

제3장 C 프로그래밍 환경

숙명여대 창병모
2015

3.1 컴파일러

gcc 컴파일러

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



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
   라고 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 충분히 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
 - make 시스템을 이용하여 효과적으로 작업
- Makefile
 - 실행 파일을 만들기 위해 필요한 파일들과 만드는 방법을 기술
 - make 시스템은 파일의 상호 의존 관계를 파악하여 실행 파일을 쉽게 다시 만듦.
- \$ make [-f 메이크파일]
 - 옵션이 없으면 Makefile 혹은 makefile을 사용



메이크파일의 구성

- Makefile의 구성 형식

대상리스트: 의존리스트
명령리스트

- 예: Makefile

```
main:main.o copy.o
```

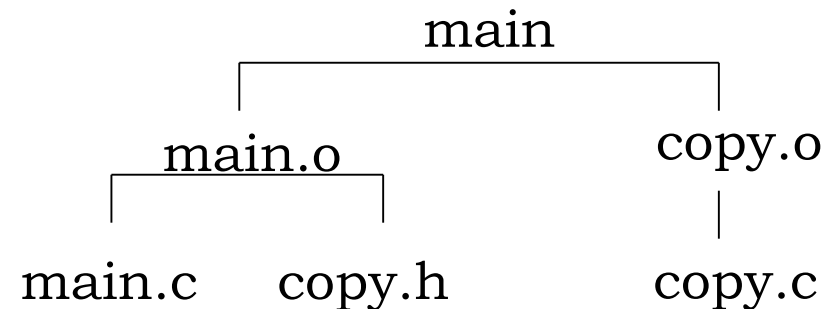
```
    gcc -o main main.o copy.o
```

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

```
    gcc -c main.c
```

```
copy.o: copy.c
```

```
    gcc -c copy.c
```



메이크파일의 구성

- make 실행

\$ make 혹은 \$ make main

gcc -c main.c

gcc -c copy.c

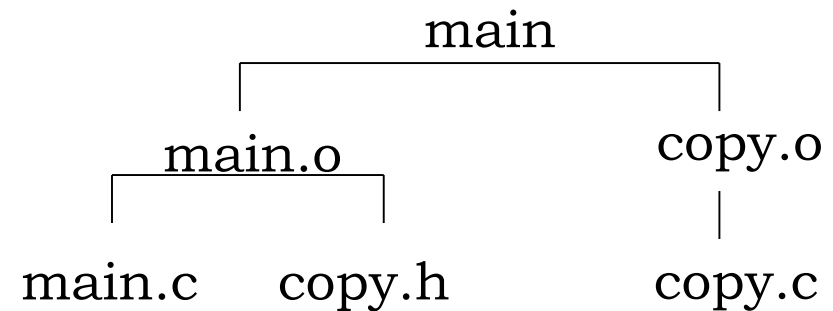
gcc -o main main.o copy.o

- copy.c 파일이 변경된 후

\$ make

gcc -c copy.c

gcc -o main main.o copy.o



3.3 디버거

gdb

- 가장 대표적인 디버거
 - GNU debugger(gdb)
- gdb 주요 기능
 - 정지점(breakpoint) 설정
 - 한 줄씩 실행
 - 변수 접근 및 수정
 - 함수 탐색
 - 추적(tracing)

- gdb 사용을 위한 컴파일
 - -g 옵션을 이용하여 컴파일
\$ gcc -g -o longest longest.c
 - 다중 모듈 프로그램
\$ gcc -g -o main main.c copy.c
- gdb 실행
\$ gdb [실행파일]



gdb 기능

- 소스보기 : l(ist)
 - l [줄번호] 지정된 줄을 프린트
 - l [파일명]:[함수명] 지정된 함수를 프린트
 - set listsize n 출력되는 줄의 수를 n으로 변경

```
(gdb) l copy
```

```
1 #include <stdio.h>
2
3 /* copy: copy 'from' into 'to'; assume to is big enough */
4 void copy(char from[], char to[])
5 {
6     int i;
7
8     i = 0;
9     while ((to[i] = from[i]) != '\0')
10         ++i;
```



gdb 기능

- 정지점 : b(reak), clear, d(elete)
 - b [파일:]함수 파일의 함수 시작부분에 정지점 설정
 - b n n번 줄에 정지점을 설정
 - b +n 현재 줄에서 n개 줄 이후에 정지점 설정
 - b -n 현재 줄에서 n개 줄 이전에 정지점 설정
 - info b 현재 설정된 정지점을 출력
 - clear 줄번호 해당 정지점을 삭제
 - d 모든 정지점을 삭제

(gdb) b copy

Breakpoint 1 at 0x804842a: file copy.c, line 9.

