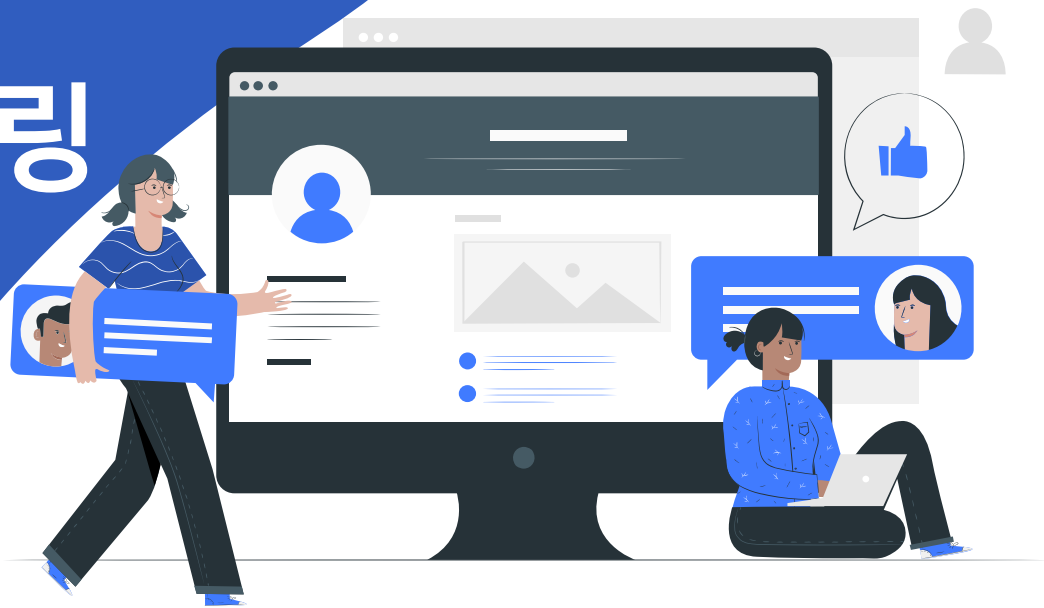


# 리눅스 튜터링



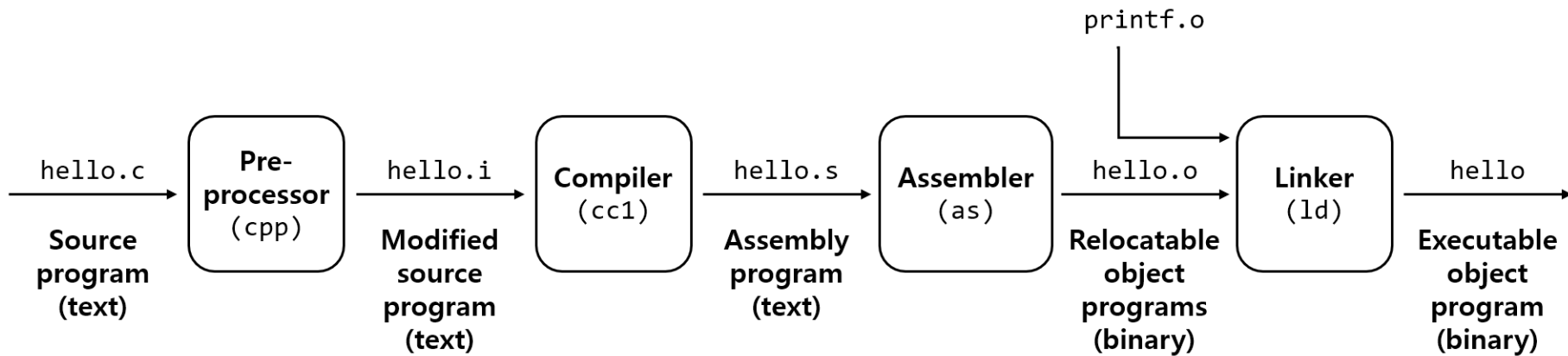
# 01. C언어 컴파일 과정



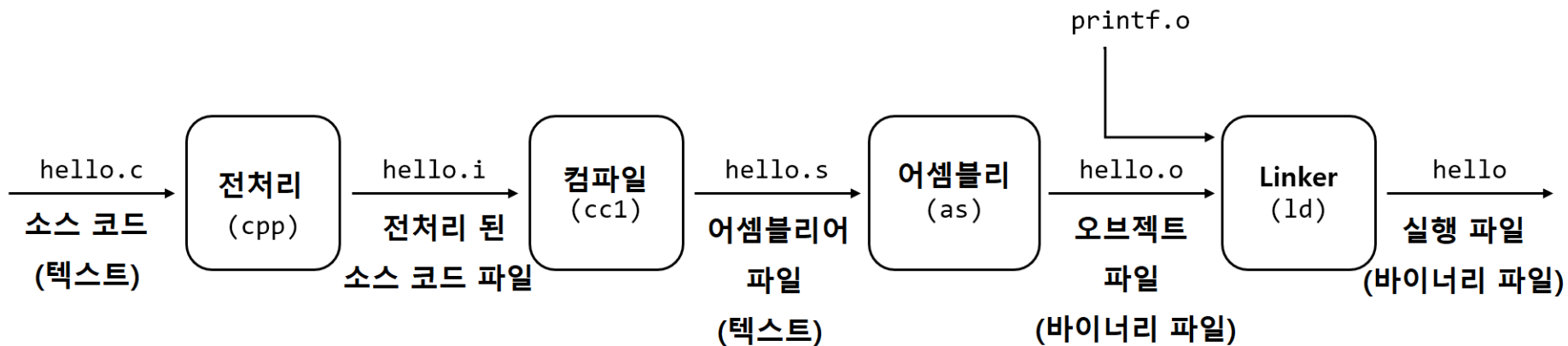
# 목차

- C언어 컴파일 과정
  - 컴파일 환경
  - 전처리 단계
  - 컴파일 단계
  - 어셈블리 단계
  - 링크 단계
- Visual Studio에서의 C언어 컴파일 과정

# C언어 컴파일 과정



# C언어 컴파일 과정



# 컴파일 환경

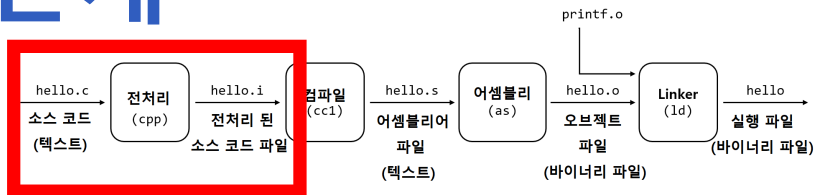
```
neofetch
OS: Arch Linux x86_64
Host: 2007501000 ThinkPad L15 Gen 1
Kernel: 6.3.9-arch1-1
Uptime: 59 mins
Packages: 804 (pacman)
Shell: zsh 5.9
Resolution: 1920x1080
WM: i3
Theme: Adwaita [GTK2/3]
Icons: Adwaita [GTK2/3]
Terminal: terminator
CPU: AMD Ryzen 7 PRO 4750U with Rad
GPU: AMD ATI 700:00.0 Renoir
Memory: 1988MiB / 15355MiB
Disk (/): 9.6G / 63G (17%)

gcc --version
gcc (GCC) 13.1.1 20230429
Copyright (c) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

## <컴파일 환경>

- OS : 리눅스
- CPU : AMD Ryzen
- Compiler : GCC 13

# 전처리 단계

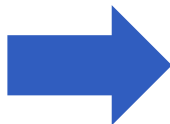


```
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");

    return 0;
}
```

gcc -E hello.c -o hello.i

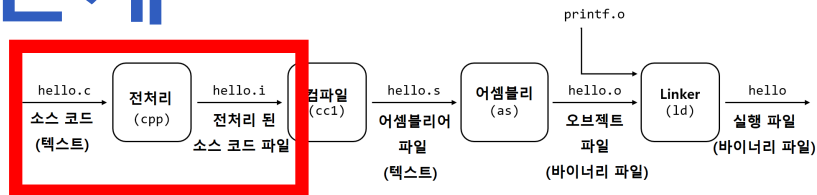


