



3. C 프로그래밍 환경



리눅스 시스템 개요

유닉스/리눅스 구조

- 운영체제
 - 컴퓨터의 하드웨어 자원을 운영 관리하고
 - 프로그램을 실행할 수 있는 환경을 제공.
- 커널(kernel)
 - 운영체제의 핵심으로 하드웨어 운영 및 관리
- 시스템 호출(system call)
 - 커널이 제공하는 서비스에 대한 프로그래밍 인터페이스 역할
- 셸(shell)
 - 사용자와 운영체제 사이의 인터페이스
 - 사용자로부터 명령어를 입력 받아 해석하여 수행해주는 명령어 해석기

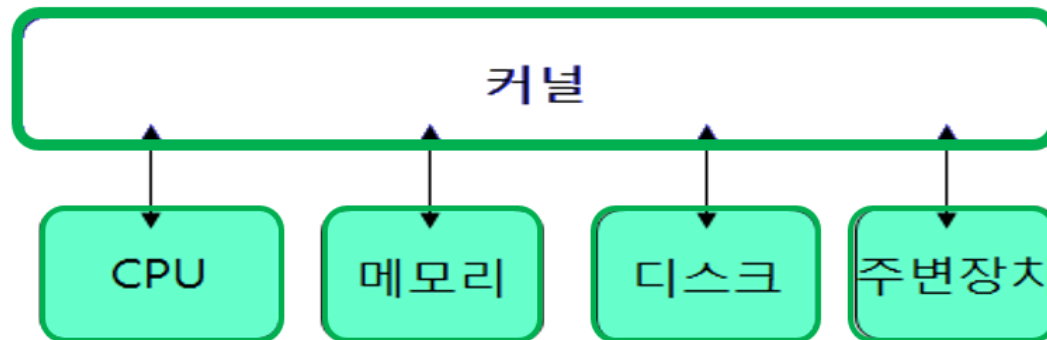


그림 1.3 유닉스 운영체제 구조

커널

■ 커널의 역할

- 하드웨어를 운영 관리하여
- 프로세스, 파일, 메모리, 통신, 주변장치 등을
- 관리하는 서비스를 제공한다.



커널의 역할

- 프로세스 관리(Process management)
 - 여러 프로그램이 실행될 수 있도록
 - 프로세스들을 CPU 스케줄링하여 동시에 수행되도록 한다.
- 파일 관리(File management)
 - 디스크와 같은 저장장치 상에 파일 시스템을 구성하여 파일을 관리한다.
- 메모리 관리(Memory management)
 - 메인 메모리가 효과적으로 사용될 수 있도록 관리한다.
- 통신 관리(Communication management)
 - 네트워크를 통해 정보를 주고받을 수 있도록 관리한다.
- 주변장치 관리(Device management)
 - 모니터, 키보드, 마우스와 같은 장치를 사용할 수 있도록 관리한다.

C 프로그래밍 환경

1. C 언어 요약
2. C 컴파일러
3. 자동 빌드 도구
4. gdb 디버거
5. 이클립스 통합개발환경
6. Vi 에디터



C 프로그래밍 환경



3.1 C 언어 요약

C 프로그램

- 여러 개의 .c 파일로 구성됨
- 하나의 .c 파일
 - #include ...
 - #define ...
 - 전역 변수 선언들
 - 함수 정의

프로그램 3.1

```
#include <stdio.h>
int fact(int n); // 함수 선언
int global = 10; // 전역 변수 선언

int main() {      // main 함수
    int a = 4, b; // 지역 변수 선언
    b = fact(a);  // 함수 호출
    printf("b=%d", b);
    global = global + b;
}

int fact(int n) { // 함수 정의
    int result = 1; // 지역 변수 선언
    for (int i = 1; i < n; i++)
        result = result * i;
    return result; // 함수 리턴
}
```

함수 정의

- 함수 정의 형식

```
리턴타입 함수명(매개변수 선언) {  
    함수 본체  
}
```

```
int fact(int n)  
{  
    int result = 1;    // 지역 변수  
    ...  
}
```

