UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 150 *Fundamentals of Programming*

Compile-time errors

ECE150

Douglas Wilhelm Harder, M.Math., LEL
Prof. Hiren Patel, Ph.D., P.Eng.
Prof. Werner Dietl, Ph.D.

Outline

- In this lesson, we will:
  - Define compile-time errors
  - Look at many different examples where mistakes in coding result in compile-time errors

Compile-time errors

- Sometimes, in English, you can say something wrong, but the person listening to you will understand what you meant
  - If you're unlucky, they'll misunderstand you…
- The compiler is not so forgiving: if you enter C++ code that does not make sense within the C++ programming language, the compiler will simply tell you
  - Fortunately, it often tries to help you understand what it is confused about
  - It will never make assumptions about what you meant; after all, if it "guessed wrong", you may have the firmware on your pacemaker product malfunctioning with rather catastrophic consequences

Compile-time errors

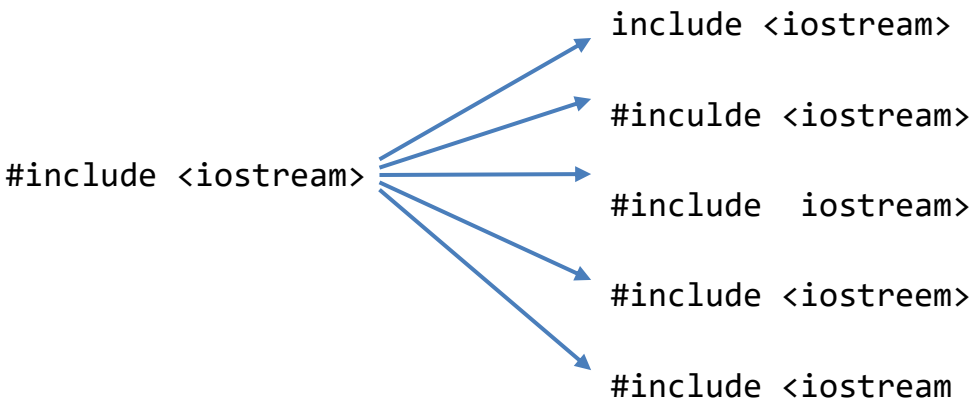
- We will take working code, and introduce errors to see how the compiler responds:

```
#include <iostream>

// Function declarations
int main();
double my_sin( double x );

// Function definitions
int main() {
    std::cout << "Hello world!" << std::endl;
    std::cout << "sin(0.5) = " << my_sin(0.5) << "!" << std::endl;

    return 0;
}

double my_sin( double x ) {
    // This uses a Taylor series approximation of sin(x)
    return ((-0.00019841269841269841*x*x
            + 0.008333333333333333)*x*x
            - 0.16666666666666667)*x*x + 1.0;
}
```

Note that the response of the compiler will change from compiler to compiler and version to version, so do not memorize these error messages; instead, learn how to read them

Line 1

- Let's make some changes to the first line:



- These authors have made each of these mistakes from time-to-time…

Line 1: `include <iostream>`

In `example.cpp`, on line 1 starting at column 1: look for the ^

- The error message:

