Biostatistics 615/815 - Lecture 2
Introduction to C++ Programming

Hyun Min Kang

September 6th, 2012

Last Lecture

- Algorithms are sequences of computational steps transforming inputs into outputs
- Insertion Sort
  - ✓ An intuitive sorting algorithm
  - ✓ Loop invariant property
  - ✓ $\Theta(n^2)$ time complexity
  - ✓ Slower than default `sort` application in Linux.
- A recursive algorithm for the Tower of Hanoi problem
  - ✓ Recursion makes the algorithm simple
  - ✓ Exponential time complexity
- C++ Implementation of the above algorithms.

Fill missing steps below to complete homework 0

1. `ssh uniqname@scs.itd.umich.edu`
2. `mkdir --p ~/Private/biostat615/hw0/`
3. `cd ~/Private/biostat615/hw0/`
4. `vi helloWorld.cpp` (input the code)
5. (   )
6. `vi towerOfHanoi.cpp` (input the code)
7. (   )
8. `rm *.o helloWorld towerOfHanoi`
9. `cd ../`
10. `tar czvf uniqname.hw0.tar.gz hw0/`
11. `scp uniqname@scs.itd.umich.edu:Private/biostat615/uniqname.hw0.tar.gz .` (After logout)

Algorithm InsertionSort

Data: An unsorted list $A[1 \cdots n]$
Result: The list $A[1 \cdots n]$ is sorted

```
for j = 2 to n do
    key = A[j];
    i = (      );
    while i > 0 and A[i] > key do
        (      ) = (      );
        i = i - 1;
    end
    (      ) = key;
end
```

Today

- Hanoi Tower Example
- Basic Data Types
- Control Structures
- Pointers and Functions
- Fisher's Exact Test

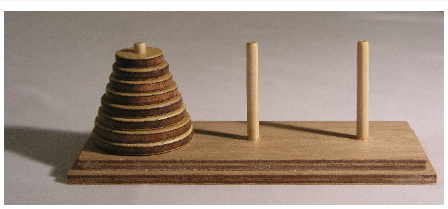
Next few lectures

- The class does NOT focus on teaching programming language itself
- Expect to spend time to be familiar to programming languages yourself
  - ✓ Online reference : *http://www.cplusplus.com/doc/tutorial/*
  - ✓ Offline reference : C++ Primer Plus, 6th Edition
- VERY important to practice writing code on your own.
- Utilize office hours or after-class minutes for detailed questions in practice

Tower of Hanoi

Problem

- Input
  - A (leftmost) tower with n disks, ordered by size, smallest to largest
  - Two empty towers
- Output: Move all the disks to the rightmost tower in the original order
- Condition
  - One disk can be moved at a time.
  - A disk cannot be moved on top of a smaller disk.



How many moves are needed?

A Working Example

http://www.youtube.com/watch?v=aGlt2G-DC8c

Think Recursively

Key Idea

- Suppose that we know how to move $n-1$ disks from one tower to another tower.
- And concentrate on how to move the largest disk.

How to move the largest disk?

- Move the other $n-1$ disks from the leftmost to the middle tower
- Move the largest disk to the rightmost tower
- Move the other $n-1$ disks from the middle to the rightmost tower

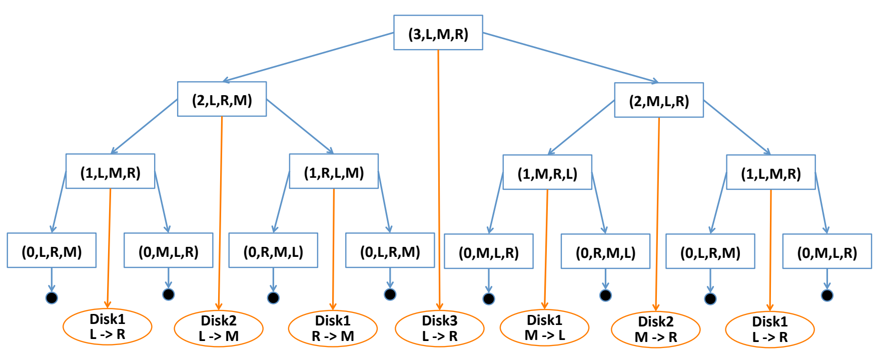
A Recursive Algorithm for the Tower of Hanoi Problem

