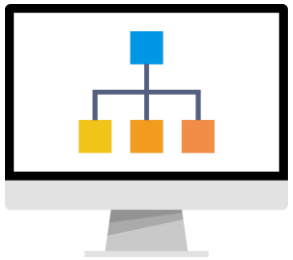


Data Structure & Algorithm
자료구조 및 알고리즘

3. 소스 파일과 헤더 파일 (Sources and Headers)

C 언어

Mar 2020	Mar 2019	Change	Programming Language
1	1		Java
2	2		C
3	3		Python
4	4		C++
5	6	^	C#
6	5	v	Visual Basic .NET
7	7		JavaScript

- 1972년 개발된 프로그래밍 언어
- 소스 파일: *.c, 헤더 파일: *.h
- 여러 운영체제, 임베디드 시스템, 라이브러리, 다른 언어를 만드는 데 쓰이는 중임
  - 라이브러리: 특정 기능을 수행하는 재사용 가능한 코드 뭉치
  - C++, C#, …
- 현업에서는 더욱 추상화된 상위 언어를 쓸 확률이 높지만 (Python, Javascript 등) 자료구조 및 알고리즘과 컴퓨터의 작동 원리를 학습하는 데 가장 적합한 언어

헤더 파일과 소스 파일

- 헤더 파일 (*.h)
  - 함수의 헤더(함수 이름, 리턴 타입, 인자 목록 및 타입)만 남겨두고 몸통(함수 본문)은 없는 파일
    - **함수의 선언**
  - 이 파일만 가지고 있으면
    - 함수가 어떤 입력을 받아 어떤 결과를 돌려주는지는 알지만
    - 그 결과가 어떻게 계산되는지는 모른다.

```
1 // sum.h
2
3 int sum(int a, int b);
4
```

- 소스 파일 (*.c)

헤더 파일과 소스 파일

- 소스 파일 (*.c)
  - 함수의 헤더 뿐만 아니라 몸통(함수 본문)도 같이 있는 파일
    - **함수의 정의**
  - 이 파일이 있으면 함수가 실제로 어떤 동작을 하는지도 알 수 있다.

```c
// sum.c

int sum(int a, int b) {
    return a + b;
}
```

왜 구분하나요?

- 소스 파일이 헤더 파일을 포함하는데 왜 구분하나요?

- 표면적인 이유:
  - 함수가 무엇을 하는지(what, 헤더)와 어떻게 하는지(how, 소스)를 구분하기 위해.

  - “재민씨, 명세는 이렇게 하기로 했으니, 내부 구현은 알아서 해보세요.”

컴파일

- C언어는 기계가 알아들을 수 있는 언어가 아니다! 기계가 알아들을 수 있는 기계어로 바꾸는 작업을 해야 한다.

컴파일
(Compile)

C 코드
(sum.c)

전처리기
컴파일러
어셈블러

오브젝트 파일
(sum.o)

인간어
실행 불가능

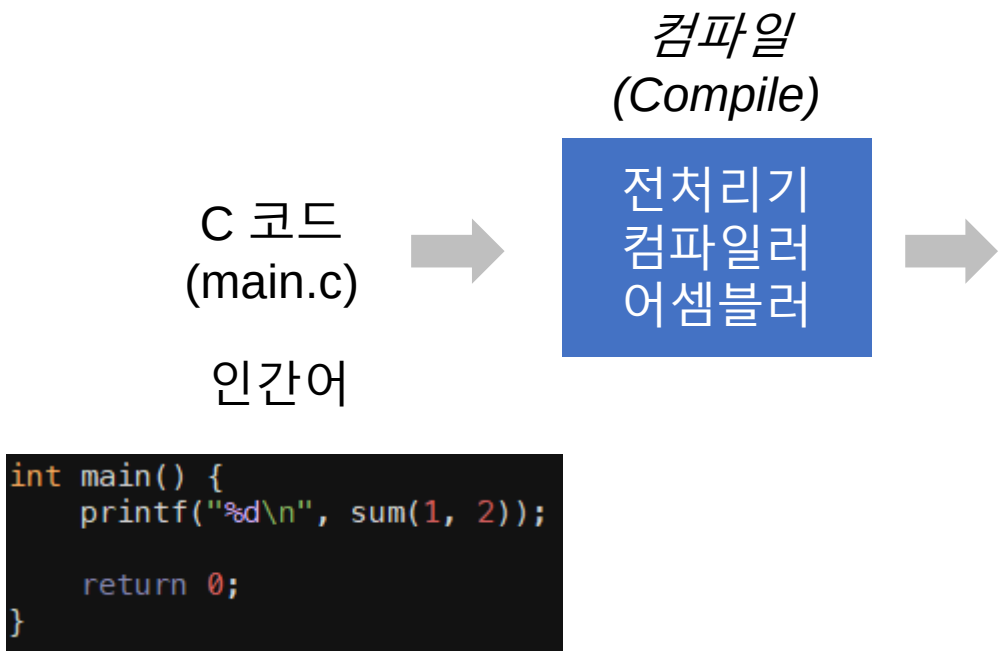
기계어
실행 불가능

```
1 // sum.c
2
3 int sum(int a, int b) {
4     return a + b;
5 }
```

```
1 ^?ELF^B^A^A^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@
2
3
```

