ebp+arg_0], 238
jnz short loc_402AC5
```

loc_402AC5:

```
cmp [ebp+arg_0], 239
jnz short loc_402AE3
```

loc_402AE3:

```
cmp [ebp+arg_0], 240
jnz short loc_402AF1
```

loc_402AF1:

```
cmp [ebp+arg_0], 241
jnz short loc_402B09
```

loc_402B09:

```
cmp [ebp+arg_0], 242
jnz short loc_402B27
```

loc_402B27:

```
cmp [ebp+arg_0], 243
jnz short loc_402B45
```

loc_402B45:

```
cmp [ebp+arg_0], 244
jnz short loc_402B63
```

loc_402B63:

```
cmp [ebp+arg_0], 245
jnz short loc_402B81
```

loc_402B81:

```
cmp [ebp+arg_0], 246
jnz short loc_402B99
```

loc_402B99:

```
cmp [ebp+arg_0], 247
jnz short loc_402BB7
```

loc_402BB7:

```
cmp [ebp+arg_0], 248
jnz short loc_402BD5
```

loc_402BD5:

```
cmp [ebp+arg_0], 249
jnz short loc_402BF3
```

loc_402BF3:

```
cmp [ebp+arg_0], 250
jnz short loc_402C11
```

loc_402C11:

```
cmp [ebp+arg_0], 251
jnz short loc_402C29
```

loc_402C29:

```
cmp [ebp+arg_0], 252
jnz short loc_402C47
```

loc_402C47:

```
cmp [ebp+arg_0], 253
jnz short loc_402C65
```

loc_402C65:

```
cmp [ebp+arg_0], 254
jnz short loc_402C83
```

loc_402C83:

```
cmp [ebp+arg_0], 255
jnz short loc_402CA1
```

loc_402CA1:

```
cmp [ebp+arg_0], 256
jnz short loc_402CC1
```

loc_402CC1:

```
cmp [ebp+arg_0], 257
jnz short loc_402CE1
```

loc_402CE1:

```
cmp [ebp+arg_0], 258
jnz short loc_402CF9
```

loc_402CF9:

```
cmp [ebp+arg_0], 259
jnz short loc_402D17
```

loc_402D17:

```
cmp [ebp+arg_0], 260
jnz short loc_402D35
```

loc_402D35:

```
cmp [ebp+arg_0], 261
jnz short loc_402D53
```

loc_402D53:

```
cmp [ebp+arg_0], 262
jnz short loc_402D71
```

loc_402D71:

```
cmp [ebp+arg_0], 263
jnz short loc_402D89
```

loc_402D89:

```
cmp [ebp+arg_0], 264
jnz short loc_402DA7
```

loc_402DA7:

```
cmp [ebp+arg_0], 265
jnz short loc_402DC5
```

loc_402DC5:

```
cmp [ebp+arg_0], 266
jnz short loc_402DE3
```

loc_402DE3:

```
cmp [ebp+arg_0], 267
jnz short loc_402DF1
```

loc_402DF1:

```
cmp [ebp+arg_0], 268
jnz short loc_402E09
```

loc_402E09:

```
cmp [ebp+arg_0], 269
jnz short loc_402E27
```

loc_402E27:

```
cmp [ebp+arg_0], 270
jnz short loc_402E45
```

loc_402E45:

```
cmp [ebp+arg_0], 271
jnz short loc_402E63
```

loc_402E63:

