DataRescue sa/nv

IDA Pro

An Advanced Interactive Multi-Processor Disassembler
by Ilfak Guilfanov

IDA Pro 3.8x
QuickStart Guide

DataRescue sa/nv
45 quai de la Dérivation
4020 Liège, Belgium
tel : +32-4-3446510
fax : +32-4-3446514
www.datarescue.com/ida.htm

Copyright Notice

Trademarks

NOTE

IDA Pro incorporates compression code by the Info-ZIP group. There are no extra charges or costs due to the use of this code, and the original compression sources are freely available from CompuServe in the IBMPRO forum and by anonymous ftp from the Internet site ftp.uu.net:/pub/archiving/zip. We will also, upon request, mail you the full sources on a 3.5" MSDOS-format diskette for the cost of mailing.

Edition

Summer 1999, rev 3.8x
© 1996-99 DataRescue sa/nv
Proudly Made in Europe

Published by

DataRescue sa/nv
45 Quai de la Dérivation
4020 Liège
Belgium
+32-4-3446510 (tel)
+32-4-3446514 (fax)
http://www.datarescue.com

A few words from the team

First of all, we would like to thank you for purchasing or considering the purchase of IDA Pro. If you decide to buy IDA Pro, let us stress that we don't see this as an end, but rather as the beginning of a relationship : our goal is not only to offer timely technical support but also to respond to your future needs. That is why your feedback is so valuable to us : please fell free to contact us; IDA Pro's users have made it what it is now.

Based on your feedback, we continue to improve IDA Pro. Be sure to regularly check our web pages for enhancements, corrections and new releases. All IDA Pro customers are entitled to free updates over the Internet for one year.

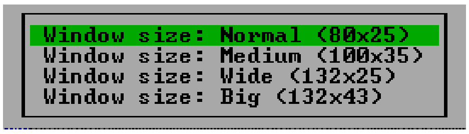
Writing a manual for IDA Pro is probably an impossible task : disassembler users are highly skilled specialists, IDA itself is hard to use, counterintuitive at times and, difficult to master. In addition, IDA Pro is so versatile that what applies to Java class disassemblies hardly matters for segmented 80x86 architectures and vice-versa. No matter how hard we try, the perfect manual is out of our reach. It is unlikely that we will ever be able to cover all your questions in advance but we are here to help you. Therefore, this startup guide does not aim to be an exhaustive introduction to IDA Pro. Rather, our hope is that it will expose the general philosophy behind its operation and help you get a faster start with IDA Pro.

Ilfak Guilfanov, Main Developer
Pierre Vandevenne, Manager

Screen Resolution

IDA Pro also runs on non-Windows platforms, that is why it is still a character mode application. The default 80x25 text screen is probably not the environment you want to work in. When it first starts, IDA Pro will offer you a choice of available resolutions.

If you run the DOS32 version of IDA Pro (IDAX), the program will adapt to any active resolution, provided it is within bounds accepted by your video card. For further configuration, you may want to examine the IDA.CFG configuration file and customize the workspace resolution to your liking.



```
Window size: Normal (80x25)
Window size: Medium (100x35)
Window size: Wide (132x25)
Window size: Big (132x43)
```

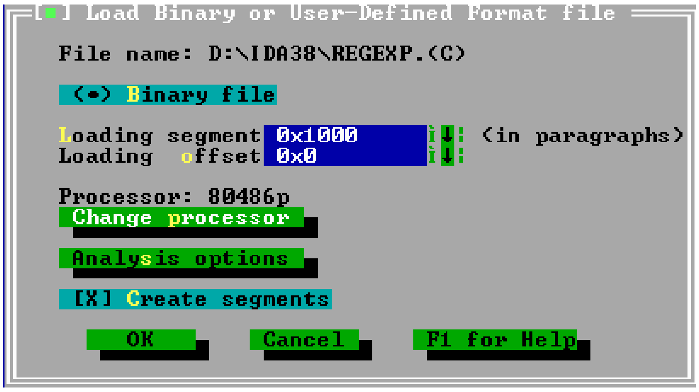
Load this file in any text editor and search for SCREEN_MODE. You'll find something like this, where

```
#ifdef __MSDOS__

SCREEN_MODE = 0 // Screen mode to use
                // 0 - don't change screen mode
                // DOS: AL for INT 10
#else

SCREEN_MODE = 0x8040 // Screen mode to use
                     // high byte/cols, low byte/rows                      //
i.e. 0x5020 is 80cols, 32rows
```

which we suggest you adapt to your need.

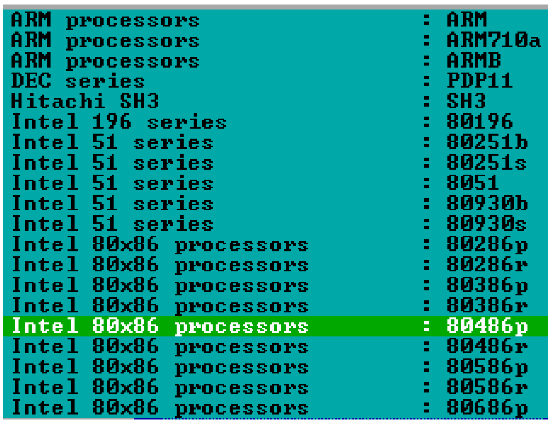
Processors, Processors

When IDA Pro loads a binary image, it will try to determine the format of the image and the processor that was targeted. If it cannot automatically make this determination, you will see the following dialog



```
[■] Load Binary or User-Defined Format file

File name: D:\IDA38\REGEXP.(C)

(•) Binary file

Loading segment 0x1000      ↓  (in paragraphs)
Loading  offset 0x0         ↓

Processor: 80486p
Change processor

Analysis options

[X] Create segments

    OK        Cancel      F1 for Help
```

You can then select the appropriate processor for your project. Some of the processors we support need to be specified explicitly, for example if you want to force the endianness (ARM) or use specific processor extensions such as MMX or 3D-Now, you will have to select them manually.

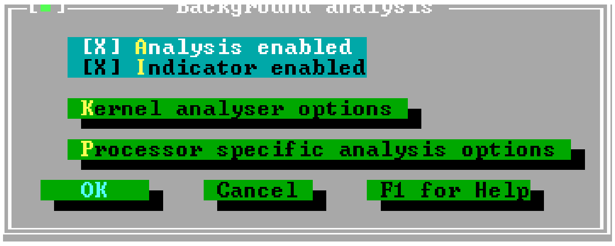


```
ARM processors                : ARM
ARM processors                : ARM710a
ARM processors                : ARMB
DEC series                    : PDP11
Hitachi SH3                   : SH3
Intel 196 series              : 80196
Intel 51 series               : 80251b
Intel 51 series               : 80251s
Intel 51 series               : 8051
Intel 51 series               : 80930b
Intel 51 series               : 80930s
Intel 80x86 processors        : 80286p
Intel 80x86 processors        : 80286r
Intel 80x86 processors        : 80386p
Intel 80x86 processors        : 80386r
Intel 80x86 processors        : 80486p
Intel 80x86 processors        : 80486r
Intel 80x86 processors        : 80586p
Intel 80x86 processors        : 80586r
Intel 80x86 processors        : 80686p
```

Often, IDA Pro will auto detect the processor type (Intel 386 in protected mode for example), the file type (Portable Executable for example) and will use the information collected from the header of the file to initiate auto-analysis. This will start exploring the obvious execution paths in the target program.

Analysis Options

Analysis options can be defined initially from this menu.



☒ Analysis enabled
☒ Indicator enabled

The defaults are usually good for most purposes and will not be reviewed in details here. Remember that all the IDA Pro analysis parameters can also be configured through the IDA Pro configuration file and the application menus. It should be noted that the configuration file is probably the best place to store settings which you frequently use.

Defining Code

Sometimes, either because the file has no specific entry point (a ROM for example) or because the automatic analysis has not found an execution path, it will be necessary to help IDA Pro. This combination of automatic analysis and human intervention is what allows IDA Pro to obtain results that the non-interactive products cannot reach.