컴파일

- sum함수를 정의했지만 호출하지 않는다. main함수를 main.c에 작성하자.
  - 그냥 한 파일에 하면 안되나요? 확장성이 없음



컴파일

- sum함수는 sum.c에 정의가 되어있으나, 이 함수를 부르는 다른 코드에 sum함수의 최소한의 정보(결과 값, 인자는 몇 개를 받는지)는 알려줘야 컴파일이 용이하다 -> 헤더 파일

컴파일
(Compile)

C 코드
(sum.h,
main.c)

전처리기
컴파일러
어셈블러

오브젝트 파일
(main.o)

기계어
실행 불가능

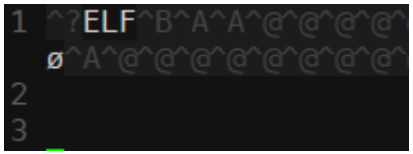
```
1 // sum.h
2
3 int sum(int a, int b);
4
```

```
1 // main.c
2
3 #include <stdio.h>
4 #include "sum.h"
5
6 int main() {
7     printf("%d\n", sum(1, 2));
8
9     return 0;
10 }
```

```
1 ^?ELF^B^A^A^@^@^@^@^@^@^@
  0^C^@^@^@^@^@^@^@^@^@^@^@
  @^@è^@^@^@^@<89>Æ¿^@^@^@
  ^@^@è^@^@^@^@<89>Æ¿^@^@^@
2 ^@^@GCC: (GNU) 7.4.0^@^@^
  @^\^@^@^@^@^@^@^@Z^@^@^@
```

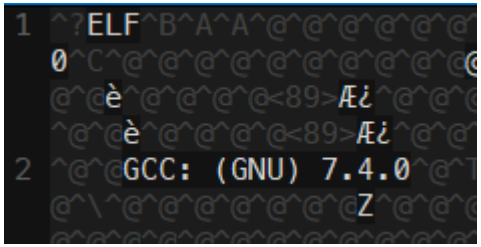
링킹의 필요성

오브젝트 파일
(sum.o)
기계어
실행 불가능



sum 함수의 **정의**가 있다.
main 함수가 없다!

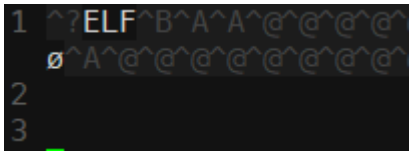
오브젝트 파일
(main.o)
기계어
실행 불가능



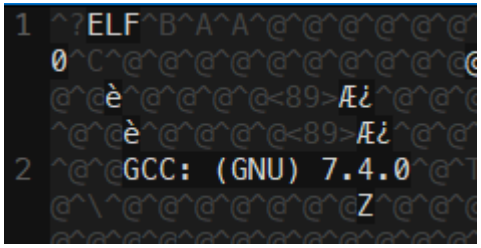
sum 함수의 **정의**가 없다.
main 함수가 있다.

