오픈소스 소프트웨어 실습

Open-Source Software Lab

#5

실습 담당 조교 연락처

◆실습 조교 : 정만성

◆연구실 : 학연산클러스터 604호

◆과제 제출 및 문의 - 이메일

◆이메일 : wjdakstjd@hanyang.ac.kr

Compile

- Pre-processing :

  #include, #define, #if와 같은 지시자를 처리하는 과정

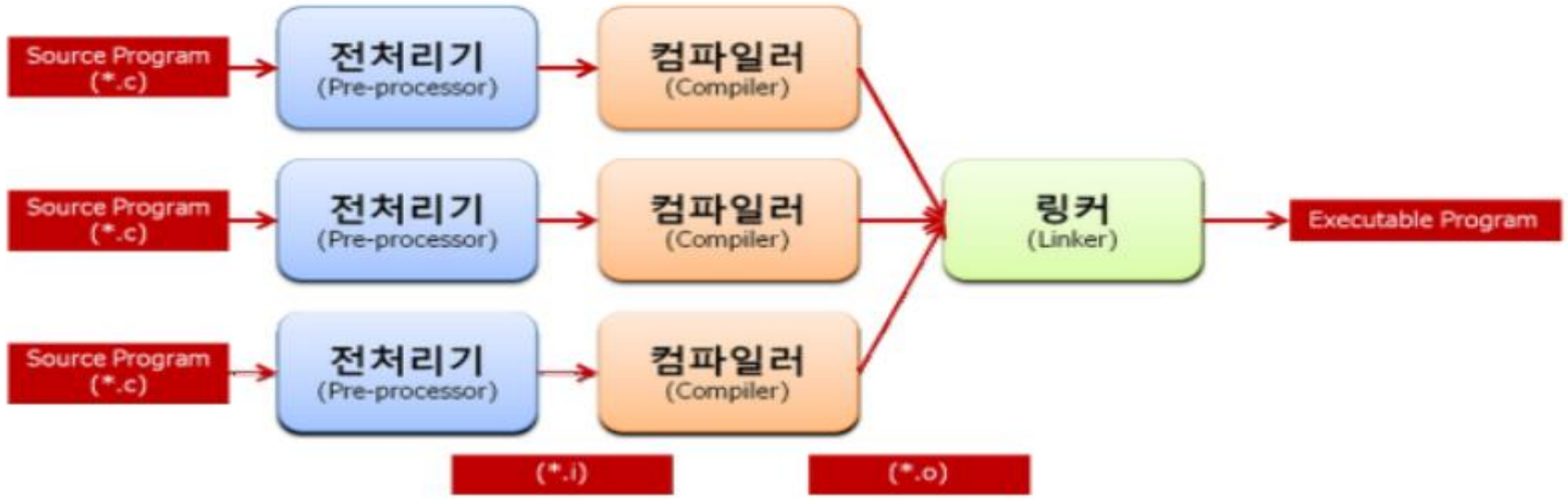
- Compile :

  어셈블리 코드로 변환하는 과정

- Assemble

  바이너리 파일로 변환하는 과정

- Linking



Compiler

- Compile 작업을 실행하는 소프트웨어
- C lang을 CPU가 이해할 수 있는 기계어로 변환
- 가장 대표적인 소프트웨어 : gcc

Starting from 0

- C의 자연스러운 시작점
- 숫자를 0부터 셈 (배열의 첫 원소)
- 0은 False, 0아닌 수는 True
- 문자열의 끝, 포인터의 NULL
- 외부/정적 변수의 초기화 값

Dissection of sea Program

```c
#include <stdio.h>


int main(void) {
    printf("from sea to shining C\n");
    return 0;
}
```

◆ C 시스템 : C 언어, 전처리기, 라이브러리, 편집기, 컴파일러

Characters and Lexical Elements & Syntax

- 문자와 어휘 원소
  - 구문 : 올바른 프로그램을 만들 수 있게 하는 규칙
  - 컴파일러는 구문을 검사

- ANSI C 토큰
  - 키워드, 식별자, 상수, 문자열 상수, 연산자, 구두점

Keywords

- C언어에서 고유한 의미를 가지는 토큰(문법단위)로 예약어

auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	extern	register	switch	
continue	float	return	typedef	
default	for	short	union	

Identifiers

- 문자, 숫자, 밑줄문자(_)로 구성된 토큰, 대소문자를 구별
- 식별자의 첫 번째 문자 : 문자, 밑줄문자(_)
- 밑줄문자(_)로 시작되는 식별자는 가급적 사용 X

Constants (0)

- C의 상수 : 수치 상수, 문자 상수, 문자열 상수, 열거 상수
  - 수치 상수 : 정수 / 실수 (n진수, 지수)
    - > 8진수 : 0을 맨앞에 붙임 (011, 0345)
    - > 16진수 : 0x 또는 0X를 맨 앞에 붙임 (0xab, 0x2bcd)

Constants (1)

- 문자 상수 : 한 쌍의 작은 따옴표 (' ')에 묶인 문자

  > c = 'A';

Name of character	Written in C	Integer value
'a'	a	97
'A'	A	65
'0'	0	48
alert	\a	7
backslash	\\	92
backspace	\b	8
carriage return	\r	13
double quote	\"	34
formfeed	\f	12
horizontal tab	\t	9
newline	\n	10
null character	\0	0
single quote	\'	39
vertical tab	\v	11
question mark	\?	63

Constants (2)

- 문자열 상수 : 한 쌍의 큰 따옴표 (" ")에 묶인 문자열
  > ASCII 코드
  > 확장 문자열 : ₩n, ₩t, ₩0
  > C 언어는 개행이 없음!, 개행 ₩n을 직접해야함
- 열거 상수 : enum에 의해 선언된 상수

