제3장 C 프로그래밍 환경

3.1 컴파일러

gcc 컴파일러

- gcc(GNU cc) 컴파일러 상업용 C 컴파일러(cc)
  $ gcc [-옵션] 파일 $ cc [-옵션] 파일
- 컴파일
  $ gcc long.c
  $ gcc mysrc1.c mysrc2.c
  $ a.out // 실행 파일
- -c 옵션(object code 생성)
  $ gcc -c long.c
  $ gcc long.o
- -o 옵션
  $ gcc -o long long.o
  혹은
  $ gcc -o long long.c
  $ long // 실행 파일

참고(최적화 옵션)

$ gcc –O –o long long.c

기타 컴파일 옵션

- 디버거를 위한 option
  - % gcc -g -c mysrc1.c mysrc2.c mysrc3.c
  - % gcc -o hello mysrc1.o mysrc2.o mysrc3.o

- 외부 라이브러리의 정적 링크
  - % gcc mymath.c –louter
    - 외부 라이브러리 링크
    - eg) libmylib.a 의 경우 –lmylib

정적 라이브러리 파일: lib*.a
동적 라이브러리 파일: lib*.so

정적 라이브러리 파일 생성 및 활용

```
# gcc -c test.c
# gcc -c app.c

# ar cr libtest.a test.o

#gcc -o  app app.o -L. -ltest
```

단일 모듈 프로그램:long.c

```
#include <stdio.h>
#define MAXLINE 100
void copy(char from[], char to[]);
char line[MAXLINE]; // 입력 줄
char longest[MAXLINE]; // 가장 긴 줄
/*입력 줄 가운데 가장 긴 줄 프린트 */
main()
{
    int len;
    int max;
    max = 0;
    while (gets(line) != NULL) {
        len = strlen(line);
        if (len > max) {
            max = len;
            copy(line, longest);
        }
    }
    if (max > 0) // 입력 줄이 있었다면
        printf("%s", longest);

    return 0;
}
/* copy: from을 to에 복사; to가 충분히 크
    다고 가정*/
void copy(char from[], char to[])
{
    int i;
    i = 0;
    while ((to[i] = from[i]) != '₩0')
        ++i;
}
```

다중 모듈 프로그램

- 단일 모듈 프로그램
  - 코드의 재사용(reuse)이 어렵고,
  - 여러 사람이 참여하는 프로그래밍이 어렵다
  - 예를 들어 다른 프로그램에서 copy 함수를 재사용하기 힘들다
- 다중 모듈 프로그램
  - 여러 개의 .c 파일들로 이루어진 프로그램
  - 일반적으로 복잡하며 대단위 프로그램인 경우에 적합

다중 모듈 프로그램: 예

- main 프로그램과 copy 함수를 분리하여 별도 파일로 작성
  - main.c
  - copy.c
  - copy.h // 함수의 프로토타입을 포함하는 헤더 파일

- 컴파일

  $ gcc -c main.c

  $ gcc -c copy.c

  $ gcc -o main main.o copy.o

  혹은

  $ gcc -o main main.c copy.c

main.c

```
#include <stdio.h>
#include "copy.h"
char line[MAXLINE]; // 입력 줄
char longest[MAXLINE]; // 가장 긴 줄
/*입력 줄 가운데 가장 긴 줄 프린트 */
main()
{
    int len;
    int max;
    max = 0;
    while (gets(line) != NULL) {
        len = strlen(line);
        if (len > max) {
            max = len;
            copy(line, longest);
        }
    }
    if (max > 0) // 입력 줄이 있었다면
    printf("%s", longest);

    return 0;
}
```

copy.c

```
#include <stdio.h>
#include "copy.h"
/* copy: from을 to에 복사; to가 충분
    히 크다고 가정*/
void copy(char from[], char to[])
{
    int i;
    i = 0;
    while ((to[i] = from[i]) != '₩0')
        ++i;
}
```