링킹의 필요성

오브젝트 파일
(sum.o)
기계어
실행 불가능



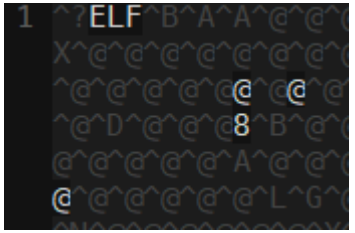
오브젝트 파일
(main.o)
기계어
실행 불가능



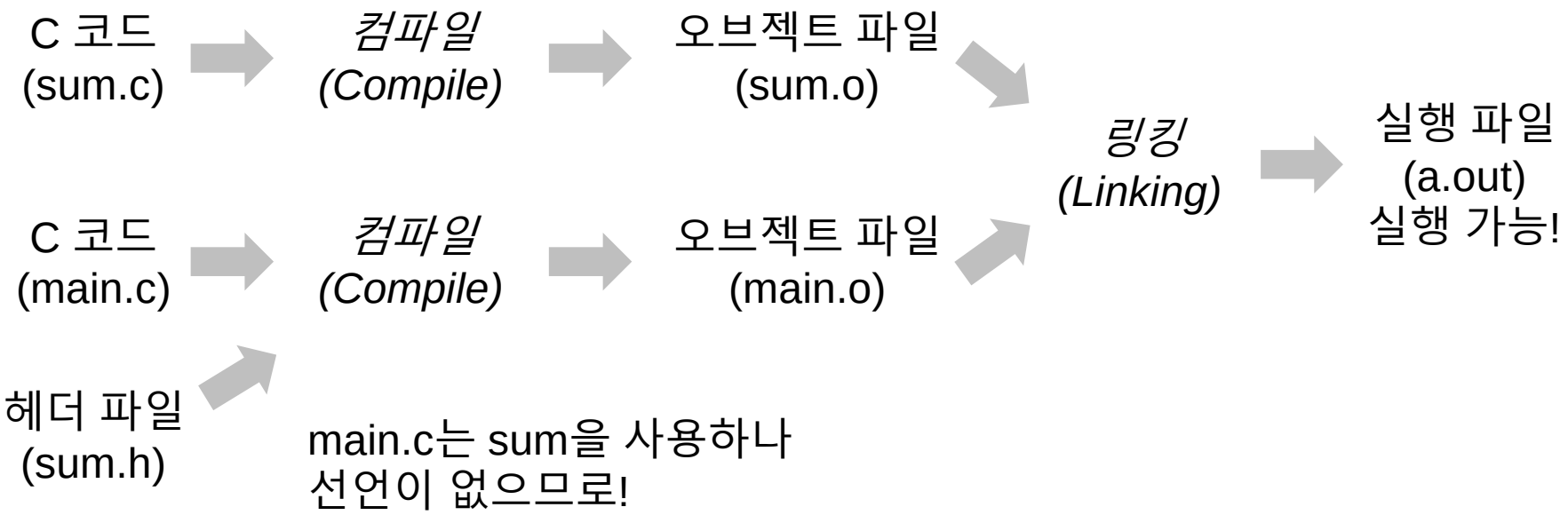
링킹
(Linking)