In the following situation, assume IDA Pro hasn’t recognized that this sequence of byte is actually a meaningful code sequence. Move your cursor on the seg000:0b91 line and press C

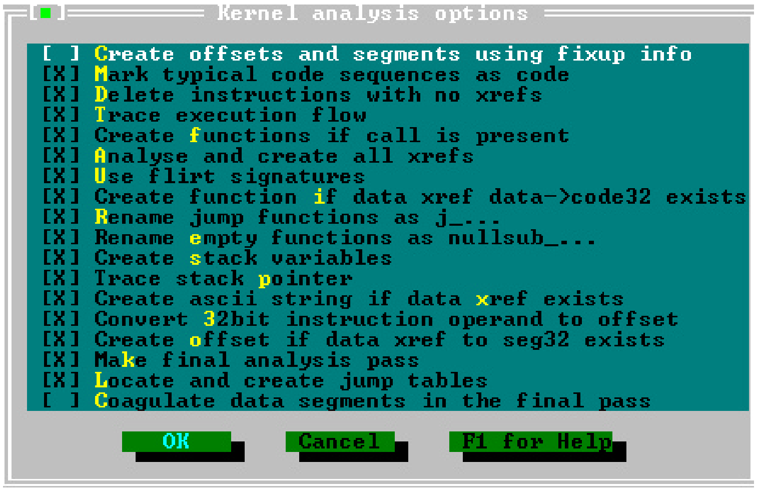
```
seg000:0B9B                 db 0B0h ; _
seg000:0B9C                 db  90h ; É
seg000:0B9D                 db  26h ; &
seg000:0B9E                 db  88h ; ê
seg000:0B9F                 db    4 ;
seg000:0BA0                 db 0BEh ; ¥
seg000:0BA1                 db    1 ;
seg000:0BA2                 db    0 ;
seg000:0BA3                 db  26h ; &
seg000:0BA4                 db  8Ah ; è
seg000:0BA5                 db    4 ;
seg000:0BA6                 db  3Ch ; <
seg000:0BA7                 db  20h ;
seg000:0BA8                 db 0C7h ; Ã
seg000:0BA9                 db    6 ;
seg000:0BAA                 db  0Fh ;
seg000:0BAB                 db    5 ;
seg000:0BAC                 db    1 ;
seg000:0BAD                 db    0 ;
seg000:0BAE                 db 0F8h ; °
seg000:0BAF                 db  0Fh ;
seg000:0BB0                 db  84h ; ä
seg000:0BB1                 db 0C1h ; -
seg000:0BB2                 db    0 ;
```

And IDA Pro converts this sequence to

```
seg000:0B9B                 mov     al, 90h
```

```
seg000:0B9D                 mov     es:[si], al
seg000:0BA0                 mov     si, 1
seg000:0BA3                 mov     al, es:[si]
seg000:0BA6                 cmp     al, 20h
seg000:0BA8                 mov     word_148_50F, 1
seg000:0BAE                 clc
seg000:0BAF                 jz      loc_0_C74
```

IDA Pro will not always automatically recognize all the code in a given program : this situation is perfectly normal. It is possible to influence how IDA Pro handles unrecognized code through the analysis option configuration panel. The kernel analysis options have an impact on the auto-analysis IDA Pro performs.



```
Kernel analysis options
[ ] Create offsets and segments using fixup info
[X] Mark typical code sequences as code
[X] Delete instructions with no xrefs
[X] Trace execution flow
[X] Create functions if call is present
[X] Analyse and create all xrefs
[X] Use flirt signatures
[X] Create function if data xref data->code32 exists
[X] Rename jump functions as j_...
[X] Rename empty functions as nullsub_...
[X] Create stack variables
[X] Trace stack pointer
[X] Create ascii string if data xref exists
[X] Convert 32bit instruction operand to offset
[X] Create offset if data xref to seg32 exists
[X] Make final analysis pass
[X] Locate and create jump tables
[ ] Coagulate data segments in the final pass
```

In most cases, the default options offer a good compromise between accuracy and convenience. If IDA Pro identified code where it should not have, it may be a good idea to try deactivating the **Make final analysis pass** option. In those situations, where some code is not identified because it is not located in expected locations, **Coagulate Data Segments** may be useful. Remember that these analysis options can also be defined through the configuration file and, in most cases, this is the best place to modify them.

**** When the input program or binary has been encrypted or compressed, IDA Pro will not be able to disassemble the part of the program that is not in clear text. In this situation, you have to solutions - either write a decryptor in IDA C or use a file unpacker to pre-process the target file.**

Pressing ‘C’ in an undefined section restarts the IDA Pro code analyzer. **All execution paths starting from the newly defined code will be explored and analyzed**. Sometimes, a simple manual code definition will help IDA Pro discover dozens of execution paths. Note : this operation will not adversely affect what you have already defined.

Defining Strings and Data

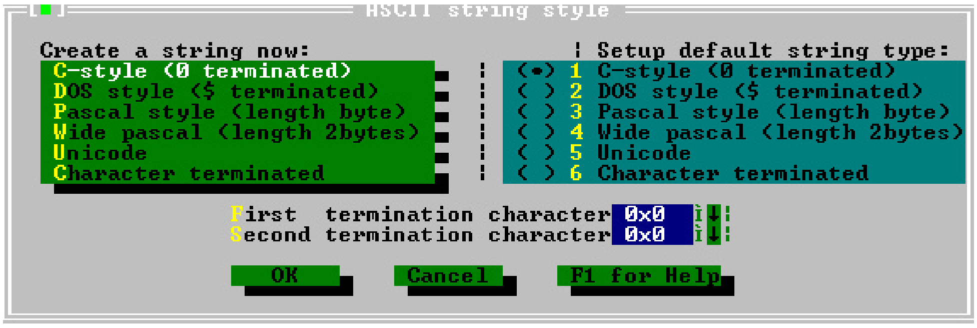
In this situation, IDA Pro failed to identify what is clearly an ASCII string. This misidentification occurred because the string is not actually directly referenced by the program

```
dseg:0146                 db  0Dh ;
dseg:0147                 db  14h ;
dseg:0148                 db  43h ; C
dseg:0149                 db  61h ; a
dseg:0149                 db  61h ; a
dseg:014A                 db  6Eh ; n
dseg:014B                 db  20h ;
dseg:014C                 db  6Eh ; n
dseg:014D                 db  6Fh ; o
dseg:014E                 db  74h ; t
dseg:014F                 db  20h ;
dseg:0150                 db  6Fh ; o
dseg:0151                 db  70h ; p
dseg:0152                 db  65h ; e
dseg:0153                 db  6Eh ; n
dseg:0154                 db  20h ;
dseg:0155                 db  66h ; f
dseg:0156                 db  69h ; i
dseg:0157                 db  6Ch ; l
dseg:0158                 db  65h ; e
dseg:0159                 db  20h ;
dseg:015A                 db  2Eh ; .
dseg:015B                 db  24h ; $
```

Move your cursor on the dseg:0148 line and press A. The string is now defined and an **automatic name** has been generated. From now on, this name will be used by all past and future references to this string, either the ones IDA Pro will discover or the ones you will tell IDA about.

```
dseg:0148 aCanNotOpenFile db 'Can not open file .$'
```

This string is $ terminated. IDA Pro usually handles most string types automatically. Special situations are best handled through the ASCII Style dialog box.



ASCII string style

Create a string now:
C-style (0 terminated)
DOS style ($ terminated)
Pascal style (length byte)
Wide pascal (length 2bytes)
Unicode
Character terminated

Setup default string type:
(•) 1 C-style (0 terminated)
( ) 2 DOS style ($ terminated)
( ) 3 Pascal style (length byte)
( ) 4 Wide pascal (length 2bytes)
( ) 5 Unicode
( ) 6 Character terminated

First termination character 0x0
Second termination character 0x0

The word at dseg:0146 is actually an attribute used when the string is displayed. Moving the cursor on that line and pressing 'D' will eventually cycle through the 'db' and the 'dw' data type. Either one could be the one you wish to define, depending on how the program actually handles those values. Had the next word been undefined, dseg:0146 could eventually have been defined as a a 'dd'. You may also define a structure.

