M6 – C/C++ Preprocessor

CS 136L F23 – LEC 8

Yiqing Irene Huang, Qianqiu Zhang

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS DAVID R. CHERITON SCHOOL OF COMPUTER SCIENCE

Disclaimer

- The following slides were not presented page by page in class.
- They are my own study notes to share with students.
- In the lab session, we will cover key points, do small demos and give hints on commonly seen errors

Main Points

Learn about the C preprocessor and its features

- Use the `#include` directive to copy and paste header files
- Use the `#define` directive to create macros
- Use `#if`, `#ifndef`, and `#ifdef` directives to conditionally compile code
- Use `#include` **guards** in header files to avoid including the header files multiple times

Lab Thresholds

Question	Description	# of Tests	Pass	Complete
Q1	Include Guards	5	4	5
Q2	Testing Suite Writing	10	4	6
Q3	Conditional Compilation	8	3	8
Q4	Command-line Macros	8	3	8

Shell Scripting Tips:

- Do not forget the shebang line
- Be careful with the white spaces
- Recall the shell-defined variables $1, $2, et. al..

Testing Tips:

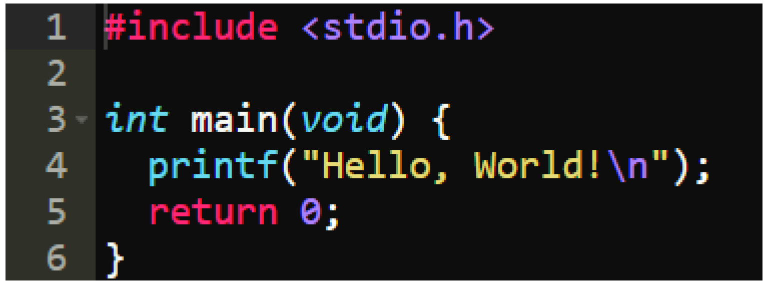
- Run the viewer program and understand its functionality
- Think Edge Cases!

Preprocessor

- Preprocessor directives are lines in C beginning with `#` symbol
  - Header file include: `#include`
  - Macro expansion: `#define`
  - Conditional compilation: `#if, #ifdef, #ifndef`, and `#endif`

Preprocessor and Header Files

- Preprocessor directives are lines in C beginning with # symbol
  - Header file include: #include
  - Macro expansion: #define
  - Conditional compilation: #if, #ifdef, and #ifndef
- C preprocessor `#include`
  - The include path: `/usr/include` etc. al. and `-I`
    - `clang: clang -v <file.h>, clang -Iheader -v <file.h>`
    - `gcc: cpp -v, cpp -Iheader -v`
  - `<file.h>`: search include path
  - `"file.h"`: search current source file dir and the include path
  - Replaces the `#include` line with the contents of `file.h`
- The `clang -E` option
  - Run the preprocessor and output the modified C source code with preprocessor directives being acted upon and removed.



```c
#include <stdio.h>

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

Macro Expansion

- The `#define` directive
  - Object-like
    - `#define identifier value,` no space in identifier
    - The `const` in modern C make most of use of #define obsolete
  - Function-like (not in scope)

Constant Length Array

```c
#define MAX 10

int array[MAX];
```

```c
int array[10];
```

Variable Length Array

```c
const int x = 10;

int array[x];
```

No syntax error,
but don’t do it in real code

```c
#include <stdio.h>
#define EVER (;;)

int main(void){
  for EVER {
    printf("Hello\n");
  }
}
```

Expansion of one macro affects another

```c
#include <stdio.h>
#define FIRST SECOND
#define SECOND third
#define third int

FIRST main(void){
  printf("Hello\n");
}
```

Exercise 1

```
#include <stdio .h>
#define SEVEN 3 + 4

int main (void) {
     printf ("%d\n", SEVEN * 2) ;
     return 0;
}
```

a) 11
b) 12
c) 13
d) 14
e) None of the above

Exercise 2

- Which one(s) define(s) a variable length array?

