리버싱 이 정도는 알아야지

보충 자료

http://securityfactory.tistory.com

목 차

보충 자료

SECURITY FACTORY

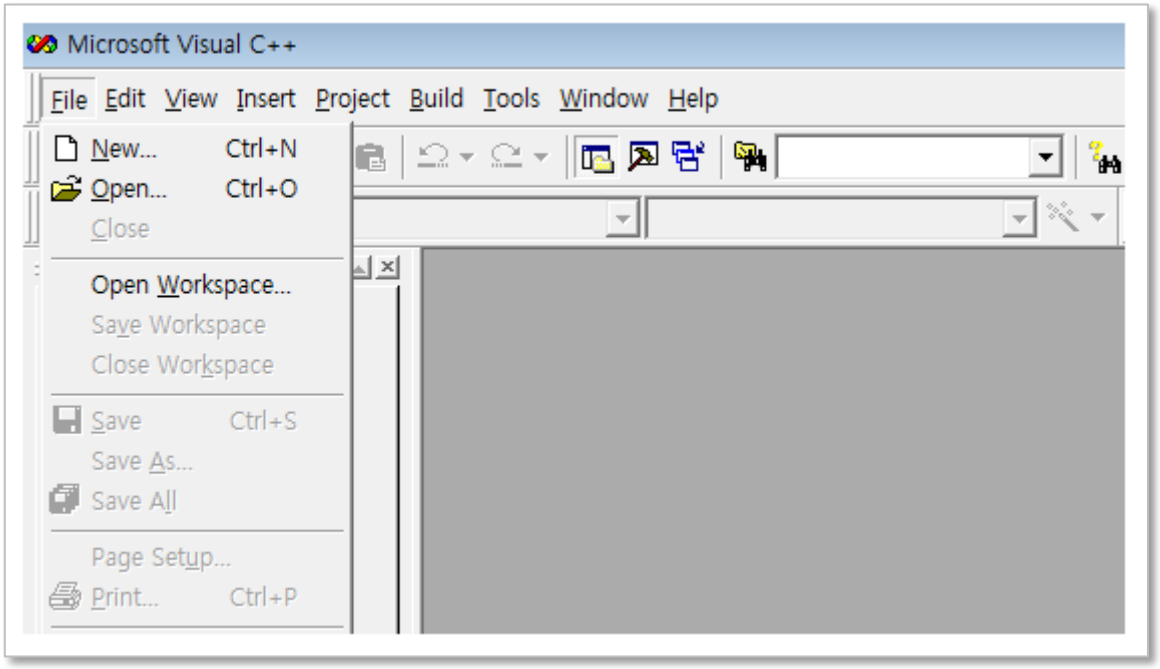
1. 개발도구 사용하기

코드를 작성하기 위해서는 기본적으로 프로젝트를 생성하고, 거기에 소스코드를 추가해줘야 합니다.

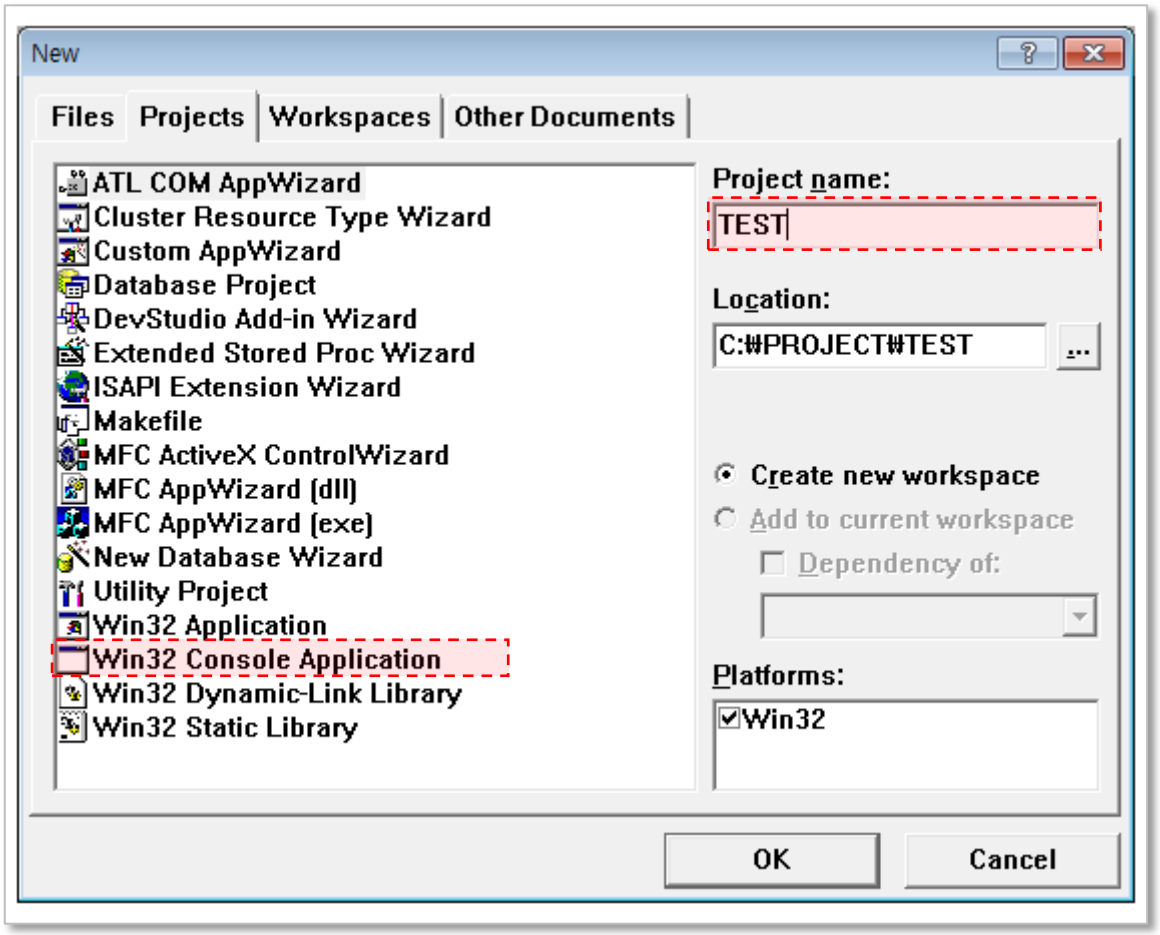
- 프로젝트 생성: 코드 개발에 필요한 자료(코드 등)를 하나로 모아서 관리할 수 있게 방을 만드는 작업
- 소스파일 추가: 코드를 작성하기 위해 도화지를 추가하는 작업

