exceptions, streams, containers/iterators in C++

ese2025

… a note on using “using [namespace]”

- rule-of-thumb:
  - DO NOT place a “using” statement inside of a header file
    - why? because if a user includes your header, you will be COMMITTING THEM to using that namespace (or abbreviation)
      - this could lead to CONFLICTS (for example, if the user is using “cout” from another library such as Qt, but you have place “using namespace std” in a header file, the compiler may not know which cout is meant)
  - DO use “using” in source files
    - this does not typically create problems because the use of “using” is limited to a particular source file

exception handling in C++

- basic try/catch form:

```
try
    {
        // code here might throw an exception
    }
catch(t)
    {
        // handles the exception (and only executes if
        // an exception occurred in the try portion)
    }
```

- *t* can be of various *exception classes*: logic_error, domain_error, invalid_argument, length_error, out_of_range, runtime_error, range_error, overflow_error, underflow_error

exception handling in C++ (cont'd)

- **to get these exception classes, one needs to use**: #include <stdexcept>
- EXAMPLE: Declaring:

```
domain_error t;
```

and then writing:

```
std::cout << t.what();
```

produces a message explaining what caused the error.

- simple exception handling:

```
throw e; // simply terminates the function, returns value e to caller
```

std::istream, std::ostream

- streams are objects used for reading or writing to: files, communication ports, or strings; these are accessible if you have #include <iostream>
- you have already seen some standard streams:
  - cin (console input)
  - cout (console output, screen)
  - cerr (console error, screen), which we haven't yet seen
- there are also stream manipulators, such as std::endl, and std::flush, std::hex, std::oct, std::dec, std::boolalpha
- you can declare functions to be of these classes as well for example:

streams (cont'd)

```
std::istream& read(std::istream& is, Student_info& s)
{
	// read and store the name, midterm and final grades
	is >> s.name >> s.midterm >> s.final;
	read_hw(is, s.homework);
	return is;
}
```

this function can be called as in:

```
vector<Student_info> students;
Student_info record;
while (read(std::cin, record))
{
	// find length of longest name
	maxlen = max(maxlen, record.name.size());
	students.push_back(record);
	cout << "Next student:" << endl;
}
```