제 1 장
고급 프로그래밍의 개요

ACS30021
고급 프로그래밍

나보균 (bkna@kpu.ac.kr)

컴퓨터 공학과
한국산업기술 대학교

학습 목표

- 운영체제
  - ✓ Unix, Linux, MS Windows 특징
- 유닉스 시스템 프로그래밍 정의
  - ✓ 기본적 프로그래밍 언어 복습
  - ✓ C 컴파일러와 make 도구
- 유닉스 시스템 호출과 라이브러리 함수
- 유닉스 시스템의 기본 명령

Operating System 이란?

- 응용프로그래밍
- 시스템 프로그래밍
  - ✓ 시스템 호출(시스템 콜, 시스템 함수)을 사용해 프로그램을 작성하는 것
  - ✓ 시스템 호출 – 파일 시스템 접근, 사용자 정보, 시스템 정보, 시스템 시간 정보, 네크워킹 등의 서비스를 이용해 프로그램을 구현할 수 있도록 제공 되는 인터페이스
- 운영체제란?
  - ✓ 자원 관리자 (Resource Manager)
  - ✓ 응용에게 자원에 대한 서비스 제공 (Computing Environment)
- 자원의 종류
  - ✓ 물리적인 자원 : 처리기, 메모리, 디스크, 터미널, 네트워크, ...
  - ✓ 추상적인 자원 : 태스크, 세그먼트/페이지, 파일, 드라이버, 통신 프로토콜, 패킷, 보안, ...

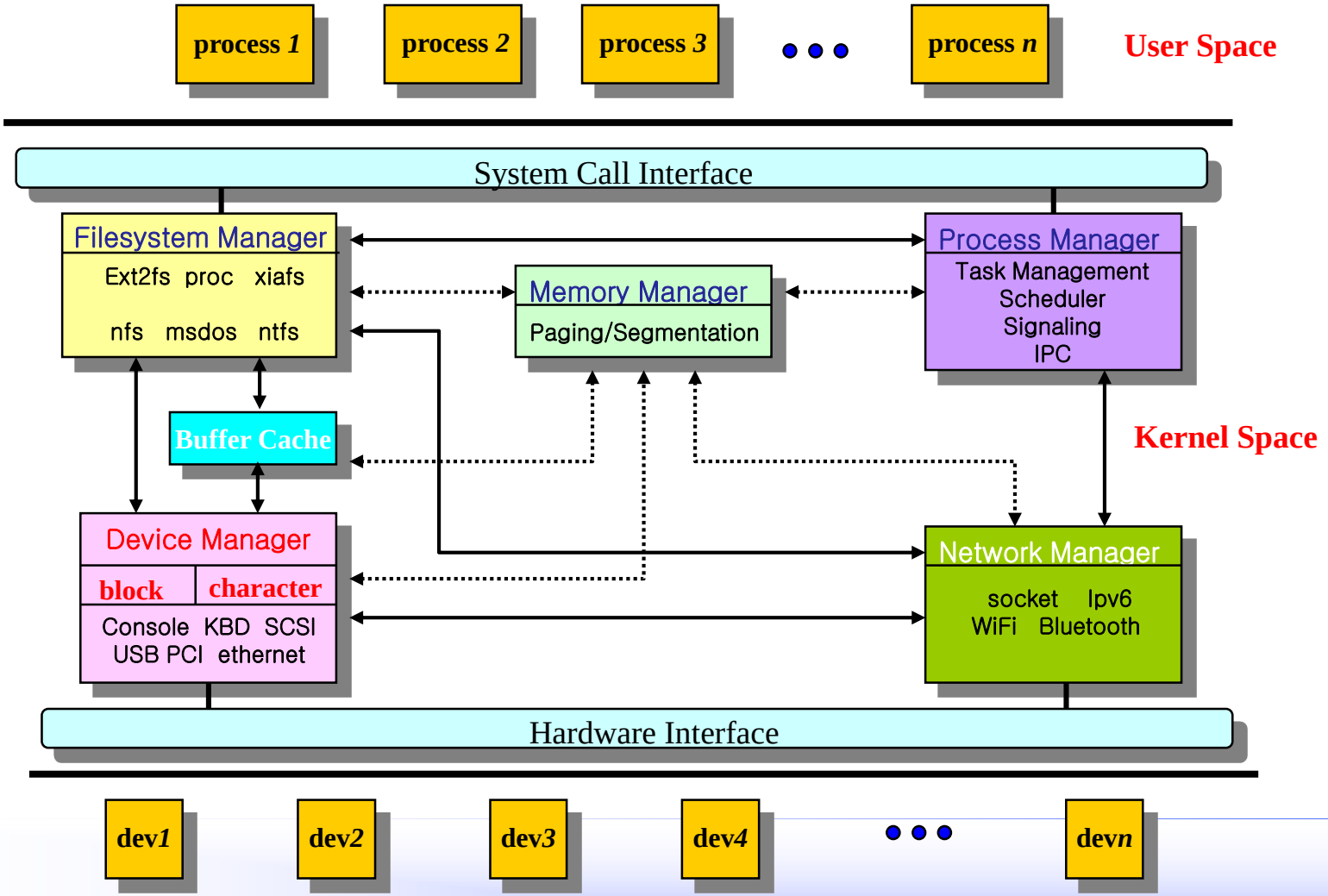
LINUX Operating System

- 리눅스 특징 (from the Book of "Running Linux by M.K Dalheimer")
  - It's free: open source
  - It's popular: ARM, Pentium(II, III, IV), Sparc, Alpha, m68k, PowerPC, MIPS,
  - It's powerful
  - It's under your control: 내부를 볼 수 있다.
  - It's robust: say goodbye to the "blue screen"
  - It's full-featured: multi-thread, advanced networking, GUI
  - It's small: 1MB 이하로 커널 구성 가능
  - It's big: powerful utilities and applications
  - It's well documented: LDP (Linux Documentation Projects)