[프로젝트 생성]

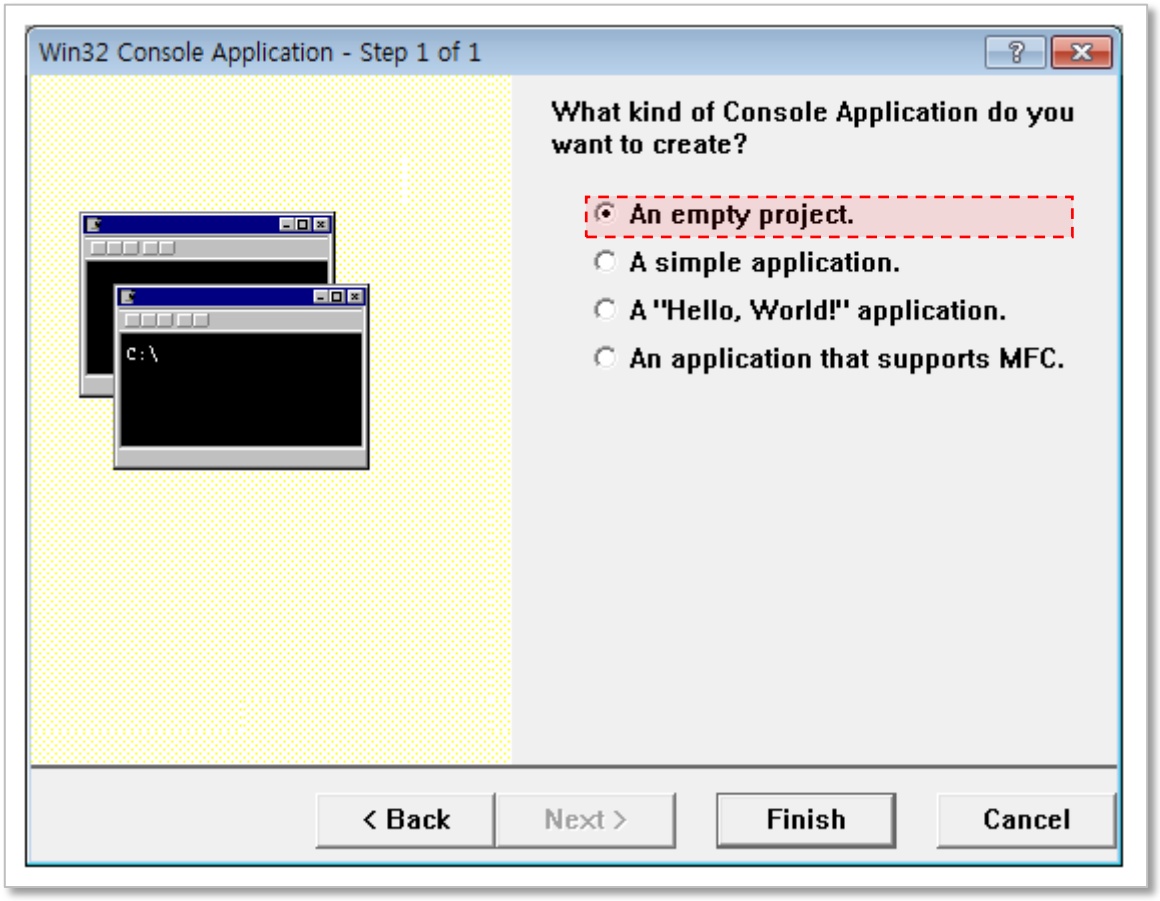
① 메뉴 → File → New Project



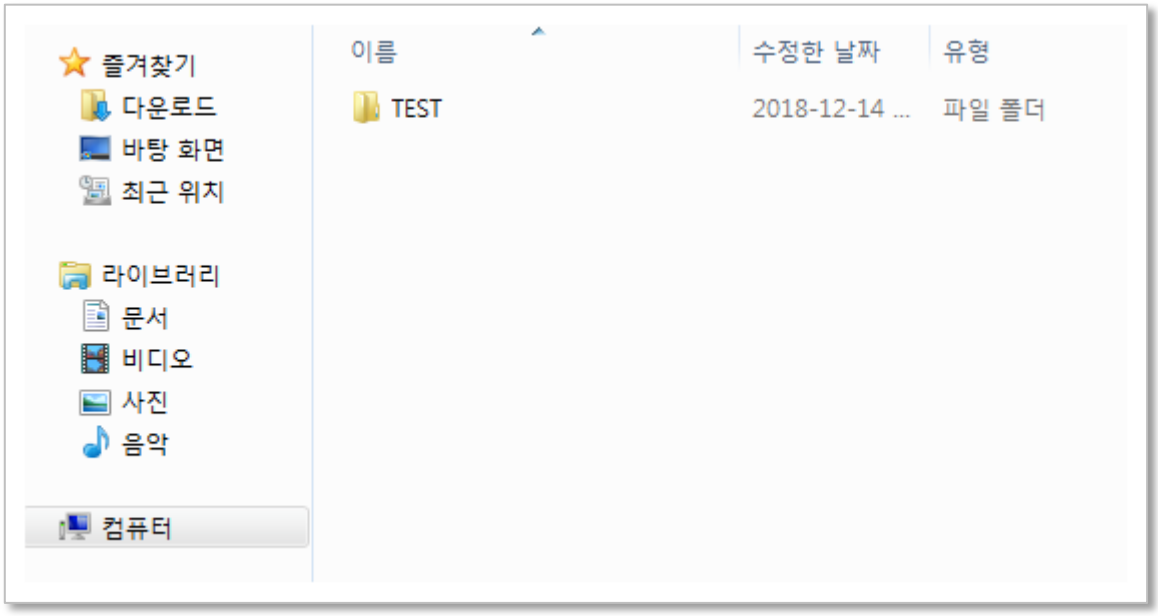
② “Win32 콘솔 어플리케이션” 선택 → 프로젝트 이름 부여



