CSCI 135
Abstract Data Types

Review – Section Data Structure

1. Components of Section definition: Many fields for major, instructorName, students, etc.
2. Operations to do things with/to a variable of type Section.

```
void setInstructorName(Section sec);
string getInstructorName(Section sec);
add1ToTestGrade(Section sec, int testid);
addStudToSec(Section sec, Student stud);
...
```

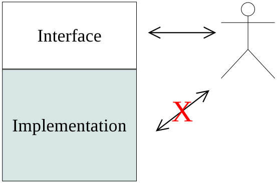
⚠ Nobody needs to know about Section's definition, since it relates purely to the *implementation* of the Section data structure, NOT its *interface* to its users.

Abstract Data Types (ADT)

A **data type** is a collection of values together with a set of operations defined on these values.
An **abstract data type** is a data type whose implementation is hidden from users.

- Concept developed by Barbara Liskov (2008 Turing Award winner) and Stephen N. Zilles in 1974. Fundamental basis of object oriented programming languages.

Key point: **implementation** vs. **interface**, with implementation hidden from the user to enable true modular programming.



You saw the ADT of integers along with the +/- operators in kindergarden!

Why ADTs?

- Encapsulation: Brings together data and operations allowed on that data. ADT writer specifies which operations are allowed on the data!
  Ex: `setInstructorName`, `addStudToSec`, . . . are the *only* operations allowed on sections.
- Information Hiding: User doesn't need to (and shouldn't) know details of ADT implementation (*e.g.*, names of Student and Section fields).
- Genericity: Type of data is like an argument to the ADT (so the same Section ADT (and `addStudToSec(Section sec, Student stud)` can be used as is even if the Student definition changes).
- Testing: The ADT can be tested in isolation. The interface provides a basis for generating suitable test cases.

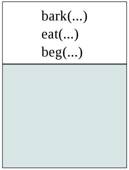
Why ADTs - 2

- Maintenance: The implementation of the ADT could (and often does) change, but the other million lines of code are unaffected. Ex: if Section is re-implemented using vectors instead of arrays of test grades, nothing else changes.
- Flexibility: Multiple ADT implementations, with varying performance characteristics (*e.g.*, more memory but faster), may be written. The choice between these may then be made depending on the application.
- Reasoning about program: We can argue about program correctness based on only the [much smaller] interface instead of having to understand details (assuming a separate argument is made about implementation correctness)

ADTs and Object Oriented Programming

ADTs can be implemented as a software engineering guideline in any language, but would require heavy enforcing – even a single non-ADT breaks the above advantages. OO languages explicitly support ADTs, though few languages support pure ADTs.

- An ADT is called a **class** in OO languages.
- Each instance of the class is called an **object**
- Each operation is called a **member function**
- Many OO languages (including C++) add various 'features' to ADTs that allow some user control of what is hidden



bark(...)
eat(...)
beg(...)

class Dog

```
// Dog is a class
// fido and spot are objects
Dog fido, spot;
fido.bark(...);
spot.beg(...);
```

We had fido bark and spot beg without knowing anything about Dog's implementation!

Class Components

- Name of class: convention to start with uppercase character
- Class interface: data and operations accessible by user
  C++: `public` keyword
- Private data and operations, that may be accessed only by the object itself (not by user)
  C++: `private` keyword
- Other things that we will talk about later (and in 235)

```
class DayOfYear {
public:                 interface follows
  int month;            data member of class
  int day;
  void output();        member functions
  void set(int newMonth, int newDay);
  void next();
};
```

? What are problems in above class?

