

Errors and exceptions

Michael Nowak

Texas A&M University

September 14, 2017

Acknowledgement: Some lecture slides based on those created by Bjarne Stroustrup for use with his textbook

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Overview

Functions

Errors

- Sources of errors

- Your program

- Kinds of errors

Compile-time errors

- Syntax errors

- Type errors

Link-time errors

Run-time errors

- Detected by the computer

- Detected by a library

- Detected by user-code

 - Local

 - Non-local

Logic errors

Handling non-local errors at run-time

- How to report an error

Exceptions

References

Functions

- ▶ A `function` is a named block of code that can be passed arguments and returns a value to the caller

Functions

- ▶ A **function** is a named block of code that can be passed arguments and returns a value to the caller
- ▶ We can declare a function by writing a **declarator** of the form **f(args)**, where **f** is the name being introduced and **args** is the parameter list, for example:

Functions

- ▶ A **function** is a named block of code that can be passed arguments and returns a value to the caller
- ▶ We can declare a function by writing a **declarator** of the form **f(args)**, where **f** is the name being introduced and **args** is the parameter list, for example:
 - ▶ `double mult2(double d);`

Functions

- ▶ A **function** is a named block of code that can be passed arguments and returns a value to the caller
- ▶ We can declare a function by writing a **declarator** of the form **f(args)**, where **f** is the name being introduced and **args** is the parameter list, for example:
 - ▶ `double mult2(double d);`
 - ▶ Note: the **base type** specifies the **return type** of the **function**

Functions

- ▶ A **function** is a named block of code that can be passed arguments and returns a value to the caller
- ▶ We can declare a function by writing a **declarator** of the form **f(args)**, where **f** is the name being introduced and **args** is the parameter list, for example:
 - ▶ `double mult2(double d);`
 - ▶ Note: the **base type** specifies the **return type** of the **function**
- ▶ We can define a function by including the declaration with the definition provided in `{ }` directly following the parameter list (like a compound statement, we don't have a terminating semi-colon)

Functions

- ▶ A **function** is a named block of code that can be passed arguments and returns a value to the caller
- ▶ We can declare a function by writing a **declarator** of the form **f(args)**, where **f** is the name being introduced and **args** is the parameter list, for example:
 - ▶ `double mult2(double d);`
 - ▶ Note: the **base type** specifies the **return type** of the **function**
- ▶ We can define a function by including the declaration with the definition provided in `{ }` directly following the parameter list (like a compound statement, we don't have a terminating semi-colon)
 - ▶ `double mult2(double d) { return d*2; }`

Functions

- ▶ We will get into more details about `functions` later, but its helpful to understand them as they help motivate the necessity of `exceptions`

Overview

Functions

Errors

- Sources of errors

- Your program

- Kinds of errors

Compile-time errors

- Syntax errors

- Type errors

Link-time errors

Run-time errors

- Detected by the computer

- Detected by a library

- Detected by user-code

 - Local

 - Non-local

Logic errors

Handling non-local errors at run-time

- How to report an error

Exceptions

References

Errors

- ▶ When we write programs, errors are natural and unavoidable; the question is, how do we deal with them?
 - ▶ Organize software to minimize errors
 - ▶ Eliminate most of the errors we made anyway
 - ▶ Debugging
 - ▶ Testing

“My guess is that avoiding, finding, and correcting errors is 95% or more of the effort for serious software development.”

– Bjarne Stroustrup

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Sources of errors

- ▶ Poor specification
 - ▶ “What's this suppose to do?”
- ▶ Incomplete programs
 - ▶ “but I'll get around to it... tomorrow...”
- ▶ Unexpected arguments to functions
 - ▶ “but `sqrt()` isn't suppose to be called with `-1` as its argument”
- ▶ Unexpected input
 - ▶ “but the user was suppose to input an integer”
- ▶ Code that simply doesn't do what it was supposed to do
 - ▶ “so fix it...”