함수 호출

- 함수 호출
 - 인수(argument)의 값을 매개변수(parameter)에 전달
- 매개변수 전달
 - 값 전달(pass by value)만 사용됨
- 값 전달 (pass by value)
 - 실 매개변수의 값을 형식 매개변수에 전달
 - 포인터 값을 전달할 때는 주소가 전달됨
- fact 함수 호출 예
 - 매개변수 x에 변수 a의 값 4가 전달된다.
 - 마치 $x = a$; 처럼

return 문과 리턴타입

- return 문
 - 형식: `return` 수식;
 - 함수가 어떤 값을 돌려주는지 명시함
 - 리턴 값은 리턴타입(return type)과 일치해야 한다.

배열

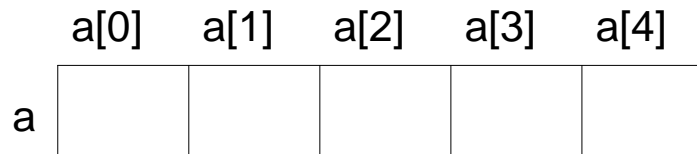
자료형 배열명[N];

크기가 N인 일차원 배열을 선언한다.

- 예

```
int a[5];
```

```
int a[]={0,2,4,6,8};
```



- for 문과 배열

```
int a[N], sum = 0;
```

```
for (int i= 0; i< N; ++i) {
```

```
    sum = sum + a[i];
```

```
}
```

배열 예: 프로그램 3.2

```
#include <stdio.h>
#define N 5
int main()
{
    int sum = 0;
    int a[N];
    for(int i=0; i < N; i++)
        a[i] = i*2;
    for(int i=0; i < N; i++)
        sum += a[i];
    printf("배열 a[]의 합=%d\n", sum);
}
```

이차원 배열

자료형 배열명[N][M];

크기가 $N \times M$ 인 이차원 배열을 선언한다.

- 예

```
int x[2][3];
```

```
int y[2][3] = {{1,3,5}, {2,4,6}};
```

- 중첩 for 문과 이차원 배열

```
int a[N][M], sum = 0;
```

```
for(int i=0; i < N; i++) {
```

```
    for(int j=0; j < M; j++) {
```

```
        sum = sum + a[i][j];
```

```
    }
```

```
    ...
```

```
}
```

배열 x

	0	1	2
0	?	?	?
1	?	?	?

초기화되지 않은 배열

배열 y

	0	1	2
0	1	3	5
1	2	4	6

초기화된 배열

포인터와 포인터 변수

■ 포인터 변수

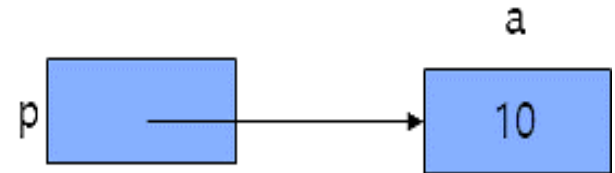
- 주소 값(포인터)을 저장할 수 있는 변수로 다른 메모리 공간을 가리킨다.

■ 포인터 변수 선언

```
int a=10;
```

```
int *p=&a;
```

- p는 정수형 변수를 가리키는 **포인터 변수**
- p는 변수 a의 주소값으로 초기화 됨.
- 변수 a와 *p는 동일하게 사용 가능.



■ 주소 연산자 &

&a 변수 a의 주소를 나타냄

■ 주소 참조(dereferencing)

*p 포인터 p가 가리키는 곳에 있는 값

포인터 예: 프로그램 3.3

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

```
int main(){
```

```
    int a=10;
```

```
    int *p=&a;
```

```
    printf("%p %d\n", &a, a);
```

```
    printf("%p %d\n", p, *p);
```

```
    *p = *p + 10;
```

```
    printf("%p %d\n", &a, a);
```

```
    printf("%p %d\n", p, *p);
```

```
    a=30;
```

```
    printf("%p %d\n", &a, a);
```

```
    printf("%p %d\n", p, *p);
```

```
}
```

