



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



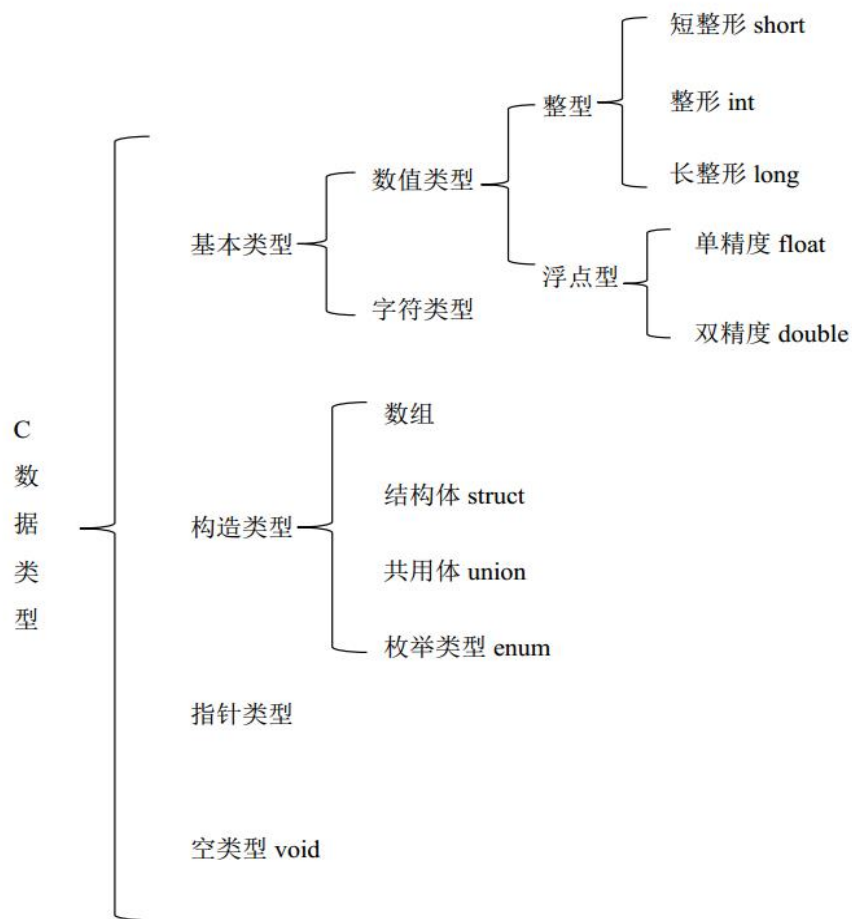
第三章从C语言看汇编

- 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

@ 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

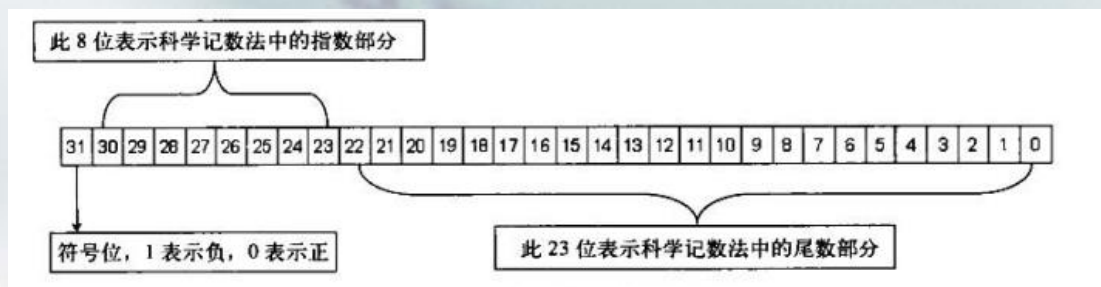