  - http://www.linux.org, http://www.kernel.org , http://www.unix.org

  - VMWare

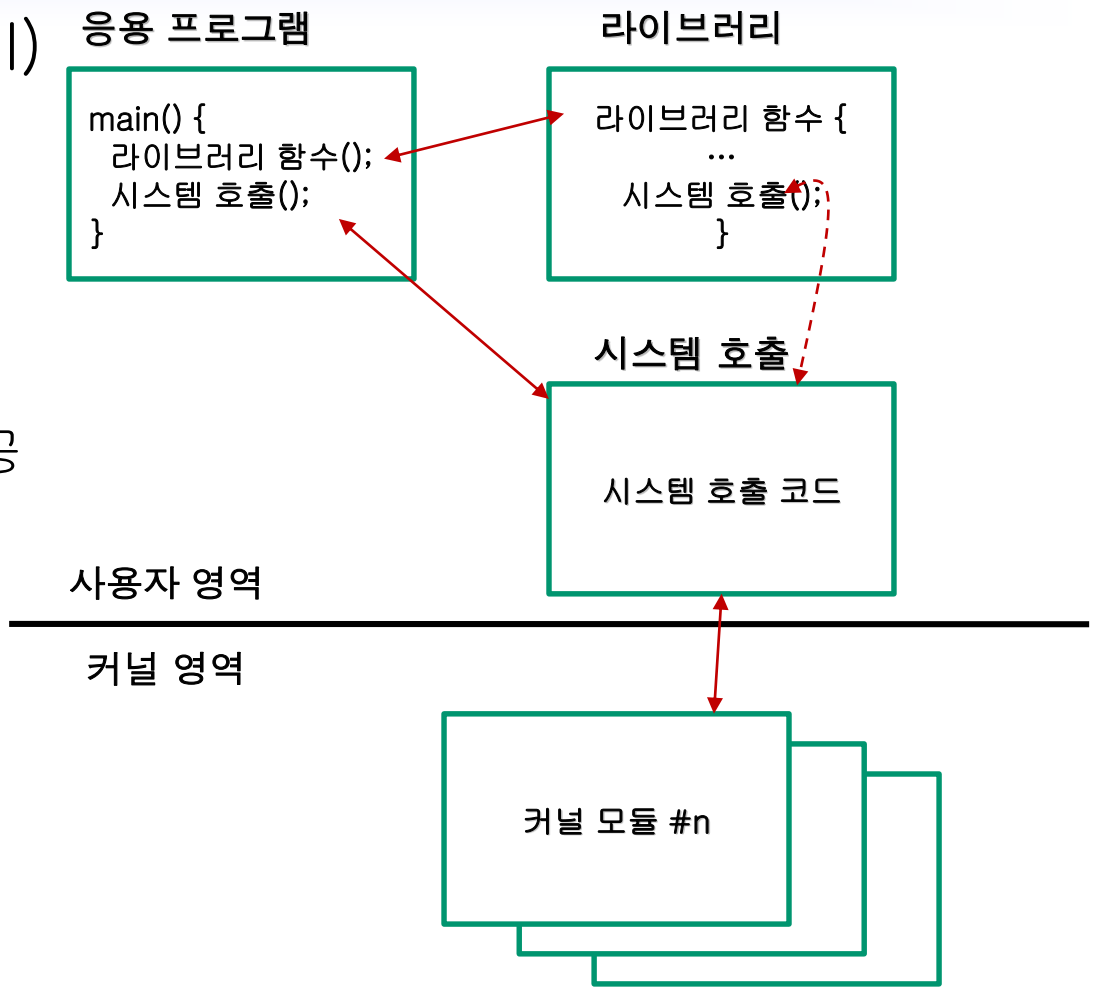
Operating System의 구조(Linux)



(Source : Linux Kernel Internals)

System Calls vs. Libraries

- 시스템 호출 (system call)
  - ✓ OS가 직접 제공하는 함수
  - ✓ 커널 수준에서 수행됨
- 라이브러리(library)
  - ✓ 사용자의 편의를 위해 제공되는 함수들의 집합
  - ✓ 사용자 수준에서 수행됨



시스템 호출과 라이브러리 함수의 비교

- 시스템 호출 : man 페이지가 섹션 2에 속함

```
System Calls                                                    open(2)

NAME
        open, openat - open a file
SYNOPSIS
        #include <sys/types.h>
```

- 라이브러리 함수 : man 페이지가 섹션 3에 속함

```
Standard C Library Functions                                    fopen(3C)

NAME
        fopen - open a stream
SYNOPSIS
        #include <stdio.h>
```

- man 2 fopen v.s. man fopen

시스템 호출과 라이브러리 함수의 비교

- 시스템 호출의 오류 처리방법
  - ✓ 성공하면 0을 리턴, 실패하면 -1을 리턴
  - ✓ 전역변수 errno에 오류 코드 저장 : man 페이지에서 코드값 확인 가능

[예제 1-1] 시스템 호출 오류 처리하기 ex1_1.c

```
01  #include <unistd.h>
02  #include <stdio.h>
03
04  extern int errno;
05
06  int main(void) {
07      if (access("unix.txt", F_OK) == -1) {
08          printf("errno=%d\n", errno);
09      }
10
11      return 0;
12  }
```

```
# ex1_1.out
errno=2
```

```
# vi /usr/include/sys/errno.h
......
/*
* Error codes
*/
#define EPERM   1       /* Not super-user */
#define ENOENT  2       /* No such file or directory */
......
```

시스템 호출과 라이브러리 함수의 비교

- 라이브러리 함수의 오류 처리방법
  - ✓ 오류가 발생하면 NULL을 리턴, 함수의 리턴값이 int 형이면 -1 리턴
  - ✓ errno 변수에 오류 코드 저장