Undefining Things

In this admittedly artificial example, a sequence of spaces has been wrongly converted to three dd's and a meaningless sequence of instructions. (these conversions do not occur anymore in IDA Pro 3.82 and up)

```
dseg:02B6                 dd 20202020h
dseg:02BA                 dd 20202020h
dseg:02BE                 dd 20202020h
dseg:02C2 ; ---------------------------------
dseg:02C2                 and     [bx+si], ah
dseg:02C4                 and     [bx+si], ah
dseg:02C6                 and     [bx+si], ah
dseg:02C8                 and     [bx+si], ah
dseg:02CA                 and     [bx+si], ah
dseg:02CC                 and     [bx+si], ah
dseg:02CE                 and     [bx+si], ah
dseg:02D0                 and     [bx+si], ah
dseg:02D2                 and     [bx+si], ah
dseg:02D4                 and     [bx+si], ah
dseg:02D6                 and     [bx+si], ah
dseg:02D8                 and     [bx+si], ah
dseg:02DA                 and     [bx+si], ah
dseg:02DC                 and     [bx+si], ah
dseg:02DE                 and     [bx+si], ah
dseg:02E0                 and     [si], ah
```

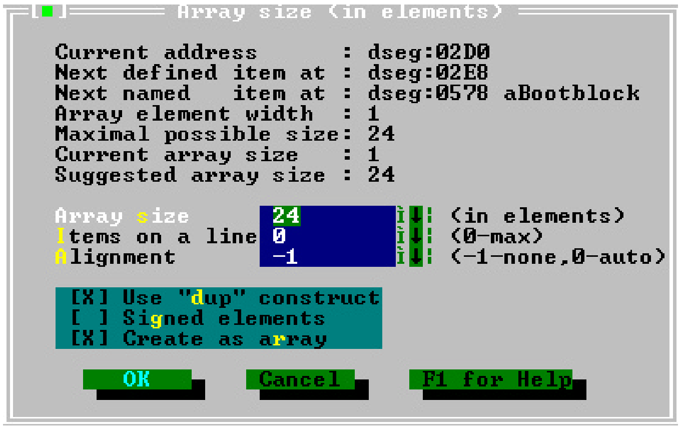
It is not possible to redefine them immediately as an ASCII string. Incorrect definitions must be **undefined** before new definitions are applied.

First move the cursor on dseg:02B6 and press 'U' to undefine all dd's in turn, then undefine the stream of instructions. Now, the 'A' key can be used to redefine the stream of 20h as an ASCII string. By now you are probably thinking that this is a bit slow. Isn't there a faster way ? You bet there is. Simply move the cursor on the first line you want to undefine, press SHIFT and DOWN ARROW simultaneously to mark the area to undefine and then press 'U'.

The Undefine command is your best friend. Although IDA Pro Is not likely to produce an output as outrageous as our example, misdefinitions can happen, particularly if data is moved around at run-time and references to some addresses are meaningless on the binary itself. Because one single change code definition can change the whole disassembly, a typical undo is not practical in IDA Pro as it would force IDA Pro to save the state of the entire disassembly, a time consuming operation.

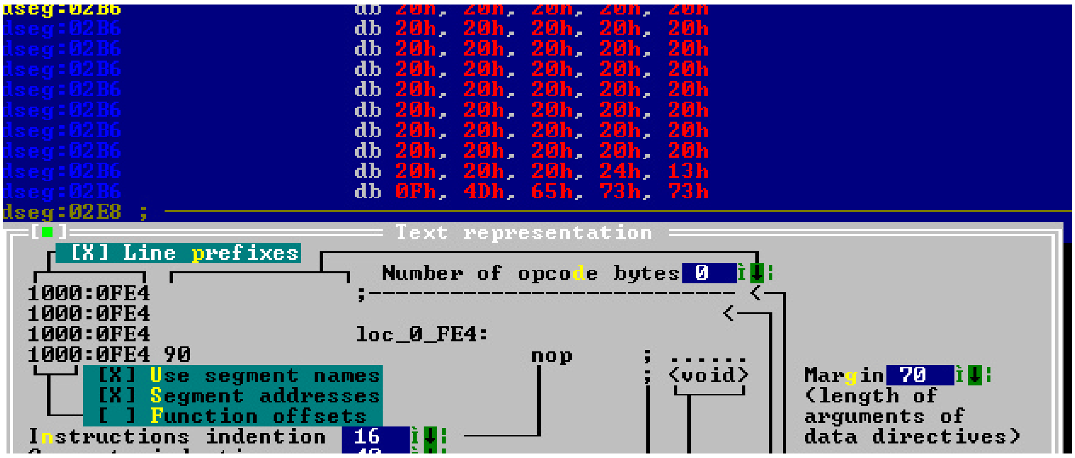
Arrays

Arrays are a fairly obvious extension to the standard data types. Their definition is straightforward and controlled by this dialog box that pops whenever you attempt to define an array.



```
Array size (in elements)

Current address       : dseg:02D0
Next defined item at  : dseg:02E8
Next named   item at  : dseg:0578 aBootblock
Array element width   : 1
Maximal possible size: 24
Current array size    : 1
Suggested array size  : 24

Array size            24          ì↓¦ (in elements)
Items on a line       0           ì↓¦ (0-max)
Alignment             -1          ì↓¦ (-1-none,0-auto)

 [X] Use "dup" construct
 [ ] Signed elements
 [X] Create as array

    OK          Cancel        F1 for Help
```

Tip ! One of the most frequently asked question about array definition is : "How do I fit more items on a line". Well, the answer is at the same time obvious and hard to find : you just increase the line length. Consider these examples :



```
dseg:02B6                 db 20h, 20h, 20h, 20h, 20h
dseg:02B6                 db 20h, 20h, 20h, 20h, 20h
dseg:02B6                 db 20h, 20h, 20h, 20h, 20h
dseg:02B6                 db 20h, 20h, 20h, 20h, 20h
dseg:02B6                 db 20h, 20h, 20h, 20h, 20h
dseg:02B6                 db 20h, 20h, 20h, 20h, 20h
dseg:02B6                 db 20h, 20h, 20h, 20h, 20h
dseg:02B6                 db 20h, 20h, 20h, 20h, 20h
dseg:02B6                 db 20h, 20h, 20h, 24h, 13h
dseg:02B6                 db 0Fh, 4Dh, 65h, 73h, 73h
dseg:02E8 ;
```

Text representation

☒ Line prefixes

Number of opcode bytes 0

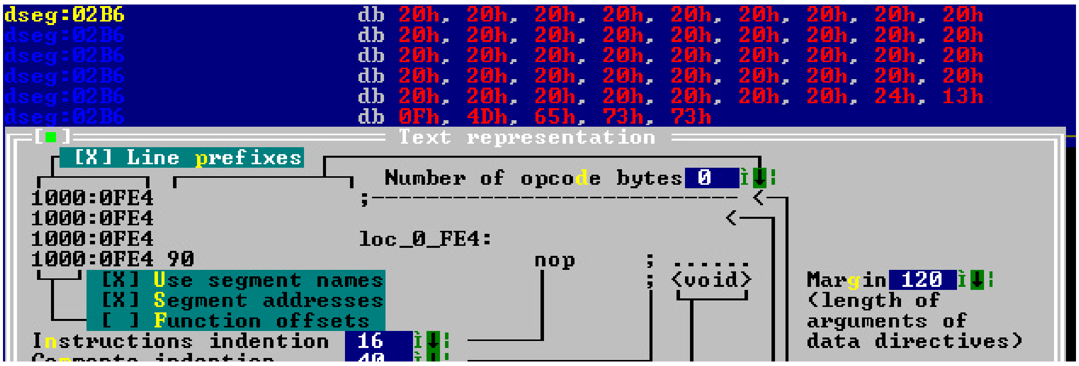
```
1000:0FE4                 ;------------------------------
1000:0FE4
1000:0FE4                 loc_0_FE4:
1000:0FE4 90                          nop       ; ......
                                                ; <void>
```

☒ Use segment names
☒ Segment addresses
☐ Function offsets

Instructions indention 16

Margin 70 (length of arguments of data directives)

Now this



```
dseg:02B6                 db 20h, 20h, 20h, 20h, 20h, 20h, 20h, 20h, 20h
dseg:02B6                 db 20h, 20h, 20h, 20h, 20h, 20h, 20h, 20h, 20h
dseg:02B6                 db 20h, 20h, 20h, 20h, 20h, 20h, 20h, 20h, 20h
dseg:02B6                 db 20h, 20h, 20h, 20h, 20h, 20h, 20h, 20h, 20h
dseg:02B6                 db 20h, 20h, 20h, 20h, 20h, 20h, 20h, 24h, 13h
dseg:02B6                 db 0Fh, 4Dh, 65h, 73h, 73h
```

