COMSC-165 Lecture Topic 16
Applied Recursion

Reference

Tutorial 1
Tutorial 2

Recursive Loops

recursion is another syntax for loops
  using functions that call themselves
next cycle: a call to self
if-break: skip call to self
  "collapsing" the stacked calls
advantage: each cycle has its own memory
  persisting while other cycles run
a cycle does not need to complete before...
  ...downstream cycles can begin
recall: Fibonacci series (0 1 1 2 3 5 8 13 21 34...)

Binary File I/O

use `fstream` objects *only*
  for compatibility with all compilers
record I/O (i.e., structures)

```
fstream fin; fin.open("data.dat", ios::binary|ios::in);
.read((char*)&noon, sizeof(tod));
fstream fout; fout.open("data.dat", ios::binary|ios::out);
.write((char*)&noon, sizeof(tod));
```

to get file size

```
fin.seekg(0, ios::end);
long size = fin.tellg();
```

  to rewind: **`seekg(0, ios::beg)`**
file headers
random access with **`seekg()`**

Binary Tree Persistence

a useful application of recursion

```
// utility function prototypes
void saveTree(animal*, fstream&);
void restoreTree(animal*, fstream&);

// restore tree from a disk file
fstream fin;
fin.open("animal.dat", ios::in|ios::binary);
restoreTree(root, fin); // create an empty root node before calling this
fin.close();

// save tree to a disk file
fstream fout;
```

```
  fout.open("animal.dat", ios::out|ios::binary);
  saveTree(root, fout);
  fout.close();

  // function definitions -- save and restore
  void saveTree(animal* a, fstream& out) // saves tree to disk file
  {
    if (a)
    {
      out.write((char*)a, sizeof(animal));
      saveTree(a->yes, out);
      saveTree(a->no, out);
    }
  }

  void restoreTree(animal* a, fstream& in) // loads tree from disk file
  {
    in.read((char*)a, sizeof(animal));
    if(a->yes)
    {
      restoreTree(a->yes = new animal, in);
      restoreTree(a->no = new animal, in);
    }
  }
```
