LECTURE 2: HELLO, WORLD (K&R § 1.1)

GOALS

1.	create your program text somewhere
2.	compile it successfully
3.	load it
4.	run it
5.	find out where your output went

CREATE YOUR PROGRAM TEXT

The C code to print "hello, world" is:

```
/* Project: hw1
 * Name : Your Name
 * Date : 01/26/2016
 * File : hello.c
 * Notes : [e.g., pseudo code]
 */
#include <stdio.h>

main() {
  printf("hello, world\n");
  return 0;
}
```

where do you put this code?

Use a text editor (e.g., vi or emacs) to create a source file named hello.c.

Example:
$ emacs hello.c
or
$ vi hello.c
	Launches the editor and creates the blank file hello.c
	(if a file named hello.c already exists in the current working directory, it will open that file).

hello.c DISSECTED

`/* Project: hw1`
` * Name : Your Name`
` * Date : 01/26/2016`
` * File : hello.c`
` * Notes : [...]`
` */`
	comment

`#include <stdio.h>`	C preprocessor directive
`main() {`	C function header

`printf("hello, world\n");`
`return 0;`
	C function body
	(contains statements)

`}`	

COMMENTS

Any characters between /* and */ are ignored by the compiler.

These ignored characters are comments, and should be used to make your program easier to understand.

NOTE:	Do not forget the closing */!

comment examples:

```
/* basic comment on its own line */
```

```
/*********************************************************
 * multiline comments sometimes use formatting like
 * this to make the commment stand out.
 *********************************************************/
```

```
/* but this works
   just as well */
```

```
printf("example\n"); /* comments can follow statements */
```

C++ single line comments

C++ introduced the double slash for single line comments.

Most newer C compilers will recognize this style, but it is not used in K&R.

```
// double slash comment goes from slashes to end of line
printf("example\n"); // and can follow statements
```

#include <stdio.h>

header file	uses the .h extension contains: C function declarations macro definitions to be shared between several source files.

Here, we are using printf, which is part of the standard I/O library, so we need the header file stdio.h.

gcc will not compile hello.c without #include <stdio.h>

#include	preprocessor directive to include a header file

main()

int main (void)	This is where the UNIX system starts execution of your program.	
main()	Note that K&R use this form. This is because:	
	1.	the return type defaults to int if not specified
	2.	in older versions of C, empty parentheses in a function definition indicated the function does not take arguments. this still works for compatibility reasons, but void should be used.

The body of your program is between the braces, { and }.

Two styles of braces (both are acceptable):

`int main(void) {`
`  [program body];`
`}`
	`int main(void)`
	`{`
	`  [program body];`
	`}`

printf

The standard I/O library provides the function printf().

printf()	prints its argument (the text between the parentheses) on the screen.

```
printf("Hello, world\n");
```

All C program statements end with a semicolon (;).

The argument in this statement is: "Hello, world\n"

The quotation marks denote a character string.

\n is C notation for the new line character.	When printed, this advances output to the left margin of the next line.

LECTURE 2: COMPILING (K&R § 1.1)

GNU C COMPILER (gcc)

Alternately stands for: GNU Compiler Collection.

On most GNU/Linux systems, cc (C compiler) is an alias for gcc and will produce the same result.

gcc --version	Execute this command from a shell to find out if you have gcc installed.

If it is installed, it will output something like:

```
   gcc (Ubuntu 4.8.4-2ubuntu1~14.04) 4.8.4
   Copyright (C) 2013 Free Software Foundation, Inc.
   This is free software; see the source for copying
   conditions.  There is NO warranty; not even for
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

USING gcc

```
$ gcc -Wall [sourceFile] -o [executableName]
```

For example,

```
$ gcc -Wall hello.c -o hello
```

produced an executable file named hello.

-Wall	this option causes the compiler to warn you about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning)
-o [file]	this option causes the executable file produced by gcc to be named [file]. if this option is not used, the filename defaults to a.out.

WHAT gcc DOES

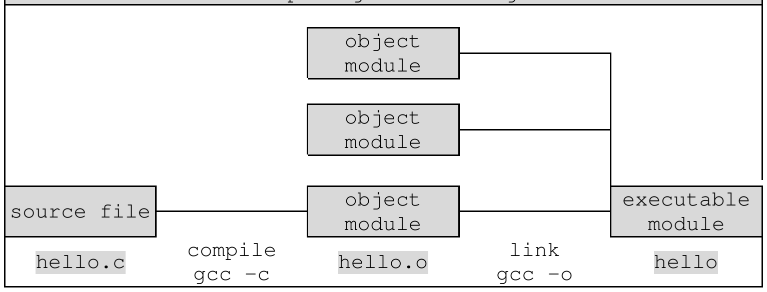
The term compile is often used as a high-level way to say "convert source code to a program."

A more specific use of the term: "convert source code to object code."

source code	the human-readable instructions that a programmer writes (e.g., our hello.c file).
object code	object code files use the .o extension a portion of machine code that has not yet been linked into a complete program.

the steps of compilation:	
1.	the preprocessor Invoked automatically by the compiler before actual compilation begins. Creates a modified source file using the commands preceded by #.
2.	the compiler as mentioned above, converts source code to object code. using the -c option tells gcc to compile the source code into object code, but not invoke the linker
3.	the linker links object files (.o files) into a binary executable. uses both object files created by the compiler from your source code and precompiled objects that have been collected in library files.

compiling and linking



RUNNING YOUR PROGRAM

If you have successfully produced an executable file called hello, you can run it by typing:

```
$ ./hello
```

Recall that . denotes the current working directory.

Because the executable file you just created is not included in your PATH environment variable (which tells the shell where to find executable files), you are explicitly specifying where to find hello.

DEBUGGING A C PROGRAM ERROR

The types of errors you will encounter are generally divided into three categories:

1.	runtime errors the program compiles, but performs some illegal operation during execution.
2.	compile errors an error makes the compiler unable to create an executable file.
3.	logical errors the program compiles and runs without generating errors, but it does not provide the desired result.

compiler error messages
may direct you to a specific error in your program
may be vague about what the error is and why it is an error
some compilers are better than others in this regard

Example:

If we try to compile hello.c with the closing brace missing, we get the following message:

```
jmcurran@itserver6:~/scratch$ gcc hello_broken.c
hello_broken.c: In function ‘main’:
hello_broken.c:11:3: error: expected declaration or
statement at end of input
   return 0;
   ^
jmcurran@itserver6:~/teaching$
```

The message itself is not very helpful, but it does tell you something is wrong at line 11, column 3.