③ “An empty project” 선택

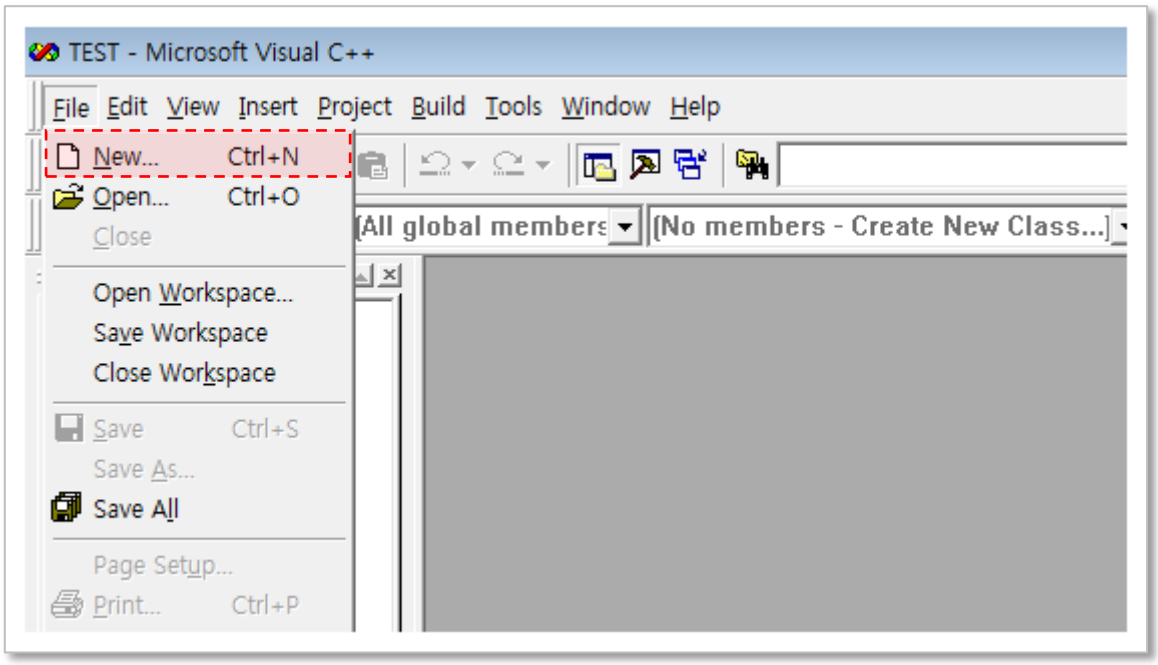


④ 프로젝트 폴더 생성 확인

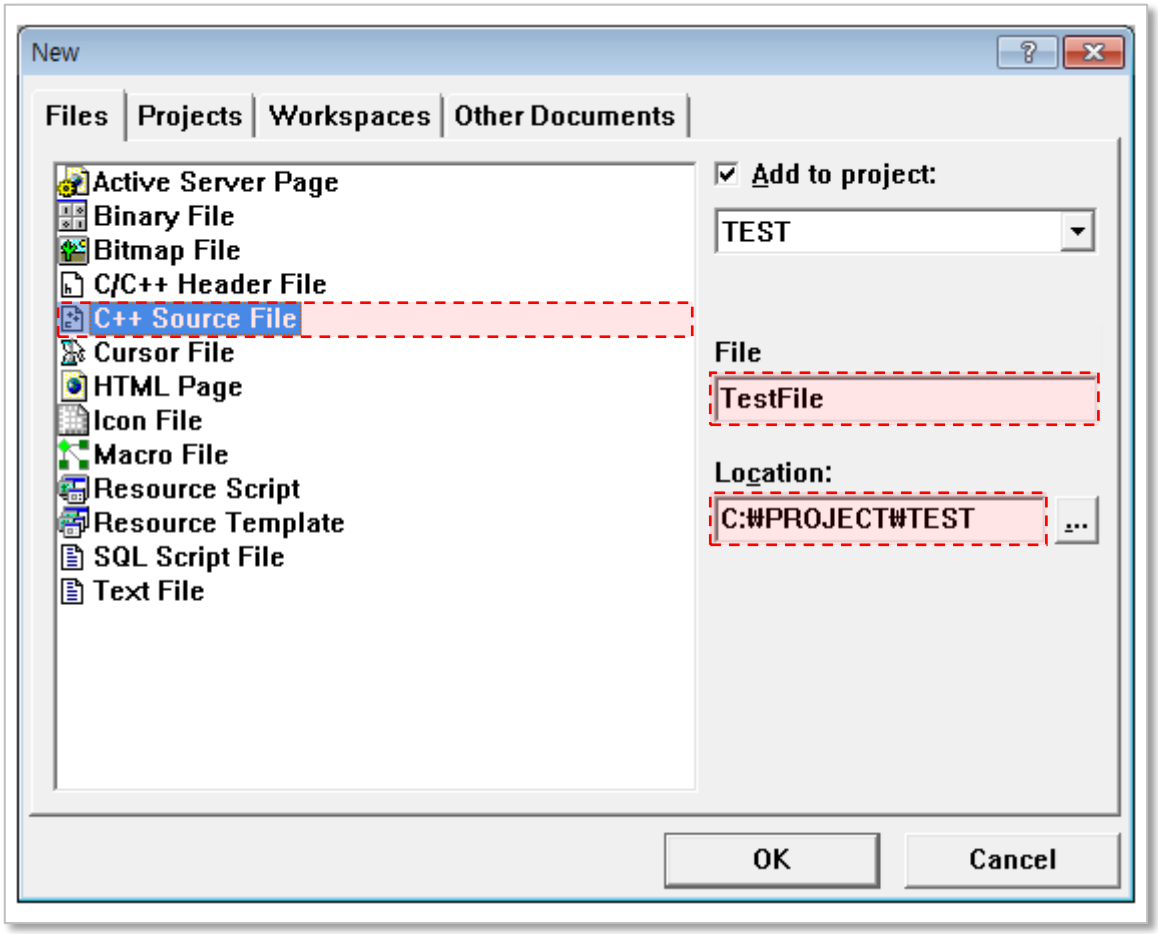


[소스파일 추가]

