

# Biostatistics 615/815 C++ Basics and Precisions

Hyun Min Kang

Januray 11th, 2011

## Recap from the Last Lecture

- Algorithm : A sequence of computational steps from input to output
  - ✓ SINGOLDMACDONALDSONG
  - ✓ INSERTIONSORT
  - ✓ TOWEROFHANOI
- Implementation in C++
  - ✓ helloWorld
  - ✓ towerOfHanoi
  - ✓ insertionSort (skipped)

## Recap - helloWorld

## How helloWorld works

### Writing helloWorld.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
}
```

### main() : First function to be called

```
// 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
// name (identifier) of function is 'main'
// 'main' is a special function, invoked at the beginning of a program.
main
(
    // 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
}
```

### Compiling helloWorld.cpp

Install Cygwin (Windows), Xcode (MacOS), or nothing (Linux).

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

### Running helloWorld

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

## How helloWorld works

### Using iostream to output strings to console

```
// 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;
}
```

## Declaring Variables

### Variable Declaration and Assignment

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

## Today's Lecture

### What to expect

- Basic Data Types
- Control Structures
- Pointers and Functions

### What are expected for the next few weeks

- The class does NOT focus on teaching programming language itself
- Expect to spend time to be familiar to programming languages
  - ✓ Online reference : <http://www.cplusplus.com/doc/tutorial/>
  - ✓ Offline reference : C++ Primer Plus, 5th Edition
- VERY important to practice writing code on your own.
- Utilize office hours or after-class minutes for detailed questions in practice

## Data Types

### Signed Integer Types

```
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 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)        | $1.2 \times 10^{-38}$ | $2.2 \times 10^{-308}$ | $3.4 \times 10^{-4932}$ |
| Maximum             | $3.4 \times 10^{38}$  | $1.8 \times 10^{308}$  | $1.2 \times 10^{4932}$  |

## 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"
```

## 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;
}
```

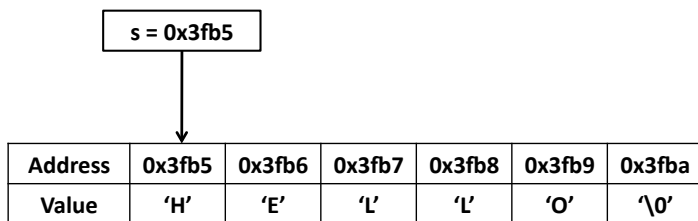
## Assignment and Arithmetic Operators

```
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;
```

## Pointers



### 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
}
```

## 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 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

```
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 3 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.
```

## Command line arguments

```
int main(int argc, char** argv)
```

`int argc` Number of command line arguments, including the program name itself

`char** argv` List of command line arguments as double pointer

- One \* for representing 'array' of strings
- One \* for representing string as 'array' of characters
- ✓ `argv[0]` represents the program name (e.g., `helloWorld`)
- ✓ `argv[1]` represents the first command-line argument
- ✓ `argv[2]` represents the second command-line argument
- ✓ ...
- ✓ `argv[argc-1]` represents the last command-line argument

## Handling command line arguments

`echo.cpp` - echoes command line arguments to the standard output

```
#include <iostream>
int main(int argc, char** argv) {
    for(int i=1; i < argc; ++i) { // i=1 : 2nd argument (skip program name)
        if ( i > 1 ) // print blank if there is an item already printed
            std::cout << " ";
        std::cout << argv[i]; // print each command line argument
    }
    std::cout << std::endl; // print end-of-line at the end
}
```

### Compiling and running `echo.cpp`

```
user@host:~/ $ g++ -o echo echo.cpp
user@host:~/ $ ./echo 1 2 3 my name is foo
1 2 3 my name is foo
```

## Functions

### Core element of function

**Type** Type of return values

**Arguments** List of comma separated input arguments

**Body** Body of function with "return [value]" at the end

### Defining functions

```
int square(int a) {
    return (a*a);
}
```

### Calling functions

```
int x = 5;
std::cout << square(x) << std::endl; // prints 25
```

## Let's implement Fisher's exact Test

### A 2 × 2 table

|          | Placebo | Treatment | Total |
|----------|---------|-----------|-------|
| Diseased | a       | b         | a+b   |
| Cured    | c       | d         | c+d   |
| Total    | a+c     | b+d       | n     |

### Desired Program Interface and Results

