

11장 프로그래밍 환경

창병모 숙명여대

11장 프로그래밍 환경

- 01 프로그램 작성과 컴파일
- 02 자동 빌드 도구
- 03 gdb 디버거
- 04 이클립스 통합개발환경
- 05 vi 에디터

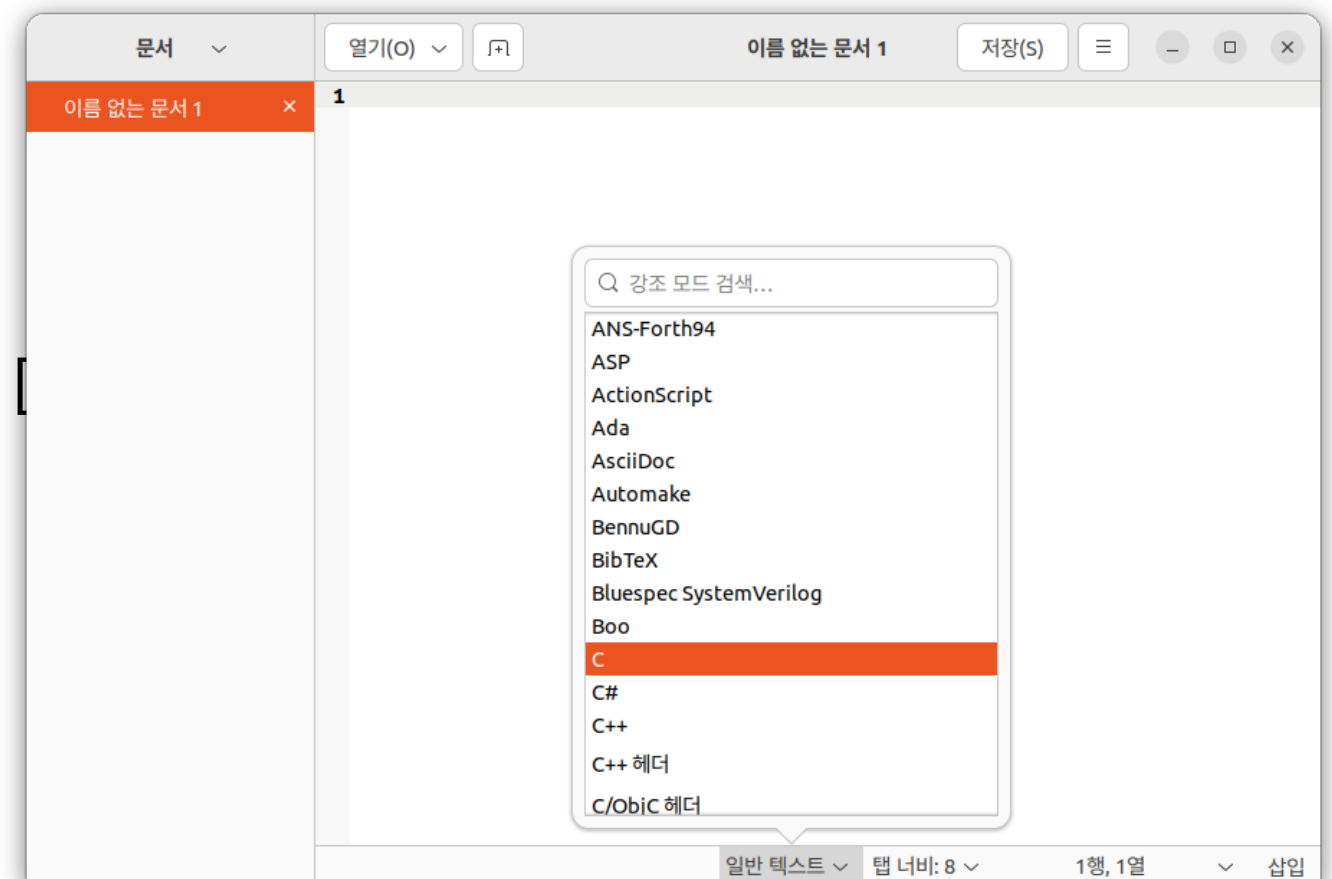


11.1 프로그램 작성과 컴파일

gedit 문서편집기

vi: CLI

- GNU의 대표적인 GUI 텍스트 편집기
- GNOME 환경의 기본 편집기
 - 텍스트, 프로그램 코드, 마크업 언어 편집에 적합
 - 깔끔하고 단순한 GUI
- gedit 실행 방법
 - 메인 메뉴
 - [프로그램] -> [보조 프로그램] -> [에디트] 선택
 - 터미널
 - \$ gedit [파일이름] &
 - 파일 관리자:
 - 텍스트 파일 클릭하면 자동실행



