Lecture 13: The C Preprocessor

Sarah Nadi
nadi@ualberta.ca
Department of Computing Science
University of Alberta

CMPUT 201 - Practical Programming Methodology

[With material/slides from Guohui Lin, Davood Rafei, and Michael Buro. Most examples taken from K.N. King's book]

Agenda

- How the preprocessor works
- Categories of directives
- Macro definitions
- Conditional compilation
- Other important directives

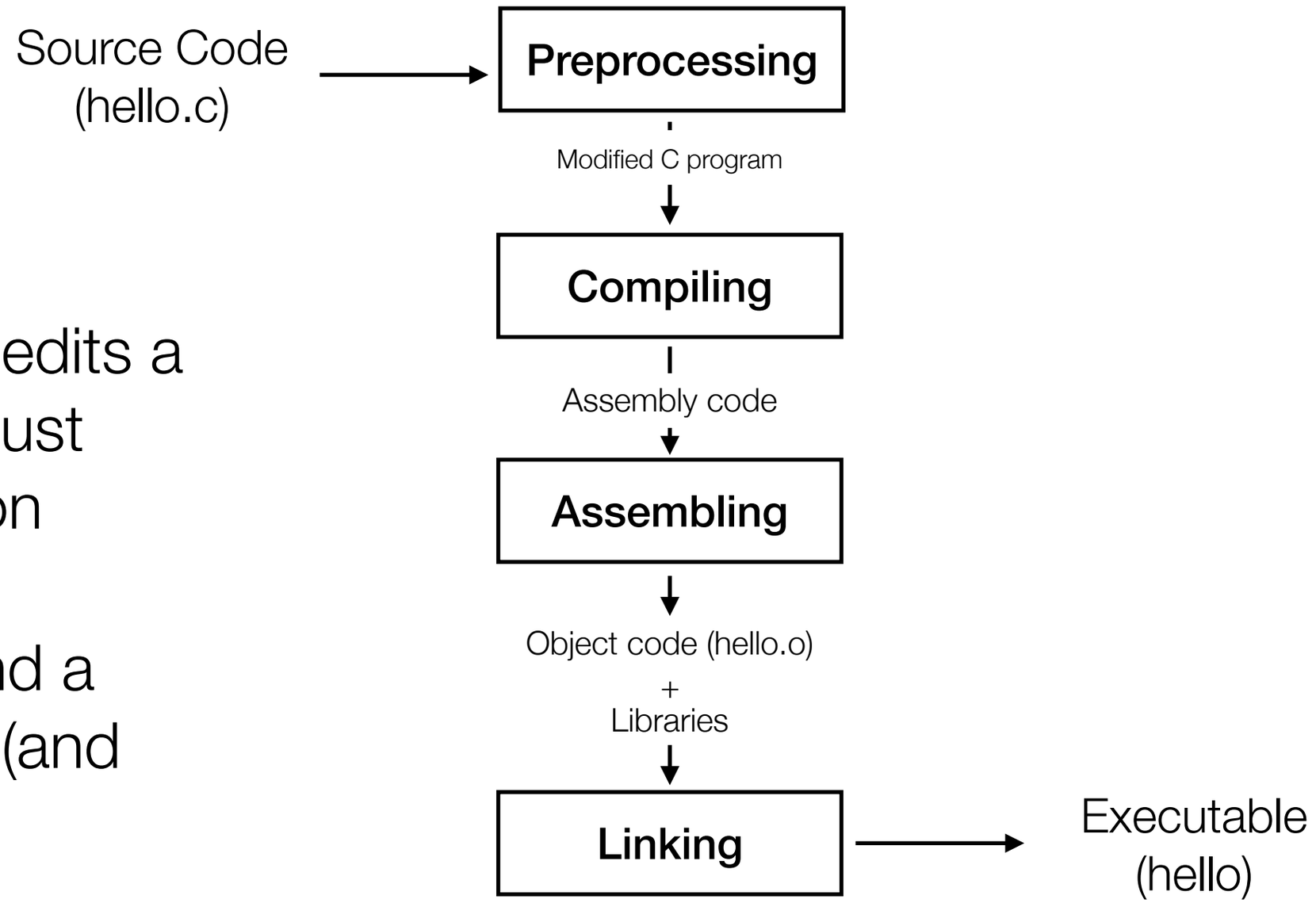
Readings

- Textbook Chapter 14 (not all parts covered)

The Preprocessor

- Is a software that edits a given C program just prior to compilation

- It is part of gcc and a main feature of C (and C++)



