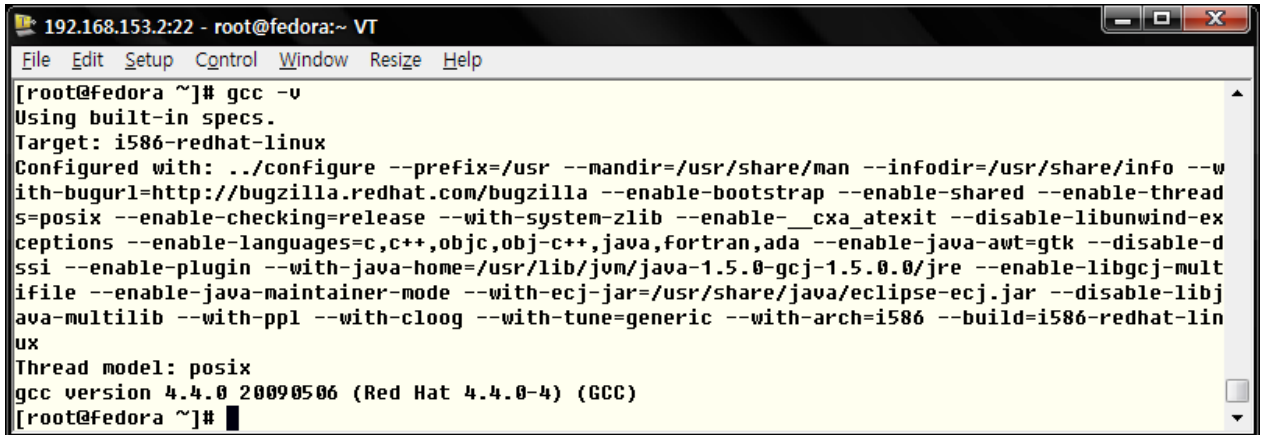


gcc 사용법

1. gcc 개요

프롬프트 상태에서 다음과 같이 입력해보자.



```

192.168.153.2:22 - root@fedora:~ VT
File Edit Setup Control Window Resize Help
[root@fedora ~]# gcc -v
Using built-in specs.
Target: i586-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-bootstrap --enable-shared --enable-threads=posix --enable-checking=release --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-languages=c,c++,objc,obj-c++,java,fortran,ada --enable-java-awt=gtk --disable-dssi --enable-plugin --with-java-home=/usr/lib/jvm/java-1.5.0-gcj-1.5.0.0/jre --enable-libgcj-multifile --enable-java-maintainer-mode --with-ecj-jar=/usr/share/java/eclipse-ecj.jar --disable-libjava-multilib --with-ppl --with-cloog --with-tune=generic --with-arch=i586 --build=i586-redhat-linux
Thread model: posix
gcc version 4.4.0 20090506 (Red Hat 4.4.0-4) (GCC)
[root@fedora ~]#
  
```

위에 출력된 내용은 현재 시스템에 설치된 gcc 버전을 비롯한 여러 정보이다.

2. gcc가 실행 시키는 프로그램

우리가 프롬프트상에서 gcc 명령을 내리게 되면 Batch Processing 되는 것과 같이 다음의 프로그램이 차례로 실행되는 일련의 과정을 거치게 된다.

즉, gcc가 실행시키는 프로그램은 다음과 같다.

```

cpp      : 전처리기
ccache   : 컴파일러
as       : 어셈블러
ld       : 링커
  
```

ccache 이 진짜 C 컴파일러 본체이다. gcc 는 단지 적절하게 C 인가, C++ 인가 아니면 검사하고 컴파일 작업만 아니라 "링크"라는 작업까지 하여 C 언어로 프로그램 소스를 만든 다음, 프로그램 바이너리가 만들어지기까지의 모든 과정을 관장해주는 "조정자" 역할을 할 뿐이다.

cpp는 전처리기로써 #include 등의 문장을 본격적인 ccache 컴파일에 들어 가기에 앞서 먼저(pre) 처리(process)해주는 역할을 한다.

