Abstract Data Types
[Sections 1.1-1.5, 3.1-3.2]

Abstraction in Programming

- The type **`int`** is an abstraction for a way of interpreting bits in memory as a number
- A **`struct`** is an abstraction of a collection of related data items
- A **`while`** loop is an abstraction for conditional repetition
- A function is a programmer-designed abstraction for some computation

Data Abstraction

- Data abstractions have two components
  - Set of possible values (state)
  - Operations that can be applied to values (behaviour)
- Examples
  - Integers:
    - 1, 2, -17148, etc.
    - arithmetic operations, printing, etc.
  - Arrays:
    - { 1, 2, 3 }, { true, true, false, true }, etc.
    - indexing operations, other operations?

Abstract Data Types (ADTs)

- A (user-defined) data abstraction that acts as a new type in the language
- Specification defines behaviour, including constructors
  - Range of possible values usually defined implicitly
- Implementation defines representation and gives implementations for operations
  - May contain "private" data and operations
  - Often, many implementations are possible

Types vs. Instances

- Types
  - General category
  - Usually few in number
  - Some built in (**`int`**, **`char`**, **`double`**, etc.)
  - Programmer-defined (arrays, **`struct`**s, **`enum`**s, ADTs, etc.)
- Instances
  - Particular values of a given type
  - In C++, variables can only hold instances, not types
  - May have many instances of a given type

Classes

- C++ has a **`class`** construct for building new ADTs
- Syntax is

```
class Student
{
        double gpa;
        // Other class member
        // declarations here
};
```

- Enhancements over **`struct`**
  - Can specify private vs. public members
  - Members can be functions, not just data

A Simple Class

```
class Student
{
public:
        void init( char name[] );
        void giveBonus( double amount );
        double getGPA();
private:
        char name[ 65 ];
        double gpa;
};
```

- Inside of a class definition, you can declare variables and functions
- Members can be public or private

Class Basics

- Private access is the default
- Private members are “hidden” from clients. The compiler will not allow client code to access them.
- Public members may be used directly by clients
- For the given class, tell me:
  - How many data members? private? public?
  - How many member functions?
  - What can the client use directly?

How Clients Use a Class

- A class is a new type!
  - Declare variables of that type:

```
Student craig;
```

  - Use as parameter

```
void processStudent( Student aStudent ) {…};
```

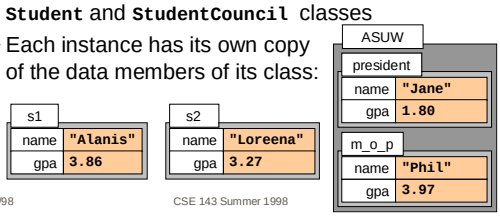
  - Use it to build other types (data decomposition):

```
class StudentCouncil
{
        Student president;
        Student minister_of_propaganda;
        …
};
```


A Class is a Type

```
Student s1, s2;
StudentCouncil ASUW;
```

- The code above creates fresh instances of the **`Student`** and **`StudentCouncil`** classes
- Each instance has its own copy of the data members of its class:

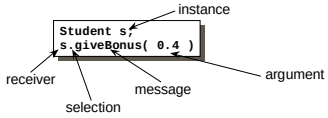


Operations on instances

- Most built-in operations DO NOT apply to class instances (**yet**)
- You cannot (for example)
  - use the "+" to add two **`Student`** instances
  - use the "==" to compare two instances for equality
- To the client, the only valid operations on instances are
  - assignment ("="), selection ("."), parameter passing, etc.
  - Operations defined in the public interface of the class

Terminology

- Think of a class as a cookie cutter, used to stamp out concrete objects (instances)
- Think of objects as simple creatures that we communicate with via “messages.”



Information Hiding

- The private access modifier supports and enforces information hiding (at compile time)

```
// A sample client program

        …
        Student test;

        test.gpa = 4.0;
        cout << test.gpa;

        test.init( "Jean Chretien" );
        test.giveBonus( 2.5 );
        cout << test.getGPA();
        …
```


Classes and Modules

- Typically put each ADT in its own module (file)
  - One .h file for interface
  - One .cpp file for implementation
- Can put multiple classes in one module
  - Why might one want to do this?
- In C++, must list private data and functions in the class definition, which goes in .h file
  - Why is this unfortunate?

Building the Class: Specification

```
// Student.h -- The Student ADT

#ifndef __STUDENT_H__
#define __STUDENT_H__
const int MAX_STUDENT_NAME = 65;

class Student
{
public:
        void init( char name[] );
        void giveBonus( double amount );
        double getGPA();
private:
        char name[ MAX_STUDENT_NAME ];
        double gpa;
};

#endif // __STUDENT_H__
```

Student.h

Building the Class: Implementation

```
#include "Student.h"

void Student::init( char str[] )
{
        strcpy( name, str );
        gpa = 4.0;
}

void Student::giveBonus( double amount )
{
        gpa += amount;
}

double Student::getGPA()
{
        return gpa;
}
```

scope resolution operator

class name here

Student.cpp

Class Scope: Implementation

- Implementations of member functions use **`Classname::`** prefix to indicate which class
- "::" is the scope resolution operator
- Within member function body:
  - ***Implicit reference to receiver***
  - Refer to members directly
  - Don't use class member names for formal parameters and local variables
- private members visible to public members, but not clients

Summary (I)

- A data abstraction is an abstraction of a set of values with similar properties, and the behaviour of those values
- An abstract data type is a data abstraction which acts like a type
- Types vs. instances

Summary (II)

- C++'s **`class`** construct is a mechanism for building ADTs
  - Data members (representation, state)
  - Function members (operations, behaviour)
- public vs. private members
- Member scope
- Assignment, parameter passing
- Classes vs. modules