IS 0020 Program Design and Software Tools

Preprocessing Lecture 12

April 7, 2005

Preprocessing

 $\sqrt{2}$

- Preprocessing
 - Occurs before program compiled
 - Inclusion of external files
 - Definition of symbolic constants
 - Macros
 - Conditional compilation
 - Conditional execution pf preprocessing directive
 - All directives begin with #
 - Can only have whitespace before directives
 - Directives not C++ statements
 - Do not end with;

The #include Preprocessor Directive

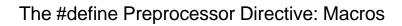
- **#include** directive
 - Puts copy of file in place of directive
 - Two forms
 - #include <filename>
 - For standard library header files
 - Searches pre-designated directories
 - #include "filename"
 - Searches in current directory
 - Normally used for programmer-defined files
- Usage
 - Loading header files
 - #include <iostream>
 - Programs with multiple source files
 - Header file
 - · Has common declarations and definitions
 - Classes, structures, enumerations, function prototypes
 - · Extract commonality of multiple program files

© 2003 Prentice Hall, Inc. All rights reserved.

The #define Preprocessor Directive: Symbolic Constants



- Symbolic constants
 - Constants represented as symbols
 - · When program compiled, all occurrences replaced
- Format
 - #define identifier replacement-text
 - #define PI 3.14159
- Everything to right of identifier replaces text
 - #define PI=3.14159
 - Replaces PI with "=3.14159"
 - · Probably an error
- Cannot redefine symbolic constants
- Advantage: Takes no memory
- Disadvantages
 - Name not seen by debugger (only replacement text)
 - Do not have specific data type
- const variables preferred



- Macro
 - Operation specified in **#define**
 - Macro without arguments
 - Treated like a symbolic constant
 - Macro with arguments
 - Arguments substituted for replacement text
 - · Macro expanded
 - Performs a text substitution
 - No data type checking

© 2003 Prentice Hall, Inc. All rights reserved.

The #define Preprocessor Directive: Macros

Example

```
#define CIRCLE_AREA( x ) ( PI * ( x ) * ( x ) )
area = CIRCLE_AREA( 4 );
   becomes
area = ( 3.14159 * ( 4 ) * ( 4 ) );
```

- Use parentheses
 - Without them,
 #define CIRCLE_AREA(x) PI * x * x
 area = CIRCLE_AREA(c + 2);
 becomes
 area = 3.14159 * c + 2 * c + 2;
 which evaluates incorrectly

© 2003 Prentice Hall, Inc. All rights reserved.

ر

The #define Preprocessor Directive: Macros

Multiple arguments

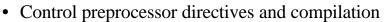
```
#define RECTANGLE_AREA( x, y ) ( ( x ) * ( y ) )
rectArea = RECTANGLE_AREA( a + 4, b + 7 );
  becomes
rectArea = ( ( a + 4 ) * ( b + 7 ) );
```

#undef

- Undefines symbolic constant or macro
- Can later be redefined

© 2003 Prentice Hall, Inc. All rights reserved.

Conditional Compilation



- Cannot evaluate cast expressions, sizeof, enumeration constants
- Structure similar to if

```
#if !defined( NULL )
  #define NULL 0
#endif
```

- Determines if symbolic constant **NULL** defined
- If **NULL** defined,
 - defined(NULL) evaluates to 1
 - #define statement skipped
- Otherwise
 - #define statement used
- Every #if ends with #endif

© 2003 Prentice Hall, Inc. All rights reserved.

8

Conditional Compilation

- Can use else
 - #else
 - #elif is "else if"
- Abbreviations
 - **#ifdef** short for
 - #if defined(name)
 - #ifndef short for
 - #if !defined(name)
- "Comment out" code
 - Cannot use /* ... */ with C-style comments
 - Cannot nest /* */
 - Instead, use

#if 0

code commented out

#endif

- To enable code, change 0 to 1

© 2003 Prentice Hall, Inc. All rights reserved.

Conditional Compilation

Debugging

#define DEBUG 1

#ifdef DEBUG

cerr << "Variable x = " << x << endl;</pre>

#endif

- Defining **DEBUG** enables code
- After code corrected
 - Remove #define statement
 - Debugging statements are now ignored

© 2003 Prentice Hall, Inc. All rights reserved.

10

The #error and #pragma Preprocessor Directives

11

- #error tokens
 - Prints implementation-dependent message
 - Tokens are groups of characters separated by spaces
 - #error 1 Out of range error has 6 tokens
 - Compilation may stop (depends on compiler)

• #pragma tokens

- Actions depend on compiler
- May use compiler-specific options
- Unrecognized **#pragmas** are ignored

© 2003 Prentice Hall, Inc. All rights reserved.

The # and ## Operators

12

- # operator
 - Replacement text token converted to string with quotes #define HELLO(x) cout << "Hello, " #x << endl;</p>
 - **HELLO(JOHN)** becomes
 - cout << "Hello, " "John" << endl;
 - Same as cout << "Hello, John" << endl;
- ## operator
 - Concatenates two tokens #define TOKENCONCAT(x, y) x ## y
 - TOKENCONCAT (O, K) becomes
 - OK

Line Numbers



• #line

- Renumbers subsequent code lines, starting with integer
 - #line 100
- File name can be included
- #line 100 "file1.cpp"
 - Next source code line is numbered 100
 - For error purposes, file name is "file1.cpp"
 - Can make syntax errors more meaningful
 - Line numbers do not appear in source file

 $\hbox{@ 2003 Prentice Hall, Inc. }$ All rights reserved.

Predefined Symbolic Constants



- Five predefined symbolic constants
 - Cannot be used in **#define** or **#undef**

Symbolic constant	Description
LINE	The line number of the current source code line (an integer constant).
FILE	The presumed name of the source file (a string).
DATE	The date the source file is compiled (a string of the form "Mmm dd yyyy" such as "Jan 19 2001").
TIME	The time the source file is compiled (a string literal of the form "hh:mm:ss").

Assertions

15

- assert is a macro
 - Header <cassert>
 - Tests value of an expression
 - If O (false) prints error message, calls abort
 - Terminates program, prints line number and file
 - Good for checking for illegal values
 - If 1 (true), program continues as normal
 - assert(x <= 10);</pre>
- To remove **assert** statements
 - No need to delete them manually
 - #define NDEBUG
 - All subsequent assert statements ignored