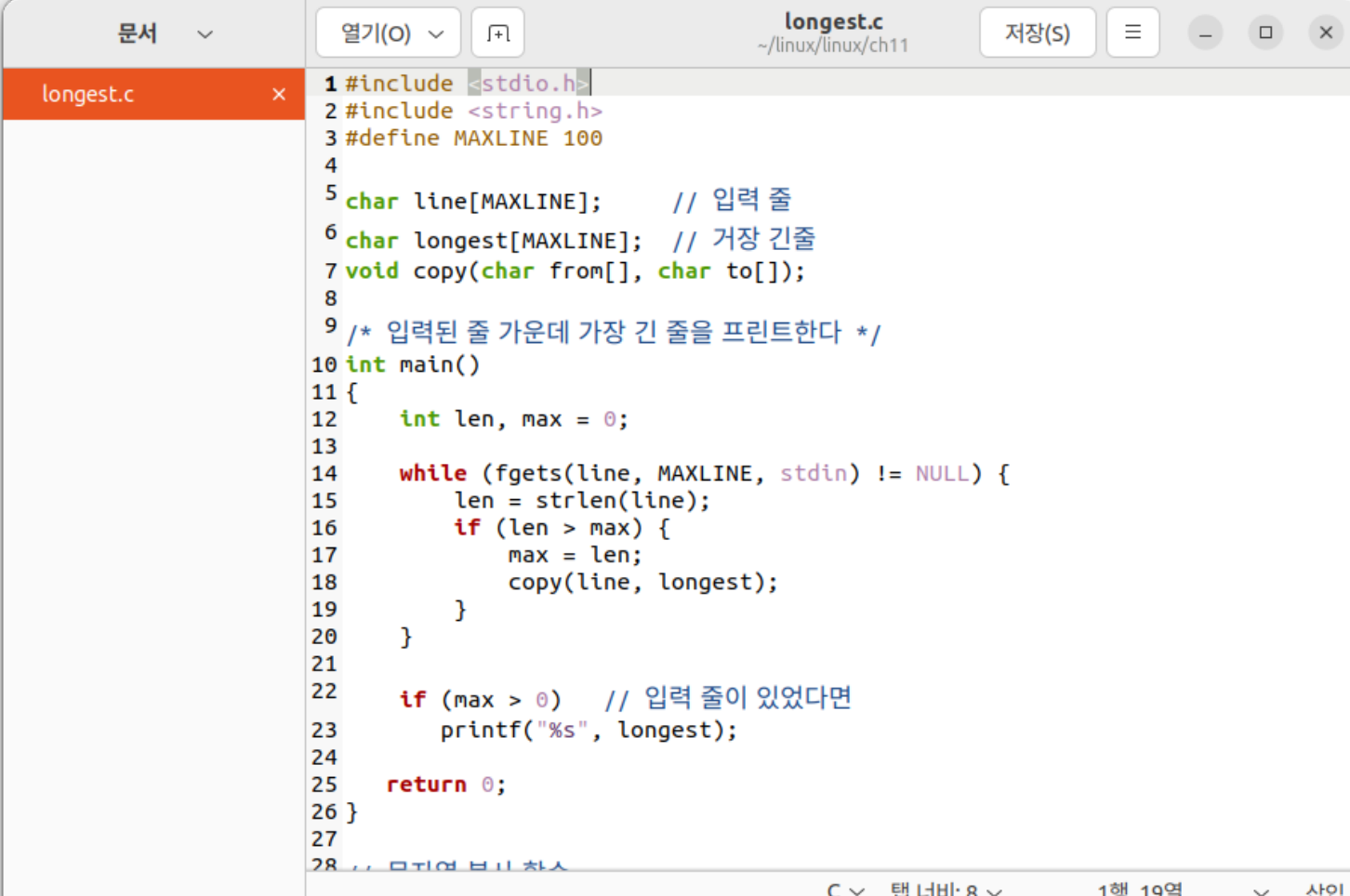
gedit 메뉴

- 파일
 - 새로 만들기, 열기, 저장, 되돌리기, 인쇄
- 편집
 - 입력 취소, 다시 실행, 잘라내기, 복사, 붙여넣기, 삭제
- 보기
 - 도구모음, 상태표시줄, 전체화면, **강조 모드**
- 검색
 - 찾기, 바꾸기, 줄로 이동
- 도구
 - 맞춤법 검사, 오타가 있는 단어 강조, 언어 설정, 문서 통계
- 문서
 - 모두 저장, 모두 닫기, 새 탭 그룹, 이전 문서



단일 모듈 프로그램

- 프로그램 작성
 - gedit 이용
- [보기] 메뉴
 - C 구문 강조 기능 설정
- 프로그램 편집하는 화면
 - #include 같은 전처리 지시자는 황색
 - 주석은 파란색
 - 자료형 이름은 초록색
 - if나 while 같은 문장 키워드는 브라운 색



```
1 #include <stdio.h>
2 #include <string.h>
3 #define MAXLINE 100
4
5 char line[MAXLINE];    // 입력 줄
6 char longest[MAXLINE]; // 저장 긴줄
7 void copy(char from[], char to[]);
8
9 /* 입력된 줄 가운데 가장 긴 줄을 프린트한다 */
10 int main()
11 {
12     int len, max = 0;
13
14     while (fgets(line, MAXLINE, stdin) != NULL) {
15         len = strlen(line);
16         if (len > max) {
17             max = len;
18             copy(line, longest);
19         }
20     }
21
22     if (max > 0) // 입력 줄이 있었다면
23         printf("%s", longest);
24
25     return 0;
26 }
27
28 // 문지영 박사 한스
```

gcc 컴파일러

- gcc(GNU cc) 컴파일러

`$ gcc [-옵션] 파일`

C 프로그램을 컴파일한다. 옵션을 사용하지 않으면 실행파일 a.out를 생성한다.

- 간단한 컴파일 및 실행

`$ gcc longest.c`

`$ a.out`

// 실행

- -c 옵션: 목적 파일 생성 (각각 컴파일)