④ 以浮点数**400.1**和**500.02**为例

- ❑ 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

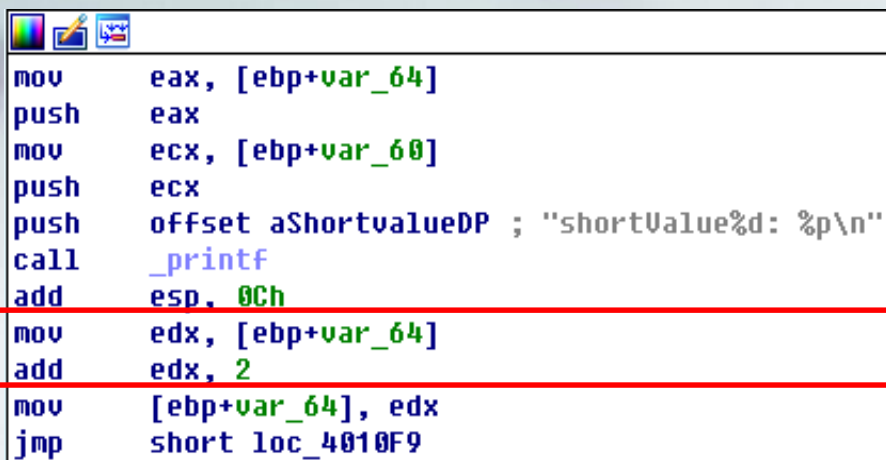
□ 例

- 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

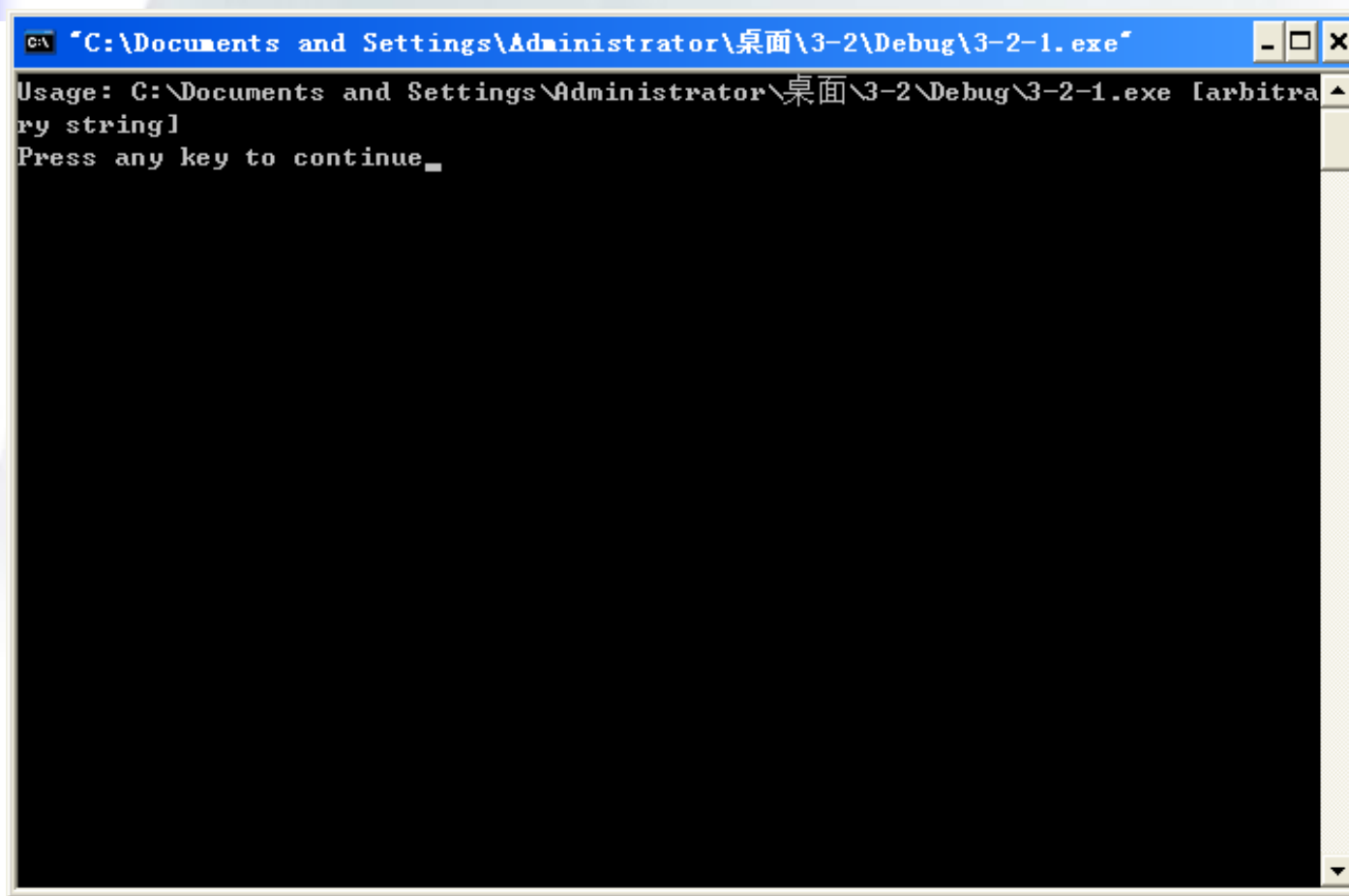
@if语句 (vc++ 6.0 debug版本)