Operators (0)

- 연산자와 피연산자
  - Operator : 연산을 수행하는 기호
  - Operand : 연산에 포함되는 변수나 상수
- 연산자의 종류
  - 배정 연산자 (assignment), 산술 연산자, 관계/등가 연산자
  - 논리 연산자, 증감 연산자, sizeof 연산자
  - 조건부 연산자, 비트 연산자

Operators (1)

- 연산자 우선순위

'() 함수호출' '[] 배열첨자' '. 멤버' '-> 포인터멤버' ++(post) --(post)	
++(pre) --(pre) ! ~ sizeof +(단항) -(단항) '& 주소' '* 역참조' '(datatype) 캐스팅'	R→L
* / %	
+ -	
<< >>	
< <= > >=	
== !=	
&	
^	
|	
&&	
||	
? :	R→L
= += -= *= /= %= <<= >>= &= ^= |=	R→L
, (콤마 연산자)	

Declarations, Expressions, Assignment

- 선언
  - 모든 변수는 사용 전에 선언
  - 변수를 선언 할 때는 자료형과 변수명을 입력
- 수식
  - 상수, 변수, 연산자, 함수 호출 등의 의미있는 결합
  - 상수, 변수, 함수 호출은 그 자체가 수식

Declarations, Expressions, Assignment

- 배정
  - 문장 : 수식 뒤에 세미콜론이 오면, 수식은 문장이 됨
    > i = 7 [배정 수식]
    > i = 7; [문장]
  - 원하는 경우 초기 값을 적용할 수 있음
  - 초기화되지 않은 변수는 쓰레기 값이 들어감
  - 정적 변수로 선언된 것은 기본적으로 0으로 값이 초기화

The Fundamental Data Types (byte)

- Integral types
  : char 1 / signed char 1 / unsigned char 1
  : short 2 / int 4 / long 8 / (long long) 8
  : unsigned short 2 / unsigned 4 / unsinged long 8
- Floating types
  : float 4 (유효숫자6) / double 8 (유효숫자15) / long double 16
  : 3.14F / 3.14L
- Boolean types <stdbool.h>
  : bool [true / false]

Conversions and Casts

- 일반적 자동 변환 (The usual arithmetic conversions)
  - 수식에서 제일 큰 형으로 변환
- 캐스트(casts) : 명시적 변환
  - (double) i
  - (long) ('A' + 1.0)
  - f = (float) ((int)d + 1)
  - d = (double) i / 3

Standard Input and Output (0)

```c
#include <stdio.h>

int main(void) {
    int a;
    scanf("%d", &a);
    printf("number a : %d\n", a);
    return 0;
}
```

Standard Input and Output (1)

- scanf() :C언어에서 특정한 변수에 값을 넣기 위해서 사용
  - &는 변수의 주소를 의미
  - 실제로 컴퓨터는 특정한 메모리 주소에 접근하여 데이터를 수정

c	문자 char	f	float
d / i	10진 정수 int	lf	double
ld / lld	10진 정수 long (long)	LF	long double
u	10진 정수	s	공백 없는 문자열
o	8진 정수	p	
x / X	16진 정수	n	
e E g G	실수형	[...]	스캔 집합

Standard Input and Output (2)

- printf()

c	문자 char	E	지수형
d / i	10진 정수 int	f	실수형
ld / lld	10진 정수 long (long)	g	e와 f중 짧은 쪽
u	부호 없는 10진 정수	G	E와 f중 짧은 쪽
o	부호 없는 8진 정수	s	문자열
x / X	부호 없는 16진 정수	p	void 포인터
e	지수형	n	정수 포인터

Standard Input and Output (3)

- 변환문자의 옵션 지정
  - %[필드폭].[자리수][변환문자]

```c
#include <stdio.h>

int main(void) {
    double a;
    scanf("%lf", &a);
    printf("%.2f\n", a);
    return 0;
}
```

변환문자의 옵션 지정		
%[필드폭].[자릿수][변환문자]		
%d	→	123
%5d	→	␣␣123
%10d	→	␣␣␣␣␣␣␣123
%2d	→	123 (지정 필드폭의 칸수가 필요한 자릿수보다 작아도 필요한 숫자는 모두 출력)
%f	→	654.321000 (표준출력, 소수점 이하 6자리)
%12f	→	␣␣654.321000 (12칸에 출력, 소수점 이하는 6자리로 표준출력)
%9.2f	→	␣␣␣654.32 (9칸에 출력, 소수점 이하는 2자리로 출력)

Conditional

- if-else if-else

```c
#include <stdio.h>

int main(void) {
    int n;

    printf("몇 명? : ");
    scanf("%d", &n);

    if ( n==1 || n==2 ) {
        printf("1~2\n");
    } else if ( n==3 || n==4) {
        printf("3~4\n");
    } else {
        printf("a lot of people\n");
    }
    return 0;
}
```

Conditional

- switch

```c
#include <stdio.h>

int main(void) {
    int n;
    scanf("%d", &n);

    switch(n) {
        case (12):
        case (1):
        case (2):
            printf("겨울\n");
            break;
        case (3):
        case (4):
        case (5):
            printf("봄\n");
            break;
        case (6):
        case (7):
        case (8):
            printf("여름\n");
            break;
        case (9):
        case (10):
        case (11):
            printf("가을\n");
            break;
        default:
            printf("error\n");
            break;
    }

    return 0;
}
```