```
cmp [ebp+arg_0], 272
jnz short loc_402E81
```

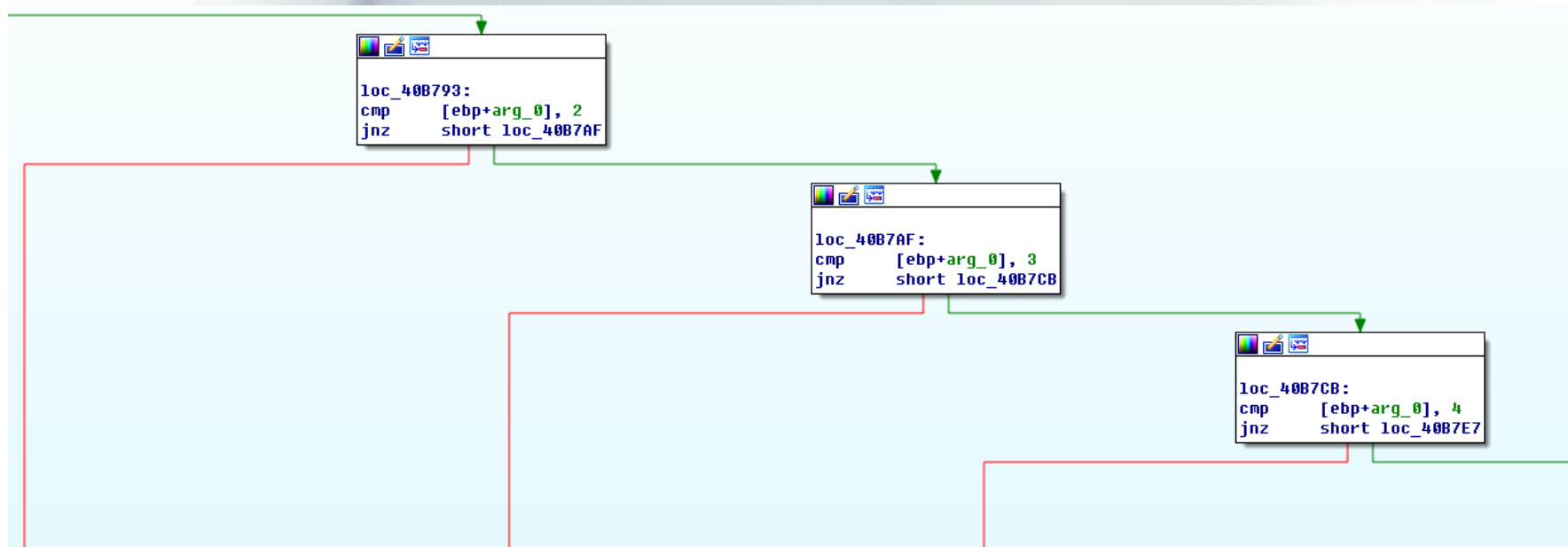
loc_402E81:

```
cmp [ebp+arg_0], 273
jnz short loc_402E99
```

<

二. 流程控制语句

○ Jnz 条件转移指令。结果不为零（或不相等）则转移



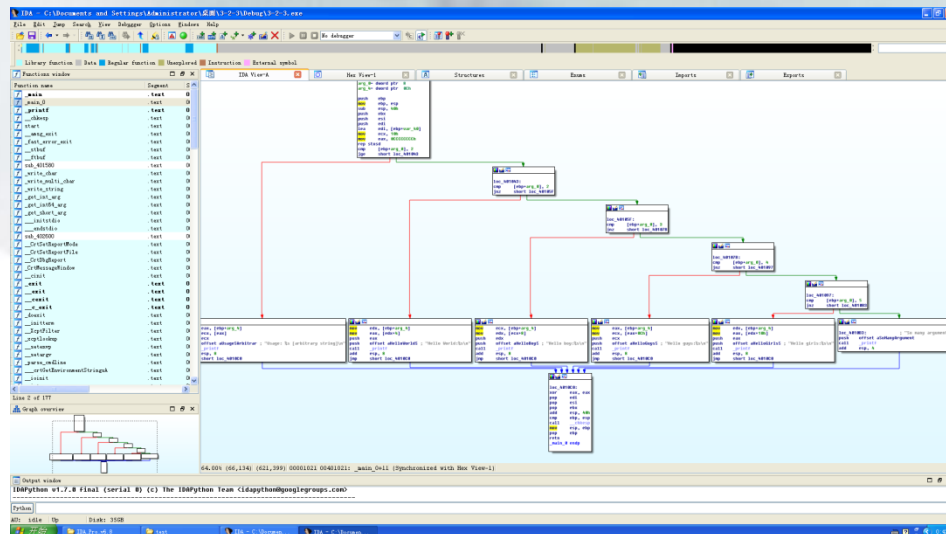
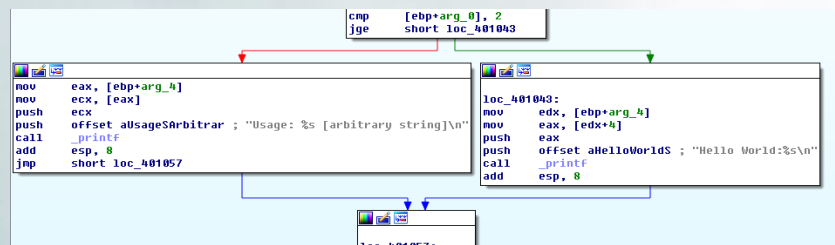
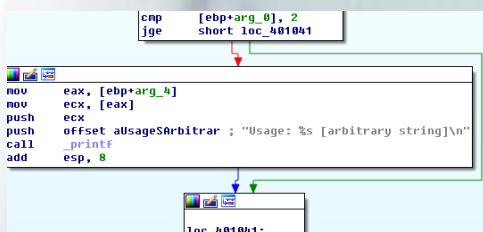
二. 流程控制语句

- 通过 **if else if else** 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，每个判断语句做为其中的一条分支。



二. 流程控制语句

④ 通过if, if else, if else 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, else if, 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

3-2-4

