

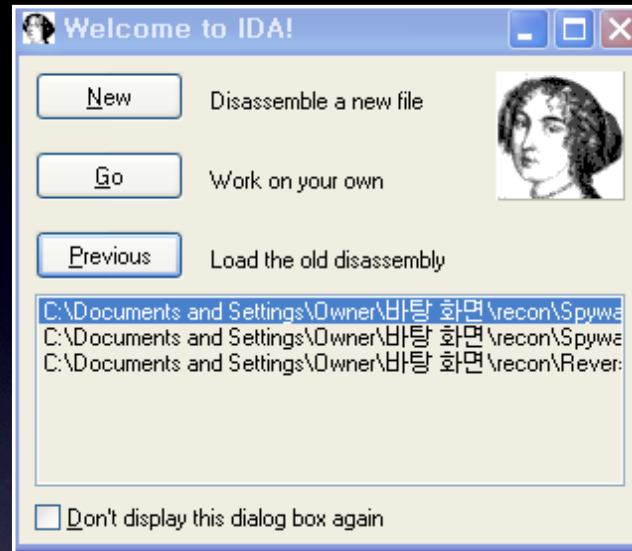
# The IDA Pro Book

Hacking Group OVERTIME  
force (forceteam01@gmail.com)

# GETTING STARTED WITH IDA

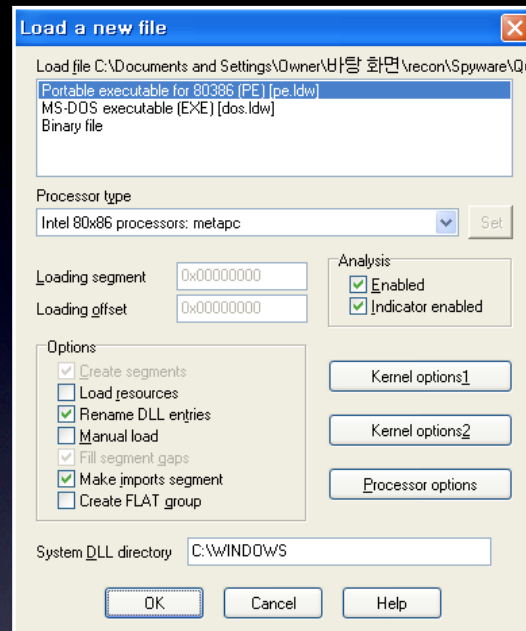


# IDA 구동



- New : 분석할 새로운 파일을 선택하기 위한 마법사를 실행한다
- Go : IDA가 마법사 실행 없이 빈 작업공간을 만든다
- Previous : 버튼 바로 밑 윈도우의 가장 최근 작업 내역을 실행한다  
리스트 윈도우 항목 중 더블클릭으로 실행 가능

# IDA File 로딩

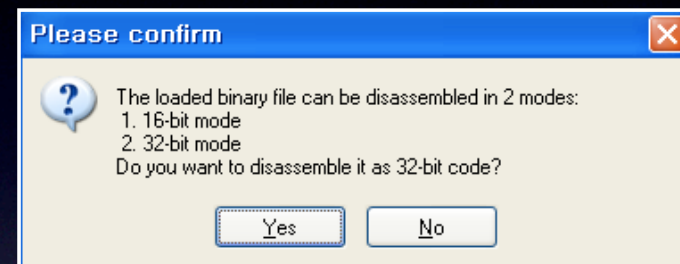
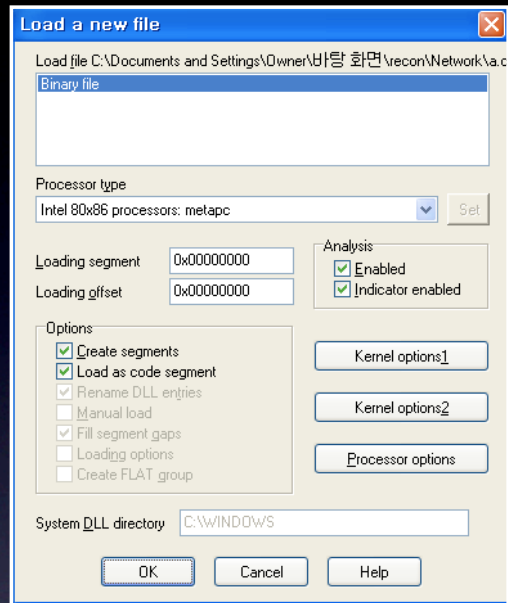


File -> Open 명령을 이용하여 새로운 파일을 열 경우 로딩 창을 볼수 있다

- Processor type : 사용할 특정 프로세서 모듈을 선택할 수 있다
- Loading Segment and Loading Offset  
    바이너리 파일이 x86 계열의 프로세서에서 실행되는 경우에만 활성화 된다  
    바이너리 로더가 메모리 배열 정보를 수집하지 못할 경우 사용자가 임의의 값을 입력할 수 있다



# IDA File 로딩



Rom 이미지 또는 네트워크 패킷 캡처에서 얻은 Payload를 분석할 경우

- 바이너리 파일은 자신의 메모리 배열에 대한 정보를 전혀 가지고 있지 않으므로 Loading segment & Loading offset 값을 사용자가 임의로 지정해야 하며 바이너리 파일이 32bit일 경우 Yes를 선택한다
- 이와같은 경우 사용자가 해당 파일의 entry point 부분을 찾아서 해당 바이트를 코드로 변환해야 한다
- 이와같은 바이너리 파일의 경우 사용자가 최소 하나의 byte를 코드로 변환하지 않을 경우 IDA는 어떤 초기화 과정도 수행하지 않는다

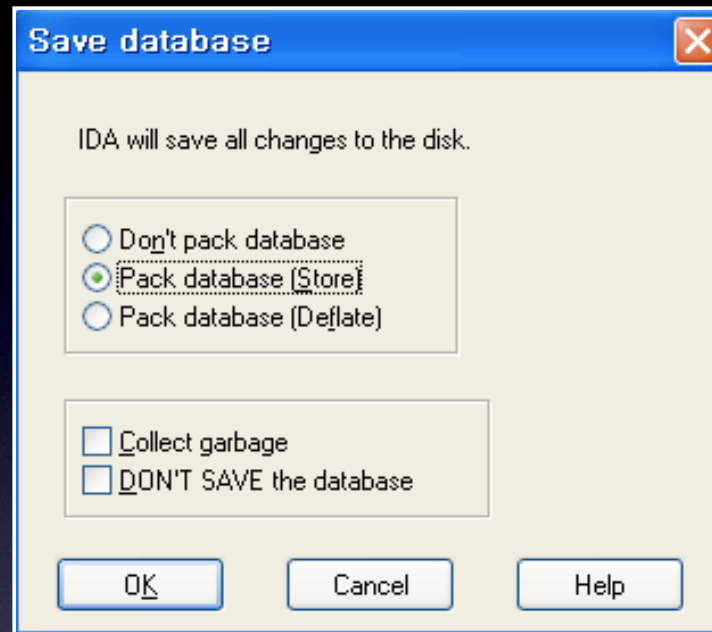
# IDA Database File

IDA Database File은 4개의 개별 파일로 구성되어 있다

- idb file : B-tree-style database 콘텐츠를 포함하고 있다
- idl file : 각각의 Program 바이트를 설명한 플래그를 포함하고 있다
- nam file : IDA Name window에 표시되는 named program 위치와 관련된 인덱스 정보를 포함하고 있다
- til file : 로컬 타입 정의와 관련된 정보를 저장하고 있다