Overview

Functions

Errors

Sources of errors

Your program

Kinds of errors

Compile-time errors

Syntax errors

Type errors

Link-time errors

Run-time errors

Detected by the computer

Detected by a library

Detected by user-code

Local

Non-local

Logic errors

Handling non-local errors at run-time

How to report an error

Exceptions

References

Your program

- ▶ Should produce the desired results for all legal inputs

Your program

- ▶ Should produce the desired results for all legal inputs
- ▶ Should give reasonable error messages for all illegal inputs

Your program

- ▶ Should produce the desired results for all legal inputs
- ▶ Should give reasonable error messages for all illegal inputs
- ▶ Need not worry about misbehaving hardware

Your program

- ▶ Should produce the desired results for all legal inputs
- ▶ Should give reasonable error messages for all illegal inputs
- ▶ Need not worry about misbehaving hardware
- ▶ Need not worry about misbehaving system software

Your program

- ▶ Should produce the desired results for all legal inputs
- ▶ Should give reasonable error messages for all illegal inputs
- ▶ Need not worry about misbehaving hardware
- ▶ Need not worry about misbehaving system software
- ▶ Is allowed to terminate after finding an error

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Kinds of errors

Compile-time errors Errors found by the compiler

- ▶ Syntax errors
- ▶ Type errors

Kinds of errors

Compile-time errors Errors found by the compiler

- ▶ Syntax errors
- ▶ Type errors

Link-time errors Errors found by the linker when it is trying to combine object files into an executable program

Kinds of errors

Compile-time errors Errors found by the compiler

- ▶ Syntax errors
- ▶ Type errors

Link-time errors Errors found by the linker when it is trying to combine object files into an executable program

Run-time errors Errors found by checks made during a running program; that is, errors detected by

- ▶ the computer (hardware and/or the operating system)
- ▶ by a library (e.g., the standard library)
- ▶ by user code

Kinds of errors

Compile-time errors Errors found by the compiler

- ▶ Syntax errors
- ▶ Type errors

Link-time errors Errors found by the linker when it is trying to combine object files into an executable program

Run-time errors Errors found by checks made during a running program; that is, errors detected by

- ▶ the computer (hardware and/or the operating system)
- ▶ by a library (e.g., the standard library)
- ▶ by user code

Logic errors Errors found by the programmer looking for the causes of erroneous results

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors**

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors**

 - Syntax errors**

 - Type errors

- Link-time errors

- Run-time errors

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Compile-time errors : Syntax errors

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
```

```
int main ( ) {
    string first_name = "Michael";
    string last_name = "Nowak";
    string full_name = first_name + ' ' + last_name;
    cout << full_name << endl

    return 0;
}
```

Desktop/LX_Errors-Exceptions/code

% g6 CompileTimeErrors1.cpp

CompileTimeErrors1.cpp: In function 'int main()':

CompileTimeErrors1.cpp:12:5: **error:** expected ';' before 'r
eturn'

return 0;

A few more lines

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors**

 - Syntax errors

 - Type errors**

- Link-time errors

- Run-time errors

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Compile-time errors : Type errors

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
```

```
int main ( ) {
    string first_name = "Michael";
    string last_name = "Nowak";
```

```
    string sub_name = first_name - last_name;
    cout << sub_name;
```

```
    return 0;
```

```
}
```

Desktop/LX_Errors-Exceptions/code

% g6 CompileTimeErrors2.cpp

CompileTimeErrors2.cpp: In function 'int main()':
CompileTimeErrors2.cpp:11:34: **error:** no match for 'operator-' (operand types are 'std::__cxx11::string {aka std::__cxx11::basic_string<char>}' and 'std::__cxx11::string {aka std::__cxx11::basic_string<char>}')

string sub_name = first_name - last_name;

In file included from /usr/local/Cellar/gcc/6.2.0/include/c++/6.2.0/bits/stl_algobase.h:67:0,
from /usr/local/Cellar/gcc/6.2.0/include/c++/6.2.0/bits/char_traits.h:39,
from /usr/local/Cellar/gcc/6.2.0/include/c++/6.2.0/ios:40,
from /usr/local/Cellar/gcc/6.2.0/include/c++/6.2.0/ostream:38,
from /usr/local/Cellar/gcc/6.2.0/include/c++/6.2.0/istream:39