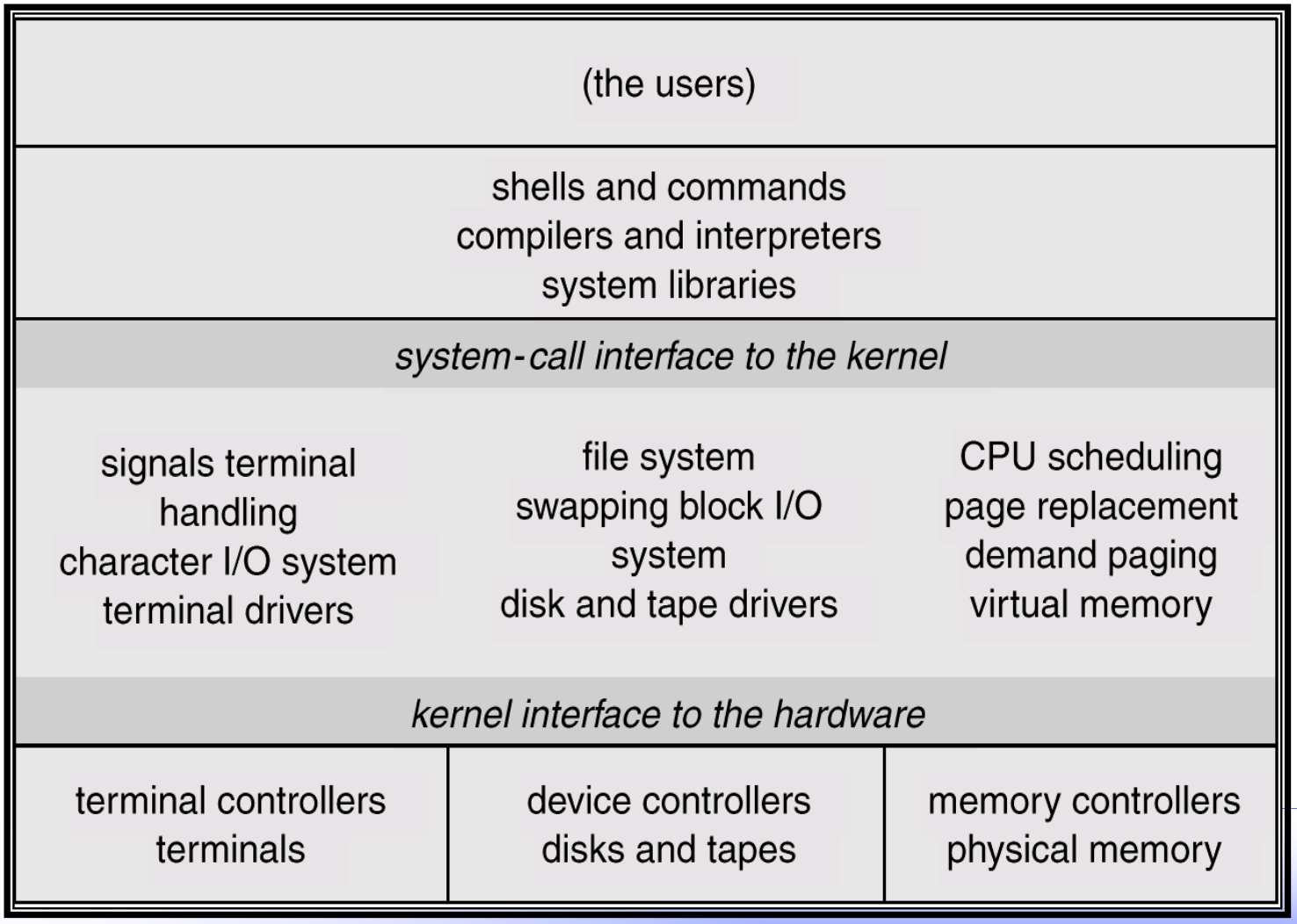
[예제 1-2] 라이브러리 함수 오류 처리하기

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  extern int errno;
05
06  int main(void) {
07      FILE *fp;
08
09      if ((fp = fopen("unix.txt", "r")) == NULL) {
10          printf("errno=%d\n", errno);
11          exit(1);
12      }
13      fclose(fp);
14
15      return 0;
16  }
```

```
# ex1_2.out
errno=2
```

man fopen에서 확인

Software 계층 구조



Concurrent 와 Parallel

- Concurrent Processing
  - 시분할 멀티 프로세싱
    - 퀀텀 – 시분할 멀티 프로세싱 시스템에서 프로그램을 실행하는데 있어 여러 프로세스들이 단위 시간당 많은 시간을 나누어 실행되는데 있어서 한 프로세스에게 할당된 CPU 실행 시간
    - 퀀텀이 모여서 하나의 프로그램이 실행
  - 주변 장치의 공유
- Parallel Processing
  - 다중 프로세서에 의한 프로세싱
- Interrupt and Signal
  - 인터럽트 – 전기적인 신호로 프로세스 내의 하드웨어 플래그를 설정 . 예, CPU에 있는 플래그 레지스터
  - 시그널 – 이벤트 발생을 알리는 소프트웨어 모듈로 인터럽트에 대한 운영체제의 반응을 발생

컴파일 환경

- Makefile과 make
  - ✓ 소스 파일이 여러 개를 묶어서 실행파일을 생성하는 도구
  - ✓ make 명령은 Makefile의 내용에 따라 컴파일
  - ✓ /usr/ccs/bin을 경로에 추가해야함

```
# vi ~/.profile
......
PATH=$PATH:/usr/local/bin:/usr/ccs/bin
export PATH
```

[예제 1-3] ex1_3_main.c

```
01  #include <stdio.h>
02  extern int addnum(int a, int b);
03
04  int main(void) {
05      int sum;
06
07      sum = addnum(1, 5);
08      printf("Sum 1~5 = %d\n", sum);
09
10      return 0;
11  }
```

[예제 1-3] ex1_3_addnum.c

```
01  int addnum(int a, int b) {
02      int sum = 0;
03
04      for (; a <= b; a++)
05          sum += a;
06      return sum;
07  }
```

컴파일 환경

[예제 1-3] make 명령 사용하기 Makefile

```
01  # Makefile
02
03  CC=gcc
04  CFLAGS=
05  OBJS=ex1_3_main.o ex1_3_addnum.o
06  LIBS=
07  all:    add
08
09  add:    $(OBJS)
10    $(CC) $(CFLAGS) -o add $(OBJS) $(LIBS)
11
12  ex1_3_main.o:          ex1_3_main.c
  $(CC) $(CFLAGS) -c ex1_3_main.c
14  ex1_3_addnum.o:        ex1_3_addnum.c
15    $(CC) $(CFLAGS) -c ex1_3_addnum.c
16
17  clean:
18    rm -f $(OBJS) add core
```

ex1_3_main.c와 ex1_3_addnum.c를 묶어서 add라는 실행파일 생성

```
# make
gcc  -c ex1_3_main.c
gcc  -c ex1_3_addnum.c
gcc  -o add ex1_3_main.o
        ex1_3_addnum.o
# ls
Makefile   add* ex1_3_addnum.c
ex1_3_addnum.o   ex1_3_main.c
ex1_3_main.o
# add
Sum 1~5 = 15
```

오류 처리 함수

- 오류 메시지 출력 : perror(3)

```
#include <stdio.h>
void perror(const char *s);
```

[예제 1-4] perror 함수 사용하기 ex1_4.c

