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

Hyun Min Kang

September 8th, 2011
1. Equip the ability to IMPLEMENT computational/statistical IDEAS into working SOFTWARE PROGRAMs

2. Learn COMPUTATIONAL COST management in developing statistical methods.

3. Understand NUMERICAL and RANDOMIZED ALGORITHMS for statistical inference
Algorithm **SingOldMacDonald**

**Data:** \(\text{animals}[1 \cdots n], \text{noises}[1 \cdots n]\)

**Result:** An “Old MacDonald” Song with \(\text{animals}\) and \(\text{noises}\)

**for** \(i = 1\) **to** \(n\) **do**

- Sing "Old MacDonald had a farm, E I E I O”;
- Sing "And on that farm he had some \(\text{animals}[i]\), E I E I O”;
- Sing "With a \(\text{noises}[i]\) \(\text{noises}[i]\) here, and a \(\text{noises}[i]\) \(\text{noises}[i]\) there”;
- Sing "Here a \(\text{noise}[i]\), there a \(\text{noise}[i]\), everywhere a \(\text{noise}[i]\) \(\text{noise}[i]\)”;
  **for** \(j = i - 1\) **downto** 1 **do**
  - Sing "\(\text{noise}[j]\) \(\text{noise}[j]\) here, \(\text{noise}[j]\) \(\text{noise}[j]\) there”;
  - Sing "Here a \(\text{noise}[j]\), there a \(\text{noise}[j]\), everywhere a \(\text{noise}[j]\) \(\text{noise}[j]\)”;
  **end**
- Sing "Old MacDonald had a farm, E I E I O.”;

**end**

(adapted from Jeff Erickson(UIUC)’s class notes)
Key Idea of Insertion Sort

- For $k$-th step, assume that elements $a[1], \cdots, a[k-1]$ are already sorted in order.
- Locate $a[k]$ between index 1, $\cdots$, $k$ so that $a[1], \cdots, a[k]$ are in order
- Move the focus to $k+1$-th element and repeat the same step
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**

- $key = A[j];$
- $i = j - 1;$
- **while** $i > 0$ **and** $A[i] > key$ **do**
  - $A[i + 1] = A[i];$
  - $i = i - 1;$
- end
- $A[i + 1] = key;$

end
Recap - Tower of Hanoi Problem

**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.

**Key Idea - Think Recursively**
- 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 \textsc{TowerOfHanoi}

\textbf{Data:} \( n \): \# disks, \((s, i, d)\): source, intermediate, destination towers

\textbf{Result:} \( n \) disks are moved from \( s \) to \( d \)

\begin{verbatim}
if \( n == 0 \) then
  do nothing;
else
  \textsc{TowerOfHanoi}(n - 1, s, d, i);
  move disk \( n \) from \( s \) to \( d \);
  \textsc{TowerOfHanoi}(n - 1, i, s, d);
end
\end{verbatim}
Today’s and Next Lectures

Today

- Basic Data Types
- Control Structures
- Pointers and Functions

Next few lectures

- The class does NOT focus on teaching programming language itself
- Expect to spend time to be familiar to programming languages yourself
- VERY important to practice writing code on your own.
- Utilize office hours or after-class minutes for detailed questions in practice
Example C++ Development Environment

1 UNIX / gcc environment

- Instructor’s preference
- UNIX environment will be commonly used in large-scale data analysis, so it would be good to be familiar with it.
- Ways to set up UNIX environment
  - Install Linux (e.g. Ubuntu) locally to your computer
  - Download and install Xcode in Mac OS X, and use terminal to access UNIX interface.
  - Install Cygwin to a windows machine (mimics UNIX environment)
  - Connect to U-M login service via SSH using PuTTY or similar software (Refer to http://www.itd.umich.edu/login/ for details)
- Learning Unix-cultured editors such as vi or emacs is also recommended.
Example C++ Development Environment

1. UNIX / gcc environment
2. Windows / Microsoft Visual C++
3. Windows / Borland C++ Builder
4. Mac OS X / Xcode development environment
Getting Started with C++

Writing helloWorld.cpp

```cpp
#include <iostream> // import input/output handling library
int main(int argc, char** argv) {
    std::cout << "Hello, World" << std::endl;
    return 0; // program exits normally
}
```