copy.h

```
#define MAXLINE 100
void copy(char from[], char to[]);
```

3.2 make 시스템

make 시스템

- make 시스템
  - 대규모 프로그램의 경우에는 헤더, 소스 파일, 목적 파일, 실행 파일의 모든 관계를 기억하고 체계적으로 관리하는 것이 필요
- 다중 모듈 프로그램을 구성하는 일부 파일이 변경된 경우?
  - 변경된 파일만 컴파일하고, 파일들의 의존 관계에 따라서 필요한 파일만 다시 컴파일하여 실행 파일을 만들면 좋다.
- 예
  - copy.c 소스 코드를 수정
  - 목적 파일 copy.o 생성
  - 실행파일을 생성

makefile

- makefile
  - 실행 파일을 만들기 위해 필요한 파일들과 그들 사이의 의존 관계, 만드는 방법 등을 기술
  - make 시스템은 makefile을 이용하여 파일의 상호 의존 관계를 파악하여 실행 파일을 쉽게 다시 만듬
  - 구성
    - 목표(target), 의존 관계(dependency), 명령(command)의 세개로 이루어진 기본적인 규칙(rule)들이 계속적으로 나열

```
target ... : dependency ...
        command
...
```

명령 부분은 꼭 *TAB* 글자로 시작해야

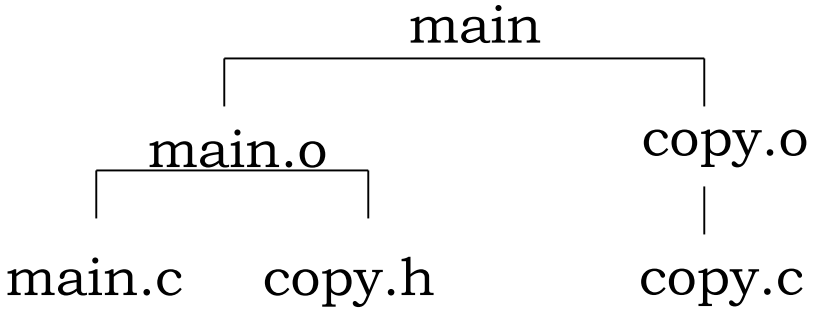
- $ make [-f makefile이름]
  - 옵션이 없으면 Makefile 혹은 makefile을 사

makefile의 구성

- makefile의 구성 형식
  대상리스트: 의존리스트
  명령리스트

- 예: Makefile

```
main:main.o copy.o
        gcc -o main main.o copy.o
main.o: main.c copy.h
        gcc -c main.c
copy.o: copy.c
        gcc -c copy.c
```



