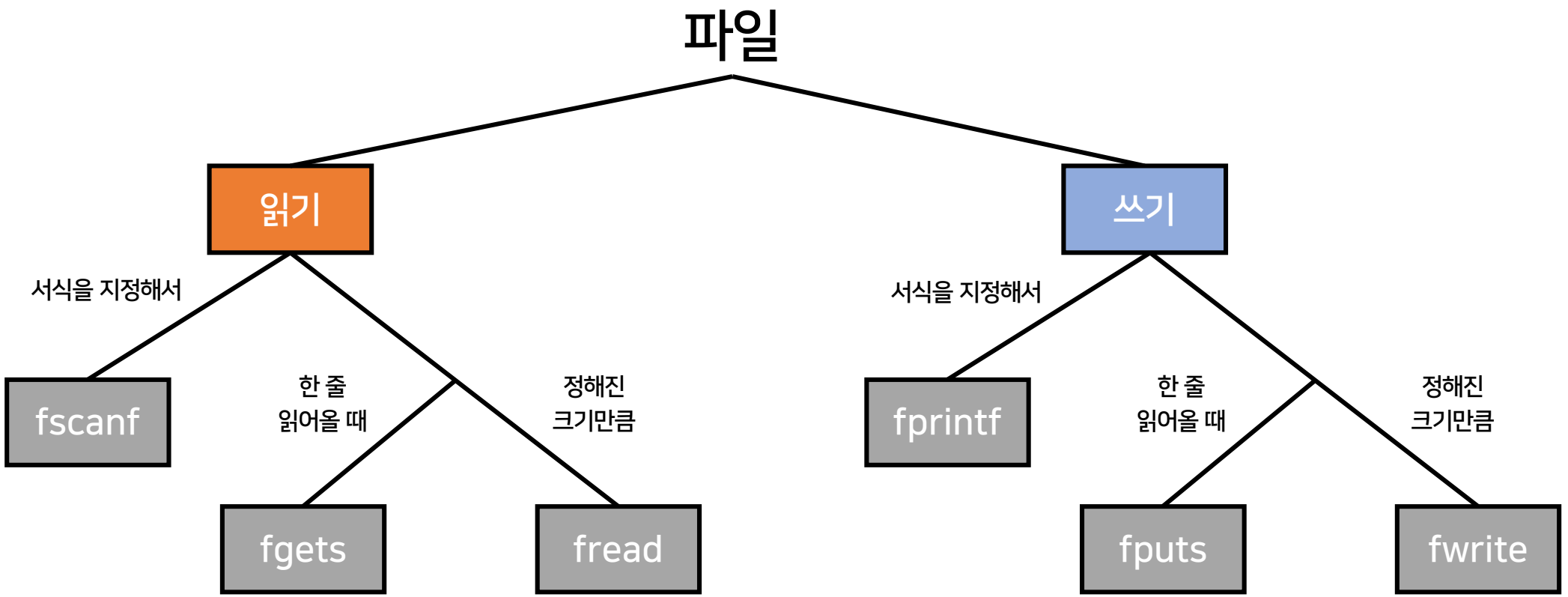
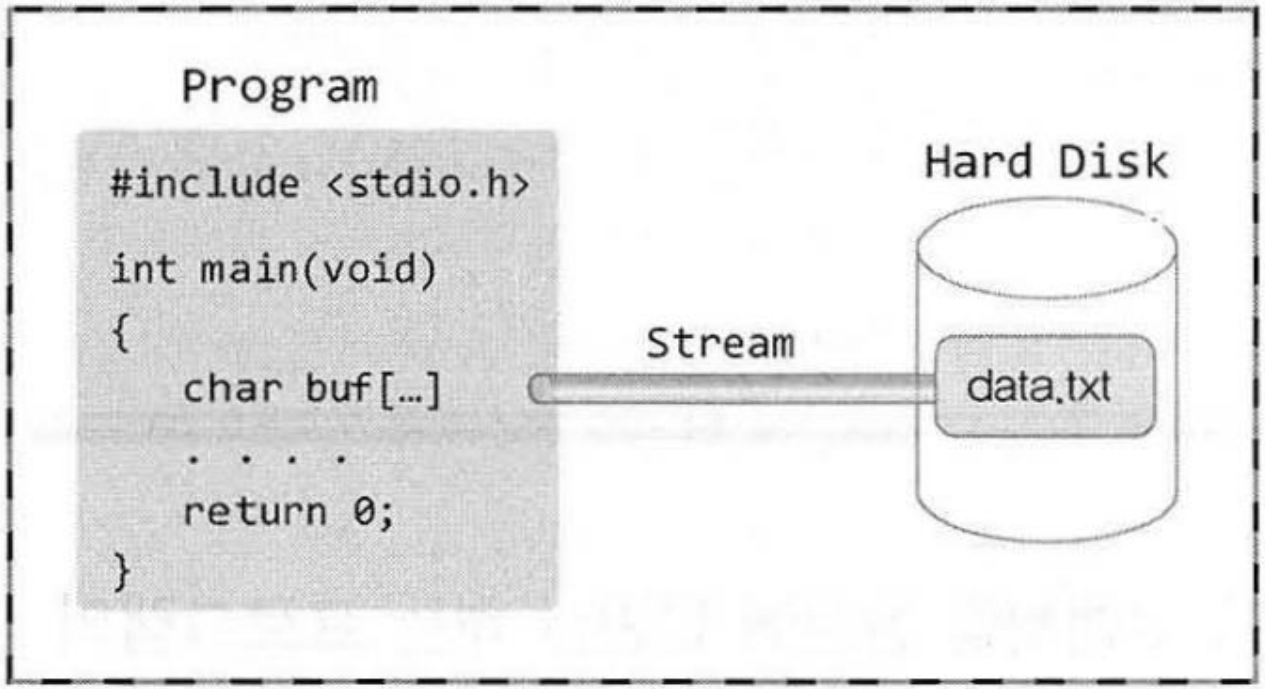
파일 입출력