다음의 소스를 입력한 후 hello.c로 저장하고 gcc를 이용해 컴파일을 해보자

```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# cat hello.c
#include <stdio.h>

int main(){
    printf("Hello, Linuxers Please!\n");

    return 0;
}
[root@fedora C Study]#

```

다음과 같이 결과 프로그램을 실행시키려 하면 hello라는 명령이 없다는 에러 메시지가 출력된다. 이것은 C 프로그램을 옵션 없이 컴파일 했을 때 소스파일과 다른 이름으로 컴파일된 실행파일이 생성된다는 것을 의미한다. 즉, 위와 같이 아무런 옵션 없이 소스파일명만으로 컴파일 하면 기본적으로 a.out이라는 파일이 생성된다.

파일을 실행하고자 할 때에는 기본적으로 디렉토리명도 명시를 하여야 한다.

```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# ./a.out
Hello, Linuxers Please!
[root@fedora C Study]#

```

맨 앞의 점 (.)은 현재 디렉토리를 의미한다. 그 다음 디렉토리 구분 문자 슬래시(/)를 쓰고 C 컴파일러의 출력 결과 바이너리 파일인 a.out 을 써준다.

리눅스는 기본적으로 PATH 라는 환경변수에 있는 디렉토리에서만 실행파일을 찾으려고 한다. 만약 PATH 라는 환경변수에 현재 디렉토리를 의미하는 점(.)이 들어있지 않으면 현재 디렉토리의 실행파일은 실행되지 않는다. 게다가 현재 디렉토리를 PATH 환경 변수에 넣어준다 할 지라도 도스처럼 현재 디렉토리를 먼저 찾는든지 하는 일은 없다. 오로지 PATH 에 지정한 순서대로 수행할 뿐이다.

다음은 gcc 컴파일 명령 사용 시 사용할 수 있는 옵션이다.

옵션	설명
-o<file>	옵션(소문자 o)은 출력(output) 파일명을 정하는 옵션이다.
-c	오로지 컴파일(compile) 작업만 하고 싶은 경우가 있는데, 그럴 때는 다음과 같이 한다.
-I<dir>	옵션은 아주 많이 사용되는 옵션중 하나인데, 이것은 #include 문장에서 지정한 헤더 파일이 들어있는 곳을 정한다.
-l<lib> (소문자 L)	컴파일 시 어떤 라이브러리를 사용할 것인지 라이브러리명을 지정하는 옵션이다.
-L<dir>	컴파일 시 참조한 라이브러리가 위치한 디렉토리 경로를 지정하는 옵션이다.
-O<level>	Optimize Level을 설정한다. 이 option을 주면 실행파일의 크기가 작아지고, 실행속도가 빨라지나 그만큼 위험성이 따른다. 대부분의 경우 -O1 또는 -O2를 사용한다.
-E	전처리만 실행하며 컴파일은 하지 않는다.

-S	컴파일만 실행하며 어셈블이나 링크를 하지 않는다.
-g	운영체제 고유의 형식으로 디버깅 정보를 생성한다.
-D<macro>	매크로를 지정한다.
-Wall	gcc가 제공할 수 있고, 일반적으로 유용한 모든 경고 메시지를 출력한다.

-o 옵션

-o 옵션은 컴파일 후, 생성되는 파일의 이름을 지정하는 옵션이다. 전체 컴파일이라면, 생성할 실행파일명을 지정하며, 컴파일만 하는 경우에는 생성할 오브젝트 파일명을 지정한다.

```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# gcc -o hello hello.c
[root@fedora C Study]# ls -l hello
-rwxrwxr-x 1 root root 4835 2009-10-15 01:46 hello
[root@fedora C Study]# ./hello
Hello, Linuxers Please!
[root@fedora C Study]#