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors**

- Run-time errors

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Link-time errors

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
```

```
/*
```

```
declaration, for an undefined
```

```
*/
```

```
string make_full_name (string f, string l);
```

```
int main ( ) {
    string first_name = "Michael";
    string last_name = "Nowak";
    string full_name = make_full_name(first_name, last_name);

    return 0;
}
```

Desktop/LX_Errors-Exceptions/code

% g6 LinkTimeErrors1.cpp

Undefined symbols for architecture x86_64:

"make_full_name(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >>)", referenced from:

_main in ccvmwpd9.o

ld: symbol(s) not found for architecture x86_64

collect2: error: ld returned 1 exit status

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors**

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors**

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Run-time errors : detected by the computer

```
#include <iostream>
#include <vector>
using namespace std;

int main ( ) {

    int x = -1;
    int y = 0;
    /*
       divide by zero
    */
    int z = x / y;
    cout << z;

    return 0;
}
```

Desktop/LX_Errors-Exceptions/code

% g6 RunTimeErrors1.cpp

Desktop/LX_Errors-Exceptions/code

% ./a.out

[1] 46493 floating point exception ./a.out

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors**

 - Detected by the computer

 - Detected by a library**

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Run-time errors : detected by a library

```
#include <iostream>
#include <vector>
using namespace std;
```

```
int main ( ) {
```

```
    vector<int> v(10);
```

```
    /*
```

*when we are at v.size(), we are out of
v's range of elements*

```
    */
```

```
    for (int i = 0 ; i <= v.size() ; ++i)
```

```
        cout << v.at(i) << ' ';
```

```
    return 0;
```

```
}
```

Desktop/LX_Errors-Exceptions/code

% g6 RunTimeErrors2.cpp

Desktop/LX_Errors-Exceptions/code

% ./a.out

terminate called after throwing an instance of 'std::out_of_range'

what(): vector::_M_range_check: __n (which is 10) >= th
is->size() (which is 10)

0 0 0 0 0 0 0 0 0 [1] 50620 abort ./a.out

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors**

 - Detected by the computer

 - Detected by a library

 - Detected by user-code**

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Run-time errors : detected by user-code

- ▶ We can find errors through various checks made during a running program...

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors**

 - Detected by the computer

 - Detected by a library

 - Detected by user-code**

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Local run-time errors

- ▶ Easy to do for local run-time errors

- ▶

```
int i;  
std::cin >> i;  
if (i < 0)  
    return 1;
```

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors**

 - Detected by the computer

 - Detected by a library

 - Detected by user-code**

 - Local

 - Non-local**

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Non-local run-time errors

- How can we handle non-local errors during run-time?

```
// necessary #includes...
```

```
int area (int length, int width) { return length * width; }  
int framed_area (int x, int y) { return area(x-2, y-2); }
```

```
int main ( ) {  
    int x = -1;  
    int y = 2;  
    int z = 4;  
    // ...  
    int area1 = area(x, y);  
    int area2 = framed_area(1, z);  
    int area3 = framed_area(y, z);  
    double ratio = double(area1)/area3;  
    return 0;  
}
```

- Need some means of error reporting... will discuss this shortly

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors**

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

Logic errors

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

int main ( ) {

    vector<double> temps { -16.5, -23.2, -24.0, -25.7, -26.1, -18.6, -9.7, -2.4,
        7.5, 12.6, 23.8, 25.3, 28.0, 34.8, 36.7, 41.5, 40.3, 42.6, 39.7, 35.4,
        12.6, 6.5, -3.7, -14.3};

    double sum = 0;
    double high_temp = 0;
    double low_temp = 0;

    for (double t : temps) {
        if (t > high_temp) high_temp = t;
        if (t < low_temp) low_temp = t;
        sum += t;
    }

    double avg_temp = sum/temps.size();
    for (int i = 1 ; i <= temps.size() ; ++ i) {
        cout << temps.at(i-1) << '\t';
        if (i % 4 == 0) cout << endl;
    }
    cout << endl;
    cout << "High temperature: \t" << high_temp << endl;
    cout << "Low temperature: \t" << low_temp << endl;
    cout << "Average temperature: \t" << avg_temp << endl;

}
```