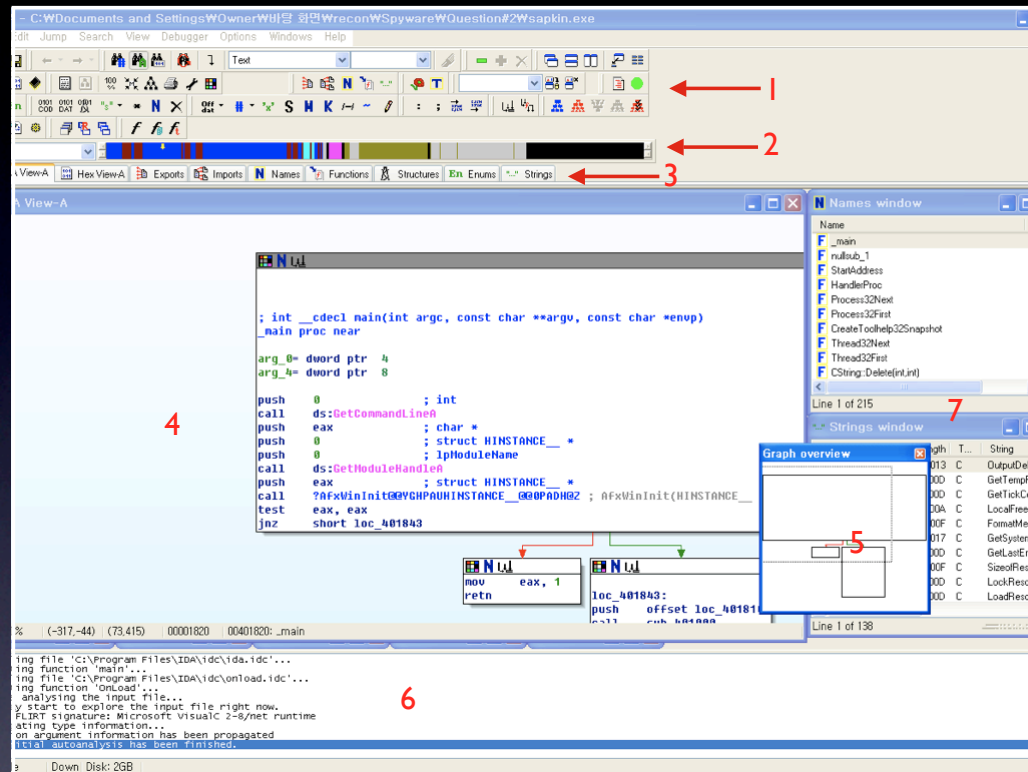


# IDA Database 닫기



- Don't pack database : IDB 파일을 생성하지 않고 4개의 파일에 분산 저장한다
- Pack database(Store) : 압축된 하나의 IDB 파일에 저장한다
- Pack database(Deflate) : 압축되지 않은 하나의 IDB 파일에 저장한다
- Collect garbage : garbage Collection을 수행한다
- DON'T SAVE the database : 기존 database 파일에 현재 변화를 적용하지 않는다

# IDA Desktop 소개

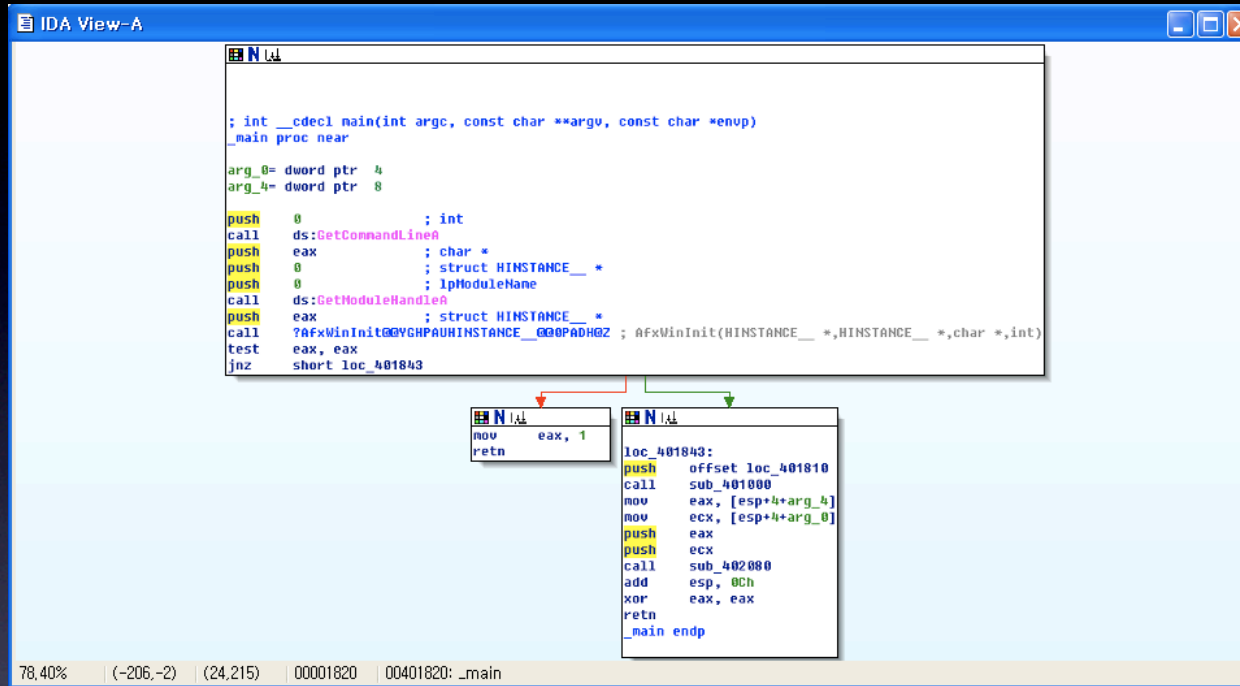


1. toolbar area
2. overview navigator 또는 navigation band : 로드된 파일의 주소 공간을 순차적으로 보여준다
3. tabs
4. disassembly view
5. graph overview : 현재 보여지는 구간의 함수를 그래프로 표기
6. message window : IDA의 메시지를 표기
7. sub window : Names, Strings windows와 같은 sub windows