(gdb) info b

Num Type Disp Enb Address What

1 breakpoint keep y 0x0804842a in copy at copy.c:9



gdb 기능

- 프로그램 수행

- | | |
|--------------|--------------------------|
| ▪ r(un) 인수 | 명령줄 인수를 받아 프로그램 수행 |
| ▪ k(ill) | 프로그램 수행 강제 종료 |
| ▪ n(ext) | 멈춘 지점에서 다음 줄을 수행하고 멈춤 |
| ▪ s(tep) | n과 같은 기능 함수호출 시 함수내부로 진입 |
| ▪ c(ontinue) | 정지점을 만날 때 까지 계속 수행 |
| ▪ u | 반복문에서 빠져나옴 |
| ▪ finish | 현재 수행하는 함수의 끝으로 이동 |
| ▪ return | 현재 수행중인 함수를 빠져나옴 |
| ▪ quit | 종료 |

(gdb) r

Starting program: /home/chang/바탕화면/src/long

Merry X-mas !

Breakpoint 1, copy (from=0x8049b60 "Merry X-mas !", to=0x8049760 "")
at copy.c:9

8 i = 0;



gdb 기능

- 변수 값 프린트: p(rint)
 - p [변수명] 해당 변수 값 프린트
 - p 파일명::[변수명] 특정 파일의 전역변수 프린트
 - p [함수명]::[변수명] 특정 함수의 정적 변수 프린트
 - info locals 현재 상태의 지역변수 리스트

(gdb) p from

\$1 = 0x8049b60 "Merry X-mas !"

(gdb) n

9 while ((to[i] = from[i]) != '\0')

(gdb) n

10 ++i;

(gdb) p to

\$2 = 0x8049760 "M"



gdb 기능

(gdb) c

Continuing.

Happy New Year !

Breakpoint 1, copy

(from=0x8049b60 "Happy New Year !",

to=0x8049760 "Merry X-mas !") at
copy.c:9

9 i = 0;

(gdb) p from

\$3 = 0x8049b60 "Happy New Year !"

(gdb) n

10 while ((to[i] = from[i])!='\0')

(gdb) n

11 ++i;

(gdb) p to

\$4 = 0x8049760 "Herry X-mas !"

(gdb) c

Continuing.

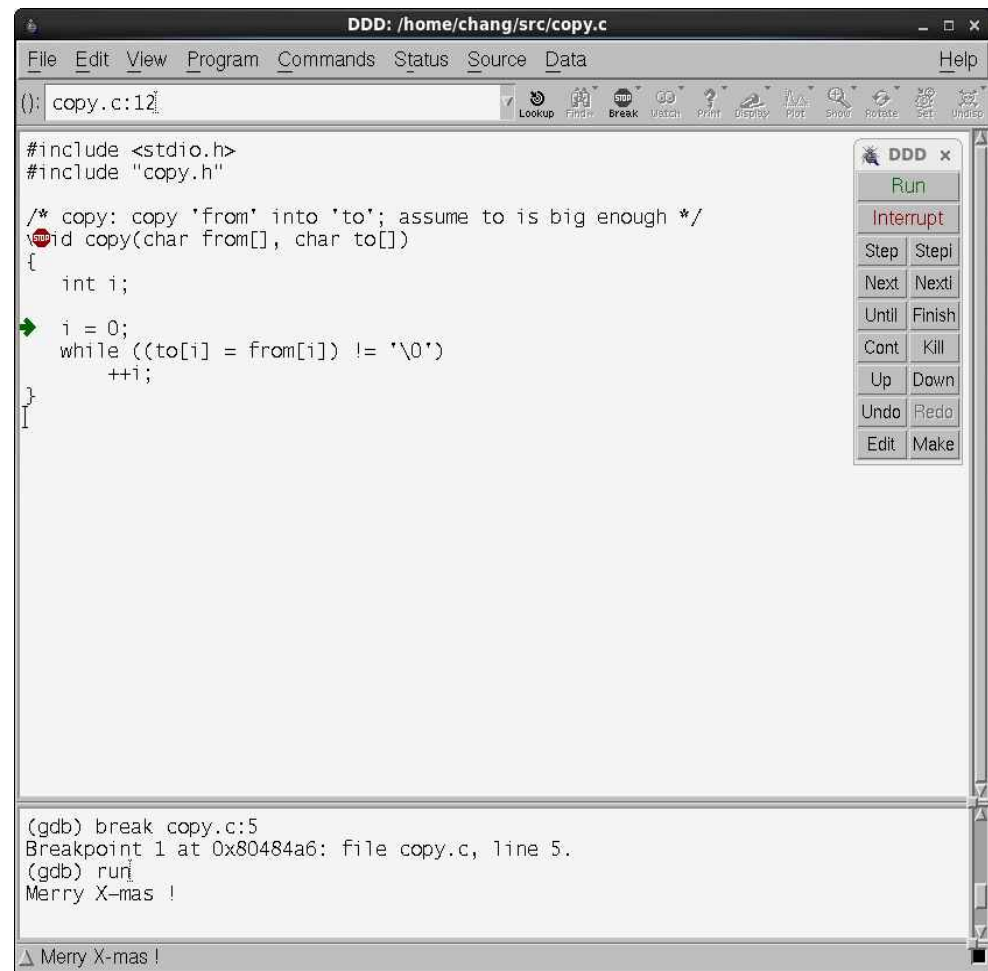
Happy New Year !

