The Preprocessor (3)

Program Design (II)

2022 Spring

Fu-Yin Cherng
Dept. CSIE, National Chung Cheng University

slido

Which parameterized macro do the following computation: 1 if the product of x and y is less than 100. Otherwise, 0 if the product of x and y is not less than 100.

ⓘ Start presenting to display the poll results on this slide.

slido

#define DOUBLE(x) 2*x

What is the value of DOUBLE(1+2)?

ⓘ Start presenting to display the poll results on this slide.

Outline

- Other Miscellaneous topics about Directives
  - Miscellaneous (adj.): various types or from different sources. 😎
- More Examples of Using Directives to Debug

The # Operator

- Macro definitions may contain two special operators: #
- Neither operator is recognized by the compiler; instead, they're executed during preprocessing.
- it can appear only in the replacement list of a parameterized macro.
- The # operator converts a macro argument into a **string literal**

```
#define PRINT_MSG(n) printf(#n)

int main()
{
    PRINT_MSG(abc);
    // after preprocessor: printf("abc")

    return 0;
}
```

abc

The # Operator

- Useful when debugging!
- Suppose that we decide to use the `PRINT_INT` macro during debugging as a convenient way to print the values of integer variables and expressions.
- The # operator makes it possible for `PRINT_INT` to label each value that it prints.

The # Operator

```c
#include <stdio.h>
#define PRINT_INT(n) printf(#n " = %d\n", n)

int main()
{
    int i = 1, j = 2;
    PRINT_INT(i/j);

    // after preprocessing:
    printf("i/j" " = %d\n", i/j);

    // compiler automatically joins adjacent string literals
    // so equivalent to:
    printf("i/j = %d\n", i/j);

    return 0;
}
```

```
i/j = 0
i/j = 0
i/j = 0


...Program finished with exit code 0
Press ENTER to exit console.
```

Predefined Macros: `__LINE__` and `__FILE__`

- We can use the `__LINE__` and `__FILE__` macros to help locate errors.
  - `__LINE__` : Line number of file being compiled
  - `__FILE__` : Name of file being compiled
- Error-detecting macros are quite useful.

Predefined Macros: __LINE__ and __FILE__

- For example, A macro that can help pinpoint the location of a division by zero

```
#define CHECK_ZERO(divisor) \
      if (divisor == 0) \
        printf("*** Attempt to divide by zero on line %d " \
               "of file %s ***\n", __LINE__, __FILE__)

int main(){
    int i, j, k;
    …
    CHECK_ZERO(j);
    k = i / j;
    return 0;
}
```

The `__func__` Identifier

- `__func__` identifier is another useful tool for debugging
- The `__func__` identifier behaves like a string variable that stores the name of the currently executing function.
- although `__func__` is actually not a macro, we can write some debugging macros such as the following:

The `__func__` Identifier

- These macros can used to trace function calls

```
#define FUNCTION_CALLED() printf("%s called\n", __func__);
#define FUNCTION_RETURNS() printf("%s returns\n", __func__);

void f(void){
     FUNCTION_CALLED();   /* displays "f called" */
     …
     FUNCTION_RETURNS();  /* displays "f returns" */
}
```

The `defined` Operator

- The other operator preprocessor supports: `defined`.
- When applied to an identifier, `defined` produces the value 1 if the identifier is a currently defined macro; it produces 0 otherwise.

The `defined` Operator

- The lines between `#if` and `#endif` will be included only if `DEBUG` is defined as a macro (`defined(DEBUG)` is 1).
- However, you may have questions about why we need `defined()`? Since #if can directly treat the undefined macro as zero

```
#if defined(DEBUG) // the same effct as #if DEBUG
…
#endif
```

The `defined` Operator

- defined adds flexibility!
- `#if` can only test the existence of one macro
- We can now test any number of macros using `#if` with `defined`.
- For example, the following directives checks whether `FOO` and `BAR` are defined but `BAZ` is not defined

```
#if defined(FOO) && defined(BAR) && !defined(BAZ)
```

The `#ifdef` and `#ifndef` Directives

- For conditional compilation
- The `#ifdef` directive tests whether an identifier is currently defined as a macro
- The `#ifndef` directive tests whether an identifier is *not* currently defined as a macro

```
#ifdef identifier //same effect as #if defined(identifier)
#ifndef identifier //same effect as #if !defined(identifier)
```

slido

Suppose that the macro M has been defined as follows:

#define M 10

Which of the following tests will fail (Multiple Choice)?

ⓘ Start presenting to display the poll results on this slide.

Summary

- The `#` Operator
- Predefined Macros: `__LINE__` and `__FILE__`
- The `__func__` Identifier
- The `defined` Operator
- The `#ifdef` and `#ifndef` Directives