############## Compilation process in C - Overview ##############

The compilation process in C transforms a human-readable code into a machine-readable format. For C programming language, it happens before a program starts executing to check the syntax and semantics of the code. The compilation process in C involves four steps: pre-processing, compiling, assembling, and linking then, we run the obtained executable file to get an output on the screen.
What is a Compilation?

Before diving into the traditional definition of compilation, let us consider an example where there is a person A who speaks Hindi language and person A wants to talk to person B who only knows English language, so now either of them requires a translator to translate their words to communicate with each other. This process is known as translation, or in terms of programming, it is known as compilation process.

The compilation process in C is converting an understandable human code into a Machine understandable code and checking the syntax and semantics of the code to determine any syntax errors or warnings present in our C program. Suppose we want to execute our C Program written in an IDE (Integrated Development Environment). In that case, it has to go through several phases of compilation (translation) to become an executable file that a machine can understand.

Compilation process in C involves four steps:

- Preprocessing
- Compiling
- Assembling
- Linking

Compilation process in C involves four steps:

############################## a. Pre-Processing ##############################

Pre-processing is the first step in the compilation process in C performed using the pre-processor tool (A pre-written program invoked by the system during the compilation). All the statements starting with the # symbol in a C program are processed by the pre-processor, and it converts our program file into an intermediate file with no # statements. Under following pre-processing tasks are performed :

############### i. Comments Removal ###############

Comments in a C Program are used to give a general idea about a particular statement or part of code actually, comments are the part of code that is removed during the compilation process by the pre-processor as they are not of particular use for the machine. The comments in the below program will be removed from the program when the pre-processing phase completes.

############### ii. Macros Expansion ###############

Macros are some constant values or expressions defined using the #define directives in C Language. A macro call leads to the macro expansion. The pre-processor creates an intermediate file where some pre-written assembly level instructions replace the defined expressions or constants (basically matching tokens). To differentiate between the original instructions and the assembly instructions resulting from the macros expansion, a + sign is added to every macros expanded statement.

############### iii. File inclusion ###############

File inclusion in C language is the addition of another file containing some pre-written code into our C Program during the pre-processing. It is done using the #include directive. File inclusion during pre-processing causes the entire content of filename to be added to the source code, replacing the #include<filename> directive, creating a new intermediate file.

Example: If we have to use basic input/output functions like printf() and scanf() in our C program, we have to include a pre-defined standard input output header file i.e. stdio.h.

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

############## iv. Conditional Compilation ##############

Conditional compilation is running or avoiding a block of code after checking if a macro is defined or not (a constant value or an expression defined using #define). The preprocessor replaces all the conditional compilation directives with some pre-defined assembly code and passes a newly expanded file to the compiler. Conditional compilation can be performed using commands like #ifdef, #endif, #ifndef, #if, #else and #elif in a C Program. Example :

Printing the AGE macro, if AGE macro is defined, else printing Not Defined and ending the conditional compilation block with an #endif directive.

```
#include <stdio.h>

// if we uncomment the below line, then the program will print AGE in the output.
// #define AGE 18

int main()
{
   // if `AGE` is defined then print the `AGE` else print "Not Defined"
   #ifdef AGE
      printf("Age is %d", AGE);
   #else
      printf("Not Defined");
   #endif

   return 0;
}
```

############################## b. Compiling ##############################

Compiling phase in C uses an inbuilt compiler software to convert the intermediate (.i) file into an Assembly file (.s) having assembly level instructions (low-level code). To boost the performance of the program C compiler translates the intermediate file to make an assembly file.

Assembly code is a simple English-type language used to write low-level instructions (in micro-controller programs, we use assembly language). The whole program code is parsed (syntax analysis) by the compiler software in one go, and it tells us about any syntax errors or warnings present in the source code through the terminal window.

############################## c. Assembling ##############################

Assembly level code (.s file) is converted into a machine-understandable code (in binary/hexadecimal form) using an assembler. Assembler is a pre-written program

that translates assembly code into machine code. It takes basic instructions from an assembly code file and converts them into binary/hexadecimal code specific to the machine type known as the object code.

The file generated has the same name as the assembly file and is known as an object file with an extension of .obj in DOS and .o in UNIX OS.

An assembly file area.s is translated to an object file area.o having the same name but a different extension.