// int *p; p=&a; 두 개의 문장

// a=a+10과 동일

// *p=30과 동일

실행결과:

```
0x7fffb1b4e634 10
0x7fffb1b4e634 10
0x7fffb1b4e634 20
0x7fffb1b4e634 20
0x7fffb1b4e634 30
0x7fffb1b4e634 30
```

함수와 포인터

- swap의 C 버전

```
void swap(int *px; int *py)
{
    int t;
    t = *px;
    *px = *py;
    *py = t;
}
```

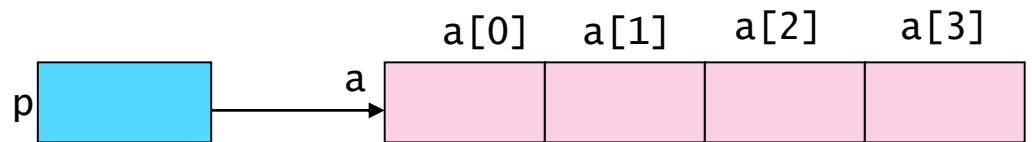
- 사용

```
int a = 10, b = 20;
intswap(&a, &b);
printf("%d %d", a, b);
```

배열과 포인터

■ 포인터 변수

- 기억장소(변수)에 대한 주소 값을 저장하는 주소 변수
- `int *p;`



■ 배열

- `int a[4];`
- `p = a;`

```
#include <stdio.h>
int main() {
    int *p, a[4] = {10,20,30,40};
    printf("%d %d %d %d\n", a[0], a[1], a[2], a[3]);
    p=a;
    printf("%d %d %d %d %d\n", *p, *(p+1), p[1], *(p+2), p[2]);
}
```

문자열과 포인터

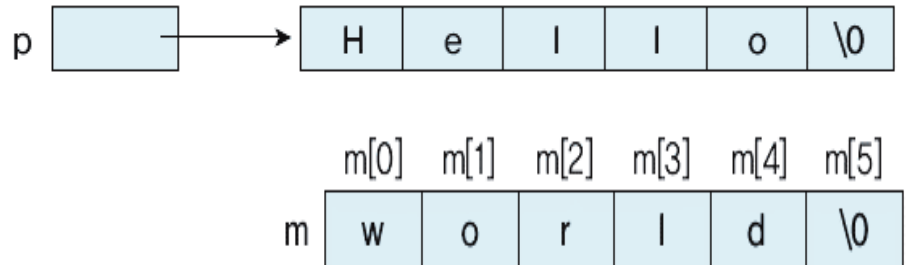
- 문자형 포인터 변수를 사용하여 문자열을 가리킬 수 있다.

```
char *변수명 [= "문자열"];
```

- 예를 들어

```
char *p = "Hello";
```

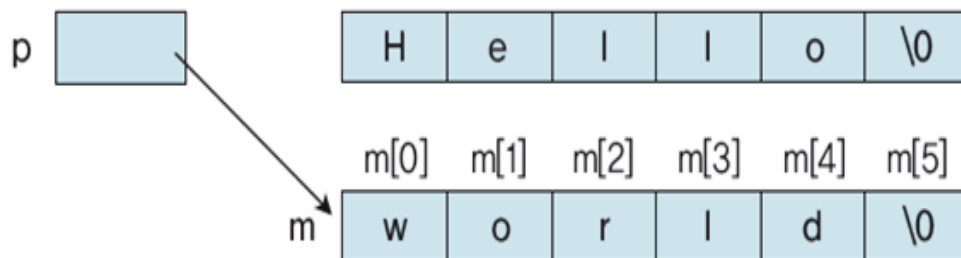
```
char m[] = "world";
```



프로그램 3.5

```
#include <stdio.h>

int main()
{
    char *p = "Hello!";
    char m[] = "world";
    printf("%s %s\n", p, m);
    p = m;
    printf("%s\n", p);
    while (*p)
        putchar(*p++);
}
```



