Assignment 2

Table of Contents

C Program Build Process

There four steps in the C program build process which converts C code into an executable file. Figure 1 shows all the steps involved in the C Program build process. It consists of four stages: (I) Pre-processor, (ii) Compiler, (iii) Assembler and (iv) Linker. The detailed description of each step is further given in this document.

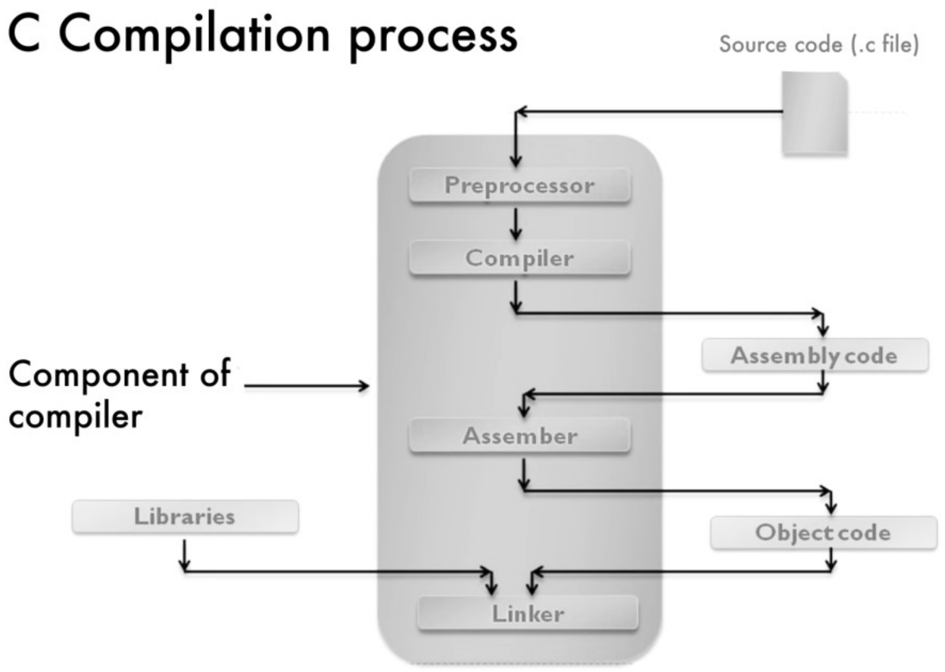


Fig. 1: C Program Build Process (taken from: https://www.linkedin.com/pulse/four-stages-compilastion-c-program-agustin-espinoza-saavedra/?trk=articles_directory)

1. Pre-Processing

Pre-processing is the first step towards building the executable file from C code file. It takes *.c file as an input and generates *.i file as an output. In this steps all the pre-processor directives are evaluated. Fig. 2 shows the graphical representation of the pre-processing step.

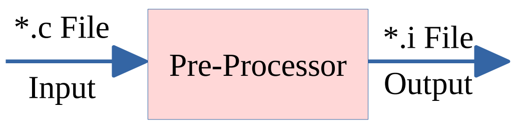


Fig. 2: Pre-Processor step

It substitutes the line starting with ‘#’ with their original code and generate a pure C code file. The generated file doesn’t have any preprocessor directive. E.g. if we have ‘#include<stdio.h>” line at the start of the C program then pre-processor step replaces this line by adding the code of ‘stdio.h’ header file.

The pre-processed file using gcc compiler can be obtained with the help of the following command:

$gcc -E hello.c > hello.i

2. Compilation

This step converts the pre-processed file to the assembly file. The converted assembly file is the architecture specific. Compiler takes the *.i file generated by pre-processor as an input file and generates the assembly file in *.s format. Figure 3 shows the compiler step.

