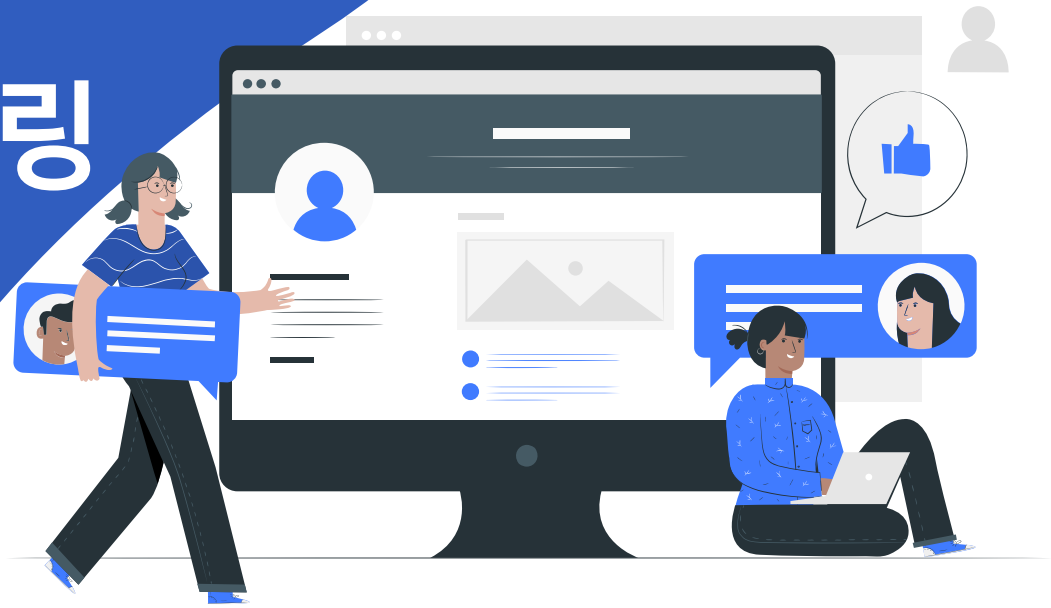
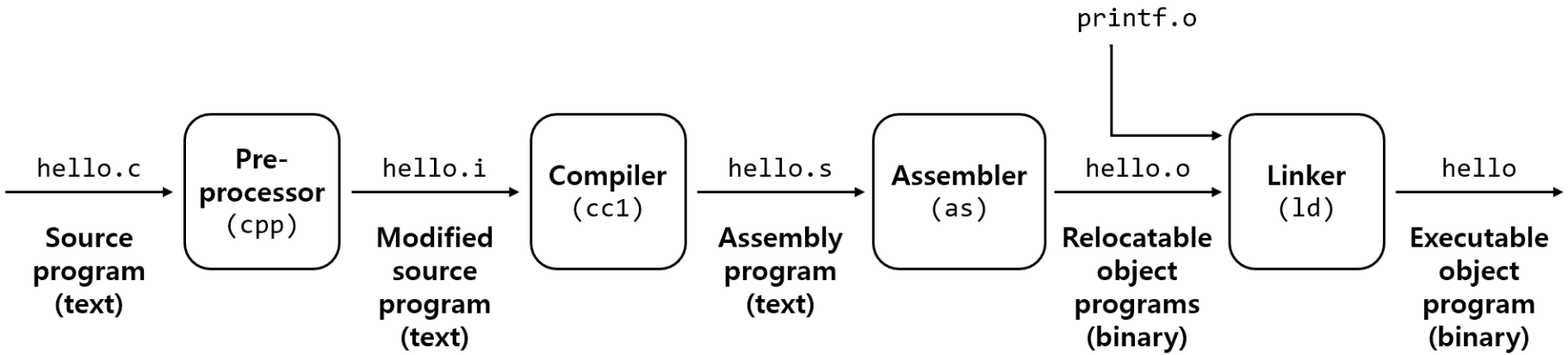
리눅스 튜터링