Program exited normally.

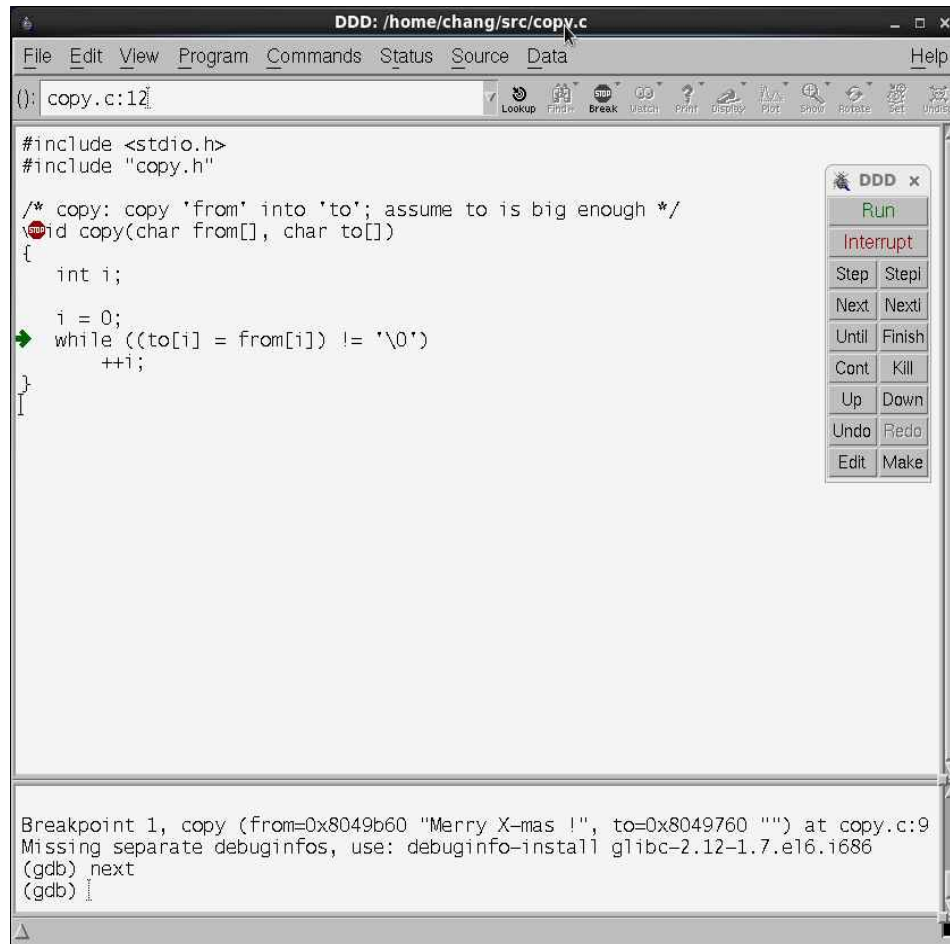


DDD(Data Display Debugger)

- gdb를 위한 그래픽 사용자 인터페이스
 - <http://www.gnu.org/software/ddd>
- 정지점을 설정
 - 소스코드의 원하는 위치에 커서를 이동하고 상 Break 버튼
 - Next나 Step 같은 명령어 버튼을 이용하여 한 줄씩 실행
 - 하단에는 gdb 명령어 입력 창



