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

Hyun Min Kang

September 15th, 2011
#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+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;
}
Recap

C++

Classes

STL

InsertionSort

Recursion

Gcd

Divide and Conquer

815 Projects

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];
}
```
Projects for BIOSTAT815

Principles

- Pairing per project is encouraged.
- Individual project is possible, but the expected amount of work is the same to paired projects.
- Each project has different levels of difficulty, which will be accounted for in the evaluation.
- Proposal of new project related to your research is more then welcomed. (Requires instructor’s approval).
Projects for BIOSTAT815

Action Items

- Rank the project preference (up to three)
- Nominate name(s) to perform the project in pairs, if desired.
- E-mail to hmkang@umich.edu, with title ”815 Project - [your name]” by next week.
1. MCMC-based p-values of large contingency table

**Given** An $I \times J$ contingency table, where $I$ and $J$ can be a large numbers

**Want** p-values of the observed contingency table

**How** Use Markov-Chain Monte Carlo (MCMC) method
2. Clustering gene expression data

**Input**  \( n \times g \) matrix of normalized gene expression across \( n \) samples and \( g \) genes

**Output** Clusters of genes into \( k \) different clusters

**How** Using at least two of the following algorithms (a) hierarchical clustering (where \( k \) is unnecessary), (b) \( k \)-means clustering, (c) spectral clustering (d) E-M clustering (e) or other robust clustering algorithms
3. Rapid inference of large-scale GLM inference

**Input**
- \( X : m \times n \) matrix of predictor variables
- \( Y : g \times n \) matrix of response variables
- \( Z : p \times n \) matrix of covariates
- Link function: must include linear and logit link

**Output**
For each \((i, j)\) representing GLM \( y_j \sim x_i \beta_{ij} + Z \gamma \)
- \( P : m \times g \) matrix of p-values
- \( B : m \times g \) matrix of \( \hat{\beta}_{ij} \)
- \( E : m \times g \) matrix of \( SE(\beta_{ij}) \)

**How**
By implementing efficient linear and logistic regression
4. EM-algorithm for genotype calling from intensities

**Input** List of two dimensional intensities across \( n \) unrelated samples and \( m \) independent markers

**Output** Possible genotype label AA, AB, BB, NN and posterior probability of each individual genotype, based on EM algorithm with mixture of Gaussian or Student \( t \)

**How** By fitting to mixture of Gaussian or t-distribution
5. A Bayesian SNP calling algorithm from shotgun sequence data

**Input**
For each individual and genomic position, genotype likelihood, defined as $\Pr(\text{Reads}|G_1 G_2)$, for each possible genotype $G_1 G_2$.

**Output**
Posterior probability of a position being SNP.

**How**
Using a Bayesian model with MLE allele frequency estimated from a population-based prior.
6. Suggest your own topic

- Propose the topic within your research interest
- Review with instructor the computational / statistical requirements to be implemented and set the goal for the class project
- Get it done; get a good grade; and write your paper!
C++ is a flexible language

- C++ offers both reference and pointer types
  - C does not support reference type
  - Java supports only reference type for user-defined objects
- C++ offers abstraction through user-defined data type (unlike C, like Java)
- Inheritance and dynamic polymorphism (unlike C)
- Explicitly memory management (unlike Java)
- Templates that operate with generic types (unlike C or earlier Java)
C++ can be complicated to learn

- An anecdote
- Bjarne Stroustrup revealed a motivation to design C++ in an interview
- He said that, C language is too easy to distinguish talented programmers from ordinary programmers.
- He also said that, he designed C++ language mainly to created high-paying jobs for talented programmers.
- The story above was turned out to be a hoax
- But many people still think this is a true story, because it was believeable, suggesting that C++ does appear that much more complex than C.
- Let’s keep it simple in this class
  - We want to leverage the flexibility of C++
  - But we don’t want to suffer from the complexities
  - So this class will selectively cover C++ specific features
Why using 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 using 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.
Classes and user-defined data type

C++ Class

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

An example C++ Class

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.;
# 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
    return 0;
}
```
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
    return 0;
}
```
Recap
815 Projects
C++
Classes
STL
InsertionSort
Recursion
Gcd
Divide and Conquer

Recap
815 Projects
C++
Classes
STL
InsertionSort
Recursion
Gcd
Divide and Conquer

Hyun Min Kang
Biostatistics 615/815 - Lecture 4
September 15th, 2011
19 / 43

### Built-in data types also have constructors