Preprocessor Directives

- The preprocessor only deals with preprocessor directives.
- *Preprocessor directives* start with # and fall into 3 main categories (a few extra ones exist): macro definitions, file inclusion, and conditional compilation
- Examples of preprocessor directives:
  - `#include <stdio.h>`
    - opens the indicated file an includes its contents as part of the file being compiled
  - `#define STR_LEN 80`
    - expands the macro by replacing any occurrence of it with its defined value
- To see what the program looks like after preprocessing, you can use `gcc -E program.c`

demo: demo_preproc.c

Rules for Preprocessor Directives

- Always start with #
- Spaces and tabs in the middle are irrelevant
- A directive ends at the first newline (\n), unless explicitly continued by a \ character
- Directives can appear anywhere in the program and they have an effect from that point on. That said, `#define` and `#include` directives typically appear at the top of the file.
- Comments can appear on the same line as the directive

Conditional Compilation

- *Conditional compilation* is the inclusion or exclusion of a section of program text depending on the outcome of a test performed by the preprocessor

Conditional Compilation:

`#if` and `#if defined()`

Conditional Compilation:

`#if` and `#if defined()`

```
int i = 3;
#if DEBUG
printf(“DEBUG: Value of i is %d\n”, i);
#endif /* DEBUG */
i++;
```

Conditional Compilation:

`#if` and `#if defined()`

```
int i = 3;
#if DEBUG
printf("DEBUG: Value of i is %d\n", i);
#endif /* DEBUG */
i++;
```

Code between the #if and #endif will not be included for compilation unless DEBUG has the a non-zero value. E.g.:

```
#define DEBUG 1
#define DEBUG 10
```

Conditional Compilation:

`#if` and `#if defined()`

```
int i = 3;
#if DEBUG
printf(“DEBUG: Value of i is %d\n”, i);
#endif /* DEBUG */
i++;
```

Code between the #if and #endif will not be included for compilation unless DEBUG has the a non-zero value. E.g.:

```
#define DEBUG 1
#define DEBUG 10
```

```
int i = 3;
#if defined(DEBUG)
printf(“DEBUG: Value of i is %d\n”, i);
#endif /* DEBUG */
i++;
```

Conditional Compilation:

`#if` and `#if defined()`

```
int i = 3;
#if DEBUG
printf("DEBUG: Value of i is %d\n", i);
#endif /* DEBUG */
i++;
```

Code between the #if and #endif will not be included for compilation unless DEBUG has the a non-zero value. E.g.:

```
#define DEBUG 1
#define DEBUG 10
```

```
int i = 3;
#if defined(DEBUG)
printf("DEBUG: Value of i is %d\n", i);
#endif /* DEBUG */
i++;
```

Code between the #if defined(..) and #endif will not be included for compilation unless DEBUG has been explicitly defined somewhere (regardless of its value). E.g.:

```
#define DEBUG
```

Conditional Compilation:

`#if` and `#if defined()`

```
int i = 3;
#if DEBUG
printf("DEBUG: Value of i is %d\n", i);
#endif /* DEBUG */
i++;
```

Code between the #if and #endif will not be included for compilation unless DEBUG has the a non-zero value. E.g.:

```
#define DEBUG 1
#define DEBUG 10
```

```
int i = 3;
#if defined(DEBUG)
printf("DEBUG: Value of i is %d\n", i);
#endif /* DEBUG */
i++;
```

Code between the #if defined(..) and #endif will not be included for compilation unless DEBUG has been explicitly defined somewhere (regardless of its value). E.g.:

```
#define DEBUG
```

```
int i = 3;
#if INT_MAX < 1000
…
#endif /* INT_MAX */
i++;
```

Conditional Compilation:

`#if` and `#if defined()`

```
int i = 3;
#if DEBUG
printf("DEBUG: Value of i is %d\n", i);
#endif /* DEBUG */
i++;
```

Code between the #if and #endif will not be included for compilation unless DEBUG has the a non-zero value. E.g.:

```
#define DEBUG 1
#define DEBUG 10
```

```
int i = 3;
#if defined(DEBUG)
printf("DEBUG: Value of i is %d\n", i);
#endif /* DEBUG */
i++;
```

Code between the #if defined(..) and #endif will not be included for compilation unless DEBUG has been explicitly defined somewhere (regardless of its value). E.g.:

```
#define DEBUG
```

```
int i = 3;
#if INT_MAX < 1000
…
#endif /* INT_MAX */
i++;
```

can also check for values, rather than just Boolean expressions

Conditional Compilation: #ifdef, #ifndef, #else, #elif

```
int i = 3;
#ifdef DEBUG
   printf("DEBUG: Value of i is %d\n", i);
#endif /* DEBUG */
i++;
```

```
int i = 3;
#ifndef DEBUG
  printf("Production message %d\n", i);
#else
  printf("Production message: Initialized program\n");
#endif /* DEBUG */
i++;
```

```
#if defined(WIN32)
…
#elif defined (MAC_OS)
..
#elif defined (LINUX)
…
#endif
```