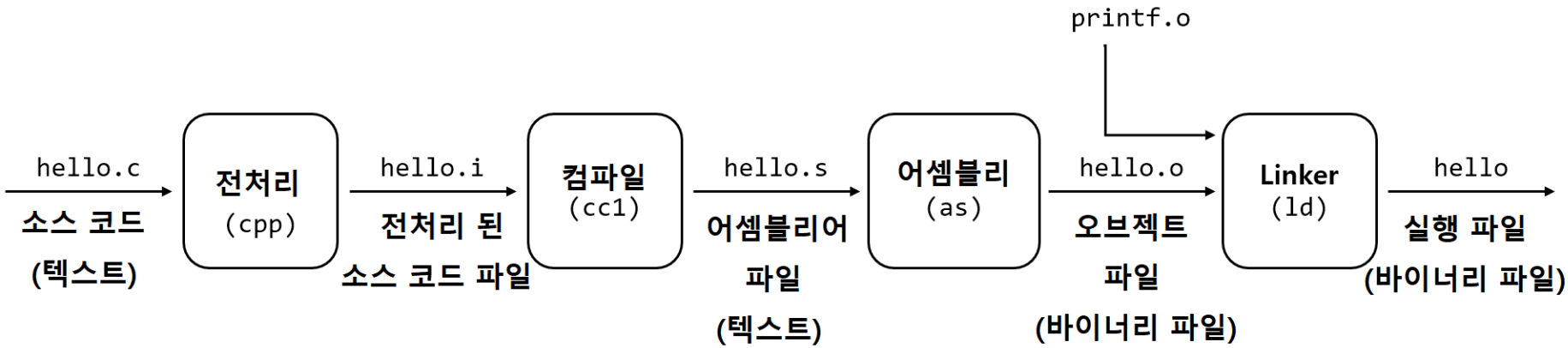
01. C언어 컴파일 과정

목차

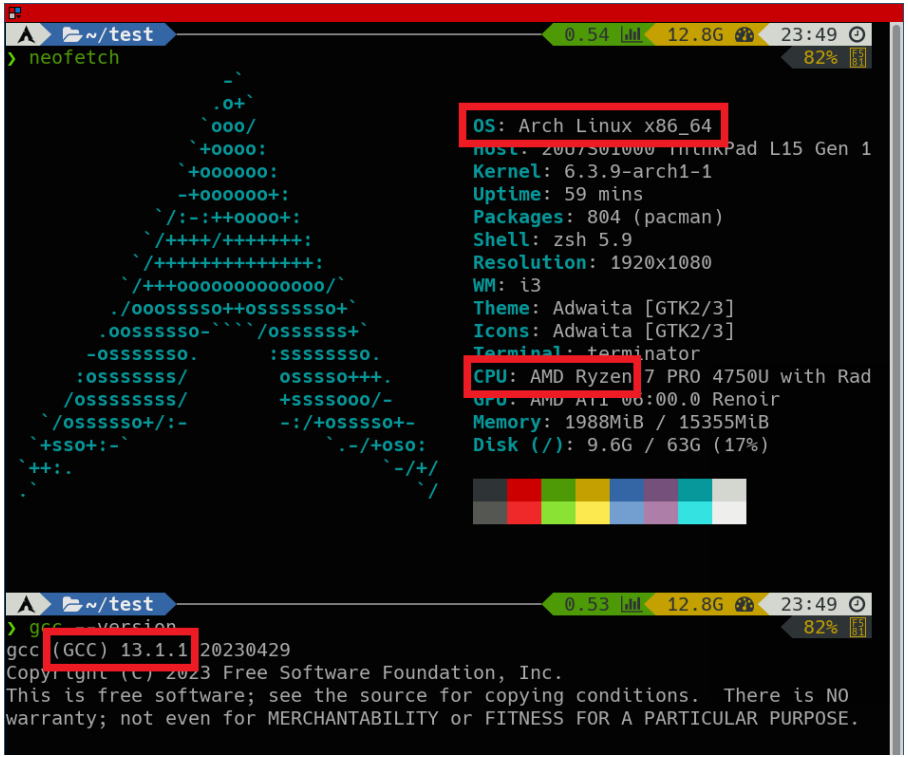
C언어 컴파일 과정


printf.o
hello.c
Source program (text)
Pre-processor (cpp)
hello.i
Modified source program (text)
Compiler (cc1)
hello.s
Assembly program (text)
Assembler (as)
hello.o
Relocatable object programs (binary)
Linker (ld)
hello
Executable object program (binary)