11월 셋째 주

오늘 배울 내용 구조도


파일
읽기
쓰기
서식을 지정해서
서식을 지정해서
fscanf
한 줄
읽어올 때
정해진
크기만큼
fprintf
한 줄
읽어올 때
정해진
크기만큼
fgets
fread
fputs
fwrite

스트림 (Stream)



파일에 저장되어 있는 데이터를 참조하기 위해서는

프로그램과 참조할 데이터가 저장되어 있는 파일 사이에 데이터가 이동할 수 있는 이동 통로가 필요

프로그램과 파일 사이에 스트림을 형성해야 데이터를 주고 받을 수 있다!

파일 입출력의 전체적인 구조

```c
#include <stdio.h>

int main(void) {
    FILE* fp = fopen("hello.txt", "w");

    fprintf(fp, "%s %d\n", "Hello", 100);

    fclose(fp);

    return 0;
}
```

1. 파일 열기

fopen(“파일 이름 혹은 경로”, 모드);

2. 파일 읽거나 쓰기

fprintf(파일 포인터, 서식, 값 1, 값 2, …);

3. 파일 닫기

fclose(파일 포인터);

파일 스트림 형성과 FILE 구조체

파일 입출력의 기본

```
FILE * fp = fopen("data.txt", "wt");     // 출력 스트림의 형성
```

1. fopen 함수가 호출되면 FILE 구조체 변수가 생성

2. 생성된 FILE 구조체 변수에는 파일에 대한 정보가 담김

3. FILE 구조체의 포인터는 사실상 파일을 가리키는 포인터 역할

(스트림이 형성되어 fp라는 변수만 가지고 파일에 접근할 수 있다는 것임!)