여러 개의 문자열 저장

- 문자형의 이차원 배열을 사용

```
char colors[3][10] = {"red", "blue", "white"};
```

- 포인터 배열이용 : 배열의 각 원소가 포인터인 배열

```
char *ptr[3] = {"red", "cyan", "black"};
```

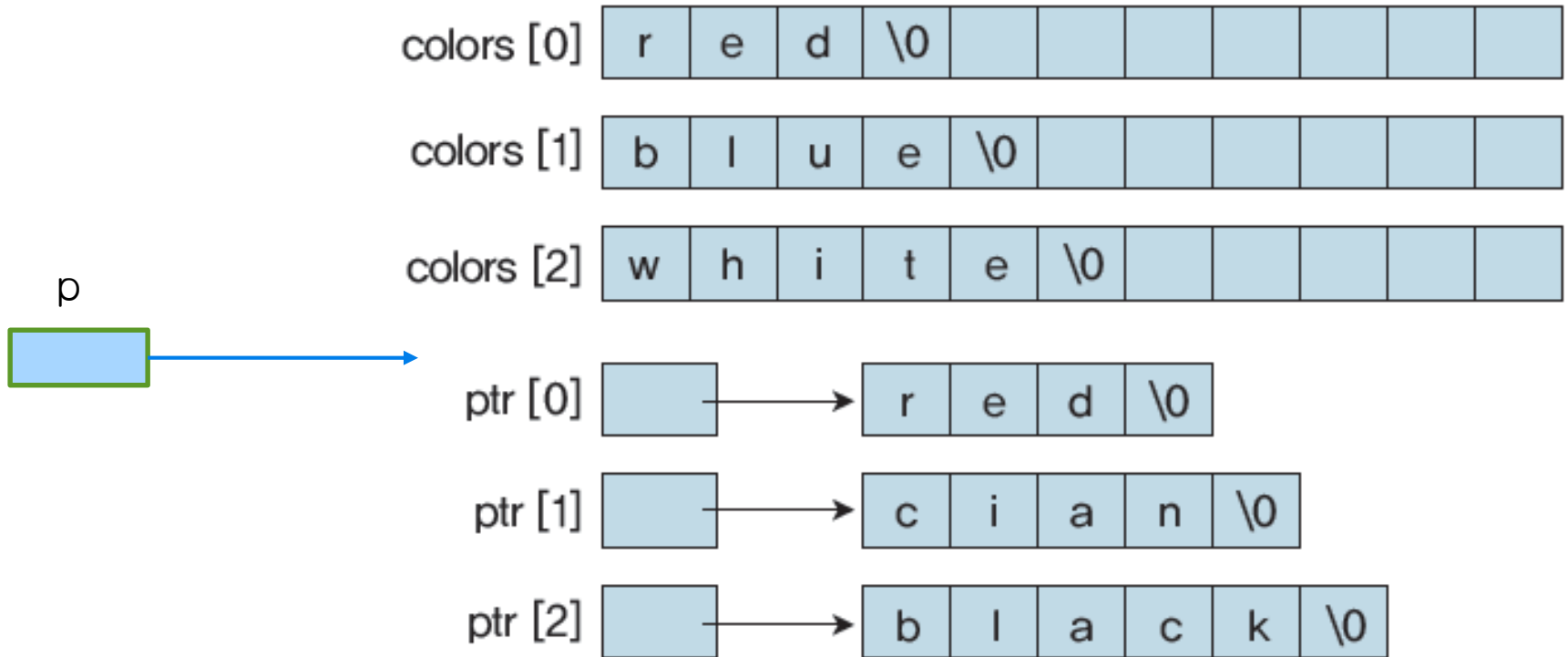
- 포인터의 포인터

- 포인터의 대상이 포인터인 포인터 변수

```
char *ptr[3] = {"red", "cyan", "black"};
```

```
char **p = ptr;
```