Conditional Compilation: `#ifdef, #ifndef, #else, #elif`

```
int i = 3;
#ifdef DEBUG
   printf("DEBUG: Value of i is %d\n", i);
#endif /* DEBUG */
i++;
```

Behaves similar to #if defined(..). Code between the #ifdef and #endif will not be included for compilation unless DEBUG has been explicitly defined somewhere (regardless of its value). E.g.:

```
#define DEBUG
```

```
int i = 3;
#ifndef DEBUG
  printf("Production message %d\n", i);
#else
  printf("Production message: Initialized program\n");
#endif /* DEBUG */
i++;
```

```
#if defined(WIN32)
…
#elif defined (MAC_OS)
..
#elif defined (LINUX)
…
#endif
```

Conditional Compilation: `#ifdef, #ifndef, #else, #elif`

```
int i = 3;
#ifdef DEBUG
   printf("DEBUG: Value of i is %d\n", i);
#endif /* DEBUG */
i++;
```

Behaves similar to #if defined(..). Code between the #ifdef and #endif will not be included for compilation unless DEBUG has been explicitly defined somewhere (regardless of its value). E.g.:

```
#define DEBUG
```

```
int i = 3;
#ifndef DEBUG
  printf("Production message %d\n", i);
#else
  printf("Production message: Initialized program\n");
#endif /* DEBUG */
i++;
```

Can test whether something is ***undefined*** and can also have else parts (#ifdef can also have #else)

```
#if defined(WIN32)
…
#elif defined (MAC_OS)
..
#elif defined (LINUX)
…
#endif
```

Conditional Compilation: `#ifdef, #ifndef, #else, #elif`

```
int i = 3;
#ifdef DEBUG
   printf("DEBUG: Value of i is %d\n", i);
#endif /* DEBUG */
i++;
```

Behaves similar to #if defined(..). Code between the #ifdef and #endif will not be included for compilation unless DEBUG has been explicitly defined somewhere (regardless of its value). E.g.:

```
#define DEBUG
```

```
int i = 3;
#ifndef DEBUG
  printf("Production message %d\n", i);
#else
  printf("Production message: Initialized program\n");
#endif /* DEBUG */
i++;
```

Can test whether something is ***undefined*** and can also have else parts (#ifdef can also have #else)

```
#if defined(WIN32)
…
#elif defined (MAC_OS)
..
#elif defined (LINUX)
…
#endif
```

Can use cascaded "if-else" directives. This is especially useful to write code that is portable to several machines or works with several compilers, for example.

Default Values

```
#ifndef BUFFER_SIZE
  #define BUFFER_SIZE 10
#endif
```

Default Values

```
#ifndef BUFFER_SIZE
  #define BUFFER_SIZE 10
#endif
```

This is similar to the structure you have seen when defining header files. When this structure is used with header files, it is called an *#include guard* or a *header guard*. We will talk more about this next class.

How to Define a Macro

- So how do we define the macros we check for in #if defined, #ifdef etc.?

- We can define (or actually un-define) them in the program itself, before the check happens. E.g.:
  - `#define DEBUG`
  - `#define SIZE 10`
  - `#undef DEBUG`
- Or we can define them from outside the program by providing them during compilation

(Un-)Defining Macros During Compilation

- `-D name`
  - Predefine name as a macro, with definition 1
  - e.g.,: `gcc -Wall -std=c99 -DDEBUG -D 64BIT -o test test.c`
- `-D name=definition`
  - contents of definition are processed as if you had a #define name (definition)
  - e.g., `gcc -Wall -std=c99 -D MAX_SIZE=1000 -o test test.c`
- `-U name`
  - e.g., `gcc -Wall -std=c99 -UDEBUG -o test test.c`

demo: preproc.c

The `#error` Directive

```
#ifndef BUFFER_SIZE
#error BUFFER_SIZE must be defined
#endif
```

- Encountering #error indicates a serious problem and causes the compilation to terminate (well, technically it causes the preprocessing stage to terminate)

```
#if defined (WIN32)
…
#elif defined (MAC_OS)
…
#elif defined (LINUX)
…
#else
#error No operating system specified
#endif
```

demo: preproc-error.c