Algorithm TowerOfHanoi

Data: n : # disks, (s, i, d) : source, intermediate, destination towers
Result: n disks are moved from s to d

if $n == 0$ **then**
  do nothing;
else
  TowerOfHanoi($n-1, s, d, i$);
  move disk n from s to d;
  TowerOfHanoi($n-1, i, s, d$);
end

How the Recursion Works



Analysis of TowerOfHanoi Algorithm

Correctness

- Proof by induction - Skipping

Time Complexity

- $T(n)$: Number of disk movements required
  - ✓ $T(0) = 0$
  - ✓ $T(n) = 2T(n-1) + 1$
- $T(n) = 2^n - 1$
- If $n = 64$ as in the legend, it would require $2^{64} - 1 = 18,446,744,073,709,551,615$ turns to finish, which is equivalent to roughly 585 billion years if one move takes one second.

Implementing TowerOfHanoi Algorithm in C++

towerOfHanoi.cpp

```
#include <iostream>
#include <cstdlib>
// recursive function of towerOfHanoi algorithm
void towerOfHanoi(int n, int s, int i, int d) {
  if ( n > 0 ) {
    towerOfHanoi(n-1,s,d,i); // recursively move n-1 disks from s to i
    // Move n-th disk from s to d
    std::cout << "Disk " << n << " : " << s << " -> " << d << std::endl;
    towerOfHanoi(n-1,i,s,d); // recursively move n-1 disks from i to d
  }
}
// main function
int main(int argc, char** argv) {
  int nDisks = atoi(argv[1]); // convert input argument to integer
  towerOfHanoi(nDisks, 1, 2, 3); // run TowerOfHanoi(n=nDisks, s=1, i=2, d=3)
  return 0;
}
```


Running TowerOfHanoi Implementation

Running towerOfHanoi

```
user@host:~/Private/biostat615/hw0$ ./towerOfHanoi 3
Disk 1 : 1 -> 3
Disk 2 : 1 -> 2
Disk 1 : 3 -> 2
Disk 3 : 1 -> 3
Disk 1 : 2 -> 1
Disk 2 : 2 -> 3
Disk 1 : 1 -> 3
```


Summary : Tower of Hanoi Problem

- *Recursion* : Simple definition using induction
  - Move the other $n-1$ disks from the leftmost to the middle tower
  - Move the largest disk to the rightmost tower
  - Move the other $n-1$ disks from the middle to the rightmost tower
- Digesting the concept can sometimes be tricky
- Exponential time complexity : $\Theta(2^n)$

Declaring Variables

Variable Declaration and Assignment

```
int foo; // declare a variable
foo = 5; // assign a value to a variable.
int foo = 5; // declaration + assignment
```

Variable Names

```
int poodle; // valid
int Poodle; // valid and distinct from poodle
int my_stars3; // valid to include underscores and digits
int 4ever;  // invalid because it starts with a digit
int double; // invalid because double is C++ keyword
int honky-tonk; // invalid -- no hyphens allowed
```

Basic Digital Units

bit A single binary digit number which can represent either 0 or 1

byte A collection of 8 bits which can represent $256 (= 2^8)$ unique numbers. One character can typically be stored within one byte.

word An ambiguous term for the natural unit of data in each processor. Typically, a word corresponds to the number of bits to represent a memory address. In 32-bit address scheme which can represent up to 4 gigabytes, 32 bits (4 bytes) are spent to represent a memory address. In 64-bit address scheme, up to 18 exabytes can be represented by using 64 bits (8 bytes) to represent a memory address.