3-2-1

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



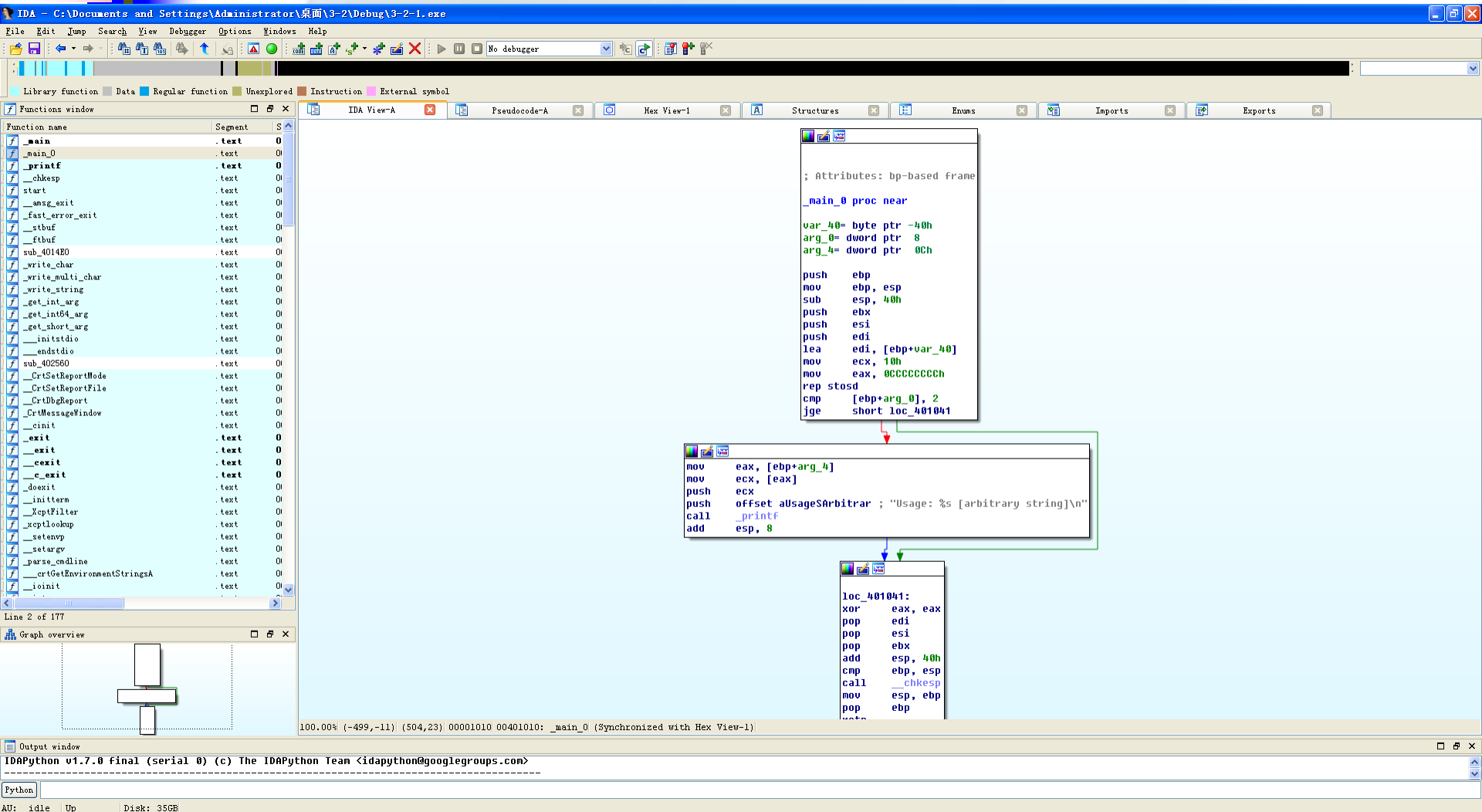
二. 流程控制语句



```
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_
```