C언어 컴파일 과정


printf.o
hello.c
소스 코드
(텍스트)
전처리
(cpp)
hello.i
전처리 된
소스 코드 파일
컴파일
(cc1)
hello.s
어셈블리어
파일
(텍스트)
어셈블리
(as)
hello.o
오브젝트
파일
(바이너리 파일)
Linker
(ld)
hello
실행 파일
(바이너리 파일)

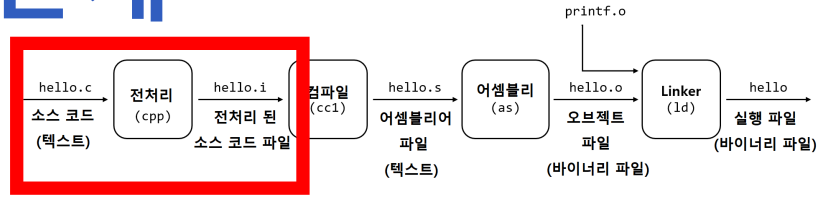
컴파일 환경



<컴파일 환경>

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

전처리 단계

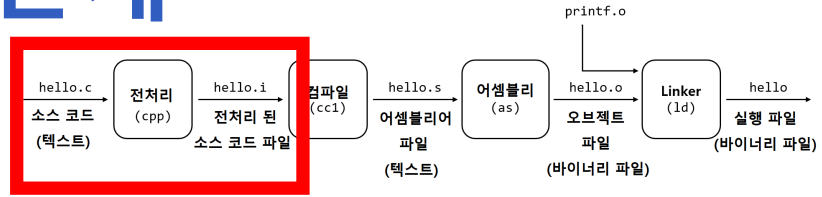



```c
#include <stdio.h>

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

    return 0;
}
```

gcc -E hello.c -o hello.i

```c
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
```

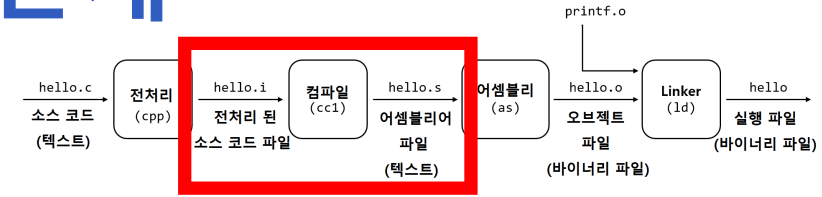
전처리 단계




```
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

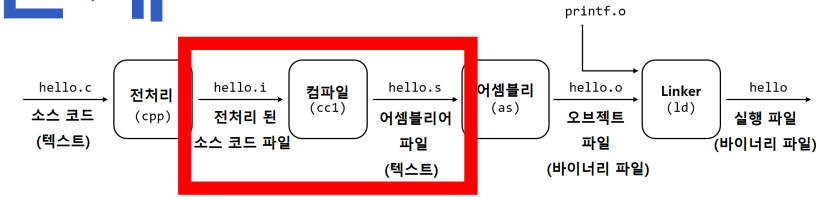



```c
typedef struct _IO_FILE FILE;
# 43 "/usr/include/stdio.h" 2 3 4
# 1 "/usr/include/bits/types/struct_FILE.h" 1 3 4
# 35 "/usr/include/bits/types/struct_FILE.h" 3 4
struct _IO_FILE;
struct _IO_marker;
struct _IO_codecvt;
struct _IO_wide_data;

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



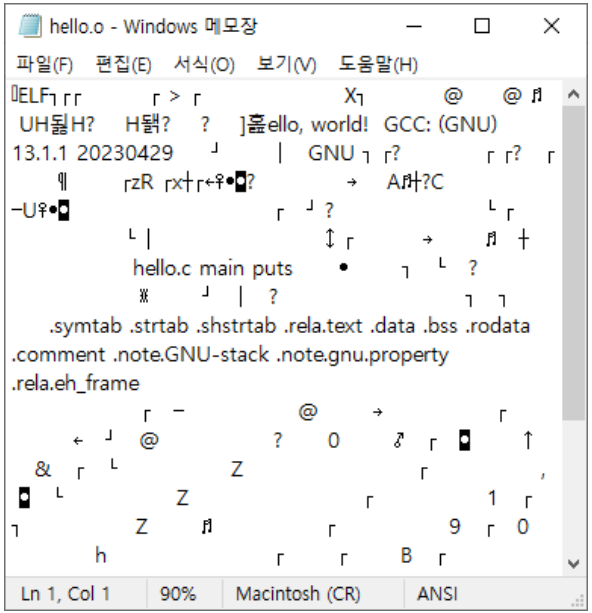

```
    .file   "hello.c"
    .text
    .section    .rodata
.LC0:
    .string "Hello, world!"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
```