Iteration

- for/while
  - 반복문을 탈출하고자 하는 위치에 break문 삽입
  - for문과 while문은 상호 치환이 가능
  - 의도하지 않은 무한 루프가 발생하지 않도록 유의

```c
#include <stdio.h>

int main(void) {
    // ( 초기화; 조건; 조건 명령어 )
    for ( int i=0; i<10; i++ ) {
        printf("%d ", i);
    }

    printf("\n");

    int i=0; // 초기화
    while ( i<10 ) { // 조건
        printf("%d ", i);
        i++; // 조건 명령어
    }

    printf("\n");

    return 0;
}
```

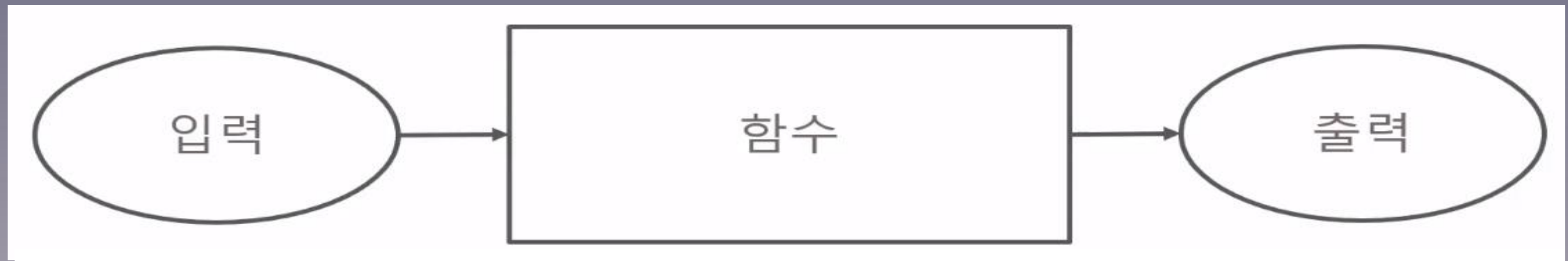
Iteration

- do-while

```
do {
        sum += i;
        scanf("%d", &i);
} while (i > 0);
```

- break/continue
  - break와 countinue는 정상적인 제어의 흐름을 제어
    - > break : 루프의 내무나 switch문으로부터 빠져나옴
    - > continue : 루프의 현재 반복을 멈추고 즉시 다음 반복

Functions (0)




```
type function_name ( parameter list ) { declarations statements }
```

function head

function body

Functions (1)

```
반환자료형 함수 이름 (매개변수) {
        몸체
        return 반환 값
}
```

- 반환자료형과 반환 값의 Data Type은 일치해야 함
- 반환자료형이 없거나 매개변수가 필요 없는 경우 : void 형
- return문에 도달하면 해당 함수 종료

  - return; or return expression;

Functions (2)

- 함수 원형 (function prototype)
  - 컴파일러에게 인자 type, 인자 수, return 값의 형을 알림
  - type function function_name(parameter type list);
- 재귀 함수
  - 함수나 알고리즘이 수행 도중에 자신을 다시 호출

Functions (3)

```c
#include <stdio.h>

int add(int, int);
void shining(void);

int main(void) {
    int a, b;
    scanf("%d %d", &a, &b);
    printf("a + b = %d\n", add(a, b));
    shining();
    return 0;
}

int add(int x, int y) {
    return (x+y);
}

void shining(void) {
    printf("from sea to shining C!\n");
}
```

```c
#include <stdio.h>

int fac(int);

int main(void) {
    int n;
    scanf("%d", &n);
    printf("%d\n", fac(n));
    return 0;
}

int fac(int n) {
    if ( n==0 ) {
        return 1;
    } else {
        return n * fac(n-1);
    }
}
```

Functions (4)

◆ main() 함수의 인자

```
void main(int argc, char *argv[]) {
int i;
printf("argc = %d\n", argc);
for (i = 0; i < argc; ++i)
printf("argv[%d] = %s\n", i, argv[i]);
}
```

→ argc : 명령어 라인 인자의 개수를 가짐
→ argv : 명령어 라인을 구성하는 문자열들을 가짐

Functions (5)

- 앞의 프로그램을 컴파일하여 my_echo로 한 후, 다음 명령으로 실행

```
$ my_echo a is for apple
```

- 출력:

```
argc = 5
argv[0] = my_echo
argv[1] = a
argv[2] = is
argv[3] = for
argv[4] = apple
```

Scope Rule (0)

- 유효범위의 규칙
  - 기본적인 유효범위 규칙은 변수가 선언된 블록 안에서만 유효
  - 외부 블록의 변수는 내부에서 다시 정의하지 않는한 여전히 유효