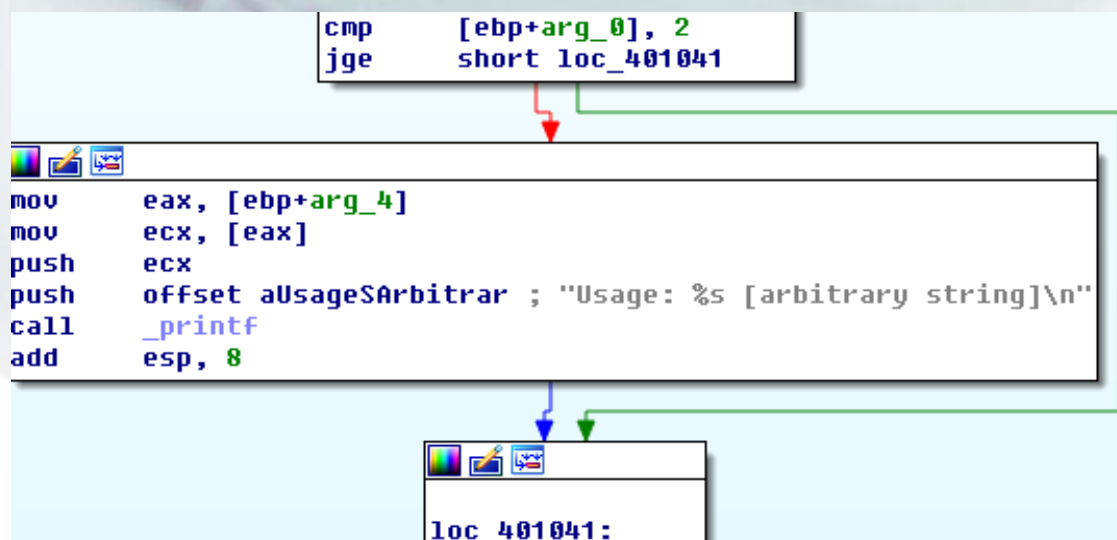


二. 流程控制语句



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 the file `C:\Documents and Settings\Administrator\桌面\3-2-2\Debug\3-2-2.exe`. The left sidebar shows the **Functions window** with a list of functions including `_main`, `_main_0`, `_printf`, `_chkesp`, `start`, `_msg_exit`, `_fast_error_exit`, `_stbuf`, `_ftbuf`, `sub_4014F0`, `_write_char`, `_write_multi_char`, `_write_string`, `_get_int_arg`, `_get_int04_arg`, `_get_short_arg`, `_initstdio`, `_endstdio`, `sub_402570`, `_CrISetReportMode`, `_CrISetReportFile`, `_CrIDbgReport`, `_CrIMessageWindow`, `_cinit`, `_exit`, `_cexit`, `_c_exit`, `_doesit`, `_initterm`, `_XcptFilter`, `_xceptlookup`, `_setenv`, `_setargv`, `_parse_cmdline`, `_CrIGetEnvironmentStringsA`, and `_joinit`.

The main window shows the assembly code for the `_main_0` function, which is a `bp-based frame`. The code includes variable declarations for `var_40` (byte ptr -40h), `arg_0` (dword ptr 8), and `arg_4` (dword ptr 0Ch). The function body consists of several instructions: `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`, and `jge short loc_401043`. The control flow graph shows a jump from the `jge` instruction to `loc_401043` if the condition is met, and a fall-through path to `loc_401057` otherwise.

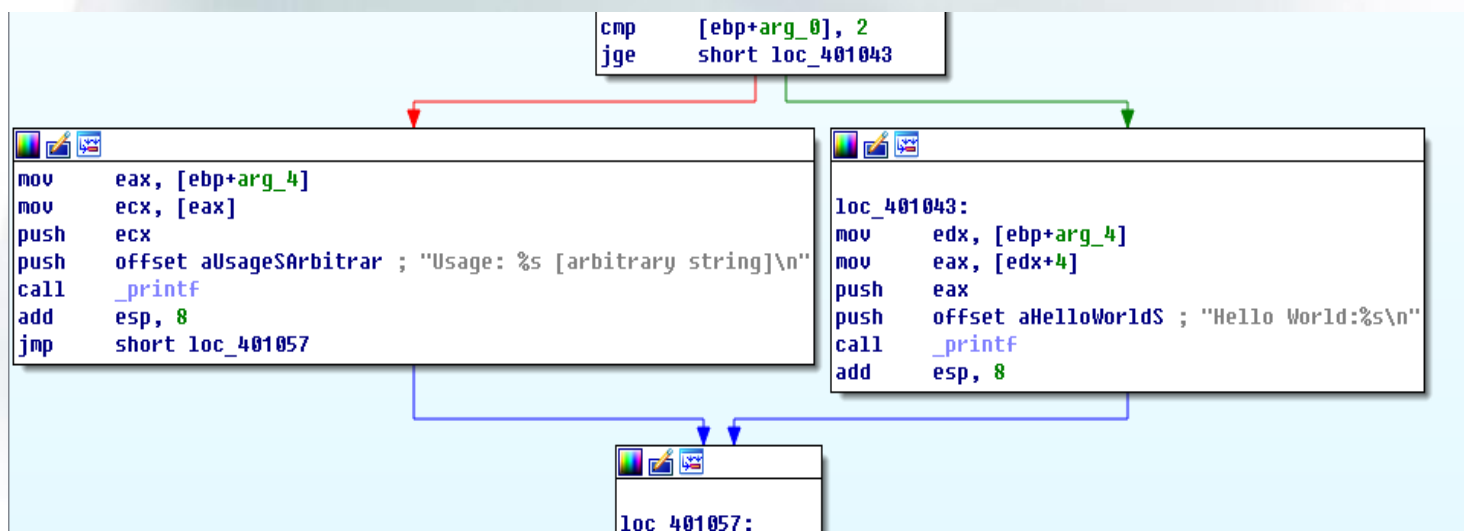
The `loc_401043` block contains the following instructions: `mov edx, [ebp+arg_4]`, `mov eax, [edx+4]`, `push eax`, `push offset aUsageSArbitrar ; "Usage: %s [arbitrary string]\n"`, `call _printf`, `add esp, 8`, and `jmp short loc_401057`.

The `loc_401057` block contains the following instructions: `xor eax, eax`, `pop edi`, `pop esi`, `pop ebx`, `add esp, 40h`, `cmp ebp, esp`, and `call _chkesp`.

The bottom status bar shows the output window with the text: `IDA Python v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>` and the Python interpreter status: `Python`. The system information at the bottom indicates: `AU: idle`, `Down`, and `Disk: 35GB`.