Next: 한 줄 진행



DDD: /home/chang/src/copy.c

File Edit View Program Commands Status Source Data Help

(:) copy.c:12

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

/* copy: copy 'from' into 'to'; assume to is big enough */
void copy(char from[], char to[])
{
    int i;

    i = 0;
    while ((to[i] = from[i]) != '\0')
        ++i;
}
```

DDD x

Run

Interrupt

Step Step

Next Next

Until Finish

Cont Kill

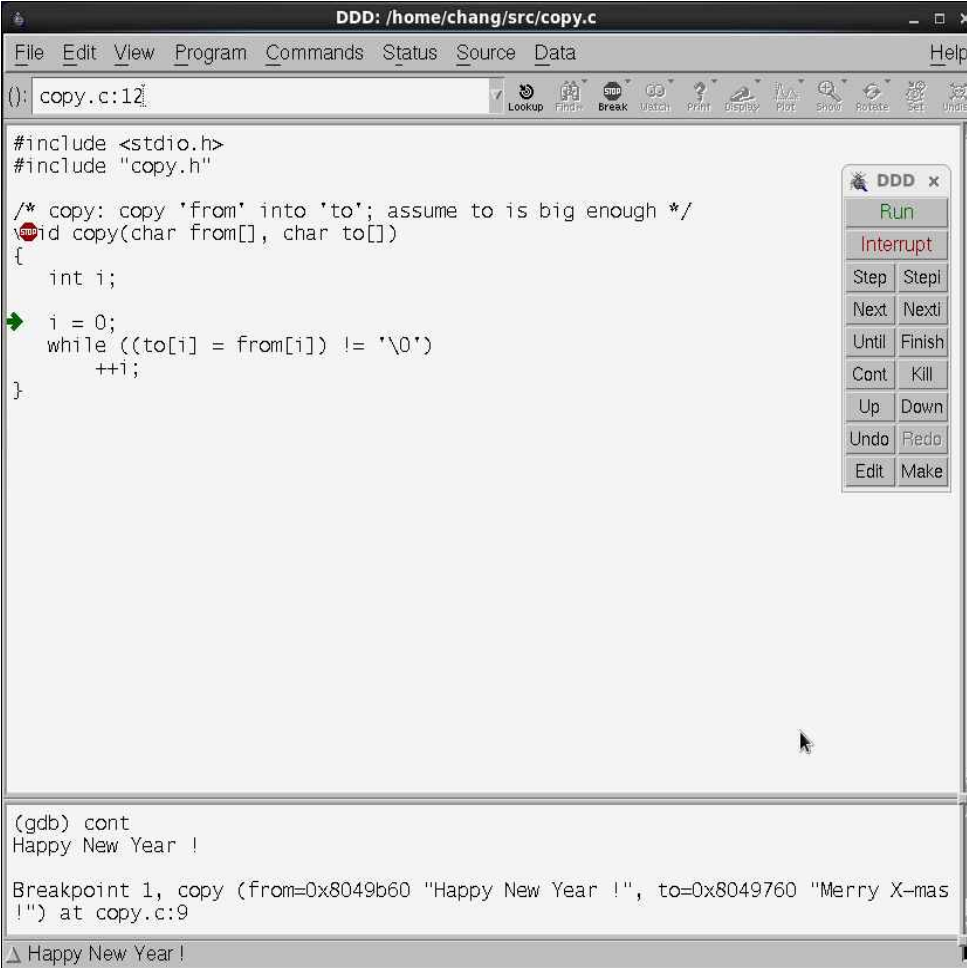
Up Down

Undo Redo

Edit Make

Breakpoint 1, copy (from=0x8049b60 "Merry X-mas!", to=0x8049760 "") at copy.c:9
Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.7.el6.i686
(gdb) next
(gdb)

Cont: 계속 진행



DDD: /home/chang/src/copy.c

File Edit View Program Commands Status Source Data Help

() copy.c:12

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

/* copy: copy 'from' into 'to'; assume to is big enough */
void copy(char from[], char to[])
{
    int i;
    i = 0;
    while ((to[i] = from[i]) != '\0')
        ++i;
}
```

