Compile and Link C Program

There are three basic phases occurred when we execute any C program.

- Preprocessing
- Compiling
- (assembler)
- Linking

Preprocessing Phase :A C pre-processor is a program that accepts C code with preprocessing statements and produces a pure form of C code that contains no preprocessing statements (like #include).

Compilation Phase: The C compiler accepts a preprocessed output file from the preprocessor and produces a special file called an object file. Object file contains machinecode generated from the program.

Linking Phase: The link phase is implemented by the linker. The linker is a process that accepts as input object files and libraries to produce the final executable program.

Compiling and Linking a C program is a multi-stage process.

The process can be split into four separate stages: **Preprocessing**, **compilation**, **assembly**, **and linking**.

```
/*
 * "Hello, World!": A classic.
 */

#include

<stdio.h>int

main(void)
{
        puts("Hello,
        World!");return 0;
```

Preprocessing

The first stage of compilation is called preprocessing. In this stage, lines starting with a # character are interpreted by the *preprocessor* as *preprocessor commands*. These commands form a simple macro language with its own syntax and semantics.

reduce repetition in source code
invoke inline files
define macros
joining continued lines (lines ending with a \)
removes comments.

To print the results of the preprocessing stage, pass the -E option to cc:

```
gcc -E hello_world.c
```

[lines omitted for brevity]

```
extern int __vsnprintf_chk (char * restrict, size_t,
      int, size_t, const char * restrict, va_list);
# 493 "/usr/include/stdio.h" 2 3 4
# 2 "hello_world.c" 2

int main(void) {
 puts("Hello, World!");
 return 0;
}
```

Compilation

In this stage, the preprocessed code is translated to *assembly instructions* These form an intermediate readable language.

Some compilers also supports the use of an integrated assembler, in which the compilation stage generates *machine code* directly, avoiding the overhead of generating the intermediate assembly instructions and invoking the assembler. *object code is directly produced by compiler.*

to view the result of the compilation stage, pass the -S option to cc:

```
gcc -S hello_world.c
```

This will create a file named hello_world.s

```
.section    TEXT,__text,regular,pure_instructions
    .macosx_version_min 10, 10
    .globl  _main
    .align  4, 0x90
_main:                                  ## @main
    .cfi_startproc
## BB#0:
    pushq   %rbp
Ltmp0:
    .cfi_def_cfa_offset 16
Ltmp1:
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
Ltmp2:
    .cfi_def_cfa_register %rbp
    subq    $16, %rsp
    leaq    L_.str(%rip), %rdi
    movl    $0,  -4(%rbp)
    callq   _puts
    xorl    %ecx, %ecx
    movl    %eax, -8(%rbp)          ## 4-byte Spill
    movl    %ecx, %eax
    addq    $16, %rsp
    p
    o
    p
    q
            %rbpretq
    .cfi_endproc

    .section    TEXT,__cstring,cstring_literals
L_.str:                                 ## @.str
    .asciz  "Hello, World!"
```

Assembly

During the assembly stage, an assembler is used to translate the assembly instructions tomachine code, or *object code*.
--The output consists of actual instructions to be run by the target processor.

Input to assembler :cc -c hello_world.c Output of assembler phase: hello-worls.o
The contents of this file is in a binary format and can be viewed using hexdump or odcommands:

```
gcc  -C hello_world.c
```

Linking

The object code generated in the assembly stage is composed of machine instructions that theprocessor understands but some pieces of the program are out of order or missing. To produce an executable program, the existing pieces have to be rearranged and the missing ones filled in. This process is called linking.

The result of this stage is the final executable program(.exe)

Finally to run

a.out hello_world.c

Commands to compile and execute C program

Save the program file using .c extention

To compile:

```
gcc filename.c
```

To run the program

```
./a.out
```