What goes inside the compilation process?

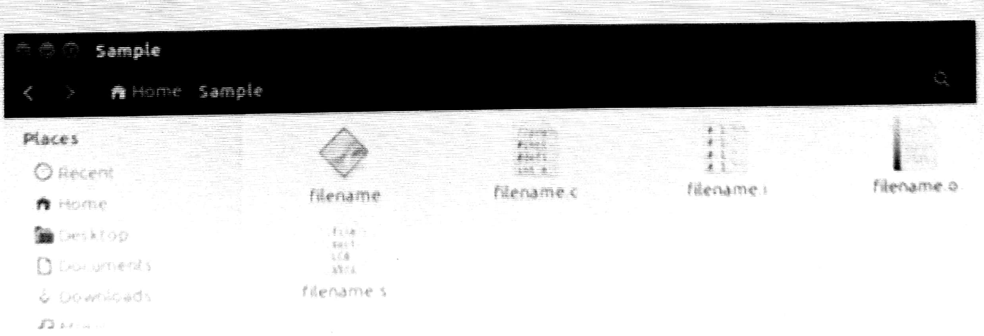
Compiler converts a C program into an executable. There are four phases for a C program to become an executable:

1. Pre-processing
2. Compilation
3. Assembly
4. Linking

By executing below command, We get the all intermediate files in the current directory along with the executable.

```
$gcc -Wall -save-temps filename.c -o filename
```

The following screenshot shows all generated intermediate files.



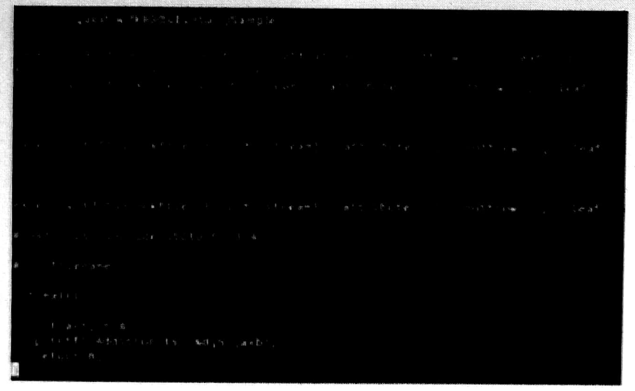
Let us one by one see what these intermediate files contain.

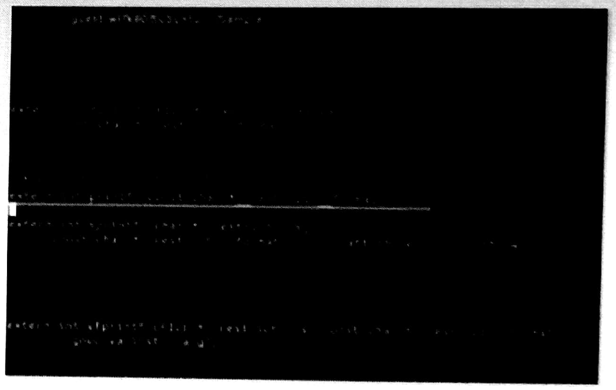
Pre-processing

This is the first phase through which source code is passed. This phase include:

- Removal of Comments
- Expansion of Macros
- Expansion of the included files.
- Conditional compilation

The preprocessed output is stored in the **filename.i**. Let's see what's inside filename.i: using **$vi filename.i**





