Biostatistics 615/815 Lecture 4:
User-defined Data Types,
Standard Template Library,
and Divide and Conquer Algorithms

Hyun Min Kang

Januray 18th, 2011

---

### Recap
- **C++**
- **STL**
- **Recursion**
- **Gcd**
- **Division and Conquer**
- **MergeSort**

---

**fastFishersExactTest.cpp**

#### fastFishersExactTest.cpp - main() function

```cpp
#include <iostream>  // everything remains the same except for lines marked with ***
#include <cmath>

double logHypergeometricProb(double* logFacs, int a, int b, int c, int d); // ***
void initLogFacs(double* logFacs, int n); // *** New 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+2];  // *** dynamically allocate memory logFacs[0..n] ***
    initLogFacs(logFacs, n);          // *** initialize logFacs array ***
    double logCutoff = 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 <= logCutoff ) pFraction += exp(1 - logCutoff);
        }
    }
    double logValue = logCutoff + log(pFraction);
    std::cout << "Two-sided p-value is " << logValue/log(10.) << std::endl;
    std::cout << "Two-sided p-value is " << exp(logValue) << std::endl;
    delete [] logFacs;
    return 0;
}
```

---

**fastFishersExactTest.cpp - other functions**

#### function initLogFacs()

```cpp
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()

```cpp
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];
}
```

---

### Announcements
- **C++**
- **STL**
- **Recursion**
- **Gcd**
- **Division and Conquer**
- **MergeSort**

---

**Seating in classes**

- Currently # enrollment is around 25-26
- The classroom is supposed to hold up to 36
- When the classroom is full, the seating priority should be given to students enrolled in the class.
- Any idea to resolve seating issue?

---

**Homework #1**

- How is it going?
- Any questions?
Projects for BIOSTAT815

Principles

- Project can be done in pairs
- Single-individual project is possible, but will be graded in the same basis with pair-of-individuals projects.
- Each project has different levels of difficulty, which will be accounted for in the evaluation.
- Suggestions of new projects will be welcomed (subject to discussion with the instructor).

Action Items

- Rank the project preference (for every project)
- Nominate name(s) to perform the project in pairs, if desired.
- E-mail to hmkang@umich.edu, with title "815 Project - [your name]" by Friday 11:59pm.

List of 815 Projects

1. MCMC-based p-values of large contingency table
   - **Input** An \( I \times J \) contingency table
   - **Output** p-values of the contingency table, based on MCMC method
   - **Note** Need to demonstrate that the method provides p-values consistent to exact method when possible to compute

2. Rapid evaluation of logistic regression models
   - **Input** \( n \times p \) matrix \( X \) and binary response variables \( y \) of size \( n \)
   - **Output** MLE \( \beta \), \( SE(\beta) \) and p-values \( \logit[\Pr(y = 1)] = X\beta \)
   - **Note** Need to be fast to be able to apply for a large number of tests simultaneously

3. HMM-based profile alignment of sequence pairs
   - **Input** Two sequences of \( \{A, C, G, T\} \)
   - **Output** HMM-based probabilistic alignment between the two sequences, and comparison with Smith-Waterman algorithm
   - **Note** Allow banded computation for improved efficiency. Multiple sequence alignment algorithms are more than welcomed

4. Rapid clustering of gene expression data
   - **Input** \( n \times g \) matrix of normalized gene expression across \( n \) samples and \( g \) genes
   - **Output** Clusters of genes using at least two clustering methods, among (a) hierarchical clustering, (b) k-means clustering, (c) spectral clustering, (d) E-M clustering, and (e) other robust clustering methods
List of 815 Projects

5. EM-algorithm for genotype calling from intensities

<table>
<thead>
<tr>
<th>Input</th>
<th>List of two dimensional intensities across $n$ unrelated samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Possible genotype label AA, AB, BB, NN and posterior probability of each individual genotype, based on EM algorithm with mixture of Gaussian or Student $t$</td>
</tr>
</tbody>
</table>

6. A Bayesian SNP calling algorithm from sequence data

| Input | For each individual and genomic position, genotype likelihood, defined as $\Pr(Reads|G_1 G_2)$, for each possible genotype $G_1 G_2$ |
|-------|------------------------------------------------------------------------------------------------------------------|
| Output| Posterior probability of a position being SNP |
| Note  | Alternatively, starting from aligned sequence (BAM format) is also possible |

7. Short read alignment

<table>
<thead>
<tr>
<th>Input</th>
<th>Short sequence reads ($n \sim 100$), and a reference genome up to the size of human genome ($3 \times 10^9$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Best possible genomic position to align the sequence onto</td>
</tr>
<tr>
<td>Note</td>
<td>OK to mimic existing short aligning software, or have special feature such as statistical alignment into multiple places</td>
</tr>
</tbody>
</table>

8. Solution using MapReduce Framework

<table>
<thead>
<tr>
<th>Input</th>
<th>Any of the problems suggested by 1-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Solution implemented under MapReduce framework</td>
</tr>
<tr>
<td>Note</td>
<td>For extra credit. MapReduce framework is a scalable parallel programming technique for cloud computing</td>
</tr>
</tbody>
</table>

The flexibility and complexity of C++

**Flexibility of C++ : What C++ offers**

- Both reference and pointer types (unlike C or Java)
- User-defined data type via classes (unlike C)
- Inheritance (unlike C) and multiple inheritance (unlike C or Java)
- Explicit allocation and deallocation of memory (unlike Java)
- Templates that operate with generic types (unlike C or earlier Java)
- And more.. (operator overloading, dynamic polymorphism, etc)

**Complexity of C++**

There is a hoax claiming that the C++ designer Bjarne Stroustrup admitted in an interview that he developed the C++ language solely to create high-paying jobs for programmers, because C language is too easy to distinguish talented programmers from ordinary programmers.

Why use C++ in the class?

**C**

- C is relatively simple to use
- Library support for basic data structure (array, hash, etc) is limited.
- Limited support on object-oriented programming.

**Java (or C#)**

- Object-oriented, clear and simple language
- No explicit control on memory management
- Performance can be substantially worse than C/C++ in some applications
Why use C++ in the class?

C++

- Explicit memory control with great performance
- Support from standard template library and other libraries
- High complexity - will use only core features during lectures
  - Classes with member variable, member function, inheritance, and dynamic polymorphism
  - No operator overloading, multiple inheritance, deep/shallow copy
  - Standard Template Library (STL)
  - Other useful libraries
- For advanced use of C++, read Effective C++ or take another programming course.

Adding member functions