```
Int main(int argc, char *argv[])
{
    switch(argc)
    {
        case 1:
            printf("Usage: %s [arbitrary string]\n", argv[0]);
            break;
        case 2:
            printf("Hello World:%s\n", argv[1]);
            break;
        case 3:
            printf("Hello boy:%s\n", argv[2]);
            break;
        case 4:
            printf("Hello guys:%s\n", argv[3]);
            break;
        case 5:
            printf("Hello girls:%s\n", argv[4]);
        default:
            printf("So many arguments!\n");
    };
    return 0;
}
```

北邮网安学院 崔宝江



二. 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	S
_main	.text	0
_main_0	.text	0
_printf	.text	0
_chkexp	.text	0
start	.text	0
_ansg_exit	.text	0
_fast_error_exit	.text	0
_stbuf	.text	0
_ftbuf	.text	0
sub_401590	.text	0
_write_char	.text	0
_write_multi_char	.text	0
_write_string	.text	0
_get_int_arg	.text	0
_get_int64_arg	.text	0
_get_short_arg	.text	0
_initstdio	.text	0
_endstdio	.text	0
sub_402610	.text	0
_CrSetReportMode	.text	0
_CrSetReportFile	.text	0
_CrDbgReport	.text	0
_CrMessageWindow	.text	0
_cinit	.text	0
_exit	.text	0
_c_exit	.text	0
_doexit	.text	0
_initterm	.text	0
_XcptFilter	.text	0
_xcplookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse_cmdline	.text	0
_crtGetEnvironmentStringsA	.text	0
_iointit	.text	0

IDA View-A

Hex View-1

Structures

Enums

Imports

Exports

Line 2 of 177

Graph overview

64.00% (-88,11) (979,405) 00001010 00401010: _main_0 (Synchronized with Hex View-1)

Output window

Python 0.7.2.0 Find (Serial 0) (C) The IDApython team (idapython@googlegroups.com)

AU: idle Up Disk: 35GB

开始 IDA_Pro_v6.8 test 3-2 IDA - C:\Documen... IDA - C:\Documen...

0:52

Switch statement analysis:

```
loc_401007: ; jumptable 00401000 case 0
mov     eax, [ebp+arg_4]
mov     ecx, [eax]
push    ecx
push    offset aUsageIsArbitraryString ; "Usage: %s [arbitrary string]\n"
call    _printf
add     esp, 8
jmp     short loc_40100F

loc_40100C: ; jumptable 00401000 case 1
mov     edx, [ebp+arg_4]
mov     ecx, [edx+4]
push    ecx
push    offset aHelloWorld ; "Hello World!\n"
call    _printf
add     esp, 8
jmp     short loc_40100F

loc_40100E: ; jumptable 00401000 case 2
mov     edx, [ebp+arg_4]
mov     ecx, [ecx+8]
push    ecx
push    offset aHelloBoy ; "Hello boy!\n"
call    _printf
add     esp, 8
jmp     short loc_40100F

loc_401008: ; jumptable 00401000 case 3
mov     eax, [ebp+arg_4]
mov     ecx, [eax+8]
push    ecx
push    offset aHelloGuys ; "Hello guys!\n"
call    _printf
add     esp, 8
jmp     short loc_40100F

loc_401002: ; jumptable 00401000 default case
push    offset aHelloGirls ; "Hello girls!\n"
call    _printf
add     esp, 8
jmp     short loc_40100F

