C Programming – Day 4

2018.05.10

JunGu Kang
Whols

아주대학교 Whols

구조체

구조체

하나 이상의 변수를 묶어 새로 정의한 자료형

구조체의 정의와 구조체 변수의 선언

```
struct name {
    variable_declarations;
};

struct name variable_name;
```

구조체의 정의와 구조체 변수의 선언

```
typedef struct name {
    variable_declarations;
} new_name;

new_name variable_name;
```

구조체의 멤버

```
typedef struct name {
    int normal_variable;
    int array[10];
    int * ptr;
    struct another_struct structure;
    // 구조체가 멤버로 가질 수 있는 변수에는 제한이 없다.
    // 특히 구조체를 멤버로 가지는 구조체를 중첩 구조체라 한다.
} new_name;
```

구조체의 초기화

```
typedef struct {
    int number;
    char string[10];
    float * ptr;
} Member;

main() {
    float a = 10.3;
    Member members = {123, “string”, &a};
}
```

구조체 멤버에 접근

```
typedef struct {
    int number;
    char string[10];
    float * ptr;
} Member;

main() {
    float a = 10.3;
    Member members = {123, “string”, &a};
    printf(“%d\n”, members.number);  // 123
    printf(“%s\n”, members.string);  // string
    printf(“%p\n”, members.ptr);  // &a
    printf(“%d\n”, *members.ptr);  // 10.3
}
```

중첩 구조체의 정의

```
typedef struct {
    char first_name[20];
    char last_name[20];
} Name;

typedef struct {
    int student_no;
    Name student_name;
} Student;
```

중첩 구조체의 초기화

```
typedef struct {
    char first_name[20];
    char last_name[20];
} Name;

typedef struct {
    int student_no;
    Name student_name;
} Student;

main() {
    Student = {123, {“GilDong”, “Hong”}}
    // Student = {123, “GilDong”, “Hong”} 로 써도 된다.
}
```

구조체 포인터

구조체도 메모리에 저장되므로 포인터로 가리킬 수 있다.

구조체 포인터

```
typedef struct {
    int number1;
    int number2;
} Numbers;

main() {
    Numbers num = {123, 234}
    Numbers * num_ptr = &num;
}
```

구조체 포인터

```
typedef struct {
    int number1;
    int number2;
} Numbers;

main() {
    Numbers num = {123, 234}
    Numbers * num_ptr = &num;
    printf("%d\n", num_ptr->number1);
    printf("%d\n", num_ptr->number2);
}
```

중첩 구조체

```
(*ptr).member == ptr->member
```

구조체 배열

구조체도 type이므로 **배열로 선언할 수 있다.**

구조체 배열

```
typedef struct {
    int number1;
    int number2;
} Numbers;

main() {
    Numbers num_arr[3] = { {1, 2}, {3, 4}, {5, 6} };
    for(int i = 0; i < 3; i++)
        printf("%d, %d\n", num_arr[i].number1, num_arr[i].number2);
}
```

구조체 연산

구조체를 대상으로 하는 연산은 제한적이다.

구조체 연산

```
typedef struct {
    int number1;
    int number2;
} Numbers;

main() {
    Numbers num1 = {123, 234};
    Numbers num2 = num1;  // num1을 num2로 복사
    printf("%d, %d\n", num2.number1, num2.number2);
}
```

구조체 연산

```
typedef struct {
     int number1;
     int number2;
} Numbers;

main() {
     Numbers num = {123, 234};
     printf(“%ld\n”, sizeof num);
}
```

구조체 연산

```
typedef struct {
    int number1;
    int number2;
} Numbers;

main() {
    Numbers num1 = {123, 234};
    Numbers num2 = {345, 456};
    num1 + num2;  // 이런 연산을 할 수 없다.
}
```

구조체 연산

구조체 변수를 대상으로 연산을 하기 위해서는
따로 함수를 정의해야 한다.

공용체

공용체

같은 메모리 공간을 다양한 방법으로 접근한다.

