

Biostatistics 615/815 - Lecture 3

C++ Basics & Implementing Fisher's Exact Test

Hyun Min Kang

September 13th, 2011

helloWorld.cpp : Getting Started with C++

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
}
```

helloWorld.cpp : Getting Started with C++

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
}
```

Compiling helloWorld.cpp

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

helloWorld.cpp : Getting Started with C++

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
}
```

Compiling helloWorld.cpp

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

Running helloWorld

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

towerOfHanoi.cpp : Tower of Hanoi Algorithm in C++

towerOfHanoi.cpp

```
#include <iostream>
#include <cstdlib> // include this for atoi() function
// 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;
}
```

Recap - Floating Point Precisions

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;
}
```

Quiz - Precision Example

pValueExample.cpp

```
#include <iostream>
int main(int argc, char** argv) {
    float pUpper = 1e-8; // small p-value at upper tail
    float pLower = 1-pUpper; // large p-value at lower tail
    std::cout << "upper tail p = " << pUpper << std::endl;
    std::cout << "lower tail p = " << pLower << std::endl;

    float pLowerComplement = 1-pLower; // complement of lower tail
    std::cout << "complement lower tail p = " << pLowerComplement << std::endl;
    return 0;
}
```

Recap - Arrays and Pointers

```
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] == 3) << std::endl; // true
std::cout << (*p == 3) << std::endl; // true
std::cout << (p[2] == 8) << std::endl; // true
std::cout << (*(p+2) == 8) << std::endl; // true

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 == 4) << std::endl; // true, *q == b == 4
```


String as an Array of Characters

```
char s[] = "Hello"; // Array of characters as string
std::cout << s << std::endl; // prints "Hello"
char *t = s;        // t is address containing 'H'
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
```

Pointers and References

Valid and invalid pointer expressions

```
int* pb; // undefined address -- valid
int* pc = NULL; // null address -- points nothing -- valid
std::cout << *pc << std::endl; // Run-time error : pc cannot be dereferenced.
int& rb; // Compile-time error : undefined reference -- invalid
int& rb = 2; // Compile-time error : reference to non-variable -- invalid
```

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

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);  
}
```

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
```


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
    return 0;
}
```

Call by value vs. Call by reference

Call-by-value is useful

- If you want to avoid unwanted changes in the caller's variables by the callee
- If you want to abstract the callee as a function only between inputs and outputs.

Call-by-reference is useful

- If you want to update the caller's variables by invoking the function.
- If you want to avoid copying an object consuming large memory to reduce memory consumption and computational time for copying the object.
 - As an extreme example, passing an 1GB object using call-by-value consumes additional 1GB of memory, but call-by-reference requires almost zero additional memory.

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

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
```

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;
}

```

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
```

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
```


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

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

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

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;
}

```

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
```

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

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

How to make it faster?

- Most time consuming part is the repetitive computation of factorial
 - # of `logHypergeometricProbs` calls is $\leq a + b + c + d = n$
 - # of `logFac` call $\leq 9n$
 - # of `log` calls $\leq 9n^2$ - could be billions in the example above
- Key Idea is to store `logFac` values to avoid repetitive computation

newFac.cpp : new operator for dynamic memory allocation

```
#include <iostream>
#include <cstdlib>
int main(int argc, char** argv) {
    int n = atoi(argv[1]); // takes an integer argument
    double* facs = new double[n+1]; // allocate variable-sized array
    facs[0] = 1;
    for(int i=1; i <= n; ++i) {
        facs[i] = facs[i-1] * i; // calculate factorial
    }
    for(int i=n; i >= 0; --i) { // prints factorial values from n! to 0!
        std::cout << i << "! = " << facs[i] << std::endl;
    }
    delete [] facs; // if allocated by new[], must be freed by delete[]
    return 0;
}
```

fastFishersExactTest.cpp

Preambles and Function Declarations

```
#include <iostream>
#include <cmath>
#include <cstdlib>

// *** defined previously
double logHypergeometricProb(double* logFacs, int a, int b, int c, int d);

// *** New function ***
void initLogFacs(double* logFacs, int n);

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


fastFishersExactTest.cpp - main() function

```

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* logFacs = new double[n+1]; // *** dynamically allocate memory logFacs[0..n] ***
    initLogFacs(logFacs, n);           // *** initialize logFacs array ***
    double logpCutoff = logHypergeometricProb(logFacs,a,b,c,d); // *** logFacs added
    double pFraction = 0;
    for(int x=0; x <= n; ++x) {
        if ( a+b-x >= 0 && a+c-x >= 0 && d-a+x >=0 ) {
            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;
    delete [] logFacs;
    return 0;
}

```

fastFishersExactTest.cpp - other functions

function initLogFacs()

```
void initLogFacs(double* logFacs, int n) {
    logFacs[0] = 0;
    for(int i=1; i < n+1; ++i) {
        logFacs[i] = logFacs[i-1] + log((double)i); // only n times of log() calls
    }
}
```

function logHyperGeometricProb()

```
double logHypergeometricProb(double* logFacs, int a, int b, int c, int d) {
    return logFacs[a+b] + logFacs[c+d] + logFacs[a+c] + logFacs[b+d]
        - logFacs[a] - logFacs[b] - logFacs[c] - logFacs[d] - logFacs[a+b+c+d];
}
```

Summary so far

- Algorithms are computational steps
- towerOfHanoi utilizing recursions
- insertionSort
 - ✓ Loop invariant property
- Data types and floating-point precisions
- Operators, if, for, and while statements
- Arrays and strings
- Pointers, References, and Functions
- Implementations of Fisher's Exact Test
 - ✓ intFishersExactTest - works only tiny datasets
 - ✓ doubleFishersExactTest - handles small datasets
 - ✓ logFishersExactTest - handles hundreds of observations
 - ✓ fastFishersExactTest - avoid repetitive computations

At Home : Write, Compile and Run..

The following list of programs

- helloWorld.cpp
- towerOfHanoi.cpp
- echo.cpp
- precisionExample.cpp
- callByValRef.cpp
- factorial.cpp
- newFac.cpp
- intFishersExactTest.cpp
- doubleFishersExactTest.cpp
- logFishersExactTest.cpp
- fastFishersExactTest.cpp

At Home : Write, Compile and Run..

The following list of programs

- helloWorld.cpp
- towerOfHanoi.cpp
- echo.cpp
- precisionExample.cpp
- callByValRef.cpp
- factorial.cpp
- newFac.cpp
- intFishersExactTest.cpp
- doubleFishersExactTest.cpp
- logFishersExactTest.cpp
- fastFishersExactTest.cpp

How to ...

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

Compile `g++ -Wall -o [progName] [progName].cpp`

Run `./[progName] [list of arguments]`

Assignments and Next Lectures

Problem Set #1

- Posted on the class web page.
- Due on September 27th.

Assignments and Next Lectures

Problem Set #1

- Posted on the class web page.
- Due on September 27th.

More on C++ Programming

- Standard Template Library
- User-defined data types

Assignments and Next Lectures

Problem Set #1

- Posted on the class web page.
- Due on September 27th.

More on C++ Programming

- Standard Template Library
- User-defined data types

Divide and Conquer Algorithms

- Binary Search
- Merge Sort