# The IDA DATA DISPLAY

# The Disassembly Window(IDA Graph View)



- 함수의 제어 흐름을 각 함수별로 block을 구성해서 보여준다
- 사용자는 그래프 뷰화면에서 CTRL+Wheel 구성으로 화면의 크기를 조정할 수 있다
- 사용자는 그래프 뷰화면에서 CTRL+, CTRL- 화면크기를 조정할 수 있다  
여기서 +,- 는 숫자키패드의 키를 나타낸다
- Graph Overview 화면의 점선박스를 이동시키면 박스안의 화면으로 View가 이동한다
- 각 라인별 Virtual Address를 표현하고 싶으면 아래 설명대로 Line Prefixes를 선택한다  
Options -> General -> Disassembly Tab-> Display disassembly line parts "Line Prefixes" check



# The Disassembly Window(IDA Text View)

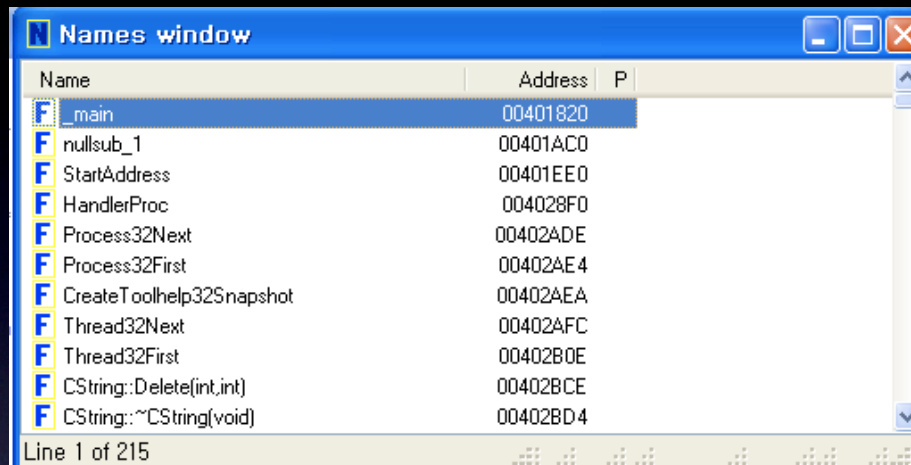
The screenshot shows the IDA Text View window with the following assembly code and annotations:

```
.text:004029E2      mov     ecx, eax
.text:004029E4      test    ecx, ecx
.text:004029E6      jz      short loc_402A0D
.text:004029E8      lea     esi, [esp+1Ch+var_10]
.text:004029EC      loc_4029EC:
.text:004029EC      push    0                ; CODE XREF: sub_4029A0+6B↓j
.text:004029EC      push    eax              ; lpWindowName
.text:004029EE      push    eax              ; lpClassName
.text:004029EF      call    sub_401B20
.text:004029F4      add     esp, 8
.text:004029F7      test    eax, eax
.text:004029F9      jz      short loc_402A04
.text:004029FB      push    eax              ; Parameter
.text:004029FC      call    sub_401F20
.text:00402A01      add     esp, 4
.text:00402A04      loc_402A04:
.text:00402A04      mov     eax, [esi]        ; CODE XREF: sub_4029A0+59↑j 2
.text:00402A04      add     esi, 4
.text:00402A06      test    eax, eax
.text:00402A09      jnz     short loc_4029EC
.text:00402A0D      loc_402A0D:
.text:00402A0D      push    3E8h            ; CODE XREF: sub_4029A0+46↑j
.text:00402A0D      call    edi              ; dwMilliseconds
.text:00402A12      cmp     dword_404150, 1
.text:00402A14      jz      short loc_4029DD
.text:00402A18      pop     edi
.text:00402A1D
```

Annotations include cross-references (e.g., `CODE XREF: sub_4029A0+6B↓j`), comments (e.g., `lpWindowName`, `lpClassName`, `Parameter`, `Sleep`, `dwMilliseconds`), and control flow arrows (dashed lines for jumps, solid lines for calls). A red vertical bar is visible on the left side of the window.

- Virtual address는 [SECTION NAME]:[VIRTUAL ADDRESS] 형태로 표기된다
- 1번 좌측 끝의 화살표 부분을 “arrows window” 라고 부른다  
Conditional jump의 경우 파선, Unconditional jump의 경우 일반선으로 표기되며  
이전 주소로 점프하는 경우는 굵은 선으로 표기한다
- 2번 cross-references 이 경우 code cross-reference로 cross-references는 향 후 알아본다

# The Names Window



- F : regular function (IDA 가 라이브러리 function으로 인식하지 못한 것들을 나타낸다)
- L : Library function (IDA는 signature matching algorithms를 사용하여 Library function을 확인한다)
- I : Imported name (고유 라이브러리로부터 임포트된 가장 일반적인 function name)
- C : Named Code (IDA가 어떤 함수의 일부분인지 인식하지 못하는 named program instruction locations)  
예) Program의 Symbol table에 이름은 있지만 실제 프로그램에서 호출되지않는 함수)
- D : Data (Named data locations 일반적으로 글로벌 변수들을 의미한다)
- A : ASCII string data (null byte로 종료되는 4개 이상의 ASCII 문자를 포함하는 data location을 참조한다)

## 역자주

named program instruction or Named data의 경우는 한글로 번역하기 모호해서 원문을 사용한다  
한글로 번역한다면 특정 이름으로 명명된 “프로그램 명령 위치” 또는 “데이터 위치” 로 풀이한다



# The Names Window

IDA는 위치정보에 대한 이름을 선택할때 가상주소 앞에 이름지어지는 위치의 타입이 어떤가에 따라서 아래와 같은 형태를 나타낸다

sub_xxxxxx	: 주소 xxxxxx 서브루틴
loc_xxxxxx	: 주소 xxxxxx 명령
byte_xxxxxx	: 주소 xxxxxx 8-bit data
word_xxxxxx	: 주소 xxxxxx 16-bit data
dword_xxxxxx	: 주소 xxxxxx 32-bit data
unk_xxxxxx	: 주소 xxxxxx 사이즈를 알지 못하는 data