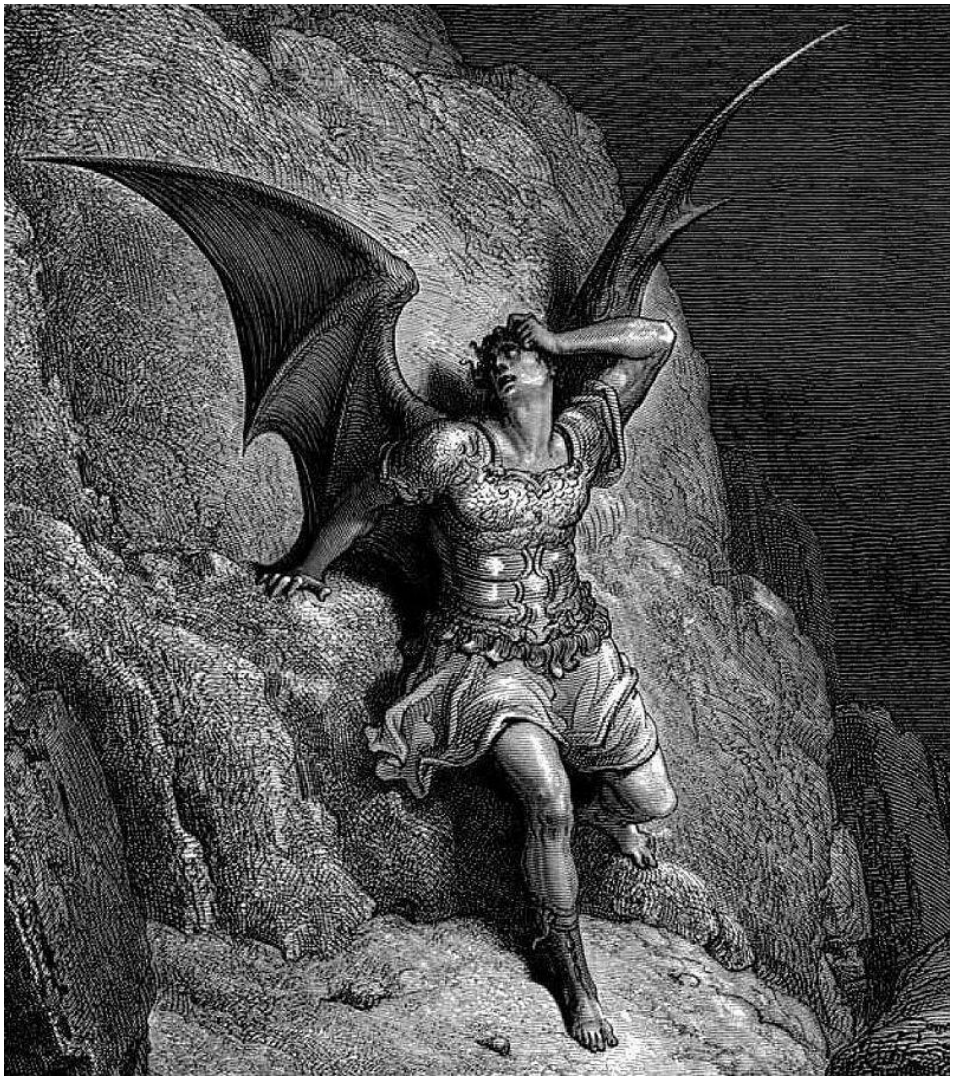
```
example.cpp:1:1: error: 'include' does not name a type
 include <iostream>
 ^
example.cpp: In function 'int main()':
example.cpp:7:5: error: 'cout' is not a member of 'std'
     std::cout << "Hello world!" << std::endl;
     ^
example.cpp:7:36: error: 'endl' is not a member of 'std'
     std::cout << "Hello world!" << std::endl;
                                    ^
example.cpp:8:5: error: 'cout' is not a member of 'std'
     std::cout << "sin(0.5) = " << my_sin(0.5) << "!" << std::endl;
     ^
example.cpp:8:57: error: 'endl' is not a member of 'std'
     std::cout << "sin(0.5) = " << my_sin(0.5) << "!" << std::endl;
                                                         ^
```

- Always look at the first error message first
  - The compiler is trying to interpret `include` as a type like `int` or `double`

Line 1: `include <iostream>`

- With so many error messages, you may think you just committed a cardinal sin…
- Don't despair: Often fixing the first error will eliminate many of the others

Baron Cohen as Borat



Gustave Doré, *Lucifer*

Line 1: `inculde <iostream>`

- The error message:

```
example.cpp:1:2: error: invalid preprocessing directive #inculde
 #inculde <iostream>
  ^
example.cpp: In function 'int main()':
example.cpp:7:5: error: 'cout' is not a member of 'std'
     std::cout << "Hello world!" << std::endl;
     ^
example.cpp:7:36: error: 'endl' is not a member of 'std'
     std::cout << "Hello world!" << std::endl;
                                    ^
example.cpp:8:5: error: 'cout' is not a member of 'std'
     std::cout << "sin(0.5) = " << my_sin(0.5) << "!" << std::endl;
     ^
example.cpp:8:57: error: 'endl' is not a member of 'std'
     std::cout << "sin(0.5) = " << my_sin(0.5) << "!" << std::endl;
                                                         ^
```