`$ gcc -c longest.c`

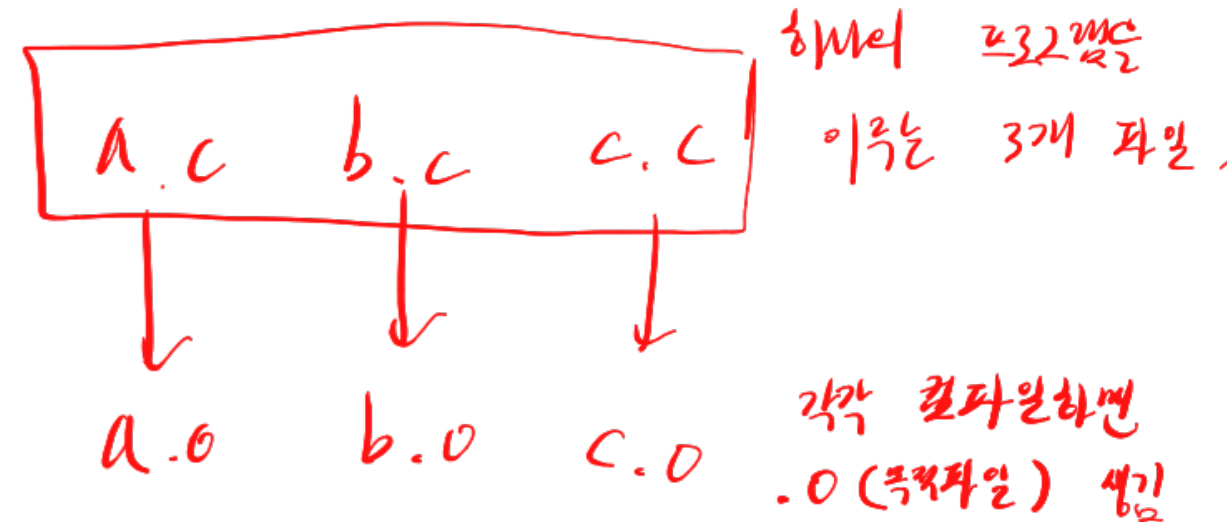
- -o 옵션: 실행 파일 생성

`$ gcc -o longest longest.o` 혹은 `$ gcc -o longest longest.c`

- 실행

`$ longest`

// 실행



단일 모듈 프로그램:longest.c

```
#include <stdio.h>
#include <string.h>
#define MAXLINE 100

char line[MAXLINE]; // 입력 줄
char longest[MAXLINE]; // 가장 긴 줄
void copy(char from[], char to[]);

/*입력 줄 가운데 가장 긴 줄 프린트 */
int main()
{
    int len, max = 0;

    while(fgets(line,MAXLINE,stdin) != 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 크다고 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

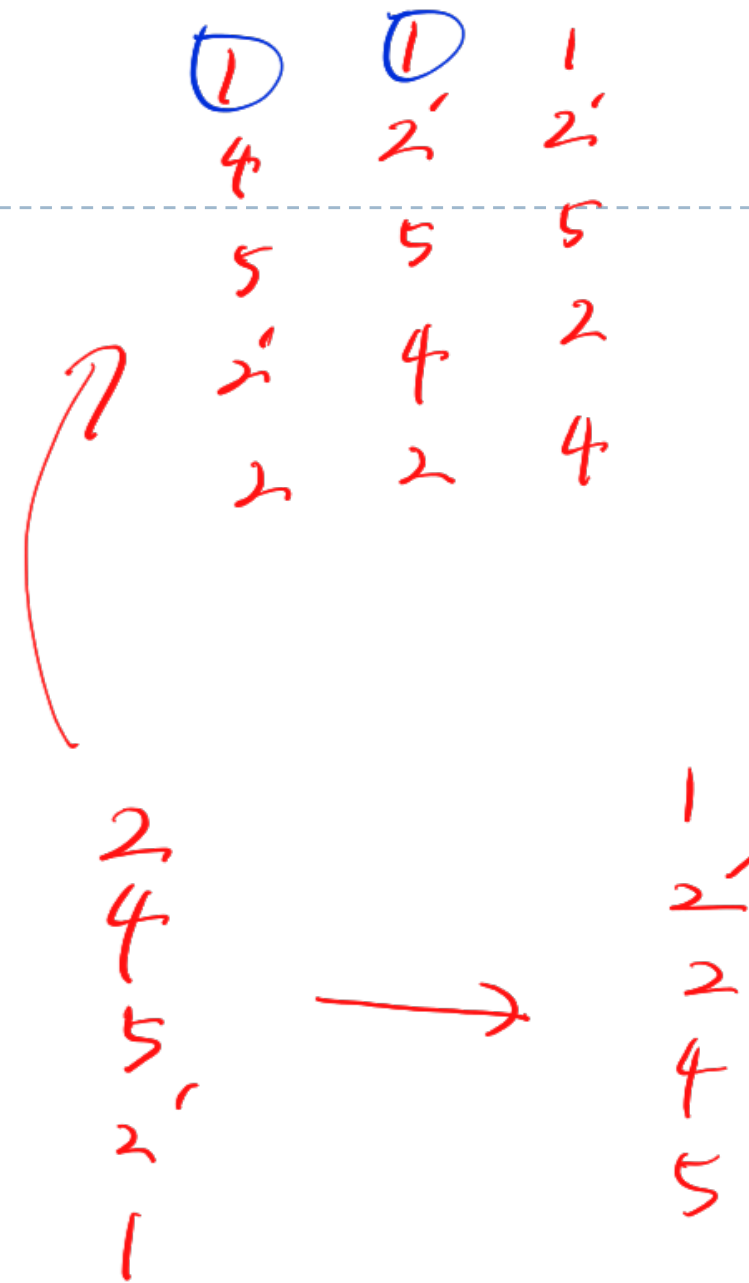
```
#include <stdio.h>
#include <string.h>
#include "copy.h"
char line[MAXLINE]; // 입력 줄
char longest[MAXLINE]; // 가장 긴 줄

/*입력 줄 가운데 가장 긴 줄 프린트 */
int main()
{
    int len, max = 0;

    while (fgets(line, MAXLINE, stdin) != 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>
```

```
/* 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[]);
```

0 1 2 (count = 2).

i : 0 ~ 1

j : 1 ~ 2



11.2 자동 빌드 도구

make 시스템의 필요성

- 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
- make 시스템
 - 대규모 프로그램의 경우에는 헤더, 소스 파일, 목적 파일, 실행 파일의 모든 관계를 기억하고 체계적으로 관리하는 것이 필요
 - make 시스템을 이용하여 효과적으로 작업



메이크파일

- 메이크파일

- 실행 파일을 만들기 위해 필요한 파일들
- 그들 사이의 의존 관계
- 만드는 방법을 기술

- make 시스템

- 메이크파일을 이용하여 파일의 상호 의존 관계를 파악하여 실행 파일을 쉽게 다시 만듦

- 사용법

```
$ make [-f 메이크파일]
```

make 시스템은 메이크파일(makefile 혹은 Makefile)을 이용하여 보통 실행 파일을 빌드한다. 옵션을 사용하여 별도의 메이크파일을 지정할 수 있다.



메이크파일의 구성

- 메이크파일의 구성 형식

목표(target): 의존리스트(dependencies)

명령리스트(commands)

- 예: Makefile

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

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

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

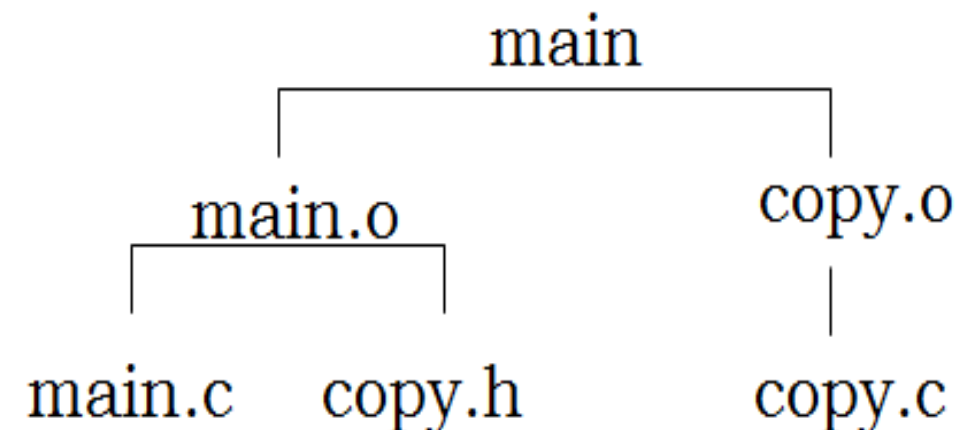
```
    gcc -c main.c
```

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

```
    copy.c
```

```
gcc -c
```

- 의존 관계 그래프



메이크파일의 구성

- make 실행

\$ make 혹은 \$ make main

gcc -c main.c

gcc -c copy.c

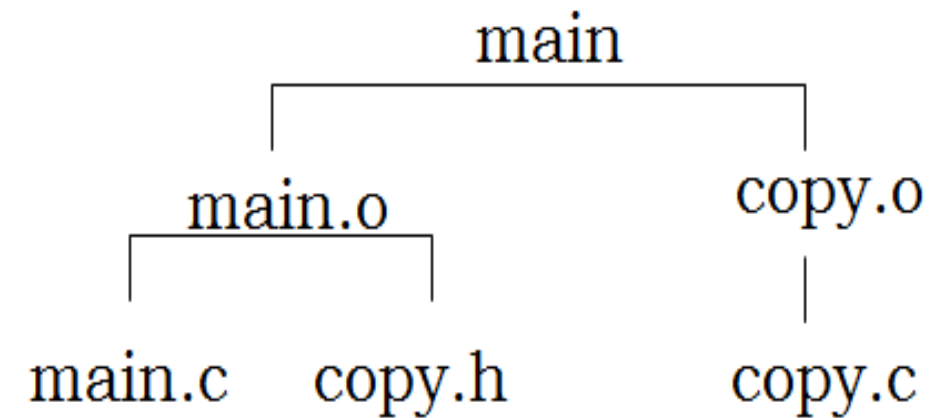
gcc -o main main.o copy.o

- copy.c 파일이 변경된 후

\$ make

gcc -c copy.c

gcc -o main main.o copy.o



11.3 gdb 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 [실행파일]

gdb 디버거는 실행파일을 이용하여 디버깅 모드로 실행한다.



gdb

- gdb 사용을 위한 컴파일

- -g 옵션을 이용하여 컴파일

```
$ gcc -g -o longest longest.c
```

symbolic information *없이 (함수명, 변수명)*

- 다중 모듈 프로그램

위지 안하면 다 쿼리로 바뀐다.