############################## d. Linking ##############################

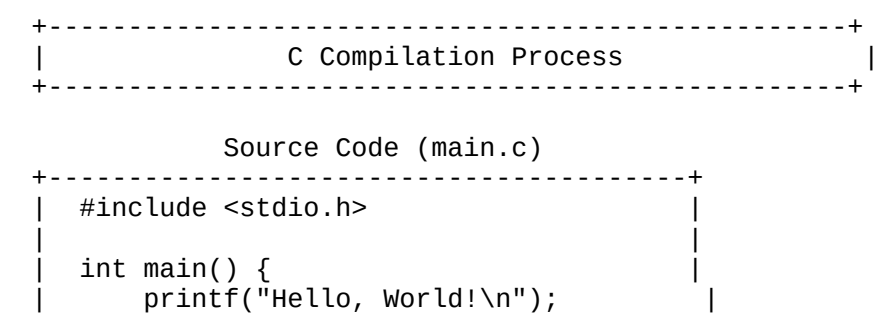
Linking is a process of including the library files into our program. Library Files are some predefined files that contain the definition of the functions in the machine language and these files have an extension of .lib. Some unknown statements are written in the object (.o/.obj) file that our operating system cant understand. You can understand this as a book having some words that you dont know, and you will use a dictionary to find the meaning of those words. Similarly, we use Library Files to give meaning to some unknown statements from our object file. The linking process generates an executable file with an extension of .exe in DOS and .out in UNIX OS.

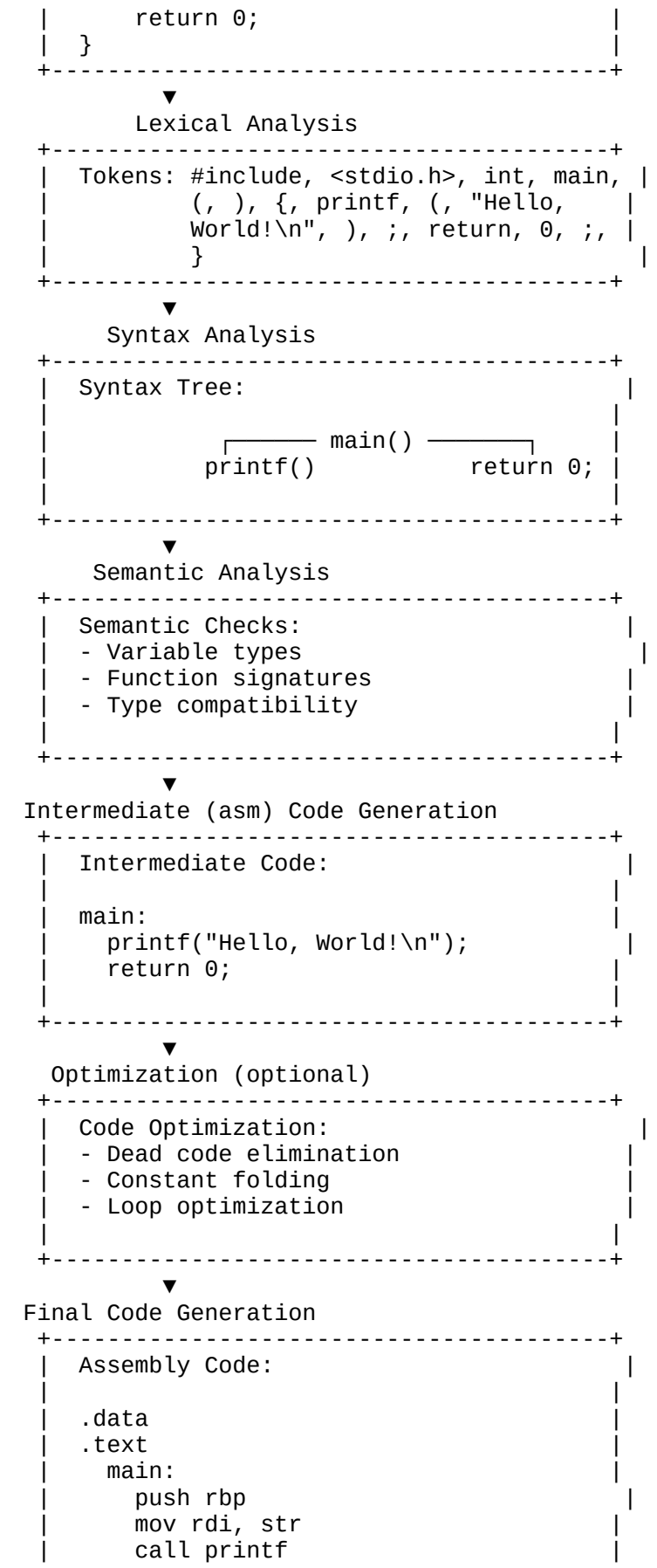
The below image shows an example of how the linking phase works, and we have an object file having machine-level code, it is passed through the linker which links the library files with the object file to generate an executable file.