loc_40100F:
xor     eax, eax
pop     edi
pop     esi
pop     ebx
add     esp, 4h
cmp     ebp, esp
call    _chown
mov     esp, ebp
pop     ebp
ret     0
```

二. 流程控制语句

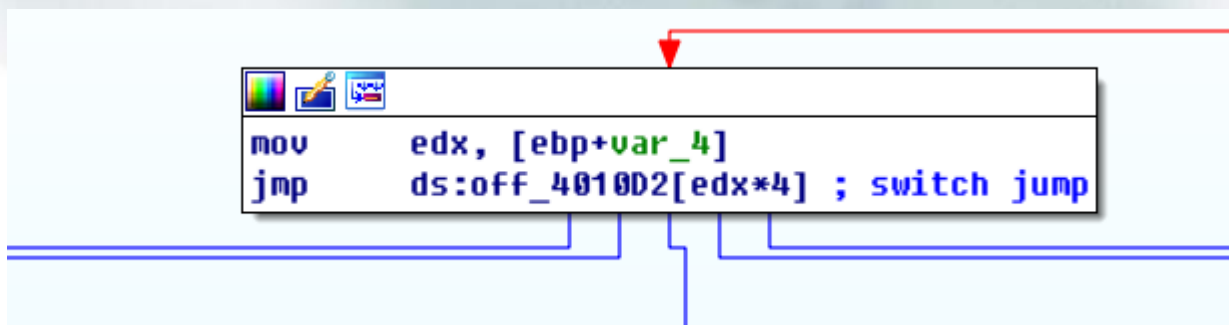
- 从控制流程图上将其与上一小节讲的**if else if**作一下比较
- **switch case**控制块的左边有点类似于分发器，由一个基本块来决定执行哪一块函数功能。



二. 流程控制语句

④ 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_4010D2**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

3-2-5

```
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");
    };
    return 0;
}
```



二. 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	S
_main	.text	0
_main_0	.text	0
_printf	.text	0
_chkexp	.text	0
start	.text	0
_ansg_exit	.text	0
_fast_error_exit	.text	0
_stbuf	.text	0
_ftbuf	.text	0
sub_401520	.text	0
_write_char	.text	0
_write_multi_char	.text	0
_write_string	.text	0
_get_int_arg	.text	0
_get_int64_arg	.text	0
_get_short_arg	.text	0
_ini_stdio	.text	0
_end_stdio	.text	0
sub_4025A0	.text	0
_CrtSetReportMode	.text	0
_CrtSetReportFile	.text	0
_CrtDbgReport	.text	0
_CrMessageWindow	.text	0
_cinit	.text	0
_exit	.text	0
_cexit	.text	0
_c_exit	.text	0
_doexit	.text	0
_initterm	.text	0
_XcptFilter	.text	0
_xcptlookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse_cmdline	.text	0
_crtGetEnvironmentStringsA	.text	0
_j0init	.text	0

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

Line 2 of 177

Graph overview

100.00% (-128,198) | (1316,639) | 00001010 00401010: _main_0 (Synchronized with Hex View-1)

Output window

IDA Python v1.7.0 Final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

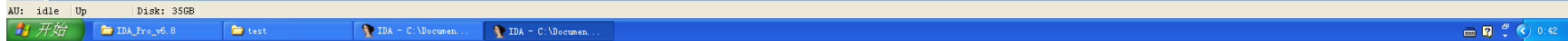
Python

AU: idle Up Disk: 35GB

开始 IDA_Pro_v6.8 test 3-2 IDA - C:\Documen... IDA - C:\Documen...

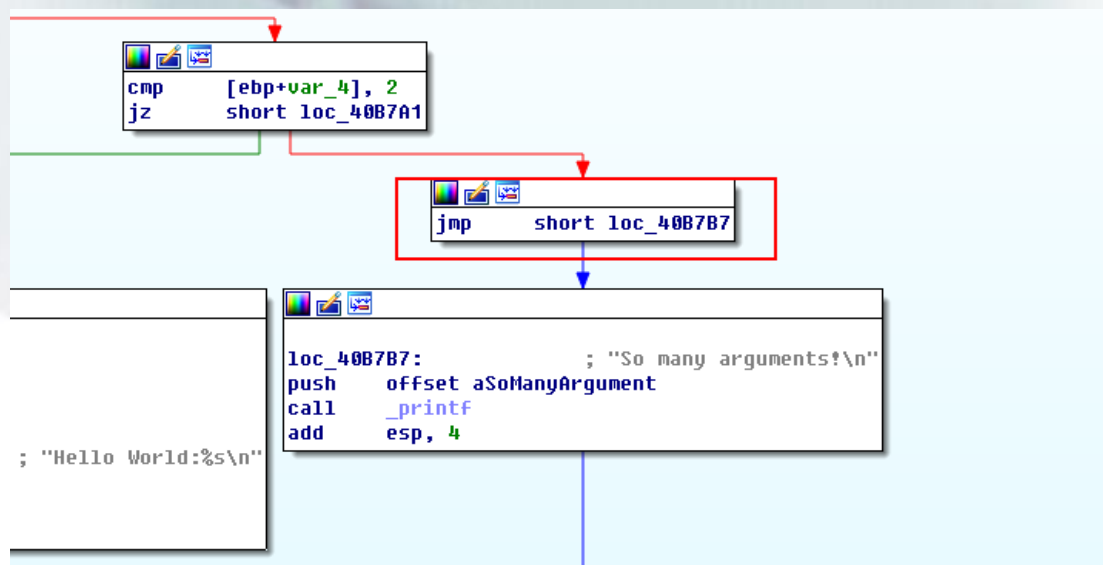
1:01

④ 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

④ 非线性case值 (vc++ 6.0 debug版本)



二. 流程控制语句

3-2-6

