CSC180: Lecture 25

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

Preprocessor directives: #define value

```
#define N 100
#define PI 3.14159
#define WARNING_MSG "Warning: nonstandard feature"
#define BEGIN {
#define END }
#define BOOL int
```

```
if ( nIndex < N )
BEGIN
      printf( “%s”, WARNING_MSG );
END
```

This is what you see

```
if ( nIndex < 100 )
{
 printf( “%s”, "Warning: nonstandard
                        feature" );
}
```

This is what the compiler see/compile

Macros: undef

Attempting to redefine a macro without un-defining it first is illegal.

Attempting to un-define an undefined macro is legal.

How to undefine a macro?

```
#undef macro-name
```

General Properties of Macros

• One macro may be defined in terms of another:

```
#define PI 3.14159
#define TWO_PI (2*PI)
```

When the preprocessor encounters the symbol TWO_PI later in the program, it replaces it by (2*PI). The preprocessor then rescans the replacement list to see if it contains invocations of other macros (PI in this case). The preprocessor will rescan the replacement list as many times as necessary to eliminate all macro names.

General Properties of Macros

• The preprocessor replaces only entire symbols, not portions of symbols. It ignores macro names embedded in identifiers, character constants, and string literals.

```
#define SIZE 256
char error_msg[] = "Error: SIZE exceeded"; /* not replaced */
...
if (BUFFER_SIZE > SIZE) /* only SIZE is replaced */
printf("%s\n", error_msg);
```

• A macro definition normally remains in effect from the point at which it appears to the end of the file.

Conditional Macro: #if

- #if directive tests an expression to determine whether or not a particular section of text should be included in a program.

- Syntax:
  - #if constant-expression
    statements (could be C statements and/or other # statements)
    #endif

    Or if you have more than one condition:

    #if
    statements
    #elif
    statements
    #else
    statements
    #endif

Conditional Macro: #if

- Rules for *#if, #elif, #else, #endif*
  - Behave exactly like their C counterparts.
  - The test condition:
    - Must evaluate to a constant integer.
    - Will be evaluated as logical T/F condition
    - May contain operators but only in combination w/integer constants.

- Using #if you can selectively incorporates/omits program statements during compilation.

Conditional Macro: #if and defined()

- defined( ) is a function supplied by the preprocessor to check the existence of a macro:
  - #if defined(some-macro-name)
    - Short-hand: #ifdef some-macro-name
  - #if !defined(some-macro-name)
    - Short-hane: #ifndef some-macro-name

Conditional Macro: #if and defined()

```
#if !defined(CUBE)
    #define CUBE(x) ((x)*(x)*(x))
#endif
```

Or

```
#ifndef CUBE
   #define CUBE(x) ((x)*(x)*(x))
#endif
```

Uses of Conditional Compilation

➔ Providing a default definition for a symbol:

```
#ifndef ARRAY_SIZE
#define ARRAY_SIZE 256
#endif

#ifndef NULL
#define NULL 0
#endif

#ifndef ERROR_MSG
#define ERROR_MSG “You have specified an invalid input”
#endif
```

Uses of Conditional Compilation

➔ Including debugging code:

```
#ifdef DEBUG
printf("Value of i: %d\n", i);
printf("Value of j: %d\n", j);
#endif
```

Conditional compilation for debugging

```
#include <stdio.h>

int main() {
        float friction, number;
        unsigned int zip_code;

#ifndef DEBUG
        zip_code = 13285;
#else
        zip_code = 00001;
#endif

        friction = .04;
        number = (zip_code * friction) – 3.2;

#ifdef DEBUG
        printf(“friction: %f  number %f\n”,friction,number);
#endif

        printf(“The final number was %f\n”, number);
        return 0;
}
```

Conditional compilation for debugging

```
> gcc preprocDebug.c -o pdg
> pdg
The final number was 528.199951
> gcc -DDEBUG preprocDebug.c -o pdg
> pdg
friction: 0.040000  number -3.160000
The final number was -3.160000
>
```

```
#include <stdio.h>

int main() {
        float friction, number;
        unsigned int zip_code;

#ifndef DEBUG
        zip_code = 13285;
#else
        zip_code = 00001;
#endif

        friction = .04;
        number = (zip_code * friction) – 3.2;

#ifdef DEBUG
        printf(“friction: %f  number %f\n”,friction,number);
#endif

        printf(“The final number was %f\n”, number);
        return 0;
}
```

Uses of Conditional Compilation

➔ Writing code to run on different machines or under different operating systems:

