C++: Tour Leftovers

Plan

- const / static
- The Project

const

- Any variable declared as `const` cannot be changed
  - Like final in Java.
  - If the variable is a pointer then the object / data item pointed to cannot be changed.
    - `const aClass *a (new aClass(7))`
    - `a->moo = 7;  // error`

const

- Works for function arguments as well.

```
aClass i(7);
foo (&i);
```

```
void foo (const aClass *a)
{
    a->moo = 12; // error
}
```

const

- Furthermore

```
const aClass *i
(new aClass(7));

foo (i); // error
```

```
void foo (aClass *a)
{
   a->moo = 13;
}
```

const

- A class method can also be declared as `const`
  - This means that the method will not change the object.

```
class aClass {
  public printValues () const;
  public changeValues();
  }
```

const

- Const methods

```
void foo (const aClass *a)
{
    a->printValues(); // okay
    a->changeValues(); // error
}
```

static

- `static` class members like in Java:
  - the member has no knowledge of any particular instance of the class
  - data member: there is only one copy shared by all
  - member function: cannot access non-static data or functions

static

- In header file:

```
class Tribble {
public:
   Tribble();
  ~Tribble();
public:
   static void report();
   static unsigned long count;
};
```

static

- In source file

```
void Tribble::report()
{ ... }
unsigned long Tribble::count = 0;
```

static

- Local variables can also be declared as `static`.

```
void foo ()
{
        static int a = 0;
        …
        a = a + 1;
}
```

const / static

- Questions?

The Project

- Configuration Puzzles
- Goals:
  - Improve your design skills
  - C++ Programming
  - Use of unique C++ constructs

The Project

- Your mission…
  - Build a "generic" problem solver framework.
  - Apply the framework to 3 specific, yet different problems.
    - Fixing the time on your clock
    - Farmer's Dilemma problem
    - Parking Lot Problem

The Project

- So what exactly is a "generic" problem?
  - In each problem there is some kind of world
  - The world can be in one of many different configurations.
    - There is an initial configuration of the world
    - There is a set of testable goal configurations
  - Actions cause the world to change in an incremental way.
  - Solution: a set of actions that move the world from the initial configuration to one the goal configurations.

The Project

- Let's look at an example
  - Tic-Tac-Toe
    - World:  and Xs and Os
    - Initial configuration
    - Goal configuration: 3 Xs or 3Os in a row
    - Action: Alternately place Xs and Os in empty spots.

The Project

- Let's look at an example
  - Tic-Tac-Toe

X	X	X
	O	
O		

Solution:

Place X in UL

Place O in Center

Place X in UM

Place O in LL

Place X in UR

The Project

- "generic problem"
  - Questions?

The Project

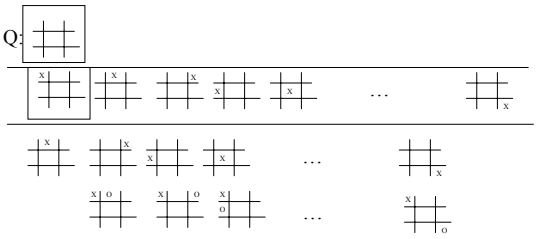
- Brute force algorithm for computing:
  - `Create an empty queue of configurations`
  - `Place initial configuration on the queue`
  - `While (queue not empty && first configuration on queue doesn't meet the goal) do`
    - `C = configuration removed from front of queue`
    - `For each action applicable to C do`
      - `Apply A to C and enqueue the result (if this configuration has not yet been seen)`
    - `End`

The Project

- Brute force algorithm for computing:
  - At the end of the processing either:
    - An acceptable configuration is at the head of the queue (Solution found)
    - The queue is empty (No solution found)

The Project

- Apply algorithm to TTT



The Project

- Things to note about the algorithm:
  - No concrete problem is mentioned
  - What needs to be done:
    - Need to store sequence of actions for a "solution"
    - Need to develop a way to determine if a configuration has already been seen.
  - Questions so far?

The Project

- Possible design approaches
  - Inheritance
    - abstract configuration
    - Subclass for each problem
  - Templates
  - Common Configuration Data Structure

The Project

- Concrete problems:
  - Fixing time on clock
  - Farmer's Dilemma
  - Parking lot jam
- Define world, configuration, actions, and goals for each.

The Project

- Deliverables:
  1. Design framework for "generic problem" using UML
  2. Implement abstract framework in C++ and test on set-the-clock problem
  3. Implement the Farmer's Dilemma Problem
  4. Implement the Parking Lot Problem
  5. Team Evaluations

The Project

- Grading:
  1. Design 20 pts
  2. Clock 20 pts
  3. Farmer 20 pts
  4. Parking 40 pts
  5. Team bonus points or deductions

The Project

- Deadlines:
  1. Design October 1
  2. Clock October 16
  3. Farmer October 30
  4. Parking November 11
  5. Team November 12

The Project

- Logistics
  - Teams of 2
    - Will distribute team accounts for submission
  - We expect you will
    - Follow style guidelines (C++ and UML)
    - Use RCS for code management.

The Project

- More details to come
- Questions?