```cpp
#include <iostream>
#include <cmath>
class Point {
public:
    double x;
    double y;
    double distanceFromOrigin() { // member function
        return sqrt( x*x + y*y );
    }
};

int main(int argc, char** argv) {
    Point p;
    p.x = 3.;
    p.y = 4.;
    std::cout << p.distanceFromOrigin() << std::endl; // prints 5
}
```

Classes and user-defined data type

C++ Class

- A user-defined data type with
  - Member variables
  - Member functions

An example C++ Class

```cpp
class Point { // definition of a class as a data type
public: // making member variables/functions accessible outside the class
    double x; // member variable
    double y; // another member variable
};
Point p; // A class object as an instance of a data type
p.x = 3.; // assign values to member variables
p.y = 4.;
```

Constructor - A better way to initialize an object

```cpp
#include <iostream>
#include <cmath>
class Point {
public:
    double x;
    double y;
    Point(double px, double py) { // constructor defines here
        x = px;
        y = py;
    }
    // equivalent to -- Point(double px, double py) : x(px), y(py) {
    double distanceFromOrigin() { return sqrt( x*x + y*y );}
};

int main(int argc, char** argv) {
    Point p(3,4); // calls constructor with two arguments
    std::cout << p.distanceFromOrigin() << std::endl; // prints 5
}
```
More member functions

```cpp
#include <iostream>
#include <cmath>

class Point {
public:
    double x, y;
    Point(double px, double py) { x = px; y = py; }
    double distanceFromOrigin() { return sqrt(x*x + y*y); }
    double distance(Point& p) { // call-by-reference to avoid unnecessary copy
        return sqrt((x-p.x)*(x-p.x) + (y-p.y)*(y-p.y));
    }
    void print() { // print the content of the point
        std::cout << "(" << x << ", " << y << ")" << std::endl;
    }
};

int main(int argc, char** argv) {
    Point p1(3,4), p2(15,9);
    p1.print(); // prints (3,4)
    std::cout << p1.distance(p2) << std::endl; // prints 13
}
```

Points to an object: objectPointers.cpp