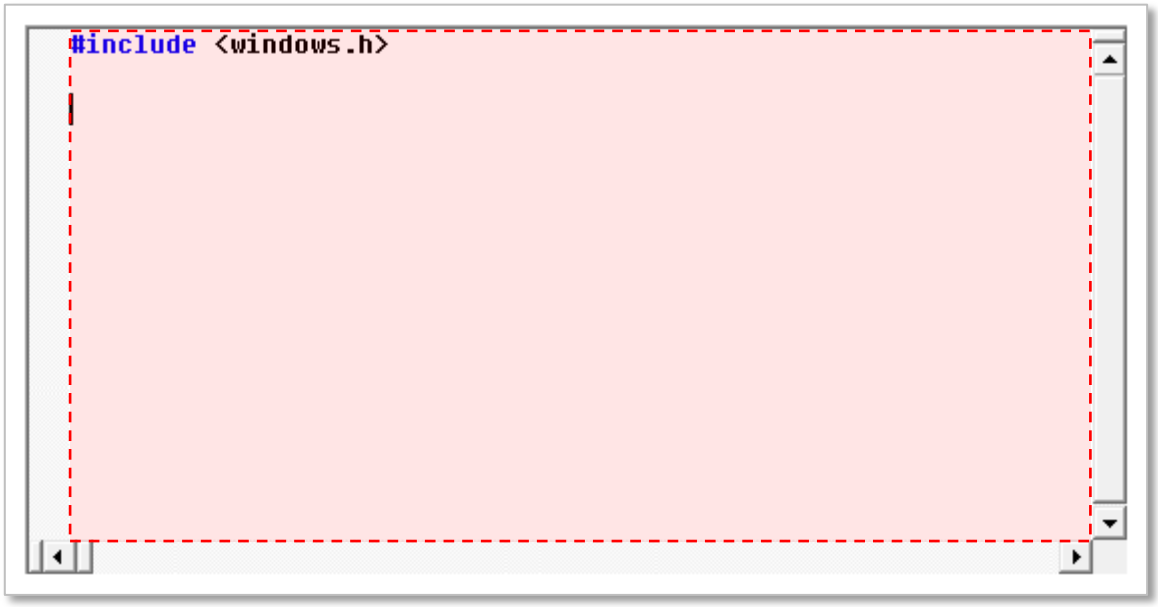
① 메뉴 → File → New Files



② “C++ Source File” 선택 → 소스파일 이름 부여

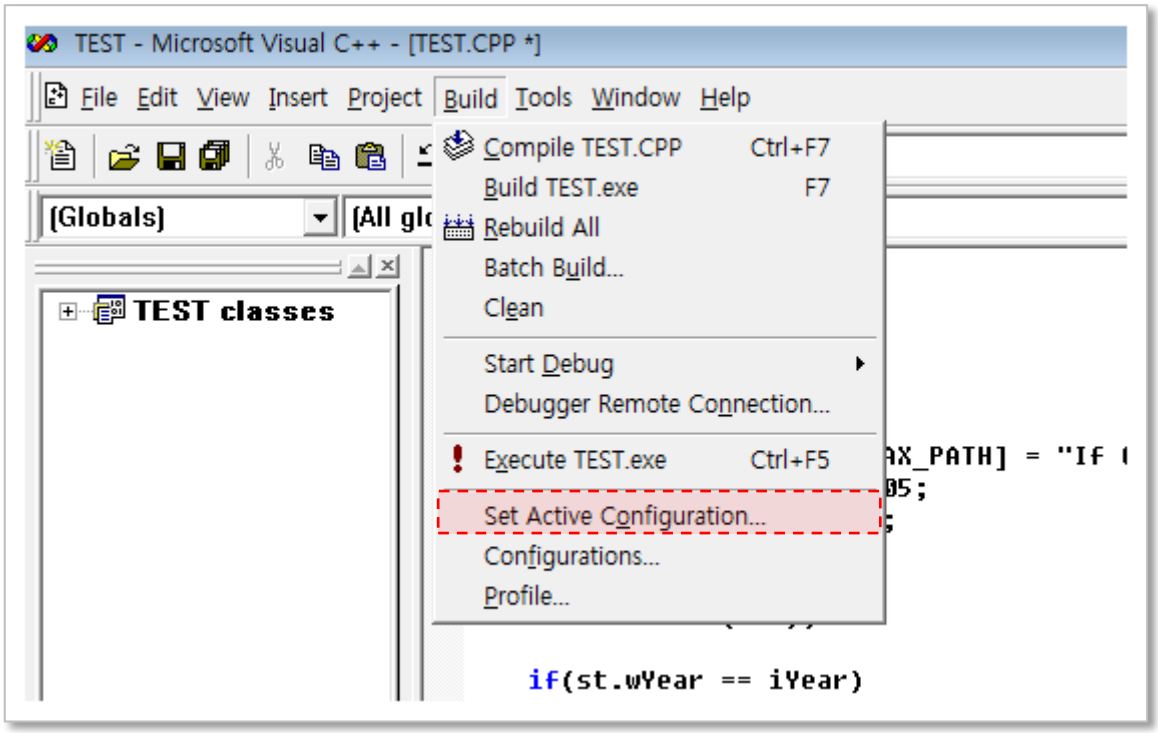


③ 생성된 소스파일에 코드 작성

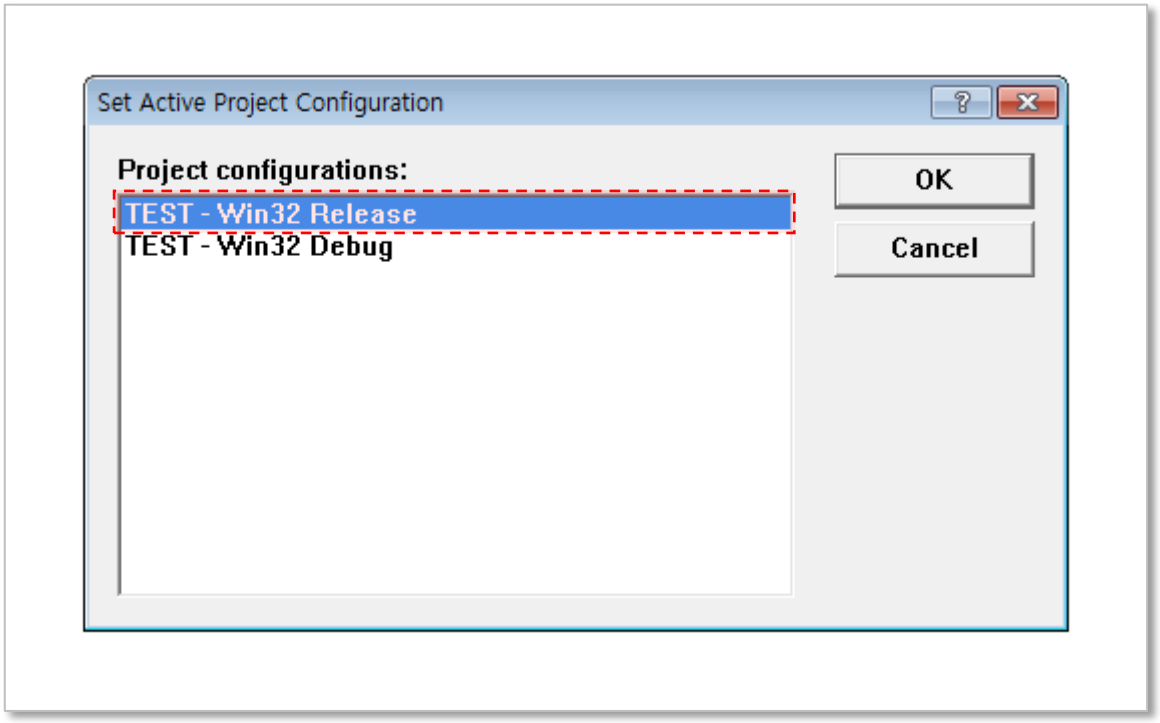


작성이 끝난 코드를 실행파일로 만들려면 릴리즈 모드로 설정을 변경해줘야 합니다.