Text representation

☒ Line prefixes

Number of opcode bytes 0

```
1000:0FE4                 ;------------------------------
1000:0FE4
1000:0FE4                 loc_0_FE4:
1000:0FE4 90                          nop       ; ......
                                                ; <void>
```

☒ Use segment names
☒ Segment addresses
☐ Function offsets

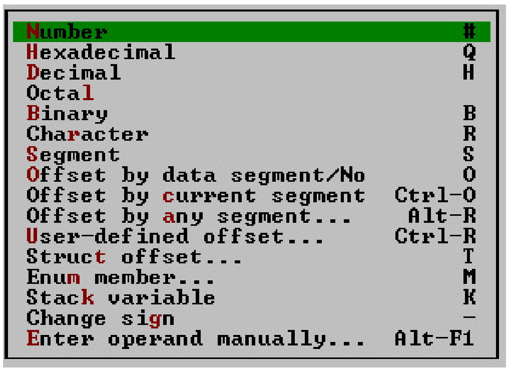
Instructions indention 16

Margin 120 (length of arguments of data directives)

See the difference ? The Text Representation menu is the key to wider arrays !

Operands

IDA Pro has a wide array of options when it comes to operand, as shown in the following menu. One interesting thing to know is that the block shortcut first encountered with the undefine command still works. Define a block and convert "en-masse".


Number #
Hexadecimal Q
Decimal H
Octal
Binary B
Character R
Segment S
Offset by data segment/No O
Offset by current segment Ctrl-O
Offset by any segment... Alt-R
User-defined offset... Ctrl-R
Struct offset... T
Enum member... M
Stack variable K
Change sign -
Enter operand manually... Alt-F1

Using Structures

Soon, you will want to use IDA Pro more advanced features - for example structures. It is possible to interactively define and manipulate structures in the disassembly. Consider this simple sample C program:

```
#include <stdio.h>
struct client {
  char code;
  long id;
  char name[32];
  client *next;
};

void print_clients(client *ptr) {
  while ( ptr != NULL ) {
    printf("ID: %4ld Name: %-32s\n",ptr->id,ptr->name);
    ptr = ptr->next;
  }
}
```

Here is the disassembly without any structures defined, as IDA Pro automatically generates it:

```
@print_clients$qp6client proc near
ptr             = word ptr  4

                push     bp
                mov      bp, sp
                push     si
                mov      si, [bp+ptr]
                jmp      short loc_1_32

loc_1_19:                           ; CODE XREF: print_clients(client *)+24j
                mov      ax, si
                add      ax, 5
                push     ax
```

```
                  push     word ptr [si+3]
                  push     word ptr [si+1]
                  mov      ax, offset aId4ldName32s
                  push     ax
                  call     _printf
                  add      sp, 8
                  mov      si, [si+25h]

loc_1_32:                           ; CODE XREF: print_clients(client *)+7j
                  or       si, si
                  jnz      loc_1_19
                  pop      si
                  pop      bp
                  retn
@print_clients$qp6client endp
```

In order to use meaningful names instead of numbers, we open the structure view (View - Structure) and press 'Ins' to define a new structure type. Structure members are then added with the 'D' key for data and the 'A' key for ASCII strings. As we add new structure members, IDA Pro automatically names them. Thereafter, you may change any member's name by pressing N.

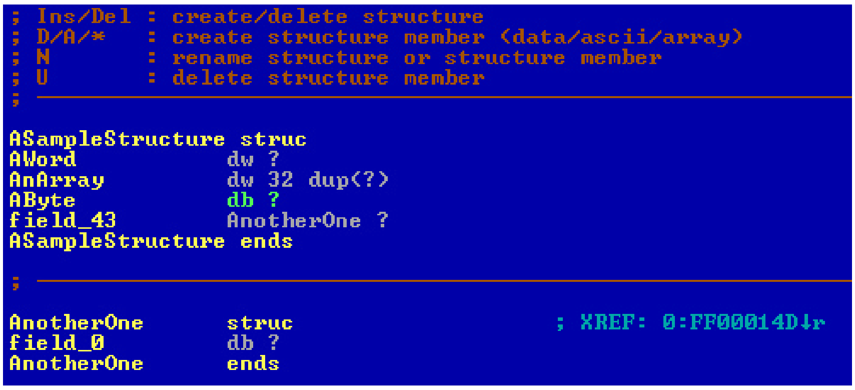
```
client_t struc
code              db ?
id                dd ?
name              db 32 dup(?)
next              dw ?
client_t ends
```

Finally, the defined structure type can be used to specify the type of an instruction operand. (menu Edit|Operand types|Struct offset).

```
@print_clients$qp6client proc near
ptr               = word ptr  4
                  push     bp
                  mov      bp, sp
                  push     si
                  mov      si, [bp+ptr]
                  jmp      short loc_1_32

loc_1_19:                           ; CODE XREF: print_clients(client *)+24j
                  mov      ax, si
                  add      ax, client_t.name
                  push     ax
                  push     word ptr [si+client_t.id+2]
                  push     word ptr [si+client_t.id]
                  mov      ax, offset aId4ldName32s
                  push     ax
                  call     _printf
                  add      sp, 8
                  mov      si, [si+client_t.next]

loc_1_32:                           ; CODE XREF: print_clients(client *)+7j
                  or       si, si
                  jnz      loc_1_19
                  pop      si
                  pop      bp
                  retn
@print_clients$qp6client endp
```

What about structures within structures ?

Yes, it is possible. First, define each structure by itself. Then, from within the higher level structure, use alt-Q to embed an instance of the member structure. Here is the result.



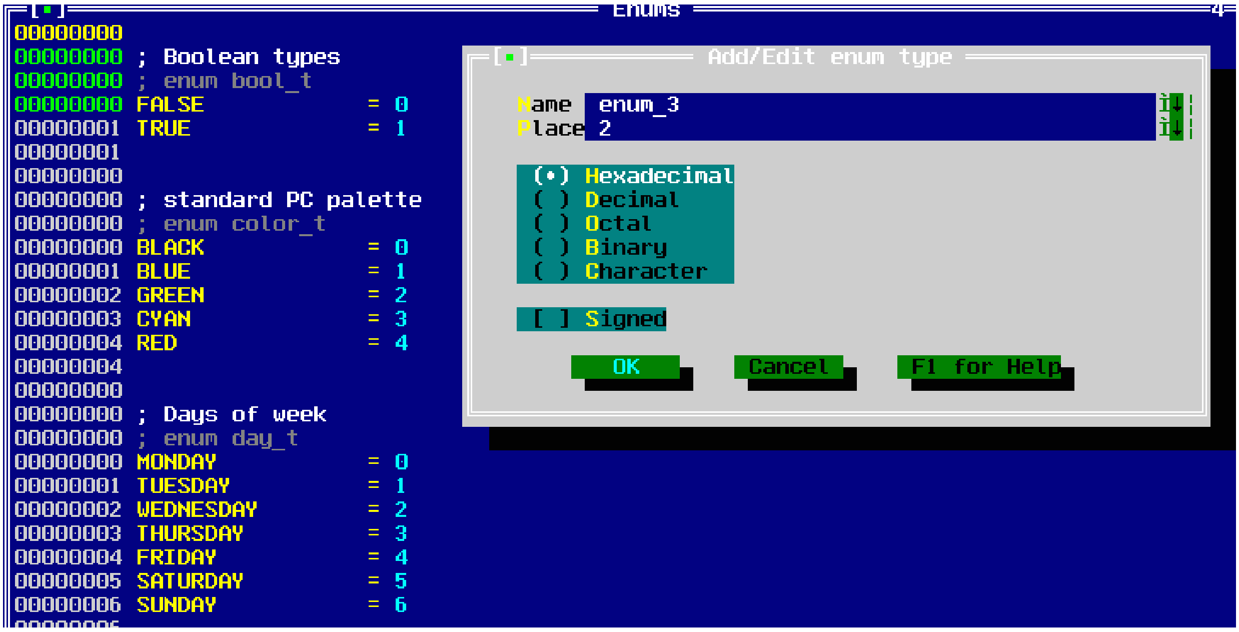
```
; Ins/Del : create/delete structure
; D/A/*   : create structure member (data/ascii/array)
; N       : rename structure or structure member
; U       : delete structure member
;

ASampleStructure struc
AWord            dw ?
AnArray          dw 32 dup(?)
AByte            db ?
field_43         AnotherOne ?
ASampleStructure ends

;

AnotherOne       struc                       ; XREF: 0:FF00014D↓r
field_0          db ?
AnotherOne       ends
```