```cpp
#include <iostream>

int main(int argc, char** argv) {
    int a;   // declare a first - value of a is undefined
    a = 1;   // assign the value of a
    int b = 2; // declare and assign simultaneously
    int c(3); // call constructor when declaring variable

    std::cout << (a == 1) << std::endl; // true
    std::cout << (b == 2) << std::endl; // true
    std::cout << (c == 3) << std::endl; // true
}
```
Constructor calls constructors

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

class Point {
public:
    double x;
    double y;
    // A constructor can call constructor of each member variable
    Point(double px, double py) : px(x), py(y) {}  
    // 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
    return 0;
}
```
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
    return 0;
}
```
More class examples - pointRect.cpp

class Points { ... }; // assumes that Point is defined here
class Rectangle { // Rectangle

public:
    Point p1, p2; // rectangle defined by two points
    // Constructor 1: initialize by calling constructors of member variables
    Rectangle(double x1, double y1, double x2, double y2) : p1(x1,y1), p2(x2,y2) {}
    // Constructor 2: from two existing points
    // Passing user-defined data types by reference avoid the overhead of creating new objects
    Rectangle(Point& a, Point& b) : p1(a), p2(b) {}

    double area() { // area covered by a rectangle
        return (p1.x-p2.x)*(p1.y-p2.y);
    }
};
Initializing objects with different constructors

```cpp
int main(int argc, char** argv) {
    Point p1(3,4), p2(15,9);    // initialize points
    Rectangle r1(3,4,15,9);     // constructor 1 is called
    Rectangle r2(p1,p2);        // constructor 2 is called
    std::cout << r1.area() << std::endl; // prints 60
    std::cout << r2.area() << std::endl; // prints 60
    std::cout << r1.p2.print() << std::endl; // prints (15,9)
    return 0;
}
```
#include <iostream>
#include <cmath>

class Point { ... }; // same as defined before

int main(int argc, char** argv) {
    // allocation to "stack" : p1 is alive within the function
    Point p1(3,4);
    // allocation to "heap" : *pp2 is alive until delete is called
    Point* pp2 = new Point(5,12);
    Point* pp3 = &p1; // pp3 is simply the address of p1 object
    p1.print();    // Member function access - prints (3,4)
    pp2->print();  // Member function access via pointer - prints (5,12)
    pp3->print();  // Member function access via pointer - prints (3,4)
    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
    return 0;
}
Summary : Classes

- Class is an abstract data type
- A class object may contain member variables and functions
- Constructor is a special class for initializing a class object
  - There are also destructors, but not explained today
  - The concepts of default constructor and copy constructor are also skipped
- `new` and `delete` operators to dynamically allocate the memory in the heap space.
// assume that Point class defined above
Point* foo(double x, double y) {
    Point p(x,y); // local variable in stack space. valid only within a function
    return &p; // WARNING: return value is invalid if function terminates
}
Point* bar(double x, double y) {
    Point* p = new Point(x,y); // heap spaces
    return p; // object is alive until delete is called
}
int main(int argc, char** argv) {
    Point* p1 = foo(3,4); // p1 is invalid after foo() is terminated.
    Point* p2 = bar(5,12); // p2 is a valid pointer
    p1->print(); // prints arbitrary value (may cause fatal error)
    p2->print(); // prints (5,12)
    delete p2; // object created by 'new' must be 'delete'ed.
    return 0;
}
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>`
STL in practice

**sortedEcho.cpp**

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

int main(int argc, char** argv) {
    std::vector<std::string> vArgs; // vector of strings
    for(int i=1; i < argc; ++i) {
        vArgs.push_back(argv[i]); // append each arguments to the vector
    }
    std::sort(vArgs.begin(),vArgs.end()); // sort the vector in alphanumerical order
    std::cout << "Sorted arguments :"; // print the sorted arguments
    for(int i=0; i < vArgs.size(); ++i) { std::cout << " " << vArgs[i]; }
    return 0;
}
```