문자열과 포인터 배열



구조체 struct

- 구조체 선언

```
struct 구조체_이름 {  
    자료형1 멤버변수1;  
    자료형1 멤버변수1;  
    ...  
} [변수명 [= {초기값,...}]];
```

- 구조체 선언 후에 구조체 변수선언

```
struct 구조체명 변수명 [= {초기값, ...}];
```

구조체 예제 1

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

```
struct my {
```

```
    int  a;
```

```
    char b;
```

```
    float c;
```

```
} x={1, 'a', 1.5};
```

```
int main()
```

```
{
```

```
    struct my y={3, 'c', 3.5};
```

```
    printf("\n x.a=%d x.b=%d x.c=%3.1f", x.a, x.b, x.c);
```

```
    printf("\n y.a=%d y.b=%d y.c=%3.1f", y.a, y.b, y.c);
```

```
}
```

구조체 예제 2

```
#include <stdio.h>
void main()
{
    int i;
    for (i=0; i<3; i++ ) {
        printf("이름: ");    scanf("%s", man[i].name);
        printf("나이: ");    scanf("%d", &man[i].age);
    }

    for (i=0; i<3; i++)
        printf("\n이름=%10s  나이=%3d", man[i].name, man[i].age);

    for (i=0, p=man; i<3; i++, p++)
        printf("\n이름=%10s  나이=%3d", (*p).name, (*p).age);

    for (i=0, p=man; i<3; i++, p++)
        printf("\n이름=%10s  나이=%3d", p->name, p->age);
}
```

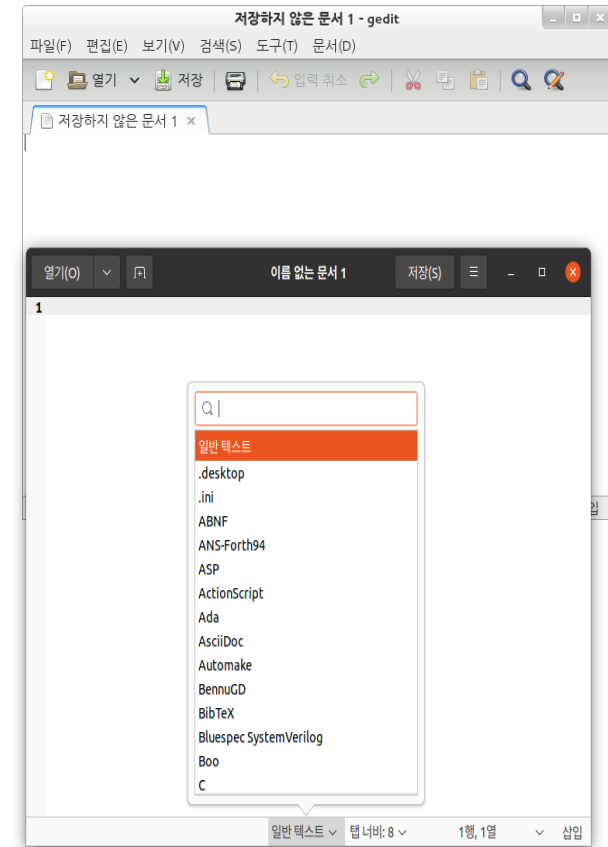
```
struct student {
    char name[10];
    int age;
} man[3], *p;
```