```
01  #include <sys/errno.h>
02  #include <unistd.h>
03  #include <stdlib.h>
04  #include <stdio.h>
05
06  int main(void) {
07      if (access("unix.txt", R_OK) == -1) {
08          perror("unix.txt");
09          exit(1);
10      }
11
12      return 0;
13  }
```

```
# ex1_4.out
unix.txt: No such file or directory
```

오류 처리 함수

❑ 오류 메시지 출력 : strerror(3)

```
#include <string.h>
char *strerror(int errnum);
```

[예제 1-5] strerror 함수 사용하기 ex1_5.c

```
01  #include <sys/errno.h>
02  #include <unistd.h>
03  #include <stdlib.h>
04  #include <stdio.h>
05  #include <string.h>
06
07  extern int errno;
08
09  int main(void) {
10      char *err;
11
12      if (access("unix.txt", R_OK) == -1) {
13          err = strerror(errno);
14          printf("오류:%s(unix.txt)\n", err);
15          exit(1);
16      }
17
18      return 0;
19  }
```

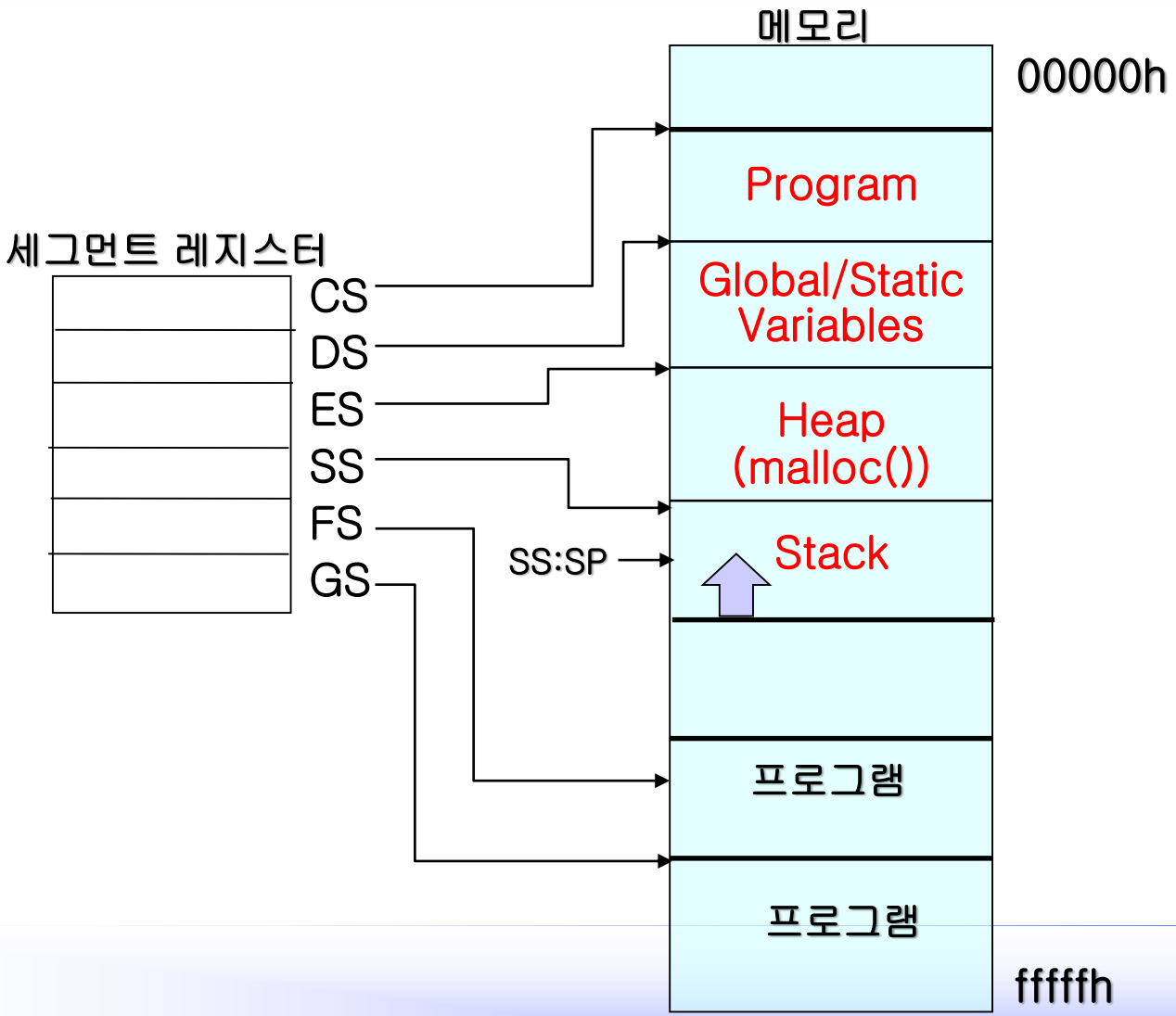
오류에 따라 메시지를 리턴

```
# ex1_5.out
오류: No such file or directory(unix.txt)
```

메모리 세그먼트

- 코드
  - ✓ 프로그램의 실행을 지시,
  - ✓ IP(Instruction Pointer/Program counter)와 함께 다음 실행 명령어 지시
- 데이터
  - ✓ 전역(global), 정적(static) 변수
- Heap
  - ✓ “Garbage collected storage”
  - ✓ 프로그램 실행 중 동적 메모리 할당
- Stack
  - ✓ Return address of function calls
  - ✓ Arguments to functions
  - ✓ Local variables

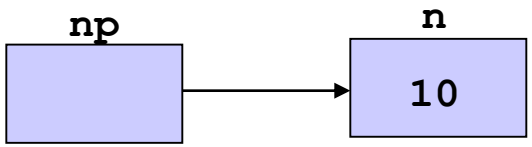
세그먼트 레지스터


메모리
00000h
Program
Global/Static Variables
Heap (malloc())
Stack
프로그램
프로그램
fffffh
세그먼트 레지스터
CS
DS
ES
SS
FS
GS
SS:SP

포인터 (pointer)

- 포인터
  - ✓ 주소를 값으로 갖는 변수
  - ✓ * (pointer-following 연산자; 역참조, 간접 연산자)
  - ✓ & (address-of 연산자; 참조 연산자)
- 예