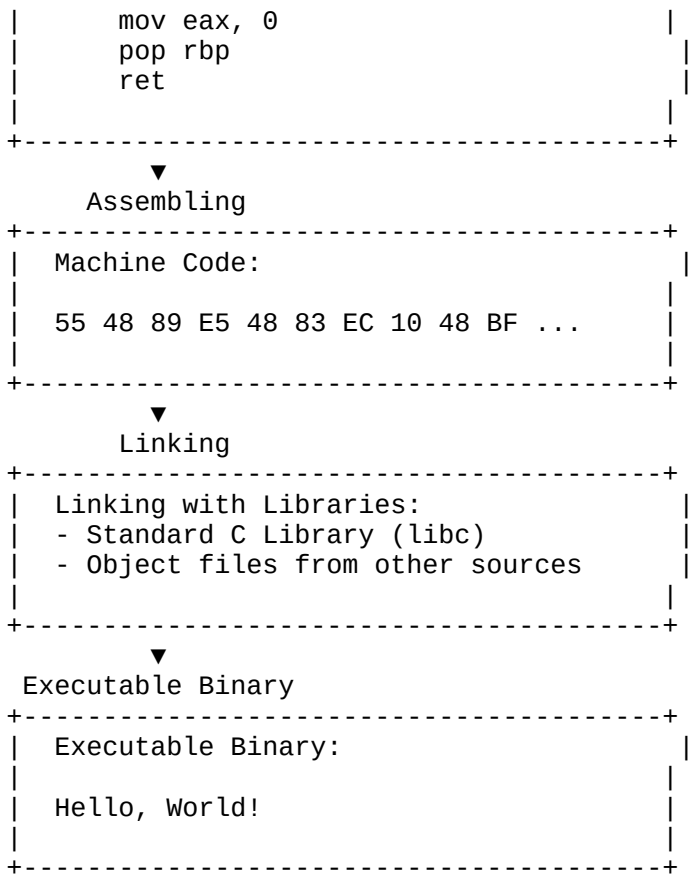
################## Compilation Flow Diagram ##################

We have a C Program file with an extension of .c i.e. hello.c file.
Step 1 is preprocessing of header files, all the statements starting with # (hash symbol) and comments are replaced/removed during the pre-processing with the help of a pre-processor. It generates an intermediate file with .i file extension i.e. a hello.i file.
Step 2 is a compilation of hello.i file. Compiler software translates the hello.i file to hello.s with assembly level instructions (low-level code).
Step 3, assembly-level code instructions are converted into machine-understandable code (binary/hexadecimal form) by the assembler. The file generated is known as the object file with an extension of .obj/.o i.e. hello.obj/hello.o file.
Step 4, Linker is used to link the library files with the object file to define the unknown statements. It generates an executable file with .exe/.out extension i.e. a hello.exe/hello.out file.
Next, we can run the hello.exe/hello.out executable file to get the desired output on our output window, i.e., Hello World!.

############################## Scheme ##############################




return 0;
}
Lexical Analysis
Tokens: #include, <stdio.h>, int, main, (,), {, printf, (, "Hello, World!\n",), ;, return, 0, ;, }
Syntax Analysis
Syntax Tree:
main()
printf()
return 0;
Semantic Analysis
Semantic Checks:
- Variable types
- Function signatures
- Type compatibility
Intermediate (asm) Code Generation
Intermediate Code:
main:
printf("Hello, World!\n");
return 0;
Optimization (optional)
Code Optimization:
- Dead code elimination
- Constant folding
- Loop optimization
Final Code Generation
Assembly Code:
.data
.text
main:
push rbp
mov rdi, str
call printf


mov eax, 0
pop rbp
ret
Assembling
Machine Code:
55 48 89 E5 48 83 EC 10 48 BF ...
Linking
Linking with Libraries:
- Standard C Library (libc)
- Object files from other sources
Executable Binary
Executable Binary:
Hello, World!