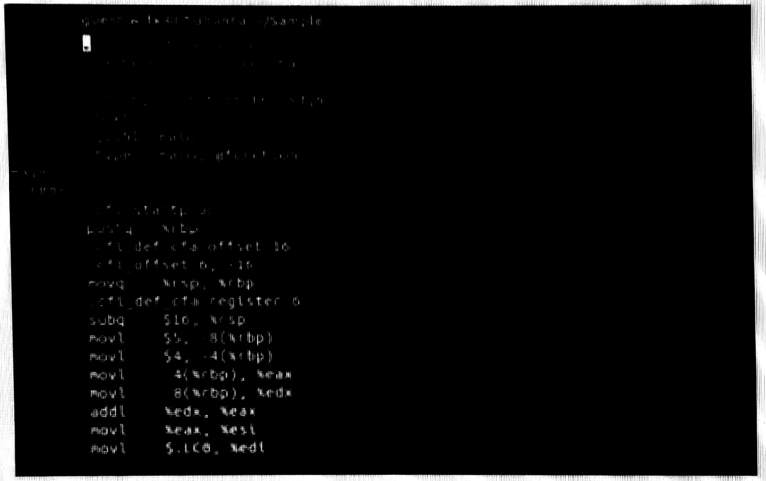
```
[illegible]
```

In the above output, source file is filled with lots and lots of info, but at the end our code is preserved.
Analysis:

- printf contains now a + b rather than add(a, b) that's because macros have expanded.
- Comments are stripped off.
- **#include<stdio.h>** is missing instead we see lots of code. So header files has been expanded and included in our source file.

Compiling

The next step is to compile filename.i and produce an; intermediate compiled output file **filename.s**. This file is in assembly level instructions. Let's see through this file using **$vi filename.s**

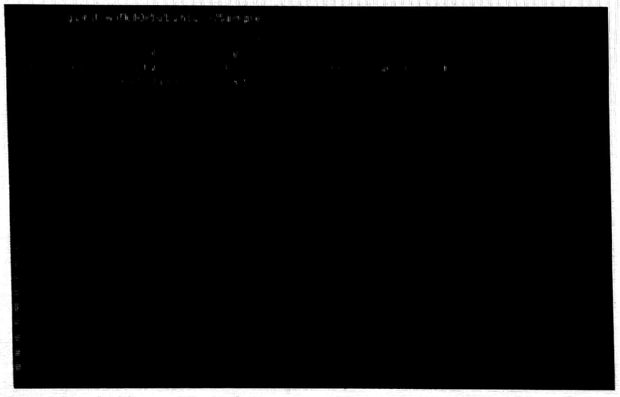


```
[illegible]
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $5, -8(%rbp)
        movl    $4, -4(%rbp)
        movl    -4(%rbp), %eax
        movl    -8(%rbp), %edx
        addl    %edx, %eax
        movl    %eax, %esi
        movl    $.LC0, %edi
```

The snapshot shows that it is in assembly language, which assembler can understand.

Assembly

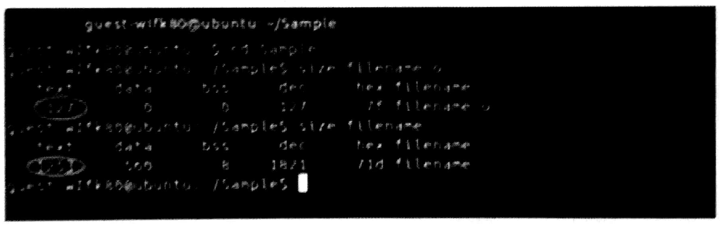
In this phase the filename.s is taken as input and turned into **filename.o** by assembler. This file contain machine level instructions. At this phase, only existing code is converted into machine language, the function calls like printf() are not resolved. Let's view this file using **$vi filename.o**



```
[illegible]
```

Linking

This is the final phase in which all the linking of function calls with their definitions are done. Linker knows where all these functions are implemented. Linker does some extra work also, it adds some extra code to our program which is required when the program starts and ends. For example, there is a code which is required for setting up the environment like passing command line arguments. This task can be easily verified by using **$size filename.o** and **$size filename**. Through these commands, we know that how output file increases from an object file to an executable file. This is because of the extra code that linker adds with our program.



```
          guest-wifk80@ubuntu ~/Sample
guest-wifk80@ubuntu:~$ cd Sample
guest-wifk80@ubuntu:~/Sample$ size filename.o
   text    data     bss     dec     hex filename
    127       0       0     127      7f filename.o
guest-wifk80@ubuntu:~/Sample$ size filename
   text    data     bss     dec     hex filename
[illegible]     560       8    1821     71d filename
guest-wifk80@ubuntu:~/Sample$ 
```

Note that GCC by default does dynamic linking, so printf() is dynamically linked in above program.