```
int main( )
{
    int n, *np=NULL;
    int *pt=np;
    char c, *cp=&c;

    np = &n;
    *np = 10;

    return 0;
}
```



포인터와 함수

- call-by-value
- call-by-reference
- return type
  - ✓ dynamic memory allocation and return

포인터와 메모리 할당(1/3)

- 포인터
  - ✓ C언어에서는 어떤 타입 T에 대해 T의 포인터 타입이 존재
  - ✓ 포인터 타입의 실제 값은 메모리 주소가 됨
  - ✓ 포인터 타입에 사용되는 연산자
    - ➢ & : 참조(주소) 연산자 – reference operator
    - ➢ * : 역참조(간접 지시) 연산자 – dereference, indirect operator
  - ✓ 널 포인터
    - ➢ 정수 0값으로 표현
    - ➢ 널 포인터에 대한 검사
      if (pi ==NULL) 또는 if(!pi)

Ex) i(정수 변수), pi(정수에 대한 포인터)

```
int i, *pi=NULL;
pi = &i;

// i에 10을 저장
i = 10; 또는 *pi = 10;
```

*, & 연산자 구분

- C++ 언어에서 함수 매개변수
  - * : 포인터 변수 선언 – 예, int func (int *a)
  - & : nickname 변수 선언 – 예, int funct (int &a)
- Python 언어에서 함수 매개변수
  - 기호 * 과 & 를 사용하지 않고 nickname 변수 선언
  - **Mutable object** – you can change their content without changing their identity
    - list, dict, set, bytearray, user-defined classes (unless specifically made immutable)

```
def modify2 (li):
    li += [100, 200]

list = [1, 2, 3, 4, 5]
print (list)
modify2 (list)
print (list)
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 100, 200]
```

C++ 예제 코드

```
// g++ -std=c++11 -o refDeref_1 refDeref_1.cpp

#include <iostream>

using namespace std;

class PilotObj {
public:
  int a = 3;

  void func (int *, int &);
};

void  PilotObj:: func (int *ptr, int &nick) {
  cout << "Org is " << a << " deref is " <<  *ptr << " and ref is " << nick << endl;

  *ptr = 13;
  cout << "Org is " << a << " deref is " <<  *ptr << " and ref is " << nick << endl;

  nick = 23;
  cout << "Org is " << a << " deref is " <<  *ptr << " and ref is " << nick << endl;
}

int main () {
  PilotObj po;

  po.func (&po.a, po.a);

  return 0;
}
```

포인터와 메모리 할당(2/3)

- 동적 메모리 할당
  - ✓ 프로그램을 작성할 당시 또는 컴파일 당시 얼마나 많은 공간이 필요한지 알 수 없을 때 사용
  - ✓ Extra segment 에 저장 – 히프(heap) 기법
  - ✓ 새로운 메모리 공간이 필요할 때마다 함수 malloc을 호출해서 필요한 양의공간을 요구

```
int i, *pi=NULL;
float f, *pf=NULL;
pi = (int *) malloc(sizeof(int));
pf = (float *) malloc(sizeof(float));
*pi = 1024;
*pf = 3.14;
printf("an integer = %d, a float = %f\n", *pi, *pf);
free(pi);
free(pf);
```

메모리 할당과 반환

포인터와 메모리 할당(3/3)

- 포인터의 위험성
  - ✓ 포인터가 대상을 가리키고 있지 않을 때는 값을 전부 null로 정하는 것이 바람 직
  - ✓ 이는 프로그램 범위 밖이나 합당하지 않은 메모리 영역을 참조할 가능성이 적어짐
  - ✓ 명시적인 형 변환(type cast)

```
int *pi = NULL
pi = malloc(sizeof(int));        /* pi에 정수에 대한 포인터를 저장 */
pf = (float *) pi;               /* 정수에 대한 포인터를 부동소수에 대한
                                    포인터로 변환 */
```

동적 메모리 관리

- 동적 메모리의 할당과 반납 (stdlib.h)
  - ✓ void *malloc(size_t size)
  - ✓ free(left)
- 예

```
int *numberp=NULL;
char *characterp=NULL;

numberp = (int *) malloc(sizeof(int));
characterp=(char *) malloc(sizeof(char));

*numberp = 239;
*characterp = 'C';

printf("%d\n",  *numberp);
printf("%c\n", *characterp);

free(numberp);
free(characterp);
```

변수 포인터와 주소

- 아래 문장을 실행하면?

```
char *s= "한국산업기술대";

printf ("var s has %s at %p\n", s, s);

s = (char *) malloc(sizeof(char)* 16);

*s = 'C';
printf ("var s has %s at %p\n", s, s);

strcpy (s, "kpu");
printf ("var s has %s at %p\n", s, s);

free(s);
```

- 왜 잘못된 프로그래밍인가?

동적 할당 배열

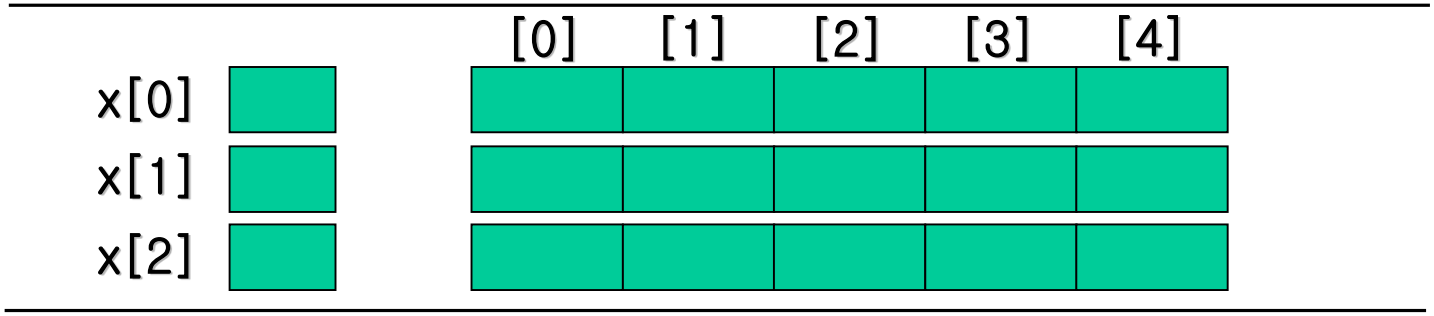
1차원 배열