Improving the DayOfYear Class

```
class DayOfYear {
public:                      interface follows
  void output();
  void set(int newMonth, int newDay);
  void advanceDay();
private:                     Not accessible by user
  int month;                 Data is almost always private
  int day;
  outputMonth();
};
```

Using the class:

```
int main() {
  DayOfYear today, tomorrow;
  today.set(11,30);
  tomorrow = today;
  tomorrow.advanceDay();
  tomorrow.output();
};
```

Writing Member Functions

- Same as writing regular functions
- Can call/access both public and private member functions/data
- Use scope resolution operator :: to specify class name (since the same function name can be used in multiple classes). Item before operator is called a **type qualifier**.

```
class DayOfYear {
  ...
};
void DayOfYear::output() {
  cout << month << ”/” << day << endl;
  return;
};
```

Note: No need for scope resolution operator when accessing members from member function.

Another Output() Implementation

```
class DayOfYear {
  ...
};
void DayOfYear::outputMonth() {
  switch(month) {
    case 1: cout << "January"; break;
    ...
    case 12: cout << "December"; break;
  };
  return;
};
void DayOfYear::output() {
  outputMonth();        calling a private member function
  cout << "/" << day << endl;
  return;
};
```

Summary of Class Operators

- Dot operator: specifies member of a particular object.
  Ex: `tomorrow.output()`
- Scope resolution operator: specifies which class the function definition comes from.
  Ex: `DayOfYear::output()`

Accessor/Mutator Functions

Most classes will have accessor and mutator functions (also called get and set).

An **accessor** function retrieves the value of data elements. Users of a class *must* access class data through accessor functions (not directly, since its private).

A **mutator** function sets the value of data elements. Users of a class *must* assign to data through mutator functions.

```
class Country {
  public:
    string getName();
    double getPopulation();
    void setName(string newName);
    void setPopulation(double newPop);
  private:
    string name;
    double pop2015;
};
```

Using Accessor/Mutator Functions

```
class Country {
  public:
    string getName();
    double getPopulation();
    void setName(string newName);
    void setPopulation(double newPop);
  private:
    string name;
    double pop2015;
};

int main() {
  Country usa;
  usa.setPopulation(320000000);  NOT usa.pop2015=...
};
```

Yes, its extra typing, but it makes the software maintainable in the long run!

What Exactly is a C++ Class? Object?

- A class is a full-fledged ADT that can be used in the same way as any other types (*e.g.*, int, char, double).
  ⇒ variable and parameters can be of that class type.
- Some languages define all types (including int, char, double, etc.) to be classes; C++ does not since this leads to inefficiency with OO overhead for doing simple operations such as 2+2.
- A variable of a class type is called an object.
- A class can also be considered as a generalization of a struct, where struct fields are restricted to data (not functions) and are implicitly public. Thus, C++ programs often don't have structs and use classes instead. However, structs avoid various OO overhead. Also, the two involve different types of thought (and thus software design).

Object Oriented Thinking

- First step in software design is data, not algorithms. Think about what major collections of data are, and the operations on them. These are your classes.
- Also think about interaction between objects. The interactions may guide your partitioning of the problem into objects, when choices exist.
- Algorithms aren't unimportant, but they come later when implementing member functions. You can always redesign an algorithm if its inefficient.

Example – Object Oriented Thinking

1. Which data types are involved?

2. What are the data and public operations for each of these?

3. Any private operations?

Example – Object Oriented Thinking

Example: Maintain the population of all countries in the world for 1940, 1950, . . ., 2010. Output the growth rate of countries as requested by user of data.

1. Which data types are involved?

2. What are the data and public operations for each of these?

3. Any private operations?

Example – Object Oriented Thinking

1. Which data types are involved?
   Country, World, maybe internal representation of data read from a file
2. What are the data and public operations for each of these?
   - get/set for each datum allowed to be read/written by user of object
   - compute a country's growth rate
   - I/O (after deciding whether it should be associated with the class or the user of the class)
3. Any private operations?
   Computing growth rates and storing them in the ADT?

Exercises

1. For any of the exercises given with structures (*e.g.*, Section, Transcript), design an ADT for that data structure.
   - Think about answers to questions on previous slides! *i.e.*, What do you want to allow the user to do to the data? What other internal [private] operations might be needed to support user functions?
2. Repeat the above for the structures lab in 136.
3. Repeat the above for the histogram example from the vectors unit