WS02: 4 Stages of Building Code

A C program goes through 4 stages to become an executable:

|  |  |  |  |
| --- | --- | --- | --- |
| # | Stage | Input | Output |
| 1 | Pre-processing | file.c | file.i |
| 2 | Compilation | file.i | file.s |
| 3 | Assembly | file.s | file.o |
| 4 | Linking | file.o | file |

This worksheet is based on the following code.   
  
Create a file (file.c) and copy this code into it:

#include <stdio.h>   
#define MESSAGE "Most people's C programs should be indented six feet downward and covered with dirt."  
  
int main(void) {   
 /\* Blair P. Houghton said the following \*/   
 printf(MESSAGE);   
 return 0;   
}

In Linux, calling `man` on any program will show the manual for that program.

**man gcc > gcc\_manual.txt**

This will create a file (gcc\_manual.txt) with all the information needed to use gcc. Keep this file handy.

Run the following command:

**gcc -Wall -save-temps file.c -o file**

Normally, gcc creates temporary intermediate files and deletes them after linking.   
With -save-temps, gcc keeps all intermediate files in the current directory.

For file.c, this will generate:   
file.i - Preprocessed code   
file.s - Assembly code  
file.o - Object code  
file - Final executable (note how in linux an executable has no extension unlike windows .exe

# 1. Pre-processing

The pre-processed output is stored in file.i. Open it and check the number of lines (should be more than 700). Most of the file content is added by gcc, but at the end you should see our code.

Key points to note:

1. Macro expansion: The argument to printf() now contains the full string; the macro definition is no longer in the code.

2. Comment removal: Original comments are removed.

3. Header inclusion: Lines starting with #include are replaced by hundreds of lines from stdio.h.

Search for printf in file.i, you should find:

extern int printf (const char \*\_\_restrict \_\_format, ...);

# 2. Compilation

This stage takes file.i as input, compiles it, and produces an intermediate output: file.s (assembly code).

Open file.s to observe assembly instructions. Understanding the assembly is not required.

# 3. Assembly

At this stage, file.s is converted to file.o (object file) by the assembler.

file.o contains machine-level instructions. You can peek at it using:

cat file.o

Most of it is unreadable, but you can recognize some strings. The ELF (Executable and Linkable Format) header appears at the beginning, indicating the structure of the object file.

# 

# 4. Linking

The linker completes the final stage by:

- Resolving function calls (e.g., linking printf() to its implementation).

- Adding startup code for command-line arguments, environment variables, and system return values.

The newly added code is not human-readable. You can compare the sizes of file.o and file:

**$ size file.o  
text data bss dec hex filename**199 0 0 199 c7 file.o

**$ size file  
text data bss dec hex filename**1195 276 4 1475 5c3 file

# Further exercise

Create a new C program and run separate gcc commands to transform the source code from one stage to the next

1. from source code to preprocessed code
2. from preprocessed code to compiled code
3. from compiled code to assembled code
4. from assembled code to executable code

# References

https://en.wikipedia.org/wiki/Executable\_and\_Linkable\_Format

http://www.thegeekstuff.com/2011/10/c-program-to-an-executable/

https://codingfreak.blogspot.com.mt/2008/02/compilation-process-in-gcc.html

http://academic.udayton.edu/saverioperugini/courses/cps346/lecture\_notes/compiling.html