```
int i, n, *list=NULL;

printf("Enter the number of number to generate: ");
scanf("%d", &n");
if (n < 1) {
      fprintf(stderr, "Improper value of n \n");
      exit (EXIT_FAILURE);
}

list = malloc (n * sizeof(int));
if (list == NULL) exit();

list [0] = 3;
*(list +1) = 8;   // *(&list[0] + 1) = 8; list[1] = 8;
```

※ N<1이거나 정렬할 수의 리스트를 저장 할 메모리가 충분치 않을 때 실패

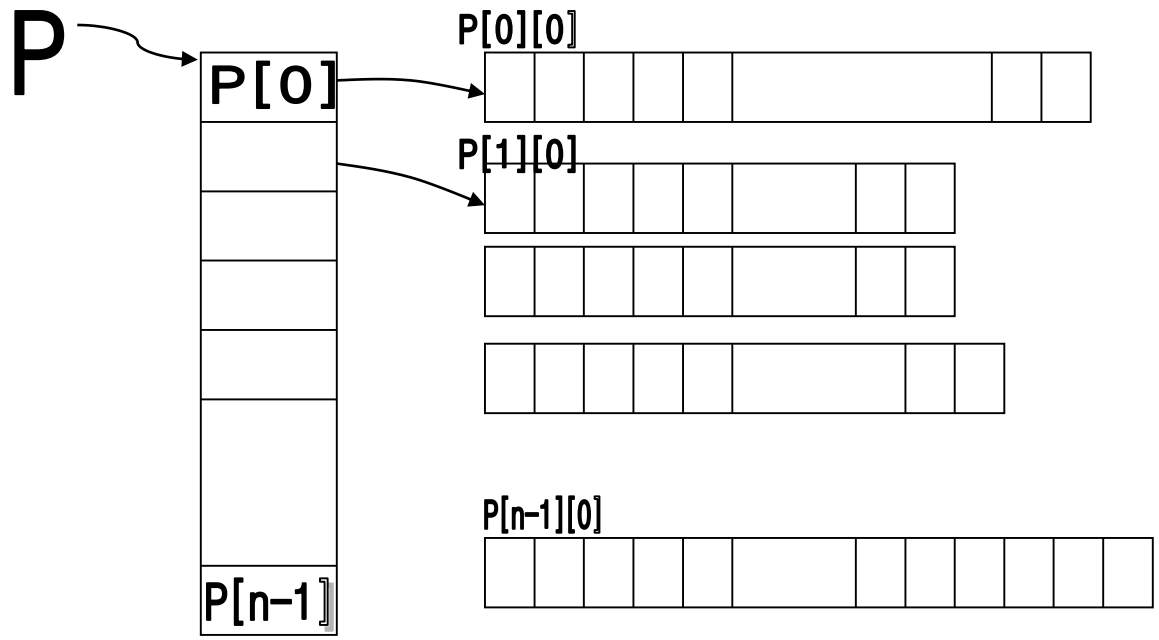
동적 할당 배열

- **2차원 배열 - int x[3][5]**



포인터에 의한 다차원 배열

**P의 경우




```
p = (char **) malloc (n * sizeof(char *));
for (I=0; I<n; I++)
   p[I] = (char *) malloc (m * sizeof(char));
```

도전과제

- 포인터에 의한 다차원 배열을 만들어 반환하는 함수를 작성하시오.

  - ✓ 힌트:
    - … operator
    - 2 차원의 경우: p[n][m]

```
p = (char **) malloc (n * sizeof(char *));
 for (i=0; i<n; i++)
    p[i] = (char *) malloc (m * sizeof(char));
```

구조체

- 서로 다른 형의 변수들을 하나로 묶어 하나의 변수형을 정의
- 예제 – 카드

```
struct card {
   int      pips;
   char     suit;
};
```

  - ✓ struct : 키워드
  - ✓ card : 구조체 태그 이름
  - ✓ pips와 suit : 구조체의 멤버
  - ✓ 이것은 struct card 형의 정의이고 변수의 선언은 아님

※ 구조체와 열거형 (enum)의 형 및 변수의 선언과 형태가 같음

구조체 변수 선언

- struct 형 변수 선언 방법 1

```
struct card {
   int      pips;
   char     suit;
};
struct  card    c1, c2;
```

- struct 형 변수 선언 방법 2

```
struct card {
   int      pips;
   char     suit;
} c1, c2;
```

- struct 형 변수 선언 방법 3

```
struct card {
   int      pips;
   char     suit;
};
typedef struct card card;
card    c1, c2;
```

- struct 형 변수 선언 방법 4

```
typedef struct {
   int      pips;
   char     suit;
} card;
card  c1, c2;
```

- struct 형 변수 선언 방법 5

```
typedef struct card_tag {
   int      pips;
   char     suit;
} card;
card  c1, c2;
```

내재형 구조체에서도 사용가능

```
typedef struct card_tag {
   int      pips;
   char     suit;
   struct card_tag my;
} card;
```

- struct 형 변수 선언 방법 6

```
struct {
   int      pips;
   char     suit;
} c1, c2;
```

태그 이름이 없어, 동일형의 변수 선언 불가
c1, c2는 c3, c4와는 다른 형임
즉, c1 = c2; // O
c1 = c3; // X

```
struct {
   int      pips;
   char     suit;
} c3, c4;
```

구조체 변수 선언

- 한 구조체내에서 멤버 이름은 유일해야 하나, 서로 다른 구조체에서는 같은 멤버 이름을 사용할 수 있음

```
struct fruit {
    char    *name;
    int     calories;
};
struct vegetable {
    char    *name;
    int     calories;
};
struct fruit            a;
struct vegetable    b;
```

구조체의 초기화

- card c = {13, 'h'}; /* the king of hearts */
- complex a[3][3] = {
  {{1.0, -0.1}, {2.0, 0.2}, {3.0, 0.3}},
  {{4.0, -0.4}, {5.0, 0.5}, {6.0, 0.6}},
  }; /* a[2][] is assigned zeroes */
- struct fruit frt = {"plum", 150};
- struct home_address {
  char *street;
  char *city_and_state;
  long zip_code;
  } address = { "87 West Street", "Aspen, Colorado", 80526};
- struct home_address previous_address = {0};

  이경우 포인터멤버들은 NULL로 그 외는 0으로 초기화
- (a[2][2] = a[1][2];) == (a[2][2].real = 6.0; a[2][2].imagine = 0.6)

지정멤버변수의 초기화 – C99 ANSI/ISO 표준

- 특정 멤버변수만 초기화
- 두 가지 방법
  - ✓ 배열인 경우 [순번] = 초기값
    - ✓ **int a[10] = { [0]= 100, [3] = 200};**
  - ✓ 구조체나 공용체인 경우 **.memberVar = value**