링커

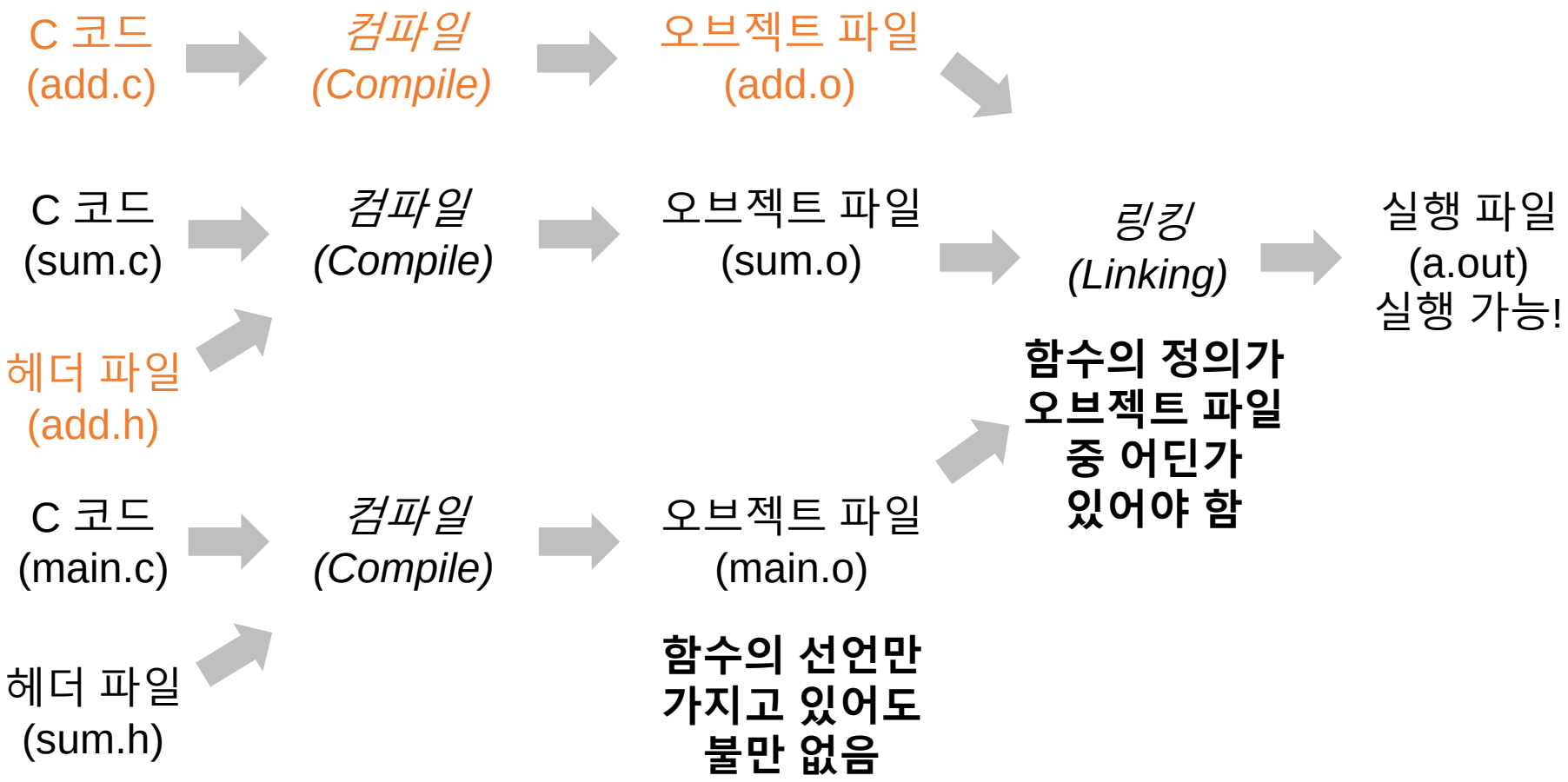
실행 파일
(a.out)
기계어
실행 가능



C 코드의 간략화 된 생명주기



C 코드의 간략화 된 생명주기



왜 소스와 헤더를 구분하나요?

- 실질적인 이유
  - 큰 프로젝트 개발하기 위해서는 코드를 분리해야 하는데 각 코드를 컴파일 시 함수의 선언을 공유하기 위해
  - 분리해서 얻는 이점?
    - 컴파일 시간 단축: 수정된 파일만 다시 오브젝트 파일로 만들면 됨

왜 구분하나요?

- 헤더는 알지만 소스 코드가 없어서
  - 소스 코드가 없어도 충분: printf 등 (헤더: stdio.h)
  - 헤더와 미리 컴파일 된(precompiled) 오브젝트 파일 제공: 오픈 소스 라이브러리
    - “컴파일 하려면 1시간 걸리지? 미리 해줄게”
  - 의도적으로 소스 코드를 비공개하고 헤더와 오브젝트 파일만 제공: 상용 라이브러리
    - “가져다 써. 그런데 고치지는 마”
  - 소스 코드가 더 빠른 저 수준의 언어로 작성되어 있음 (어셈블러 등)

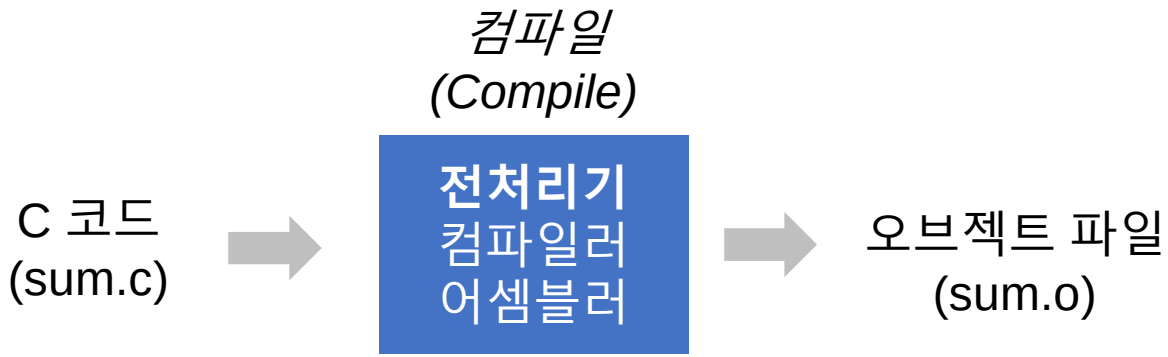
생각해볼 것

- 엄청 큰 라이브러리를 include에서 사용하면 그 라이브러리의 오브젝트 파일이 링킹 과정에서 실행 파일에 포함될 텐데 실행 파일이 엄청 커지지 않을까요?
  - 정적(static) 링킹 vs 동적(dynamic) 링킹

