7.11 2019

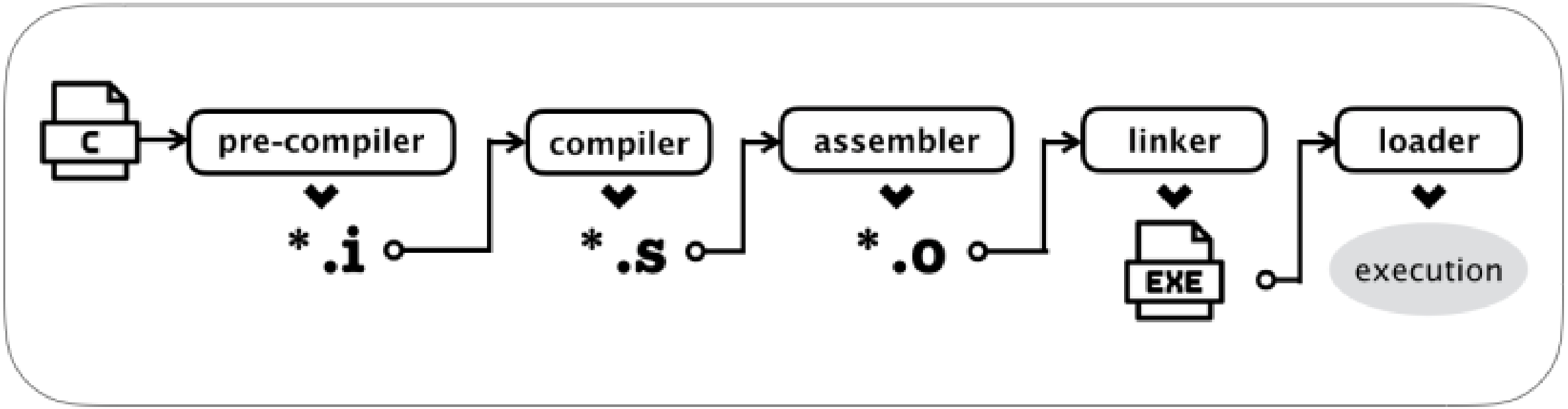
Jaehoon

C++ in Python

contents

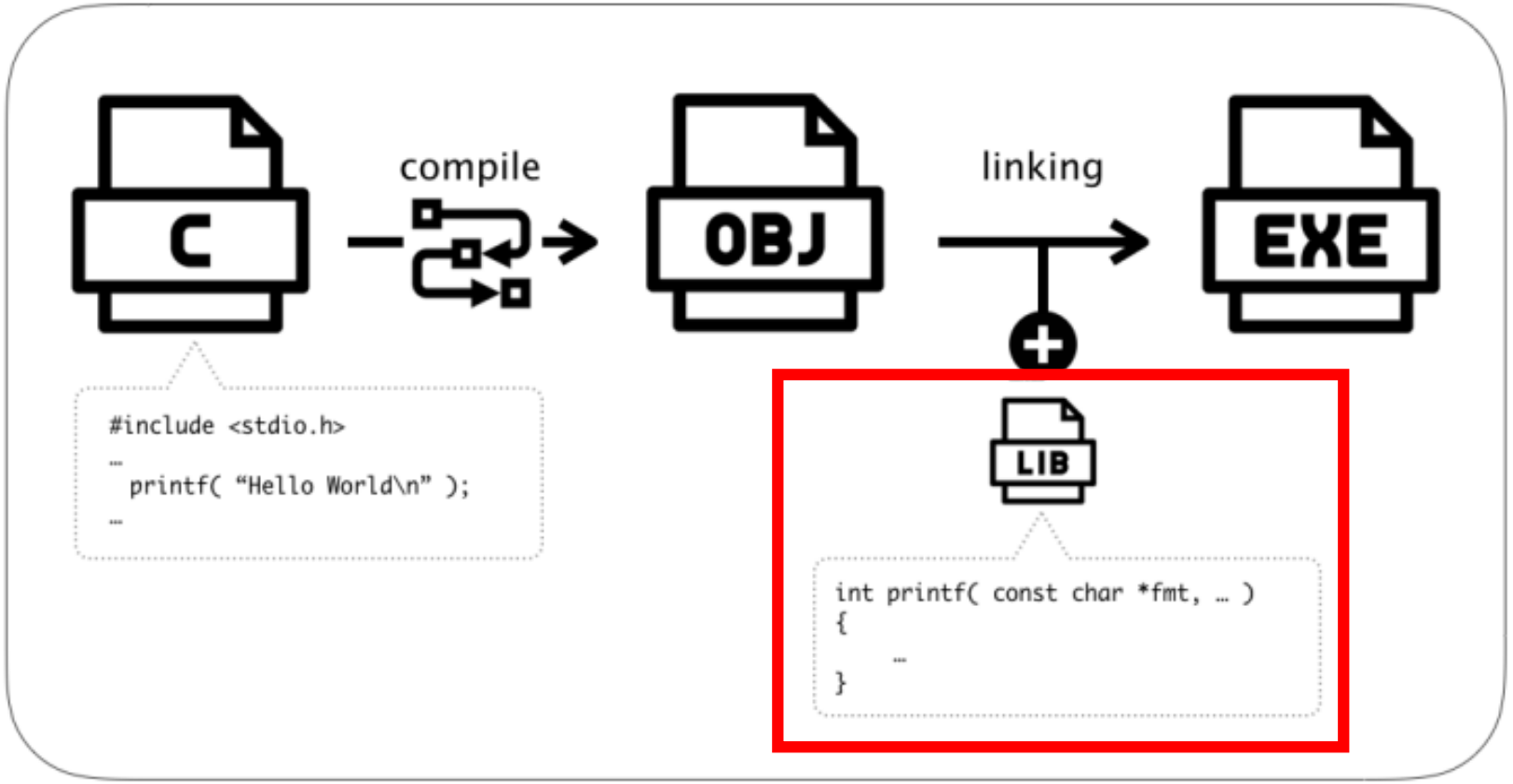
- GNU C Compile
- Dll & So
- C 오브젝트 파일
- 함수 오버로딩
- C++ 오브젝트 파일(mangling)
- How to use C++ in python

GNU C Compile



출처 : https://bpsecblog.wordpress.com/2016/03/07/about_got_plt_1/

GNU C Compile



DLL & SO

DLL

- Dynamic Linked Library
- 동적 링크 라이브러리
- Windows

SO

- Shared Object file
- 공유 라이브러리
- Linux

C로 작성된 오브젝트 파일

```
root@kali:~/Documents/tmp/test# ca
int plus(int num)
{
    int i,sum=0;
    for(i=1;i<=num;i++)
    {
        sum+=i;
    }
    return sum;
}
```