```
typedef struct _IO_FILE FILE;
# 43 "/usr/include/stdio.h" 2 3 4
# 1 "/usr/include/bits/types/struct_FILE.h" 1 3 4
# 35 "/usr/include/bits/types/struct_FILE.h" 3 4
struct _IO_FILE;
struct _IO_marker;
struct _IO_codecvt;
struct _IO_wide_data;

extern void funlockfile (FILE *__stream)
__attribute__ ((__nothrow__ , __leaf__));
# 885 "/usr/include/stdio.h" 3 4
extern int __uflow (FILE *);
extern int __overflow (FILE *, int);
# 909 "/usr/include/stdio.h" 3 4

# 2 "hello.c" 2
```

# 전처리 단계



```
typedef struct _IO_FILE FILE;
# 43 "/usr/include/stdio.h" 2 3 4
# 1 "/usr/include/bits/types/struct_FILE.h" 1 3 4
# 35 "/usr/include/bits/types/struct_FILE.h" 3 4
struct _IO_FILE;
struct _IO_marker;
struct _IO_codecvt;
struct _IO_wide_data;

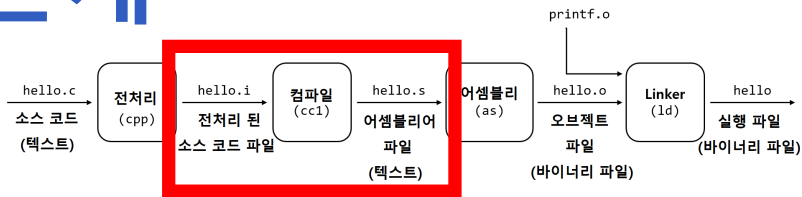
extern void funlockfile (FILE *__stream)
__attribute__ ((__nothrow__ , __leaf__));
# 885 "/usr/include/stdio.h" 3 4
extern int __uflow (FILE *);
extern int __overflow (FILE *, int);
# 909 "/usr/include/stdio.h" 3 4

# 2 "hello.c" 2
```

- 전처리(cpp)는 본래의 C프로그램을 #문자로 시작하는 디렉티브(directive)에 따라 수정한다.
- 예를 들어 hello.c파일 첫 줄의 #include<stdio.h>는 전처리기에게 시스템 헤더파일인 stdio.h를 프로그램 문장에 직접 삽입하라고 지시한다.
- 그 결과 일반적으로 \*.i로 끝나는 새로운 C 프로그램이 생성된다.



# 컴파일 단계

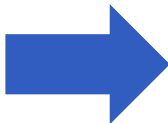


```
typedef struct _IO_FILE FILE;
# 43 "/usr/include/stdio.h" 2 3 4
# 1 "/usr/include/bits/types/struct_FILE.h" 1 3 4
# 35 "/usr/include/bits/types/struct_FILE.h" 3 4
struct _IO_FILE;
struct _IO_marker;
struct _IO_codecvt;
struct _IO_wide_data;

extern void funlockfile (FILE *__stream)
__attribute__ ((__nothrow__ , __leaf__));
# 885 "/usr/include/stdio.h" 3 4
extern int __uflow (FILE *);
extern int __overflow (FILE *, int);
# 909 "/usr/include/stdio.h" 3 4

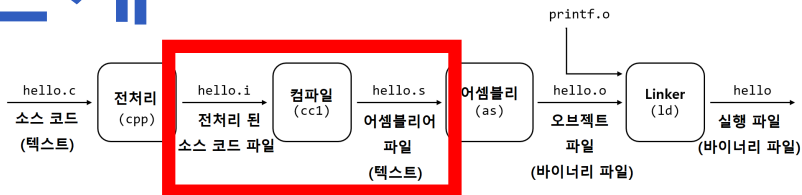
# 2 "hello.c" 2
```

gcc -S hello.c



```
.file "hello.c"
.text
.section .rodata
.LC0:
.string "Hello, world!"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
```

# 컴파일 단계