Enumerated Types

You can use IDA Pro to interactively define and manipulate enumerated types in the disassembly. Consider this simple sample C program:

```
enum color_t {
    BLACK,          /* dark colors */
    BLUE,
    GREEN,
    CYAN,
    RED,
    MAGENTA,
    BROWN,
    LIGHTGRAY,
    DARKGRAY,           /* light colors */
    LIGHTBLUE,
    LIGHTGREEN,
    LIGHTCYAN,
    LIGHTRED,
    LIGHTMAGENTA,
    YELLOW,
    WHITE
};

enum day_t { MONDAY, TUESDAY, WEDNESDAY, THUSDAY, FRIDAY, SATURDAY, SUNDAY };

enum bool_t { FALSE, TRUE };

int is_suitable_color(day_t day,color_t color) {
  if ( (day == SUNDAY || day == SATURDAY) && color == RED ) return TRUE;
  if ( color == BLACK || color == BLUE ) return TRUE;
  return FALSE;
}
```

In order to use meaningful names instead of numbers, you simply have to open the enums window and press insert to define a new enumerated type.



```
00000000
00000000 ; Boolean types
00000000 ; enum bool_t
00000000 FALSE               = 0
00000001 TRUE                = 1
00000001
00000000
00000000 ; standard PC palette
00000000 ; enum color_t
00000000 BLACK               = 0
00000001 BLUE                = 1
00000002 GREEN               = 2
00000003 CYAN                = 3
00000004 RED                 = 4
00000004
00000000
00000000 ; Days of week
00000000 ; enum day_t
00000000 MONDAY              = 0
00000001 TUESDAY             = 1
00000002 WEDNESDAY           = 2
00000003 THURSDAY            = 3
00000004 FRIDAY              = 4
00000005 SATURDAY            = 5
00000006 SUNDAY              = 6
00000006
```

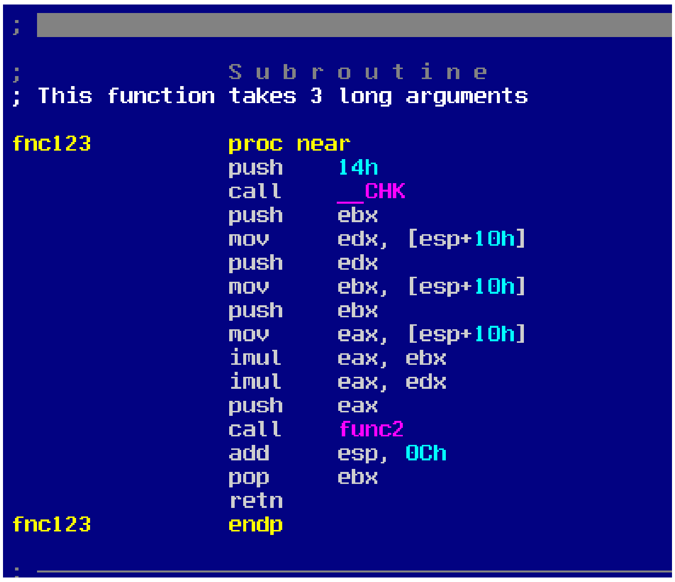
Name enum_3
Place 2

(•) Hexadecimal
( ) Decimal
( ) Octal
( ) Binary
( ) Character

[ ] Signed

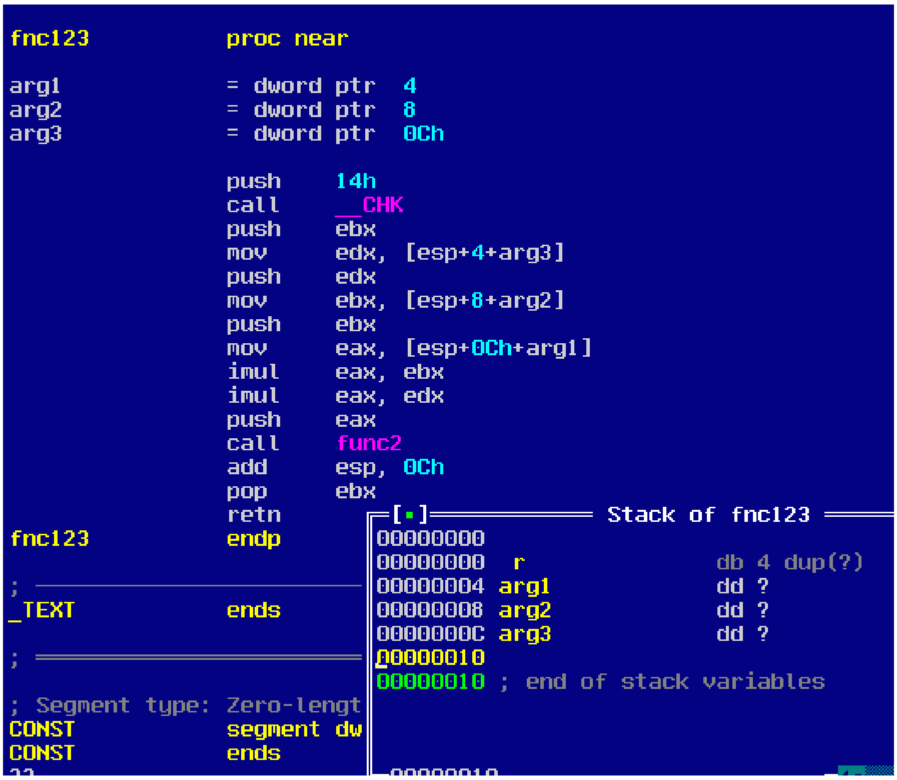
Stack Variables

Obviously the following disassembly could be improved : the parameter passing is far from evident, we simply know that a certain number of bytes are passed to the function.



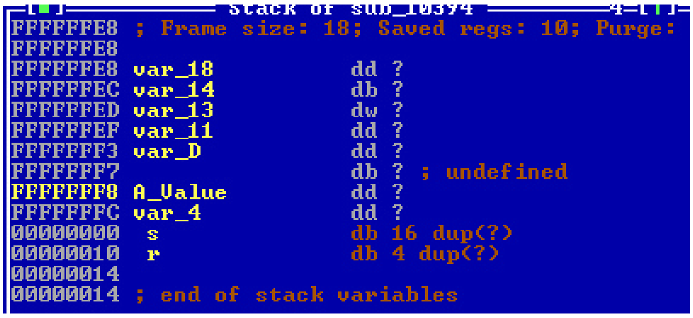
```
;
;                 Subroutine
; This function takes 3 long arguments

fnc123           proc near
                 push     14h
                 call     __CHK
                 push     ebx
                 mov      edx, [esp+10h]
                 push     edx
                 mov      ebx, [esp+10h]
                 push     ebx
                 mov      eax, [esp+10h]
                 imul     eax, ebx
                 imul     eax, edx
                 push     eax
                 call     func2
                 add      esp, 0Ch
                 pop      ebx
                 retn
fnc123           endp
```

IDA Pro will automatically recognize the parameters passed on the stack. Don't you prefer this representation ?



```
fnc123           proc near

arg1             = dword ptr  4
arg2             = dword ptr  8
arg3             = dword ptr  0Ch

                 push     14h
                 call     __CHK
                 push     ebx
                 mov      edx, [esp+4+arg3]
                 push     edx
                 mov      ebx, [esp+8+arg2]
                 push     ebx
                 mov      eax, [esp+0Ch+arg1]
                 imul     eax, ebx
                 imul     eax, edx
                 push     eax
                 call     func2
                 add      esp, 0Ch
                 pop      ebx
                 retn    ╔[■]═══════════ Stack of fnc123 ════
fnc123           endp    ║00000000
                         ║00000000  r               db 4 dup(?)
; ───────────────────────║00000004 arg1             dd ?
_TEXT            ends    ║00000008 arg2             dd ?
                         ║0000000C arg3             dd ?
; ═══════════════════════║00000010
                         ║00000010 ; end of stack variables
; Segment type: Zero-lengt║
CONST            segment dw║
CONST            ends    ║
```

Just as about everything in IDA Pro, stack variables may be given meaningful names. Here is how to do it. The stack variables of any function can be reached by pressing "CTRL-K" when the cursor is located at any position in that function. The local stack window appears and the 'N' key can be used to name stack variables. Try it an see for yourself !



```
FFFFFFE8 ; Frame size: 18; Saved regs: 10; Purge:
FFFFFFE8
FFFFFFE8 var_18            dd ?
FFFFFFEC var_14            db ?
FFFFFFED var_13            dw ?
FFFFFFEF var_11            dd ?
FFFFFFF3 var_D             dd ?
FFFFFFF7                   db ? ; undefined
FFFFFFF8 A_Value           dd ?
FFFFFFFC var_4             dd ?
00000000  s                db 16 dup(?)
00000010  r                db 4 dup(?)
00000014
00000014 ; end of stack variables
```

.

