



COMPILATION PROCESS IN C PROGRAM

Faculty of Information Technology
University of Moratuwa

What is Compilation?

```
#include <stdio.h>

int main() {
    printf("Hello world");
    return 0;
}
```

Source code

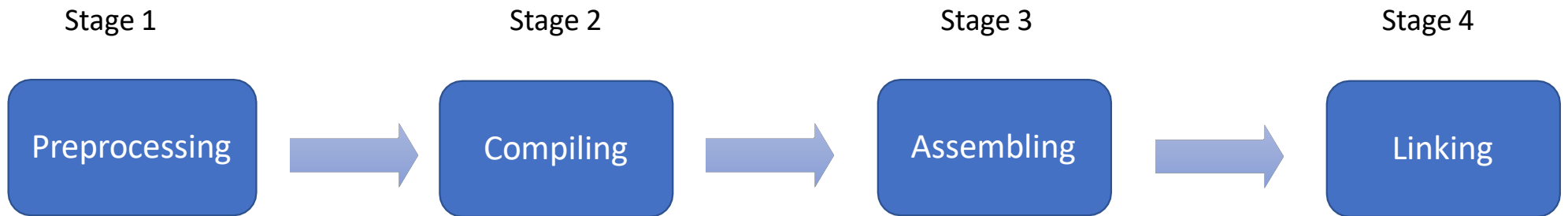


```
110101010101010010100010
100101111000001000100101
010101010101001011100011
010001010010101110001001
010101010000111110101010
010101010101010100100011
```

Machine code



4 stages of compilation process



Stage 1 - Preprocessing

- Preprocessor take care of this stage.
- Preprocessor goes through the entire source code and :
 - ✓ Remove comments
 - ✓ Expand macros
 - ✓ Expand included files
- Source code get expanded.
- Command : **gcc hello.c -E -o hello.i**
- Output file : hello.i

```
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _vprintf(const wchar_t *Format, locale_t _locale, va_list _Arglist);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _vprintf_p(const wchar_t *Format, locale_t _locale, va_list _Arglist);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _wprintf(FILE *File, const wchar_t *Format, locale_t _locale, ...);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _wprintf_p(FILE *File, const wchar_t *Format, locale_t _locale, ...);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _vfwprintf(FILE *File, const wchar_t *Format, locale_t _locale, va_list _Arglist);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _vfwprintf_p(FILE *File, const wchar_t *Format, locale_t _locale, va_list _Arglist);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _swprintf(const wchar_t *DestBuf, size_t _MaxCount, const wchar_t *Format, locale_t _locale, ...);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _swprintf_c(const wchar_t *DestBuf, size_t _MaxCount, const wchar_t *Format, locale_t _locale, va_list _Arglist);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _vswprintf(const wchar_t *DestBuf, size_t _MaxCount, const wchar_t *Format, locale_t _locale, va_list _Arglist);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _vswprintf_c(const wchar_t *DestBuf, size_t _MaxCount, const wchar_t *Format, locale_t _locale, va_list _Arglist);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _scprintf(const wchar_t *Format, locale_t _locale, ...);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _scprintf_p(const wchar_t *Format, locale_t _locale, va_list _Arglist);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _vsnprintf(const wchar_t *DestBuf, size_t _MaxCount, const wchar_t *Format, locale_t _locale, va_list _Arglist);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _vsnprintf_p(const wchar_t *DestBuf, size_t _MaxCount, const wchar_t *Format, locale_t _locale, va_list _Arglist);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _snprintf(const wchar_t *Dest, const wchar_t *Format, locale_t _locale, va_list _Args);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _vsnprintf(const wchar_t *Dest, const wchar_t *Format, locale_t _locale, va_list _Args);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _fscanf(FILE *File, const wchar_t *Format, locale_t _locale, ...);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _wscanf(const wchar_t *Src, const wchar_t *Format, locale_t _locale, ...);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _swscanf(const wchar_t *Src, size_t _MaxCount, const wchar_t *Format, locale_t _locale, ...);
__attribute__((__declspec__)) int __attribute__((__cdecl__)) _wscanf(const wchar_t *Src, size_t _MaxCount, const wchar_t *Format, locale_t _locale, ...);

__attribute__((__declspec__)) size_t __attribute__((__cdecl__)) _fread_nolock_s(void *DstBuf, size_t _DstSize, size_t _ElementSize, size_t _Count, FILE *File);
# 1398 "C:/Program Files/CodeBlocks/MingW/x86_64-mingw32/include/stdio.h" 2 3
# 1 "C:/Program Files/CodeBlocks/MingW/x86_64-mingw32/include/mingw_print_pop.h" 1 3
# 1400 "C:/Program Files/CodeBlocks/MingW/x86_64-mingw32/include/stdio.h" 2 3
# 2 "hello.c" 2

# 3 "hello.c"
int main()
{
    printf("hello world");
}
```



Stage 2 - Compiling

- C Compiler is the tool used in here.
- Checks for valid syntax of C language.
- If any wrong syntax is found throws an error.
- Converts the pre-processed code into assembly code.
- Command : `gcc hello.i -S -o hello.s`
- Output file : `hello.s`

```
.file "hello.c"
.text
.def __main; .scl 2; .type 32; .endif
.section .rdata,"dr"
.LC0:
.ascii "Hello World\0"
.text
.globl main
.def main; .scl 2; .type 32; .endif
.seh_proc main
main:
pushq %rbp
.seh_pushreg %rbp
movq %rsp, %rbp
.seh_setframe %rbp, 0
subq $32, %rsp
.seh_stackalloc 32
.seh_endprologue
call __main
leaq .LC0(%rip), %rcx
call printf
movl $0, %eax
addq $32, %rsp
popq %rbp
ret
.seh_endproc
.ident "GCC: (x86_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0"
.def printf; .scl 2; .type 32; .endif
```