```
.file "hello.c"
.text
.section .rodata
.LC0:
.string "Hello, world!"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
```

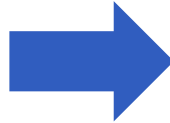
- 컴파일러(cc1)은 **hello.i**를 **hello.s**로 번역하며, 이 파일에는 어셈블리어 프로그램이 저장된다.
- 어셈블리어는 여러 상위수준 언어의 컴파일러들을 위한 공통의 출력언어를 제공한다.
- 그래서 C나 Fortran 등의 컴파일러에서는 둘 다 동일한 어셈블리어로 출력이 된다.

# 어셈블리 단계



```
.file "hello.c"
.text
.section .rodata
.LC0:
.string "Hello, world!"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
```

gcc -c hello.c



```
hello.o - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
[ELF] rr r > r X1 @ @ # ^
UH 쫄H? H뽁? ? ] 쫄ello, world! GCC: (GNU)
13.1.1 20230429 J | GNU 1 r? r r? r
  ll rzR r+r+?•? → A#?C
-U?•  r J? L r
  L | ↓ r → # †
  hello.c main puts • 1 L ?
  * J | ? 1 1
  .symtab .strtab .shstrtab .rela.text .data .bss .rodata
.comment .note.GNU-stack .note.gnu.property
.rela.eh_frame
  r - @ → r
  ← J @ ? 0 & r # ↑
  & r L Z Z r ,
  L Z Z r 1 r
  1 Z # r 9 r 0
  h r r B r
```

# 어셈블리 단계



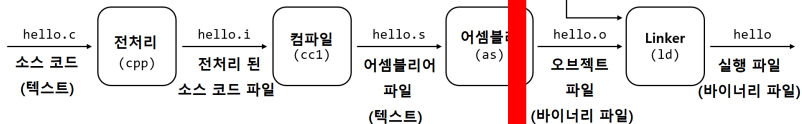
hello.o - Windows 메모장

```
ELF rrr r > r X1 @ @ # ^
UH될H? H뵈? ? ]흠ello, world! GCC: (GNU)
13.1.1 20230429 J | GNU 1 r? r r? r
|| rzR r+r+? → A#?C
-U?
L | ↓ r + # +
hello.c main puts • 1 L ?
# J | ? 1 1
.symtab .strtab .shstrtab .rela.text .data .bss .rodata
.comment .note.GNU-stack .note.gnu.property
.rela.eh_frame
r - @ → r
+ J @ ? 0 8 r # ↑
& r L Z r ,
L Z r 1 r
1 h Z # r 9 r 0
h r r B r
```

Ln 1, Col 1 | 90% | Macintosh (CR) | ANSI

- 어셈블리(as)가 **hello.s**를 기계어로 번역하고, 이들을 재배치가 가능 목적프로그램의 형태로 묶어서 **hello.o**라는 목적파일에 그 결과를 저장한다.
- 이 파일은 main함수의 인스트럭션들을 인코딩하기 위한 17바이트를 포함하는 바이너리 파일이다.
- hello.o 파일을 텍스트 편집기로 열어보면 문자가 깨져서 보일 것이다.

# 링크 단계



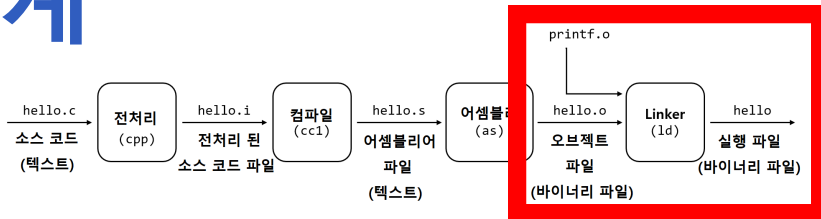
```
hello.o - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
|ELF rrr r > r X1 @ @ #
UH될H? H됨? ? ]홀ello, world! GCC: (GNU)
13.1.1 20230429 | GNU r? r r? r
| rzR r+r+? → A#?C
-U?
L | r J? L r
L | ↓ r + # +
hello.c main puts • 1 L ?
# J | ? 1 1
.symtab .strtab .shstrtab .rela.text .data .bss .rodata
.comment .note.gnu-stack .note.gnu.property
.rela.eh_frame
r - @ → r ↑
+ J @ ? 0 # r # ↑
& r L Z r r ,
L L Z r 1 r
h Z # r r B r
Ln 1, Col 1 90% Macintosh (CR) ANSI
```

gcc hello.c -o hello



