CS1020E: DATA STRUCTURES AND ALGORITHMS I

Tutorial 3 – Template, String, Streams, Vector, Iterator

(Week 5, starting 5 September 2016)

1. Template Class, New Data Structures

You are given a template Pair<TL, TR> class. Each object of this class can point to 2 objects of different types:

```
template <typename TL, typename TR>
class Pair {
     TL* _objLeft; TR* _objRight;
  public:
     Pair(TL* pobjLeft, TR* pobjRight) :
          _objLeft(pobjLeft), _objRight(pobjRight) {}
     TL* getLeft() { return _objLeft; }
     TR* getRight() { return _objRight; }
};
```

We now want to create a TemplateTriple<TL, TM, TR> class. Each object of this class can **point to 3 objects** of different types. **Restriction** for each of parts (a) - (d): Your class should have only **ONE member variable**, and you should be using the Pair class where possible.

There are 2 different ways to achieve this:

(a) Use inheritance.
(b) Use composition. TemplateTriple is composed of a Pair, i.e. it has a Pair object as a member variable. [Hint: How do you point to 3 objects in a Pair? Use 2 Pair objects, but only as a member variable]

We can now instantiate a TemplateTriple object that points to 3 objects. A Person has a name (string), weight (double), and height (double). Create a Person data structure and use a TemplateTriple to help you store data:

(c) Use inheritance. [Hint: Are you inheriting a family of classes, or just one specific type?]
(d) Use composition. Person is composed of a TemplateTriple.

A **Person** object should have 3 getters, one for each attribute. Each **getter** should return the **value** of the name/weight/height itself, and NOT a pointer to the value.

2. *STL Vector and Iterator*

A logistics company uses RFID tags to track the movement of hundreds of thousands of pallets. As pallets arrive, they pass through a scanner, and the pallet ID is added to the end of an STL `vector<string>` called `pallets`.

e.g. pallets [“20-0314”, “20-A921”, “20-A921”, “20-A921”, “20-A921”, “01-0003”, ”D9-3210” …]

Quite often, the same pallet is read repeatedly and consecutively, due to incorrectly configured hardware. We need to remove all consecutive (side-by-side) repeated pallet IDs from the vector `pallets`.

```
void cleanUp(vector<string>& pallets) { // why the & ?
     /* your code here */
}
```

(a) Use a single loop over `pallets`, directly removing the undesired elements one at a time
(b) Do the same as (a), this time using ONLY STL iterators instead of indexes
(c) Can you see that the algorithm in (a) & (b) is inefficient, even though there is just one loop? How do we improve?

3. *String, Streams*

You are interested in finding out the volume and weight of some products. Each product record contains (**product ID**, ☺garbage☺, **volume** in mm^3, **weight** in grams) in that order.

The following are examples of records, all valid:

- **1234567**:Wheel bearing|Yamaha XJ900s|Front:**9000 50**
- 00**900**#acm327df2mm3d1f0#Carburetor needle;Honda CB400;4 pcs;**8 5**
- 000000,Oil filter,Yamaha,3FV-13440-00,**225000 200**

As the data comes from various sources, the delimiter between various parts of the data may be any one of {',', ‘:’, ‘;’, ‘|’, ‘#’}. The **product ID** is guaranteed to be a non-negative integer, while the (**volume weight**) part is guaranteed to be the only data after the last delimiter.

The above 3 records should be **formatted** as:

```
| 1234567|    9000|   50|
|     900|       8|    5|
|       0|  225000|  200|
```

← Each line is one record

[Questions on next page...]

(a) Complete the implementation of the two methods in the given class:

```
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

class Product {
     long _productID; // any non-negative int is a valid ID
     long _volume; // in cubic mm
     long _weight; // in grams
  public:
     Product(string pInput) { ... } // parse 1 record - set member vars
     string str() { ... } // return the nicely formatted record

     long getProductID() { return _productID; }
     long getVolume() { return _volume; }
     long getWeight() { return _weight; }
};
```

Tip: Check out functions of <string> to help with parsing, that of <iomanip> to help with formatting

(b) Besides returning a formatted string through format(), how can we allow the formatted representation of a Product object to be easily printed?

i.e. How do we enable `cout << someProduct << endl;` to work?

- Learn how to learn ☺ -

Explore std library
Test its functions
Code incrementally