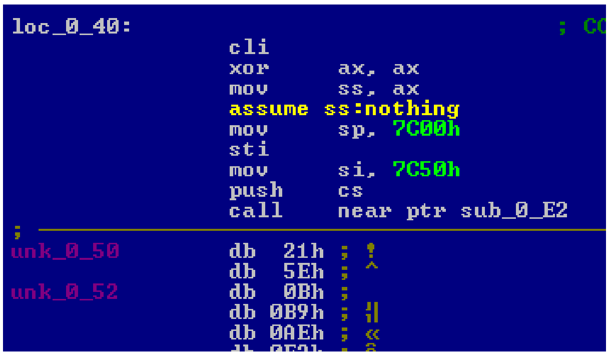
Programming with IDC

In a typical disassembly, there are a lot of repetitive tasks that could be automated or special situations that require an additional bit of control. IDA Pro offers IDC, a powerful internal C-Like language. It is documented in the IDC.IDC files and several samples examples are provided with the standard distribution. You may want to examine them carefully. Below is a real life practical example.

Using IDC to analyze encrypted code

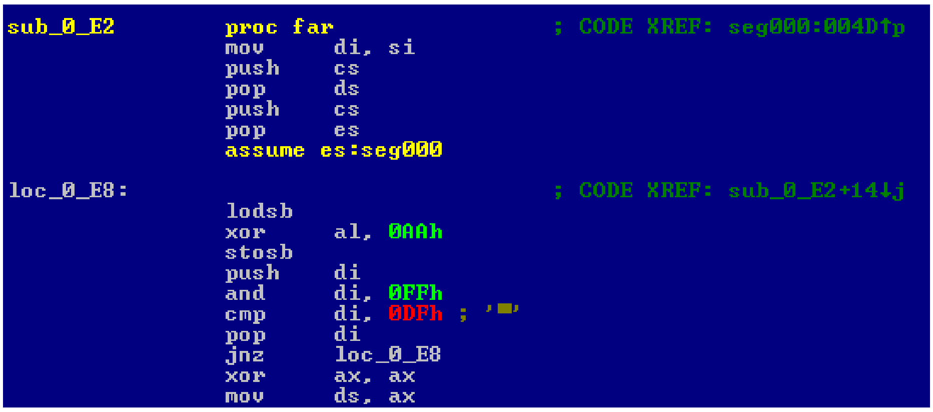
This small tutorial demonstrates how to use IDC to decrypt part of a program during analysis. The sample file is a portion of the Ripper virus.

The binary image of the virus is loaded into IDA and analysis is started at the entry point.



```
loc_0_40:                                                   ; CC
                cli
                xor     ax, ax
                mov     ss, ax
                assume ss:nothing
                mov     sp, 7C00h
                sti
                mov     si, 7C50h
                push    cs
                call    near ptr sub_0_E2
; ─────────────────────────────────────────────────────────────
unk_0_50        db  21h ; !
                db  5Eh ; ^
unk_0_52        db  0Bh ;
                db 0B9h ; ╣
                db 0AEh ; «
```

Obviously, the bytes right after the call don't make sense, but the call gives us a clue : it is a decryption routine. What we need is a small IDC routine to mimic the decryption and get at the plain text bytes.

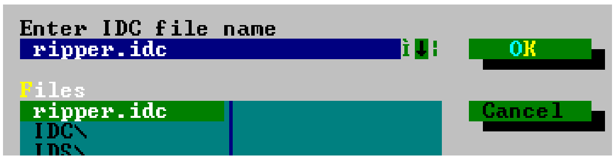


```
sub_0_E2        proc far                    ; CODE XREF: seg000:004D↑p
                mov     di, si
                push    cs
                pop     ds
                push    cs
                pop     es
                assume es:seg000

loc_0_E8:                                   ; CODE XREF: sub_0_E2+14↓j
                lodsb
                xor     al, 0AAh
                stosb
                push    di
                and     di, 0FFh
                cmp     di, 0DFh ; '▀'
                pop     di
                jnz     loc_0_E8
                xor     ax, ax
                mov     ds, ax
```

We create a small IDC program that mimics the decryption routine.

```
static decrypt(from, size, key ) {
  auto i, x;                                // we define the variables
  for ( i=0; i < size; i=i+1 ) {
    x = Byte(from);                         // fetch the byte
    x = (x^key);                            // decrypt it
    PatchByte(from,x);                      // put it back
    from = from + 1;                        // next byte
  }
}
```

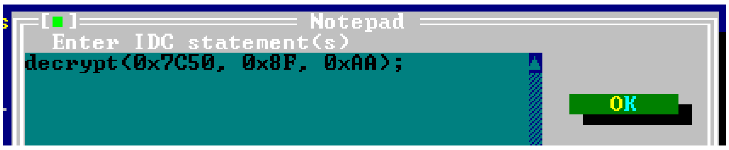
We save this IDC routine into a file and press F2 to load it into IDA's interpreter.



Enter IDC file name
ripper.idc

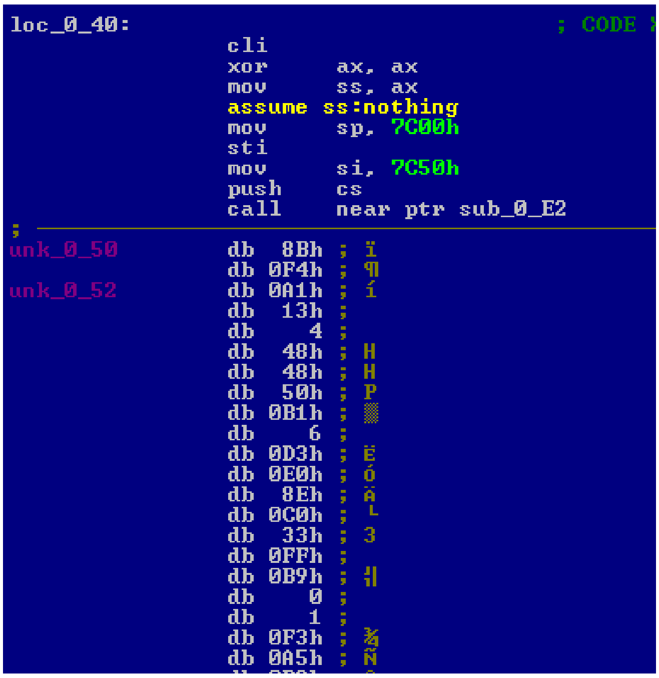
Files
ripper.idc
IDC\
IDS\

Then, we press shift-F2 to call it with the appropriate values. Please note the linear address used for the starting point. Pressing OK executes the statement.