공용체의 정의

```
union test {
    char char_member;
    int int_member;
    long long_member;
};
```

공용체의 선언

```
union test {
    char char_member;
    int int_member;
    long long_member;
};

main {
    union test test_union;
    test_union = 0x12345678;
}
```

공용체의 정의

```
typedef union {
    char char_member;
    int int_member;
    long long_member;
} Test;
```

공용체의 선언

```
typedef union {
    char char_member;
    int int_member;
    long long_member;
} Test;

main {
    Test test_union;
    test_union = 0x12345678;
}
```

공용체

```
typedef union test {
    char char_member;
    int int_member;
    long long_member;
} Test;

main {
    Test test_union;
    test_union = 0x12345678;
    printf("%ld\n", sizeof test_union);  // 8
}
```

공용체

```
typedef union test {
    //생략
} Test;

main {
    Test test_union;
    test_union = 0x12345678;
    printf(“%p\n”, &test_union.char_member);
    printf(“%p\n”, &test_union.int_member);
    printf(“%p\n”, &test_union.long_member);
}
```

공용체

```
typedef union test {
    //생략
} Test;

main {
    Test test_union;
    test_union = 0x12345678;
    printf(“%ld\n”, test_union.char_member);
    printf(“%ld\n”, test_union.int_member);
    printf(“%ld\n”, test_union.long_member);
}
```

열거형

열거형 정의

```
enum color {
    RED = 1, BLUE = 2, GREEN = 3
};
```

열거형 정의

```
enum color {
    ZERO, ONE, TWO, THREE  // 0, 1, 2, 3
};
```

열거형 정의

```
enum color {
    ZERO, ONE, FOUR = 4, FIVE  // 0, 1, 4, 5
};
```

열거형 정의

```
typedef enum color {
    RED = 1, BLUE = 2, GREEN = 3
} Color;
```

열거형 변수의 선언

```
typedef enum color {
    RED = 1, BLUE = 2, GREEN = 3
} Color;

main() {
    Color favorite_color;
    favorite_color = BLUE;
    printf("%d\n", favorite_color);  // 2
}
```

열거형 변수의 선언

```
typedef enum color {
    RED = 1, BLUE = 2, GREEN = 3
} Color;

main() {
    Color favorite_color;
    favorite_color = BLUE;

    switch(favorite_color) {
        case RED:
            printf(“Favorite Color is Red!\n”);
        // 생략
    }
}
```

표준 입출력

printf / scanf

Formatting된 문자열을 입출력한다.

printf

```
main() {
    printf(“hello, world!”);
}
```

printf

```
main() {
    int a = 10;
    printf(a);  // 이렇게 쓸 수도 있음
                // 그러나 FSB 취약점이 존재하기 때문에 사용해서는 안 됨
}
```

printf

```
main() {
    int a = 10;
    float b = 7.32;
    printf("a는 %d, b는 %f", a, b);
}
```

Format String

Format string	Meaning
%d, %i	10진수 정수(Decimal)
%u	부호 없는 10진수 정수(Unsigned Decimal)
%o	부호 없는 8진수 정수(Unsigned Octal)
%x, %X	부호 없는 16진수 정수(Unsigned Hexadecimal)
%c	문자 하나(Single Character)
%s	문자열(String)
%f, %F	10진수 실수(Double)
%e, %E	부동소수점 표현으로 나타낸 10진수 실수(Double)
%g, %G	%f 또는 %g중에서 알아서 결정(Double)
%a, %A	16진수로 실수(Double)
%p	포인터(Pointer)
%n	지금까지 출력된 글자 수를 메모리 공간에 저장한다.

Escape Sequence

Character	Escape Sequence
비프음(Alert)	\a
백스페이스(Backspace)	\b
폼피드(Formfeed)	\f
줄 바꿈(New Line)	\r
줄 바꿈(Carriage Return)	\n
수평 탭(Horizontal Tab)	\t
수직 탭(Vertical Tab)	\v
백슬래쉬(Backslash)	\\
물음표(Question Mark)	\?
작은따옴표(Single Quote)	\’
큰따옴표(Double Quote)	\”
8진수(Octal Number)	\ooo
16진수(Hexadecimal Number)	\xhh

scanf

```
main() {
    int a;
    float b;
    scanf(“%d”, &a);  // 10진수 정수로 입력 받기
    scanf(“%f”, &b);  // 10진수 실수로 입력 받기
}
```

왜 변수 앞에 &가 붙지?