makefile의 구성

- make 실행

  $ make 혹은 $ make main

  gcc -c main.c

  gcc -c copy.c

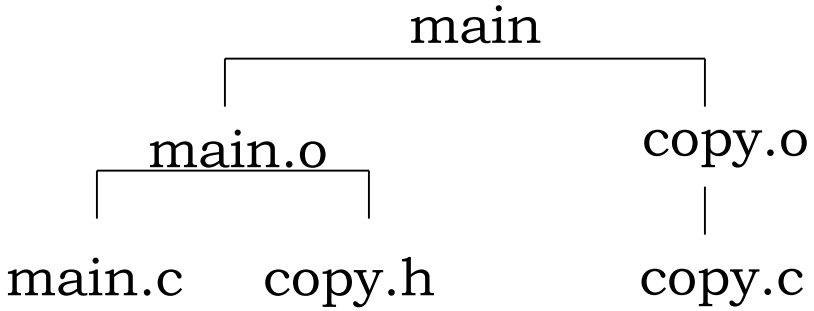
  gcc -o main main.o copy.o

- copy.c 파일이 변경된 후

  $ make

  gcc -c copy.c

  gcc -o main main.o copy.o

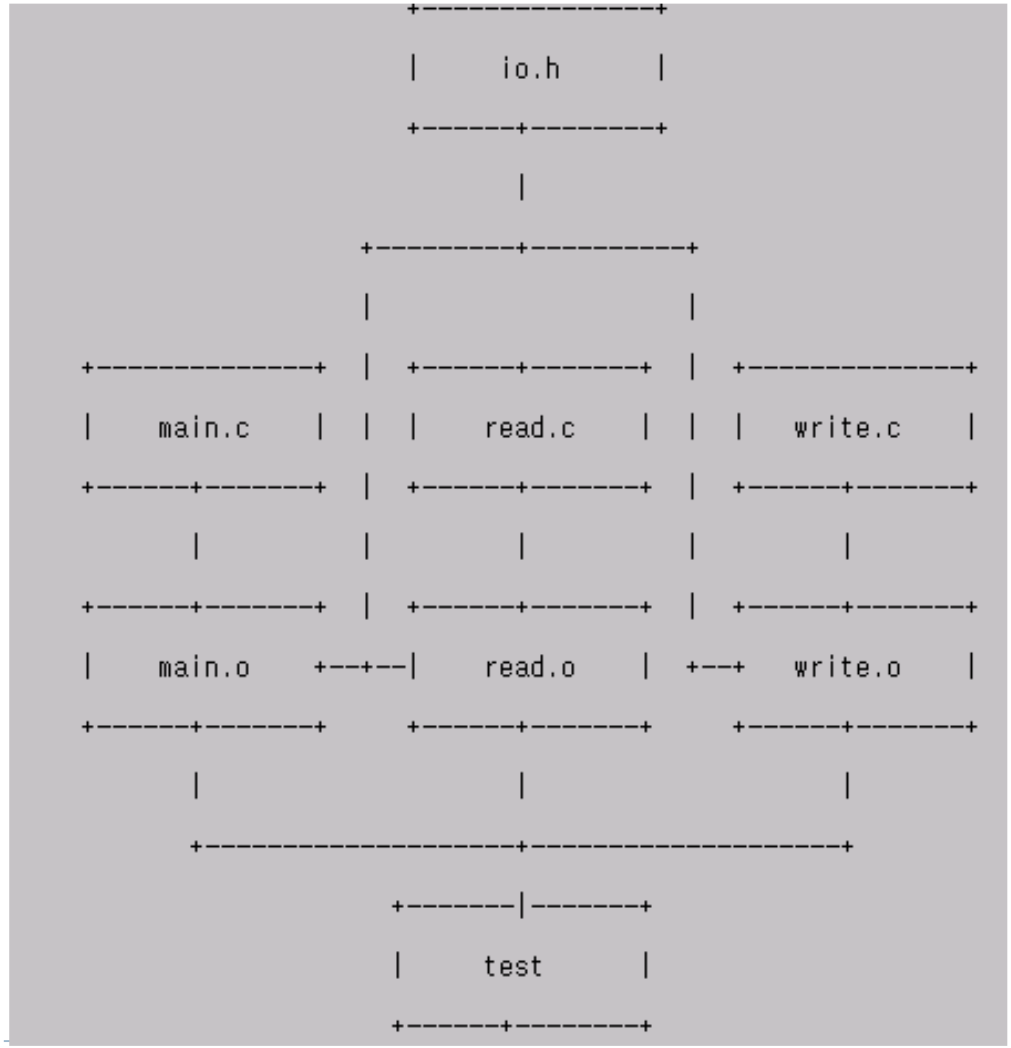


예)

- **main.c read.c write.c로 구성되어 있고 모두 io.h라는 헤더 파일을 사용**
  - % gcc -c main.c
  - % gcc -c read.c
  - % gcc -c write.c
  - % gcc -o test main.o read.o write.o

- **Makefile의 예**

```
test : main.o read.o write.o
        gcc -o test main.o read.o write.o
main.o : io.h main.c
        gcc -c main.c
read.o : io.h read.c
        gcc -c read.c
write.o: io.h write.c
        gcc -c write.c
```


io.h
main.c
read.c
write.c
main.o
read.o
write.o
test

실행

- % make

```
gcc -c main.c
gcc -c read.c
gcc -c write.c
gcc -o test main.o read.o write.o
```

3.4 이클립스 통합개발환경

이클립스(Eclipse)

- 통합 개발 환경
  - 윈도우, 리눅스, 맥 등의 다양한 플랫폼에서 사용 가능
  - 다양한 언어(C/C++, Java 등)를 지원
  - 막강한 기능을 자랑하는 자유 소프트웨어

- 이클립스 설치
  - CentOS 6 설치: [S/W Development Workstation] 선택하면 자동으로 설치됨
  - 메인메뉴: [시스템]->[관리]->[소프트웨어 추가/제거] 이용하여 이클립스를 선택하여 설치할 수 있음.
  - https://www.eclipse.org: 리눅스용 이클립스를 다운받아 설치가능