CSCI 390 – Special Topics in C++

Lecture 16 (10/11/18)

Time To Turn Off Cell Phones

Consider a Flu Shot
You Don't Want To Be Sick During Finals

C++ Preprocessor
Defining Simple User Macros

- Define syntax:
  - **`#define`** <macro id> <macro text>
  - By convention, <macro ids> are uppercase so that the reader knows it is a macro.
- Once defined, works just like predefined macros.
- Undefine syntax:
  - **`#undef`** <macro id>

Example User Defined Macro Expansion

```
#include <iostream>
using std::cout;  using std::endl;

int main(void)
{
  #define GREETING "Boo!"
  cout <<"GREETING: " << GREETING << endl;
  #undef GREETING

  #define GREETING "Hi!"
  cout <<"GREETING: " << GREETING << endl;
  #undef GREETING

  return 0;
}
```

```
GREETING: Boo!
GREETING: Hi!
```

C++ Preprocessor
Defining Parameterized Macros

- Define syntax:
  - **`#define`** <macro id>**(**<parm list>**)** <macro text>
  - By convention, <macro ids> are uppercase so that the reader knows it is a macro.
- Once defined, invocations must include parm list. Works just like predefined macros.
- Undefine syntax:
  - **`#undef`** <macro id>

Example Parameterized Macro Expansion

```
#include <iostream>
using std::cout;  using std::endl;

#include "Helper.h"

int main(void)
{
  #define GREET(TYPE, GREETING) cout << TYPE " GREETING: " << GREETING << endl;

  GREET("Halloween", "Boo!")
  // Expands to: cout << "Halloween" " GREETING: " << "Boo!" << endl;
  // Which is: cout << "Halloween GREETING: " << "Boo!" << endl;
  GREET(   "Normal", "Hi!")
  GREET(  "Walmart", "Welcome to Walmart.")

  return 0;
}


Halloween GREETING: Boo!
Normal GREETING: Hi!
Walmart GREETING: Welcome to Walmart.
```

C++ Preprocessor Stringizing Tokens

- Any token prefixed by # goes through replacement and the result is enclosed in quotes.
  - This is a unary operator.

Example Parameterized Macro Expansion

```
#include <iostream>
using std::cout;  using std::endl;

int main(void)
{
  #define SHOWVAR(VAR) #VAR ": " << VAR

  auto Hello{"Hello world!"};

  cout << SHOWVAR(Hello) << endl;

  return 0;
}

Hello: Hello world!
```

C++ Preprocessor Concatenating Tokens

- <token1> **##** <token2>
  - Both <token1> and <token2> undergo replacement and then the strings are concatenated (placed back to back).

Example Concatenation Macro Expansion

```
#include <iostream>
using std::cout;  using std::endl;

int main(void)
{
  auto HelloWorld{"Hello world!"};
  auto GoodMorning{"Good morning!"};

  #define SHOWVAR(LEFT, RIGHT) #LEFT #RIGHT ": " << LEFT ## RIGHT


  cout << SHOWVAR(Hello, World) << endl;
  // Expands to: "Hello" "World" ": " << HelloWorld << endl;
     cout << SHOWVAR(Good, Morning) << endl;

  return 0;
}

HelloWorld: Hello world!
GoodMorning: Good morning!
```

C++ Preprocessor
#if ... #else ... #endif

- #if … #else ... #endif can be used to conditionally include/exclude source.
- Most common form:
  - **`#ifdef`** <macro id> or **`#ifndef`** <macro id>
  - Used to include/exclude source if <macro id> is defined/not defined.
  - **`#if defined(`**<macro id>**`)`** or
    `#if !defined(`<macro id>**`)`**
- Alternate form:
  - **`#if`** <expression>

Example Concatenation Macro Expansion

```
#include <iostream>
using std::cout;  using std::endl;

int main(void)
{
  #ifdef __cplusplus
    cout << "Running C++!" << endl;
  #endif

  #if __cplusplus >= 201103L
    cout << "Running at least C++ 11." << endl;
  #else
    cout << "Running old version of C++." << endl;
  #endif

  return 0;
}

Running C++!
Running at least C++ 11.
```

C++ Preprocessor **`#include`**

- **`#include`** can be used to include contents of file. Usually this is an interface (.h) file.
  - **`#include`** <path> -- includes file from system library.
  - **`#include`** **"**path**"** -- includes file from user path. If not found, attempt to include from system library.