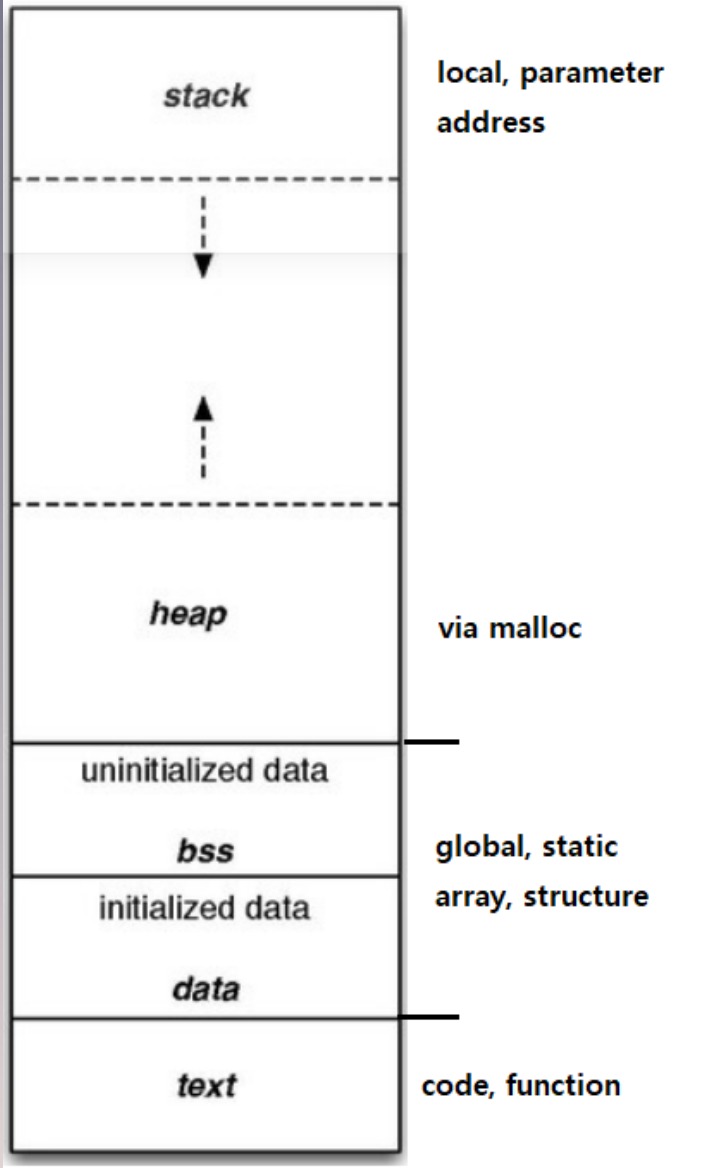
```c
#include <stdio.h>

int a = 33;

int main(void) {
    // int a = 3;
    int b = 12;
    printf("%d %d\n", a, b);
    return 0;
}
```

```
Wed 25 Sep - 12:43
@p1n9u ./test
3 12

Wed 25 Sep - 12:44
@p1n9u ./test
33 12
```

Scope Rule (1)

```c
#include <stdio.h>

int main(void) {
    int a=1, b=2, c=3;
    printf("%d %d %d\n", a, b, c); // 1 2 3
    {
        int b=4;
        float c=4.0;
        printf("%d %d %f\n", ++a, b, c); // 2 4 4.000000
        printf("%d %d %f\n", a++, b, c); // 2 4 4.000000
        a = b;
        {
            int a, c;
            a = c = b;
            printf("%d %d %d\n", a, b, c); /// 4 4 4
        }
        printf("%d %d %f\n", a++, b, c); // 4 4 4.000000
    }
    printf("%d %d %d\n", a, b, c); // 5 2 3
    return 0;
}
```

Variables



◆ 함수의 매개 변수가 처리될 때

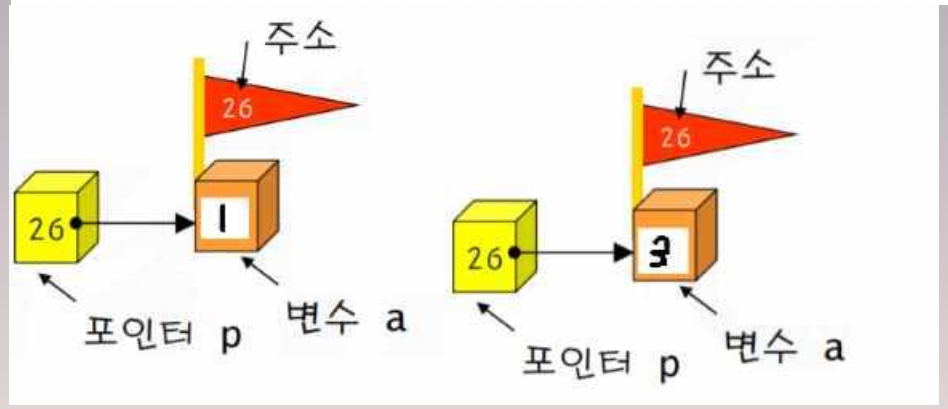
1. call by value : 단지 값만 전달하므로 함수 내에서 변수가 새롭게 생성되며 재정의
2. call by reference : 주소를 참조하므로 원래 변수 자체에 접근 가능

Pointer (0)

- 변수로서 지정된 데이터 타입의 메모리 주소를 저장
- 포인터가 아무것도 가리키고 있지 않을 때는 NULL로 설정
- 포인터는 연산을 통해 자료형의 크기만큼 이동

```
int a = 1;
int *p;
p = &a;
*p = 3;
```

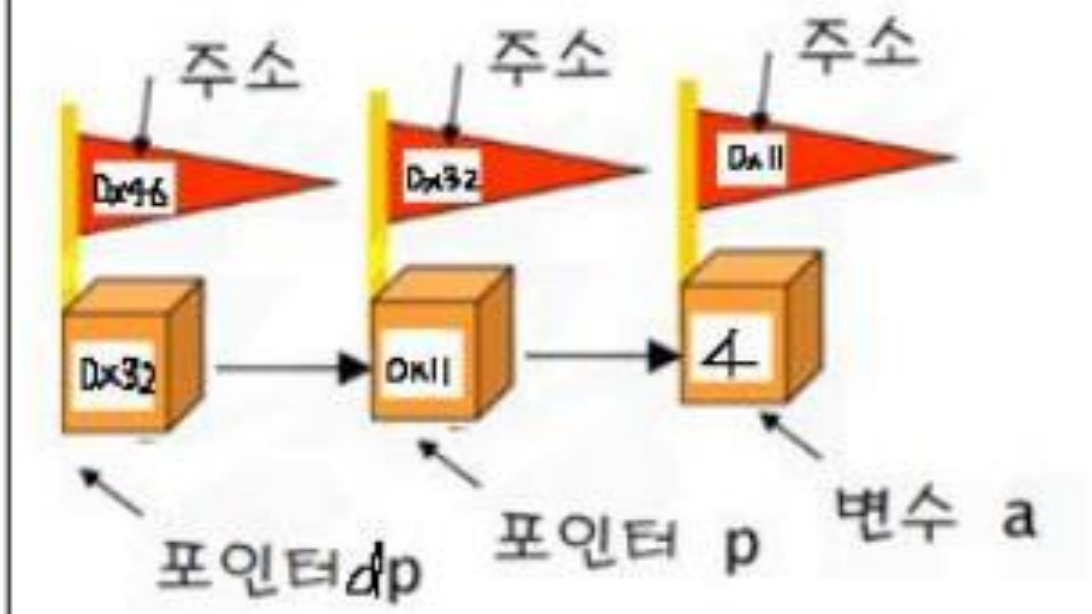
```
int a = 1;
int *p = &a; [ & 주소연산자 ]
*p = 3;
[ 포인터연산자 " * " vs 참조연산자 " * " ]
```