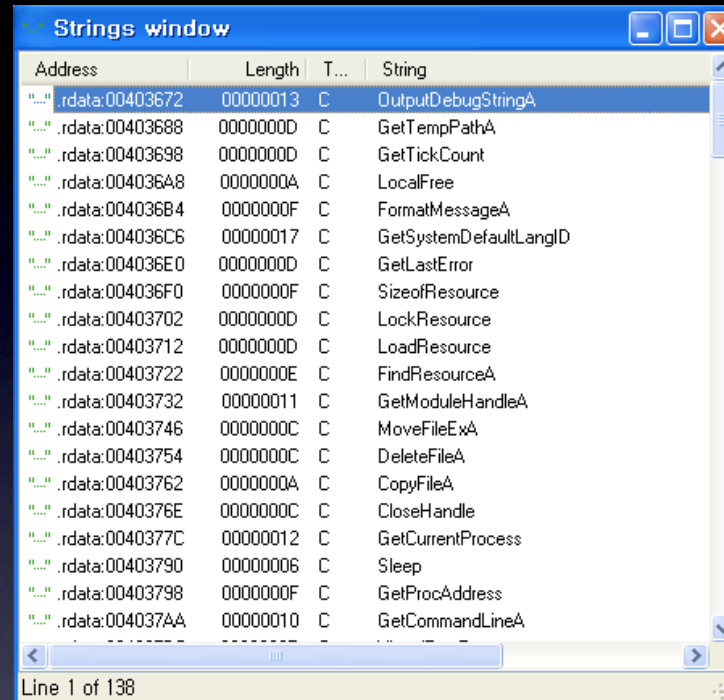
# The Message Window

새로운 파일을 열었을 때 하단부에 나타나는 메시지 윈도우는 IDA output Console로 IDA가 수행하는 작업에 대한 정보를 나타낸다

마우스 우측 클릭으로 해당 되는 메뉴를 선택하면 복사 또는 삭제 기능을 수행한다



# The Strings Window

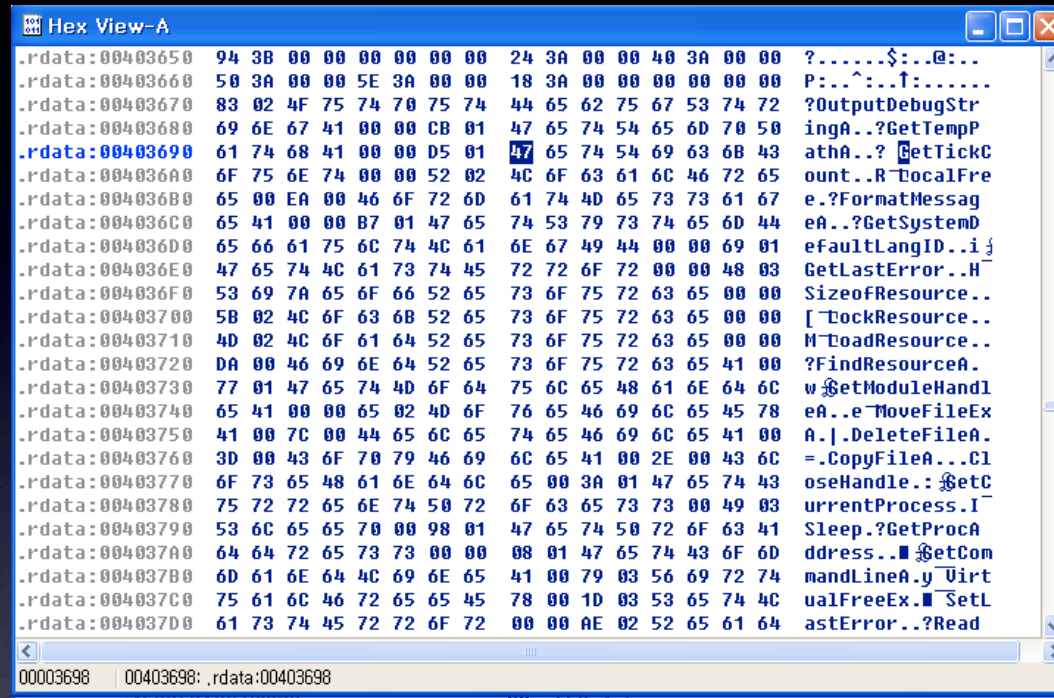


Address	Length	T...	String
... .rdata:00403672	00000013	C	OutputDebugStringA
... .rdata:00403688	0000000D	C	GetTempPathA
... .rdata:00403698	0000000D	C	GetTickCount
... .rdata:004036A8	0000000A	C	LocalFree
... .rdata:004036B4	0000000F	C	FormatMessageA
... .rdata:004036C6	00000017	C	GetSystemDefaultLangID
... .rdata:004036E0	0000000D	C	GetLastError
... .rdata:004036F0	0000000F	C	SizeofResource
... .rdata:00403702	0000000D	C	LockResource
... .rdata:00403712	0000000D	C	LoadResource
... .rdata:00403722	0000000E	C	FindResourceA
... .rdata:00403732	00000011	C	GetModuleHandleA
... .rdata:00403746	0000000C	C	MoveFileExA
... .rdata:00403754	0000000C	C	DeleteFileA
... .rdata:00403762	0000000A	C	CopyFileA
... .rdata:0040376E	0000000C	C	CloseHandle
... .rdata:0040377C	00000012	C	GetCurrentProcess
... .rdata:00403790	00000006	C	Sleep
... .rdata:00403798	0000000F	C	GetProcAddress
... .rdata:004037AA	00000010	C	GetCommandLineA

Line 1 of 138

- Binary에 포함된 string에 대하여 주소, 길이, 타입, String을 표현한다
- Strings Window의 각 리스트를 더블클릭하면 디스어셈블리 윈도우에 해당 주소가 나타난다
- Strings Window에서 마우스 우측클릭으로 setup을 선택하면 List 와 Type을 설정할 수 있다
- 5.1 버전 이하에서는 기본 window 였으니 5.2 버전 이상에서는 기본 window로 열리지 않는다

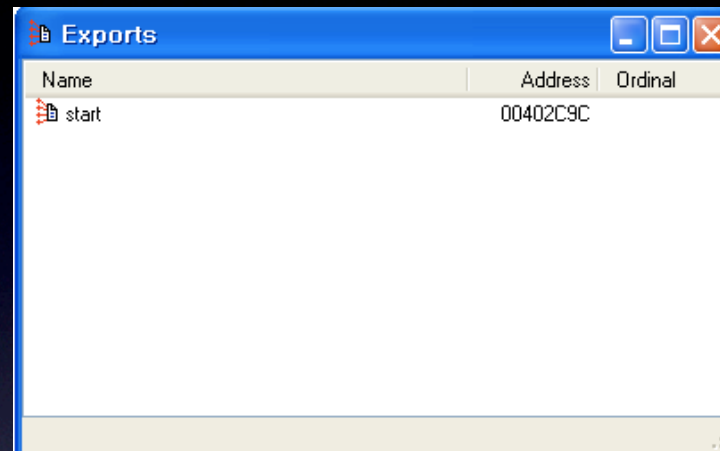
# The Hex View Window



- Disassembly, Names, message, Strings windows 에 더해서 IDA는 몇개의 windows를 최소화 상태로 오픈한다
- 네비게이션 밴드 밑의 각 탭들은 해당 window를 나타낸다
- 첫번째 Hex Window는 첫번째 Disassembly Window와 동기화 된다
- Hex Window 또는 Disassembly Window 위치를 이동하면 다른 하나의 윈도우 역시 같은 위치로 이동한다
- 특정 Hex Window 또는 Disassembly Window 간 동기화가 필요한 경우 해당 window에서 우측 마우스클릭 메뉴 중 “Synchronize with” 메뉴에서 해당 window를 지정하면 된다
- Hex Window에 “?” 표시가 나타나면 IDA가 해당 주소 범위에 어떤 값이 채워지는지 알지 못할 경우이다