```
Int main(int argc, char *argv[])
{
    switch(argc)
    {
        case 1:
            printf("Usage: %s [arbitrary string]\n", argv[0]);
            break;
        case 2:
            printf("Hello World:%s\n", argv[1]);
            break;
        case 3:
            printf("Hello boy:%s\n", argv[2]);
            break;
        case 4:
            printf("Hello guys:%s\n", argv[3]);
            break;
        case 255:
            printf("Hello girls:%s\n", argv[4]);
            break;
        default:
            printf("So many arguments!\n");
    };
    return 0;
}
```



二. 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

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
_main	.text	0
_main_0	.text	0
_printf	.text	0
_chkexp	.text	0
_start	.text	0
_amsg_exit	.text	0
_fast_error_exit	.text	0
_stbuf	.text	0
_ftbuf	.text	0
sub_401620	.text	0
_write_char	.text	0
_write_multi_char	.text	0
_write_string	.text	0
_get_int_arg	.text	0
_get_int64_arg	.text	0
_get_short_arg	.text	0
_ini_stdio	.text	0
_end_stdio	.text	0
sub_402760	.text	0
_CrtSetReportMode	.text	0
_CrtSetReportFile	.text	0
_CrtDbgReport	.text	0
_CrMessageWindow	.text	0
_cinit	.text	0
_exit	.text	0
_cexit	.text	0
_c_exit	.text	0
_doexit	.text	0
_initterm	.text	0
_XcptFilter	.text	0
_xcplookump	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse_cmdline	.text	0
_crtGetEnvironmentStringsA	.text	0
_ioinit	.text	0

Line 2 of 177

Graph overview

64.00% (223,-76) (943,632) 00001010 00401010: _main_0 (Synchronized with Hex View-1)

Output window

IDA Python 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

```
; Attributes: bp-based frame
_main_0 proc near
var_4h= byte ptr -4h
var_8= dword ptr -8
arg_0= dword ptr 0
arg_4= dword ptr 4h

push    ebp
mov     ebp, esp
sub     esp, 4h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_4h]
mov     ecx, 11h
mov     eax, 0CCCCCCC
rep stsd
mov     eax, [ebp+arg_0]
mov     [ebp+var_8], eax
mov     ecx, [ebp+var_4]
sub     ecx, 1
mov     [ebp+var_4], ecx
cmp     [ebp+var_4], 0
jnz     short loc_40100F ; jumpable 0040100B default case

; case 0
loc_401007:
mov     eax, [ebp+arg_4]
mov     ecx, [eax+4]
push    offset aHelloWorldS ; "Hello World!\n"
call    _printf
add     esp, 8
jmp     short loc_40100C

; case 2
loc_40100D:
mov     edx, [ebp+arg_4]
mov     eax, [edx+8]
push    offset aHelloBoyS ; "Hello boy!\n"
call    _printf
add     esp, 8
jmp     short loc_40100C

; case 3
loc_401009:
mov     ecx, [ebp+arg_4]
mov     edx, [ecx+0Ch]
push    offset aHelloGays ; "Hello gays!\n"
call    _printf
add     esp, 8
jmp     short loc_40100C

