Designing Abstractions
Announcements

● Review Sessions Tomorrow (Friday), 11-11:50AM in Huang Auditorium
  • Will be recorded by SCPD
  • Plan: Answer questions and go through 1 or 2 problems on the 106X midterm

● Midterm on Monday, 7-10PM in Cubberly Auditorium
  • No office hours on Monday
  • No class on Monday
Announcements

- Today's material will **not** be on the midterm
- Assignment 3 due right now
- Assignment 4: Boggle!
  - Features recursive backtracking
  - Not due until a week after the midterm
Where are We...

- Course Goal: Develop a strong understanding of basic data structures
- Class so far:
  - Week 1: Basic C++
  - Week 2: Data structures
  - Week 3: Recursion
  - Week 4: Algorithmic Analysis
We are almost ready to start implementing and analyzing data structures!

A couple C++ language features we need to cover.
Classes

- **Vector, Stack, Queue, Map, etc.** are classes in C++.

- Classes contain
  - An **interface** specifying what operations can be performed on instances of the class.
  - An **implementation** specifying how those operations are to be performed.

- To define our own classes, we must define both the interface and the implementation.
Classes in C++

- Defining a class in C++ (typically) requires two steps:
  - Create a **header file** (typically suffixed with `.h`) describing the class's member functions and data members.
  - Create an **implementation file** (typically suffixed with `.cpp`) that contains the implementation of all the class's member functions.
- Clients of the class can then include the header file to use the class.
Classes

- Having a “good” interface is very important.
  - Poor design choices can have a negative impact on every programmer who interacts with the interface.
    - This includes you!
  - Modifying an interface after an implementation has been written can result in a lot of necessary code rewrites
- It's worth spending some time to think about what you want to put in your interface
Random Bags

- A **random bag** is a data structure similar to a stack or queue.

- Supports two operations:
  - **Add**, which adds an element to the random bag, and
  - **Remove random**, which removes and returns a random element from the bag.

- Has several applications:
  - Random maze generation
  - Shuffling decks of cards.
Random Bags
(RandomBag.cpp/h)
Random Bag: Private Variables

• Why did we make the `vector` private

• 2 good reasons to do this:

  1) By not exposing the Vector, we retain the freedom to change how we represent the RandomBag
     - (e.g. swap Vector for a Queue)

  2) We prevent the user from doing something we don't want to the Vector
     - We want to “protect” the data from the user.
     We'll see a good example of this later today.
Language Philosophy

- Every programming language exports some set of **primitives**:  
  - Primitive data types (**int**, **char**, etc.)  
  - Functions  
  - Classes  
  - etc.

- We can use those primitives to construct a larger set of primitives:  
  - **Vector**, **RandomBag**, etc.
Where Does it Stop?

- The collections we've been using are not primitives in C++; they are defined in terms of other language features.
- Understanding those features will let us analyze their efficiency.
- Understanding those features will let us build other interesting abstractions.
Getting Space

```cpp
int main() {
    Vector<int> values;

    int numValues = getInteger("How many?");
    for (int i = 0; i < numValues; i++) {
        values += i;
    }
}
```
Getting Storage Space

• How do the Vector, Stack, Queue, etc. get space to store all the elements that they hold?

• C++ code can request extra storage space as the program is running.

• This is called dynamic memory allocation.
  • Before I explain this, we need to talk about memory.
What is Memory?

• All variables and objects in C++ need somewhere to live inside the computer's memory.
  • This is RAM, by the way, not disk space.
• Whenever an object is created, space needs to be reserved for it.
Memory So Far

- So far, you have seen two types of variables:
  - **Local variables** declared inside a function.
    - Space is reserved for these variables when the function is called.
    - Space is reclaimed from these variables when the function call ends.
  - **Global variables / constants** declared outside a function.
    - Space is reserved for these variables when the program starts up.
    - Space is reclaimed from these variables when the program exits.
Draw Memory
(Board)
Good luck on the exam!