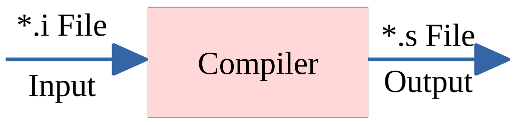


Fig. 3: Compiler step

The assembly code file can be obtained from *gcc* using the following command:

$gcc -S hello.c

There are basically two parts of the compiler: (i) Front End and (ii) Back End. Both parts of the compiler have pre-definced ste of tasks. Part of their work has been described here:

Front End: Analysis

This part of the compiler is also knows as the Analysis part and it does the analysis of program syntax and semantics. The analysis of the code is done in the following three steps:

1. **Lexical Analysis:** This stage converts the syntaxes into tokens by removing whitespaces and comments from the source code. Keywords, constants, identifiers, numbers, strings, operators etc. Are considered as the Tokens.
2. **Syntax Analysis:** It works closely with the lexical analyzer stage and generates syntax error is any problem is found in tokens as per the C rules.

3. **Semantic Analysis:** This stage checks the meaning of the syntex sentences written in the source code. It also generate semantic errors if problems are found. Some of the semantic errors are:
   a. type mismatch
   b. undeclared variable
   c. multiple declaration of a variable etc.

Once code is find symantically correct, it gets converted into intermediate representation which is very close to the assembly.

Another important part of the compiler is Symbol Table which is used by both Front End (Analysis) and Back End (Synthesis) parts. It is generally used to store the occurances of various entities such as function names, variable names, objects, classes , their scopes, types etc. A symbol table serves the following purposes

1. To store all the entities name at one place.
2. To verify the declaration of the variable.
3. To implement type checking, by verifying assignments and expressions in the source code.
4. To determine the scope of variables

Figure 4, effectively shows the different stages involved in the front end part of the compiler

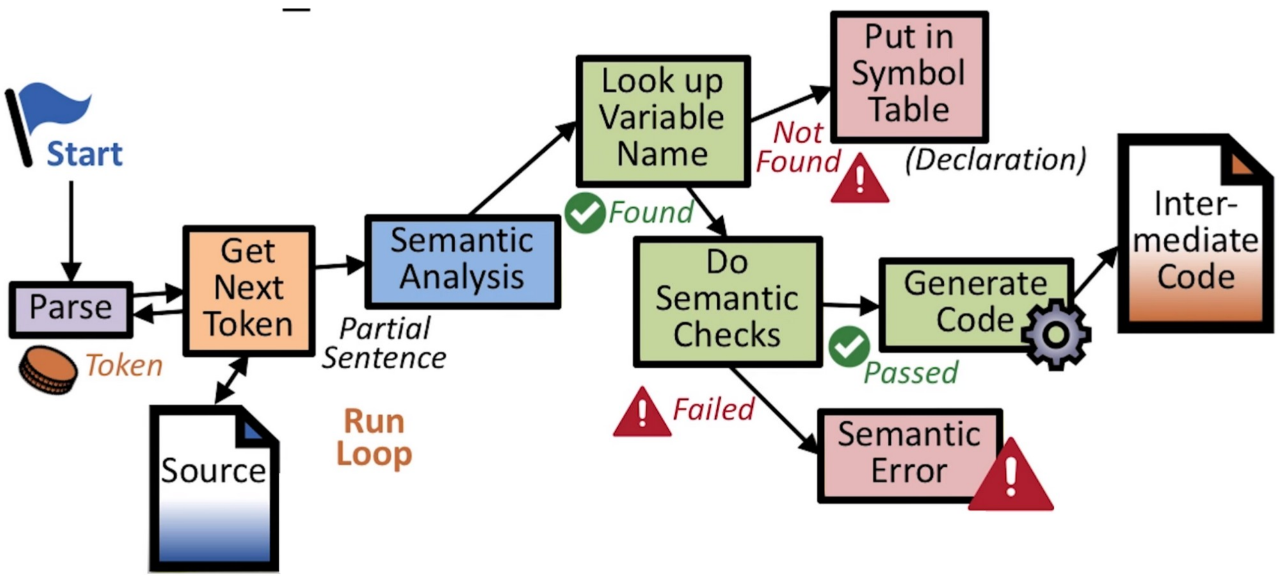


Fig. 4: Fron end working of the compiler

Back End: Synthesis

The stages in the back end part of the compiler are as follows:

1. **Optimization:** First stage of the back end part do the code optimization. It converts the code into functionally equivalent smaller and faster program. It removes the dead code part.
2. **Code Generation:** The last stage of the compiler takes the optimized code and convert into the assembly code.