① 메뉴 → Build → Set Active Configuaration…



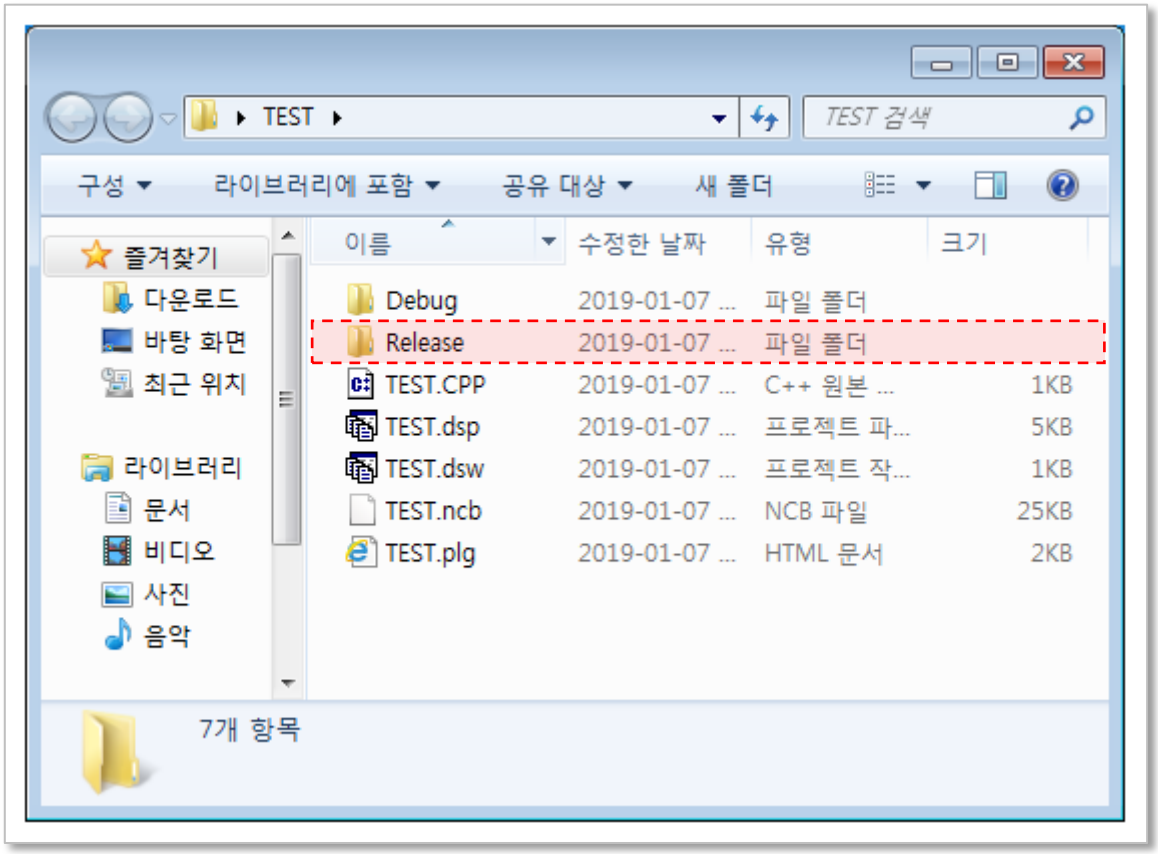
② “프로젝트명 – Win32 Release” 선택



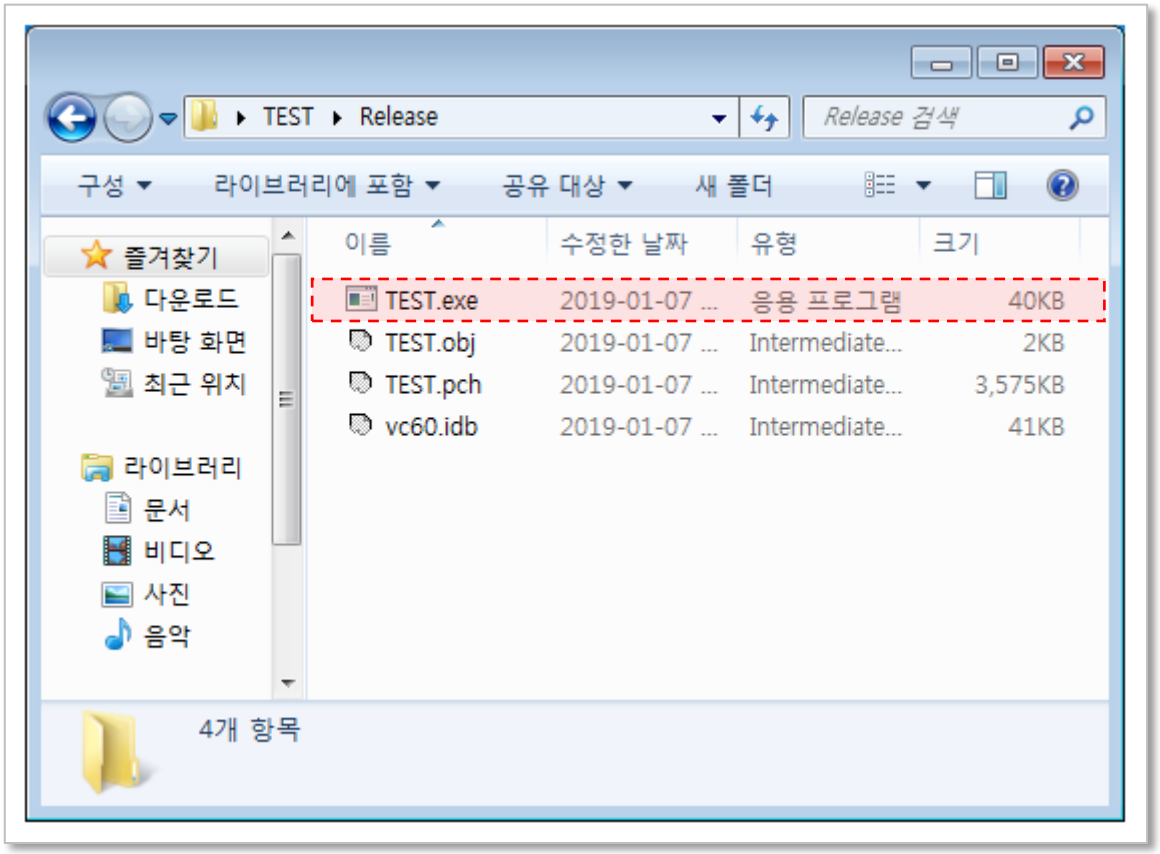
③ 컴파일 (“Ctrl+F7” 또는 “Ctrl+F5”)

- Compile: Ctrl+F7
- Build: F7
- Excute: Ctrl+F5

④ 프로젝트 내 릴리즈 폴더로 이동



⑤ 실행파일 생성 확인


TEST ▸ Release
Release 검색
구성
라이브러리에 포함
공유 대상
새 폴더
즐겨찾기
다운로드
바탕 화면
최근 위치
라이브러리
문서
비디오
사진
음악
이름
수정한 날짜
유형
크기
TEST.exe 2019-01-07 ... 응용 프로그램 40KB
TEST.obj 2019-01-07 ... Intermediate... 2KB
TEST.pch 2019-01-07 ... Intermediate... 3,575KB
vc60.idb 2019-01-07 ... Intermediate... 41KB
4개 항목

2. 데이터 표기 방식