- Again, look at the first error message first
  - The compiler does not recognize `#inculde` as a preprocessing directive

Line 1: #include iostream>

- The error message:

```
example.cpp:1:11: error: #include expects "FILENAME" or <FILENAME>
 #include  iostream>
           ^
example.cpp: In function 'int main()':
example.cpp:7:5: error: 'cout' is not a member of 'std'
     std::cout << "Hello world!" << std::endl;
     ^
example.cpp:7:36: error: 'endl' is not a member of 'std'
     std::cout << "Hello world!" << std::endl;
                                    ^
example.cpp:8:5: error: 'cout' is not a member of 'std'
     std::cout << "sin(0.5) = " << my_sin(0.5) << "!" << std::endl;
     ^
example.cpp:8:57: error: 'endl' is not a member of 'std'
     std::cout << "sin(0.5) = " << my_sin(0.5) << "!" << std::endl;
                                                         ^
```

- The included file must be started with either a " or a <

Line 1: `#include <iostraem>`

- The error message:

```
example.cpp:1:20: fatal error: iostraem: No such file or directory
 #include <iostraem>
                    ^
```

  - If the file cannot be found, chances are it is misspelled

Line 1: `#include <iostream`

- The error message:

```
example.cpp:1:19: error: missing terminating > character
 #include <iostream
                   ^
```

  - The error message here is very clear

Line 4: `integer main();`

- Suppose you forget that the type is `int`, and instead use `integer`

```
// Function declarations
integer main();
double my_sin( double x );
```

In `example.cpp`, on line 4 starting at column 1…

  - The error message is clear, too:

```
example.cpp:4:1: error: 'integer' does not name a type
 integer main();
 ^
```

Line 4: `int Main();`

- Some programming languages use `Main()`

  Suppose you forgot you were using C++:

```
// Function declarations
int Main();
double my_sin( double x );

// Function definitions
int Main() {
    …
```

- The error message is initially unclear, but the point is made:

```
/usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../../../lib64/crt1.o: In function '_start':
(.text+0x20): undefined reference to 'main'
collect2: error: ld returned 1 exit status
```

- Our function `int Main()` is a perfectly good function
  - It's just not the one that C++ executes when launching a program

Line 5: `double my_sin(…)`

- Suppose you forgot the semicolon after the function declaration:

```
// Function declarations
int main();
double my_sin( double x )

// Function definitions
int main() {
    …
```

- The error message is less clear:

```
example.cpp:8:1: error: expected initializer before 'int'
 int main() {
 ^
```

- It's trying to understand:

```
double my_sin( double x ) int main() { …
```

Line 9 & 10: cout << …

- Suppose you forget the namespace `std::`

```
cout << "Hello world!" << endl;
cout << "sin(0.5) = " << mysin(0.5) << "!" << endl;
```

- The compiler makes suggestions:

```
example.cpp: In function 'int main()':
example.cpp:9:5: error: 'cout' was not declared in this scope
     cout << "Hello world!" << endl;
     ^
example.cpp:9:5: note: suggested alternative:
In file included from example.cpp:1:0:
/usr/include/c++/4.8.2/iostream:61:18: note:   'std::cout'
   extern ostream cout;  /// Linked to standard output
                  ^
example.cpp:9:31: error: 'endl' was not declared in this scope
     cout << "Hello world!" << endl;
                              ^
example.cpp:9:31: note: suggested alternative:
In file included from /usr/include/c++/4.8.2/iostream:39:0,
                 from example.cpp:1:
/usr/include/c++/4.8.2/ostream:564:5: note:   'std::endl'
     endl(basic_ostream<_CharT, _Traits>& __os)
     ^
```