Desktop/LX_Errors-Exceptions/code
% g6 LogicErrors1.cpp

Desktop/LX_Errors-Exceptions/code
% ./a.out

| | | | |
|-------|-------|------|-------|
| -16.5 | -23.2 | -24 | -25.7 |
| -26.1 | -18.6 | -9.7 | -2.4 |
| 7.5 | 12.6 | 23.8 | 25.3 |
| 28 | 34.8 | 36.7 | 41.5 |
| 40.3 | 42.6 | 39.7 | 35.4 |
| 12.6 | 6.5 | -3.7 | -14.3 |

High temperature: 42.6
Low temperature: -26.1
Average temperature: 9.29583

Logic errors

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
```

```
int main ( ) {
```

```
    vector<double> temps {76.5, 73.5, 71.0, 73.6, 70.1, 73.5, 77.6, 85.3, 88.5,
        91.7, 95.9, 99.2, 98.2, 100.6, 106.3, 112.4, 110.2, 103.6, 94.9, 91.7,
        88.4, 85.2, 85.4, 87.7};
```

```
    double sum = 0;
    double high_temp = 0;
    double low_temp = 0;
```

```
    for (double t : temps) {
        if (t > high_temp) high_temp = t;
        if (t < low_temp) low_temp = t;
        sum += t;
    }
```

```
    double avg_temp = sum/temps.size();
    for (int i = 1 ; i <= temps.size() ; ++ i) {
        cout << temps.at(i-1) << '\t';
        if (i % 4 == 0) cout << endl;
    }
    cout << endl;
    cout << "High temperature: \t" << high_temp << endl;
    cout << "Low temperature: \t" << low_temp << endl;
    cout << "Average temperature: \t" << avg_temp << endl;
```

```
}
```

Desktop/LX_Errors-Exceptions/code
% g6 LogicErrors2.cpp

Desktop/LX_Errors-Exceptions/code
% ./a.out

| | | | |
|-------|-------|-------|-------|
| 76.5 | 73.5 | 71 | 73.6 |
| 70.1 | 73.5 | 77.6 | 85.3 |
| 88.5 | 91.7 | 95.9 | 99.2 |
| 98.2 | 100.6 | 106.3 | 112.4 |
| 110.2 | 103.6 | 94.9 | 91.7 |
| 88.4 | 85.2 | 85.4 | 87.7 |

High temperature: 112.4
Low temperature: 0
Average temperature: 89.2083

Logic errors

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
```

```
int main ( ) {
```

```
    vector<double> temps {76.5, 73.5, 71.0, 73.6, 70.1, 73.5, 77.6, 85.3, 88.5,
        91.7, 95.9, 99.2, 98.2, 100.6, 106.3, 112.4, 110.2, 103.6, 94.9, 91.7,
        88.4, 85.2, 85.4, 87.7};
```

```
    double sum = 0;
```

```
    double high_temp = temps[0];
```

```
    double low_temp = temps[0];
```

```
    for (double t : temps) {
        if (t > high_temp) high_temp = t;
        if (t < low_temp) low_temp = t;
        sum += t;
    }
```

```
    double avg_temp = sum/temps.size();
```

```
    for (int i = 1 ; i <= temps.size() ; ++ i) {
        cout << temps.at(i-1) << '\t';
        if (i % 4 == 0) cout << endl;
    }
```

```
    cout << endl;
```

```
    cout << "High temperature: \t" << high_temp << endl;
```

```
    cout << "Low temperature: \t" << low_temp << endl;
```

```
    cout << "Average temperature: \t" << avg_temp << endl;
```

```
}
```