DDD x

Run

Interrupt

Step StepI

Next NextI

Until Finish

Cont Kill

Up Down

Undo Redo

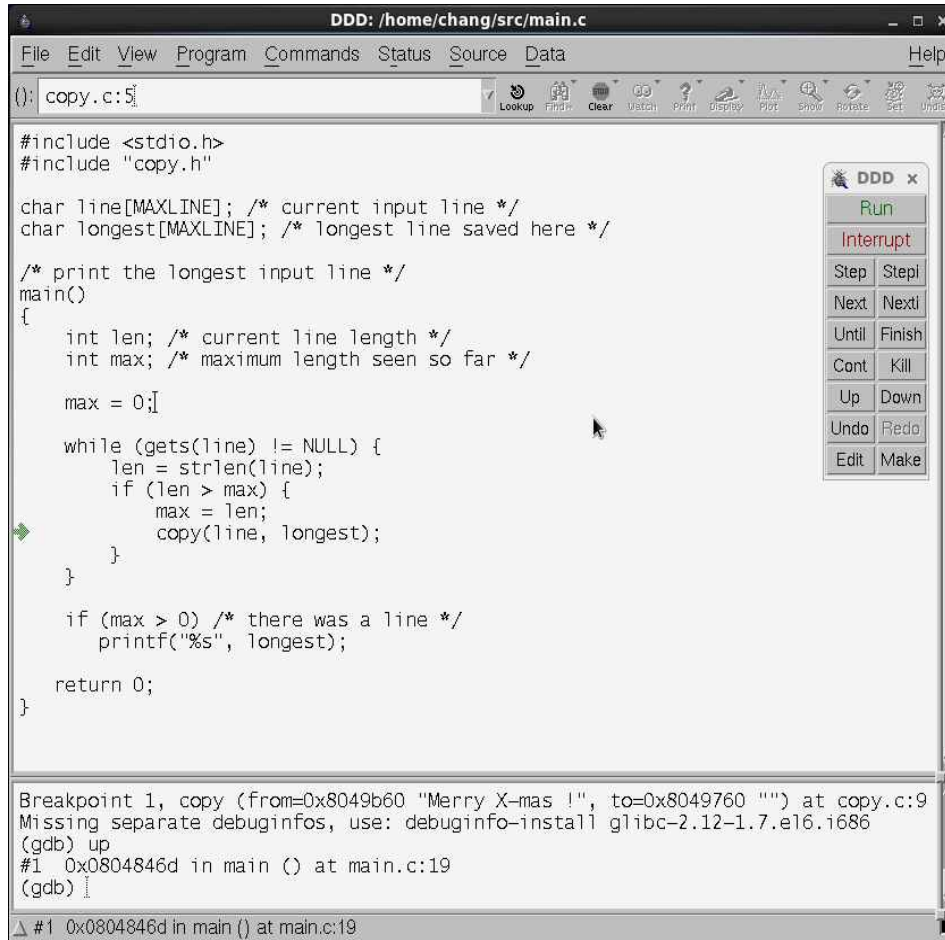
Edit Make

(gdb) cont
Happy New Year !

Breakpoint 1, copy (from=0x8049b60 "Happy New Year !", to=0x8049760 "Merry X-mas !") at copy.c:9

△ Happy New Year !

Up: 함수 호출자 보기



```
DDD: /home/chang/src/main.c
File Edit View Program Commands Status Source Data Help

(): copy.c:5

#include <stdio.h>
#include "copy.h"

char line[MAXLINE]; /* current input line */
char longest[MAXLINE]; /* longest line saved here */

/* print the longest input line */
main()
{
    int len; /* current line length */
    int max; /* maximum length seen so far */

    max = 0;

    while (gets(line) != NULL) {
        len = strlen(line);
        if (len > max) {
            max = len;
            copy(line, longest);
        }

        if (max > 0) /* there was a line */
            printf("%s", longest);

        return 0;
    }
}

Breakpoint 1, copy (from=0x8049b60 "Merry X-mas !", to=0x8049760 "") at copy.c:9
Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.7.el6.i686
(gdb) up
#1 0x0804846d in main () at main.c:19
(gdb)

△ #1 0x0804846d in main () at main.c:19
```