Pointer (1)

- 다중포인터
  - int **p (?)
  - int *(*p) : 인트형 포인터의 포인터, 인트형 포인터의 주소

```
int a; int *p; int **dp;
a = 4; p = &a; dp = &p;
```



Pointer (2)

- 추가학습
  1. https://www.youtube.com/watch?v=dgYOGw9-JLo
  2. https://www.youtube.com/watch?v=HJs0xsTPEHw
  3. https://www.youtube.com/watch?v=CqqBPL6Hvyk
  4. OSW_실습05_pointer

위의 순서대로 진행

Array (0)

- 동일한 데이터 타입의 데이터를 여러개 만드는 경우에 사용
- 반복 코드 등에서 배열을 사용하면 효율적
  - 자료에 순차적으로 접근 가능
- 선언과 초기화
  - type name[size] = { init value };
  - index : 0 ~ (size-1)

Array (1)

```
int a[4]; : 컴파일러 마다 다름 / 0으로 초기화되거나 쓰레기 값으로 초기화됨
int a[4] = { 0, }; : 전부 0으로 초기화됨
int a[4] = { 1, 2, 3, 4 };
int a[4] = { 1, 2 }; : [ 1, 2, 0, 0 ] 으로 초기화
int a[] = { 1, 2 }; : SIZE가 2인 배열이 됨
```

```
char s[] = "abc"; <-> [ 'a', 'b', 'c', '\0' ]
  : 문자배열은 항상 끝에 끝을 알리는 '\0'이 있어 size가 +1이 됨
  ( printf의 종료조건? : '\0' 을 만났을 때 ! )
```

- 배열의 이름은 고정/상수 포인터
  - 배열의 이름은 배열의 [0] index 변수의 주소를 가리킴
  - a[0] : *a / a[1] : *(a+1) / +1 : 4bytes(int 자료형의 크기) 이동

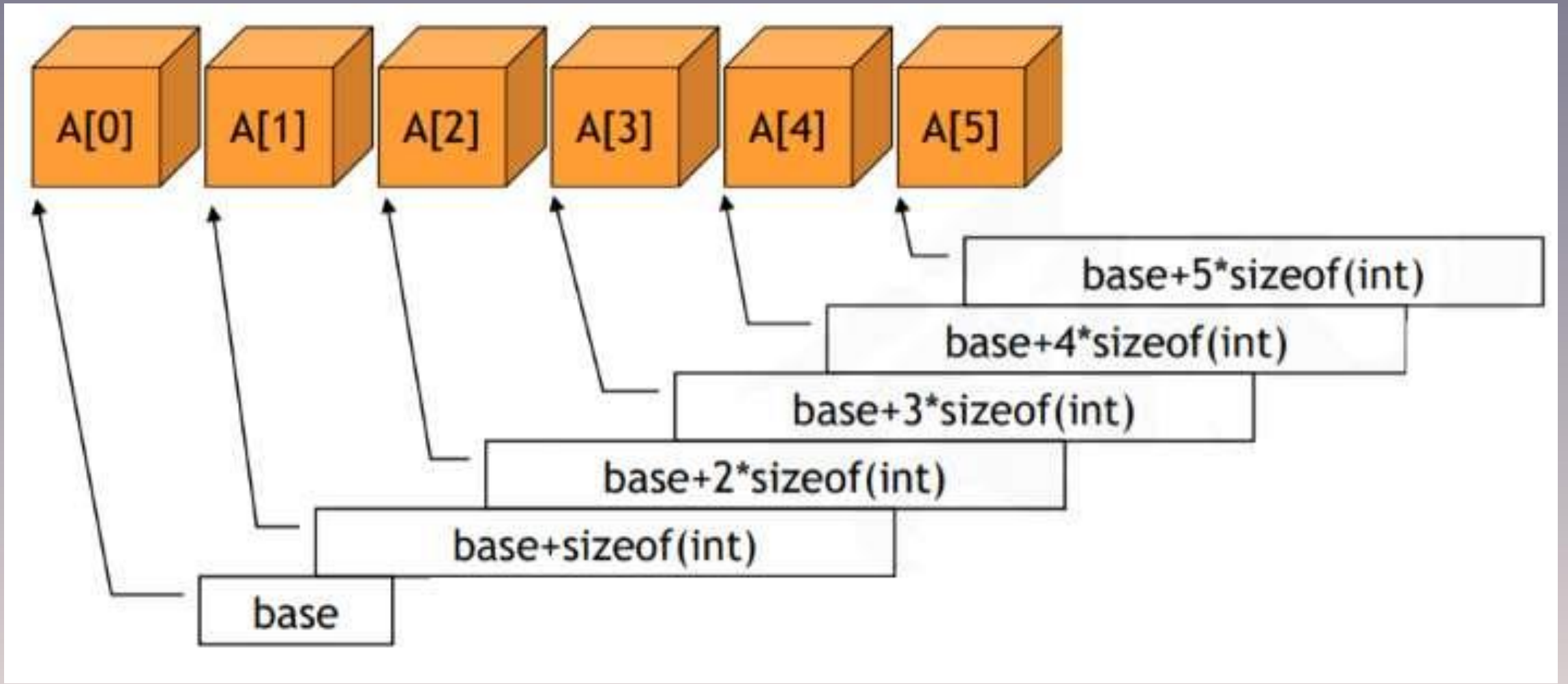
Array (2)