Data Types

Signed Integer Types

```
char foo;  // 8 bits (1 byte) : -128 <= foo <= 127
short foo; // 16 bits (2 bytes) : -32,768 <= foo <= 32,767
int foo;   // Mostly 32 bits (4 bytes) : -2,147,483,648 <= foo <= 2,147,483,647
long foo;  // 32 bits (4 bytes) : -2,147,483,648 <= foo <= 2,147,483,647
long long foo; // 64 bits
short foo = 0;  foo = foo - 1;   // foo is -1
```

Unsigned Integer Types

```
unsigned char foo; 8 bits (1 byte) : 0 <= foo <= 255
unsigned short foo; // 16 bits (2 bytes) : 0 <= foo <= 65,535
unsigned int foo;   // Mostly 32 bits (4 bytes) : 0 <= foo <= 4,294,967,295
unsigned long foo;  // 32 bits (4 bytes) : 0 <= foo <= 4,294,967,295
unsigned long long foo; // 64 bits
unsigned short foo = 0;  foo = foo - 1;   // foo is 65,535
```


Floating Point Numbers

Comparisons

Type	`float`	`double`	`long double`
Precision	Single	Double	Quadruple
Size	32 bits	64 bits	128 bits
(in most modern OS)	4 bytes	8 bytes	16 bytes
Sign	1 bit	1 bit	1 bit
Exponent	8 bits	11 bits	15 bits
Fraction	23 bits	52 bits	112 bits
(# decimal digits)	7.2	16	34
Minimum (>0)	1.2×10^{-38}	2.2×10^{-308}	3.4×10^{-4932}
Maximum	3.4×10^{38}	1.8×10^{308}	1.2×10^{4932}

Handling Floating Point Precision Carefully

precisionExample.cpp

```
#include <iostream>
int main(int argc, char** argv) {
  float smallFloat = 1e-8; // a small value
  float largeFloat = 1.;   // difference in 8 (>7.2) decimal figures.
  std::cout << smallFloat << std::endl; // "1e-08" is printed
  smallFloat = smallFloat + largeFloat; // smallFloat becomes exactly 1
  smallFloat = smallFloat - largeFloat; // smallFloat becomes exactly 0
  std::cout << smallFloat << std::endl; // "0" is printed
  // similar thing happens for doubles (e.g. 1e-20 vs 1).
  return 0;
}
```

Basics of Arrays and Strings

Array

```
int A[] = {3,6,8}; // A[] can be replaced with A[3]
std::cout << "A[0] = " << A[0] << std::endl; // prints 3
std::cout << "A[1] = " << A[1] << std::endl; // prints 6
std::cout << "A[2] = " << A[2] << std::endl; // prints 8
```

String as an array of characters

```
char s[] = "Hello, world"; // or equivalently, char* s = "Hello, world"
std::cout << "s[0] = " << s[0] << std::endl; // prints 'H'
std::cout << "s[5] = " << s[5] << std::endl; // prints ','
std::cout << "s = " << s << std::endl; // prints "Hello, world"
```


Summary - Data Types and Precisions

- Each data type consumes different amount of memory
  - For example, 1GB can store a billion characters, and 125 million double precision floating point numbers
  - To store a human genome as character types, 3GB will be consumed, but 12GB will be needed if each nucleotide is represented as an integer type
- Precision is not unlimited.
  - Unexpected results may happen if the operations require too many significant digits.

Assignment and Arithmetic Operators

```
int a = 3, b = 2;  // valid
int c = a + b;     // addition : c == 5
int d = a - b;     // subtraction : d == 1
int e = a * b;     // multiplication : e == 6
int f = a / b;     // division (int) generates quotient : f == 1
int g = a + b * c; // precedence - add after multiply : g == 3 + 2 * 5 == 13
a = a + 2;         // a == 5
a += 2;            // a == 7
++a;               // a == 8
a = b = c = e;     // a == b == c == e == 6
```