- 컴파일러(cc1)은 **hello.i를 hello.s로 번역하며, 이 파일에는 어셈블리어 프로그램이 저장된다.**
- 어셈블리어는 여러 상위수준 언어의 컴파일러들을 위한 공통의 출력언어를 제공한다.
- 그래서 **C나 Fortran 등 의 컴파일러에서는 둘 다 동일한 어셈블리어로 출력**이 된다.

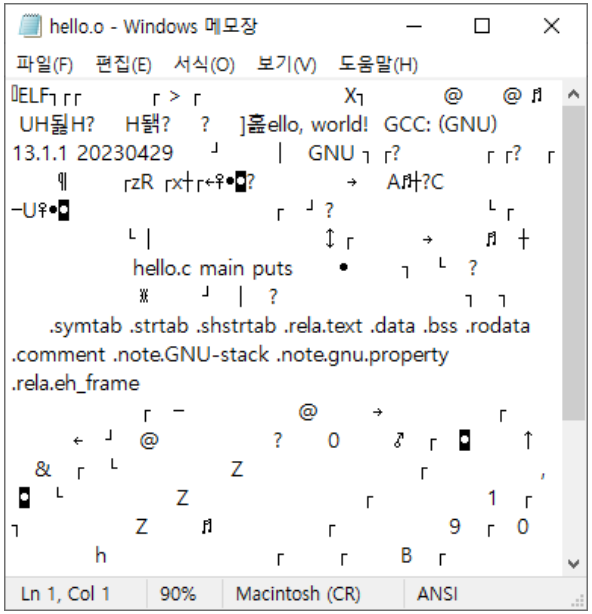
어셈블리 단계


```
    .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

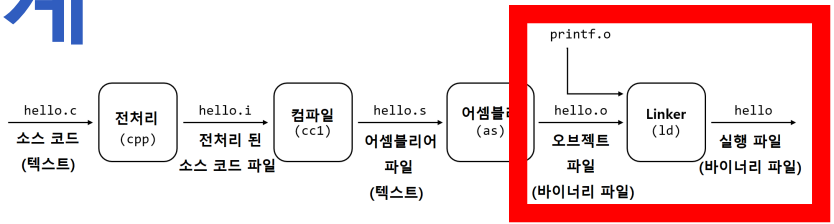


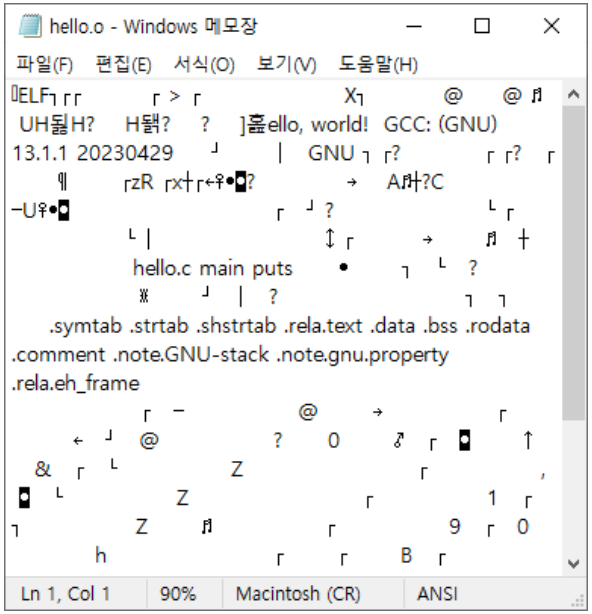