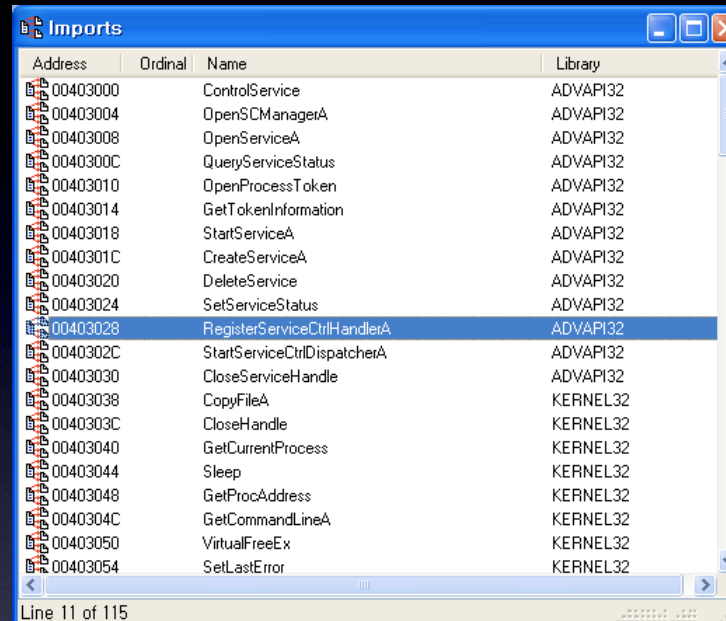


# The Exports Window



- Exports window는 entry point와 export된 function 들을 나타낸다
- 프로그램의 Entry point는 IDA에서는 start로 표시한다
- Window에서 표현은 Name, Address, 가능하다면 Ordinal 순서로 표현한다
- 해당 항목을 더블 클릭하면 해당 주소로 Disassembly window가 이동한다

# The Imports Window



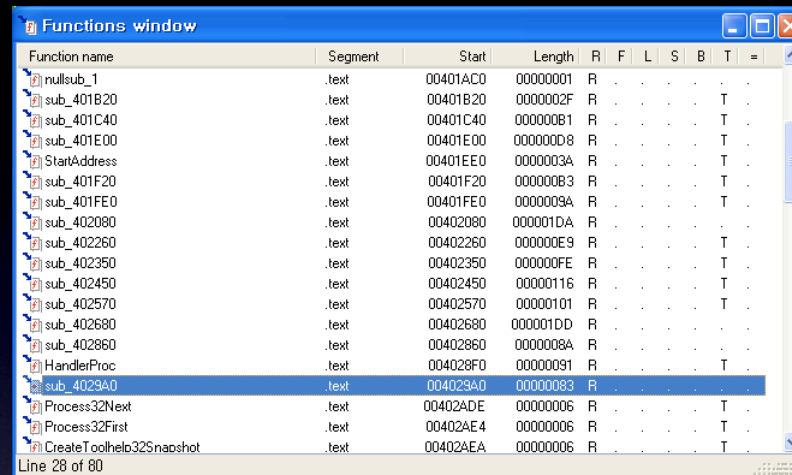
- Imports window는 분석과정에서 import된 function 들을 나타낸다
- 해당 항목을 더블 클릭하면 해당 주소로 Disassembly window가 이동한다

\* 주의 \*

Imports window에는 dynamic loader에 의해서 자동적으로 처리된 symbol들만 나타난다. dlopen/dlsym 또는 LoadLibrary/GetProcAddress와 같이 자신만의 방법을 이용하는 경우에는 나타나지 않는다



# The Functions Window



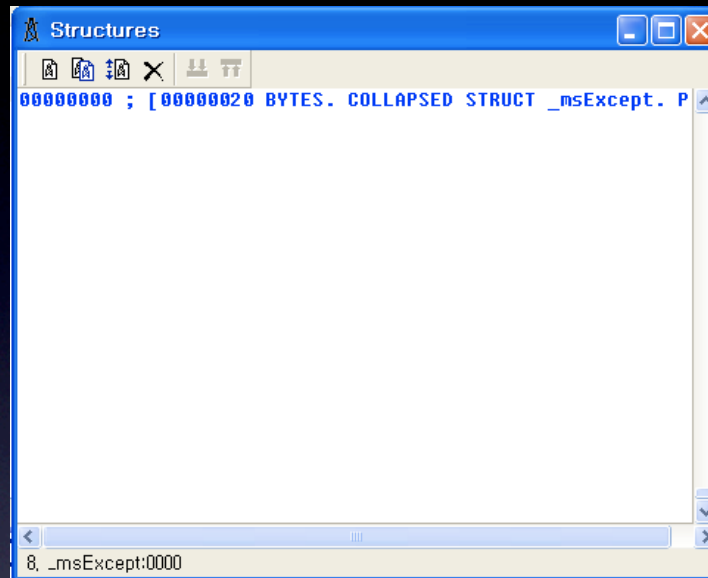
Function name	Segment	Start	Length	R	F	L	S	B	T	=
sub_401B20	.text	00401B20	0000002F	R	.	.	.	.	T	.
sub_401C40	.text	00401C40	000000B1	R	.	.	.	.	T	.
sub_401E00	.text	00401E00	000000D8	R	.	.	.	.	T	.
sub_401F20	.text	00401F20	000000B3	R	.	.	.	.	T	.
sub_401FE0	.text	00401FE0	0000009A	R	.	.	.	.	T	.
sub_402080	.text	00402080	000001DA	R	.	.	.	.	T	.
sub_402260	.text	00402260	000000E9	R	.	.	.	.	T	.
sub_402350	.text	00402350	000000FE	R	.	.	.	.	T	.
sub_402450	.text	00402450	00000116	R	.	.	.	.	T	.
sub_402570	.text	00402570	00000101	R	.	.	.	.	T	.
sub_402680	.text	00402680	000001DD	R	.	.	.	.	T	.
sub_402860	.text	00402860	0000008A	R	.	.	.	.	T	.
sub_4029A0	.text	004029A0	00000083	R	.	.	.	.	T	.
Process32Next	.text	00402ADE	00000006	R	.	.	.	.	T	.
Process32First	.text	00402AE4	00000006	R	.	.	.	.	T	.
CreateToolhelp32Snapshot	.text	00402AEA	00000006	R	.	.	.	.	T	.