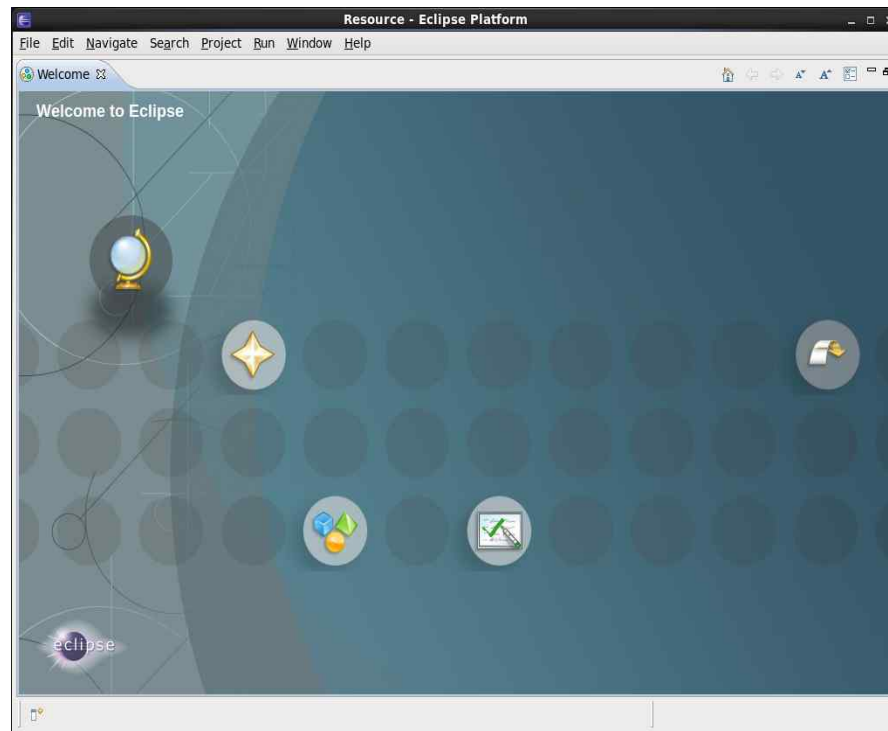
3.4 이클립스 통합개발환경

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

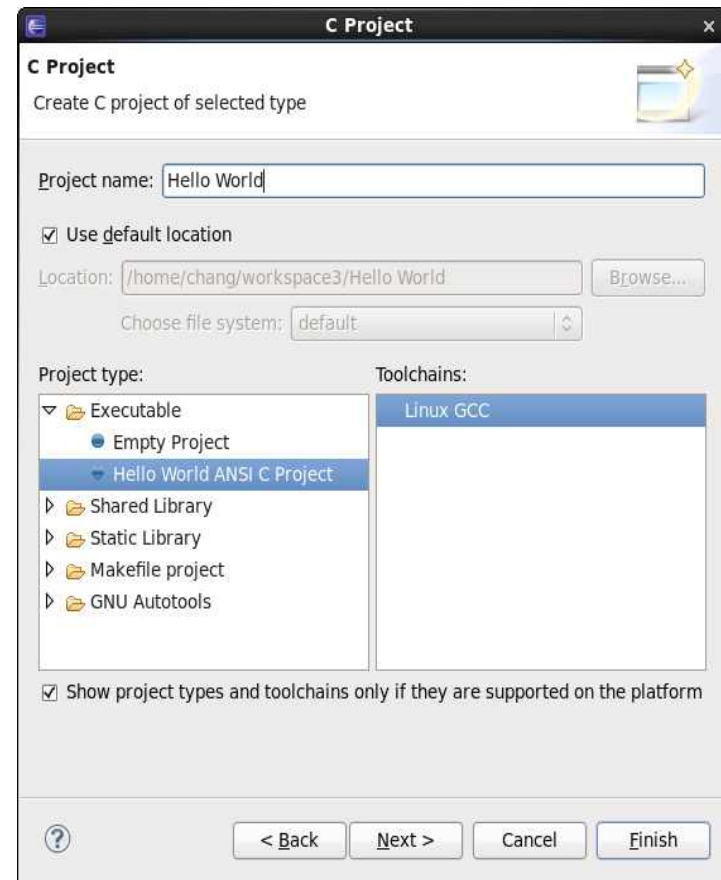


이클립스 시작화면



새로운 C 프로젝트를 생성하기

- 'File → New → C/C++ Projects'
- 프로젝트 선택 화면
 - 프로젝트 이름 지정
 - 프로젝트 타입:
 - 'Hello World ANSI C Project' 선택
 - 'Finish' 버튼 클릭



이클립스 메인화면

- 좌측 탐색 창:
 - 새로 생성된 프로젝트 확인 및 프로젝트, 파일 탐색
 - 소스 파일은 src 폴더에 헤더 파일은 include 폴더에 저장됨
- 중앙
 - 상단은 소스 및 각종 파일 등을 편집 수정할 수 있는 창
 - 하단은 C 파일을 컴파일 혹은 실행한 결과를 보여주는 창
- 화면의 우측
 - 이클립스 사용법을 보여준다.



이클립스 메인화면