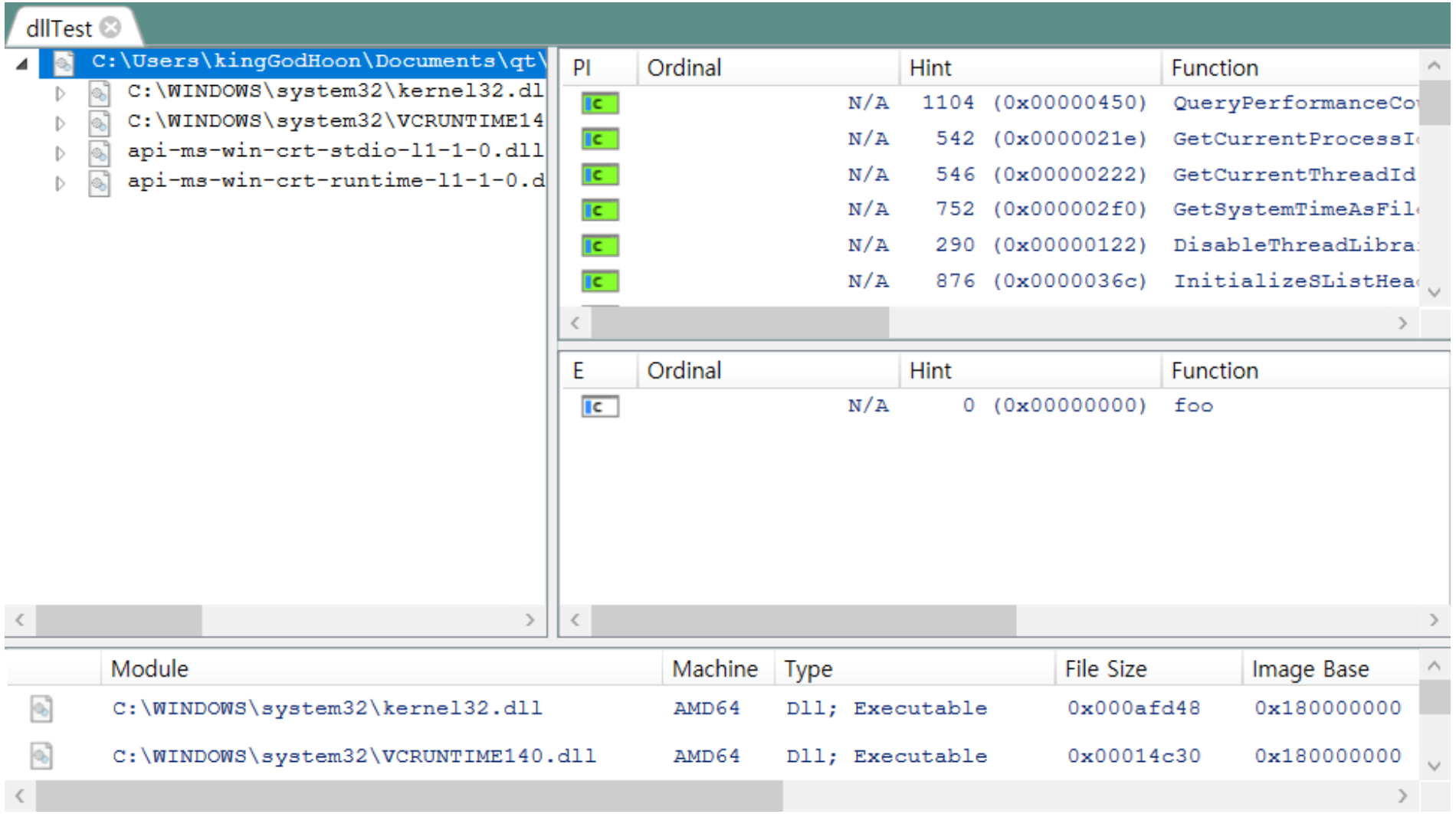
```
root@kali:~/Documents/tmp/test# objdump -d plus.o

plus.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <plus>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   89 7d ec                mov    %edi,-0x14(%rbp)
   7:   c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%rbp)
   e:   c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp)
  15:   eb 0a                   jmp    21 <plus+0x21>
  17:   8b 45 fc                mov    -0x4(%rbp),%eax
  1a:   01 45 f8                add    %eax,-0x8(%rbp)
  1d:   83 45 fc 01             addl   $0x1,-0x4(%rbp)
  21:   8b 45 fc                mov    -0x4(%rbp),%eax
  24:   3b 45 ec                cmp    -0x14(%rbp),%eax
  27:   7e ee                   jle    17 <plus+0x17>
  29:   8b 45 f8                mov    -0x8(%rbp),%eax
  2c:   5d                      pop    %rbp
  2d:   c3                      retq
root@kali:~/Documents/tmp/test#
```

C로 작성된 오브젝트 파일

```
root@kali:~/Documents/tmp/test# cat plus.c
int plus(int num)
{
    int i,sum=0;
    for(i=1;i<=num;i++)
    {
        sum+=i;
    }
    return sum;
}
root@kali:~/Documents/tmp/test# nm plus.o
0000000000000000 T plus
root@kali:~/Documents/tmp/test#
```

nm 명령어로 확인 ㄱㄴ

windows의 경우 Dependencies 사용



함수의 오버로딩

```
1  void func()
2  {
3      cout << " function(void) " << endl;
4  }
5
6  void func(int a, int b)
7  {
8      cout << " function(int a, int b) " << endl;
9  }
10
11 void func(char c)
12 {
13     cout << " function(char c) " << endl;
14 }
```