데이터를 바라볼 때, 2가지 기준을 가지고 접근해야 합니다.

- 데이터의 종류
- 데이터 표기 방식

① 데이터의 종류

데이터는 숫자와 문자로 구성되어 있습니다. 조금 더 세부적으로 나누면 주소정보, 상수, 문자 정도가 될 수 있지 않을까 싶습니다. 하나씩 살펴보겠습니다.

다음 어셈블리 코드가 있습니다.

```
PUSH 0x4E000
CALL DOWRD PTR DS:[0x00400000]
```

여기서 "0x4E000"는 상수이고, "0x400000"은 주소정보 입니다. 어떻게 알 수 있냐고요? 이 둘은 데이터의 쓰임에 차이가 있습니다. PUSH 명령어는 그 뒤에 상수 값을 가집니다. 반면 CALL 명령어에서 "[]" 안에는 주소 값이 들어가죠. 이걸 보면 데이터는 어떻게 쓰이냐에 따라 그 성질이 결정 된다는 것을 알 수 있습니다. (물론 스택에 입력된 값이 추후에 주소를 가리키는 용도로 활용될 여지는 충분하지만 그렇게까지 앞서 나가진 않겠습니다.)

문자도 마찬가지입니다. 예를 들어서 "0x53"은 16진수 데이터입니다. 그러나 문자 출력코드를 거치면 대문자 'S'가 출력됩니다. 이는 "0x53"이 문자를 의미하는 값으로 인식됐기 때문입니다.

