Error Handling

CMSC 240 Software Systems Development

Today

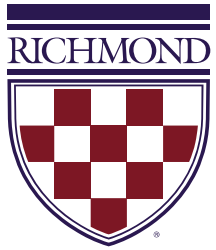
- Errors
- Exception Handling with Try/Catch

RICHMOND


Errors
RICHMOND

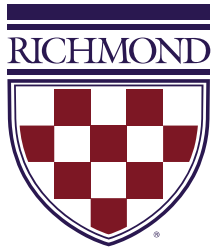
"**Avoiding**, **finding**, and **correcting errors** is **95%** or more of the effort for serious software development."

– Bjarne Stroustrup



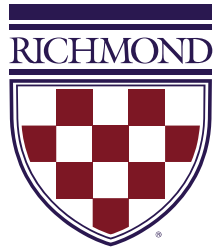
Common Sources of Errors

- Poor specification
  - "What's this supposed to do?"
- Unexpected arguments
  - "but sqrt() isn't supposed to be called with -1 as its argument"
- Unexpected input
  - "but the user was supposed to input an integer"
- Code that simply doesn't do what it was supposed to do

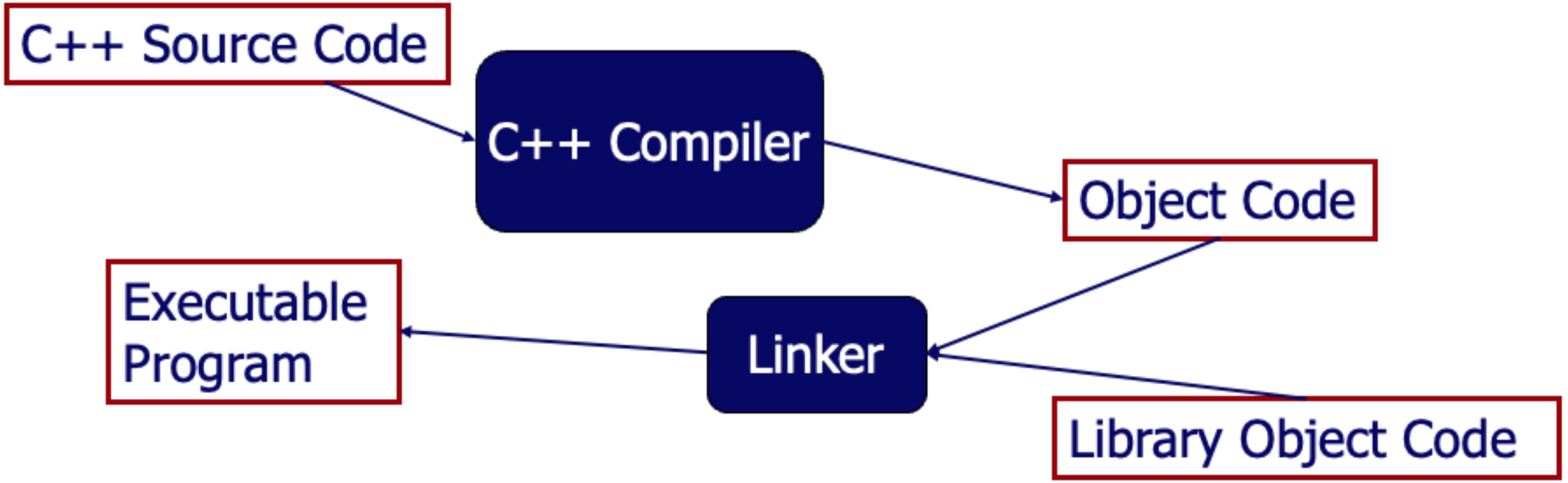


Kinds of Errors

- Compile-time errors
  - Syntax errors
  - Type errors
- Link-time errors
- Run-time errors
  - Detected by user code (code fails while running)
- Logic errors
  - Detected by programmer (code runs, but produces incorrect output)

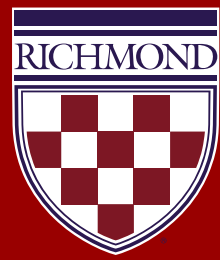


C++ compilation and linking



RICHMOND

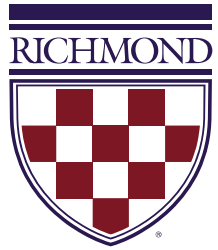
Compiler Error Demo



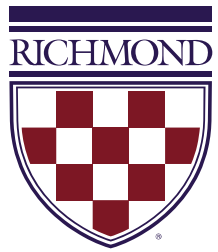
Checking Your Inputs

- One way to reduce errors is to validate your inputs
  - **Before** trying to use an input value, check that it meets your expectations/requirements

- For example:
  - Function arguments
  - Data from input (`istream`, `fstream`)



```
1   int area(int length, int width)
2   {
3       return length * width;
4   }
5
6   int main()
7   {
8       // error: wrong number of arguments
9       int result1 = area(7);
10
11      // error: 1st argument has a wrong type
12      int result2 = area("seven", 2);
13
14      // ok
15      int result3 = area(7, 10);
16
17      // ok, but dangerous: 7.5 truncated to 7
18      int result4 = area(7.5, 10);
19
20      // ok, but the values make no sense!
21      int result5 = area(10, -7);
22
23      return 0;
24  }
```

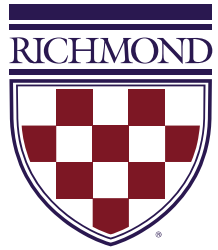


Bad Function Arguments

- What do we do in cases like this, where the types are correct, but the values don't make sense:

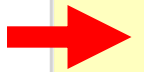
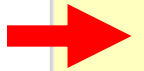
```
// ok, but the values make no sense!
int result5 = area(10, -7);
```