객체지향에서만 ㄱㄴ
Ex) C++, JAVA

오버라이딩과는 다르다! 혼동 ㄴㄴ

C++ 의 오브젝트 파일

함수 오버로딩으로 인해 이름이 그지같음!

```
root@kali:~/Documents/ccit/build-send_arp-Desktop_Qt_5_11_2_GCC_64bit-Debug# nm arp.o | grep ' T '
0000000000000ac0 T _ZN3Arp14spoofingThreadEjj
00000000000000a0 T _ZN3Arp6getMacEj
0000000000000b04 T _ZN3Arp6isSameEPhS0_
0000000000000492 T _ZN3Arp7findMacEj
00000000000004ce T _ZN3Arp7sendArpEjj
00000000000006e4 T _ZN3Arp8arpSpoofEjj
0000000000000000 T _ZN3ArpC1EPc
0000000000000000 T _ZN3ArpC2EPc
root@kali:~/Documents/ccit/build-send_arp-Desktop_Qt_5_11_2_GCC_64bit-Debug# nm --demangle arp.o | grep ' T '
0000000000000ac0 T Arp::spoofingThread(unsigned int, unsigned int)
00000000000000a0 T Arp::getMac(unsigned int)
0000000000000b04 T Arp::isSame(unsigned char*, unsigned char*)
0000000000000492 T Arp::findMac(unsigned int)
00000000000004ce T Arp::sendArp(unsigned int, unsigned int)
00000000000006e4 T Arp::arpSpoof(unsigned int, unsigned int)
0000000000000000 T Arp::Arp(char*)
0000000000000000 T Arp::Arp(char*)
root@kali:~/Documents/ccit/build-send_arp-Desktop_Qt_5_11_2_GCC_64bit-Debug#
```

Python ctypes 모듈

dll = windll.LoadLibrary('kernel32.dll')
dll.[function name]([parameters...])

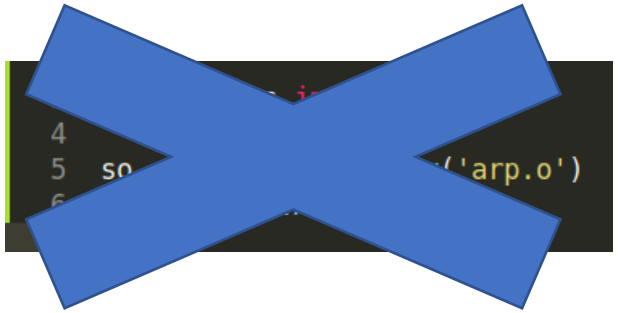
so = cdll.LoadLibrary('libc.so.6')
so.[function name]([parameters...])

```
c:\[redacted]>test.py
HelloWorld!

c:\[redacted]>
```

```
#include "dlltest.h"
#include <stdio.h>

extern "C"{

__declspec(dllexport) void foo(char* param){
    printf("%s", param);
}

}
```

```
#! python2.7

from ctypes import *

dll = windll.LoadLibrary('dllTest.dll')
dll.foo("HelloWorld!\n")
```

C++



객체 지향이기 때문에 맹글링 되어있다.

객체 지향

객체의 포인터를 핸들로 받아서 처리

extern "C"를 이용해 맹글링 하지 않게 하고 C타입으로 변환

ex

C++

객체를 생성할 때
class의 포인터 반환

```
class Foo{
        foo();
        bar();
        baz();
}
```

python

반환된 값을 받아서 handle 값으로 저장하고
클래스 내부 함수를 호출할 때는 handle을 넘겨줌

```
dll = windll.LoadLibrary('dllTest.dll')
handle = int(dll.create_foo())

dll.use_foo(handle)
```

C타입으로 바꾼 dll이나 so파일

handle 값을 받아서 객체의 인스턴스를 생성해서 캐스팅 한 후 내부 변수 사용 가능

```
extern "C"{
__declspec(dllexport) void* create_foo(){
        return 객체포인터;
}

__declspec(dllexport) int use_foo(handle){
        Foo *f = new Foo;
        f = reinterpret_cast<void*>handle;
        f.foo();
        f.bar();
        f.baz();
}
```

꿀팁쓴

```
#ifdef WIN32
#define EXPORT __declspec(dllexport)
#endif
#ifdef linux
#define EXPORT
#endif
```