~~포인터를 배우면 알게 된다.~~

포인터를 배웠으니 알 수 있다.

printf

변수에 저장된 내용만 출력하면 된다. → **Call by Value**

scanf

변수에 값을 직접 저장해줘야 한다. → **Call by Reference**

스트림

프로그램으로 입출력을 할 수 있도록 연결해주는 것

putchar / getchar

하나의 문자를 표준 스트림으로 입출력한다.

putchar / getchar

```
#include <stdio.h>

int putchar(int);
// 성공하면 출력한 문자, 실패하면 EOF

int getchar(void);
// 성공하면 입력받은 문자, 파일 끝에 도달하거나 실패하면 EOF
```

putchar / getchar

```
#include <stdio.h>

main() {
     int ch;
     ch = getchar();
     putchar(ch);
}
```

fputc / fgetc

하나의 문자를 지정한 스트림으로 입출력한다.

fputc / fgetc

```
#include <stdio.h>

int fputc(int, FILE *);
// 성공하면 출력한 문자, 실패하면 EOF

int fgetc(FILE *);
// 성공하면 입력받은 문자, 파일 끝에 도달하거나 실패하면 EOF
```

fputc / fgetc

```
#include <stdio.h>

main() {
    int ch;
    ch = fgetc(stdin);
    fputc(ch, stdout);
}
```

puts / gets

하나의 문자열을 표준 스트림으로 입출력한다.

puts / gets

```
#include <stdio.h>

int puts(const char *);
// 성공하면 음수가 아닌 값, 실패하면 EOF

char * gets(char *);
// 파일의 끝에 도달하거나 실패하면 NULL
```

puts / gets

```
#include <stdio.h>

main() {
    char str[1024];
    gets(str);
    puts(str);
}
```

fputs / fgets

하나의 문자열을 일정 길이만큼 입출력한다.

fputs / fgets

```
#include <stdio.h>

int fputs(const char *, FILE *);
// 성공하면 음수가 아닌 값, 실패하면 EOF

char * gets(char *, int, FILE *);
// 파일의 끝에 도달하거나 실패하면 NULL
```

fputs / fgets

```
#include <stdio.h>

main() {
    char str[1024];
    fgets(str, sizeof str, stdin);
    puts(str, stdout);
}
```

fgets

\n을 만날 때까지 입력받는다. 단, \n도 입력받는다.

EOF

```
// stdio.h

#define EOF (-1)
```

EOF

EOF를 반환한다 == -1을 반환한다

Buffer

데이터는 버퍼를 거쳐서 입출력된다.(Buffering)

Buffer

입출력은 매우 느린 작업이기 때문에
실시간으로 처리하면 비효율적이다.

fflush

지정한 스트림의 버퍼를 비운다.

fflush

```
#include <stdio.h>

int fflush(FILE *)
// 성공하면 0, 실패하면 EOF
```

파일 입출력

파일 스트림

파일과 프로그램을 연결하는 스트림.

fopen / fclose

파일을 열고 닫는다.

fopen / fclose

파일 스트림을 열고 닫는다.

fopen / fclose

```
#include <stdio.h>

FILE * fopen(const char *, const char *);
// 성공하면 해당 파일의 FILE 구조체 포인터, 실패하면 NULL

FILE * fclose(FILE *);
// 성공하면 0, 실패하면 EOF
```

fopen / fclose

```
#include <stdio.h>

main() {
    FILE * fp = fopen(“file_name.txt”, “rt”);  // 읽기모드
    if(fp == NULL) return -1;
    fclose(fp);
}
```

fopen

파일은 한 가지 모드로만 열 수 있다.

fopen

Mode	Meaning
rt	텍스트로 읽기(파일이 없으면 오류)
wt	텍스트로 쓰기
at	텍스트로 이어서 쓰기
rt+	텍스트로 읽고 쓰기(파일이 없으면 오류)
wt+	텍스트로 읽고 쓰기
at+	텍스트로 읽고 이어서 쓰기
rb	바이너리로 읽기(파일이 없으면 오류)
wb	바이너리로 쓰기
ab	바이너리로 이어서 쓰기
rb+	바이너리로 읽고 쓰기(파일이 없으면 오류)
wb+	바이너리로 읽고 쓰기
ab+	바이너리로 읽고 이어서 쓰기

fopen

가급적 r, w, a를 쓰자.

fclose

반드시 `fclose`를 사용해 파일 스트림을 닫아주어야 한다.