```
struct MyStruct_tag {
   int a;
   int b;
   int c;
} Object = { .c = 30, .a = 2 };   // 멤버 변수 b는  초기화 되지 않는다.
```

- 컴파일 옵션

```
$ gcc  –std=c99  –o  prog  prog.c
```

멤버 접근 연산자

- 멤버 접근 연산자: “ . ”

```
c1.pips = 3;
c1.suit = 's';
c2.pips = c1.pips;
c2.suit = c1.suit;
```

- 멤버 접근 연산자: “ ->”
  - ✓ “포인터”를 통해서 구조체 멤버를 접근할 때

```
pointer_to_structure -> member_name
```

  - ✓ 이것은 다음과 같은 형식으로도 쓸 수 있음:

```
(*pointer_to_structure).member_name
```

즉, p->item == (*p).item != *p.item

멤버 접근 연산자

선언문과 배정문

```
Struct student {
    char *last_name;
    int student_id;
    char grade;
};
struct student tmp, *p = &tmp;
tmp.grade = 'A';
tmp.last_name = "Casanova";
tmp.student_id = 910017
```

수식	동등한 수식	개념적 값
tmp.grade	p->grade	A
tmp.last_name	p->last_name	Casanova
(*p).student_id	p->student_id	910017
*p->last_name + 1	(*(p->last_name))+1	D
*(p->last_name + 2)	(p->last_name)[2]	s

함수에서 구조체

- 함수의 인자로써 함수에 전달
- 함수로부터 반환 값

- 함수의 인자로서 구조체가 전달될 때 구조체는 값으로 전달
  - ✓ 즉, 함수의 몸체에서 구조체를 사용할 때 지역 사본을 만듦
  - ✓ 구조체의 멤버가 배열일 때도 마찬가지로 복사함

- 구조체가 많은 멤버를 가지거나, 큰 배열을 멤버로 가질 경우, call by value에 의한 인자 전달은 상대적으로 비효율적

=> 대체로 함수의 인자로 구조체의 주소를 사용
(call by reference)

함수에서 구조체

- 예제

```
typedef struct employee_tag {
    char                    name[25];
    int                     employee_id;
    struct dept             department;
    struct home_address *a_ptr;
    double                  salary;
    ......
} employee_data;
```

- ✓ department는 그 자체가 구조체이고, 컴파일러는 각 멤버의 크기를 미리 알아야 하므로

  struct dept에 대한 선언과

  struct home_address에 대한 struct 형의 선언이 먼저 와야 함

함수에서 구조체

- 함수 선언 방법 1: call by value

```
employee_data update(employee_data e)
{
   .....
   printf("Input the department number: ");
   scanf("%d", &n);
   e.department.dept_no = n;
   .....
   return e;
}
```

✓ 함수 호출

```
employee_data e;
e = update(e);
```

함수에서 구조체

- 함수 선언 방법 2: call by reference

```
void update(employee_data *p)
{
   .....
   printf("Input the department number: ");
   scanf("%d", &n);
   p->department.dept_no = n;
   .....
}
```

✓ 함수 호출

```
employee_data e;
update(&e);
```

구조체(structure)

- Nested Structure

```
typedef struct person_tag {
        char name[30];
        char addr[50];
        char tel[15];
} PERSON;

typedef struct sungjuk {
        int kor, eng, math, total;
        float ave;
} STUDENT;

typedef struct one_tag{
        STUDENT a;
        PERSON b;
} ONE;
```

배열과 포인터

- 배열포인터 (array pointer)

```
int a[2][3] = {2,3,5,7,11,13}, (*p)[][3];
p = &a;
printf (“%d¥n”,  (*p)[1][2]); // 출력13
```

- 포인터 배열 (an array of pointers)

```
int *w[3];
w[0] = malloc (sizeof(int));
w[1] = malloc (sizeof(int) * 2);
w[2] = malloc (sizeof(int) );
```

함수와 포인터

- 함수 포인터

```
#include <stdio.h>
//test1 선언(인수 1개, float f)
int (*test1 (float))(const char *,...);
//test2 (char c);
int (*(*test2(char))(float))(const char *,...);
int main()
{
    int (*fp)(const char *, ...)=0;
    //int prinf(const char *, ...) = 0; //printf 문 타입
   //fp1 선언
    //int (*test1 (float))(const char *,...);
    int (*(*fp1)(float))(const char *,...)=0;
   //fp2 선언
    //int (*(*test2(char c))(float f))(const char *,...)
    int (*(*(*fp2)(char))(float))(const char *,...) = 0;

    fp2 = test2; // 괄호가 없는 것은 주소값을 넘겨주기 때문이다.
    fp1 = fp2('a');반환값이 test1이므로 test1로 선언된다.
    fp  = fp1(0.1);

    fp("Hi....₩n");
    return 0;
}
int (*test1(float f))(const char *,...)
{
  printf("float : %0.1f₩n", f);
  return printf;
}
int (*(*test2(char c))(float))(const char *,...)
{
    printf("char : %c₩n", c);
    return test1;

}
```

- *f() ! = (*f)()

명령행 인자[1]

- 명령행 : 사용자가 명령을 입력하는 행
  - ✓ 명령행 인자 : 명령을 입력할 때 함께 지정한 인자(옵션, 옵션인자, 명령인자 등)

```
int main(int argc, char *argv[])
```

[예제 1-6] 명령행 인자 출력하기 ex1_6.c

```
01  #include <stdio.h>
02
03  int main(int argc, char *argv[]) {
04      int n;
05
06      printf("argc = %d\n", argc);
07      for (n = 0; n < argc; n++)
08          printf("argv[%d] = %s\n", n, argv[n]);
09
10      return 0;
11  }
```

```
# ex1_6.out -h 100
argc = 3
argv[0] = ex1_6.out
argv[1] = -h
argv[2] = 100
```

명령행 인자[2]

- 옵션 처리함수: getopt(3)

```
#include <stdio.h>
int getopt(int argc, char * const argv[], const  char  *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```