```
$ gcc -g -o main main.c copy.c
```

- gdb 실행

```
$ gdb [실행파일]
```

gdb 디버거는 실행파일을 이용하여 디버깅 모드로 실행한다.



gdb 기능

- 소스보기 : l(ist)

- l [줄번호]
- l [파일명]:[함수명]
- set listsize n

지정된 줄을 프린트

지정된 함수를 프린트

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 *현재 설정된 정지점 출력.*

Num Type Disp Enb Address What

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



gdb 기능

● 프로그램 수행

- r(un) 인수
- k(ill)
- ◉ n(ext)
- ◉ s(tep)
- c(ontinue)
- u
- finish
- return
- quit

명령줄 인수를 받아 프로그램 수행

프로그램 수행 강제 종료

멈춘 지점에서 다음 줄을 수행하고 멈춤

n과 같은 기능 함수호출 시 함수내부로 진입

정지점을 만날 때 까지 계속 수행

반복문에서 빠져나옴

현재 수행하는 함수의 끝으로 이동

현재 수행중인 함수를 빠져나옴

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:8

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

8 i = 0;

(gdb) p from

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

(gdb) n

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

(gdb) n

10 ++i;

(gdb) p to

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

(gdb) c

Continuing.

Happy New Year !

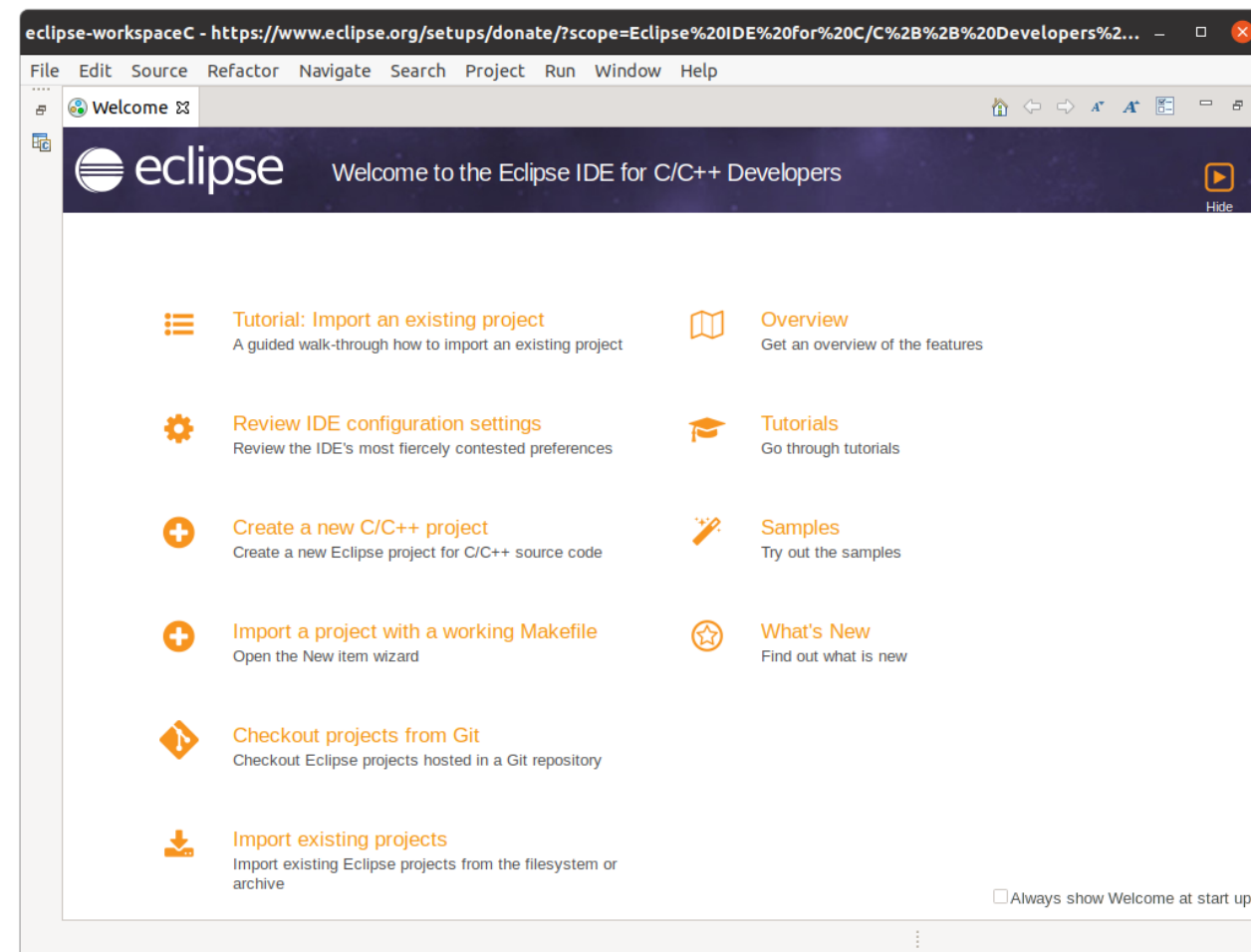