3.2 프로그램 편집과 C 컴파일러

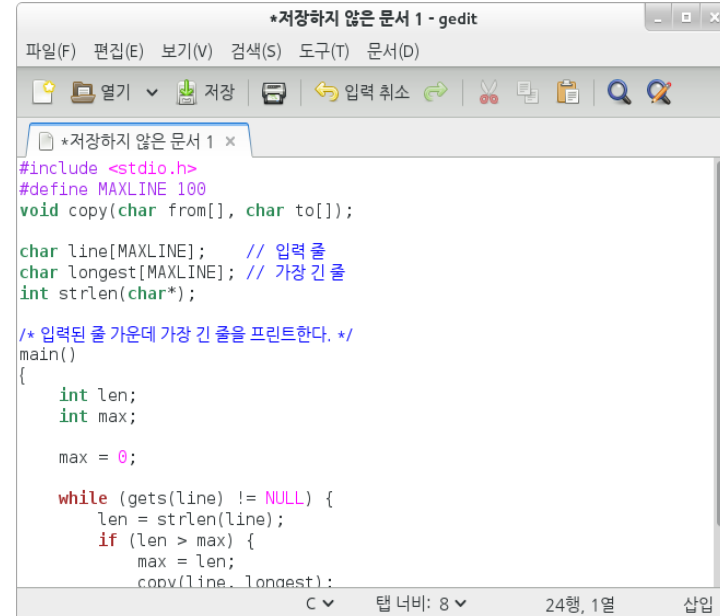
gedit 문서편집기

- 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] -> [지에디트] 선택
 - 터미널
 - `$ gedit [파일이름] &`
 - 파일 관리자:
 - 텍스트 파일 클릭하면 자동실행



단일 모듈 프로그램

- gedit를 이용한 프로그램 작성
- [보기] 메뉴에서 C 구문 강조 기능을 설정
- 프로그램 편집하는 화면
 - #include 같은 전처리 지시자는 분홍색
 - 주석은 파란색
 - 자료형 이름은 초록색
 - if나 while 같은 문장 키워드는 브라운 색



The screenshot shows the gedit text editor window titled '*저장하지 않은 문서 1 - gedit'. The menu bar includes '파일(F)', '편집(E)', '보기(V)', '검색(S)', '도구(T)', and '문서(D)'. The toolbar contains icons for opening, saving, printing, undo, redo, cut, copy, paste, and search. The code editor displays the following C code with syntax highlighting:

```
#include <stdio.h>
#define MAXLINE 100
void copy(char from[], char to[]);

char line[MAXLINE]; // 입력 줄
char longest[MAXLINE]; // 가장 긴 줄
int strlen(char*);

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

    max = 0;

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

 The status bar at the bottom indicates 'C', '탭 너비: 8', '24행, 1열', and '삽입'.

gcc 컴파일러

- gcc(GNU cc) 컴파일러

```
$ gcc [-옵션] 파일
```

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

- 간단한 컴파일

```
$ gcc longest.c
```

```
$ a.out // 실행
```

- 옵션 사용

- -c 옵션: 목적 파일 생성

```
$ gcc -c longest.c
```

- -o 옵션: 실행 파일 생성

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

- 실행

```
$ longest // 실행
```

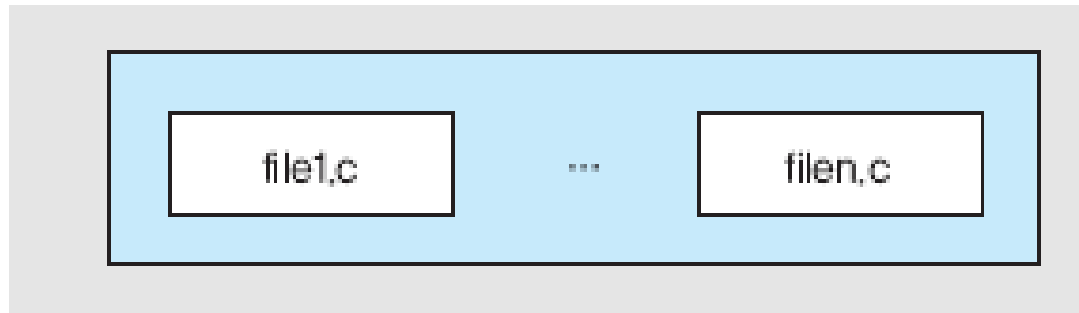
longest.c:단일 모듈 프로그램