```c
//A ex2_a.c
int main (void) {
    const int x = 5;
    int arr[x];
    return 0;
}
```

```c
//B ex2_b.c
int main (void) {
    int x = 5;
    int arr[x];
    return 0;
}
```

```c
//C ex2_c.c
#define LEN 5
int main (void) {
    int arr[LEN];
    return 0;
}
```

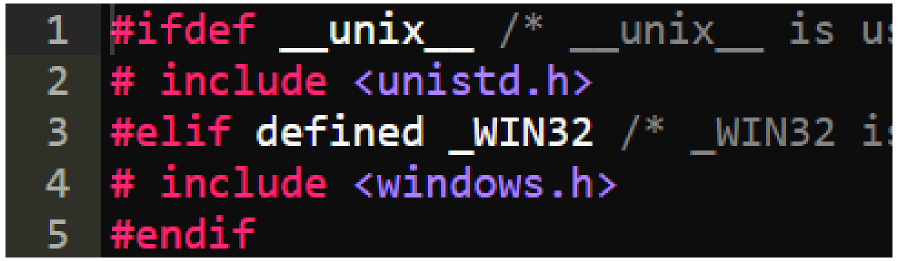
```c
//D ex2_d.c

int main (void) {
    int arr[5];
    return 0;
}
```

Conditional Compilation

`#if`, `#ifdef`, `#ifndef`, `#elif`, `#else` and `#endif`

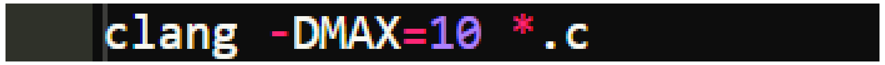
Conditional compilation happens at compile-time

Build for different Operating Systems
__unix__ and _WIN32 are compiler defined macros



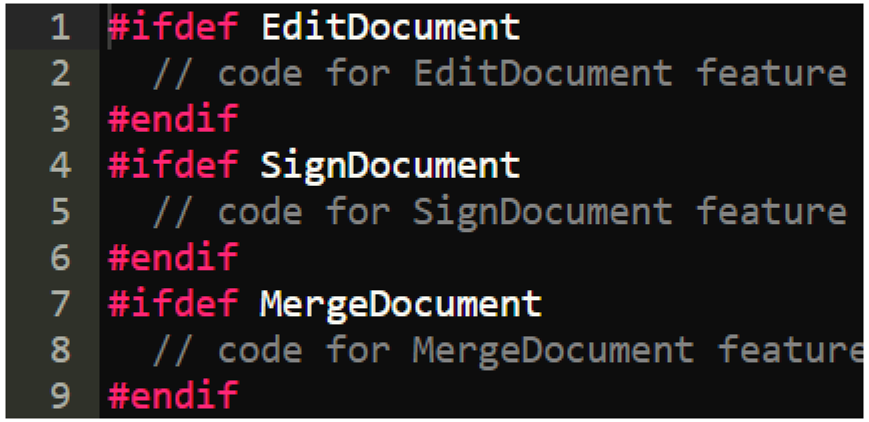
```
#ifdef __unix__ /* __unix__ is u
# include <unistd.h>
#elif defined _WIN32 /* _WIN32 i
# include <windows.h>
#endif
```

Specify macro value using command line

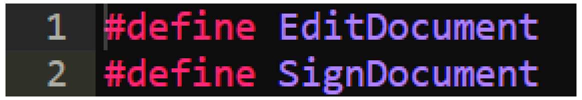


```
clang -DMAX=10 *.c
```

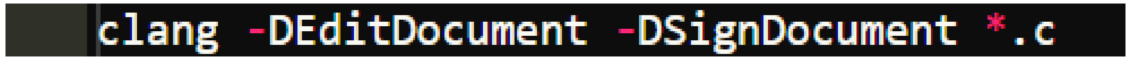
Build for different features
User defined macros in file or command line



```
#ifdef EditDocument
  // code for EditDocument feature
#endif
#ifdef SignDocument
  // code for SignDocument feature
#endif
#ifdef MergeDocument
  // code for MergeDocument feature
#endif
```



```
#define EditDocument
#define SignDocument
```



```
clang -DEditDocument -DSignDocument *.c
```

Exercise 3

```c
// ex3-1.c
#include <stdio.h>
#define A
int main()
{
    printf("A = %d\n", A);
    return 0;
}
```