File open Mode

모드(mode)	스트림의 성격	파일이 없으면?
r	읽기 가능	에러
w	쓰기 가능	생성
a	파일의 끝에 덧붙여 쓰기 가능	생성
r+	읽기/쓰기 가능	에러
w+	읽기/쓰기 가능	생성
a+	읽기/덧붙여 쓰기 가능	생성

[표 24-1: 파일 개방 모드의 첫 번째 기준]

+ 를 붙이면 읽기와 쓰기가 모두 가능한 스트림을 형성

a 는 쓰기 가능한 스트림을 형성하는데, 덧붙여 쓰기를 말하는 것임 (append)

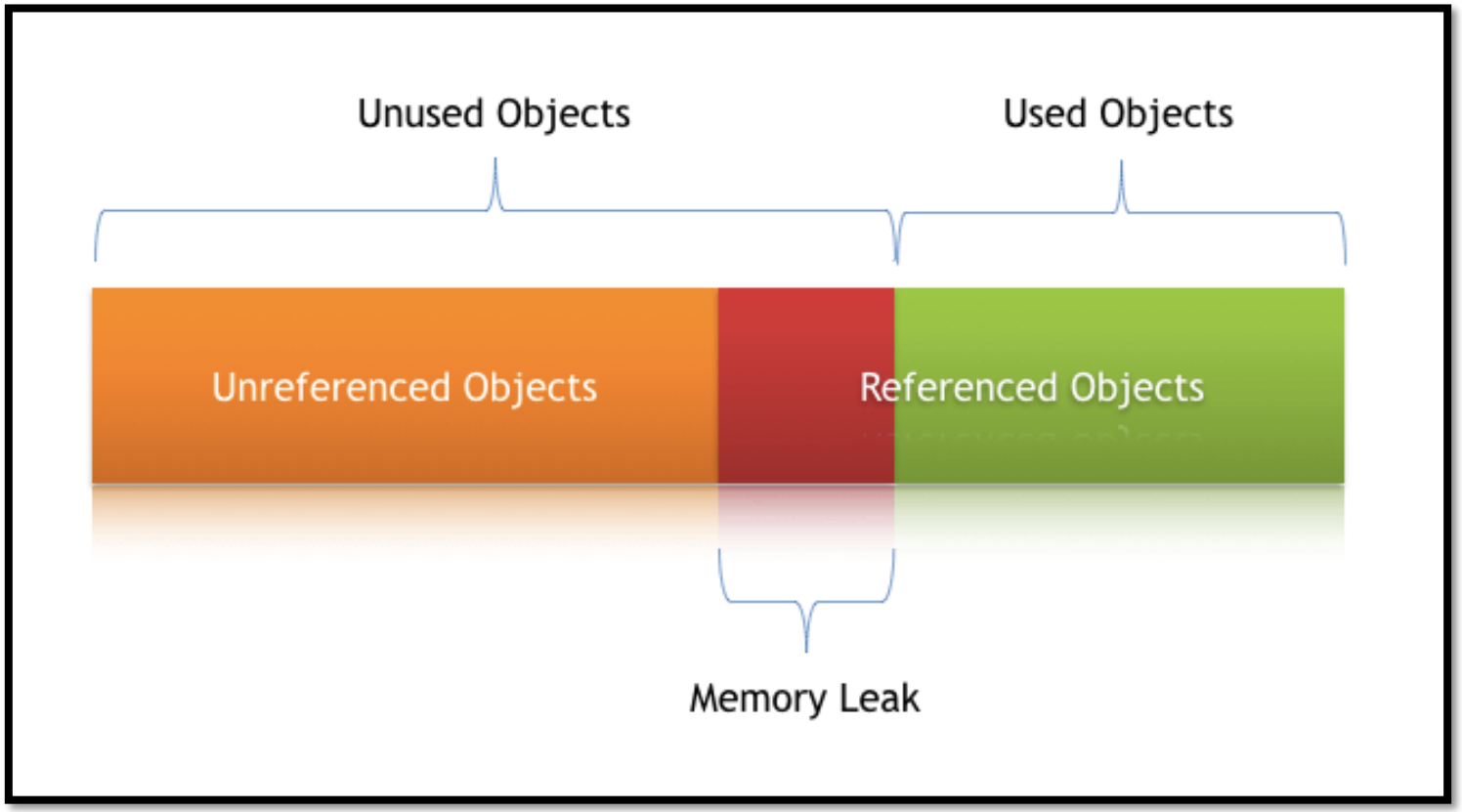
플러스는 왜 붙이는거야

참 고

웬만하면 r, w, a 중에서 선택하세요

파일의 개방 모드 중 r+, w+, a+는 읽기와 동시에 쓰기가 가능하므로 더 좋은 모드라고 생각할 수 있다. 그러나 이러한 모드를 기반으로 작업하는 경우에는 읽기에서 쓰기, 그리고 쓰기에서 읽기로 작업을 변경할 때마다 메모리 버퍼를 비워줘야 하는 등의 불편함과 더불어 잘못된 사용의 위험성도 따른다. 그래서 r, w, a 중에서 하나를 선택하여 스트림을 형성하는 것이 좋으며, 이것이 보다 일반적인 선택이다.