어셈블리 단계



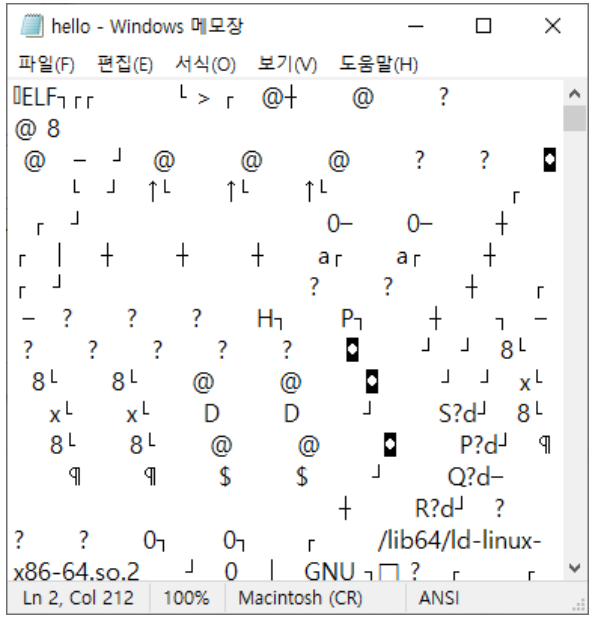
- 어셈블리(as)가 **hello.s를 기계어로 번역**하고, **이들을 재배치가능 목적프로그램의 형태로 묶어서 hello.o라는 목적파일에 그 결과를 저장한다**.
- 이 파일은 main함수의 인스트럭션들을 인코딩하기 위한 17바이트를 포함하는 바이너리 파일이다.
- hello.o 파일을 텍스트 편집기로 열어보면 문자가 깨져서 보일것이다.

링크 단계

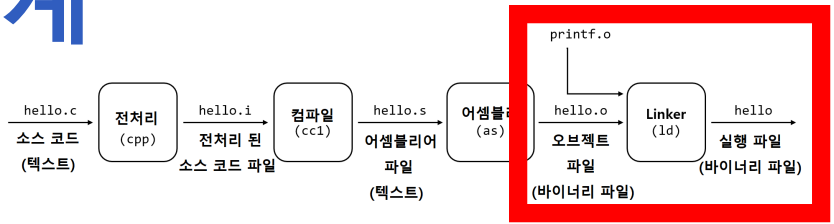


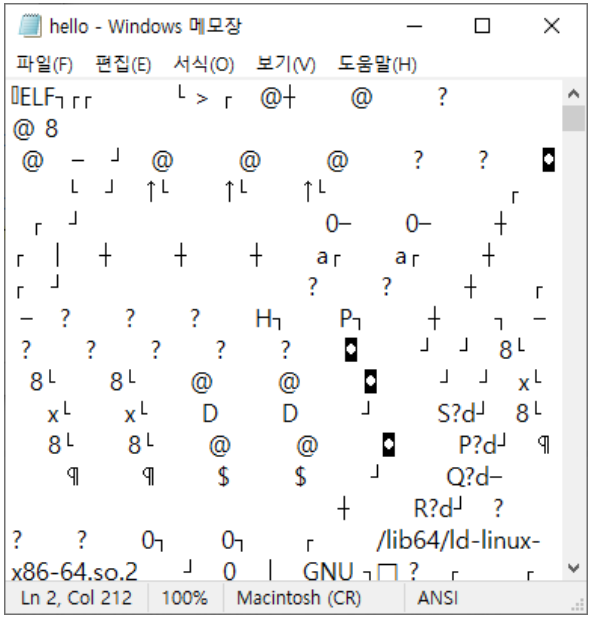


