Shiv Nadar Institution of Eminence, Delhi, NCR

Lab sheet for CSD101 (Introduction to computing and Programming)

Semester of Implementation: Monsoon, 2024

Instructor: Dr. Suchi Kumari (suchi.kumari@snu.edu.in), Dr. Sweta Kumari (sweta.kumari@snu.edu.in)

TA: Mr. Vanu Prakash (bhanu.prakash@snu.edu.in), Mr. Mithun Kumar (mithun.kumar@snu.edu.in)

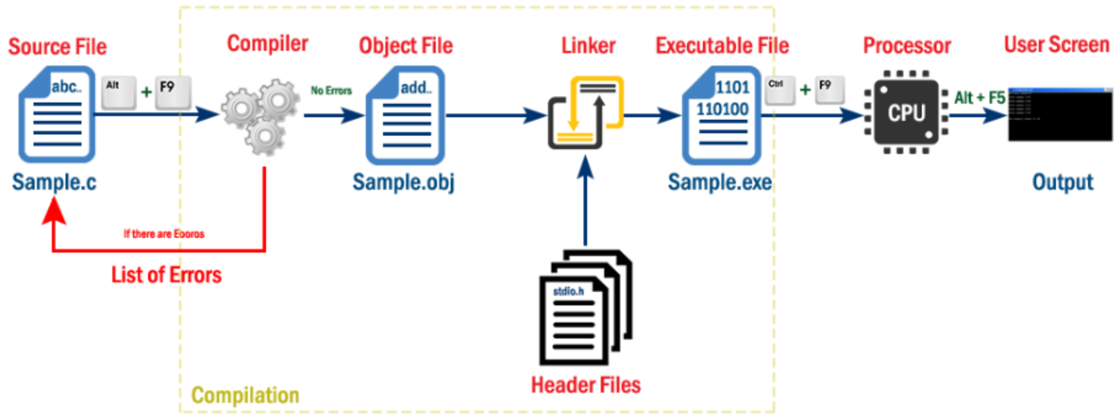
Instructions:

1. Once you complete the assignment, please show it to the TA.
2. Students must come to the lab and must show the assignments in the designated lab hours. Day-to-day lab performances will be recorded and will carry 15% weightage in internal assessment.
3. Lab will start in exact time. Students should enter the lab and take a seat 5 minutes before.
4. It is recommended to use LINUX platform for execution of the program.
5. Batch change to show the assignments WILL NOT be allowed.
6. Malpractice (in ANY form) will attract heavy penalties.
7. A useful link: https://www.w3schools.com/c/index.php

Lab Assignment 1

Introduction to Linux, Editor, GCC Compiler and Debugger. Introduction to basic Linux Commands

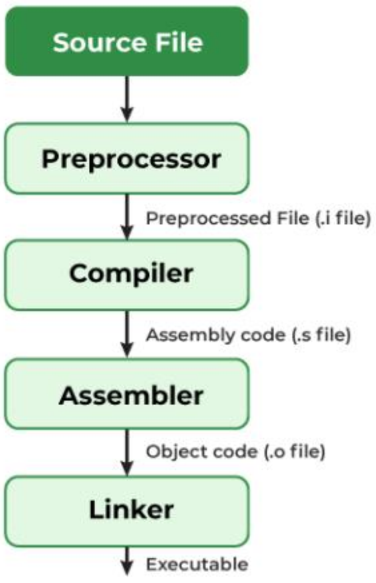
Execution process of C program



What goes inside the compilation process?

A 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**



Compilation Process in C

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

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

1. Pre-processing

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

- Removal of Comments

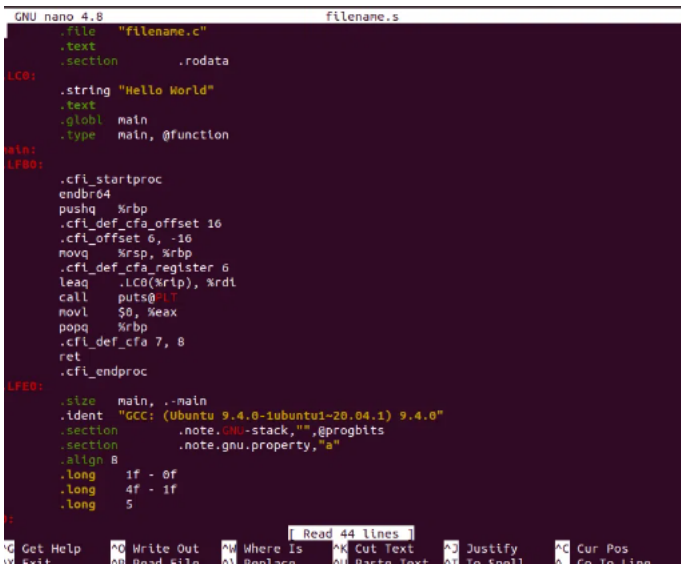
- Expansion of Macros
- Expansion of the included files.
- Conditional compilation

The preprocessed output is stored in the **filename.i**.

