Exceptions

Assert

- We use `assert` to check for things that should “never happen”
- We are protecting ourselves (the programmer) from things that we assume will never happen (but just might)

More assert

- In the `assert` statement, we write a Boolean which should always be true!
- If it is not true, then we halt the program and report the problem
- Not user friendly, but potentially programmer friendly

Defensive Programming

- Include `#include <cassert>`
- Check for successful opening of stream. If assertion is false, halt.

```
in_file.open(“file.txt”);
assert(in_file.is_open());
```

Little Trick

- We can write any assert statement and-ed together with a string:

```
assert(in_file.is_open() && “failed file open”)
```

- The string always represents a true value (Boolean). If the first value becomes false, then the assert triggers and the message at halt contains your string.

Example 11.1

Why is this a mistake?

```
assert(in_file.is_open() || “failed file open”)
```

- It is better to handle problems in code instead of using assertions.
- Because the assert will never fail (it isn't testing anything)
- Because comments are better at explaining what the code does
- I don't know

Exceptions

- Keywords
  - `try`: a block where code is run and if an error occurs an exception is thrown, potentially to `catch` with other code
  - `throw`: raises an exception
  - `catch`: a block where an exception is caught and handled (in conjunction with try)

Non-local control

- Basic idea:
  - Keep watch on a particular section of code
  - If we get an exception raise / throw that exception (let it be known)
  - Look for a catcher that can handle that kind of exception
  - If catcher found, catcher handles the error.
  - Otherwise, end the program

#include<stdexcept>

- pg 197 of the book
  - exception: superclass of all exceptions
  - logic_error: violations of logical preconditions or class invariants
  - invalid_argument: illegal arguments
  - domain_error: domain errors
  - length_error: attempts to exceed maximum allowed size
  - out_of_range: arguments outside of expected range
  - runtime_error: indicate conditions only detectable at run time
  - range_error: range errors in internal computations
  - overflow_error: arithmetic overflows
  - underflow_error: arithmetic underflows

General form, version 1

```
try {
    code to run
} catch (type err_instance) {
    stuff to do on error
}
```

Which keyword is used to create an exception?

- `raise`
- `except`
- `throw`
- `I don't know`

Try block

- The `try` block contains code that we want to keep an eye on, to watch and see if any kind of errors occur
- If an error occurs anywhere in that `try` block, execution stops **immediately** in the block, the `try` looks for an appropriate `catch` to deal with the error
  - Appropriate is determined by the type that the `catch` registers it can handle
- If no special handler exists, runtime handles the problem (i.e. stops)

Exception Block

- A `catch` block (perhaps multiple `catch` blocks) is associated with a `try` block
- The `catch` block names the type of exception it is capable of handling
  - The type can be a subtype of a more general exception type
- If the error that occurs in the `try` block matches the `catch` type then that `catch` block is activated

try-exception combination

- If no exception in the `try` block, skip past all the `catch` blocks to the following code
- If an error occurs in a `try` block, look for the right `catch` by type
    - Including super-type of the exception
- If `catch` is matched, run that `catch` block and then skip past the `try/catch` blocks to the next line of code
- If no exception handling found, give the error to the runtime

Should every exception be caught?

- Yes
- No
- In C++, yes, in Python, no.
- I don't know

throw

- When you do a `throw`, you create an instance of an exception and you can provide, in the constructor, a string to describe the problem
    - Except for the superclass exception

Example 11.2

What counts as an exception

- Not every error is throws an exception in C++
  - Division by zero does not generate an exception
- Need to check to be sure
- Can look at the docs to determine what exceptions an operation can throw

Example 11.3

stod, stol

- C++17 provides a list of functions that try to convert a string to a number
  - `stod`, `stol`, etc (read “string to double” or “string to long”)
  - requires `#include<string>`

```
string s = “123.456”;
double d = stod(s);
```

String Stream

Two problems

- Conversion could run into two problems
  - Can’t do any part of the conversion
    - `stod(“abc”)` throws an error
  - Can convert part, some is ignored.
    - `size_t pos; string s = “123.abc”;`
    - `stod(s, &pos);`
    - converts what it can (`“123”`), pos is set to the position of the first unconverted char
    - If all is converted, `pos == s.size()`

When should you supply two arguments to `stol`?

- When you need to check that the entire string was converted.
- When you need to know how much of the string wasn't converted.
- When you need to know how many digits your number has.
- I don't know

Mix of a string and a stream

- A string stream is basically a mix of string and stream:
  - Holds a string as its contents
  - Allows the use of stream operators on that string
- Two types
  - Input
  - Output

#include<sstream>

- `istringstream` is a string stream that you can use `cin`-type operators on
- To create one, two ways

```
string line = “hello world”;
istringstream iss(line); // declare
iss.str(line); // using str method
```

Use cin ops

```
string word;
char ch;
istringstream iss(“hello world”);
iss >> word;  // space sep, “hello”
iss.get(ch);  // the space
iss.get(ch);  // ‘w’
```

Example 11.6

ostringstream

- This allows you to output using all the `cout` operators, then turn it into one string at the end
- Thus you can get rounding, widths, etc., just as you would with `cout`

Example

```
ostringstream oss;
oss << fixed << setprecision(4) << boolalpha;
oss << 3.14159 << “ is great == “ << true << endl;
cout << oss.str();
```

Output: 3.1416 is great == true

Example 11.7

So, why?

- istringstream:
  - cin is tricky. Get the whole line and use stream ops to parse the line via an istringstream.
- ostringstream:
  - Write, using all the type info and stream ops into a string, then you can further manipulate

How can you convert a string to a long?

- Using stol
- Using istringstream
- Looping over the chars in the string
- I don't know