Follow the River and You Will Find the C

A systems programming course with a narrative

Jae Woo Lee, Michael Kester and Henning Schulzrinne
Columbia University

SIGCSE 2011

Objects-first

- Objects-first v. Iterative-first v. Functional-first
  - Current trend is object-first with Java or Python
- Everyone has an opinion – I have one too!

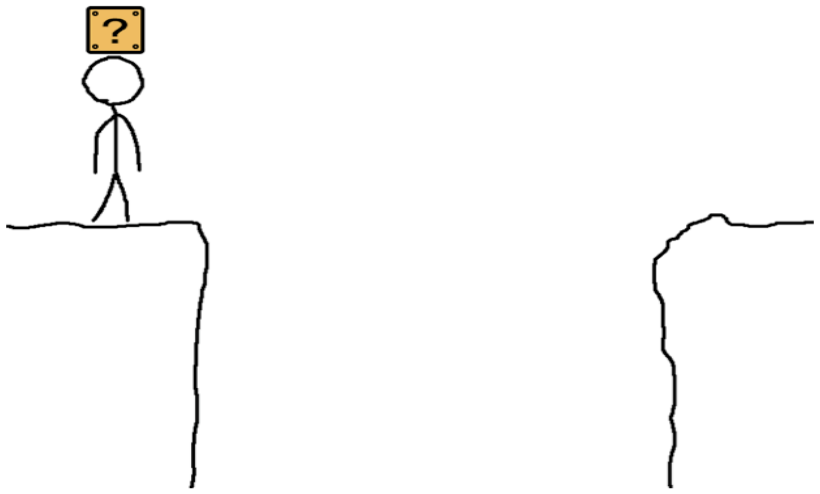
But not today.

Our course addresses a consequence of choosing objects-first.

The Gap problem

CS1, CS1.5, CS2

- Java
- Toy programs
- Eclipse
- NotePad
- …



OS

- C
- Linux kernel
- make, svn, gdb
- vi, emacs
- …

Typical hodgepodge transition courses offer either:

1. Too little – students are underprepared
2. Too much – students run away

Designing an effective transition

- One-semester course that covers:
  - The whole C
  - Some essential C++
  - A lot of UNIX and networking
- With four goals:
  1. *Don't forgo depth*
  2. *Focus on doing it right*
  3. *Lay out the big picture*
  4. *Don't be boring*

How?

- The big project: web server from scratch
  - Seemingly independent *labs* as milestones
  - Each contributes code or concept
- Rigid structure
  - Each lab builds on previous ones
    - Provide solution after each deadline
  - Super-detailed instructions
    - Not much room for creativity
- Motivating students
  - You will write a real web server from scratch!
  - You will go from a programming student to a *programmer*

The course, a drama

Exposition
- Lab1: Tools
- Lab2: Pointers
- Lab3: Linked List

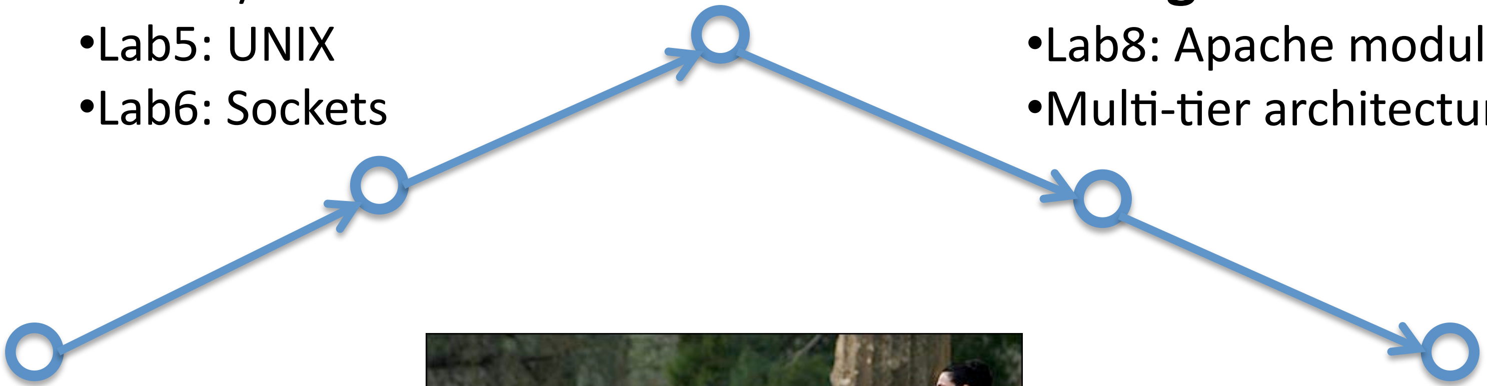
Rising action
- Lab4: I/O
- Lab5: UNIX
- Lab6: Sockets