C++ Preprocessor **`#include`**

- An **`#include`** can usually be included only once per compilation unit.
  - Standard trick #1 (Not C++, but usually available):

```
#pragma once

// Included source goes here.
// Only first time is included.
```

  - Standard trick #2 (Always available):

```
// file.h
#ifdef file_h
#define file_h

// Included source goes here.
// Only first time is included.

#endif
```

C++ STL
Standard Template Library

- **Mastering the STL is essential to mastering C++.**
- The important STL components are “containers”.
  - **`std::string`** (contains characters)
  - **`std::vector<T>`** (contains an array of type T)
  - **`std::deque<T>`** `(contains a double ended queue of type T)`
  - **`std::list<T>`** `(contains a doubly linked list of type T)`
  - **`std::forward_list<T>`** `(contains a singly linked list of type T)`

C++ STL
Standard Template Library

- The important STL components are “containers”.
  - **std::stack<T>** (contains a stack of type T)
  - **std::queue<T>** (contains a queue of type T)
  - **std::priority_queue<T>** (contains a priority queue of type T)
  - **std::map<Key, T>** (contains a dictionary of type **T**, identified and <u>sorted</u> by **Key**)
  - **std::unordered_map<Key, T>** (contains a <u>hash</u> of type **T**, identified by **Key**)

C++ STL
Iterators

- The various containers implement a mechanism for traversing the contained members.
- That mechanism is called an iterator.
  - They are very efficient.
- There are three types of itererators:
  - Forward
  - Bidirectional
  - Random
- Will be discussed with each container.

C++ STL
Standard Template Library

- A good summary can be found here: http://www.cplusplus.com/reference/stl/
- Good std::string summary can be found here: https://en.cppreference.com/w/cpp/string/basic_string

C++ STL
std::string

- Header: #include <string>
- Iterator: Random Access
- Purpose: Works like strings in other language, but no garbage collection.
- Lot's of handy constructors.

std::string Example

```
#include <iostream>
using std::cout;  using std::endl;

#include <string>
using std::string;

int main(void)
{
  // Common constructor
  string Hi{"Hello"};

  // operator << overload already provided.
  cout << Hi << endl;

  // operator+ does concatenation
  cout << Hi + " World!" << endl;

  // operator+= implemented.
  Hi += " World!";
  cout << Hi << endl;

  return 0;
}
```

```
Console Log:
Hello
Hello World!
Hello World!
```

std::string/for Example

```
#include <iostream>
using std::cout;  using std::endl;

#include <string>
using std::string;

int main(void)
{
  // Common constructor
  string Hi{"Hi"};

  for(auto i = 0u; i < Hi.size(); ++i)
  {
    cout << "Char: " << Hi[i] << endl;
  }

  return 0;
}
```

```
Console Log:
Char: H
Char: i
```

Range-Based for

- Syntax:
  `for ( <range_declaration> : <range_expression> ) <loop_body>`
- Handy for containers.

std::string/for Example

```
#include <iostream>
using std::cout;  using std::endl;

#include <string>
using std::string;

int main(void)
{
  // Common constructor
  string Hi{"Hi"};

  for (auto c: Hi)
  {
    cout << "Char: " << c << endl;
    // c is a copy, not a reference!
    c = 'x';
  }

  // So, no changes made to Hi
  cout << "First Hi: " << Hi << endl;

  for (auto &c: Hi)
  {
    // c is a reference.  This changes Hi.
    c = 'x';
  }

  cout << "Second Hi: " << Hi << endl;

  return 0;
}
```

```
Console Log:
Char: H
Char: i
First Hi: Hi
Second Hi: xx
```