```

-c 옵션

-c 옵션은 컴파일만을 실행한다. -c 옵션을 이용하는 경우, 오브젝트 파일(.o)이 생성이 된다. 즉 hello.c 파일을 컴파일 하면 hello.o 파일이 생성이 된다.

```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# gcc -c hello.c
[root@fedora C Study]# ls -l hello.o
-rw-rw-r-- 1 root root 868 2009-10-15 01:48 hello.o
[root@fedora C Study]#

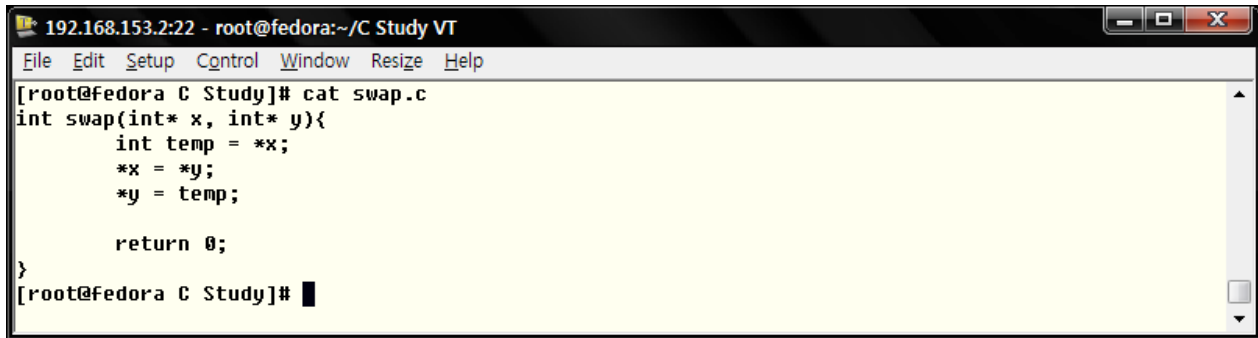
```

오브젝트 파일의 확장자는 .o 이며, 이를 모듈이라고도 불리운다. 즉 하나의 모듈은 하나의 함수를 포함하고 있으며, 하나의 프로그램은 main()함수를 포함하여, 여러개의 함수로 구성이 되어있다. 단, main()함수는 전체모듈내에 하나만이 존재하여야 한다.

-I 옵션(대문자 i)

-I 옵션은 아주 많이 사용되는 옵션중 하나인데, 이것은 #include 문장에서 지정한 헤더 파일이 들어있는 곳을 정한다.

예를들어 아래와 같이 swap 함수를 갖는 별도의 모듈이 존재한다고 가정을 해보자.



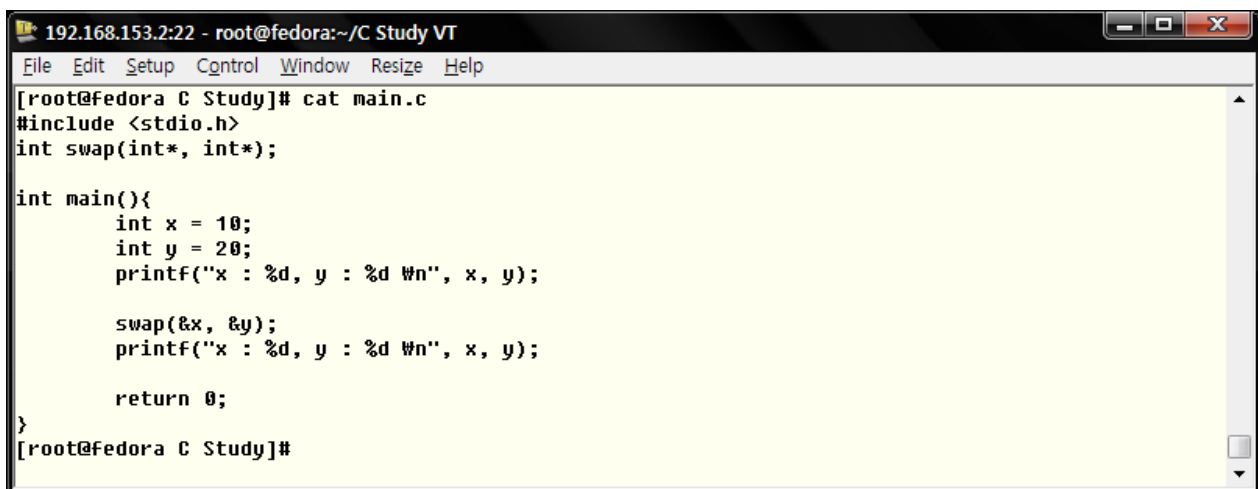
```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# cat swap.c
int swap(int* x, int* y){
    int temp = *x;
    *x = *y;
    *y = temp;

    return 0;
}
[root@fedora C Study]#