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

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
_linit	.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_4010A3

loc_4010A3:
cmp [ebp+arg_0], 2
jnz short loc_40105F

loc_40105F:
cmp [ebp+arg_0], 0
jnz short loc_401070

loc_401070:
cmp [ebp+arg_0], 4
jnz short loc_401097

loc_401097:
cmp [ebp+arg_0], 5
jnz short loc_4010B3

loc_4010B3:
; "So many arguments"
push offset aSoManyArgument
call _printf
add esp, 4

loc_4010C8:
xor eax, eax
pop edi
pop esi
pop ebx
add esp, 40h
cmp ebp, esp
call _chkexp
mov esp, ebp
pop ebp
retn
_main_0_end
```

Graph overview

64.00% (66,134) (621,399) 00001021 00401021: _main_0+11 (Synchronized with Hex View-1)

Output window

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

Python

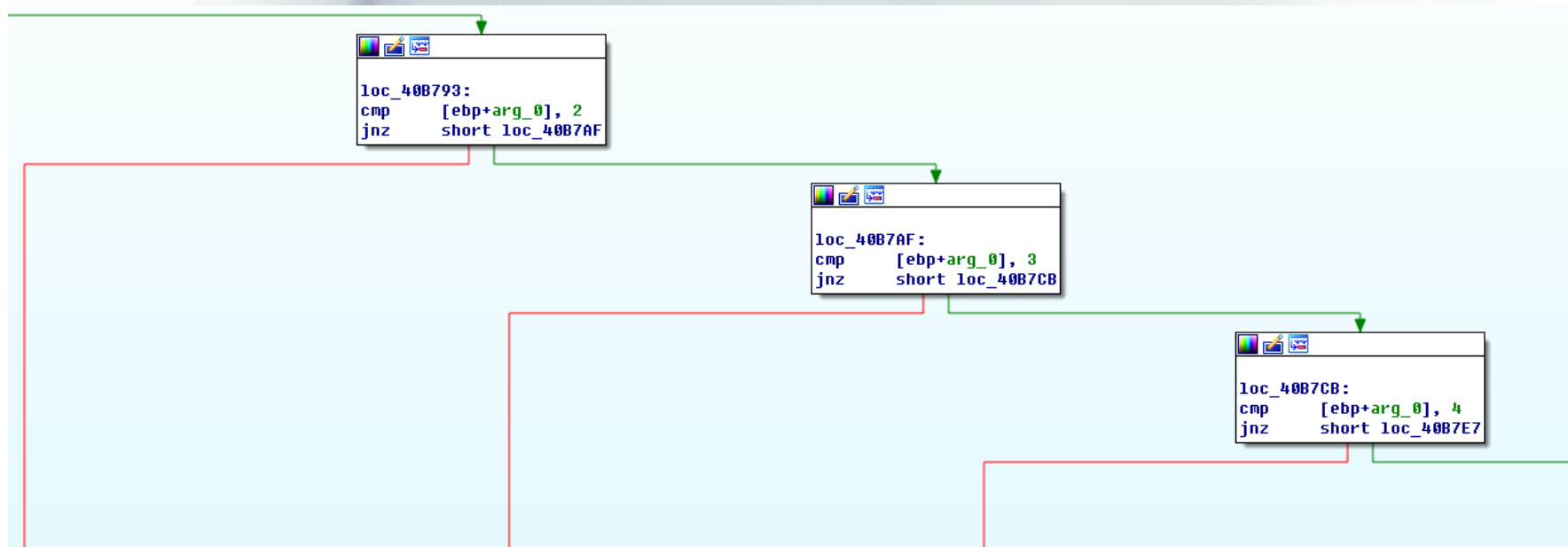
AU: idle Up Disk: 35GB

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

0:42

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



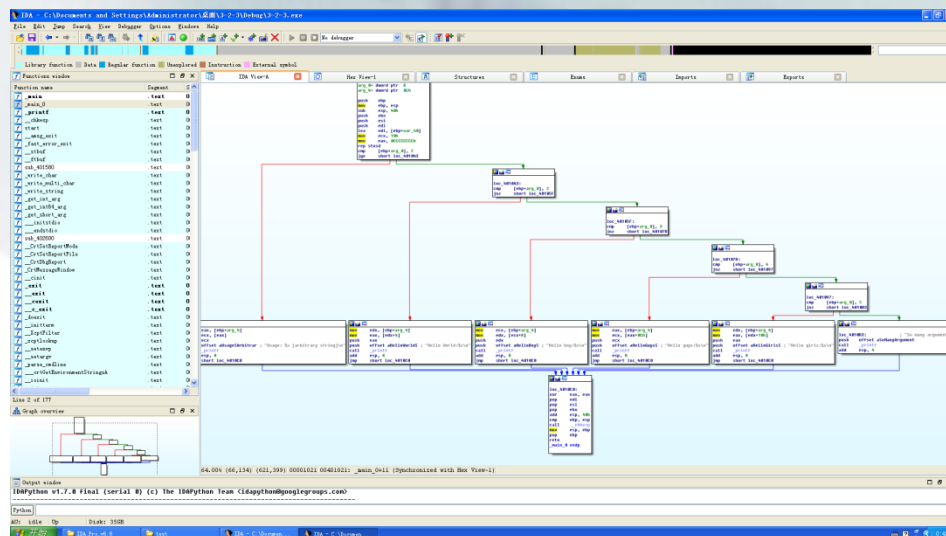
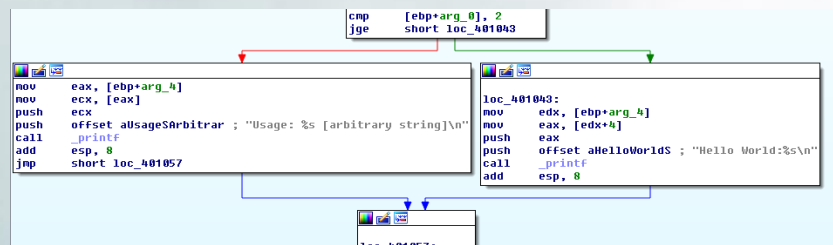
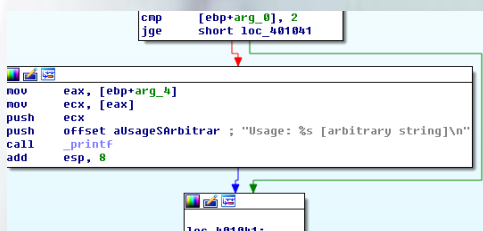
二. 流程控制语句

- 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。



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
_CrMessageBox	.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

