CS 342: Object-Oriented Software Development Lab

The C++ Preprocessor

Shawn M. Hannan
Department of Computer Science
Washington University, St. Louis
hannan@cs.wustl.edu

http://classes.cec.wustl.edu/~cs342/

The C++ Preprocessor

- What does the preprocessor do?
- Preprocessor directives
- For more information

Preprocessor Responsibilities

- Header file inclusion
  - Supports factoring of common code, notably declarations
- Macro expansion
  - Supports compile-time decisions
- Conditional compilation
  - Supports platform dependencies
  - Supports compile-time application configuration
- Miscellaneous
  - Supports source file name and line number access

Header File Inclusion

- The **`#include`** directive has two forms, *e.g.*,
  - **`#include <stdio.h>`** for system header files
  - **`#include "Stack.h"`** for other (application) headers
- The two forms differ in the include path and where subsequent headers are included from (with some compilers/options)
  - Specify the include path using **`-I`** compiler options
  - The include path is searched, in order, for each header
  - The compiler usually provides an implicit path to search for system headers
  - The most useful distinctions are 1) documentation, and 2) tool usages, such as dependency generation for Makefiles

Macro Expansion

- The **#define** directive creates a macro, *e.g.*,
  - **#define BUFSIZ 1024**
- Any occurrence of a macro is expanded in place, *e.g.*,
  - All (complete) occurrences of **BUFSIZ** will be replaced by **1024**
- Usually cleaner to use C++ static constants. And C++ constants are type-checked, while **#define**s are not.
- Can disable a macro with **#undef**, but that can be dangerous. It's mostly used for disabling troublesome macros in system header files.

Macro Example

- ACE **#define**s a macro for the size of each built-in type:

```
// The number of bytes in a long.
# if !defined (ACE_SIZEOF_LONG)
#   if (ULONG_MAX) == 65535UL
#     define ACE_SIZEOF_LONG 2
#   elif ((ULONG_MAX) == 4294967295UL)
#     define ACE_SIZEOF_LONG 4
#   elif ((ULONG_MAX) == 18446744073709551615UL)
#     define ACE_SIZEOF_LONG 8
#   else
#     error: unsupported long size, update for this plat
#   endif /* ULONG_MAX */
# endif /* !defined (ACE_SIZEOF_LONG) */
```


Macro Example, (cont'd)

- These are used to create platform-independent types of known sizes, *e.g.*,

```
# if ACE_SIZEOF_INT == 4
  typedef int ACE_INT32;
  typedef unsigned int ACE_UINT32;
# elif ACE_SIZEOF_LONG == 4
  typedef long ACE_INT32;
  typedef unsigned long ACE_UINT32;
# else
#   error Have to add to the ACE_UINT32 type setting
# endif
```

- **#if defined** and **#if ! defined** are functionally equivalent to **#ifdef** and **#ifndef**, respectively.

Conditional Compilation

- **#if**, **#elif**, **#else**, **#endif**, *e.g.*,

```
#if SIZE == 1
  [...]
#elif SIZE == 2
  [...]
#else
  [...]
#endif
```

- Header file include protection:

```
#ifndef STACK_H
#define STACK_H

[...]

#endif /* STACK_H */
```

Conditional Compilation, (cont'd)

- Can use conditional compilation to disable blocks of code.

```
#define DEBUG 1
 .
 .
 .
#if DEBUG
  cout << ''Value of x is '' << x << endl;
#endif /* DEBUG */
```


Miscellaneous

- **__FILE__** and **__LINE__** are useful predefined macros.
  - Contain the current filename and line number, respectively.
  - **cout<<''at ''<<__LINE__<<'' in ''<<__FILE__<<endl;**
- Other, less useful, predefined macros include **__DATE__** and **__TIME__** (of compilation).

Miscellaneous

- **#error** causes unconditional compilation failure, with message to user, *e.g.*,

```
#if ! defined (DEFAULT_SIZE)
# error DEFAULT_SIZE was not defined!
#endif /* ! DEFAULT_SIZE */
```

- **#pragma** can be used for compiler-dependent features, *e.g.*, to disable a specific warning or instantiate a template

The C Preprocessor and C++

- C++ tries to reduce reliance on the preprocessor
  - Typed constants are type-safer than macros.

```
#define MAX_AGE 100
          ...
const int MAX_AGE = 100;
```

  - The preprocessor complicates debugging, because the debugger sees the preprocessor output, not source code input.
- The preprocessor is typically used to enhance performance and improve portability
- Static consts and templates provide powerful compile-time alternatives.
- C++ relies on a rich set of headers for portability.

For More Information

- `man cpp`
- `info cpp`, for information on GNU `cpp`