; case 254
loc_401000:
mov     eax, [ebp+arg_4]
mov     ecx, [eax+10h]
push    offset aHelloGirlsS ; "Hello girls!\n"
call    _printf
add     esp, 8
jmp     short loc_40100C

; default case
loc_40100F:
push    offset aSoManyArgument
call    _printf
add     esp, 4

loc_40100C:
mov     eax, ecx
pop     edi
pop     esi
pop     ebx
add     esp, 4h
cmp     esp, ebp
call    _chkexp
mov     esp, ebp
pop     ebp
retn
_main_0 endp
```

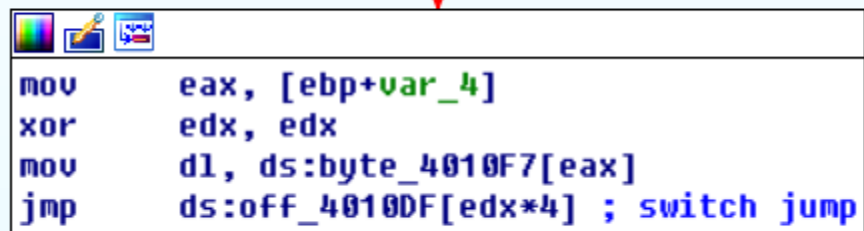
二. 流程控制语句

- 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。

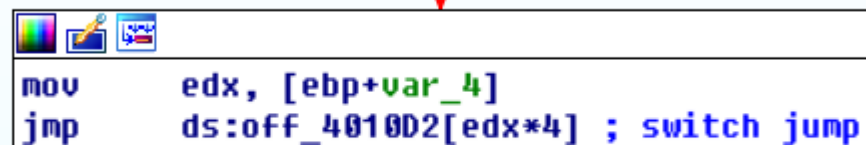


二. 流程控制语句

□但是负责分发的那个基础块则有很大的不同



```
mov     eax, [ebp+var_4]
xor     edx, edx
mov     dl, ds:byte_4010F7[eax]
jmp     ds:off_4010DF[edx*4] ; switch jump
```



```
mov     edx, [ebp+var_4]
jmp     ds:off_4010D2[edx*4] ; switch jump
```

[illegible]

二. 流程控制语句

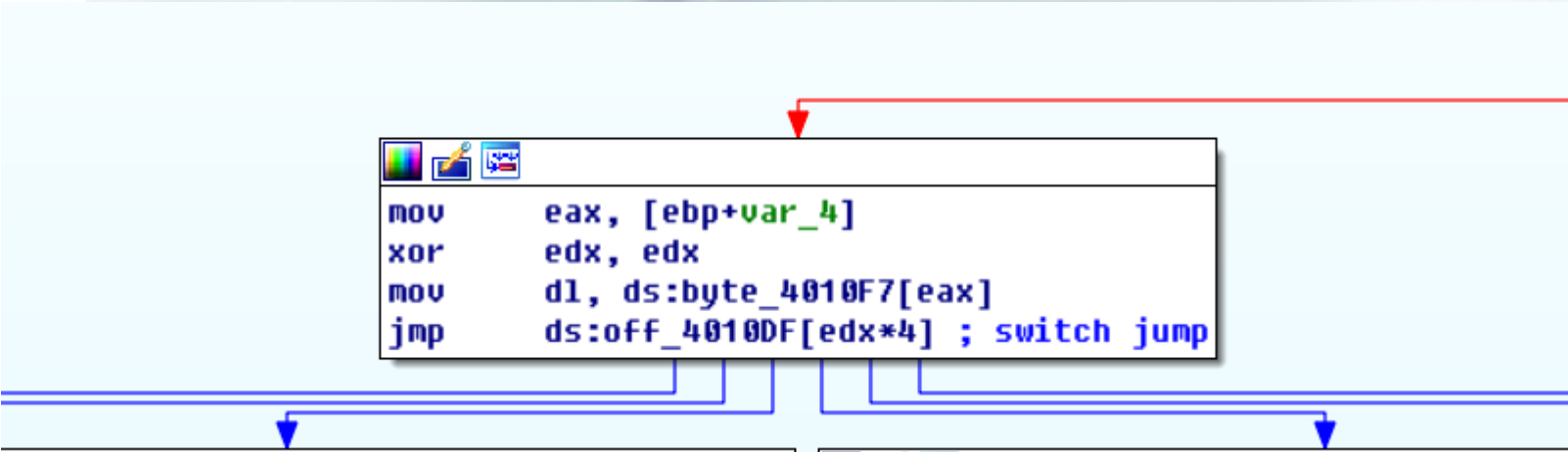
- 这张表由**255**个元素组成，数组的大小由我们**case**值最大的确定即**255**
- 程序根据输入的**case**值，在第一张表中取索引值，然后在第二个地址数组中，获取相应的地址进行跳转