```
; Attributes: bp-based frame
_main_0 proc near
var_4h= byte ptr -4h
var_4= dword ptr -4
arg_4= dword ptr 0
arg_4= dword ptr 0Ch
push ebp
mov ebp, esp
sub esp, 4h
push ebx
push esi
push edi
lea edi, [ebp+var_4h]
mov ecx, 1h
mov ecx, 0CCCCCCh
rep stqsd
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_A01002 ; jumpable 00A0100A default case
loc_A01007: ; jumpable 00A0100A case 0
mov edx, [ebp+arg_4]
mov ecx, [eax]
push ecx
push offset aUsageSbStritar ; "Usage: %s [arbitrary string]\n"
call _printf
add esp, 8
jmp short loc_A0100F
loc_A0100C: ; jumpable 00A0100A case 1
mov edx, [ebp+arg_4]
mov ecx, [edx+4]
push ecx
push offset aHelloWorldS ; "Hello World!\n"
call _printf
add esp, 8
jmp short loc_A0100F
loc_A0100E: ; jumpable 00A0100A case 2
mov edx, [ebp+arg_4]
mov ecx, [ecx+8]
push ecx
push offset aHelloBugsS ; "Hello bugs!\n"
call _printf
add esp, 8
jmp short loc_A0100F
loc_A01008: ; jumpable 00A0100A case 3
mov edx, [ebp+arg_4]
mov ecx, [ecx+Ch]
push ecx
push offset aHelloGuysS ; "Hello guys!\n"
call _printf
add esp, 8
jmp short loc_A0100F
loc_A01002: ; jumpable 00A0100A default case
push offset aHelloGirlsS ; "Hello girls!\n"
call _printf
add esp, 8
jmp short loc_A0100F
loc_A0100F:
xor eax, eax
pop edi
pop esi
pop ebx
add esp, 4h
cmp ebp, esp
call _chorep
mov esp, ebp
pop ebp
ret
_main_0 endp
```

Line 2 of 177

Graph overview

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

Output window

IDA Pro v7.7.0 Find (Serial 0) (c) The IDA Pro team - idapro.com

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

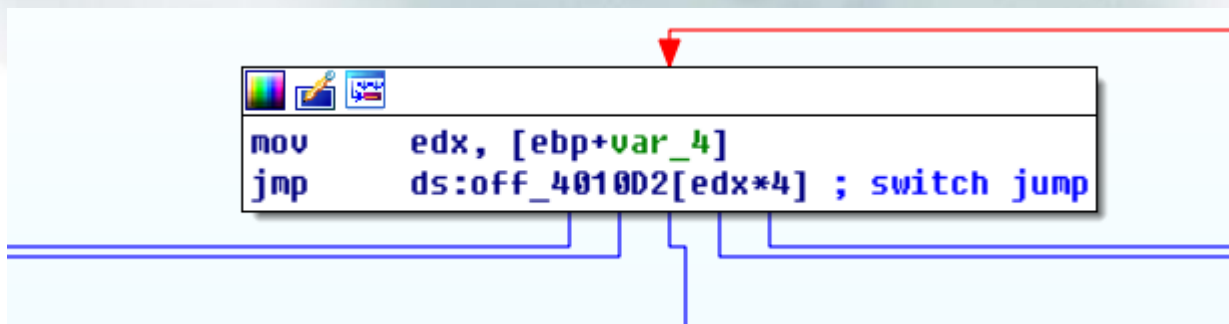
- 从控制流程图上将其与上一小节讲的**if else if**作一下比较
- **switch case**控制块的左边有点类似于分发器，由一个基本块来决定执行哪一块函数功能。



二. 流程控制语句

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

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
_iointit	.text	0

IDA View-A

```
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
```

Hex View-1

```
cmp [ebp+var_4], 2
jz short loc_401051
```

Structures

```
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
```

Enums

```
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
```

Imports

```
loc_401067:
; "So many arguments!\n"
push offset aSoManyArgument
call _printf
add esp, 4
```

Exports