Line 9: no opening quote

- Suppose you forgot an opening quote:

```
std::cout << Hello world!" << std::endl;
```

- The error message is somewhat complicated:

```
example.cpp:9:30: warning: missing terminating " character [enabled by default]
     std::cout << Hello world!" << std::endl;
                              ^
example.cpp:9:5: error: missing terminating " character
     std::cout << Hello world!" << std::endl;
     ^
example.cpp: In function 'int main()':
example.cpp:9:18: error: 'Hello' was not declared in this scope
     std::cout << Hello world!" << std::endl;
                  ^
example.cpp:9:24: error: expected ';' before 'world'
     std::cout << Hello world!" << std::endl;
                           ^
```

Line 9: no closing quote

- Suppose you forgot a closing quote:

```
std::cout << "Hello world! << std::endl;
```

- The error message is straight-forward:

```
example.cpp:9:18: warning: missing terminating " character [enabled by default]
     std::cout << "Hello world! << std::endl;
                  ^
example.cpp:9:5: error: missing terminating " character
     std::cout << "Hello world! << std::endl;
     ^
```

Line 10: no closing quote

- Suppose you forgot a different closing quote:

```
std::cout << "sin(0.5) =  << my_sin(0.5) << "!" << std::endl;
```

- The error message is somewhat confusing:

```
example.cpp:10:51: warning: missing terminating " character [enabled by default]
     std::cout << "sin(0.5) =  << my_sin(0.5) << "!" << std::endl;
                                                   ^
example.cpp:10:5: error: missing terminating " character
     std::cout << "sin(0.5) =  << my_sin(0.5) << "!" << std::endl;
     ^
example.cpp: In function 'int main()':
example.cpp:10:50: error: expected ';' before '!' token
     std::cout << "sin(0.5) =  << my_sin(0.5) << "!" << std::endl;
```

- The last message recognizes **!** as a unary operator
  - For this line to make any sense, the previous statement must end before the **!** operator

```
std::cout << "sin(0.5) =  << my_sin(0.5) << ";
!" << std::endl;
```

Line 10: misspelled identifiers

- Suppose you misspelled the function identifier `my_sin`:

```
std::cout << "sin(0.5) = " << mysin(0.5) << "!" << std::endl;
```

- The error message is less clear:

```
example.cpp: In function 'int main()':
example.cpp:10:44: error: 'mysin' was not declared in this scope
     std::cout << "sin(0.5) = " << mysin(0.5) << "!" << std::endl;
                                            ^
```

  - The issue is clear: the compiler does not know what `mysin` is…

Line 15: unmatched definition

- Suppose the function declaration and definition don't match in the return type:

```
int my_sin( double x ) {
```

- The error message points out the ambiguity:

```
example.cpp: In function 'int my_sin(double)':
example.cpp:15:22: error: new declaration 'int my_sin(double)'
 int my_sin( double x ) {
                      ^
example.cpp:5:8: error: ambiguates old declaration 'double my_sin(double)'
 double my_sin( double x );
        ^
```

  - The return types of the function declaration and definition must match

Line 15: unmatched definition

- Suppose function declaration and definition don't match in the parameter types:

```
double my_sin( int x ) {
```

- The error message points out the ambiguity:

```
/tmp/cc46i29P.o: In function `main':
example.cpp:(.text+0x39): undefined reference to 'my_sin(double)'
collect2: error: ld returned 1 exit status
```

- It was fine with you defining a `my_sin` taking an `int`,
  but it's looking for a `my_sin` taking a `double`

Line 16: invalid comments

- Suppose you accidentally used / for a comment:

```
/ This uses a Taylor series approximation of sin(x)
```

- The error message is less clear:

```
example.cpp: In function 'double my_sin(double)':
example.cpp:16:5: error: expected primary-expression before '/' token
     / This uses a Taylor series approximation of sin(x)
     ^
example.cpp:16:7: error: 'This' was not declared in this scope
     / This uses a Taylor series approximation of sin(x)
       ^
example.cpp:16:12: error: expected ';' before 'uses'
     / This uses a Taylor series approximation of sin(x)
            ^
```