Program exited normally.



11.4 이클립스 통합개발환경

이클립스 통합개발환경

- 다양한 언어(C/C++, Java 등)를 지원하는 통합개발환경
- 설치
 - 이클립스 홈페이지에서 C/C++ 개발자를 위한 리눅스용 이클립스(Eclipse IDE for C/C++ Developers)를 다운받아 설치
 - 사전에 make 시스템과 g++ 컴파일러 등이 설치되어야 함

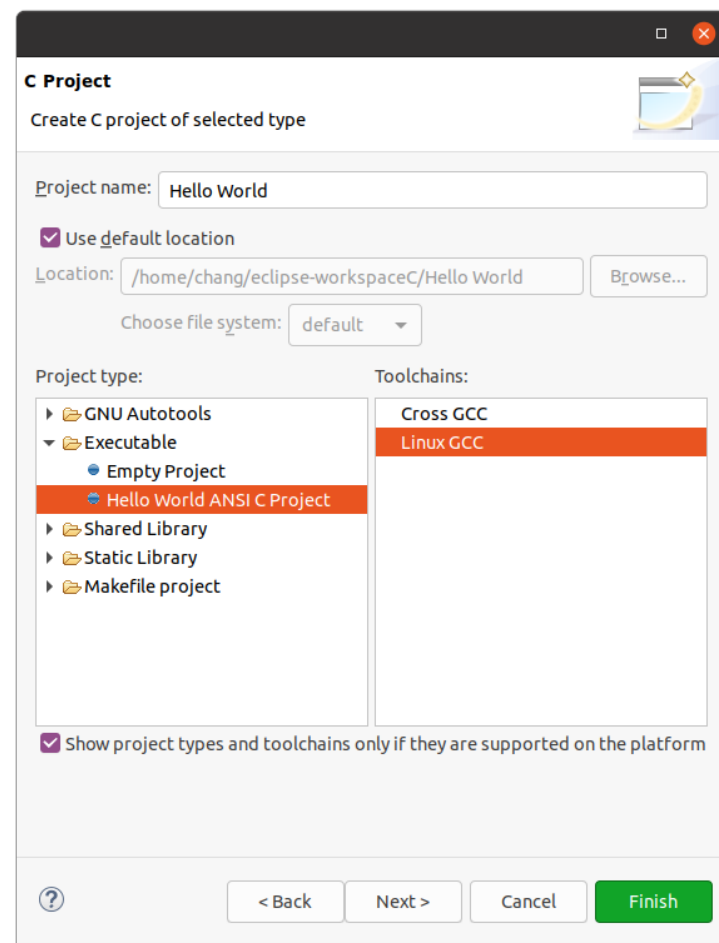
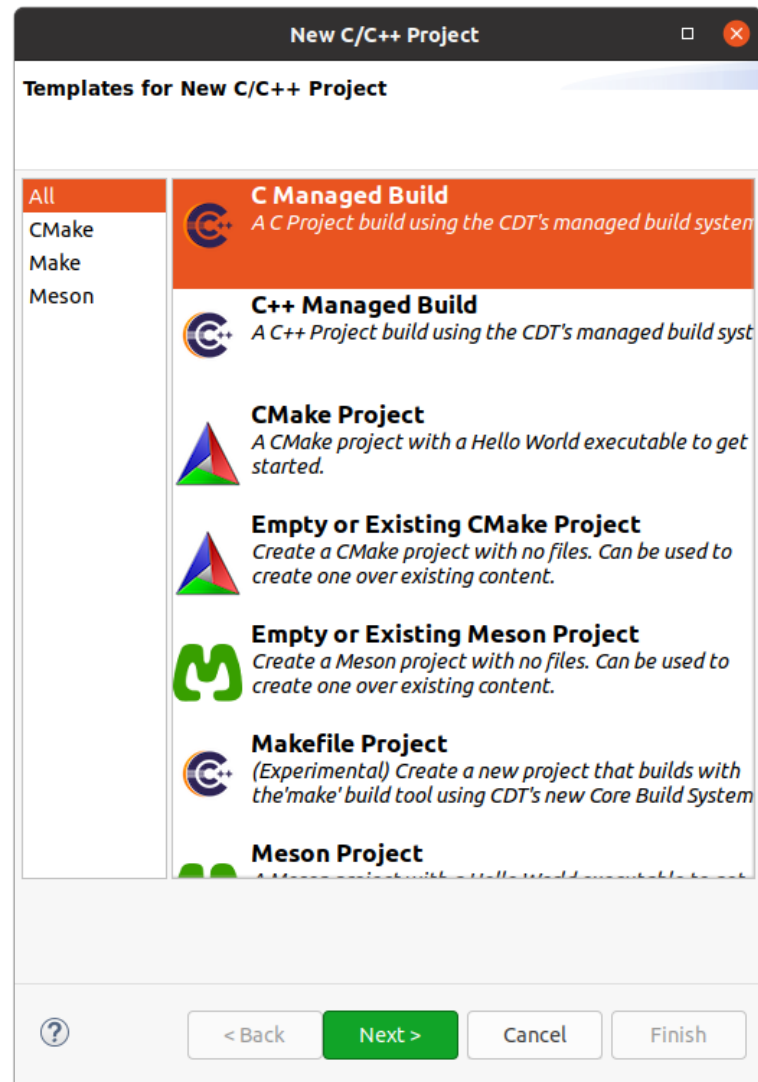


새로운 C 프로젝트 생성

- [Create a new C/C++ project] 혹은 [File]→[New]→[C/C++ Projects]
- 프로젝트 선택 화면에서 [C Managed Build] 선택
- 프로젝트 타입 [Hello World ANSI C Project] 선택
- 컴파일러 선택 후 [Finish] 버튼 클릭
- 'HelloWorld.c' 프로그램 자동으로 생성
- 프로젝트 타입 [Empty Project]를 선택하면 빈 프로젝트가 생성



새로운 C 프로젝트 생성

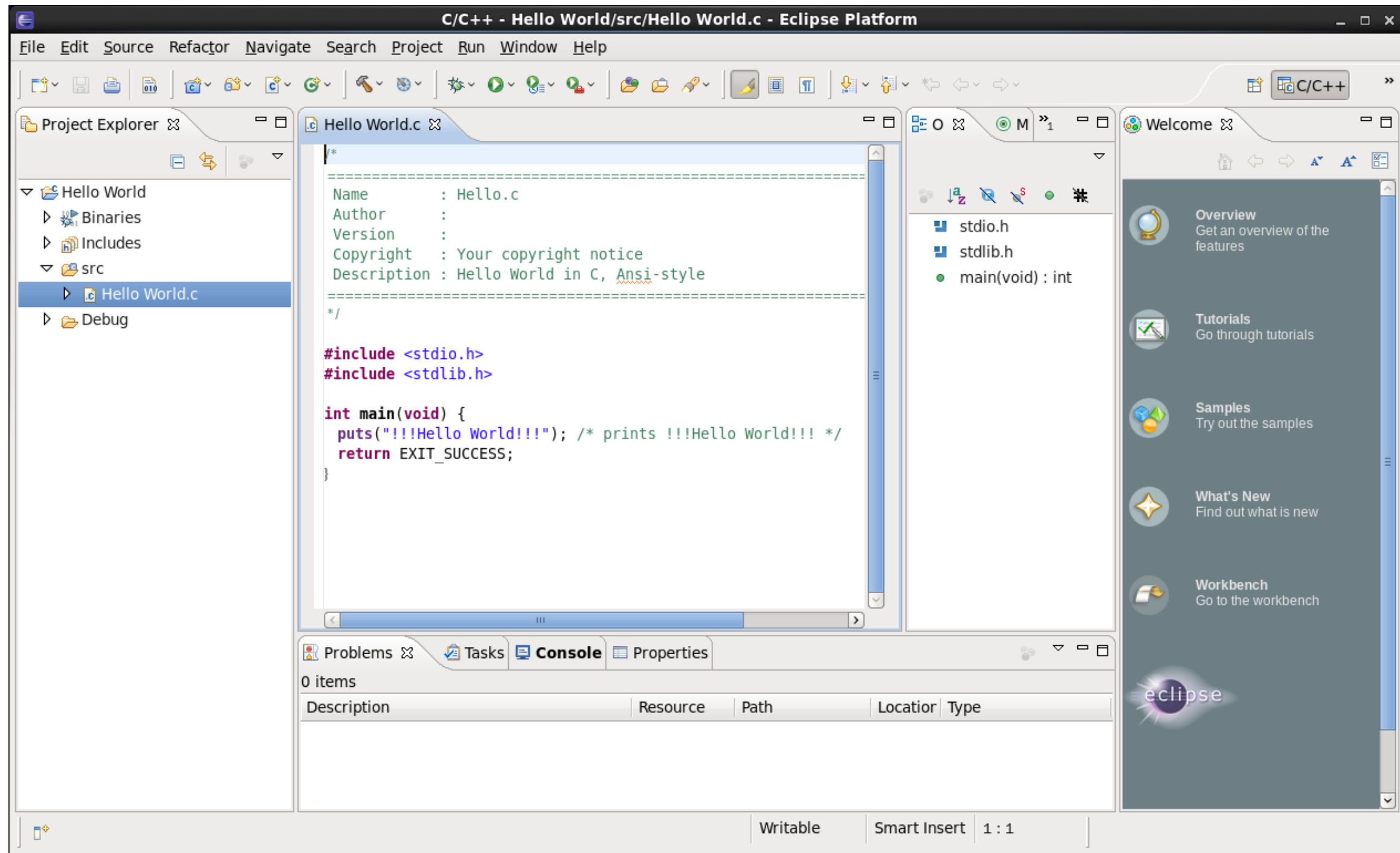


메인 화면

- 좌측 [Project Explorer] 탐색 창
 - 새로 생성된 프로젝트 확인
 - 프로젝트 및 파일들을 탐색 가능
 - 소스 파일은 src 폴더에, 헤더 파일은 include 폴더에 저장됨
- 중앙 창
 - 소스 파일 등을 편집할 수 있는 창
- 우측 창의 [Outline] 탭
 - 이 프로그램에서 사용하는 소스 파일들을 리스트
 - 해당 파일을 선택하여 파일 내용을 확인하거나 편집 가능
- 하단
 - C 파일을 컴파일 혹은 실행한 결과를 보여주는 창들



메인 화면



11.5 vi 에디터

CLI 환경 에디터

‘93로 시작하는 내용 ..

vi 에디터

- vi 에디터

- 기본 텍스트 에디터로 매우 강력한 기능을 가지고 있으나
- 배우는데 상당한 시간과 노력이 필요하다.

\$ vi 파일*