Desktop/LX_Errors-Exceptions/code
% g6 LogicErrors2Cord.cpp

Desktop/LX_Errors-Exceptions/code
% ./a.out

| | | | |
|-------|-------|-------|-------|
| 76.5 | 73.5 | 71 | 73.6 |
| 70.1 | 73.5 | 77.6 | 85.3 |
| 88.5 | 91.7 | 95.9 | 99.2 |
| 98.2 | 100.6 | 106.3 | 112.4 |
| 110.2 | 103.6 | 94.9 | 91.7 |
| 88.4 | 85.2 | 85.4 | 87.7 |

High temperature: 112.4

Low temperature: 70.1

Average temperature: 89.2083

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time**

 - How to report an error**

- Exceptions

- References

Handling non-local errors at run-time

- The caller deals with the error

```
int area1 = area(x, y);  
if (area1 < 0)  
    /* handle error */  
else  
    /* no error, continue program execution */
```

Handling non-local errors at run-time

- The caller deals with the error

```
int area1 = area(x, y);  
if (area1 < 0)  
    /* handle error */  
else  
    /* no error, continue program execution */
```

- The callee deals with errors

```
int area (int length, int width) {  
    double a = length * width;  
    if (a < 0)  
        return 0;  
    else  
        return a;  
}
```

Handling non-local errors at run-time

- The caller deals with the error

```
int area1 = area(x, y);  
if (area1 < 0)  
    /* handle error */  
else  
    /* no error, continue program execution */
```

- The callee deals with errors

```
int area (int length, int width) {  
    double a = length * width;  
    if (a < 0)  
        return 0;  
    else  
        return a;  
}
```

- Error reporting

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time**

 - How to report an error**

- Exceptions

- References

How to report an error

- ▶ Return an “error value” (not general, problematic)

```
int area(int length, int width)
{
    if(length<=0 || width<=0) return -1;
    return length*width;
}
```

- ▶ So, “let the caller beware”

```
int z = area(x,y);
if (z<0) return error(“bad area”);
//...
```

- ▶ Problems:

- ▶ What if I forget to check the value returned?
- ▶ For some functions, there isn't a “bad value”

How to report an error

- ▶ Set an error status indicator (not general, problematic, don't)

```
int errno = 0;
int area(int length, int width)
{
    if(length<=0 || width<=0) errno = 7;
    return length*width;
}
```

- ▶ So, “let the caller check”

```
int z = area(x,y);
if (errno==7) return error(‘‘bad area’’);
//...
```

- ▶ Problems:

- ▶ What if I forget to check `errno`?
- ▶ How do I pick a value for `errno` that's different from all others?
- ▶ How do I deal with that error?

How to report an error

- ▶ The previous means of error reporting are not general...

How to report an error

- ▶ The previous means of error reporting are not general...
- ▶ Consider that, most of the time we can't change a function that handles errors in a way we don't like...
 - ▶ The author of the `std::vector` can detect run-time errors; however, he/she has no idea what the user would like to do about them

How to report an error

- ▶ The previous means of error reporting are not general...
- ▶ Consider that, most of the time we can't change a function that handles errors in a way we don't like...
 - ▶ The author of the `std::vector` can detect run-time errors; however, he/she has no idea what the user would like to do about them
 - ▶ The user of the `std::vector` knows how to cope with such errors; however, he/she cannot detect them (otherwise he/she would find them in his/her own code; not left for the library to find)

How to report an error

- ▶ The previous means of error reporting are not general...
- ▶ Consider that, most of the time we can't change a function that handles errors in a way we don't like...
 - ▶ The author of the `std::vector` can detect run-time errors; however, he/she has no idea what the user would like to do about them
 - ▶ The user of the `std::vector` knows how to cope with such errors; however, he/she cannot detect them (otherwise he/she would find them in his/her own code; not left for the library to find)
- ▶ So we need a means of reporting errors in a general way...