```c
// ex3-2.c
#include <stdio.h>

int main()
{
    printf("A = %d\n", A);
    return 0;
}
```

- Select all that are true

a) `clang ex3-1.c` does not compile.

b) `clang ex3-1.c` compiles and ./a.out prints "A = " followed by a new line.

c) `clang ex3-2.c` does not compile.

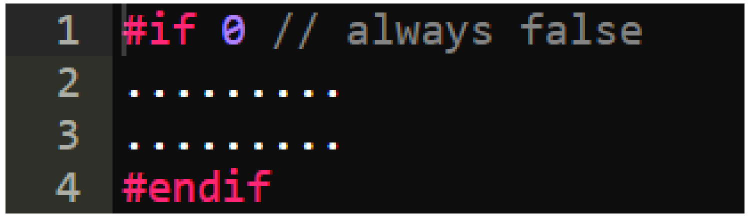
d) `clang -DA ex3-2.c` compiles and the ./a.out prints "A = " followed by a new line.

e) `clang -DA ex3-2.c` compiles and the ./a.out prints "A = 1" followed by a new line.

f) `clang -DA=5 ex3-2.c` compiles and the ./a.out prints "A = 5" followed by a new line.

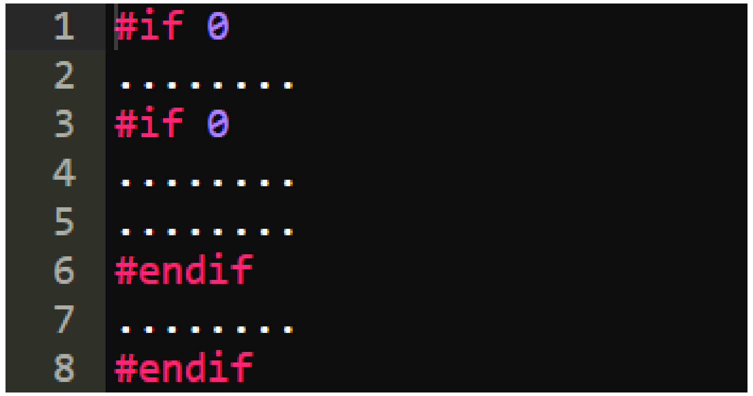
Commenting and Debugging

- We can comment out a block of code, especially if the block contains /* */ block comments



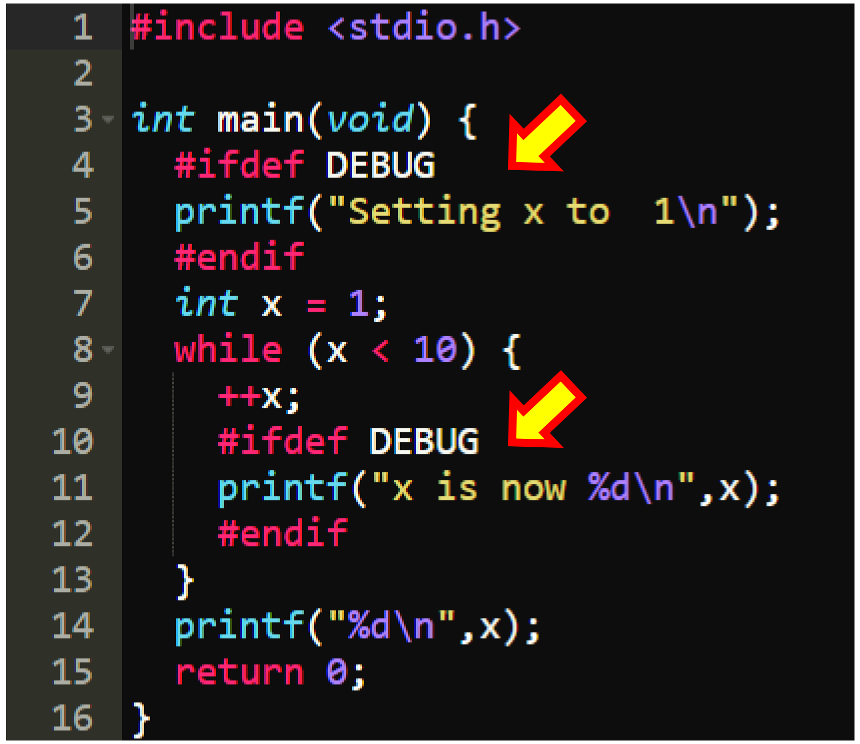
```
#if 0 // always false
..........
..........
#endif
```

- We can nest block comments