```
user@host:~/ $ ./fishersExactTest 1 2 3 0
Two-sided p-value is 0.4
user@host:~/ $ ./fishersExactTest 2 7 8 2
Two-sided p-value is 0.0230141
user@host:~/ $ ./fishersExactTest 20 70 80 20
Two-sided p-value is 5.90393e-16
```

## Call by value vs. Call by reference

### callByValRef.cpp

```
#include <iostream>
int foo(int a) {
    a = a + 1;
    return a;
}
int bar(int& a) {
    a = a + 1;
    return a;
}
int main(int argc, char** argv) {
    int x = 1, y = 1;
    std::cout << foo(x) << std::endl; // prints 2
    std::cout << x << std::endl; // prints 1
    std::cout << bar(y) << std::endl; // prints 2
    std::cout << y << std::endl; // prints 2
}
```

## Fisher's Exact Test

### Possible 2 × 2 tables

|          | Placebo | Treatment | Total |
|----------|---------|-----------|-------|
| Diseased | x       | a+b-x     | a+b   |
| Cured    | a+c-x   | d-a+x     | c+d   |
| Total    | a+c     | b+d       | n     |

### Hypergeometric distribution

Given  $a + b, c + d, a + c, b + d$  and  $n = a + b + c + d$ ,

$$\Pr(x) = \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{x!(a+b-x)!(a+c-x)!(d-a+x)!n!}$$

### Fisher's Exact Test (2-sided)

$$p_{FET}(a, b, c, d) = \sum_x \Pr(x) I[\Pr(x) \leq \Pr(a)]$$

## intFishersExactTest.cpp - main() function

```
#include <iostream>
double hypergeometricProb(int a, int b, int c, int d); // defined later
int main(int argc, char** argv) {
    // read input arguments
    int a = atoi(argv[1]), b = atoi(argv[2]), c = atoi(argv[3]), d = atoi(argv[4]);
    int n = a + b + c + d;
    // find cutoff probability
    double pCutoff = hypergeometricProb(a,b,c,d);
    double pValue = 0;
    // sum over probability smaller than the cutoff
    for(int x=0; x <= n; ++x) { // among all possible x
        if ( a+b-x >= 0 && a+c-x >= 0 && d-a+x >=0 ) { // consider valid x
            double p = hypergeometricProb(x,a+b-x,a+c-x,d-a+x);
            if ( p <= pCutoff ) pValue += p;
        }
    }
    std::cout << "Two-sided p-value is " << pValue << std::endl;
    return 0;
}
```

## Considering Precision Carefully

### factorial.cpp

```
int fac(int n) { // calculates factorial
    int ret;
    for(ret=1; n > 0; --n) { ret *= n; }
    return ret;
}
int main(int argc, char** argv) {
    int n = atoi(argv[1]);
    std::cout << n << "! = " << fac(n) << std::endl;
}
```

### Running Examples

```
user@host:~/$ ./factorial 10
10! = 362880 // correct
user@host:~/$ ./factorial 12
12! = 479001600 // correct
user@host:~/$ ./factorial 13
13! = 1932053504 // INCORRECT
```

## intFishersExactTest.cpp

### hypergeometricProb() function

```
int fac(int n) { // calculates factorial
    int ret;
    for(ret=1; n > 0; --n) { ret *= n; }
    return ret;
}
double hypergeometricProb(int a, int b, int c, int d) {
    int num = fac(a+b) * fac(c+d) * fac(a+c) * fac(b+d);
    int den = fac(a) * fac(b) * fac(c) * fac(d) * fac(a+b+c+d);
    return (double)num/(double)den;
}
```

### Running Examples

```
user@host:~/$ ./intFishersExactTest 1 2 3 0
Two-sided p-value is 0.4 // correct
user@host:~/$ ./intFishersExactTest 2 7 8 2
Two-sided p-value is 4.41018 // INCORRECT
```

## doubleFishersExactTest.cpp

### new hypergeometricProb() function

```
double fac(int n) { // main() function remains the same
    double ret; // use double instead of int
    for(ret=1.; n > 0; --n) { ret *= n; }
    return ret;
}
double hypergeometricProb(int a, int b, int c, int d) {
    double num = fac(a+b) * fac(c+d) * fac(a+c) * fac(b+d);
    double den = fac(a) * fac(b) * fac(c) * fac(d) * fac(a+b+c+d);
    return num/den; // use double to calculate factorials
}
```

### Running Examples