```cpp
#include <iostream>
#include <cmath>

class Point {
public:
    double x, y;
    Point(double px, double py) { x = px; y = py; }
    double distance(Point& p) { return sqrt((x-p.x)*(x-p.x) + (y-p.y)*(y-p.y)); }
    void print() { std::cout << "(" << x << ", " << y << ")" << std::endl; }
};

int main(int argc, char** argv) {
    Point p1(3,4); // static allocation
    Point* pp2 = new Point(5,12); // dynamic allocation
    Point* pp3 = &p1;
    pp3->print(); // Member function access - prints (3,4)
    pp3->print(); // Member function access via pointer - prints (5,12)
    std::cout << "p1.x = " << p1.x << std::endl; // prints 3
    std::cout << "pp2->x = " << pp2->x << std::endl; // prints 5
    std::cout << "(*pp2).x = " << (*pp2).x << std::endl; // same to pp2->x
    delete pp2; // allocated memory must be deleted
}
```

Static and dynamic allocation: staticVsDynamic.cpp

```cpp
#include <iostream>
#include <cmath>

class Point {
    double x, y;
    Point(double px, double py) { x = px; y = py; }
    double distance(Point& p) { return sqrt((x-p.x)*(x-p.x) + (y-p.y)*(y-p.y)); }
    void print() { std::cout << "(" << x << ", " << y << ")" << std::endl; }
};

int main(int argc, char** argv) {
    Point p1 = foo(3,4); // p1 is invalid after foo() is terminated.
    Point* p2 = &p1; // p2 is a valid pointer
    p1->print(); // prints arbitrary value (may cause fatal error)
    p2->print(); // prints (3,4)
    delete p2; // object created by 'new' must be 'delete'd.
}
```
Using Standard Template Library (STL)

Why STL?
- Included in the C++ Standard Library
- Allows to use key data structure and I/O interface easily
- Objects behaves like built-in data types

Key classes
- Strings library: `<string>`
- Input/Output Handling: `<iostream>`, `<fstream>`, `<sstream>`
- Variable size array: `<vector>`
- Other containers: `<set>`, `<map>`, `<stack>`

More STL example

```
#include <iostream>
#include <string>
#include <map>
int main(int argc, char** argv) {
    std::map<std::string,int> stringCounts; // contains a pair of string and counts
    for(int i=1; i < argc; ++i) // build (word,count) map
        stringCounts[argv[i]]++; // map[key] = value
    for(std::map<std::string,int>::iterator i = stringCounts.begin(); i != stringCounts.end(); ++i) // iterate over the map and print (key,value) pairs
        std::cout << i->first << " " << i->second << std::endl;
    return 0;
}
```

STL Use in INSERTIONSORT Algorithm

```
#include <iostream>
#include <string>
#include <map>
int main(int argc, char** argv) {
    std::map<std::string,int> stringCounts; // contains a pair of string and counts
    for(int i=1; i < argc; ++i) // build (word,count) map
        stringCounts[argv[i]]++; // map[key] = value
    for(std::map<std::string,int>::iterator i = stringCounts.begin(); i != stringCounts.end(); ++i) // iterate over the map and print (key,value) pairs
        std::cout << i->first << " " << i->second << std::endl;
    return 0;
}
```

A running example

```
user@host:~ $ ./argsCount here moo moo here moo here moo there moo there moo moo
3 here
8 moo
3 there
```
**STL Use in **InsertionSort** Algorithm**

**insertionSort.cpp - insertionSort() function**

```cpp
// perform insertion sort on A
void insertionSort(std::vector<int>& A) { // call-by-reference
    for(int j=1; j < A.size(); ++j) { // 0-based index
        int key = A[j]; // key element to relocate
        int i = j-1; // index to be relocated
        while((i >= 0) && (A[i] > key)) { // find position to relocate
            A[i+1] = A[i]; // shift elements
            --i; // update index to be relocated
        }
        A[i+1] = key; // relocate the key element
    }
}
```

**STL use in **InsertionSort** Algorithm**

**insertionSort.cpp - main() function**

```cpp
#include <iostream>
#include <vector>

void printArray(std::vector<int>& A); // declared here, defined later
void insertionSort(std::vector<int>& A); // declared here, defined later

int main(int argc, char** argv) {
    std::vector<int> v; // contains array of unsorted/sorted values
    int tok; // temporary value to take integer input
    while ( std::cin >> tok ) // read an integer from standard input
        v.push_back(tok) // and add to the array
    std::cout << "Before sorting:");
    printArray(v); // print the unsorted values
    insertionSort(v); // perform insertion sort
    std::cout << "After sorting:");
    printArray(v); // print the sorted values
    return 0;
}
```

**Example of recursion**

**Factorial**

```cpp
int factorial(int n) {
    if ( n == 0 )
        return 1;
    else
        return n * factorial(n-1); // tail recursion - can be transformed into loop
}
```

**towerOfHanoi**