gcc hello.c -o hello



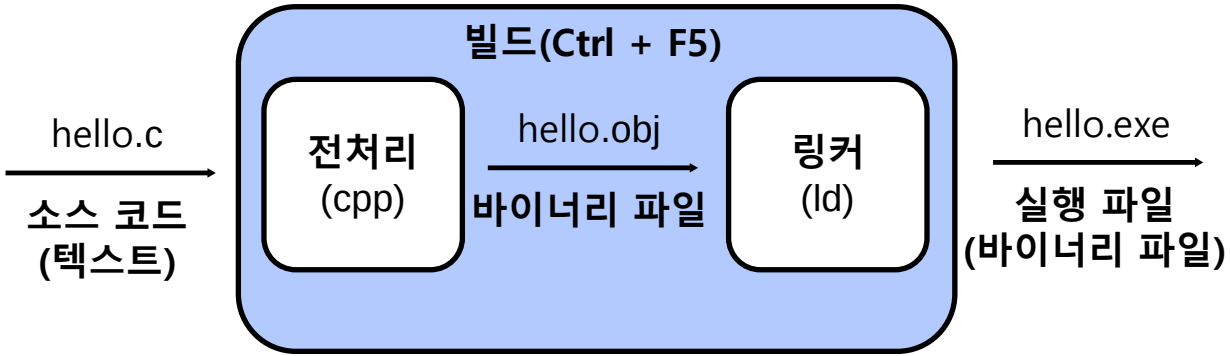
링크 단계



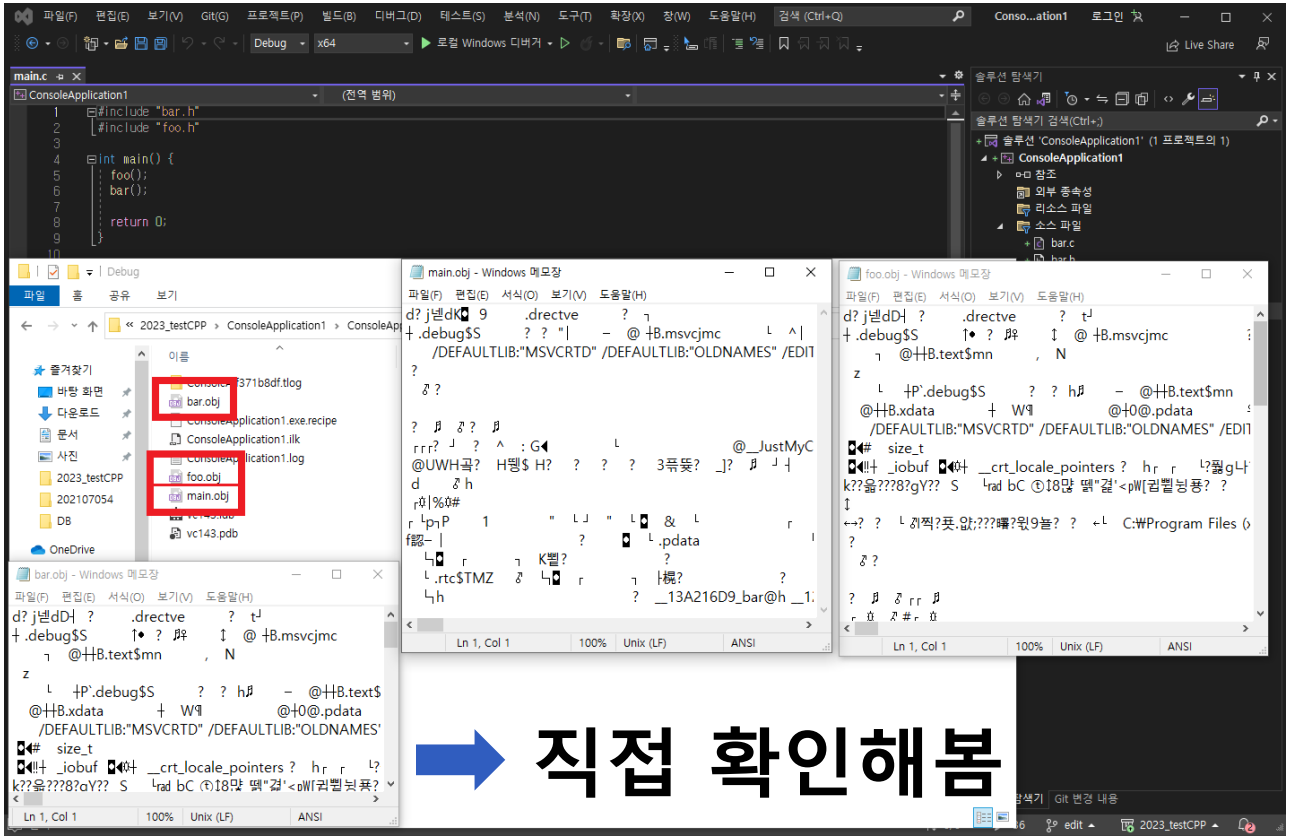


- **라이브러리 함수와 여러 오브젝트 파일들을 연결해서 최종 실행 파일을 생성한다.**
- **hello 파일은 실행가능 목적파일(즉, 실행파일)로 적재되어 시스템에 의해 실행.**
- 윈도우에서의 결과물은 코드와 데이터를 포함하는 실행 가능한 바이너리 파일인 hello.exe이다.

Visual Studio에서의 C언어 컴파일 과정



Visual Studio에서의 C언어 컴파일 과정



END