```
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。



```
mov     eax, [ebp+var_4]
xor     edx, edx
mov     dl, ds:byte_4010F7[eax]
jmp     ds:off_4010DF[edx*4] ; switch jump
```



二. 流程控制语句

④ 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)

3-2-7

```
Int main()  
{  
    int i=100;  
    while(i--)  
    {  
  
    }  
    return 0;  
}
```



二. 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

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
_main	.text	0
_main_0	.text	0
start	.text	0
_msg_exit	.text	0
_fast_error_exit	.text	0
_cinit	.text	0
_exit	.text	0
_cexit	.text	0
_c_exit	.text	0
_doexit	.text	0
_initterm	.text	0
_XcptFilter	.text	0
_xcptlookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse cmdline	.text	0
_crGetEnvironmentStringsA	.text	0
_ioinit	.text	0
_ioterm	.text	0
rub_4021C0	.text	0
rub_402220	.text	0
rub_402450	.text	0
_global_unwind2	.text	0
_unwind_handler	.text	0
_local_unwind2	.text	0
_abnormal_termination	.text	0
_NLG_Notify	.text	0
_except_handler3	.text	0
_seh_longjmp_unwind(x)	.text	0
_FF_MSGBANNER	.text	0
_NMSG_WRITE	.text	0
_GET_RTERMSG	.text	0
_malloc	.text	0
_malloc_dbg	.text	0
_nh_malloc	.text	0
_nh_malloc_dbg	.text	0

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...

1:28

二. 流程控制语句

- ④ 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)

3-2-8

```
Int main()  
{  
    int i=100;  
    for(;i;i--)  
    {  
  
    }  
    return 0;  
}
```



二. 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

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
_main	.text	0
_main_0	.text	0
start	.text	0
_msg_exit	.text	0
_fast_error_exit	.text	0
_cinit	.text	0
_exit	.text	0
_exits	.text	0
_c_exits	.text	0
_c_exits	.text	0
_doexit	.text	0
_initterm	.text	0
_xcpFilter	.text	0
_xcpLookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse cmdline	.text	0
_cr(GetEnvironmentStringsA	.text	0
_joinit	.text	0
_ioterm	.text	0
sub_402100	.text	0
sub_402220	.text	0

Line 2 of 173

Graph overview

Output window

Python 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

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)

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)

3-2-9

```
Int main()  
{  
    int i=100;  
    do  
    {  
        i--;  
    }while(i);  
    return 0;  
}
```



二. 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

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
_main	.text	0
_main_0	.text	0
start	.text	0
_msg_exit	.text	0
_fast_error_exit	.text	0
_cinit	.text	0
_exit	.text	0
_c_exit	.text	0
_c_exit	.text	0
_doexit	.text	0
_initterm	.text	0
_XcptFilter	.text	0
_xcptlookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse_cmdline	.text	0
_crtGetEnvironmentStringsA	.text	0
_j0init	.text	0
_i0term	.text	0
sub_4021C0	.text	0
sub_402220	.text	0
sub_402450	.text	0
_global_unwind2	.text	0
_unwind_handler	.text	0
_local_unwind2	.text	0
_abnormal_termination	.text	0
_NLS_Notify	.text	0
_except_handler3	.text	0
_seh_longjmp_unwind(x)	.text	0
_FF_MSGBANNER	.text	0
_NMSG_WRITE	.text	0
_GET_RTERMSG	.text	0
_malloc	.text	0
_malloc_dbg	.text	0
_nh_malloc	.text	0
_nh_malloc_dbg	.text	0