개행

개행이 항상 \n으로 처리되지는 않는다.

개행

OS	New Line
Microsoft Windows	\r\n
MacOS	\r
Unix / Linux	\n

fputc / fgetc

하나의 문자를 지정한 스트림으로 입출력한다.

fputc / fgetc

```
#include <stdio.h>

int fputc(int, FILE *);
// 성공하면 출력한 문자, 실패하면 EOF

int fgetc(FILE *);
// 성공하면 입력받은 문자, 파일 끝에 도달하거나 실패하면 EOF
```

fputc

```
#include <stdio.h>

main() {
    FILE * fp = fopen(“file_name.txt”, “wt”);
    if(fp == NULL) return -1;

    fputc(‘A’, fp);

    fclose(fp);
}
```

fgetc

```
#include <stdio.h>

main() {
    FILE * fp = fopen(“file_name.txt”, “rt”);
    int ch;
    if(fp == NULL) return -1;

    ch = fgetc(fp);
    printf(“%c\n”, ch);

    fclose(fp);
}
```

fputs / fgets

하나의 문자열을 지정한 스트림으로 입출력한다.

fputs / fgets

```
#include <stdio.h>

int fputc(int, FILE *);
// 성공하면 출력한 문자, 실패하면 EOF

int fgetc(FILE *);
// 성공하면 입력받은 문자, 파일 끝에 도달하거나 실패하면 EOF
```

fputs

```
#include <stdio.h>

main() {
    FILE * fp = fopen(“file_name.txt”, “wt”);
    if(fp == NULL) return -1;

    fputs(“Hello, World!\n”, fp);

    fclose(fp);
}
```

fgets

```
#include <stdio.h>

main() {
    FILE * fp = fopen(“file_name.txt”, “rt”);
    char str[1024];
    if(fp == NULL) return -1;

    fgets(str, sizeof str, fp);
    printf(“%s”, str);

    fclose(fp);
}
```

feof

EOF는 파일의 끝에서도 발생하고, 오류에서도 발생한다.

feof

EOF가 발생했을때 파일의 끝에 도달한 것인지,
오류가 발생한 것인지 구별할 필요가 있다.

feof

```
#include <stdio.h>

int feof(FILE *);
// 파일의 끝에 도달했다면 0이 아닌 값
// 파일의 끝이 아니라면(오류) 0
```

feof

```
#include <stdio.h>

main() {
    FILE * fp = fopen(“file_name.txt”, “rt”);
    char str[1024];
    if(fp == NULL) return -1;

    while(fgets(str, sizeof str, fp) != NULL) printf(“%s, str);

    if(feof(fp) != 0) printf(“success”);
    else printf(“failed”);

    fclose(fp);
}
```

fread / fwrite

지정한 크기만큼 바이너리로 입출력한다.

fread / fwrite

```
#include <stdio.h>

size_t fread(void *, size_t, size_t, FILE *);
// 성공하면 갯수, 실패하면 갯수보다 작은 값

size_t fwrite(const void *, size_t, size_t, FILE *);
// 성공하면 갯수, 실패하면 갯수보다 작은 값
```

fread

```
#include <stdio.h>

main() {
    FILE * fp = fopen(“file_name.txt”, “rb”);
    int data[10];
    if(fp == NULL) return -1;

    fread(data, sizeof (int), 10, fp);

    fclose(fp);
}
```

fwrite

```
#include <stdio.h>

main() {
    FILE * fp = fopen(“file_name.txt”, “wb”);
    int data[10] = { /* some data */ };
    if(fp == NULL) return -1;

    fwrite(data, sizeof (int), 10, fp);

    fclose(fp);
}
```

fprintf / fscanf

Formatting해서 입출력.