**A running example**

```
user@host:~/ > ./sortedEcho Hello, World! hello, world! 2 3 5 60 1
Sorted arguments : 1 2 3 5 60 Hello, World! hello, world!
```
If you’re tired of typing `std::...

```cpp
// exactly do the same thing to sortedEcho.cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std; // std::[classname] will be searched for
int main(int argc, char** argv) {
    vector<string> vArgs; // vector of strings
    for(int i=1; i < argc; ++i) {
        vArgs.push_back(argv[i]); // append each arguments to the vector
    }
    sort(vArgs.begin(),vArgs.end()); // sort the vector in alphanumeric order
    cout << "Sorted arguments: "; // print the sorted arguments
    for(int i=0; i < vArgs.size(); ++i) { cout << " " << vArgs[i]; }  
    cout << endl;
    return 0;
}
```
Using STL strings

```cpp
int main(int argc, char** argv) {
    char* p = "Hello pointer";       // array of characters
    char* q = p;                     // q and p point to the same address
    p[0] = 'h';
    std::cout << p << std::endl;     // "hello string"
    std::cout << q << std::endl;     // "hello string"

    std::string s("Hello string"); // STL string
    std::string t = s;              // clones the entire string
    t[0] = 'h';
    std::cout << t << std::endl;     // "hello string"
    std::cout << s << std::endl;     // "Hello string": s isn't changed

    // Below are possible with std::string, but not with char*
    s += ", you are flexible";      // s becomes "Hello string, you are flexible"
    t = s.substr(6,6);              // t becomes "string"
    return 0;
}
```
Using STL vectors

```cpp
int main(int argc, char** argv) {
    int A[] = {3,6,8};  // the array size is fixed
    int* p = A;  // p and A points to the same array
    p[0] = 10;
    std::cout << ( A[0] == 3 ) << std::endl; // false, not any more

    std::vector<int> v;  // vector is a variable-size array
    v.push_back(3);  // v.size() == 1
    v.push_back(6);  // v.size() == 2
    v.push_back(8);  // v.size() == 3
    std::vector<int> u = v;
    u[0] = 10;
    std::cout << ( v[0] == 3 ) << std::endl; // true
    return 0;
}
```
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
    $i = i - 1$;
  end
  $A[i + 1] = key$;
end
insertionSort.cpp - User Interface

insertionSort.cpp - main() function

```cpp
int main(int argc, char** argv) {
    std::vector<int> v; // contains array of unsorted/sorted values
    int tok; // temporary value to take integer input
    // read a series of input values from keyboard
    while ( std::cin >> tok ) { v.push_back(tok); }
    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;
}
```

How to feed input values

- By keyword - type [input value]+[RET] per each input entry, and put Ctrl+D when finished
STL Use in `INSERTION_SORT` Algorithm

```cpp
// print each element of array to the standard output
void printArray(std::vector<int>& A) {
    for(int i=0; i < A.size(); ++i) {
        std::cout << " " << A[i];
    }
    std::cout << std::endl;
}
```

Hyun Min Kang

Biostatistics 615/815 - Lecture 4

September 15th, 2011
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
    }
}
```

Hyun Min Kang

Biostatistics 615/815 - Lecture 4

September 15th, 2011

35 / 43
Recursion

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)
Example of recursion

Factorial

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

towerOfHanoi

```c++
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
        // 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
    }
}
```
Euclid’s algorithm

Algorithm \texttt{GCD}

\textbf{Data:} Two integers \(a\) and \(b\)

\textbf{Result:} The greatest common divisor (GCD) between \(a\) and \(b\)

\begin{verbatim}
if \(a \) divides \(b\) then
  return \(a\)
else
  Find the largest integer \(t\) such that \(at + r = b\);
  return \(\text{GCD}(r, a)\)
end
\end{verbatim}

Function \texttt{gcd()}

\begin{verbatim}
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 );
}
\end{verbatim}
A running example of Euclid’s algorithm

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 );
}
```

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 a[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);
}
Recursive Maximum

```cpp
// find maximum within an a[start..end]
int findMax(std::vector<int>& a, int start, int end) {
    if ( start == end ) return a[start];  // conquer small problem directly
    else {
        int mid = (start+end)/2;
        int leftMax = findMax(a,start,mid);  // divide the problem into half
        int rightMax = findMax(a,mid+1,end);
        return ( leftMax > rightMax ? leftMax : rightMax );  // combine solutions
    }
}
```
Next Lectures

- Sorting Algorithms
  - Merge Sort
  - Quicksort
  - Radixsort