```
hello - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
|ELF rrr L > r @+ @ ?
@ 8
@ - J @ @ @ ? ?
L J ↑L ↑L ↑L
r J + + + ar ar +
r J ? ? + r
- ? ? ? ? H P + r -
? ? ? ? ? ? # J J 8L
8L 8L @ @ J J xL
xL xL D D J S?dJ 8L
8L 8L @ @ # P?dJ #
# # $ $ J Q?d-
+ R?dJ ?
? ? 0 0 r /lib64/ld-linux-
x86-64.so.2 J 0 | GNU r? r r
Ln 2, Col 212 100% Macintosh (CR) ANSI
```

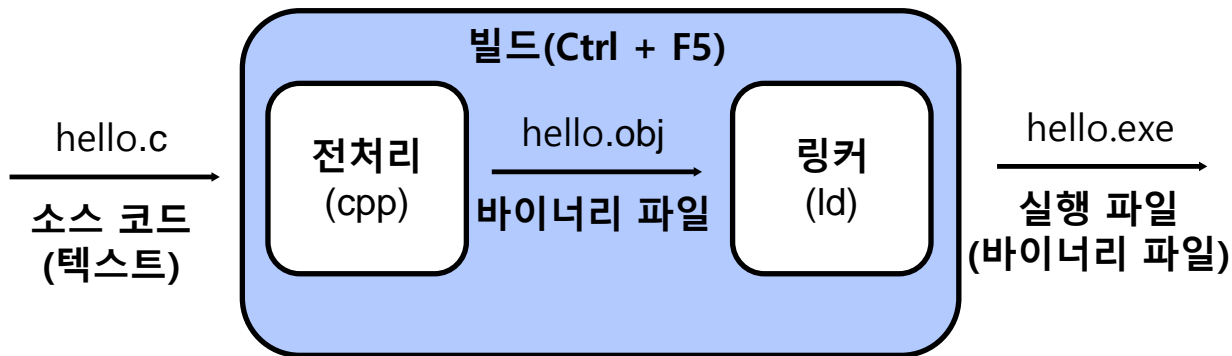
# 링크 단계



```
hello - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
@ELF_r_r L > r @+ @ ?
@ 8
@ - J @ @ @ ? ?
L J ↑L ↑L ↑L r
r J + + + ar ar +
r J ? ? + r
- ? ? ? H? P? + r -
? ? ? ? ? ? J J 8L
8L 8L @ @ J J xL
xL xL D D J S?dJ 8L
8L 8L @ @ P?dJ ¶
¶ ¶ $ $ J Q?d-
+ R?dJ ?
? ? 0? 0? r /lib64/ld-linux-
x86-64.so.2 J 0 | GNU r ? r
Ln 2, Col 212 100% Macintosh (CR) ANSI
```

- 라이브러리 함수와 여러 오브젝트 파일들을 연결해서 최종 실행 파일을 생성한다.
- hello 파일은 실행가능 목적파일(즉, 실행파일)로 적재되어 시스템에 의해 실행.
- 윈도우에서의 결과물은 코드와 데이터를 포함하는 실행 가능한 바이너리 파일인 hello.exe이다.

# Visual Studio에서의 C언어 컴파일 과정



# Visual Studio에서의 C언어 컴파일 과정

The image illustrates the compilation process in Visual Studio. It features several overlapping windows:

- main.c**: Shows the source code with `#include "bar.h"` and `foo.h`, and a `main` function that calls `foo()` and `bar()`.
- File Explorer**: Displays the project files, with `main.obj` and `bar.obj` highlighted in red boxes, indicating they are the focus of the assembly view.
- main.obj - Windows 메모장**: Shows the assembly code for `main.obj`, including directives like `.drectve` and `.debug$S`.
- foo.obj - Windows 메모장**: Shows the assembly code for `foo.obj`, including directives like `.drectve` and `.debug$S`.
- bar.obj - Windows 메모장**: Shows the assembly code for `bar.obj`, including directives like `.drectve` and `.debug$S`.

At the bottom, a large blue arrow points to the text **직접 확인해봄** (Check it out directly).





**END**