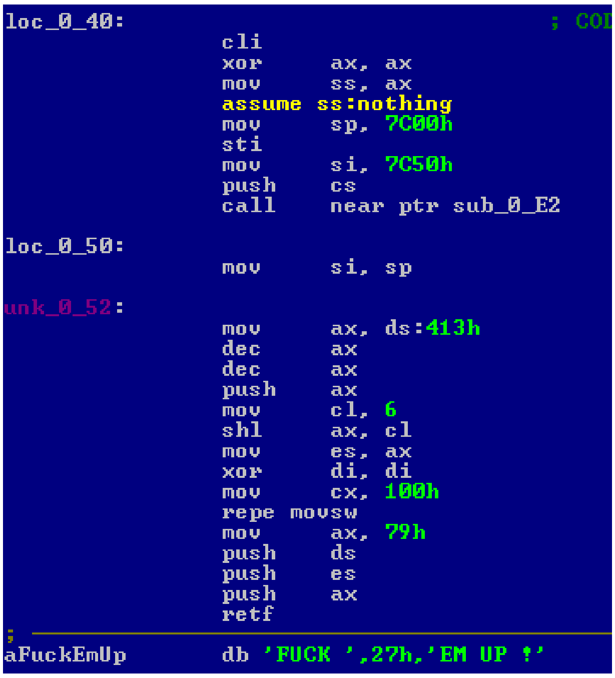
Notepad
Enter IDC statement(s)
decrypt(0x7C50, 0x8F, 0xAA);

Now that the bytes are decrypted



```
loc_0_40:                                               ; CODE
                        cli
                        xor      ax, ax
                        mov      ss, ax
                        assume ss:nothing
                        mov      sp, 7C00h
                        sti
                        mov      si, 7C50h
                        push     cs
                        call     near ptr sub_0_E2
;
unk_0_50                db  8Bh ; ï
                        db 0F4h ; ¶
unk_0_52                db 0A1h ; í
                        db  13h ;
                        db    4 ;
                        db  48h ; H
                        db  48h ; H
                        db  50h ; P
                        db 0B1h ; ▒
                        db    6 ;
                        db 0D3h ; Ë
                        db 0E0h ; Ó
                        db  8Eh ; Ä
                        db 0C0h ; └
                        db  33h ; 3
                        db 0FFh ;
                        db 0B9h ; ╣
                        db    0 ;
                        db    1 ;
                        db 0F3h ; ¾
                        db 0A5h ; Ñ
```

We move the cursor to offset 0x50 and press C to inform IDA that there is now code at that location.

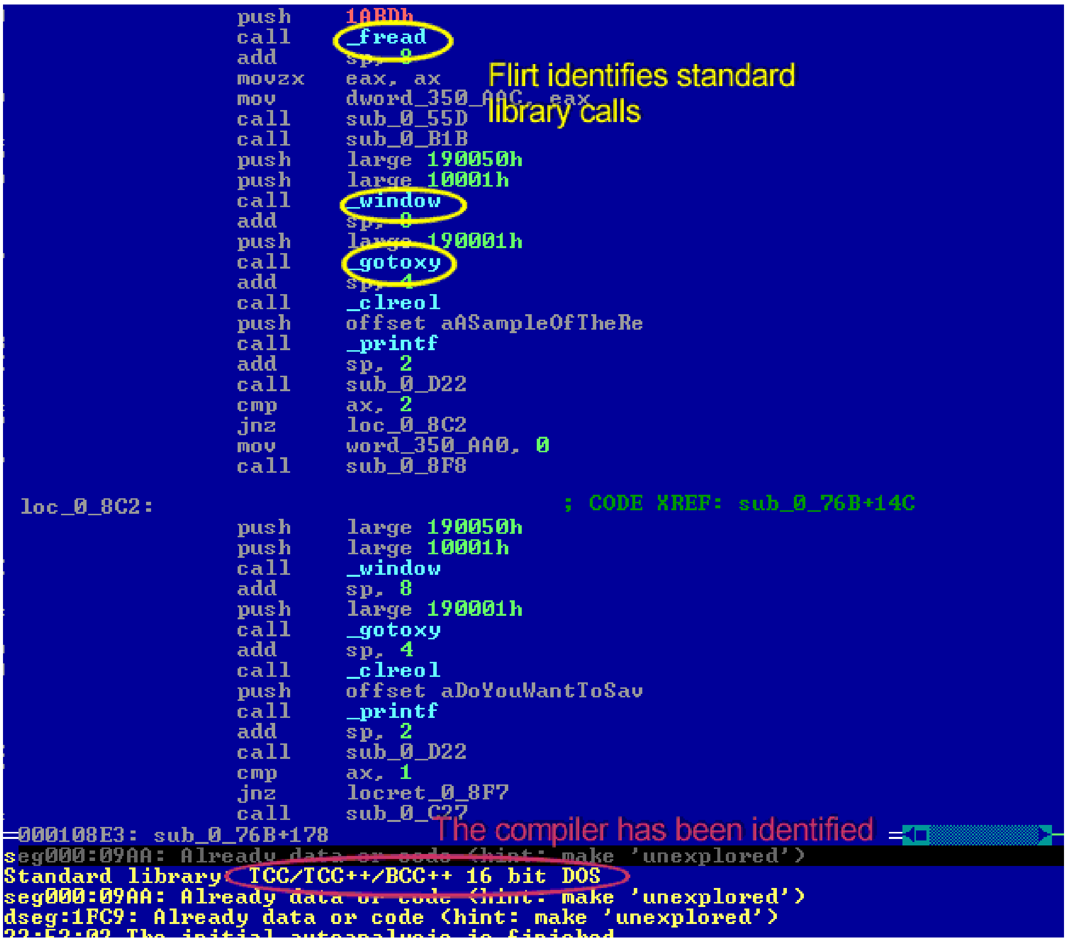


```
loc_0_40:                                               ; COD
                        cli
                        xor      ax, ax
                        mov      ss, ax
                        assume ss:nothing
                        mov      sp, 7C00h
                        sti
                        mov      si, 7C50h
                        push     cs
                        call     near ptr sub_0_E2

loc_0_50:
                        mov      si, sp

unk_0_52:
                        mov      ax, ds:413h
                        dec      ax
                        dec      ax
                        push     ax
                        mov      cl, 6
                        shl      ax, cl
                        mov      es, ax
                        xor      di, di
                        mov      cx, 100h
                        repe movsw
                        mov      ax, 79h
                        push     ds
                        push     es
                        push     ax
                        retf
;
aFuckEmUp               db 'FUCK ',27h,'EM UP !'
```

And the code to allocate memory for the virus appears, along with a rather impolite message... We can now resume analyzing the rest of the virus.

FLIRT

Fast Library Identification and Recognition Technology is another revolutionary IDA Pro capability. This technology allows IDA Pro to automatically recognize calls to the standard libraries of a long list of compilers. It makes the disassembly easier to read and saves your time. Who would want to waste time disassembling long runs of code, only to discover that is was a sequence of calls to the standard library functions ?



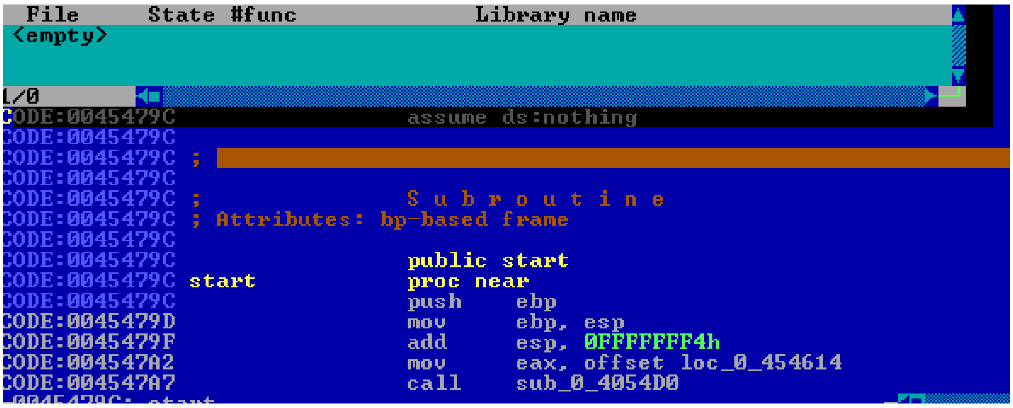
Flirt identifies standard library calls

The compiler has been identified

```
                push    1ABDh
                call    _fread
                add     sp, 8
                movzx   eax, ax
                mov     dword_350_AAC, eax
                call    sub_0_55D
                call    sub_0_B1B
                push    large 190050h
                push    large 10001h
                call    _window
                add     sp, 8
                push    large 190001h
                call    _gotoxy
                add     sp, 4
                call    _clreol
                push    offset aASampleOfTheRe
                call    _printf
                add     sp, 2
                call    sub_0_D22
                cmp     ax, 2
                jnz     loc_0_8C2
                mov     word_350_AA0, 0
                call    sub_0_8F8

 loc_0_8C2:                                     ; CODE XREF: sub_0_76B+14C
                push    large 190050h
                push    large 10001h
                call    _window
                add     sp, 8
                push    large 190001h
                call    _gotoxy
                add     sp, 4
                call    _clreol
                push    offset aDoYouWantToSav
                call    _printf
                add     sp, 2
                call    sub_0_D22
                cmp     ax, 1
                jnz     locret_0_8F7
                call    sub_0_C27
=000108E3: sub_0_76B+178
seg000:09AA: Already data or code (hint: make 'unexplored')
Standard library: TCC/TCC++/BCC++ 16 bit DOS
seg000:09AA: Already data or code (hint: make 'unexplored')
dseg:1FC9: Already data or code (hint: make 'unexplored')
22:52:02 The initial autoanalysis is finished
```

