Lab Session 01: Computing Weighted Average

Shahram Rahatlou

SAPIENZA
UNIVERSITÀ DI ROMA

http://www.roma1.infn.it/people/rahatlou/programmazione++/

Corso di Programmazione++

Roma, 31 March 2009

First Exercise

- Application to compute weighted average and error
- Application must accept an arbitrary number of input data
- Each data has a central value x and error(s)
- The error can be either symmetric ($x \pm \sigma$) or asymmetric ($x^{+\sigma 1}_{-\sigma 2}$)
- Compute the weighted average of input data and the error on the average

Scope of the Exercise

- Learn to write first application in C++ with real use case
- First encounter with use of external classes: vector
  - Use **`vector`** class from STL instead of arrays
    - http://www.sgi.com/tech/stl/Vector.html
- Suggestions on possible approaches
  - Initial menu to select averaging method
  - Provide capability of reading data from file
    - fstream and sstream

Possible extensions

- Provide different averaging methods
- Consider also systematic errors
- Accept correlated and uncorrelated errors and take into account when computing the average and error
- Make histogram of data points and average using root libraries

Use of Arrays

```
// vect1.cc
#include <iostream> // needed for input/output
#include <cmath>    // needed for math functions

int main() {

   using namespace std;

   float vect[3]; // no initialization

   cout << "printing garbage since vector not initialized" << endl;
   for(int i=0; i<3; ++i) {
     cout << "vect[" << i << "] = " << vect[i]
          << endl;
   }

   vect[0] = 1.1;
   vect[1] = 20.132;
   vect[2] = 12.66;

   cout << "print vector after setting values" << endl;
   for(int i=0; i<3; ++i) {
     cout << "vect[" << i << "] =    " << vect[i] << "\t"
          << "sqrt( vect[" << i << "] ) = " << sqrt(vect[i])
          << endl;
   }

   return 0;
}
```

Use of `std::vector`

```
// simple example of using class vector vector1.cc
#include <iostream>  // standard I/O
#include <cmath>  // math functions
#include <vector> // header for vector class

using std::vector;

int main() {

   vector<float> v1; // declare vector

   bool condition = true;

   cout << "insert -9999. to end data taking" << endl;
   while(condition) {
     float x = 0.0;
     cin >> x;
     if(x == -9999.) condition = false;
      if( condition != false ) v1.push_back( x );
   }
   // access size of vector!
   cout << "You inserted " << v1.size() << " values." << endl;


   // access individual elements of vector
   for(int i=0; i< v1.size(); ++i) {
     cout << "v1[" << i << "] = " << v1[i] << endl;
   }

   // use of iterator on elements of vector
   for(vector<float>::const_iterator it = v1.begin(); it != v1.end(); it++) {
     cout << "*it = " << *it << endl;
   }



   return 0;
}
```

If don't want to type std::vector every time!

Declare a new vector

Inserting new values

Length of vectors and access to i-th element

Possible Implementation of Our Exercise with Classes

```
// app1.cc
#include <vector>

class Datum; // basic data object
class InputService; // class dedicated to handle input of data
class Calculator; // impelments various algorithms
class Result; // how is Result different from Datum ?

int main() {

  InputService input;
  std::vector<Datum> dati = input.readDataFromUser();

  Calculator calc;
  calc.setData( dati );

  Result r1 = calc.weightedAverage();
  Result r2 = calc.arithmeticAverage();
  Result r3 = calc.geometricAverage();
  Result r3 = calc.fancyAverage();

  r1.display();

  return 0;
}
```

This code does not compile.

What is missing? ☺

Interface of Classes for Weighted Average

```
#ifndef Calculator_h
#define Calculator_h

#include <vector>
#include "Datum.h"
#include "Result.h"

class Calculator {
 public:
  Calculator();
  void setData(std::vector<Datum>& data);

  Result weightedAverage();
  Result arithmeticAverage();
  Result geometricAverage();
  Result fancyAverage();

private:
  std::vector<Datum> data_;
};
#endif
```

```
#ifndef Result_h
#define Result_h
class Result {
  public:
    Result();
    Result(double x, double y);
    Result(const Result& Result);
    double mean() { return mean_; }
    double stdDev() { return stdDev_; }
    double significance();
    void display();
  private:
    double mean_;
    double stdDev_;
};
#endif
```

You see the interface but don't know how the methods are implemented!

```
#ifndef InputService_h
#define InputService_h
#include <vector>
#include "Datum.h"

class InputService {
 public:
   InputService();
   std::vector<Datum> readDataFromUser();
 private:
};
#endif
```

```
#ifndef Datum_h
#define Datum_h
// Datum.h
#include <iostream>
using namespace std;

class Datum {
  public:
    Datum();
    Datum(double x, double y);
    Datum(const Datum& datum);
    double value() { return value_; }
    double error() { return error_; }
    double significance();
  private:
    double value_;
    double error_;
};
#endi
```

Application for Weighted Average

```
// wgtavg.cpp
#include <vector>

#include “Datum.h” // data objects
#include “InputService.h” // class dedicated to handle input of data
#inlcude “Calculator.h” // implements various algorithms
#include “Result.h” // how is Result different from Datum ?

int main() {

  InputService input;
  std::vector<Datum> dati = input.readDataFromUser();

  Calculator calc;
  calc.setData( dati );

  Result r1 = calc.weightedAverage();
  Result r2 = calc.arithmeticAverage();
  Result r3 = calc.geometricAverage();
  Result r3 = calc.fancyAverage();

  r1.display();

  return 0;
}
```

```
$ g++ -c InputService.cc
$ g++ -c Datum.cc
$ g++ -c InputService.cc
$ g++ -c Calculator.cc
$ g++ -c Result.cc
$ g++ -o wgtavg wgtavg.cpp InputService.o Datum.o Result.o Calculator.o
```

Today's Lab Session

- Use these 4 classes to write your application
- Take the interface and implement the functions
- Encapsulate your algorithms into methods of Calculator
- Adapt the exchange of data between functions to use Datum and Result classes