```
loc_401074:
xor eax, eax
pop edi
pop esi
pop ebx
add esp, 44h
cmp ebp, esp
```

Line 2 of 177

Graph overview

Output window

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

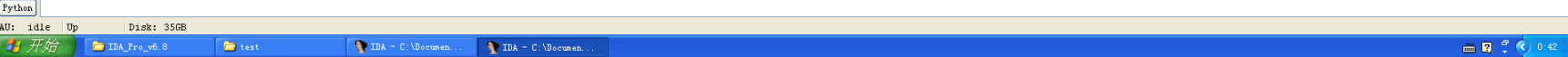
Python

AU: idle Up Disk: 35GB

开始 IDA_Pro_v6.0 test 3-2 IDA - C:\Documen... IDA - C:\Documen...

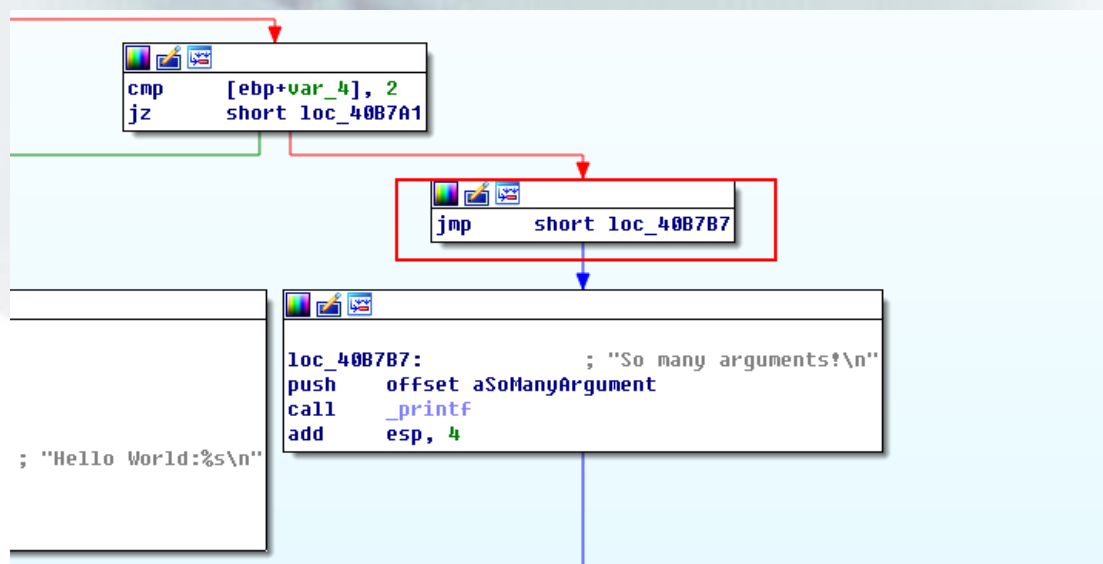
1:01

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
_xcplookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse_cmdline	.text	0
_crtGetEnvironmentStringsA	.text	0
_iointit	.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_v6.0 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
ja      short loc_40100F

mov     eax, [ebp+var_4]
mov     edx, edx
mov     di, ds:byte_40100F[edx]
jmp     ds:off_40100F[edx*4] ; switch jump

; jumpable 0040100B 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

; jumpable 0040100B case 2
loc_40107D:
mov     edx, [ebp+arg_4]
mov     eax, [edx*8]
push    ecx
push    offset aHelloWorl2S ; "Hello Worl2!\n"
call    _printf
add     esp, 8
jmp     short loc_40100C

; jumpable 0040100B case 3
loc_401092:
mov     ecx, [ebp+arg_4]
mov     edx, [ecx*0Ch]
push    ecx
push    offset aHelloWorl3S ; "Hello Worl3!\n"
call    _printf
add     esp, 8
jmp     short loc_40100C

; jumpable 0040100B case 254
loc_4010A9:
mov     eax, [ebp+arg_4]
mov     ecx, [eax*10h]
push    ecx
push    offset aHelloWorl4S ; "Hello Worl4!\n"
call    _printf
add     esp, 8
jmp     short loc_40100C

; jumpable 0040100B default case
loc_4010BF:
push    offset aSolangArgument
call    _printf
add     esp, 4
jmp     short loc_40100C

loc_4010C5:
xor     eax, eax
pop     edi
pop     esi
pop     ebx
add     esp, 4h
cmp     esp, ebp
call    _chkexp
mov     esp, ebp
pop     ebp
retn
_main_0 endp
```