Stage 3 - Assembling

- Assembler take care of this stage.
- Convert the assembly code into pure binary code or machine code.
- Also, the output code is known as the object code.
- Command : `gcc hello.s -c -o hello.o`
- Output file : `hello.o`

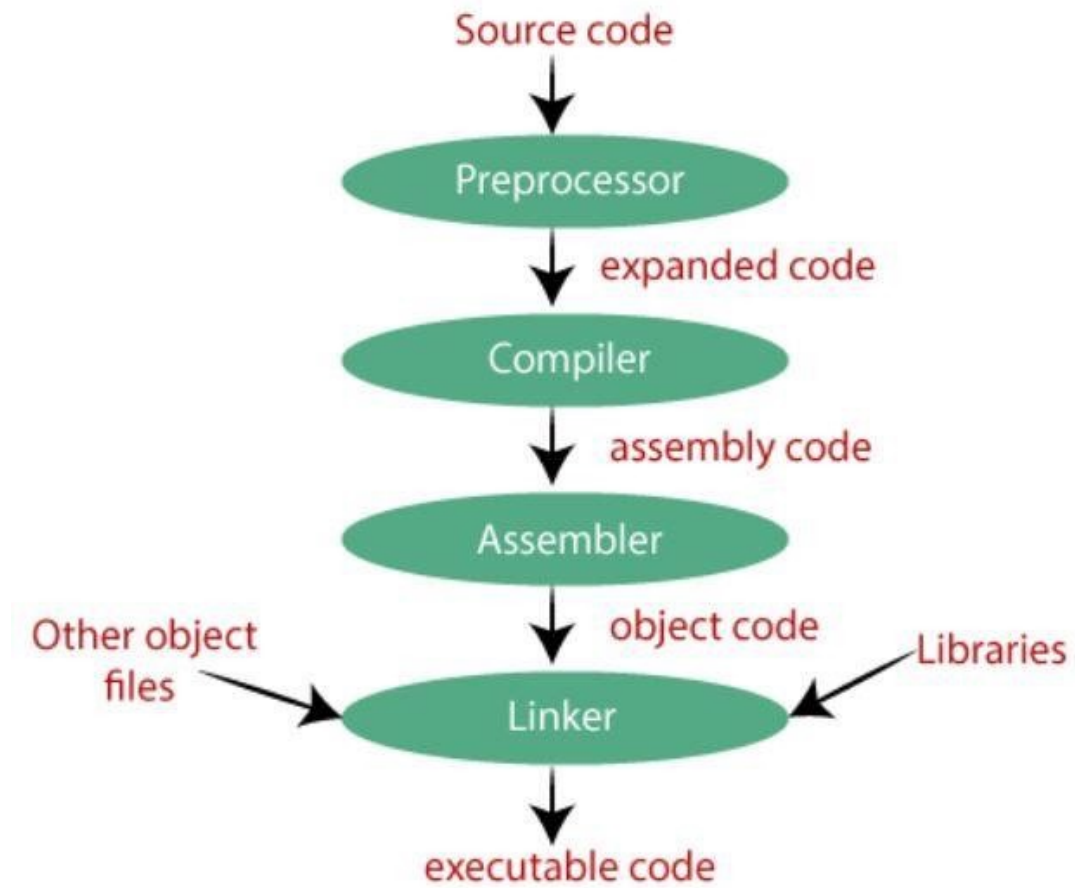
```
d+[] [] .text 0 ,[] Å[] [] P`.data
@ PÅ.bss
P@.xdata 1[] @ 0@.pdata x[] â[] [] @ 0@/4
è ,[] @ P@UH%âHfi è H
HfÅ ]ÅHello world []2[]P $ GCC: (x86_64-posix-seh-rev0, Built
by MinGW-W64 project) 8.1.0 [] []
[] [] [] [] [] [] [] [] [] .file pÿ g\hello.c main []
[] .text [] []$ [] .data [] .xdata [] []
.bss [] [] .rddata [] []
.pdata [] [] .rddata$zzz .rddata$zzz
printf [] [] ? [] _main
```



Stage 4 - Linking

- Linking is the final step of compilation.
- The linker merges all the object code from multiple modules into a single one.
- Produces final executable file.
- Command : `gcc hello.c -o hello.exe`
- Output file : hello.exe





Does the executable file depend on OS?

Refer the link : <https://www.quora.com/Are-executable-files-exe-generated-by-compilers-machine-specific-and-or-operating-system-specific-In-other-words-can-exe-files-be-ported-from-one-machine-to-another-having-the-same-or-different-OS>

Does assembly languages depend on OS?

Refer the link : <https://stackoverflow.com/questions/6859348/how-do-assembly-languages-depend-on-operating-systems>

Are compilers only operating system dependent and not hardware dependent?

Refer the link : <https://www.quora.com/Why-are-compilers-only-operating-system-dependent-and-not-hardware-dependent>

Do all programming languages compile to the same machine code?

Refer the link : <https://www.quora.com/Do-all-programming-languages-compile-to-the-same-machine-code>