```
#include <stdio.h>
#include <string.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
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 "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[]);
```



3.3 자동 빌드 도구

make 시스템의 필요성

- 다중 모듈 프로그램의 자동 빌드
- make 시스템
 - 대규모 프로그램의 경우에는 헤더, 소스 파일, 목적 파일, 실행 파일의 모든 관계를 기억하고 체계적으로 관리
 - make 시스템을 이용하여 효과적으로 작업
- 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

메이크파일

- 메이크파일
 - 실행 파일을 만들기 위해 필요한 파일들과
 - 그들 사이의 의존 관계,
 - 만드는 방법 등을 기술
- make 시스템
 - 메이크파일을 이용하여 파일의 상호 의존 관계를 파악하여 실행 파일을 쉽게 다시 만듦
- 사용법

```
$ make [-f 메이크파일]
```

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

메이크파일의 구성

- Makefile 의 구성 형식

목표(target) : 의존리스트

(tab문자) 명령리스트

목표(target): 의존리스트(dependencies)

명령리스트(commands)

- Makefile 예

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

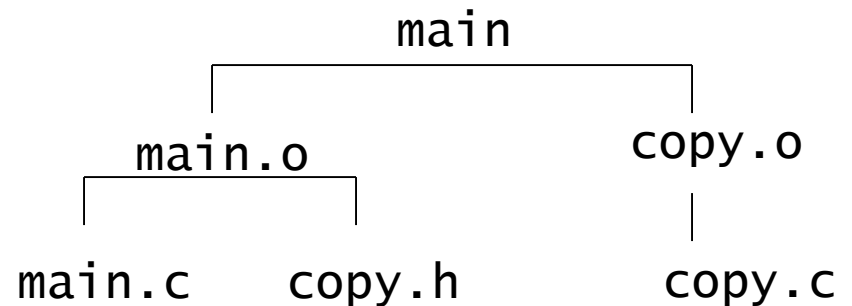
```
    gcc -o main main.o  copy.o
```

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

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

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

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



메이크파일의 구성

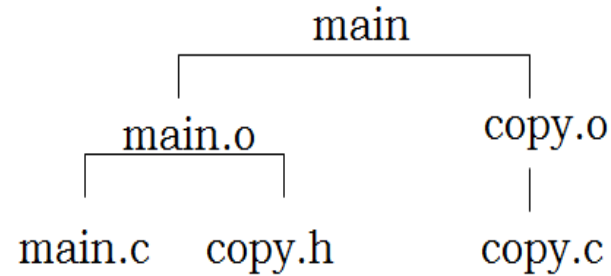
- make 실행

\$ make 혹은 \$ make [-f] filename

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



3.4 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

- 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 디버거는 실행파일을 이용하여 디버깅 모드로 실행