Compiling helloWorld.cpp

```
user@host:~/$ g++ -o helloWorld helloWorld.cpp
```

Running helloWorld

```
user@host:~/$ ./helloWorld
Hello, World
```
How `helloWorld` works

```c
// type of return value is integer
// return value of main() function is program exit code
// 0 is normal exit code and the others are abnormal exit codes
int main(){
    // name (identifier) of function is 'main'
    // 'main' is a special function, invoked at the beginning of a program.
    // function arguments are surrounded by parentheses
    int argc, // number of command line arguments
    char** argv // list of command line arguments - will explain later
{
    // ... function body goes here
    return 0; // return normal exit code
}
```
How `helloWorld` works

### Using `iostream` to output strings to console

```c++
// includes standard library for handling I/Os (inputs/outputs)
// std::cout and std::endl cannot be recognized without including <iostream>
#include <iostream>

int main (int argc, char** argv) {
    std::cout // standard output stream - messages are printed to console.
        // insertion operator : appends the next value to the output stream
        "Hello, World" // string appended to std::cout via operator<<
        // insertion operator : appends the next value to the output stream
        std::endl; // end-of-line appended to std::cout via operator<<
    return 0;
}
```
Implementing **TowerOfHanoi** Algorithm in C++

towerOfHanoi.cpp

```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:~/$ ./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
```
Homework 0

- Implement the following two programs and send the output screenshots to the instructor (hmkang at umich dot edu) by E-mail
  1. HelloWorld.cpp
  2. TowerOfHanoi.cpp

- Briefly describe your operating system and C++ development environment with your submission

- This homework will not be graded, but mandatory to submit for everyone who wants to take the class for credit

- No due date, but homework 0 must be submitted prior to submitting any other homework.
Declaring Variables

### Variable Declaration and Assignment

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

### Variable Names

```c
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 <= 128
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

<table>
<thead>
<tr>
<th>Type</th>
<th>float</th>
<th>double</th>
<th>long double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>Single</td>
<td>Double</td>
<td>Quadruple</td>
</tr>
<tr>
<td>Size</td>
<td>32 bits</td>
<td>64 bits</td>
<td>128 bits</td>
</tr>
<tr>
<td></td>
<td>(in most modern OS)</td>
<td>4 bytes</td>
<td>8 bytes</td>
</tr>
<tr>
<td>Sign</td>
<td>1 bit</td>
<td>1 bit</td>
<td>1 bit</td>
</tr>
<tr>
<td>Exponent</td>
<td>8 bits</td>
<td>11 bits</td>
<td>15 bits</td>
</tr>
<tr>
<td>Fraction</td>
<td>23 bits</td>
<td>52 bits</td>
<td>112 bits</td>
</tr>
<tr>
<td>(# decimal digits)</td>
<td>7.2</td>
<td>16</td>
<td>34</td>
</tr>
<tr>
<td>Minimum ($&gt;0$)</td>
<td>$1.2 \times 10^{-38}$</td>
<td>$2.2 \times 10^{-308}$</td>
<td>$3.4 \times 10^{-4932}$</td>
</tr>
<tr>
<td>Maximum</td>
<td>$3.4 \times 10^{38}$</td>
<td>$1.8 \times 10^{308}$</td>
<td>$1.2 \times 10^{4932}$</td>
</tr>
</tbody>
</table>
Handling Floating Point Precision Carefully

```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**

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

**String as an array of characters**

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

```c
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

```cpp
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) {
    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

```cpp
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

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

Another while loop

```c
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; points to the next element
}
```
### Recap

### Implementation

### Data Types

### Syntax

### Pointers

### FET

### Summary

---

**Pointers and Loops**

---

#### while loop

```cpp
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

```cpp
// 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

```cpp
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) == 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"
```
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
- At Home: Reading material for novice C++ users:
  
At Home: Write, Compile and Run..

The following list of programs

- helloWorld.cpp
- towerOfHanoi.cpp
- callByValRef.cpp
- precisionExample.cpp

How to ...

Write  Notepad, Vim, Emacs, Eclipse, VisualStudio, etc

Compile  g++ -Wall -o [progName] [progName].cpp
         (Unix, Mac OS X, or Cygwin)

Run  ./[progName] [list of arguments]
Next Lecture

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