```
chang@ubuntu: ~/linux
```

빔 - 향상된 vi

판 8.1.2269
by Bram Moolenaar et al.
Modified by team+vim@tracker.debian.org
빔은 누구나 소스를 볼 수 있고 공짜로 배포됩니다

빔 사용자로 등록하세요!

이에 대한 정보를 보려면	:help register<엔터>	입력
끝내려면	:q<엔터>	입력
온라인 도움말을 보려면	:help<엔터> 또는 <F1>	입력
판 정보를 보려면	:help version8<엔터>	입력

0.0-1 모두

```

#include <stdio.h>
#include <string.h>
#define MAXLINE 100

char line[MAXLINE];    // 입력 줄
char longest[MAXLINE]; // 가장 긴 줄
void copy(char from[], char to[]);

/* 입력된 줄 가운데 가장 긴 줄을 프린트한다. */
int main()
{
    int len, max = 0;

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

    if (max > 0) // 입력 줄이 있었다면
        printf("longest line : %s", longest);
}

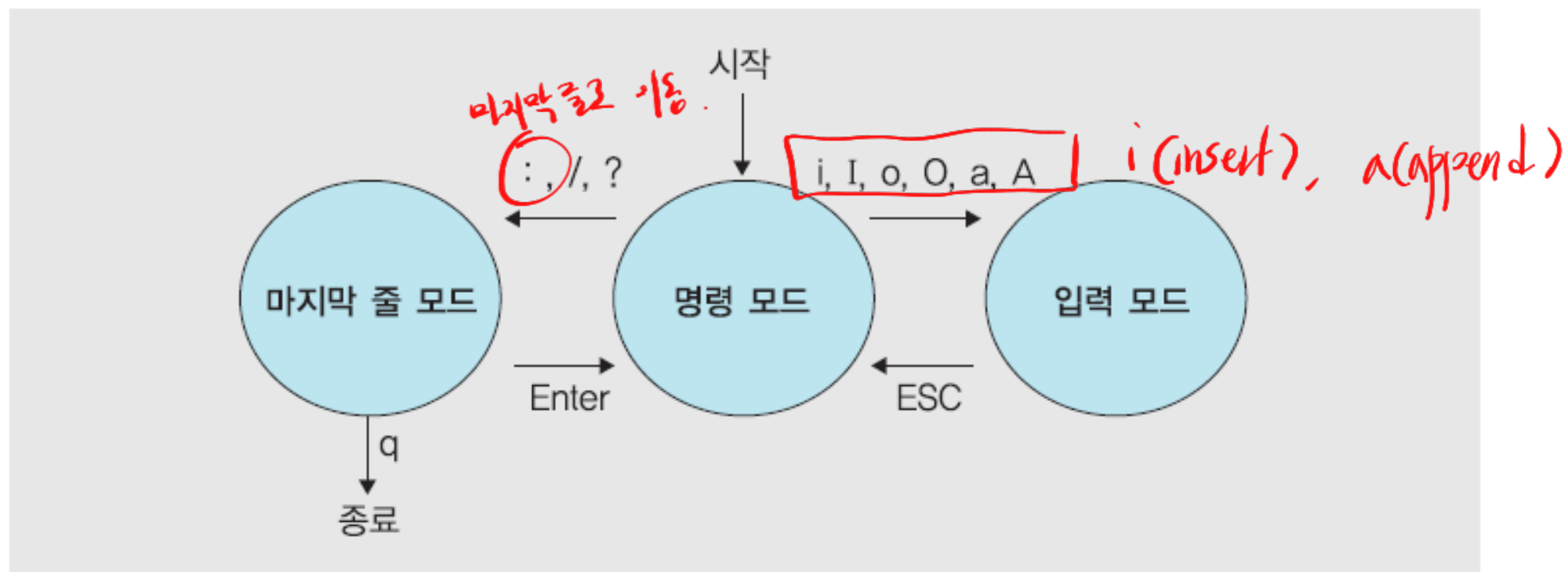
```

1,1 꼭대기

명령 모드/입력 모드

- vi 에디터는 명령 모드와 입력 모드가 구분되어 있으며
- 시작하면 명령 모드이다.

jedit : 시작하면 바로 입력모드.



- 마지막 줄 모드

- :wq 작업 내용을 저장하고 종료 (ZZ와 동일한 기능) *가장 많이 쓴다.*
- :q 아무런 작업을 하지 않은 경우의 종료
- :q! 작업을 저장하지 않고 종료

vi 내부 명령어

- 원하는 위치로 이동하는 명령