fprintf

```
#include <stdio.h>

main() {
    FILE * fp = fopen(“file_name.txt”, “rb”);
    int num1 = 123, num2 = 234, num3 = 345;
    if(fp == NULL) return -1;

    fprintf(fp, “%d %d %d”, num1, num2, num3);  // 123 234 345

    fclose(fp);
}
```

fscanf

```
#include <stdio.h>

main() {
    FILE * fp = fopen(“file_name.txt”, “rb”);
    int num1 = 123, num2 = 234, num3 = 345;
    if(fp == NULL) return -1;

    fscanf(fp, “%d %d %d”, &num1, &num2, &num3);  // 123 234 345

    fclose(fp);
}
```

파일을 중간부터 읽고싶다면?

fseek

```
#include <stdio.h>

int fseek(FILE *, long, int);
// 성공하면 0, 실패하면 0이 아닌 값
```

fprintf

```
#include <stdio.h>

main() {
    FILE * fp = fopen(“file_name.txt”, “rb”);
    int num1 = 123, num2 = 234, num3 = 345;
    if(fp == NULL) return -1;

    fseek(fp, 100, SEEK_SET);  // 처음 위치부터 100바이트 뒤로
    fseek(fp, 200, SEEK_CUR);  // 현재 위치부터 200바이트 뒤로
    fseek(fp, -100, SEEK_END);  // EOF에서부터 100바이트 앞으로

    fclose(fp);
}
```

fseek

현재 커서의 위치를 알려준다.

fseek

```
#include <stdio.h>

long ftell(FILE *)
```

fprintf

```
#include <stdio.h>

main() {
    FILE * fp = fopen(“file_name.txt”, “rb”);
    int num1 = 123, num2 = 234, num3 = 345;
    if(fp == NULL) return -1;

    fseek(fp, 100, SEEK_SET);  // 처음 위치부터 100바이트 뒤로
    ftell(fp);  // 100

    fclose(fp);
}
```

문자열 다루기

strlen

NUL을 제외한 문자열의 길이를 구한다.

strlen

```
#include <string.h>

size_t strlen(const char *);
// 문자열의 길이
```

strlen

```
#include <string.h>

main() {
     char * ptr = “Hello, World!”;
     printf(“%d\n”, strlen(ptr));
}
```

strcpy

문자열을 복사한다.

strcpy

```
#include <string.h>

char * strcpy(char *, const char *);
// 복사된 문자열의 주소
```

strcpy

```
#include <string.h>

main() {
    char src[1024] = “Hello, World!”;
    char dst[1024];
    strcpy(dst, src);
}
```

strncpy

문자열을 복사한다.

strncpy

```
#include <string.h>

char * strncpy(char *, const char *, size_t);
// 복사된 문자열의 주소
```

strncpy

```
#include <string.h>

main() {
    char src[1024] = “Hello, World!”;
    char dst[8];
    strcpy(dst, src, sizeof dst - 1);  // 7바이트만 복사
    dst[sizeof dst - 1] = 0;  // 마지막 바이트는 NULL로 고정
}
```

strcat

두 문자열을 연결한다.

strcat

```
#include <string.h>

char * strcat(char *, const char *);
// 덧붙여진 문자열의 주소
```

strcat

```
#include <string.h>

main() {
    char src[512] = “World!”;
    char dst[1024] = “Hello ”,
    strcat(dst, src);
}
```

strncat

지정한 길이만큼 가져와 두 문자열을 연결한다.

strncat

```
#include <string.h>

char * strncat(char *, const char *, size_t);
// 덧붙여진 문자열의 주소
```

strncat

```
#include <string.h>

main() {
    char src[512] = “World!”;
    char dst[1024] = “Hello ”,
    strncat(dst, src, 5);  // 5바이트만 복사
}
```

strcmp

두 문자열을 비교한다.