그래서 문자의 경우, 아스키 코드와 유니코드 체계를 알아야 합니다. 그렇게 어려운건 아니고요. 미리 약속된 값들이 표로 정리되어 있습니다. 이 표를 볼 줄 알면 됩니다. 예를 들어볼까요? 다음과 같은 아스키 코드들이 있습니다.

```
0x53, 0x65, 0x63, 0x75, 0x72, 0x69, 0x74, 0x79, 0x46, 0x61, 0x63, 0x74,
0x6F, 0x72, 0x79
```

이 값들이 무엇을 의미하는지 모르겠네요. 아스키 코드표를 참고해서 하나씩 바꿔보도록 하죠.

Hex	Bin	Symbol	Hex	Bin	Symbol
……			5D	01011101	]
53	01010011	S	5E	01011110	^
54	01010100	T	5F	01011111	_

55	01010101	U	60	01100000	‘
56	01010110	V	61	01100001	a
57	01010111	W	62	01100010	b
58	01011000	X	63	01100011	c
59	01011001	Y	64	01100100	d
5A	01011010	Z	65	01100101	e
5B	01011011	[	66	01100110	f
5C	01011100	₩	……		

아스키 코드 표에 대한 설명: http://shaeod.tistory.com/760

16진수 “53 65 63 75 72 69 74 79 46 61 63 74 6F 72 79” 값은 “SecurityFactory”를 의미합니다.

Hex	53	65	63	75	72	69	74	79	46	61	63	74	6F	72	79
Symbol	S	e	c	u	r	i	t	y	F	a	c	t	o	r	y

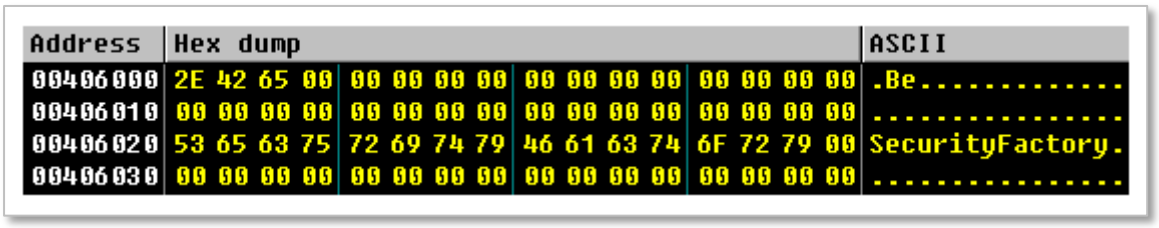
쉽지 않나요? 이게 답니다. 그러면 이 값이 숫자인지 문자인지 어떻게 알 수 있냐고요? 앞에서 얘기했듯이 코드를 보면 알 수 있습니다.

② 데이터 표기 방식

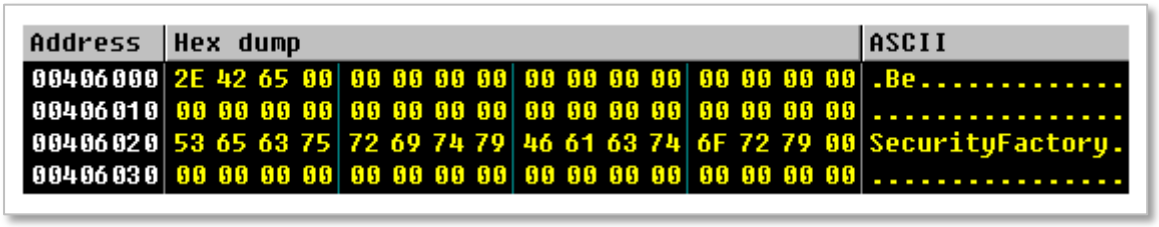
데이터 표기방식에 대해서도 알아야 합니다. “리틀 엔디안 표기법”과 “빅 엔디안 표기법”이라고 한번쯤은 들어봤을 겁니다. 이 둘을 구분 짓는 이유는 CPU 제조사에 따라 데이터를 기록하는 방식에 차이가 있기 때문입니다.

여기서 우리가 가장 많이 쓰는 인텔 CPU는 “리틀 엔디안 표기법”을 사용합니다. 값을 역순으로 기록하는 방식이죠. 그런데 여기서 우리는 CPU가 값을 어떻게 기록하는지 보다 이 값들을 어떻게 읽어야 하는지를 아는 것이 더 중요합니다. 숫자는 반대로 읽고, 문자는 그대로 읽으면 됩니다.

예를 들어 봅시다. 0x00406000 주소에 4바이트 데이터가 기록되어 있습니다. 이 값은 숫자입니다. 그래서 0x0065422E이라고 읽습니다.



이 값이 문자라면 어떻게 될까요?



Address	Hex dump				ASCII
00406000	2E 42 65 00	00 00 00 00	00 00 00 00	00 00 00 00	.Be.....
00406010	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
00406020	53 65 63 75	72 69 74 79	46 61 63 74	6F 72 79 00	SecurityFactory.
00406030	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....

문자열 ".Be"가 되고 마지막 1byte는 NULL을 의미하게 됩니다. (NULL은 '문자열이 끝났다.'는 표식 정도로 이해하면 됩니다.)

Hex	2E	42	65	00
Symbol	.	B	e	NULL

3. 변수와 구조체

변수는 말 그대로 주머니 입니다. 물론 최대로 담을 수 있는 한계가 있지만, 구슬이 최대 10개까지 들어가는 주머니라고 했을 때, 1개를 넣을 수도 있고 5개를 넣을 수도 있죠. 가변적으로 데이터를 담을 수 있다는 말입니다.

구조체는 뭐랄까 정해진 값들을 정해진 크기만큼 담을 수 있도록 만들어진 주머니입니다. 그래서 값을 담을 때, 정해놓은 규칙에 맞게 담아야 합니다.

그럼 두 개의 주머니에 같은 정보를 한번 담아보겠습니다.

김상철의 집 주소는 울산 광역시이고, 전화번호는 052-111-1111이다.