- 입력모드로 전환하는 명령
- 수정 혹은 삭제 명령
- 복사 및 붙이기
- 기타 명령

원하는 위치로 이동하는 명령

- 커서 이동

h, ←	한 칸 왼쪽
j, ↓	한 칸 아래쪽
k, ↑	한 칸 위쪽
l, →	한 칸 오른쪽
BACKSPACE	왼쪽으로 한 칸
SPACE	오른쪽으로 한 칸
-	이전 줄의 처음
+	다음 줄의 처음
RETURN	다음 줄의 처음
0	현재 줄의 맨 앞
\$	현재 줄의 끝
^	현재 줄의 첫 글자
W	다음단어의 첫 글자
B	이전단어의 첫 글자

- 화면 이동

^F	한 화면 아래로
^B	한 화면 위로
^D	반 화면 아래로
^U	반 화면 위로

- 특정 줄로 이동

nG	n번째 줄로 이동
1G	첫 줄로 이동하기
G	마지막 줄로 이동하기
:n	n번째 줄로 이동

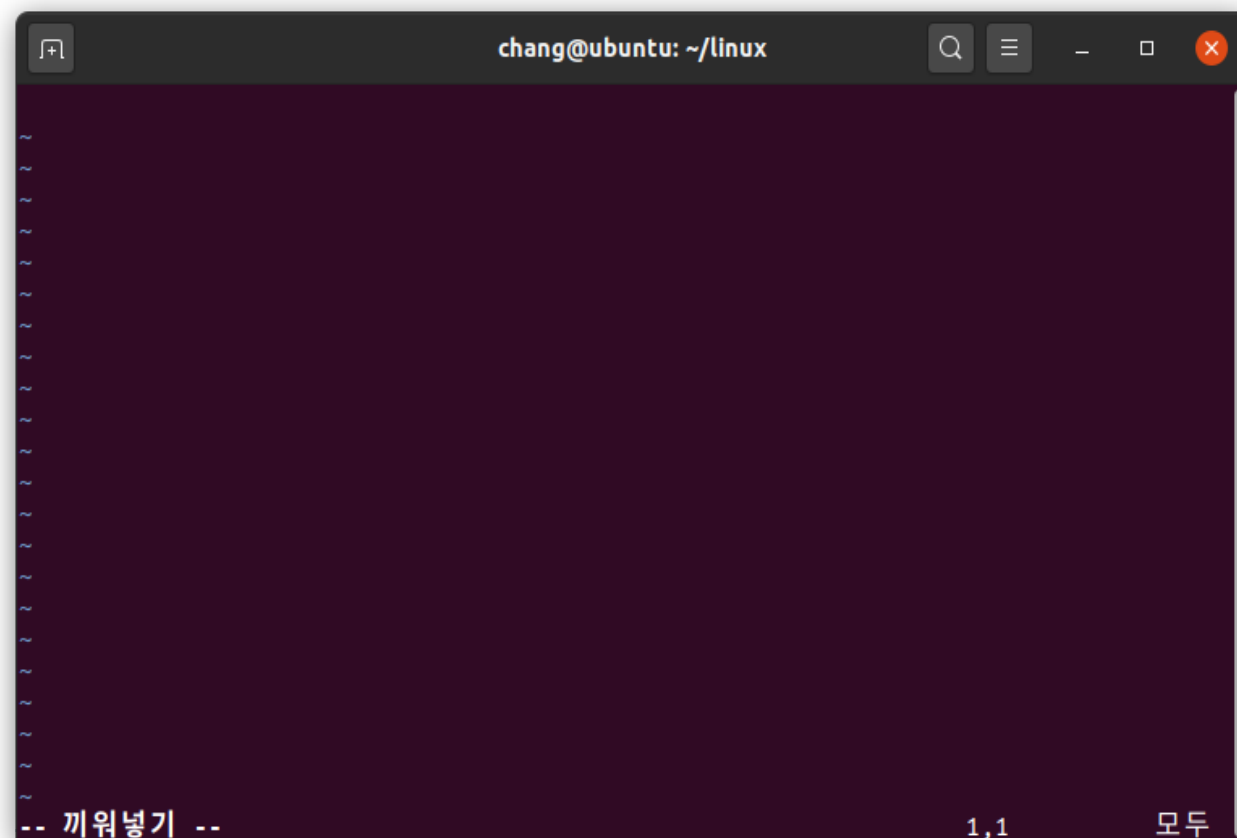
- 탐색(search)

/탐색패턴	forward 탐색
?탐색패턴	backward 탐색

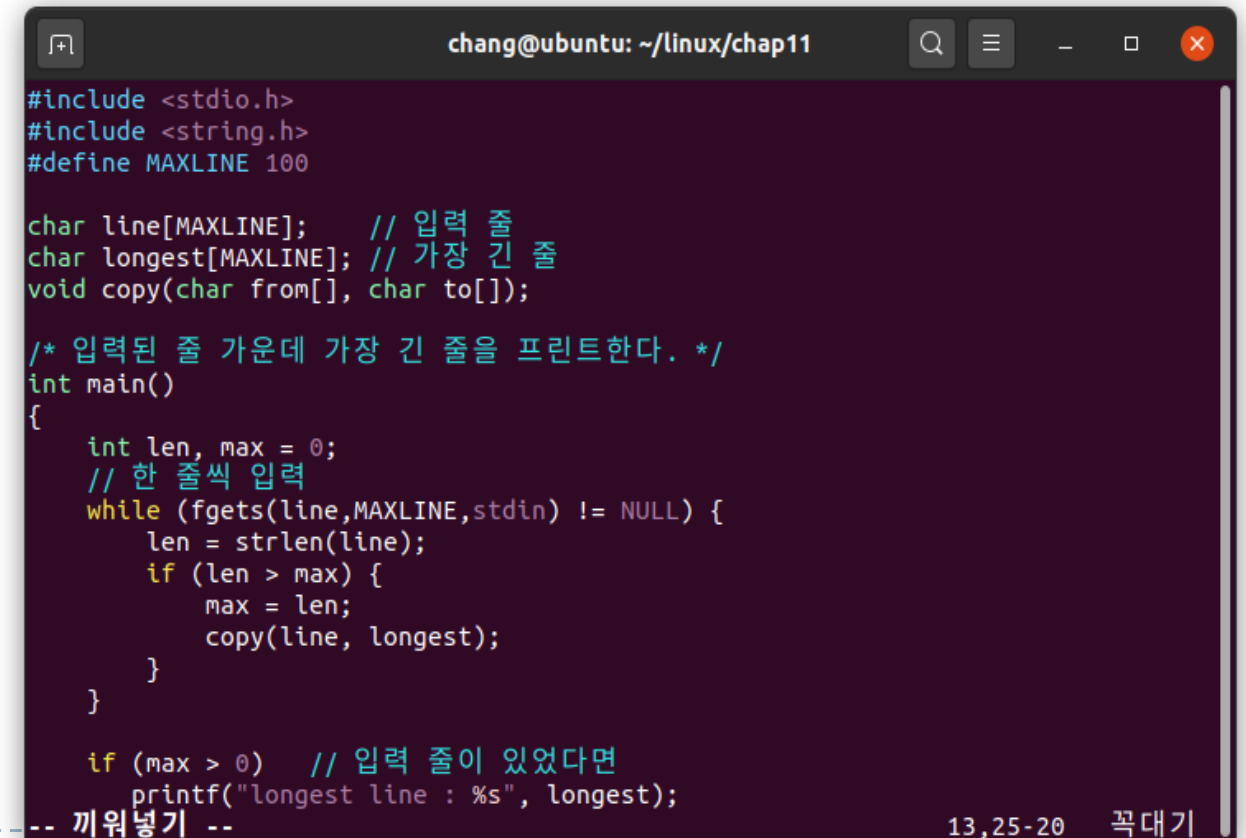
입력모드로 전환하는 명령

- 입력모드로 전환

i	커서 위치 앞에 삽입
a	커서 위치 뒤에 삽입
I	현재 줄의 앞에 삽입
A	현재 줄의 뒤에 삽입
o	현재 줄의 아래에 전개
O	현재 줄의 위에 전개



```
chang@ubuntu: ~/linux
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- 끼워넣기 --
1,1 모두
```



```
chang@ubuntu: ~/linux/chap11
#include <stdio.h>
#include <string.h>
#define MAXLINE 100

char line[MAXLINE]; // 입력 줄
char longest[MAXLINE]; // 가장 긴 줄
void copy(char from[], char to[]);