```
user@host:~/$ ./doubleFishersExactTest 2 7 8 2
Two-sided p-value is 0.023041
user@host:~/$ ./doubleFishersExactTest 20 70 80 20
Two-sided p-value is 0 (fac(190) > 1e308 - beyond double precision)
```

## How to perform Fisher's exact test with large values

### Problem - Limited Precision

- int handles only up to fac(12)
- double handles only up to fac(170)

### Solution - Calculate in logarithmic scale

$$\log \Pr(x) = \log(a+b)! + \log(c+d)! + \log(a+c)! + \log(b+d)! - \log x! - \log(a+b-x)! - \log(a+c-x)! - \log(d-a+x)! - \log n!$$

$$\log(p_{FET}) = \log \left[ \sum_x \Pr(x) I(\Pr(x) \leq \Pr(a)) \right]$$

$$= \log \Pr(a) + \log \left[ \sum_x \exp(\log \Pr(x) - \log \Pr(a)) I(\log \Pr(x) \leq \log \Pr(a)) \right]$$

## Filling the rest

### logHypergeometricProb()

```
double logFac(int n) {
    double ret;
    for(ret=0.; n > 0; --n) { ret += log((double)n); }
    return ret;
}

double logHypergeometricProb(int a, int b, int c, int d) {
    return logFac(a+b) + logFac(c+d) + logFac(a+c) + logFac(b+d) - logFac(a)
        - logFac(b) - logFac(c) - logFac(d) - logFac(a+b+c+d);
}
```

### Running Examples

```
user@host:~/ $ ./logFishersExactTest 2 7 8 2
Two-sided log10-p-value is -1.63801, p-value is 0.0230141
user@host:~/ $ ./logFishersExactTest 20 70 80 20
Two-sided log10-p-value is -15.2289, p-value is 5.90393e-16
user@host:~/ $ ./logFishersExactTest 200 700 800 200
Two-sided log10-p-value is -147.563, p-value is 2.73559e-148
```

## logFishersExactTest.cpp - main() function

```
#include <iostream>
#include <math> // for calculating log() and exp()
double logHypergeometricProb(int a, int b, int c, int d); // defined later
int main(int argc, char** argv) {
    int a = atoi(argv[1]), b = atoi(argv[2]), c = atoi(argv[3]), d = atoi(argv[4]);
    int n = a + b + c + d;
    double logpCutoff = logHypergeometricProb(a,b,c,d);
    double pFraction = 0;
    for(int x=0; x <= n; ++x) { // among all possible x
        if ( a+b-x >= 0 && a+c-x >= 0 && d-a+x >= 0 ) { // consider valid x
            double l = logHypergeometricProb(x,a+b-x,a+c-x,d-a+x);
            if ( l <= logpCutoff ) pFraction += exp(l - logpCutoff);
        }
    }
    double logpValue = logpCutoff + log(pFraction);
    std::cout << "Two-sided log10-p-value is " << logpValue/log(10.) << std::endl;
    std::cout << "Two-sided p-value is " << exp(logpValue) << std::endl;
    return 0;
}
```

## Even faster

### Computational speed for large dataset

```
time ./logFishersExactTest 2000 7000 8000 2000
Two-sided log10-p-value is -1466.13, p-value is 0
real 0m42.614s

time ./fastLogFishersExactTest 2000 7000 8000 2000
Two-sided log10-p-value is -1466.13, p-value is 0
real 0m0.007s
```

### How to make it faster?

*To be continued...*



## 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
- Functions
- Fisher's Exact Test
  - ✓ intFishersExactTest - works only tiny datasets
  - ✓ doubleFishersExactTest - handles small datasets
  - ✓ logFishersExactTest - handles hundreds of observations
- At Home : Reading material for novice C++ users :  
<http://www.cplusplus.com/doc/tutorial/>

## Next Lecture

### Fisher's Exact Test

- fastLogFishersExactTest
- oneSidedFastLogFishersExactTest - First homework

### More on C++ Programming

- Standard Template Library
- User-defined data types

### Divide and Conquer Algorithms

- Binary Search
- Merge Sort

## At Home : Write, Compile and Run..

### The following list of programs

- helloWorld.cpp
- towerOfHanoi.cpp
- insertionSort.cpp
- echo.cpp
- precisionExample.cpp
- callByValRef.cpp
- factorial.cpp
- intFishersExactTest.cpp
- doubleFishersExactTest.cpp
- logFishersExactTest.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]`