IDA View-A

Hex View-1

Structures

Enums

Imports

Exports

Line 2 of 173

Graph overview

Output window

IDA Python 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

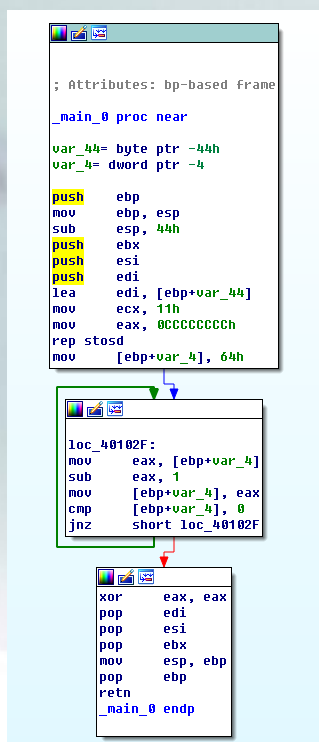
```
; Attributes: bp-based frame
_main_0 proc near
var_44= byte ptr -44h
var_4= dword ptr -4
push ebp
mov ebp, esp
sub esp, 44h
push ebx
push esi
push edi
lea edi, [ebp+var_44]
mov ecx, 11h
mov eax, 0CCCCCCCCh
rep stosd
mov [ebp+var_4], 64h

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
```

二. 流程控制语句

② 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

- ④ 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

- ① 一. 基本数据类型
- ② 二. 流程控制语句
- ③ 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
- ❑ 全局变量则属于进程作用域
- ❑ 静态变量存在于程序的整个生命周期，作用域可以为
 - 进程全局
 - 文件内部
 - 函数内部



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已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)

3-3-1

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



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_8], 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_8], 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_8       = dword ptr -08h
.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_8], 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_8], 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)

3-3-3

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



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 ; DATA XREF: _main_0+181r
```



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

④ 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)

3-3-5

```
void testStaticVar(int i)
{
    static int staticVar=i;
    static int 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");
    return 0;
}
```



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

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
main	.text	0
sub_40100A	.text	0
sub_401020	.text	0
_main_0	.text	0
_printf	.text	0
_chxesp	.text	0
_system	.text	0
start	.text	0
_msg_exit	.text	0
_fast_error_exit	.text	0
_stbuf	.text	0
_ftbuf	.text	0
_output	.text	0
_write_char	.text	0
_write_multi_char	.text	0
_write_string	.text	0
_get_int_arg	.text	0
_get_int64_arg	.text	0
_get_short_arg	.text	0
_initstdio	.text	0
_endstdio	.text	0
sub_4020B0	.text	0
_CrSetReportMode	.text	0
_CrSetReportFile	.text	0
_CrDbgReport	.text	0
_CrUMessageWindow	.text	0
_spawnvpe	.text	0
_cinit	.text	0
_exit	.text	0
_cexit	.text	0
_e_exit	.text	0
_doexit	.text	0
_initterm	.text	0
_spawnve	.text	0
_comexcmd	.text	0
_access	.text	0

IDA View-A

Hex View-1

Structures

Enums

Imports

Exports

```
push edi
lea edi, [ebp+var_40]
mov ecx, 10h
mov eax, 0CCCCCCh
rep stosd
xor eax, eax
mov al, byte_4255D8
and eax, 1
test eax, eax
jnz short loc_40105E

mov c1, byte_4255D8
or c1, 1
mov byte_4255D8, c1
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 c1, byte_4255D8
or c1, 2
mov byte_4255D8, c1
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. 变量表现

Sub-401020子函数

- **Test**对两个参数(目标, 源)执行**AND**逻辑操作, 并根据结果设置标志寄存器, 结果本身不会保存。
- **TEST AX,BX** 与 **AND AX,BX** 命令有相同效果, 只是**Test**指令不改变**AX**和**BX**的内容, 而**AND**指令会把结果保存到**AX**中
- 左分支: **byte_4255D8**最低位为零, **and**后为零, **test**后为0, **jnz**则不转移, 通过**or c1,1**, 将**byte_4255D8**最低位置为1.
- 右分支: **byte_4255D8**最低位为1, **and**后为1, **test**后为1, **jnz**则转移

北邮网

```
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，赋值的算法则不相同。
- 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

```
00401000 .data:004255D8 byte_4255D8 db 0 ; DATA XREF: sub_401020+1A1r  
00401001
```



小结

④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。



练习

- ④ 编译13个C语言文件，利用IDA分析基本数据类型、流程控制语句、变量表现形式的汇编代码。





Q & A

谢谢!