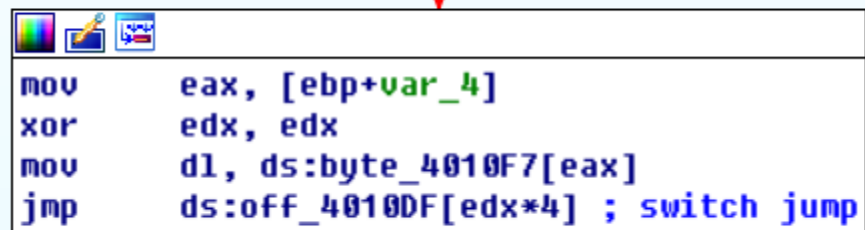
二. 流程控制语句

- 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。

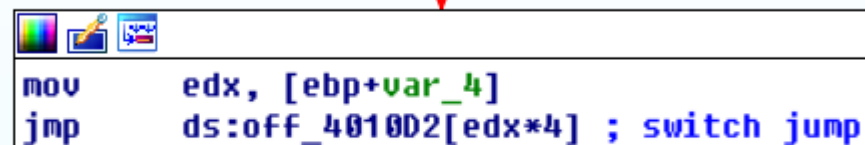


二. 流程控制语句

□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

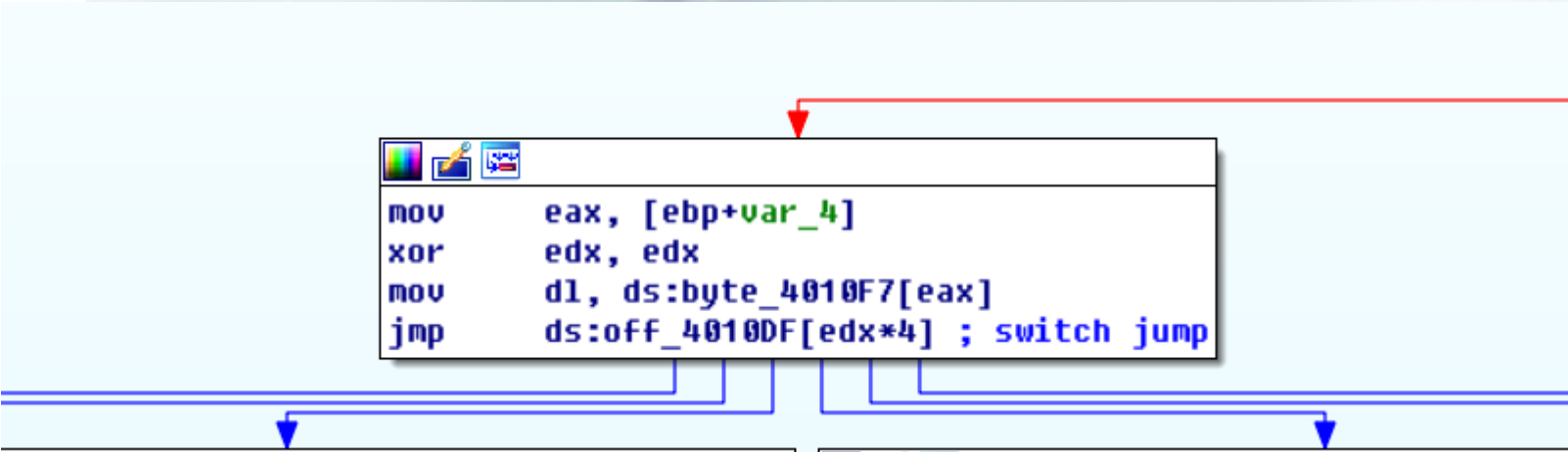
- 这张表由**255**个元素组成，数组的大小由我们**case**值最大的确定即**255**
- 程序根据输入的**case**值，在第一张表中取索引值，然后在第二个地址数组中，获取相应的地址进行跳转

```
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
_amsi_exit	.text	0
_fast_error_exit	.text	0
_cinit	.text	0
_exit	.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

not found

Graph overview

100.00% (-537,102) | (1113,382) 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_v6.0 text 3-2 IDA - C:\Documen... IDA - C:\Documen...

1:28

The screenshot displays the IDA Pro interface with the assembly view of a function named `_main_0`. The assembly code includes variable declarations for `var_44` (a byte pointer) and `var_4` (a dword pointer), followed by stack frame setup, pushing of registers, and a loop body. The control flow graph (CFG) highlights a loop structure: a block at `loc_40102F` contains the loop body instructions, which then branches to `loc_401041`. The block at `loc_401041` contains instructions to pop registers and return to `_main_0_endp`. The status bar at the bottom indicates the current instruction address is `100.00% (-537,102) | (1113,382) 00001010 00401010: _main_0`.

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
_loinit	.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
_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

Line 2 of 173

Graph overview

100.00% (-610,-27) (1054,463) 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...

1:41

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

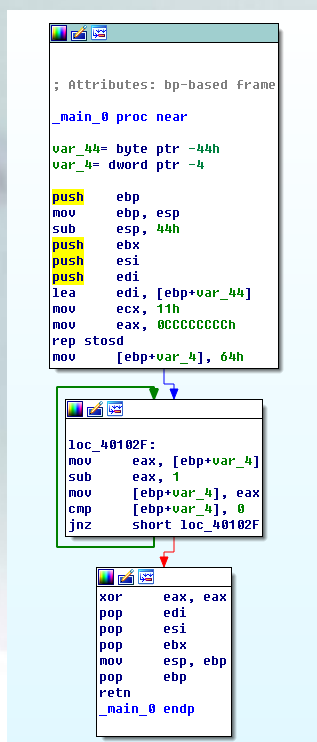
pop ebp

retn

_main_0 endp

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

- ④ (1) 栈中的局部变量
- ④ (2) 全局变量
- ④ (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
_CrMessageBox	.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
```



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!