```
#if 0
........
#if 0
........
........
#endif
........
#endif
```

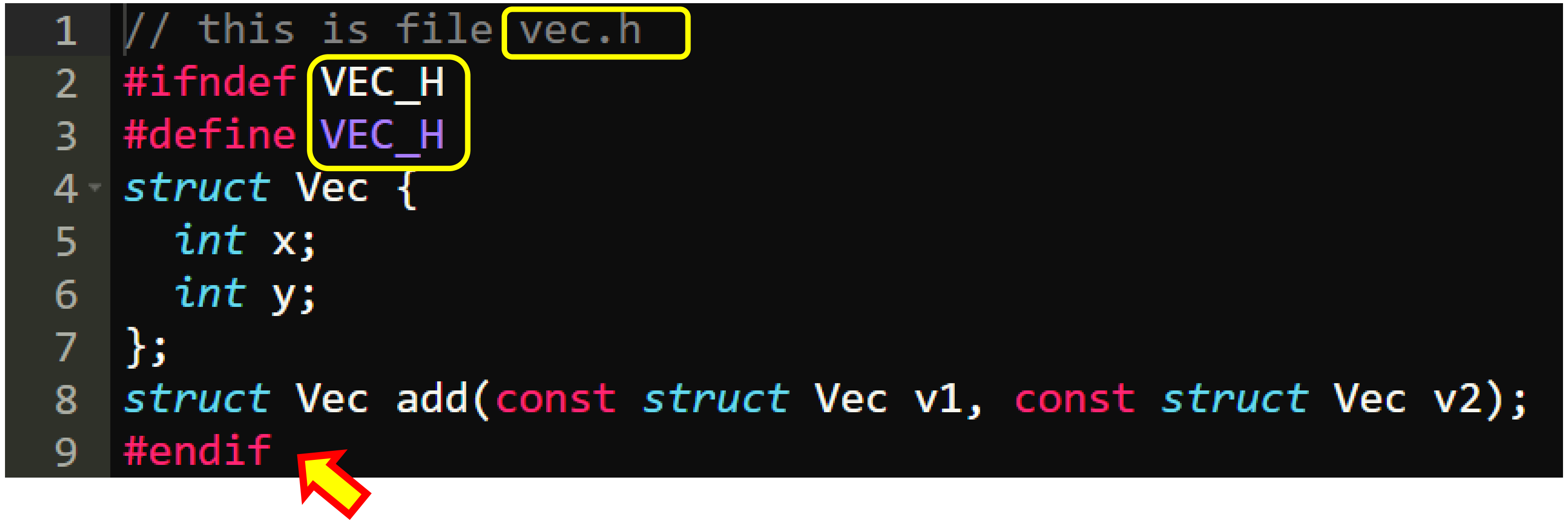
- We can conditional compile debug statement



```
#include <stdio.h>

int main(void) {
  #ifdef DEBUG
  printf("Setting x to  1\n");
  #endif
  int x = 1;
  while (x < 10) {
    ++x;
    #ifdef DEBUG
    printf("x is now %d\n",x);
    #endif
  }
  printf("%d\n",x);
  return 0;
}
```

Include Guards

```
#ifndef UNIQUE_MACRO_NAME
#define UNIQUE_MACRO_NAME

// original header file

#endif
```



```
// this is file vec.h
#ifndef VEC_H
#define VEC_H
struct Vec {
  int x;
  int y;
};
struct Vec add(const struct Vec v1, const struct Vec v2);
#endif
```

Exercise 4

- Suppose we with to write a header le **a_file.h** to be included possibly in multiple files. Which of the following is the standard name for the include guard for this file?

a) `A_FILE.H`
b) `A_FILE_H`
c) `A-FILE-H`
d) `AFILE_H`
e) `AFILE`
f) `a_file_h`
g) A_FILE_DOT_H
h) a_file_dot_h
i) AFILE_DOT_H

Exercise 5

- Which one implements the include guard for **a.h** correctly?

```
//A.
#define A_H
#ifdef A_H
// code here
# endif
```

```
//C.
#ifndef A_H
#define A_H
#endif
// code here
```

```
//B.
#ifndef A_H
#define A_H
// code here
#endif
```

```
//D.
#ifdef A_H
#define A_H
// code here
#endif
```

Discussion

- Why do we want to use conditional compilation?
  - Build for different Operating Systems (cross-platform)
  - Build for different CPU architecture (ARM vs Intel, 32-bit vs 64-bit)
  - Build to include different features
  - Build to include/exclude debugging statement
  - Comment out a block of code (nested comments)
  - Include Guards

Acknowledgement

- Slides by courtesy of Carmen Bruni, Anton Mosunov and Dave Tompkins
- Demo notes from Nomair Naeem
- Demo lectures by Carmen Bruni, Dave Tompkins, and Nomair Naeem

References

- CS 136L edX notes at https://online.cs.uwaterloo.ca/