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

Uses of Conditional Compilation

➔Writing code to work with different flavors of the same library

```
#if defined(STRING_LIB_VER95)
      strncpy(string,string2,limit);
#elif defined(STRING_LIB_VER98)
      strcpy(string1, string2, limit);
#endif
```

Uses of Conditional Compilation

➔ Temporarily disabling code that contains comments:

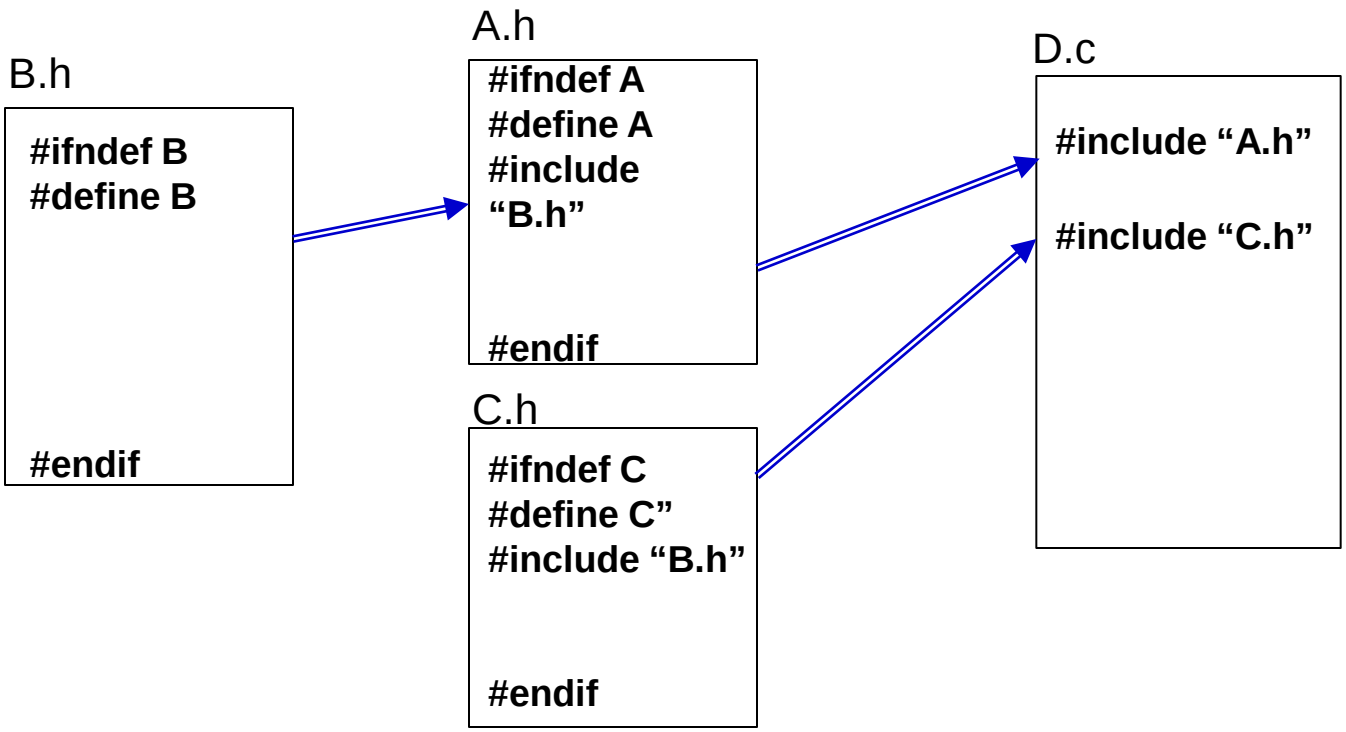
```
#if 0
…..

if( X < 10)
   bkg_color = BLACK;  /*set background color */
 Y = 20;

….
#endif
```

Uses of Conditional Compilation

➔ Protecting header files from being included more than once.

#ifndef in header files (*.h)

- Without the #ifndef lines, the compiler would complain that functions are declared multiple times.
- With the #ifndef lines, the preprocessor would completely ignore B.h the second time it is included.



Predefined Symbolic Constants

```
#include <stdio.h>
  int main(){
     printf("%d\n%s\n%s\n%s\n", __LINE__,
     __FILE__, __DATE__, __TIME__);
  }
```

- Output:

```
3
example.c
Oct 13 2003
19:27:57
```

  - line #, file name, compiled date, compiled time