Memory leak 현상



fclose로 파일 포인터를 닫아주지 않으면, 메모리 누수가 발생한다...

서식을 지정하여 파일에 문자열 쓰기

```c
#include <stdio.h>

int main(void) {
    FILE* fp = fopen("hello.txt", "w");

    fprintf(fp, "%s %d\n", "Hello", 100);

    fclose(fp);

    return 0;
}
```

fprintf(파일 포인터, 서식, 값 1, 값 2, …);

역할 : 서식을 지정하여 파일에 문자열 쓰기

파일에 문자열 쓰기

```c
#include <stdio.h>

int main(void) {
    FILE* fp = fopen("hello.txt", "w");

    fputs("Hello World!", fp);

    fclose(fp);

    return 0;
}
```

fputs(문자열, 파일 포인터);

역할 : 파일에 문자열 쓰기 (한 줄과 관련)

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

int main()
{
    char *s1 = "Hello, world!";

    FILE *fp = fopen("hello.txt", "w");

    fwrite(s1, strlen(s1), 1, fp);

    fclose(fp);

    return 0;
}
```

fwrite(문자열, 문자열 크기, 몇 개의 데이터 저장?, 파일 포인터);

역할 : 파일에 문자열 쓰기

서식을 지정하여 파일에 문자열 읽기

```c
#include <stdio.h>

int main(void) {
    char s1[10];
    int num1;

    FILE* fp = fopen("hello.txt", "r");

    fscanf(fp, "%s %d", s1, &num1);

    printf("%s %d\n", s1, num1);

    fclose(fp);

    return 0;
}
```

fscanf(파일 포인터, 서식, 변수 주소 1, 변수 주소 2, …);

역할 : 서식을 지정하여 파일에 문자열 읽기

파일에 문자열 읽기

1. 왜 배열을 다 0으로 초기화 한 것인가
2. fgets와 fread의 차이점은?

```c
#include <stdio.h>

int main()
{
    char buffer[20];

    FILE *fp = fopen("hello.txt", "r");

    fgets(buffer, sizeof(buffer), fp);

    printf("%s\n", buffer);

    fclose(fp);

    return 0;
}
```

fgets(버퍼, 버퍼크기, 파일 포인터);

역할 : 파일에 문자열 읽기 (한 줄과 관련)

메모장에 몇 줄이 있던 한 줄만 가져와

```c
#include <stdio.h>

int main()
{
    char buffer[20] = { 0, };

    FILE *fp = fopen("hello.txt", "r");

    fread(buffer, sizeof(buffer), 1, fp);

    printf("%s\n", buffer);

    fclose(fp);

    return 0;
}
```

fread(버퍼, 읽기 크기, 읽기 횟수, 파일 포인터);

역할 : 파일에 문자열 읽기

sizeof(buffer) 부분 변경해보기

텍스트 파일 vs 바이너리 파일

컴퓨터는 파일을 다음과 같이 두 가지 종류로 나누어서 다룹니다.

1. 바이너리 파일(binary file)
2. 텍스트 파일(text file)

바이너리 파일은 데이터의 저장과 처리를 목적으로 0과 1의 이진 형식으로 인코딩된 파일을 가리킵니다.
프로그램이 이 파일의 데이터를 읽거나 쓸 때는 데이터의 어떠한 변환도 일어나지 않습니다.

텍스트 파일은 사람이 알아볼 수 있는 문자열로 이루어진 파일을 가리킵니다.
프로그램이 이 파일의 데이터를 읽거나 쓸 때는 포맷 형식에 따라 데이터의 변환이 일어납니다.

개인이 소유하는 도서의 목록

슈퍼마켓의 물품 가격

스파이더맨 영상파일

블랙핑크의 히트곡 음원파일