Climax
- Lab7: Web server

Falling action
- Lab8: Apache module
- Multi-tier architecture

Resolution

C++ essentials
- Lab9: Object lifetime
- Lab10: Linked List II
- SmartPtr



Lab1: Shell basics, SVN, Make

- Learn essential UNIX command line tools
- Learn how to compile and link multiple source files
- Learn how to use SVN and Make

Lab2: Pointers and Arrays

- The most important and difficult milestone!
  - Students need plenty of time and help
- Give hard problem:

```
$ ./twecho one two three
one ONE
two TWO
three THREE
```

- Require bug-free code
  - Use Valgrind
  - *Focus on doing it right*

```
int main(int argc, char **argv)
{
    if (argc <= 1)
        return 1;

    char **copy =
        duplicateArgs(argc, argv);

    char **p = copy;

    argv++;
    p++;
    while (*argv) {
        printf("%s %s\n", *argv++, *p++);
    }

    freeDuplicatedArgs(copy);

    return 0;
}
```

Lab3: Linked List

- Rigid structure – header file given

```
struct Node {
    struct Node *next;
    void *data;
};
struct List {
    struct Node *head;
};
struct Node *addFront(struct List *lst, void *data);
struct Node *findNode(struct List *lst, const void *dataSought,
    int (*compar)(const void *, const void *));
```

- Comprehensive test driver also given
  - Again, bug-free code using Valgrind
- Pointer semantics and type unsafe
  - Will be revisited in Lab10

Lab4: Standard I/O

- Mdb: flat-file database of name and messages

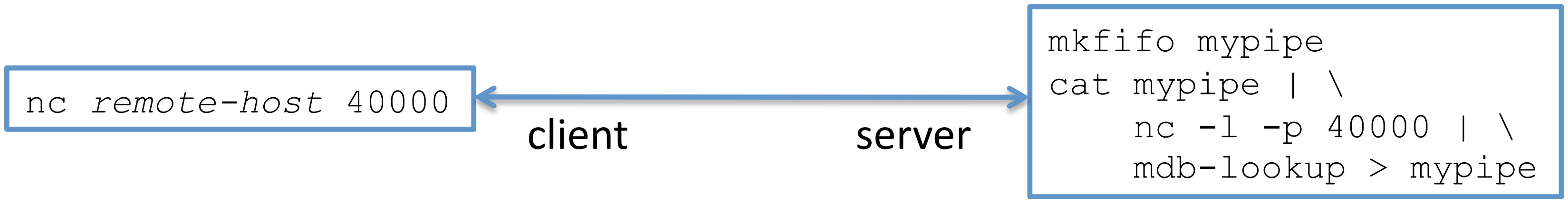
```
struct MdbRec {
   char name[16];
   char  msg[24];
};
```

- Implement MdbLookup
  - Reads shared database file into linked list on start-up
  - Use lab3's linked list as a library
  - Prompts for search string and prints matching records
  - MdbAdd binary is provided for testing

Lab5: Turning MdbLookup into a server without socket programming

- End of C; lecture shifts to UNIX and networking
  - Brief overview of OS and TCP/IP – impart the concept of *layers*
  - Process management in UNIX – fork and exec

- Turn MdbLookup into a server using Netcat



```
nc remote-host 40000
```

```
mkfifo mypipe
cat mypipe | \
    nc -l -p 40000 | \
    mdb-lookup > mypipe
```

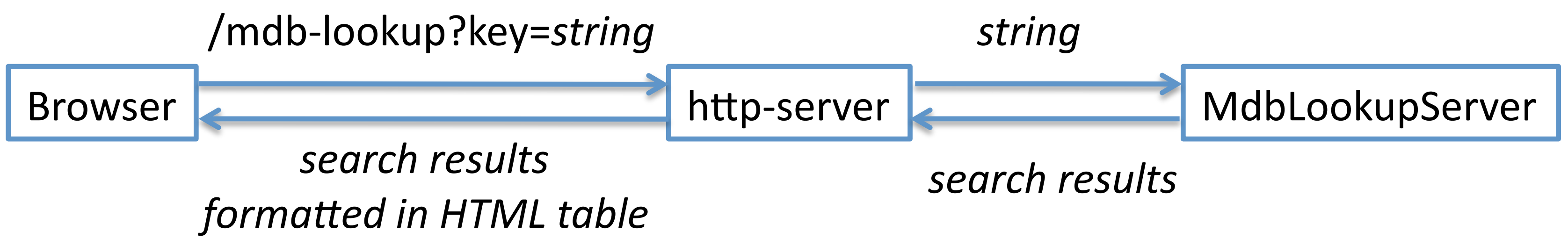
  - The server-side pipeline is given; students put it in a shell script and write a C program to fork and exec the script