- The compiler is interpreting the / as a division sign
  - As division is a binary operator, it needs a left operand

Line 17: unmatched opening parenthesis

- Suppose you forget a closing parenthesis:

```
return  ((-0.00019841269841269841*x*x
        + 0.008333333333333333 *x*x
        - 0.16666666666666667)*x*x + 1.0;
```

- Its suggesting you add a closing parenthesis, but in the wrong location:

```
example.cpp: In function 'double my_sin(double)':
example.cpp:19:45: error: expected ')' before ';' token
            - 0.16666666666666667)*x*x + 1.0;
                                            ^
```

Line 17: unmatched closing parenthesis

- Suppose you forgot an opening parenthesis:

```
return  (-0.00019841269841269841*x*x
        + 0.008333333333333333)*x*x
        - 0.16666666666666667)*x*x + 1.0;
```

- The error message suggests the statement ends after the 7:

```
example.cpp: In function 'double my_sin(double)':
example.cpp:19:34: error: expected ';' before ')' token
        - 0.16666666666666667)*x*x + 1.0;
                              ^
example.cpp:19:34: error: expected primary-expression before ')' token
example.cpp:19:34: error: expected ';' before ')' token
```

  - The suggestion is wrong, but the compiler doesn't know your intentions

Line 17: unknown identifiers

- Suppose you forgot the * and used juxtaposition for multiplication:

```
return ((-0.00019841269841269841*xx
        + 0.008333333333333333)*x*x
        - 0.16666666666666667)*x*x + 1.0;
```

- The error message says nothing about the missing *; instead:

```
example.cpp: In function 'double my_sin(double)':
example.cpp:17:38: error: 'xx' was not declared in this scope
     return ((-0.00019841269841269841*xx
                                      ^
```

  - Its just saying: “I have no clue what `'xx'` is…”

Line 17: space for multiplication

- Suppose you forgot the * and used a space:

```
return ((-0.00019841269841269841*x x
         + 0.008333333333333333)*x*x
         - 0.16666666666666667)*x*x + 1.0;
```

- The error message is a little unhelpful:

```
example.cpp: In function 'double my_sin(double)':
example.cpp:17:40: error: expected ')' before 'x'
     return ((-0.00019841269841269841*x x
                                        ^
example.cpp:19:45: error: expected ')' before ';' token
             - 0.16666666666666667)*x*x + 1.0;
                                             ^
```

  - It suggests there should be a closing parenthesis after the first `'x'`
    - After this, the second error message is confusing

Line 17: unmatched braces

- Students often forget to close braces at the end of functions:
- The error message is very clear:

```
example.cpp: In function 'double my_sin(double)':
example.cpp:19:45: error: expected '}' at end of input
             - 0.16666666666666667)*x*x + 1.0;
                                             ^
```

- It's suggesting you put the closing brace at the end of the line
  - The start of the next line is fine

```
double my_sin( double x ) {
    // This uses a Taylor series approximation of sin(x)
    return ((-0.00019841269841269841*x*x
            + 0.008333333333333333)*x*x
            - 0.16666666666666667)*x*x + 1.0; }
```

Summary

- Following this lesson, you now:
  - Understand that some mistakes lead to code that cannot be compiled
  - Understand the compiler makes attempts to point out where the issue is
    - It may be wrong…
  - Know to always try to fix the first compile-time error first
    - That may fix subsequent errors

References

[1] Wikipedia
https://en.wikipedia.org/wiki/Compilation_error

[2] cplusplus.com: list of preprocessing directives
http://www.cplusplus.com/doc/tutorial/preprocessor/

Acknowledgments

Proof read by Dr. Thomas McConkey.

Juliette Rocco for pointing out its absence.

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

Disclaimer

These slides are provided for the ECE 150 *Fundamentals of Programming* course taught at the University of Waterloo. The material in it reflects the authors' best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.