```cpp
void towerOfHanoi(int n, int s, int i, int d) { // n disks, from s to d via i
    if ( n > 0 ) {
        towerOfHanoi(n-1,s,d,i); // recursively move n-1 disks from s to i
        std::cout << "Disk " << n << " : " << s << " -> " << d << std::endl;
        towerOfHanoi(n-1,i,s,d); // recursively move n-1 disks from i to d
    }
}
```

**Defintion of recursion**

Recursion See "Recursion".

**Another definition of recursion**

Recursion If you still don’t get it, see: "Recursion"

**Key components of recursion**

- A function that is part of its own definition
- Terminating condition (to avoid infinite recursion)
Euclid’s algorithm

Algorithm GCD

Data: Two integers $a$ and $b$

Result: The greatest common divisor (GCD) between $a$ and $b$

if $a$ divides $b$ then
  return $a$
else
  Find the largest integer $t$ such that $at + r = b$;
  return GCD($r$, $a$)
end

Function gcd()

```cpp
int gcd (int a, int b) {
  if (a == 0) return b; // equivalent to returning a when b % a == 0
  else return gcd(b % a, a);
}
```

A running example of Euclid’s algorithm

Evaluation of gcd(477, 246)

```
gcd(477, 246)
gcd(231, 246)
gcd(15, 231)
gcd(6, 15)
gcd(3, 6)
gcd(0, 3)
gcd(477, 246) == 3
```

Divide-and-conquer algorithms

Solve a problem recursively, applying three steps at each level of recursion

**Divide** the problem into a number of subproblems that are smaller instances of the same problem

**Conquer** the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.

**Combine** the solutions to subproblems into the solution for the original problem

Binary Search

// assuming a is sorted, return index of array containing the key,
// among [start...end]. Return -1 if no key is found
int binarySearch(std::vector<int>& a, int key, int start, int end) {
  if (start > end) return -1; // search failed
  int mid = (start+end)/2;
  if (key == a[mid]) return mid; // terminate if match is found
  if (key < a[mid]) // divide the remaining problem into half
    return binarySearch(a, key, start, mid-1);
  else
    return binarySearch(a, key, mid+1, end);
}
mergeSort.cpp - main()

```cpp
#include <iostream>
#include <vector>
#include <climits>

void mergeSort(std::vector<int>& a, int p, int r) { // defined later
    void printArray(std::vector<int>& A); // same as insertionSort

    // same to insertionSort.cpp except for one line
    int main(int argc, char** argv) { 
        std::vector<int> v;
        int tok;
        while ( std::cin >> tok ) {
            v.push_back(tok);
        }
        std::cout << "Before sorting: ";
        printArray(v);
        mergeSort(v, 0, v.size()-1); // differs from insertionSort.cpp
        std::cout << "After sorting: ";
        printArray(v);
        return 0;
    }
```

mergeSort.cpp - merge() function

```cpp
void merge(std::vector<int>& a, int p, int q, int r) {
    std::vector<int> aL, aR; // copy a[p..q] to aL and a[q+1..r] to aR
    for (int i=p; i <= q; ++i) aL.push_back(a[i]);
    for (int i=q+1; i <= r; ++i) aR.push_back(a[i]);
    aL.push_back(INT_MAX); aR.push_back(INT_MAX); // append additional value to avoid out-of-bound
    a.push_back(INT_MAX); // pick smaller one first from aL and aR and copy to a[p..r]
    for (int k=p, i=0, j=0; k <= r; ++k) {
        if ( aL[i] <= aR[j] ) { 
            a[k] = aL[i];
            ++i;
        } 
        else {
            a[k] = aR[j];
            ++j;
        }
    }
```

### Merge Sort

**Divide and conquer algorithm**

- **Divide**  
  Divide the \( n \) element sequence to be sorted into two subsequences of \( n/2 \) elements each
- **Conquer**  
  Sort the two subsequences recursively using merge sort
- **Combine**  
  Merge the two sorted subsequences to produce the sorted answer
mergeSort.cpp - mergeSort() function

```cpp
void mergeSort(std::vector<int>& a, int p, int r) {
    if (p < r) {
        int q = (p+r)/2; // find a point to divide the problem
        mergeSort(a, p, q); // divide-and-conquer
        mergeSort(a, q+1, r); // divide-and-conquer
        merge(a, p, q, r); // combine the solutions
    }
}
```

Next Lecture

- Sorting Algorithms
  - Bubble Sort
  - Merge Sort
  - Quicksort
- Dynamic Programming