- 다차원 배열
  - type name[row][column] = { {...}, ... , {...} };
  - type name[layer][row][column] = { { {...}, ..., {...} }. .... };
  - 기억장소 사상 함수

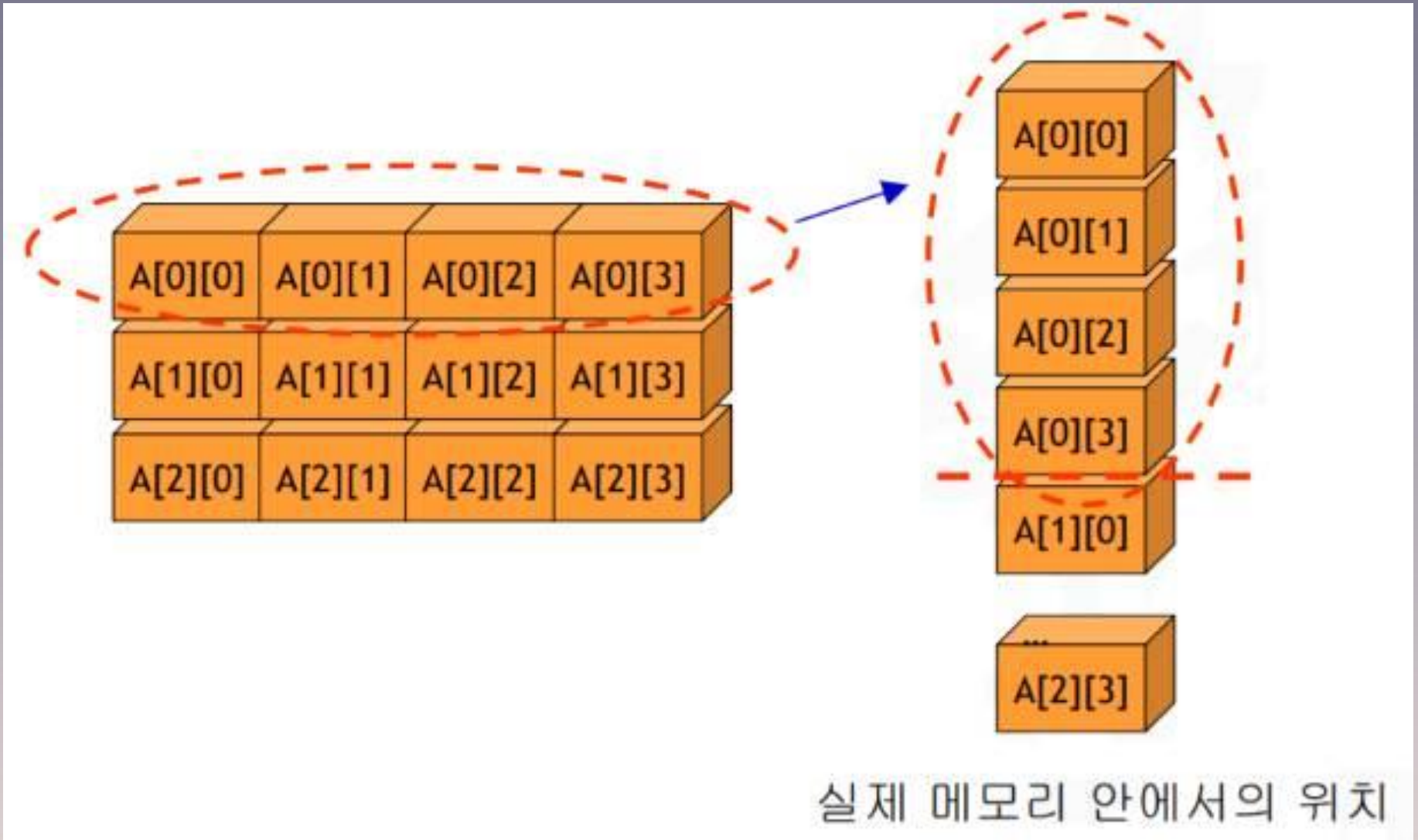
```
int a[3][5];
→ 배열 a의 a[i][j]에 대한 기억장소 사상 함수
*(&a[0][0] + 5 * i + j)
int a[][5] : j값은 명시가 되어야함
```

```
int a[7][9][2];
→ a[i][j][k]를 위한 기억장소 사상 함수:
*(&a[0][0][0] + 9 * 2 * i + 2 * j + k)
int a[][9][2] : j, k값은 명시가 되어야함
```

Array (3)


A[0]
A[1]
A[2]
A[3]
A[4]
A[5]
base+5*sizeof(int)
base+4*sizeof(int)
base+3*sizeof(int)
base+2*sizeof(int)
base+sizeof(int)
base

Array (4)