```
$ gdb [실행파일]
```

```
$ gdb main
```

```
GNU gdb (Ubuntu 9.1-0ubuntu1) 9.1
```

```
Copyright (C) 2020 Free Software Foundation, Inc.
```

```
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
```

```
This is free software: you are free to change and redistribute it.
```

```
...
```

```
Reading symbols from main...
```

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

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

9 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

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

(gdb) n

11 ++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.



3.5 이클립스 통합개발환경

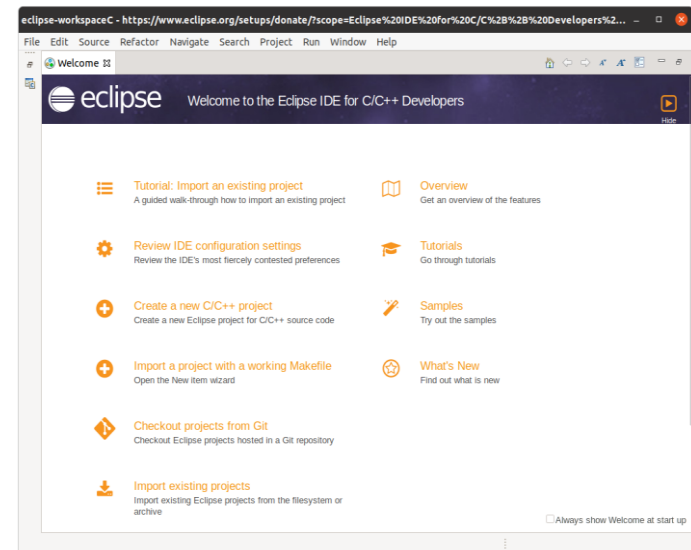
이클립스(Eclipse)

■ 통합 개발 환경

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

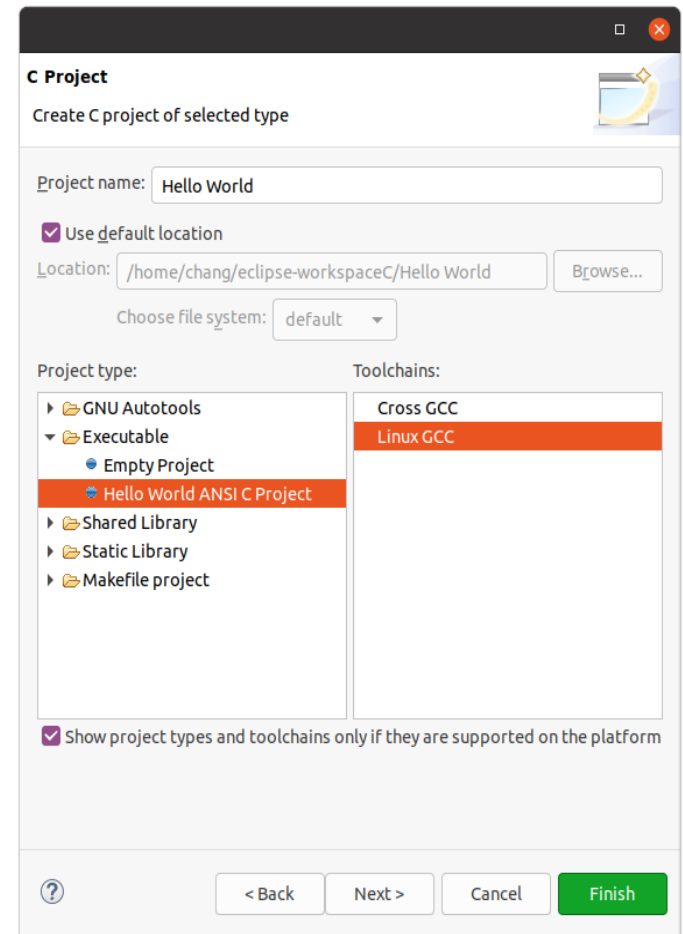
■ 이클립스 설치

- <https://www.eclipse.org>
- 리눅스용 이클립스를 다운받아 설치가능



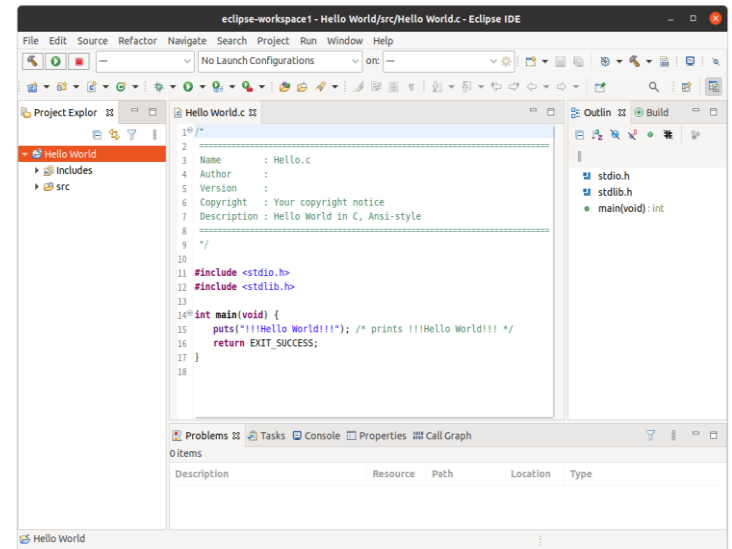
새로운 C 프로젝트를 생성하기

- 'File → New → C/C++ Projects'
- C Managed Build 선택
- 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 includes 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.



핵심개념

- 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.