3. Assembler

Assembler converts the assemly language code into machine language. It takes the assembly file generated by the compiler in *.s extension and converts it into the object code *.o format file. Figure 5 represents the action of the assembler.

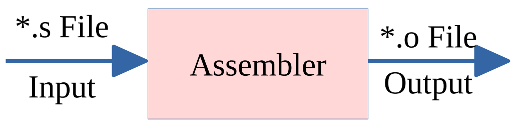
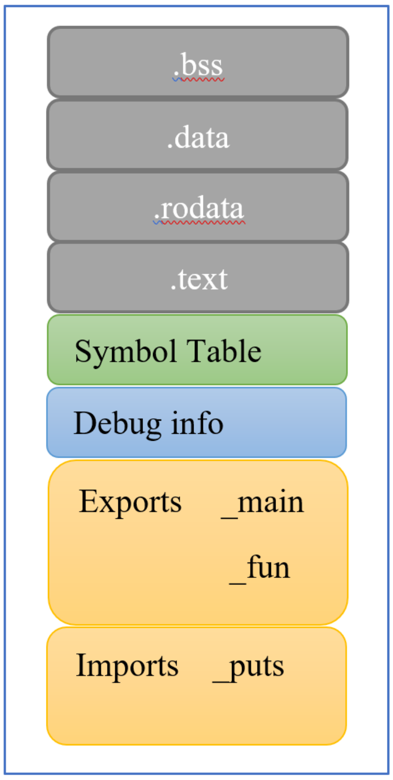


Fig. 5: Assembler

In order to generate the *.o file, *gcc* compiler can be used with c option. In this case, gcc will perform only three steps pre-processing, compilation and assembler. It will not call the linker program. The command will be as follows:

$gcc -c hello.c

1. The object file contains different sections to store static data.
2. Symbol table
3. debug info, mapping between original code and information required by the debugger.
4. Export section, containing global variables or functions.
5. Import section containing symbol name which will be needed from other object files.



4. Linker

The last stage of the compilation process is linker. This stage takes different files *.o object files, library files and generates one executable file. Figure 6 represents the working of linker.

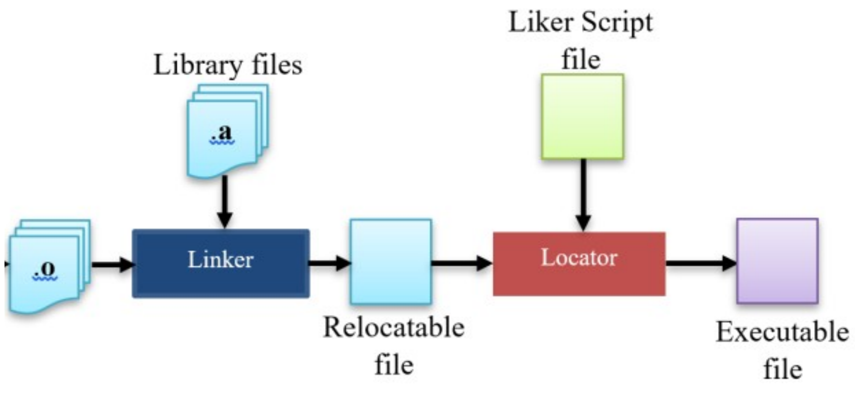


Fig. 6: Linker (Taken from: https://www.linkedin.com/pulse/c-build-process-details-abdelaziz-moustafa/)

Linker performs primary two functions:

1. **Symbol Resolution**: If there are any unresolved references or links in the object file then linker resolves them if it doesn't find the reference then it throws the error.
2. **Relocation:** It is the process of changing addresses already assigned to the labels.
3. Final part of the linker, converts relocatable file to the executable file with the help of linker script.