A[0][0]
A[0][1]
A[0][2]
A[0][3]
A[1][0]
A[1][1]
A[1][2]
A[1][3]
A[2][0]
A[2][1]
A[2][2]
A[2][3]
A[0][0]
A[0][1]
A[0][2]
A[0][3]
A[1][0]
...
A[2][3]
실제 메모리 안에서의 위치

String (0)

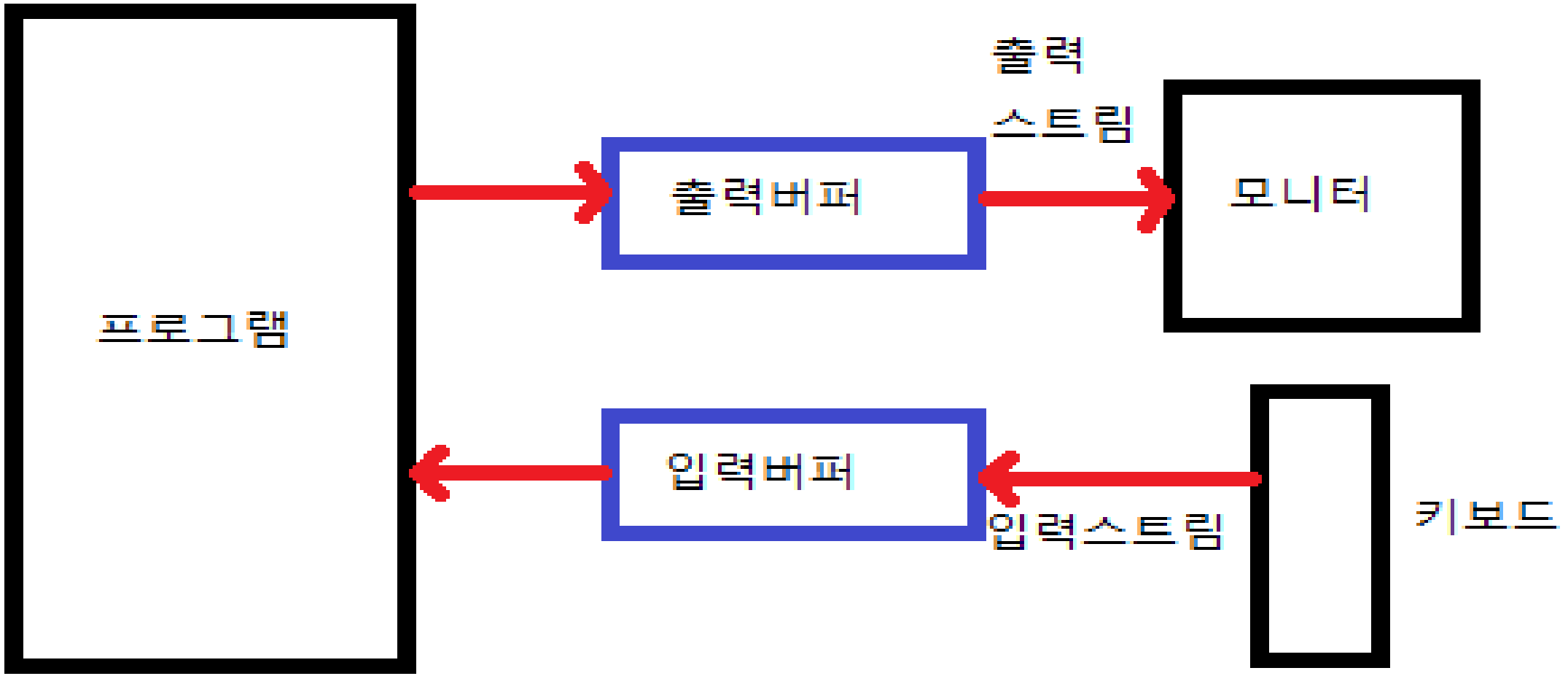
- 스트림
  - 프로그램 상에서 입출력 장치를 대상으로 데이터와 연결시켜주는 가상의 다리
  - 키보드/모니터 입출력을 위한 스트림은 자동으로 생성
  - 파일 입출력을 위한 스트림은 직접 생성해야함

stdin	표준 입력 스트림	키보드 대상으로 입력
stdout	표준 출력 스트림	모니터 대상으로 출력
stderr	표준 에러 스트림	모니터 대상으로 출력

String (1)

- 버퍼
  - 문자열을 처리할 때 버퍼의 개념이 많이 사용됨
  - C 프로그램은 기본적으로 사용자가 의도하지 않아도 자동으로 버퍼를 이용해 입출력

String (2)



String (3)

- 표준 입출력 함수를 통해 입출력을 하는 경우,
  해당 데이터들은 운영체제가 제공하는 메모리 버퍼를
  중간에 통과하게 된다.
- 메모리 버퍼 : 데이터를 임시로 모아두는 공간
- 버퍼를 왜 사용하는가? : 데이터 전송의 효율성 때문
  > 택배를 하나하나 배달하는 것보다 모아서 배송하는 이유와 같음

String (4)

◆ 문자

- C 프로그램의 문자는 ASCII code를 따름
- 아스키 코드는 0~127중의 1바이트로 구성되며 주요 문자를 출력
- 문자 입출력에서 형식 지정자로 %c를 씀

문자와 문자열을 구분

'a' : char 'a'

"a" : char* { 'a', '₩0' }

String (5)

- 문자열
  - char형의 1차원 배열
  - C 언어는 기본적으로 자체적인 문자열 자료형을 제공하지 않음
  - 문자열은 끝의 기호인 '₩0'로 끝남 (printf의 종료조건)
  - 문자열의 크기는 '₩0'까지 포함한 크기
  - %s를 형식지정자로 이용
  - 컴파일러는 문자열 상수를 배열 이름과 같이 포인터로 취급

String (6)