```

위의 함수는 두개의 정수형 포인터를 매개변수로 하여 두 변수의 값을 바꾸는 함수이다. 이 함수를 main() 함수에서 사용하기 위해서는 아래와 같이 함수의 원형이 main()함수 이전에 정의 되어있어야 할 것이다.



```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# cat main.c
#include <stdio.h>
int swap(int*, int*);

int main(){
    int x = 10;
    int y = 20;
    printf("x : %d, y : %d \n", x, y);

    swap(&x, &y);
    printf("x : %d, y : %d \n", x, y);

    return 0;
}
[root@fedora C Study]#

```

여기서 발생하는 문제는 함수의 원형을 알기 위해서는, 함수 파일을 열어봐야 한다는 것이다. 다행히 모듈이 소스프로그램으로 되어있는 상태라면 파일을 열어 함수의 구조를 파악하는것도 가능할 것이다. 하지만 이미 컴파일 되어 오브젝트파일로 되어있다면 이것조차 불가능하다. 오브젝트 파일은 이미 기계어인 상태이기 때문이다.

이러한 문제를 해결하기 위해 사용하는것이 헤더파일이다. 헤더파일은 모듈에 포함된 함수의 원형이나, 매크로 등을 포함하고 있다. 아래의 헤더파일은 swap 함수를 참조하기 위한 헤더파일의 내용이다.



```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# cat swap.h
int swap(int*, int*);
[root@fedora C Study]#

```

이렇게 헤더파일을 생성하는 경우, 모듈내의 함수를 사용하기 위해 소스를 분석해야 하는 불편함을 줄일 수가 있다. 아래의 예는 헤더파일을 이용한 예이다.

```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# cat main.c
#include <stdio.h>
#include "swap.h"

int main(){
    int x = 10;
    int y = 20;
    printf("x : %d, y : %d \n", x, y);

    swap(&x, &y);
    printf("x : %d, y : %d \n", x, y);

    return 0;
}
[root@fedora C Study]#

```

인클루드 시킨 헤더파일이 현재 디렉토리에 존재한다면 -I 옵션을 이용하여 별도로 명시하지 않아도 된다. 만약 현재의 디렉토리가 아닌 별도의 디렉토리에 헤더파일이 존재한다면 -I<dir> 옵션을 이용하여 헤더파일이 위치한 디렉토리를 명시하여야 한다.

```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# gcc -I. swap.c main.c
[root@fedora C Study]# ./a.out
x : 10, y : 20
x : 20, y : 10
[root@fedora C Study]#

```

라이브러리 개념

라이브러리란 자주 사용되는 기능, 즉 함수와 같은 특별한 코드를 컴파일 시킨 후 main함수에서 분리시켜 집합시켜 놓은 파일이다. 이렇게 main함수와 분리시켜 놓은 이유는 추후 프로그램의 디버깅이나 유지보수가 더욱 간단해지므로 컴파일을 빠르게 할 수 있기 때문이다. 라이브러리는 우리가 컴파일 과정을 거쳐서 만든 .o 파일을 한 곳에 통털어 관리하는 것에 불과하다.

리눅스에서 제공하는 라이브러리는 크게 "정적라이브러리"와 "공유라이브러리", "동적라이브러리"로 구분되는데, 이렇게 구분되는 특징은 적재되는 시간에 따른 것이다.

정적라이브러리

정적라이브러리(또는 Archive)파일이라 불리우고 .a의 확장자를 가진다. 여러개의 오브젝트 파일들을 하나로 묶어 사용하기 때문에 Archive파일이라고 부르며 컴파일 시간에 코드를 포함하여 결정하기 때문에 정적 라이브러리라고 부른다. 위의 파일을 오브젝트 파일로 컴파일 한다. 정적라이브러리는 모듈파일을 무작위로 집어넣은 것은 아니고 모듈에 대한 인덱스(index)를 가지고 있다.

라이브러리는 다음과 같이 구성되어 있다고 할 수 있다.

라이브러리 = 목차(index) + (a.o + b.o + c.o + ...)

라이브러리 파일을 생성하거나 편집할 때에는 ar 명령을 이용한다.

ar [옵션] [라이브러리명] [모듈명 ...]

ar 명령에 사용가능한 옵션은 다음과 같다.

