What is this course about?

- Dynamic data structures
- Object-oriented programming
- Generic programming
- Abstract data types
- Algorithms: searching and sorting
- Standard Template Library

Concepts

- We want to create reliable, efficient, maintainable, and reuseable software.
- **Abstraction**: capture the important features of something while suppressing or ignoring the details.
- **Encapsulation**: define the "inside" and the "outside" of something, and ensure that the inside is protected from the outside. Also called **Information Hiding**.
- **Coupling**: the client is not dependent upon implementation, only upon the interface. Number of parameters indicate the level of coupling.
- **Localization**: the effects of changes are kept inside.
- Goal: abstraction with strong encapsulation, minimal coupling and maximal localization.

Libraries

- A library is a collection of related functions, types, and constants.
- Each function has a single purpose.
- Provides procedural abstraction.

Standard Library—Stream Classes

- e.g. iostream
- file streams provide access to files in addition to standard input and output (`<fstream>`)
- Allows us to read and write data to permanent storage.

Stream Classes

- Abstraction: an iostream object (e.g. cin, cout) are defined by operations that one may perform (e.g. read an integer, end-of-file test).
- Encapsulation: we do not know (or care) how files are opened, closed, written to, etc.
- Coupling: each function requires as few parameters as possible.
- Localization: when an operation is performed, the only thing outside of iostream that is affected are those passed into the function or those assigned from the return value.

Standard Library—String Class

- An abstraction of "a sequence of characters."
- Example operations: find length, get kth character, concatenate.
- We do not know how strings are represented.

User Defined Libraries

- Design the interface
- Interface/header file (.h)—declaration
- Implementation file (.cc)—definiton
- Each function should have a single purpose

Separate Compilation

- Compile each .cc file using -c option: gives object files (.o)
- Link all .o files together to get executable
- Makefiles: automate process, compile only those needed
- Type `make`: look for `Makefile` and tries to create the first target in the file
- Type `make myprog` to create the target named `myprog`

Makefile

```
myprog: util.o myprog.o

util.o: util.h util.cc

myprog.o: util.h myprog.cc
```