- Alternatives:
  - Just don’t do that
    - Hard to control all situations
  - The **caller** of the function can check
    - Get’s messy, and is **hard to accomplish systematically**



Caller Validates

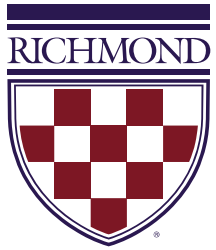
```
14      // Caller validates the inputs
15      if (length <= 0)
16      {
17          cerr << "Non-positive length value." << endl;
18          exit(1);
19      }
20
21      if (width <= 0)
22      {
23          cerr << "Non-positive length value." << endl;
24          exit(1);
25      }
26
27      int result = area(length, width);
28
29      cout << "Area == " << result << endl;
```


RICHMOND

Bad Function Arguments

- The **function** should check
  - Example: Return an "error value" (not general, problematic)
    - Now all callers need to know specific error codes for each function call



Function Validates Itself

```
 4    // Returns a negative value for bad input.
 5    int area(int length, int width)
 6    {
 7        // Validate the inputs.
 8        if(length <= 0 || width <= 0)
 9        {
10            // Return an error value.
11            return -1;
12        }
13
14        return length * width;
15    }
```

RICHMOND

Function Validates Itself

The caller must be aware of these special return values.

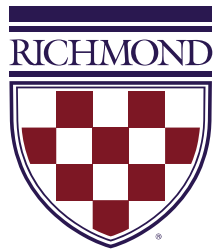
```
22      int result = area(length, width);
23
24      // Check the result for the -1 error return value.
25      if (result < 0)
26      {
27          cerr << "Bad area computation." << endl;
28          exit(1);
29      }
30
31      cout << "Area == " << result << endl;
```



RICHMOND

Bad Function Arguments

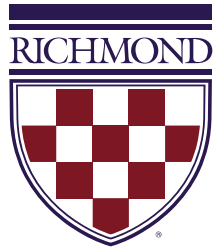
- The **function** should check!
  - Example: Function will throw an `exception` on invalid arguments
    - The caller has the **option** to `catch` the `exception`




Exception Handling
RICHMOND

Throwing Exceptions

- You could also choose from a selection of pre-defined exception classes in the `<stdexcept>` library
  - https://en.cppreference.com/w/cpp/error/exception
- Exceptions should be thrown that describe the error that occurs



Standard exceptions

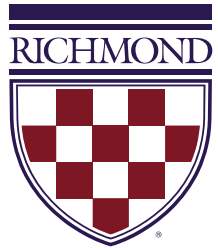
- `logic_error`
  - `invalid_argument`
  - `domain_error`
  - `length_error`
  - `out_of_range`
  - `future_error` (since C++11)
- `runtime_error`
  - `range_error`
  - `overflow_error`
  - `underflow_error`
  - `regex_error` (since C++11)
  - `system_error` (since C++11)
    - `ios_base::failure` (since C++11)
    - `filesystem::filesystem_error` (since C++17)
  - `tx_exception` (TM TS)
  - `nonexistent_local_time` (since C++20)
  - `ambiguous_local_time` (since C++20)
  - `format_error` (since C++20)

RICHMOND

Throwing Exceptions

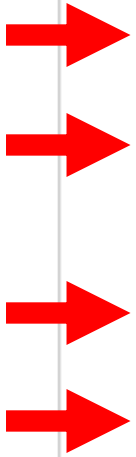


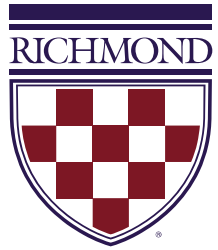
```
1   #include <iostream>
2   #include <stdexcept>
3   using namespace std;
4
5   // Will throw an exception on bad input.
6   int area(int length, int width)
7   {
8       // Validate the inputs.
9       if(length <= 0 || width <= 0)
10      {
11          // Throw an exception.
12          throw invalid_argument{"Bad argument to area()"};
13      }
14
15      return length * width;
16  }
```



Catching Exceptions

```
18  int main()
19  {
20      int length;
21      int width;
22      cout << "Enter values for length and width:" << endl;
23      cin >> length >> width;
24
25      try
26      {
27          int result = area(length, width);
28          cout << "Area == " << result << endl;
29      }
30      catch (invalid_argument exception)
31      {
32          cerr << exception.what() << endl;
33          exit(1);
34      }
35
36      return 0;
37  }
```

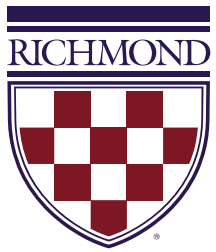




```
5   // Will throw an exception on bad input.
6   int area(int length, int width)
7   {
8       // Validate the inputs.
9       if(length <= 0 || width <= 0)
10      {
11          // Throw an invalid argument exception.
12          throw invalid_argument{"Bad argument to area()"};
13      }
14
15      int result = length * width;
16
17      // Check for an overflow in the result.
18      if (result / length != width)
19      {
20          // Throw an overflow error exception.
21          throw overflow_error{"Overflow occurred in area()"};
22      }
23
24      return result;
25  }
```

RICHMOND

```
29  int main()
30  {
31      int length, width;
32      cout << "Enter values for length and width:" << endl;
33      cin >> length >> width;
34
35      try
36      {
37          int result = area(length, width);
38          cout << "Area == " << result << endl;
39      }
40      catch (invalid_argument exception)
41      {
42          cerr << "Invalid Argument!" << endl;
43          exit(1);
44      }
45      catch (overflow_error exception)
46      {
47          cerr << "Overflow!" << endl;
48          exit(1);
49      }
50
51      return 0;
52  }
```



Exception Demo