We can check the content of filename.i using **$vi filename.i**

2. Compiling

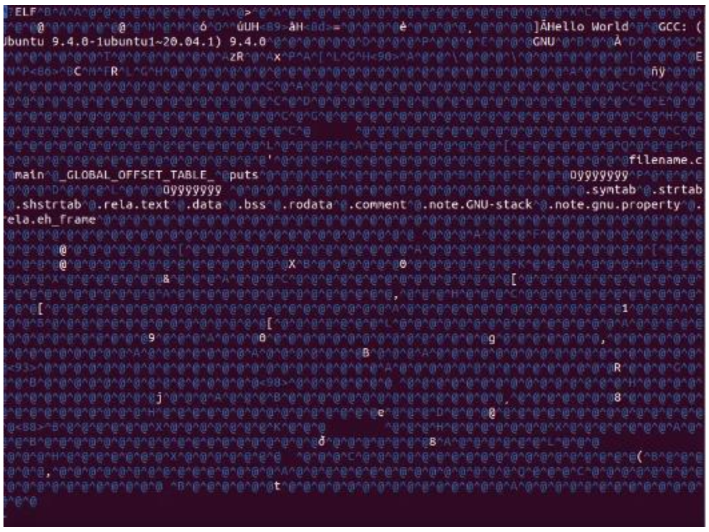
The next step is to compile filename.i and produce an; intermediate compiled output file **filename.s**. This file is in assembly-level instructions. We can see through this file using **$nano filename.s** terminal command.



```
 GNU nano 4.8                                   filename.s
        .file   "filename.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello World"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        leaq    .LC0(%rip), %rdi
        call    puts@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long    1f - 0f
        .long    4f - 1f
        .long    5
0:
                                  [ Read 44 lines ]
^G Get Help     ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos
```

3. Assembling

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



```
ELF
óUH  àH  è  ]ÃHello World  GCC: (
Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0  GNU  Ã
zR  x
C  R  ñÿ
filename.c
main  _GLOBAL_OFFSET_TABLE_  puts  0ÿÿÿÿÿÿÿ
0ÿÿÿÿÿÿÿ  .symtab  .strtab
.shstrtab  .rela.text  .data  .bss  .rodata  .comment  .note.GNU-stack  .note.gnu.property  .
rela.eh_frame
```

4. Linking

This is the final phase in which all the linking of function calls with their definitions is 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 that 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 how the output file increases from an object file to an executable file. This is because of the extra code that Linker adds to our program.

Lab Task 1: Learn some basic Linux Commands

Syntax	**Meaning**
clear	Clear the screen
mkdir [directory name]	Create new directory
ls	List the content present in a directory
cd [sub-directory name]	Enter to the sub-directory
pwd	Path of the current working directory
mv file1.c file2.c	Moving content of file1.c to file2.c (just like rename)
man ls	Open a manual page for help
cp file1.c file2.c	copy contents from file1.c to file2.c
cat	to display the contents of one file for e.g. cat text.c will display the contents of file text.c
who	Displays list of current users that are logged in.
rmdir [directory name]	Removes directory syntax rmdir directory name [before removing the directory u should delete all the files in that particular directory]
rm	Remove file (or delete file). Syntax: rm text.c removes the file text.c, similarly rm *.c will delete all the files with .c extension.

Steps to run C program

Step 1: Creating a C Source File in Linux using vi editor.

In Linux vi is one of the most popular editor to create or edit any source files.

a. Type **vi filename.c** this will open a editor.

b. To start typing the code give the command **Esc i** [press escape key once and the letter i] the word insert will appear at the left bottom of your screen now u can start typing your code.

c. Once finish typing to save the file give the command **Esc:wq** (i.e. press the escape key once the word insert at your left bottom will disappear, then hold the shift key and give colon [:] then give the command **wq** – [meaning write and quit]).

Step 2: Compiling using GCC compiler

We use the following command in the terminal for compiling our filename.c source file

```
$ gcc filename.c –o filename
```

Step 3: Executing the program

After compilation executable is generated and we run the generated executable using the below command.

```
$ ./filename
```

Note: No need to submit today's lab task.

Instruction for Submission for Lab Task:

Create a Word document and insert screenshots of the lab tasks. Save this Word document as a PDF file, using the format: YourName_EnrolmentNo_LabAssignmentNo.pdf, and upload it to Blackboard.