

# Computer Organizations and Systems

## Assignment -1

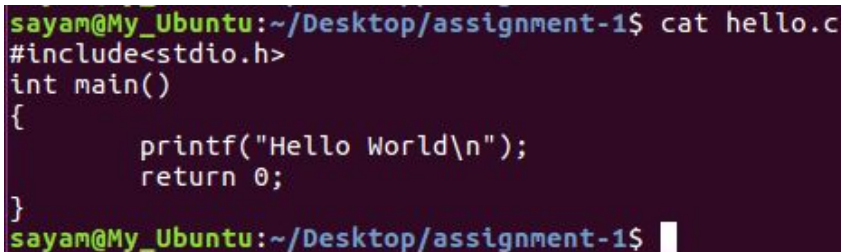
Submitted By -

Sayam Kumar

S20180010158 Sec-A

**Question-1.** Show all the steps for execution of object file for hello world program with screenshots after successfully loading into RAM.

**Answer -** Below is the screenshot of my hello world program.

A screenshot of a terminal window with a dark background. The prompt is 'sayam@My\_Ubuntu:~/Desktop/assignment-1\$'. The user has entered 'cat hello.c'. The output shows the following C code:

```
#include<stdio.h>
int main()
{
    printf("Hello World\n");
    return 0;
}
```

The prompt is now 'sayam@My\_Ubuntu:~/Desktop/assignment-1\$' with a cursor.

There are four major steps before successful execution of our C program. These are Pre-processing, compilation, assembling and linking. All these four stages are explained in detail in the forthcoming pages.

**1. Pre-processing** - Through pre-processing, all the declarations present inside the header files gets included in the C program. Also, pre-processing removes the comments before compilation of the program and completes the task of macro expansion. The return 0 statement verifies the successful execution of program to the parent process ie shell. The hello.i file generated after command **cpp hello.c** > **hello.i** includes all declarations and macros replaced. The same is shown in the screenshot below. The hello world program written by us gets appended to the last of hello.i file.

```
sayam@My_Ubuntu:~/Desktop/assignment-1$ ls
hello.c
sayam@My_Ubuntu:~/Desktop/assignment-1$ cpp hello.c > hello.i
sayam@My_Ubuntu:~/Desktop/assignment-1$ cat hello.i
# 1 "hello.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 1 "<command-line>" 2
# 1 "hello.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/features.h" 1 3 4
# 367 "/usr/include/features.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 1 3 4
# 410 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
# 411 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
# 368 "/usr/include/features.h" 2 3 4
# 391 "/usr/include/features.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 1 3 4
# 10 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/gnu/stubs-64.h" 1 3 4
# 11 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 2 3 4
# 392 "/usr/include/features.h" 2 3 4
# 28 "/usr/include/stdio.h" 2 3 4

# 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 1 3 4
```

**2. Compilation** - Through compilation, the code in hello.i gets compiled. The major process involved in compilation are parsing, lexing of the code and optimization of the unused variables. The -S flag indicates that compilation should stop before assembling the code. This will help us to dive deeply into the process of execution of the hello world program. The hello.s file generated after compilation is an assembly file written in low level language. This also contains information regarding the instruction set of the microprocessor.

```
sayam@My_Ubuntu:~/Desktop/assignment-1$ ls
hello.c  hello.i
sayam@My_Ubuntu:~/Desktop/assignment-1$ gcc -S hello.i
sayam@My_Ubuntu:~/Desktop/assignment-1$ ls
hello.c  hello.i  hello.s
sayam@My_Ubuntu:~/Desktop/assignment-1$ cat hello.s
.file "hello.c"
.section .rodata
.LC0:
.string "Hello World"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
movl $.LC0, %edi
call puts
movl $0, %eax
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size main, .-main
.ident "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.10) 5.4.0 20160609"
.section .note.GNU-stack,"",@progbits
sayam@My_Ubuntu:~/Desktop/assignment-1$
```

**3. Assembling** - Through assembling, the assembly code in hello.s file gets changed into object code. This object code is in the form of binary language which is ready to be served to the microprocessor. The command for generating hello.o file is **as -o hello.o hello.s** . The ELF(Executable and Linkable format) is a common standard file format for executables and object codes.

```
sayam@My_Ubuntu:~/Desktop/assignment-1$ ls
hello.c hello.i hello.s
sayam@My_Ubuntu:~/Desktop/assignment-1$ as -o hello.o hello.s
sayam@My_Ubuntu:~/Desktop/assignment-1$ ls
hello.c hello.i hello.o hello.s
sayam@My_Ubuntu:~/Desktop/assignment-1$ cat hello.o
ELF
UH
Hello WorldGCC: (Ubuntu 5.4.0-6ubuntu1~16.04.10) 5.4.0 20160609
hello.cmainputs
syntab.strtab.shstrtab.rela.text.data.bss.rodata.comment.note.GNU-stack.rela.eh_frame
a6
```

**4. Linking** - Through linking, we link all the library files. For fixing the error in loading, I have taken the help of the website shown to us. These crt files help in generating stack in memory and then successfully loads the program in memory. After execution of **./hello**, the output of “**Hello World**” is shown correctly.

```
sayam@My_Ubuntu:~/Desktop/assignment-1$ ls
hello.c hello.i hello.o hello.s
sayam@My_Ubuntu:~/Desktop/assignment-1$ ld -static -o hello -L `gcc -print-file-name=` /usr/lib/x86_64-linux-gnu/crt1.o /usr/lib/x86_64-linux-gnu/crti.o hello.o /usr/lib/x86_64-linux-gnu/crtn.o --start-group -lc -lgcc -lgcc_eh --end-group
sayam@My_Ubuntu:~/Desktop/assignment-1$ ls
hello hello.c hello.i hello.o hello.s
sayam@My_Ubuntu:~/Desktop/assignment-1$ ./hello
Hello World
sayam@My_Ubuntu:~/Desktop/assignment-1$
```

**Thank You!**