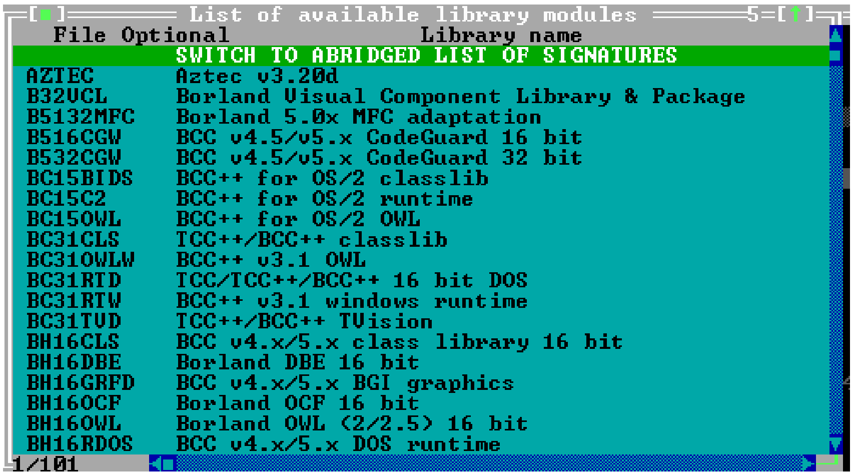
As you can see in the above screen capture, IDA Pro usually detects supported compilers automatically. However, this identification is not always 100% successful, for example because the application you are disassembling has been compiled with some specific version of a widespread compiler : this is the case for small Microsoft Windows utilities such as clock.exe. One other situation where the identification may fail is when compiler information has been stripped out of the target program, as it happens with some viruses written in high-level languages. Finally, if the compiler is not supported, recognition will fail.

If you suspect that the target program has been compiled with a supported compiler but FLIRT does not kick in automatically, you can force the application of library identifications signatures. In the example pictured on the following page - program compiled with Delphi 3 - FLIRT has not recognized the compiler, as the signature view does not list any signature set as applied.



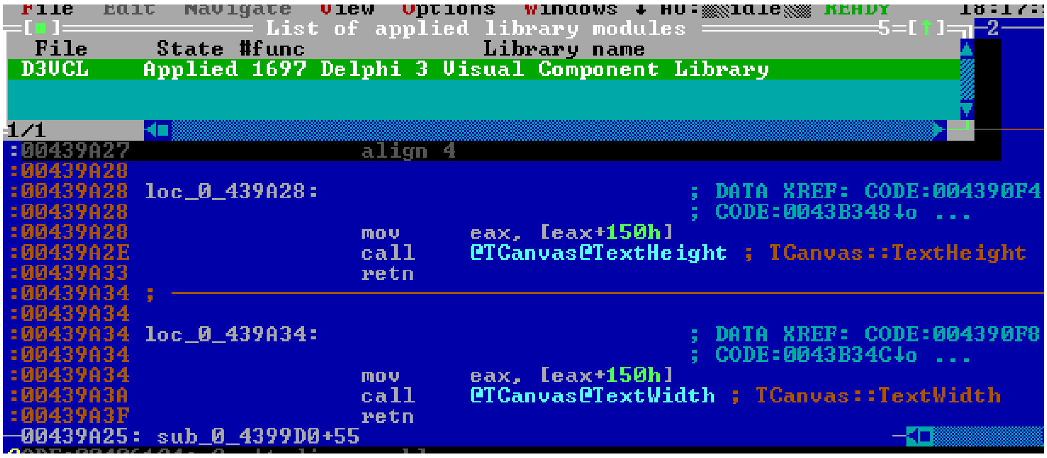
```
 File       State #func                Library name
<empty>

1/0
CODE:0045479C                         assume ds:nothing
CODE:0045479C
CODE:0045479C ;
CODE:0045479C
CODE:0045479C ;                        S u b r o u t i n e
CODE:0045479C ; Attributes: bp-based frame
CODE:0045479C
CODE:0045479C
CODE:0045479C                          public start
CODE:0045479C start                    proc near
CODE:0045479C                          push    ebp
CODE:0045479D                          mov     ebp, esp
CODE:0045479F                          add     esp, 0FFFFFFF4h
CODE:004547A2                          mov     eax, offset loc_0_454614
CODE:004547A7                          call    sub_0_4054D0
0045479C: start
```

Pressing the INS key in the signature window displays the list of available signatures.



```
[■]═════ List of available library modules ═════5=[↑]
 File Optional                Library name
           SWITCH TO ABRIDGED LIST OF SIGNATURES
 AZTEC      Aztec v3.20d
 B32VCL     Borland Visual Component Library & Package
 B5132MFC   Borland 5.0x MFC adaptation
 B516CGW    BCC v4.5/v5.x CodeGuard 16 bit
 B532CGW    BCC v4.5/v5.x CodeGuard 32 bit
 BC15BIDS   BCC++ for OS/2 classlib
 BC15C2     BCC++ for OS/2 runtime
 BC15OWL    BCC++ for OS/2 OWL
 BC31CLS    TCC++/BCC++ classlib
 BC31OWLW   BCC++ v3.1 OWL
 BC31RTD    TCC/TCC++/BCC++ 16 bit DOS
 BC31RTW    BCC++ v3.1 windows runtime
 BC31TVD    TCC++/BCC++ TVision
 BH16CLS    BCC v4.x/5.x class library 16 bit
 BH16DBE    Borland DBE 16 bit
 BH16GRFD   BCC v4.x/5.x BGI graphics
 BH16OCF    Borland OCF 16 bit
 BH16OWL    Borland OWL (2/2.5) 16 bit
 BH16RDOS   BCC v4.x/5.x DOS runtime
1/101
```

Applying the Delphi 3 Visual Component Library gives returns this result



```
 File   Edit   Navigate   View   Options   Windows  ↓ AU: idle   READY      18:17:
[■]═══════════ List of applied library modules ════════5=[↑]═ 2
  File        State #func               Library name
 D3VCL      Applied 1697 Delphi 3 Visual Component Library

1/1
:00439A27                      align 4
:00439A28
:00439A28 loc_0_439A28:                                    ; DATA XREF: CODE:004390F4
:00439A28                                                  ; CODE:0043B348↓o ...
:00439A28                      mov     eax, [eax+150h]
:00439A2E                      call    @TCanvas@TextHeight ; TCanvas::TextHeight
:00439A33                      retn
:00439A34 ;
:00439A34
:00439A34 loc_0_439A34:                                    ; DATA XREF: CODE:004390F8
:00439A34                                                  ; CODE:0043B34C↓o ...
:00439A34                      mov     eax, [eax+150h]
:00439A3A                      call    @TCanvas@TextWidth ; TCanvas::TextWidth
:00439A3F                      retn
00439A25: sub_0_4399D0+55
```

1697 functions have been identified, resulting in a much more understandable disassembly. What if your compiler is unsupported, you still may benefit from the FLIRT technology, at least if you have access to your compiler libraries. By downloading our tools and generating your own FLIRT databases, you will be able to attain the same high level of recognition that you get with the shipping defaults.

Processor SDK

A processor SDK exists. It is available for free to all of our existing customers. At this stage, it is officially unsupported, although we do provide some support when we can. How difficult is it to create your own processor module ? Well, frankly, it is not an easy task....

To be continued and expanded...