Lab6: Sockets and HTTP

- Go through sample TCP client and server code
  - TCPEchoClient.c / TCPEchoServer.c
- Lab6, part 1: MdbLookupServer
  - TCPEchoServer.c + MdbLookup.c (from lab4)
  - Fewer than 20 lines of modification
- Explain HTTP protocol
  - Show the protocol in action using Netcat
    - Netcat client posing as a browser
    - Netcat server posing as a web server
- Lab6, part 2: implement wget *lite*
  - Downloads a single file using HTTP

Lab7: Web server from scratch!

- At this point, students have all they need to implement a subset of HTTP 1.0:
  - Only GET requests
  - Does not send content-type header
- Part 1: serve static HTML page with images
- Part 2: serve dynamic page generated by MdbLookup



“OMG, this thing shows up in my FireFox!”

Lab8: Apache module

- Rewrite lab7 as an Apache module
  - Download, build and configure Apache web server
  - Write a C module to connect to MdbLookupServer
- One of the easiest labs!

Software Architecture: The Big Picture

- Retrace the evolution of MdbLookup
  - Lab4: command line, access local database
  - Lab5: server, put together with Netcat and pipes
  - Lab6: server, coded using the sockets API
  - Lab7: web-based server, written from scratch
  - Lab8: web-based server, written as Apache module
- Now students understand multi-tier client-server architecture
  - Underlying architecture for LAMP, J2EE, etc.

3 weeks left – let’s learn C++

- Focus on object lifetime and memory usage
  - Natural extension to our focus so far
  - Often poorly understood by many who use C++
- Coverage
  - Object construction and destruction
  - Templates and STL containers

Lab9: Object Construction and Destruction in C++

- Detailed study of MyString class implementation
- Trace the *Basic4*
  - Insert printf in constructor, destructor, copy and op=()
  - Analyze the output generated by add() function
  - Need to compile with

    `”-fno-elide-constructors”`

```
class MyString
{
public:
    // member functions ...
    // overloaded ops ...
private:
    char *data;
    int len;
};
```

```
MyString add(MyString s1,
             MyString s2)
{
  MyString temp(" and ");
  return s1 + temp + s2;
}
```

Lab10: Working with legacy code – Linked List Revisited

- Part 1: New face to the legacy code
  - Implement StrList, linked list of MyString, using lab3 linked list as underlying engine

```
void StrList::addFront(const MyString& str) calls:
    struct Node *addFront(struct List *list, void *data)
```

  - This is hard!
    - Need to switch from pointer semantics to value semantics
    - Comprehensive test driver provided
- Part 2: Now upgrade the engine
  - Turn StrList into a template class TList
    - For the engine, switch from lab3 linked list to STL list
  - Part 1 test drive works without modification with typedefs

```
typedef string MyString;
typedef TList<string> StrList;
```

Come full circle – Java-style object reference in C++

- "I miss Java…"
  1. Nice Java code
     ```
     Foo b = a.createFoo();  b.doSomething(); return;
     ```
  2. Same exact code in C++ (or is it?)
     ```
     Foo b = a.createFoo();  b.doSomething(); return;
     ```
  3. We can do this, but…
     ```
     Foo *b = a.createFoo();  b->doSomething(); return;
     ```
  4. Now this come pretty darn close
     ```
     SmartPtr<Foo> b = a.createFoo(); b->doSomething(); return;
     ```
- SmartPtr
  – Reference-counted, so can be freely copied
  – Initialized with pointer to heap-allocated object
  – Overloads operator->() and operator*()

Conclusion

- Students loved the course
  - Great evaluations and reviews
- They liked:
  - Single track nature of the course
  - Rigid structure
    - Detailed lab instructions
    - Immediate verification of correctness
  - Class mailing list
- Will share course materials with other instructors