strcmp

```
#include <string.h>

int * strcmp(const char *, const char *);
// 두 문자열이 같으면 0, 아니면 0이 아닌 값
// 첫 문자열이 더 크면(뒤에 있으면) 0보다 큰 값
// 나중 문자열이 더 크면(뒤에 있으면) 0보다 작은 값
```

strcmp

```
#include <string.h>

main() {
    char src[1024] = “Hello, World!”;
    char dst[1024] = “Hello, World!”,
    int result = strcmp(dst, src);
}
```

strncat

두 문자열을 지정한 길이만큼 비교한다.

strncat

```
#include <string.h>

int strncat(const char *, const char *, size_t);
// 두 문자열이 같으면 0, 아니면 0이 아닌 값
// 첫 문자열이 더 크면(뒤에 있으면) 0보다 큰 값
// 나중 문자열이 더 크면(뒤에 있으면) 0보다 작은 값
```

strncat

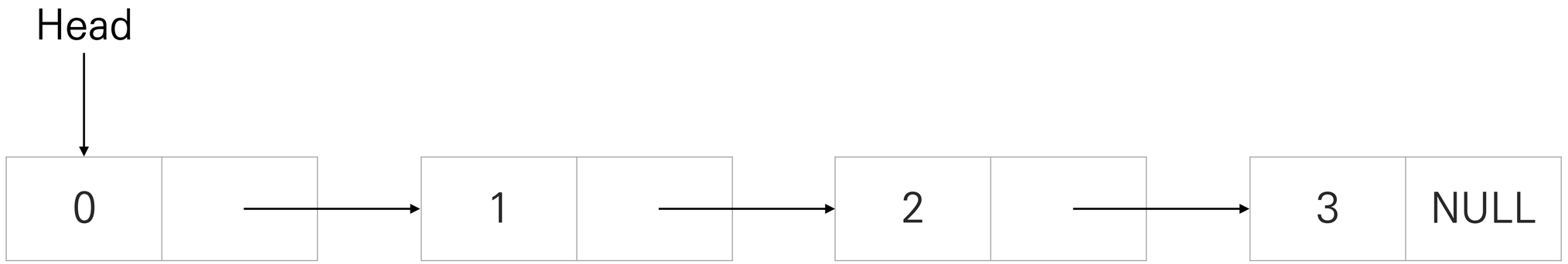
```
#include <string.h>

main() {
    char src[1024] = “Hello, World!”;
    char dst[1024] = “Hello, C!”,
    int result = strncmp(dst, src, 7);  // 7바이트만 비교
}
```