옵션	설명
t	archive 내용 확인하기
r	archive 생성하기
x	extract 실행
q	파일 추가하기
d	archive 파일 중 지정된 특정 파일을 삭제하기

아래의 예는 /usr/lib/libcom_err.a 라이브러리내의 모듈 목록을 출력한 예이다.

```

192.168.153.2:22 - root@fedora:/usr/lib VT
File Edit Setup Control Window Resize Help
[root@fedora lib]# ar t /usr/lib/libcom_err.a
error_message.o
et_name.o
init_et.o
com_err.o
com_right.o
[root@fedora lib]#
  
```

gcc 에서 사용되어지는 라이브러리는 /usr/lib/ 디렉토리에 위치한다. 또한 라이브러리는 라이브러리 파일명으로 부터 전치사인 lib과 확장자인 .a를 제외하고 읽는다. 즉 위의 예에 사용된 라이브러리의 명칭은 com_err 라이브러리 인것이다.

아래의 예는 com_err 라이브러리의 묶음을 해제한 예이다. x 는 라이브러리내의 모듈을 푸는 옵션이다.

```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# ar x /usr/lib/libcom_err.a
[root@fedora C Study]# ls
com_err.o  com_right.o  error_message.o  et_name.o  init_et.o
[root@fedora C Study]#
  
```

아래의 예는 모듈을 묶어서 라이브러리를 생성한 예이다. r 옵션은 하나 이상의 모듈을 이용하여 라이브러리를 생성한다.

```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# ar r libmylib.a com_err.o com_right.o error_message.o
[root@fedora C Study]# ls
com_err.o com_right.o error_message.o et_name.o init_et.o libmylib.a
[root@fedora C Study]# ar t libmylib.a
com_err.o
com_right.o
error_message.o
[root@fedora C Study]#

```

아래의 예는 이미 존재하는 라이브러리로부터 모듈을 추가한 예이다.

```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# ar q libmylib.a init_et.o
[root@fedora C Study]# ar t libmylib.a
com_err.o
com_right.o
error_message.o
init_et.o
[root@fedora C Study]#

```

라이브러리내의 모듈삭제는 d 옵션을 이용한다.

```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# ar d libmylib.a com_right.o
[root@fedora C Study]# ar t libmylib.a
com_err.o
error_message.o
init_et.o
[root@fedora C Study]#

```

동적라이브러리

동적라이브러리는 실행 시간에 라이브러리 파일을 찾아 코드를 포함하기 때문에 동적라이브러리라고 부른다. 즉, 정적 라이브러리는 컴파일 시간에 코드를 결정하기 때문에 소스가 변경되면 새로 컴파일 해야하지만, 동적 라이브러리의 경우 라이브러리만 교체하는 것으로도 변경이 가능하다. 동적라이브러리의 확장자는 .so 이다. 이는 Shared Object 를 의미한다.

동적라이브러리를 생성하기 위해서는 파일 자체가 동적모듈로 생성이 되어야 한다. 동적모듈은 다음과 같이 생성한다.

```
gcc -c -PIC 모듈소스파일
```

-fPIC옵션은 Position-Independent Code의 약자이며 test.o파일을 동적라이브러리로 사용하도록 컴파일 하는 옵션이다. 이렇게 컴파일된 오브젝트 파일을 .so파일로 묶으면 된다.

gcc -shared -fPIC -o 라이브러리파일명 모듈파일명 [모듈파일명 ...]

아래의 예는 swap.c 파일을 이용하여 동적라이브러리를 생성한 예이다.

```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# cat swap.c
int swap(int* x, int* y){
    int temp = *x;
    *x = *y;
    *y = temp;

    return 0;
}
[root@fedora C Study]# gcc -c -fPIC swap.c
[root@fedora C Study]# gcc -shared -fPIC -o libswap.so swap.o
[root@fedora C Study]# ls
libswap.so main.c swap.c swap.h swap.o
[root@fedora C Study]#