  - argc, argv: main 함수에서 받은 것 그대로 지정
  - optstring: 사용 가능 옵션 문자, 옵션에 인자가 있을 경우 문자 뒤에 ':'추가
  - optarg: 옵션의 인자 저장
  - optind: 다음에 처리할 argv의 주소
  - optopt: 오류를 발생시킨 문자
  - opterr: 오류 메시지를 출력하지 않으려면 0으로 지정

- 유닉스 명령의 옵션 규칙
  - ✓ 옵션의 이름은 한 글자여야 하며, 모든 옵션의 앞에는 하이픈(-)
  - ✓ 인자가 없는 옵션은 하나의 - 다음에 묶여서 올 수 있다(-xvf)
  - ✓ 옵션의 첫 번째 인자는 공백이나 탭으로 띄고 입력한다.
  - ✓ 인자가 있어야 하는 옵션에서 인자를 생략할 수 없다.
  - ✓ 명령행에서 모든 옵션은 명령의 인자보다 앞에 와야 한다.
  - ✓ 옵션의 끝을 나타내기 위해 --를 사용할 수 있다.

명령행 인자[3]

❑ 옵션 처리

[예제 1-7] getopt 함수 사용하기 ex1_7.c

```
01  #include <stdio.h>
02
03  int main(int argc, char *argv[]) {
04      int n;
05      extern char *optarg;
06      extern int optind;
07
08      printf("Current Optind : %d\n", optind);
09      while ((n = getopt(argc, argv, "abc:")) != -1) {
10          switch (n) {
11              case 'a':
12                  printf("Option : a\n");
13                  break;
14              case 'b':
15                  printf("Option : b\n");
16                  break;
17              case 'c':
18                  printf("Option : c, Argument=%s\n",
                            optarg);
19                  break;
20          }
21          printf("Next Optind : %d\n", optind);
22      }
23
24      return 0;
25  }
```

```
# ex1_7.out
Current Optind : 1
# ex1_7.out -a
Current Optind : 1
Option : a
Next Optind : 2
# ex1_7.out -c
Current Optind : 1
ex1_7.out:   option   requires   an
argument -- c
Next Optind : 2
# ex1_7.out -c name
Current Optind : 1
Option : c, Argument=name
Next Optind : 3
# ex1_7.out -x
Current Optind : 1
ex1_7.out: illegal option -- x
Next Optind : 2
```

연산자의 우선 순위와 결합 법칙

괄 > 멤 > 단 > 산 > 시 > 관 > 비 > 논 > 삼 > 대 > 콤

연산자	결합 법칙
() [] . ->	좌에서 우로
++ (후위) -- (후위) ++ (전위) -- (전위) ! ~ sizeof (형) +(단항) -(단항) &(주소) *(역참조)	우에서 좌로
* / %	좌에서 우로
+ -	좌에서 우로
<< >>	좌에서 우로
< <= > >=	좌에서 우로
== !=	좌에서 우로
&	좌에서 우로
^	좌에서 우로
|	좌에서 우로
&&	좌에서 우로
||	좌에서 우로
?:	좌에서 우로
= += -= *= /= %= <<= >>= &= ^= |=	우에서 좌로
, (콤마 연산자)	좌에서 우로

포인터와 함수 그리고 구조체

```
#include <stdio.h>

typedef struct student_t {
   char name[32];
   int id;
} student;

student *get_data (int *, int *);          // function prototype: 함수 원형에 의한 선언
void display_data (student *, int);        //

int main()
{
   int a, b=30, *ap=NULL, *pt=ap, *bp=&b;
   student *student=NULL;

   ap = &a;
   *ap = 10;

   student = get_data (&a, &b);          // function call: call by reference and pointer return
   printf ("%d, %d, 이름 %s", a, b, student[0].name);
   display_data (student, 2);

   return 0;
}
```

포인터와 함수 그리고 구조체

```
student *get_data (int *x, int *y)    // function definition
{
   student *st=NULL;                  // static 변수나, 동적 메모리 할당용 포인터만 가능

   st = (student *) malloc(sizeof(student) * 2); // 배열 크기 2인 배열 공간 확보

   strcpy (st[0].name, "도깨비");
   st[0].id = 20142005113;   //st[0].id == (st + 1)->id == (*(st+1)).id

   strcpy (st[1].name, "어벙이");
   st[1].id = 20132005234;

   printf ("Function called by reference: %d and %d n", ++(*x), (*y)++);

   return st;                         // static 변수나 동적 메모리 할당된 포인터만 반환 가능
}

void display_data (student *student, int size)
{
   int i;

   for (i=0; i<size; i++)  printf ("학번: %d, 이름 %s",  student[i].id, (student+i)->name);
}
```

실습:

- 다음은 대학교에서 학생 자료를 전산화하기 위해 필요한 자료들이다.
  - 이름
  - 학번
  - 성적
    - 국어
    - 프로그래밍

```
typedef struct stu_tag {
   int id
   char name[32];
   int kor, prog;
} student;
```

```
typedef struct dept_tag {
   student *freshmen;
   int nfreshmen;
   student *sophomore;
   int nsophomore;
   student *junior;
   int njunior;
   student *senior;
   int nsenior;
} dept;
```

1. 알맞은 자료형을 선언
2. 입력함수: dept *get_data ()
   - 학생 수 입력에 따라 변수 선언 (동적 메모리 할당)
   - 키보드에서 입력 받음
   - 자료 반환
3. 출력함수: void display_data (dept *)

c99 표준 v.s. c89

- C++처럼 변수 선언이 사용 바로 전으로 표현 가능
- gcc 3.x 버전 이후부터 가능
- 반드시 gcc에 -std=c99 옵션 추가
- ANSI C99 표준 소스코드 예

```
// gcc –std=c99 –o prog prog.c

#include <stdio.h>

int main ()
{
   printf (“Hello, C99\n”);

   int a = 9;
   for (int i=0; i<10; i++) {
      printf (“i = %d\n”, i);
   }

   int array[3]={a*2, a-1, a/2};
   for (int i=0; i<3; i++) printf (“array[%d] = %d\n”, i, array[i]);

   return 0;
}
```

How to enable c11 on later versions of gcc?

- `-std=c11` is the correct option
- not available in `gcc 4.6`.
- at least `gcc 4.7`
- In some older versions like `gcc 4.6`,
  - the option `-std=c1x` was available with very limited support

```
$ gcc -std=c11 -o test test.c
```