/* 입력된 줄 가운데 가장 긴 줄을 프린트한다. */
int main()
{
    int len, max = 0;
    // 한 줄씩 입력
    while (fgets(line, MAXLINE, stdin) != NULL) {
        len = strlen(line);
        if (len > max) {
            max = len;
            copy(line, longest);
        }
    }

    if (max > 0) // 입력 줄이 있었다면
        printf("longest line : %s", longest);
}

-- 끼워넣기 --
13,25-20 꼭대기
```

수정 혹은 삭제 명령

- 현재 커서를 중심으로 수정
 - r 단지 한 글자만 변경
 - R 입력하는 대로 겹쳐 쓰기
 - s 현재 글자 삭제 삽입 상태
 - C 커서로부터 줄 끝까지 변경
 - cc 현재 줄 전체를 변경
 - cw 현재 단어를 삭제하고 변경
- 삭제
 - x 커서가 있는 문자 지우기
 - X 커서의 왼쪽 문자 지우기
 - D 커서부터 줄 끝까지 지우기
 - dw 현재 단어 삭제
 - dd 현재 줄 삭제
 - :n,m d n~m번째 줄 삭제

대체, 수행취소/재수행

- 대체 명령

- 각 줄의 해당되는 첫 번째 단어만 대체
- :s/패턴/스트링
현재 줄에서 대체
- :n,m s/패턴/스트링
지정된 줄 범위에서 대체
- :n s/패턴/스트링
지정된 줄(n)에서 대체
- s/패턴/스트링/g
해당되는 모든 단어 대체

- 수행취소/재수행

- u 방금 전 수행 내용 취소(Undo)
- U 현재 줄 수행 내용을 취소
- . 방금 전 수행 내용을 반복(Redo)

복사/붙이기

- 줄 내용 복사(copy)

n 번째 줄에서부터 n 개의 줄을 복사

: n, m y n 번째 줄에서 m 번째 줄까지를 버퍼에 복사함

- 마지막으로 삭제/복사한 내용을 붙이기(put).

p 버퍼 내용을 커서의 뒤(혹은 아래)에 삽입

P 버퍼 내용을 커서의 앞(혹은 위)에 삽입

파일에 저장 및 끝내기

- **파일에 저장**

:w

현재 파일에 저장한다.

:w 파일이름

지정된 파일에 저장한다.

- **파일에 저장하고 끝내기**

:wq

현재 파일에 저장하고 종료한다.

ZZ

현재 파일에 저장하고 종료한다.

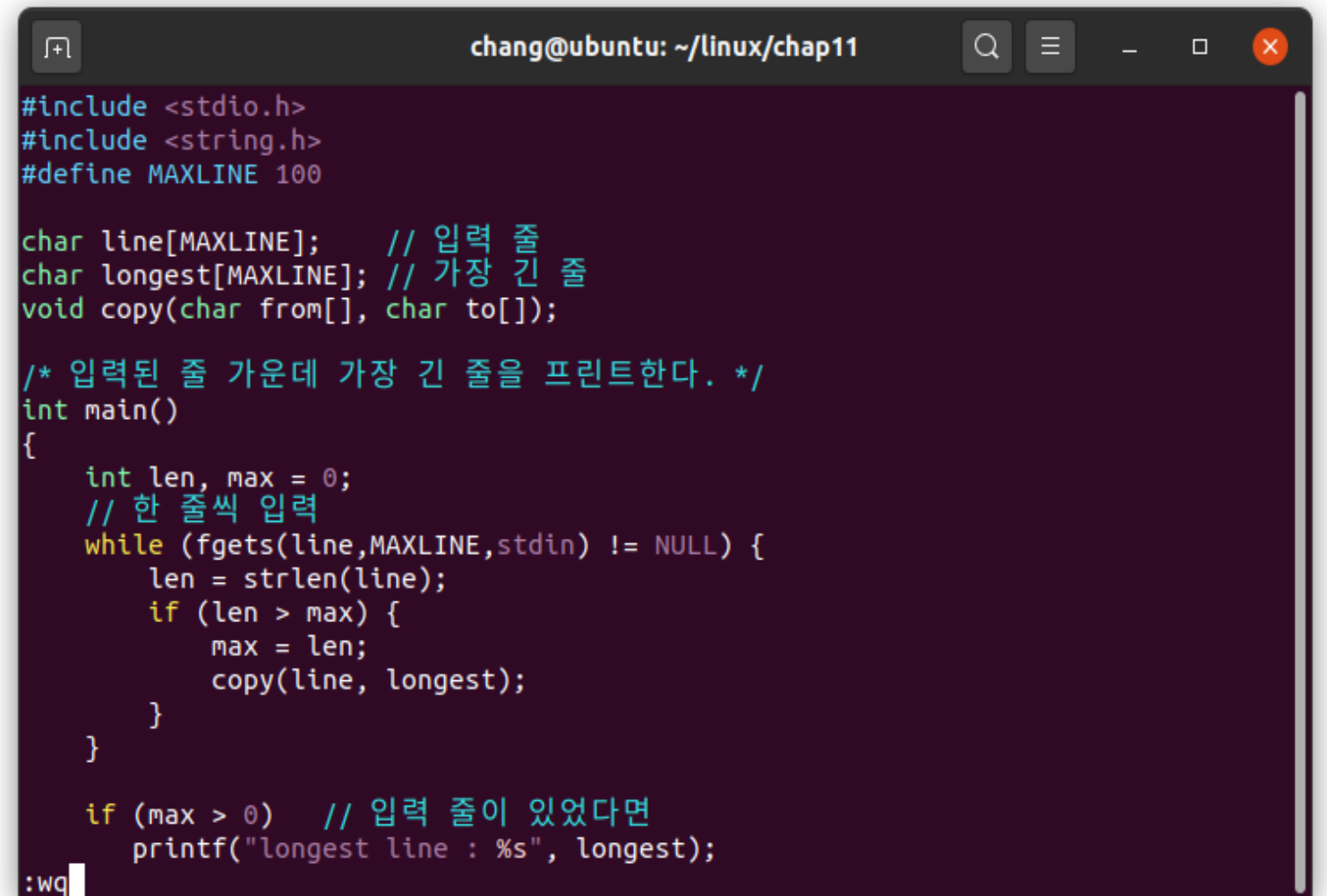
- **저장하지 않고 끝내기**

:q

아무런 작업을 하지 않은 경우 종료

:q!

작업 내용을 저장하지 않고 종료



```
chang@ubuntu: ~/linux/chap11
#include <stdio.h>
#include <string.h>
#define MAXLINE 100

char line[MAXLINE]; // 입력 줄
char longest[MAXLINE]; // 가장 긴 줄
void copy(char from[], char to[]);

/* 입력된 줄 가운데 가장 긴 줄을 프린트한다. */
int main()
{
    int len, max = 0;
    // 한 줄씩 입력
    while (fgets(line, MAXLINE, stdin) != NULL) {
        len = strlen(line);
        if (len > max) {
            max = len;
            copy(line, longest);
        }
    }

    if (max > 0) // 입력 줄이 있었다면
        printf("longest line : %s", longest);
}

:wq
```

기타

- 다른 파일 편집

- :e 파일이름
현재 파일 대신에 주어진 파일 열기
- :e#
이전 파일을 다시 열기

- 줄 번호 붙이기

줄 번호를 붙이거나 없애기

:set number 줄번호 붙이기

:se nu 줄번호 붙이기

:set nonumber 줄번호 없애기

:se non 줄번호 없애기

- 셸 명령어 수행

- 편집기 내에서 셸 명령어 수행
- :!ls
- :!cat

핵심 개념

- gedit는 GNU가 제공하는 대표적인 텍스트 편집기이다.
- gcc 컴파일러는 C 프로그램을 컴파일한다. 옵션을 사용하지 않으면 실행파일 a.out를 생성한다.
- make 시스템은 메이크파일(makefile 혹은 Makefile)을 이용하여 보통 실행 파일을 빌드한다.
- gdb 디버거는 실행파일을 이용하여 디버깅 모드로 실행한다.
- vi 에디터는 명령 모드와 입력 모드가 구분되어 있으며 시작하면 명령 모드이다.