Line 28 of 80

- Functions Window는 데이터베이스안의 모든 Functions을 나타낸다
- Names Window에는 sub\_xxxxxx 형태의 function을 나타내지 않지만 Functions Window는 모든 Function을 나타낸다
- 기타 function Flags
  - R - function returns to the caller
  - F - far function
  - L - library function
  - S - static function
  - B - BP based frame. IDA will automatically convert all frame pointer [BP+xxx] operands to stack variables.
  - T - function has type information
  - = - Frame pointer is equal to the initial stack pointer  
In this case the frame pointer points to the bottom of the frame



# The Structures Window



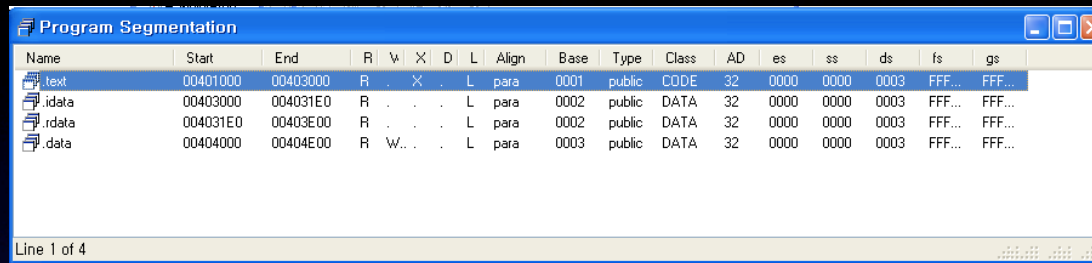
- Structure Window는 C 구조체 또는 Union 같은 복잡한 데이터 구조체를 나타낸다
- 해당 구조체를 더블 클릭하면 상세 내역에 대해서 확인할 수 있다

# The Enum Window

- Structures Window와 비슷하나 C의 enum과 같은 데이터 타입을 표시한다



# The Segments Window



The screenshot shows the 'Program Segmentation' window in IDA Pro. It contains a table with the following data:

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	es	ss	ds	fs	gs
text	00401000	00403000	R	.	X	.	L	para	0001	public	CODE	32	0000	0000	0003	FFF...	FFF...
.idata	00403000	004031E0	R	.	.	.	L	para	0002	public	DATA	32	0000	0000	0003	FFF...	FFF...
.rdata	004031E0	00403E00	R	.	.	.	L	para	0002	public	DATA	32	0000	0000	0003	FFF...	FFF...
.data	00404000	00404E00	R	W	.	.	L	para	0003	public	DATA	32	0000	0000	0003	FFF...	FFF...

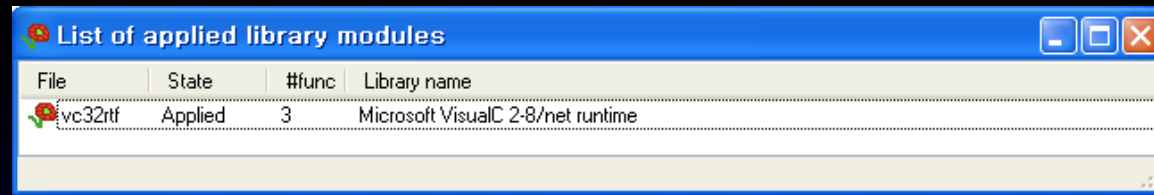
Line 1 of 4

- IDA에서는 Binary file 구조에서 표현하는 Section들을 segment로 표현한다
- 분석할 파일의 Section 을 표시한다

Name : Segment name  
 Start : 가상 시작 주소  
 End : 가상 종료 주소  
 R : 'R' readable, '.' not readable, '?' unknown  
 W : 'W' writable, '.' not writable, '?' unknown  
 X : 'X' executable, '.' not executable, '?' unknown  
 D : 'D' debugger only, '.' regular  
 L : 'L' created by loader, '.' no  
 Align : Segment alignment  
 Base : Segment base selector or address  
 Type : Segment type  
 Class : Segment class  
 AD : Segment addressing width



# The Signatures Window

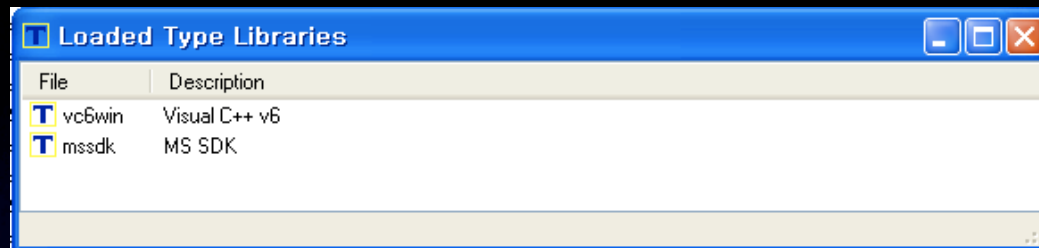


- IDA는 알려진 코드 블록을 확인하기 위해서 광범위한 시그니처 라이브러리를 사용한다
- IDA는 해당 바이너리가 어떤 컴파일러에 의해서 생성되었는지 구분하여 해당 signature를 적용한다. 따라서 해당 바이너리에서 사용된 알려진 함수들을 구분해서 보여준다

File : vc32rft signature가 적용되었음을 나타낸다

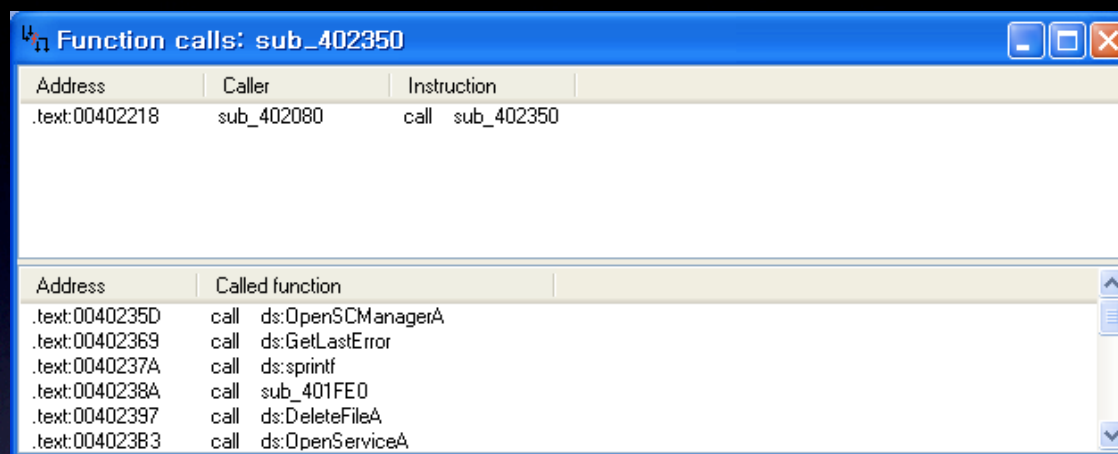
#func : 라이브러리 함수로 3개의 함수를 인식했음을 나타낸다

# The Type Libraries Window



- Type libraries는 가장 많이 사용되는 컴파일러에 포함된 헤더파일로 부터 수집한 데이터 타입과 함수 프로토타입에 대한 IDA의 축적된 기술을 보여준다