```

libswap.so 파일이 동적라이브러리 파일이다. 이제 이 라이브러리 파일을 이용하여 main.c 파일을 컴파일 하도록 하자. 컴파일시 옵션으로 사용할 라이브러리와 라이브러리 파일이 위치한 디렉토리를 명시하여 컴파일 하여야 한다. 만약 헤더파일이 같은 디렉토리가 아닌 다른 디렉토리에 위치한다면 참조한 헤더가 위치한 디렉토리 또한 지정해 주어야 할것이다.

```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# gcc -o exam01 main.c -L./ -lswap -I./
[root@fedora C Study]# ls
exam01 libswap.so main.c swap.c swap.h swap.o
[root@fedora C Study]#

```

위의 설명에서 동적라이브러리는 실행시 라이브러리를 참조한다고 하였다. 즉 동적라이브러리를 이용하는 경우 실행시에도 동적라이브러리가 필요함을 의미한다. 리눅스 시스템에서 동적라이브러리는 /usr/lib/ 디렉토리에 위치한다. 그러므로 라이브러리 파일을 /usr/lib/ 디렉토리에 복사하여야 할것이다.

라이브러리를 사용하기 위해서는 라이브러리를 /usr/lib/ 디렉토리에 복사하는것 뿐만이 아니라, ldconfig 명령을 이용하여 라이브러리를 등록하여야 한다.

```

192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# cp libswap.so /usr/lib/
[root@fedora C Study]# ldconfig
[root@fedora C Study]# ldconfig -v | grep libswap.so
    libswap.so -> libswap.so
[root@fedora C Study]#

```

위의 예는 /usr/lib/ 디렉토리에 swap 라이브러리를 복사한 후, ldconfig 명령을 이용하여 라이브러리를 등록한 예이다. 등록된 라이브러리 목록은 ldconfig -v 명령을 이용하여 확인할 수 있다.

이제 파일을 실행하여 보자.

```
192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# ./exam01
x : 10, y : 20
x : 20, y : 10
[root@fedora C Study]#
```

위에서는 참조할 라이브러리 파일을 /usr/lib/ 디렉토리에 복사하였다. 만약 라이브러리를 /usr/lib/ 디렉토리에 복사하지 않고, 특정 디렉토리에 위치한 라이브러리를 참조하고자 한다면 /etc/ld.so.conf 파일에 라이브러리가 위치한 디렉토리를 명시하도록 하자.

```
192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# cat /etc/ld.so.conf
include ld.so.conf.d/*.conf
[root@fedora C Study]#
```

/etc/ld.so.conf 파일에 직접 명시하여도 되지만, ld.so.conf 파일에서는 /etc/ld.so.conf.d/ 디렉토리에 위치한 확장자가 .conf 인 파일을 모두 인클루드 하게 되어있으므로 /etc/ld.so.conf.d/ 디렉토리에 다음과 같은 파일을 생성하도록 하자.

```
192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# cat /etc/ld.so.conf.d/my.conf
./
[root@fedora C Study]#
```

이제 ldconfig 명령을 이용하여 라이브러리 목록을 재구성하도록 하자.

```
192.168.153.2:22 - root@fedora:~/C Study VT
File Edit Setup Control Window Resize Help
[root@fedora C Study]# ldconfig
[root@fedora C Study]# ldconfig -v | grep libswap.so
libswap.so -> libswap.so
[root@fedora C Study]#
```

여러분이 프로그래밍시 동적라이브러리를 생성하고 사용하고자 한다면, 모든 파일을 컴파일하고, 동적라이브러리를 생성한 후, 반드시 실행하기전에 동적라이브러리를 등록하여야 한다는것을 명심하기 바란다.

-l 옵션

-l 옵션은 참조할 라이브러리를 지정하는 옵션이다. 라이브러리를 지정할 때에는 접두어인 lib과 확장자인 .a 또는 .so 를 삭제한 이름을 지정하여야 한다.

```
gcc main.c -lswap          : libswap.a 또는 libswap.so 를 참조
gcc main.c -lmylib         : libmylib.a 또는 libmylib.so 를 참조
```

-L 옵션

-L 옵션은 참조할 라이브러리가 위치한 디렉토리를 지정하는 옵션이다. 정적라이브러리를 참조한 경우, 실행시 라이브러리가 존재하지 않아도 된다. 하지만 동적라이브러리를 참조한 경우에는 실행시에 라이브러리가 필요하다는 것을 명심하자.