- “abc”

  - “abc”[0] or *(“abc”) : ‘a’
  - “abc”[1] or *(“abc”+1) : ‘b’
  - “abc”[2] or *(“abc”+2) : ‘c’
  - “abc”[3] or *(“abc”+3) : ‘₩0’

  “abc”[1] = ‘d’; -> assignment, 변경 불가능

String (7)

- 배열과 포인터의 차이

  - char *s = “abcde”; -> 포인터 변수 s

  - char s[] = “abcde”; -> char형 배열, 배열의 이름 s (포인터 상수)

  // char s[] = { ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘₩0’ };

String (8)

◆ 문자열함수 : <string.h>

- strstr()의 경우 실제로는 주소를 반환

- <string.h>

  strlen(s) : '₩0'을 뺀 문자의 개수를 리턴

  strcmp(s1, s2) : 사전순 비교 s1이 작으면 음수, 크면 양수, 같으면 0을 리턴

  strcpy(dest, src) : s2의 문자를 ₩0이 나올 때까지 s1에 복사

  strcat(dest, src) : 두 문자열 s1, s2를 결합하고, 결과는 s1에 저장

  strstr(dest, src) : dest에서 src를 찾아 그 이후 내용을 출력

The Use of typedef

- 식별자를 특정한 형과 연관
  - C 언어의 예약어로 다른 자료형의 별병을 만들기 위해 사용
  - enum, struct, union 타입으로 이루어진 긴 선언문을 축약해 쓸 수 있음
  - example

    typedef 자료 별명;

    typedef char uppercase; -> uppercase u;

    typedef int INCHES, FEET; -> INCHES length, width;

Structures (0)

◆ 구조체

- 서로 다른 형의 자료형을 하나로 묶어주는 방법

```
typedef struct 구조체명 {
        데이터 1;
        데이터 2;
          ....
} 구조체 별명;
```

```
typedef struct person {
        char name[10];
        int age;
        double height;
} person;
```

```
typedef struct ListNode {
        element data[N];
        struct ListNode* link;
} ListNode;
```

[자기 참조 구조체]

Structures (1)

◆ 구조체 선언

```c
// 1
struct account {
    int num;
    char *name;
};
struct account a1, a2;

// 2
struct fruit {
    int num;
    char *name;
} apple, grape;
struct fruit peach;
```

```c
// 3
struct {
    int age;
    char *job;
} kim, lee;

// 4
struct {
    int age;
    char *job;
} noh, choi;
```

```c
// 5
struct family {
    int age;
    char *name;
};
typedef struct family family;
family f1, f2;

// 6
typedef struct card {
    int pips;
    char *suit;
} card;
card c1, c2;

// 7
typedef struct {
    int value;
    char *name;
} item;
item i1, i2;
```

Structures (2)

- 1과 2는 가장 기본적인 선언 방법
- 3과 4는 선언 후 새로 할당이 불가능 또한 같아보여도 다른 형임
- 5는 6처럼 한번에 줄여 쓸 수 있음
- 일반적으로 6, 7의 방법을 가장 많이 이용함

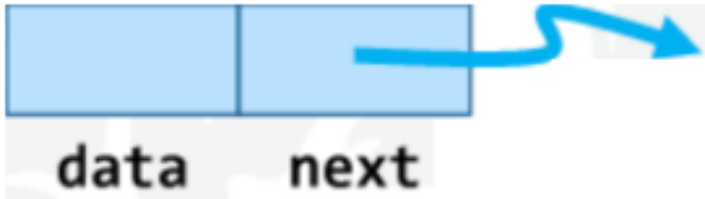
```c
// 5
struct family {
    int age;
    char *name;
};
typedef struct family family;
family f1, f2;

// 6
typedef struct card {
    int pips;
    char *suit;
} card;
card c1, c2;

// 7
typedef struct {
    int value;
    char *name;
} item;
item i1, i2;
```

Structures (3)

- 자기 참조 구조체
  - 자기 참조 구조체는 자신의 형을 참조하는 포인터 멤버를 가짐
  - 이러한 자료구조를 동적 자료구조라고 함
  - 동적 자료구조는 기억장소관리 루틴을 사용하여 명시적으로 메모리 할당을 요구함 (malloc - <stdlib.h>)
  - 또한 삭제시 원소를 free()를 사용하여 메모리 반납

Structures (4)

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

typedef int element;

typedef struct ListNode {
    element data;
    struct ListNode *link;
} ListNode;

ListNode *create_node(element item, ListNode *link) {
    ListNode *new_node = (ListNode *)malloc(sizeof(ListNode));
    if ( new_node == NULL ) {
        perror("malloc error");
    } else {
        new_node->data = item;
        new_node->link = link;
        return new_node;
    }
}
```



- 포인터 변수 next(link)를 연결이라고 하고,
  그림으로 나타낼 때 연결을 화살표로 표현하는 것이 좋음
- next(link)는 다음 list원소의 메모리주소나, NULL을 가짐

Structures (4)

* 멤버 연산자 ; membership operator
  - " . " : [person a; a.name = manz; a.age=24;]
* 멤버 포인터 연산자
  - " -> " [person *p; p =&a; p->name = manz; p->age = 24;]
  - p->name = manz; / (*p).name = manz;
* 동적 메모리 할당 ; Dynamic memory allocation
  - 프로그램이 실행하는 도중에 동적으로 메모리를 할당받는 것

```
malloc(size) : 메모리할당
free(ptr) : 메모리해제
sizeof(var) : 자료의 크기 반환 (byte)
  ex > person *ptr = (person *)malloc(sizeof(person)); free(ptr);
```

◆ 동적 메모리 할당 -> Memory의 heap 영역

끝