이 름	김상철
주 소	울산 광역시
전화번호	052-111-1111

차이가 느껴지나요? 우선 첫 번째 표에는 이름, 주소 정보 외에도 출신 학교, 가족관계 등 더 많은 정보를 담을 수 있어 보이네요. 그런데 두 번째는 그렇지 못합니다. 대신 필요한 정보를 빨리 찾고자 할 땐, 두 번째 방식이 더 좋아 보입니다. 관리의 용이성이 있습니다. 이처럼 같은 주머니이지만 차이를 가지고 있습니다.

4. PE 헤더를 구성하는 구조체 원형

PE 헤더는 작은 구조체들이 모여서 만들어진 하나의 데이터 덩어리입니다. 우리가 교재에서 확인한 헤더 값들이 구조체 주머니에 넣어서 만들어진 형태인 겁니다. 그렇다면 실제 각 구조체의 원형은 어떻게 될까요? 확인해 봅시다.

① IMAGE_DOS_HEADER 구조체

```
typedef struct _IMAGE_DOS_HEADER { // DOS .EXE header
    WORD       e_magic;       // Magic number (Signature)
    WORD       e_cblp;        // Bytes on last page of file
    WORD       e_cp;          // Pages in file
    WORD       e_crlc;        // Relocations
    WORD       e_cparhdr;     // Size of header in paragraphs
    WORD       e_minalloc;    // Minimum extra paragraphs needed
    WORD       e_maxalloc;    // Maximum extra paragraphs needed
    WORD       e_ss;          // Initial (relative) SS value
    WORD       e_sp;          // Initial SP value
    WORD       e_csum;        // Checksum
    WORD       e_ip;          // Initial IP value
    WORD       e_cs;          // Initial (relative) CS value
    WORD       e_lfarlc;      // File address of relocation table
    WORD       e_ovno;        // Overlay number
    WORD       e_res[4];      // Reserved words
    WORD       e_oemid;       // OEM identifier (for e_oeminfo)
    WORD       e_oeminfo;     // OEM information; e_oemid specific
    WORD       e_res2[10];    // Reserved words
    LONG       e_lfanew;      // File address of new exe header (NT헤더의 시작 위
치)
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

② IMAGE_NT_HEADERS 구조체

```
typedef struct _IMAGE_NT_HEADERS {
    DWORD                   Signature;
    IMAGE_FILE_HEADER       FileHeader;
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

③ IMAGE_FILE_HEADER 구조체

```
typedef struct _IMAGE_FILE_HEADER
{
    WORD       Machine;
    WORD       NumberOfSections;
    ULONG      TimeDateStamp;
    ULONG      PointerToSymbolTable;
    ULONG      NumberOfSymbols;
    WORD       SizeOfOptionalHeader;
    WORD       Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

④ IMAGE_OPTIONAL_HEADER32 구조체

```
typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD             VirtualAddress;
    DWORD             Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;

#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES    16

typedef struct _IMAGE_OPTIONAL_HEADER {
    WORD                        Magic;
    BYTE                        MajorLinkerVersion;
    BYTE                        MinorLinkerVersion;
    DWORD                       SizeOfCode;
    DWORD                       SizeOfInitializedData;
    DWORD                       SizeOfUninitializedData;
    DWORD                       AddressOfEntryPoint;
    DWORD                       BaseOfCode;
    DWORD                       BaseOfData;
    DWORD                       ImageBase;
    DWORD                       SectionAlignment;
    DWORD                       FileAlignment;
    WORD                        MajorOperatingSystemVersion;
    WORD                        MinorOperatingSystemVersion;
    WORD                        MajorImageVersion;
    WORD                        MinorImageVersion;
    WORD                        MajorSubsystemVersion;
    WORD                        MinorSubsystemVersion;
    DWORD                       Win32VersionValue;
    DWORD                       SizeOfImage;
    DWORD                       SizeOfHeaders;
    DWORD                       CheckSum;
    WORD                        Subsystem;
    WORD                        DllCharacteristics;
    DWORD                       SizeOfStackReserve;
```

```
    DWORD                   SizeOfStackCommit;
    DWORD                   SizeOfHeapReserve;
    DWORD                   SizeOfHeapCommit;
    DWORD                   LoaderFlags;
    DWORD                   NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY    DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

⑤ IMAGE_SECTION_HEADER 구조체

```
#define IMAGE_SIZEOF_SHORT_NAME     8

typedef struct _IMAGE_SECTION_HEADER {
    BYTE       Name[IMAGE_SIZEOF_SHORT_NAME];
    union {
        DWORD   PhysicalAddress;
        DWORD   VirtualSize;
    } Misc;
    DWORD      VirtualAddress;
    DWORD      SizeOfRawData;
    DWORD      PointerToRawData;
    DWORD      PointerToRelocations;
    DWORD      PointerToLinenumbers;
    WORD       NumberOfRelocations;
    WORD       NumberOfLinenumbers;
    DWORD      Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

⑥ IMAGE_IMPORT_DESCRIPTOR 구조체

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD   Characteristics;
        DWORD   OriginalFirstThunk;    //INT (Import Name Table) address  (RVA)
    };
    DWORD   TimeDateStamp;
    DWORD   ForwarderChain;
    DWORD   Name;                     //Library name string address (RVA)
    DWORD   FirstThunk;               //IAT(Import Address Table) address (RVA)
} IMAGE_IMPORT_DESCRIPTOR;
typedef IMAGE_IMPORT_DESCRIPTOR UNALIGNED *PIMAGE_IMPORT_DESCRIPTOR;
```

⑦ IMAGE_IMPORT_BY_NAME 구조체

```
typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD    Hint;       //함수의 고유번호
    BYTE    Name[1];    //함수의 이름
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

리버싱 이 정도는 알아야지

발행일 | 2018 년 11 월
발행자 | SecurityFactory
페북 주소 | http://bitly.kr/OrHQ
이메일 | itseeyou@naver.com


SECURITY FACTORY
http://securityfactory.tistory.com