- 최초의 컴파일러는 어떻게 만들어졌을까?
  - 다른 언어로? (어셈블러)
  - 손으로 컴파일해서
  - Bootstrapping

전처리기 (Preprocessor)

- 컴파일이 실행되기 전에 실행되는 코드
- C 소스 파일 자체를 수정하는 코드
- 코드에서 #으로 시작되는 문구들
  - #include, #define …



#include 예제

```c
// main.c

#include <stdio.h>
#include "sum.h"

int main() {
    printf("%d\n", sum(1, 2));

    return 0;
}
```

```c
extern int ftrylockfile (FILE *__stream) __attribute__ ((__nothrow__ , __lea
f__)) ;

extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__ , __lea
f__));
# 943 "/usr/include/stdio.h" 3 4

# 4 "main.c" 2
# 1 "sum.h" 1



# 3 "sum.h"
int sum(int a, int b);
# 5 "main.c" 2

int main() {
 printf("%d\n", sum(1, 2));

 return 0;
}
```

#include

- #include filename
  - filename 파일의 내용을 가져다 덮어쓴다.

- #include <stdio.h>
  - 시스템에 설치된 헤더 파일을 가져온다.
  - stdio.h, stdlib.h 같이 개발 환경에서 제공하는 파일을 가져올 때

- #include “sum.h”
  - 현재 컴파일이 진행되는 디렉토리에서 파일을 찾는다.

- #include “sum.c” 도 가능
- #include “main.c”를 하면?

#define

- #define A B
  - 코드에 있는 A를 모두 B로 치환한다 (문자열 안에 있는 A는 제외)

- 상수(변하지 않는 수)를 정의할 때

```
#define SECONDS_PER_DAY  (60*60*24)
#define PI       3.14159
#define C        299792.458      /*speed of light in km/sec */
#define EOF      (-1)          /*typical end-of-file value */
#define MAXINT   2147483647
```

- 가독성과 이식성을 위함

조건문 전처리기

- #if A
  - A가 참이면 #endif까지의 내용을 남겨둔다.
- #ifdef A
  - A라는 상수가 정의되어 있으면 #endif까지 내용을 남겨둔다.
- #ifndef A
  - A라는 상수가 정의되어 있지 않으면 #endif까지의 내용을 남겨둔다.
- #endif

#if vs #ifdef

- 예) 디버그 모드 설정: 코드에 오류가 있을 때 변수의 값을 실행 중간에 출력할지 말지

```c
#include <stdio.h>

#define DEBUG 1
int main() {
    int a = 5;
#if DEBUG
    printf("debug: a='%d'\n", a);
#endif
    return 0;
}
```

모드 해제:
#define DEBUG 0으로 변경
또는 #define문 삭제

```c
#include <stdio.h>
#define DEBUG

int main() {
    int a = 5;
#ifdef DEBUG
    printf("debug: a='%d'\n", a);
#endif
    return 0;
}
```

모드 해제:
#define문 삭제

#if vs if

- 명확성: 후자는 “코드의 중간에 debug의 값이 바뀌어서 출력이 될 지도 모른다”라고 해석하게 됨.
- 효율성: 전자는 디버그 모드시 printf 문이 컴파일 자체가 안되기 때문에 컴파일 시간 및 실행 파일 크기에서 유리

```c
#include <stdio.h>

#define DEBUG 1
int main() {
    int a = 5;
#if DEBUG
    printf("debug: a='%d'\n", a);
#endif
    return 0;
}
```

```c
#include <stdio.h>

int debug=1;

int main() {
    int a = 5;

    if(debug)
        printf("debug: a='%d'\n", a);

    return 0;
}
```

요약

- 헤더 파일과 소스 파일
  - 헤더 파일: 다른 소스 파일에서 참조할 수 있도록 함수의 선언만 담은 파일
  - 소스 파일: 함수의 실제 구현을 담은 파일

- 컴파일과 링킹
  - 컴파일: 인간어로 된 소스 파일을 기계어로 번역
  - 링킹: 컴파일된 코드 조각들을 모아서 실행 파일을 생성

- 전처리기
  - 컴파일 전 헤더 및 소스 파일을 변경할 수 있는 간단한 코드