Comparison Operators and Conditional Statements

```
int a = 2, b = 2, c = 3;
std::cout << (a == b) << std::endl; // prints 1 (true)
std::cout << (a == c) << std::endl; // prints 0 (false)
std::cout << (a != c) << std::endl; // prints 1 (true)
if ( a == b ) {  // conditional statement
  std::cout << "a and b are same" << std::endl;
}
else {
  std::cout << "a and b are different" << std::endl;
}
std::cout << "a and b are " << (a == b ? "same" : "different") << std::endl
  << "a is " << (a < b ? "less" : "not less") << " than b" << std::endl
  << "a is " << (a <= b ? "equal or less" : "greater") << " than b" << std::endl;
```

Loops

while loop

```
int i=0;  // initialize the key value
while( i < 10 ) { // evaluate the loop condition
  std::cout << "i = " << i << std::endl; // prints i=0 ... i=9
  ++i; // update the key value
}
```

for loop

```
for(int i=0; i < 10; ++i) { // initialize, evaluate, update
  std::cout << "i = " << i << std::endl; // prints i=0 ... i=9
}
```


Pointers

s = 0x3fb5

Address	0x3fb5	0x3fb6	0x3fb7	0x3fb8	0x3fb9	0x3fba
Value	'H'	'E'	'L'	'L'	'O'	'\0'

Another while loop

```
char* s = "HELLO"; // array of {'H','E','L','L','O','\0'}
while ( *s != '\0' ) {  // *s access the character value pointed by s
  std::cout << *s << std::endl; // prints 'H','E','L','L','O' at each line
  ++s; // advancing the pointer by one; points to the next element
}
```


Pointers and Loops

while loop

```
char* s = "HELLO"; // array of {'H','E','L','L','O','\0'}
while ( *s != '\0' ) {
  std::cout << *s << std::endl; // prints 'H','E','L','L','O' at each line
  ++s; // advancing the pointer by one
}
```

for loop

```
// initialize array within for loop
for(char* s = "HELLO"; *s != '\0'; ++s) {
  std::cout << *s << std::endl; // prints 'H','E','L','L','O' at each line
}
```


Pointers are complicated, but important

```
int A[] = {3,6,8}; // A is a pointer to a constant address
int* p = A;        // p and A are containing the same address
std::cout << p[0] << std::endl;   // prints 3 because p[0] == A[0] == 3
std::cout << *p << std::endl;   // prints 3 because *p == p[0]
std::cout << p[2] << std::endl; // prints 8 because p[2] == A[2] == 8
std::cout << *(p+2) << std::endl;   // prints 8 because *(p+2) == p[2]
int b = 3;    // regular integer value
int* q = &b;  // the value of q is the address of b
b = 4;        // the value of b is changed
std::cout << *q << std::endl;  // *q == b == 4

char s[] = "Hello";
char *t = s;
std::cout << t << std::endl; // prints "Hello"
char *u = &s[3]; // &s[3] is equivalent to s + 3
std::cout << u << std::endl; // prints "lo"
```

Pointers and References

```
int a = 2;
int& ra = a;   // reference to a
int* pa = &a;  // pointer to a
int b = a;     // copy of a
++a;  // increment a
std::cout << a << std::endl;  // prints 3
std::cout << ra << std::endl; // prints 3
std::cout << *pa << std::endl; // prints 3
std::cout << b << std::endl; // prints 2
int* pb;        // valid, but what pb points to is undefined
int* pc = NULL; // valid, pc points to nothing
std::cout << *pc << std::endl; // Run-time error : pc cannot be dereferenced.
int& rb;        // invalid. reference must refer to something
int& rb = 2;    // invalid. reference must refer to a variable.
```


Summary so far

- Algorithms are computational steps
- `towerOfHanoi` utilizing recursions
- `insertionSort`
  - ✓ Simple but a slow sorting algorithm.
  - ✓ Loop invariant property
- Data types and floating-point precisions
- Operators, `if`, `for`, and `while` statements
- Arrays and strings
- Pointers and References
- Fisher's Exact Tests
- At Home : Reading material for novice C++ users : *http://www.cplusplus.com/doc/tutorial/*

Next Lecture

- Fisher's Exact Test
- More on C++ Programming
  - Standard Template Library
  - User-defined data types