# The Function Call Window



Address	Caller	Instruction
.text:00402218	sub_402080	call sub_402350

Address	Called function
.text:0040235D	call ds:OpenSCManagerA
.text:00402369	call ds:GetLastError
.text:0040237A	call ds:sprintf
.text:0040238A	call sub_401FE0
.text:00402397	call ds>DeleteFileA
.text:004023B3	call ds:OpenServiceA

- Function Call Window를 열면 IDA는 커서가 위치한 함수의 이웃 함수들을 결정한다



# DISASSEMBLY NAVIGATION

# Basic IDA Navigation

## Double-Click Navigation

- 예제1 `.text:00401331     jg         short     loc_40134E`  
`loc_40134E`와 같이 특정 이름으로 명명된 심볼을 더블클릭하면 IDA는 선택한 지점을 표시한다.
- 예제2 `.text:0040134E   loc_40134E                     ;CODE XREF: sub_4012E4+4D^j`  
IDA는 탐색 목표로 두가지 추가적인 display entities를 취급한다.  
먼저 cross-reference 에 대해서 알아본다. cross-reference는 이름+16진수오프셋 형태이며  
예제2의 `loc_40134E` 우측의 cross-reference는 `sub_4012E4` 16진수 4D 바이트 뒤에서 참조함을  
나타낸다 cross-reference 문자열을 더블클릭하면 해당 위치로 이동한다  
예제2의 경우에는 00401331로 이동한다
- 예제3 `.data:00409013             dd   404590h`  
두번째 형태는 16진수 값을 사용하는 형태이다. 만일 바이너리에서 Virtual address를 16진수로  
표현한다면 해당 16진수 더블클릭으로 선택한 주소로 디스어셈블리 윈도우가 이동한다  
단 16진수 값을 사용하는 형태의 경우 가상주소값일 경우에만 해당 주소로 이동한다
- IDA message window에서 해당 내용을 더블클릭하면 해당 목적지로 이동한다

## Jump to Address

- Jump to Address dialog는 Jump -> Jump to Address 또는 G 단축키를 이용해서 해당 주소로 이동한다



# Navigation History

IDA는 웹 브라우저와 같이 “앞으로가기” “뒤로가기” 기능을 제공한다

- 뒤로가기 기능은 Jump -> Jump to Previous Position repositions 를 선택함과 동시에 디스어셈블리 윈도우는 바로 이전 위치로 이동한다. 웹브라우저의 뒤로가기와 동일하다  
관련된 단축키는 ESC 키를 누르면 된다 그러나 디스어셈블리 윈도우 외의 다른 윈도우에서 ESC 키를 사용하면 해당 윈도우가 닫힌다.
- 앞으로가기 기능은 Jump -> Jump to Next Position을 선택하면 해당되는 위치로 이동한다.  
웹브라우저의 앞으로가기 기능과 동일하며 관련된 단축키는 CTRL+ENTER 이다
- 앞의 두 기능은 IDA 메뉴 아이콘의 Forward and backward navigation button을 이용할수도 있다

# Stack Frames

## Calling Conventions

Calling Convention은 다른 많은 문서에도 다루어지므로 여기서는 일단 생략하고 향 후 추가예정

- C Calling Convention
- The Standard Calling Convention
- The fastcall Convention for x86
- C++ Calling Conventions
- Other Calling Conventions



# Local Variable Layout

## Stack Frame Examples

여기서는 함수에 파라미터를 전달하는 방법 및 IDA에서 지역 변수를 어떻게 표현하는지 또한 stack frame에서 어떤 방식으로 표현하는지를 나타낸다

이번 내용은 향 후 자세히 다루기로 한다. 이와같은 내용은 정리되기가 힘들며 서술형태로 내용을 다루어야 하므로 향 후 책 내용을 서술 형태로 정리해서 추가 할 것이다.

Calling Convention에 따른 스택의 구성 및 해제를 이해하고 있다면 넘어가도 상관없다

# Searching the Database

- Text Searches

텍스트 검색은 Search -> Text 메뉴 또는 ALT+T 단축키로 가능하다  
또한 검색을 계속하기를 원하면 CTRL+T 또는 Search -> Next Text 메뉴를 이용한다

- Binary Searches

바이너리 검색은 Search -> Sequence of Bytes 메뉴 또는 ALT+B 단축키로 가능하다  
또한 검색을 계속하기를 원하면 CTRL+B 또는 Search -> Next Sequence of Bytes 메뉴를 이용한다



# DISASSEMBLY MANIPULATION

# Names and Naming

기존에 명명된 이름을 변경하기 위해서는 해당 이름을 더블클릭하거나 단축키 N을 누르면 되고 또한 마우스 우측 클릭 메뉴중 **Rename** 옵션을 클릭해서 해당 이름을 변경한다

- **Parameters and Local Variables**

변경한 이름에 대해서 IDA가 생성한 기본 이름으로 변경할 경우 **Renaming dialog** 에서 공백을 입력한다

- **Named Locations**

##### Renaming Location Dialog 그림 추가 #####

**Local name** : 범위가 현재 함수에 제한된다 따라서 두개의 함수가 같은 **Local name**을 소유할 수 있다

**Include in names list** : 이 옵션을 선택하면 **Names Window**에 추가된다

**Public name** : 공유 라이브러리와 같은 바이너리에서 추출된 이름을 나타낸다

- **Register Names**

위의 경우와 마찬가지로 **Rename** 또는 단축키 N을 이용해서 변경할 수 있다.



# Commenting in IDA

IDA는 여러 스타일의 주석을 제공하며 각각의 스타일은 그 목적에 따라서 사용된다  
디스어셈블리 특정 라인에 주석을 추가하고 싶다면 **Edit -> Comments** 메뉴 또는 단축키를 이용한다  
IDA의 경우 세미콜론(;) 이후는 주석으로 취급한다.