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions**

- References

Exceptions

- ▶ Exceptions are C++'s means of separating error reporting from error handling in a general way
 - ▶ Just about every kind of error can be reported using exceptions
 - ▶ Moreover, you can't forget about an exception: the program will terminate if someone doesn't handle it...
- ▶ You still have to figure out what to do about an exception (every exception thrown in your program)

Exceptions : Example 1

```
#include <iostream>
#include <stdexcept>
#include <limits>
using namespace std;

char to_char(int i) {
    return static_cast<char>(i);
}

int main () {
    cout << to_char(97) << endl;
    cout << to_char(155) << endl;
    return 0;
}
```

Desktop/LX_Errors-Exceptions/code
% g6 ExceptionEx1.cpp

Desktop/LX_Errors-Exceptions/code
% ./a.out
a
?

Exceptions : Throw, Try and Catch

```
char to_char(int i) {  
    if (i < numeric_limits<char>::min() || numeric_limits<char>::max() < i) {  
        const string s = to_string(i);  
        throw runtime_error("int_" + s + " is not within the range of char");  
    }  
    // we get here if and only if an exception is not thrown  
    return static_cast<char>(i);  
}
```

- ▶ When an unexpected condition happens, we can **throw** an exception
 - ▶ **to_char** will either return the corresponding *char* of the numeric value **i**
 - ▶ **or** it will throw a **runtime_error**

Exceptions : Example 1b

```
#include <iostream>
#include <string>
#include <stdexcept>
#include <limits>
using namespace std;
char to_char(int i) {
    if (i < numeric_limits<char>::min() || numeric_limits<char>::max() < i) {
        const string s = to_string(i);
        throw runtime_error("int_" + s + "_is_not_within_the_range_of_char");
    }
    // we get here if and only if an exception is not thrown
    return static_cast<char>(i);
}

int main () {
    cout << to_char(97) << endl;
    cout << to_char(128);
    return 0;
}
```

Desktop/LX_Errors-Exceptions/code

% g6 ExceptionEx1b.cpp

Desktop/LX_Errors-Exceptions/code

% ./a.out

a

terminate called after throwing an instance of 'std::runtime_error'

what(): int 128 is not within the range of char

[1] 58995 abort ./a.out

Exceptions : Throw, Try and Catch

- In order to handle the problem, we must indicate that we are willing to `catch` the exception of the type used to report the problem

Exceptions : Throw, Try and Catch

- ▶ In order to handle the problem, we must indicate that we are willing to `catch` the exception of the type used to report the problem
- ▶ If we do not catch the exception anywhere, the program will terminate (as seen in the previous example)

Exceptions : Throw, Try and Catch

- ▶ In order to handle the problem, we must indicate that we are willing to **catch** the exception of the type used to report the problem
- ▶ If we do not catch the exception anywhere, the program will terminate (as seen in the previous example)
- ▶ Therefore, we introduce a try-block around the code where an exception might occur

```
try {  
    cout << to_char(97) << endl;  
    cout << to_char(128);  
}
```

Exceptions : Throw, Try and Catch

- ▶ In order to handle the problem, we must indicate that we are willing to **catch** the exception of the type used to report the problem
- ▶ If we do not catch the exception anywhere, the program will terminate (as seen in the previous example)
- ▶ Therefore, we introduce a try-block around the code where an exception might occur

```
try {  
    cout << to_char(97) << endl;  
    cout << to_char(128);  
}
```

- ▶ The try-block is followed by the *exception handler*, which specifies the type of objects that it can catch

```
catch (const runtime_error& e) { // exception handler  
    cerr << "Exception:_" << e.what() << endl;  
}
```

Overview

- Functions

- Errors

 - Sources of errors

 - Your program

 - Kinds of errors

- Compile-time errors

 - Syntax errors

 - Type errors

- Link-time errors

- Run-time errors

 - Detected by the computer

 - Detected by a library

 - Detected by user-code

 - Local

 - Non-local

- Logic errors

- Handling non-local errors at run-time

 - How to report an error

- Exceptions

- References

References

- ▶ Lippman, B., Lajoie, Josee, & Moo, B. E. (2016). *C++ primer* (5th ed.). Addison-Wesley.
- ▶ Stroustrup, B. (2014). *Programming: principles and practice using C++* (2nd ed.). Addison-Wesley.