Linked List

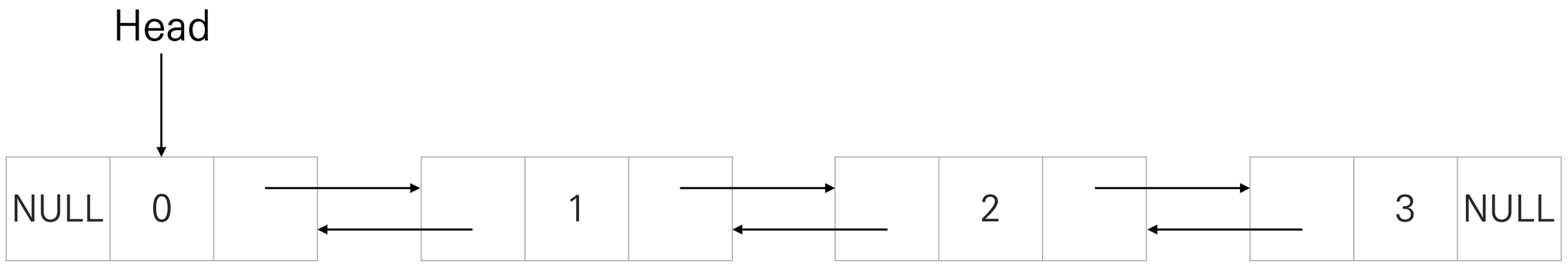
Linked List

Linked List



Double Linked List

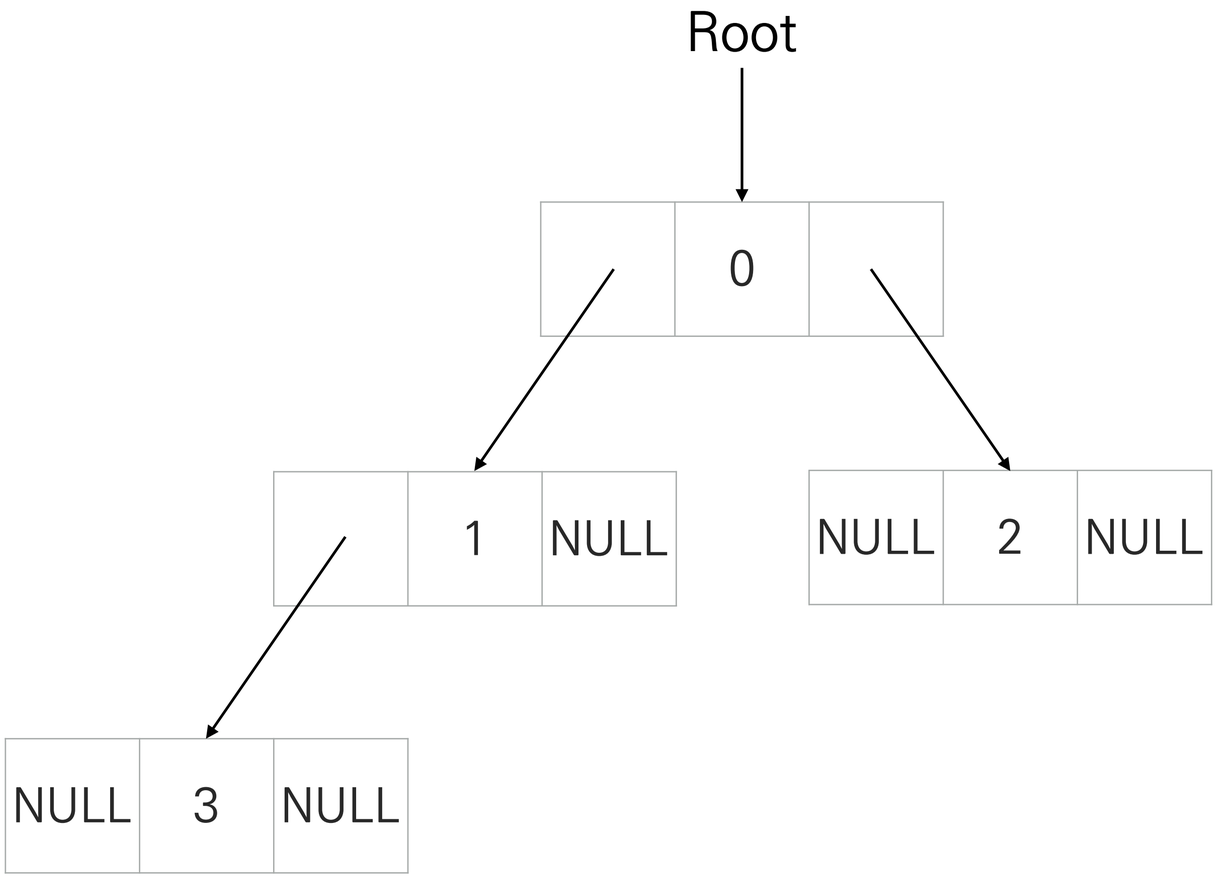
Double Linked List



Tree

pointer (left child)	value	pointer (right child)

Tree



Tree Traversal

```
void travel(node *) {
    printf(“%c\n”, node->value);
    if(node->left) travel(node->left);
    if(node->right) travel(node->right);
}
```

Tree Traversal

```
void travel(node *) {
    if(node->left) travel(node->left);
    printf("%c\n", node->value);
    if(node->right) travel(node->right);
}
```

Tree Traversal

```
void travel(node *) {
    if(node->left) travel(node->left);
    if(node->right) travel(node->right);
    printf(“%c\n”, node->value);
}
```

C Programming – Day 4

2018.05.10

JunGu Kang
Whols

아주대학교 Whols