- **Regular Comments(주석)**  
디스어셈블리 우측 끝에서 마우스 우측 클릭 또는 단축키 콜론(:)을 사용하여 **Comment entry dialog**를 실행한다.
- **Repeatable Comments(반복 주석)**  
repeatable comments의 행동(?)은 cross-reference의 컨셉을 tied  
기본적으로 참조된 주소는 회색 텍스트로 나타내서 다른 주석들과 구분한다
- **Anterior and Posterior Lines**  
Anterior and Posterior Lines은 주어진 disassembly line 전 또는 후에 나타나는 라인전체로 표시되는 주석이다
- **Function Comments(함수 주석)**  
함수 주석은 특정 함수의 disassembly list의 맨 위에 표시되는 주석 그룹을 말한다

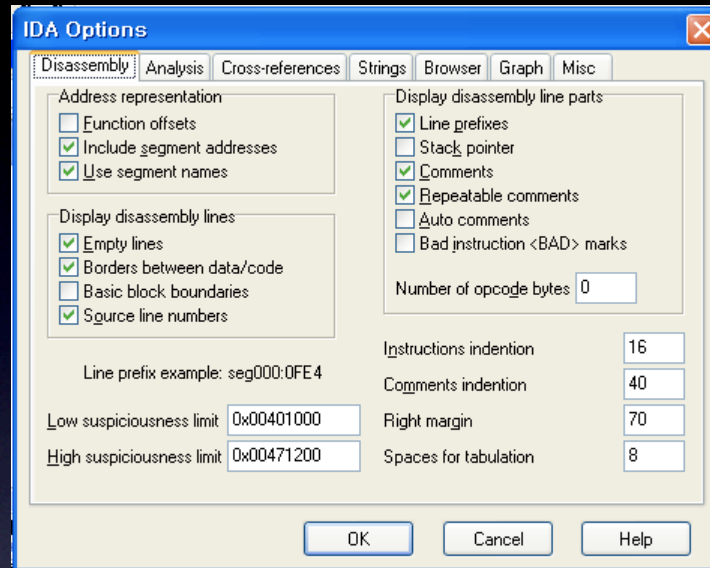
# Basic Code Transformations

많은 경우 IDA가 생성한 완벽한 disassembly list를 볼수 있지만 그렇지 못한 경우도 있다. 이와같은 경우 우리는 좀 더 효율적으로 disassembly analysis 또는 display process 해야만한다. IDA가 제공하는 Code transformation 기능들은 다음과 같다

- data를 code로 변환
- code를 data로 변환
- 특정 명령들을 함수로 변환
- 함수의 출발 또는 종료 주소 변화
- 명령 operands의 표현 포맷 변화



# Basic Code Transformations



- Code Display Options

Line prefixes : 각 disassembly line 앞에 오프셋을 표시한다

Stack pointer : 각 함수의 진행에 따른 관련된 스택 포인터의 변화를 표시한다.  
잘못된 Calling convention으로 인한 스택의 불일치를 확인할 수 있다

# Basic Code Transformations

- **Formatting Instruction Operands**  
IDA Disassembly line의 각 Operand 선택 후 우측 마우스 클릭으로 사용자 선택적 포맷으로 변경 가능함
- **Creating New Functions**  
새롭게 생성할 함수의 첫번째 바이트 또는 명령에 커서를 위치시키고 Edit->Functions->Create Function 을 선택해서 해당 명령그룹을 함수로 생성할 수 있다
- **Deleting Functions**  
Edit->Functions->Delete Function 메뉴 선택으로 삭제
- **Function Chunks**  
어떤 함수에도 속하지 않는 주소 범위를 선택해서 새로운 Function Chunk로 생성할 수 있다  
Edit->Functions->Append Function Tail 메뉴를 선택하면 현재 존재하는 함수 리스트에서 붙여 넣을 함수를 선택하면 된다



# Basic Code Transformations

- **Formatting Instruction Operands**  
IDA Disassembly line의 각 Operand 선택 후 우측 마우스 클릭으로 사용자 선택적 포맷으로 변경 가능함
- **Creating New Functions**  
새롭게 생성할 함수의 첫번째 바이트 또는 명령에 커서를 위치시키고 Edit->Functions->Create Function 을 선택해서 해당 명령그룹을 함수로 생성할 수 있다
- **Deleting Functions**  
Edit->Functions->Delete Function 메뉴 선택으로 삭제
- **Function Chunks**  
어떤 함수에도 속하지 않는 주소 범위를 선택해서 새로운 Function Chunk로 생성할 수 있다  
Edit->Functions->Append Function Tail 메뉴를 선택하면 현재 존재하는 함수 리스트에서 붙여 넣을 함수를 선택하면 된다

# Basic Code Transformations

**Edit function**

Name of function: WinMain@16

Start address: .text:0043E54F

End address: .text:0043F1E4

Color: DEFAULT

Enter size of (in bytes)

Local variables area: 0x118

Saved registers: 0x4

Purged bytes: 0x10

Frame pointer delta: 0x68

☐ Does not return

☐ Far function

☐ Library func

☐ Static func

☒ BP based frame

☐ BP equals to SP

OK Cancel Help

- Function Attributes

Name of functions : 함수명

Start(End) address : 함수 시작(종료) 주소

Local variables area : 함수를 위한 지역 변수 전용 스택의 바이트 수를 나타낸다

Saved registers : 호출자를 위한 save registers의 바이트 수를 나타낸다

Purged bytes : 함수가 호출자에게 리턴할때 스택에서 제거되는 파라미터 바이트 수를 나타낸다

Frame pointer delta : 조절된 프레임 포인터에서 saved 프레임 포인터까지의 거리를 나타낸다



# Basic Code Transformations

- Stack Pointer Adjustments
- Converting Data to Code(and Vice Versa)

바이트를 자동 분석할 경우 때때로 Data를 코드로 또는 반대의 경우로 잘못 분석되는 경우들이 나타난다. 이유와 상관없이 disassembly 포맷을 변경하려면 변경할 Data 또는 Code 에서 마우스 우측 버튼의 “undefine”을 선택한다

정의되지 않은 바이트 순서를 disassemble 하기 위해서는 첫번째 바이트에서 마우스 우측 클릭 후 Code를 선택한다. 광범위한 구역을 변환하기 위해서는 마우스로 해당 지역을 선택한 후 같은 과정으로 변환한다

# Basic Data Transformations

- Specifying Data Sizes

Data를 수정하는 가장 쉬운 방법은 그 사이즈를 조절하는것이다. 가장 자주 마주치는 지정자는 db, dw, dd 이며 각각 1,2,4 byte의 데이터이다

Data item의 사이즈를 변경하는 첫번째 방법은 Options-> Setup Data Types 을 선택하여 나타나는 dialog에서 해당 item을 수정하는 것이다

- Working with Strings

Data를 강제로 String으로 변환하기 위해서는 Edit-> Strings 메뉴의 옵션을 활용하여 특정 string 스타일을 선택한다

- Specifying Arrays

IDA는 연속적인 data 정의 그룹을 하나의 array 정의로 그룹화하는 기능을 제공한다

array를 생성하기 위해서는 array의 첫번째 요소를 선택하고 Edit-> Array 메뉴를 선택해서 array-creation dialog를 실행시킨다