CS1020E Tutorial + Lab 03

Mark NG

a0116298@u.nus.edu
http://mollymr305.github.io

September 9, 2016

Question 1.

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; }
};
```

Question 1: (a).

Create a `TemplateTriple<TL, TM, TR>` class using inheritance.

```
template <typename TL, typename TM, typename TR>

class TemplateTripleInh : public Pair <TL, TR> {
  TM* _objMid;

public:
  TemplateTripleInh(TL* pobjLeft, TM* pobjMid, TR* pobjRight)
  : Pair<TL, TR>(pobjLeft, pobjRight), _objMid(pobjMid) {}

  TM* getMid() { return _objMid; }
};
```

Question 1: (b).

TemplateTriple<TL, TM, TR> class using composition.

```
template <typename TL, typename TM, typename TR>

class TemplateTripleComp {
  Pair<TL, Pair<TM,TR> > _objPair;

public:
  TemplateTripleComp(TL* pobjLeft, TM* pobjMid, TR* pobjRight)
  : _objPair(pobjLeft, new Pair<TM, TR>(pobjMid, pobjRight)) {}

  ~TemplateTripleComp() {
    delete _objPair.getRight();
  }

  TL* getLeft() { return _objPair.getLeft(); }

  TM* getMid() { return _objPair.getRight()->getLeft(); }

  TR* getRight() { return _objPair.getRight()->getRight(); }
};
```

Question 1: (c).

Person data structure using inheritance.

```
class PersonInh : public
TemplateTriple <string, double, double> {
public:
  PersonInh(string pstrName, double pdblWt, double pdblHt)
  : TemplateTriple(new string(pstrName),
    new double(pdblWt), new double(pdblHt)) {}

  ~PersonInh() {
    delete getLeft(); delete getMid(); delete getRight();
  }

  string getName() { return *getLeft(); }

  double getWt() { return *getMid(); }

  double getHt() { return *getRight(); }
};
```

Question 1: (c).

Person data structure using composition.

```
class PersonComp {
  TemplateTriple<string, double, double> _objTriple;

public:
  PersonComp(string pstrName, double pdblWt, double pdblHt) :
  _objTriple(new string(pstrName),
    new double(pdblWt), new double(pdblHt)) {}

  ~PersonComp() {
    delete _objTriple.getLeft();
    delete _objTriple.getMid();
    delete _objTriple.getRight();
  }

  string getName() { return *_objTriple.getLeft(); }

  double getWt() { return *_objTriple.getMid(); }

  double getHt() { return *_objTriple.getRight(); }
};
```

Question 2.

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 */

}
```

Question 2 (a).

Use a single loop over pallets, directly removing the undesired elements one at a time

```
void cleanUp(vector<string>& pallets) {
  int idx = 1;
  while (idx < pallets.size()) {
    if (pallets.at(idx - 1) == pallets.at(idx))
      pallets.erase(pallets.begin() + idx);
    else
      idx++;
  }
}
```

Question 2 (b).

Same as (a), this time using ONLY STL iterators instead of indexes.

```
void cleanUp(vector<string>& pallets) {
  if (pallets.size() < 2) return;
  vector<string>::iterator prevItr = pallets.begin();
  vector<string>::iterator currItr = pallets.begin() + 1;
  while (currItr < pallets.end()) {
    if (*prevItr == *currItr) {
      currItr = pallets.erase(currItr);
    } else {
      prevItr++;
      currItr++;
    }
  }
}
```

Question 2 (c).

Improvement to (a) and (b).

```
void cleanUp(vector<string>& pallets) {
  if (pallets.size() < 2) return;
  vector<string> buffer;
  buffer.push_back(pallets.front());
  for (vector<string>::iterator itr = pallets.begin() + 1;
       itr < pallets.end(); itr++) {
    if (buffer.back() != *itr) {
      buffer.push_back(*itr);
    }
  }
  pallets.swap(buffer);
}
```

Question 3 (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 one record
  string str() { ... } // return the formatted record
  long getProductID() { return _productID; }
  long getVolume() { return _volume; }
  long getWeight() { return _weight; }
};
```

Question 3 (a).

Solution (first part).

```
Product(string pInput) { // parse one record
  int firstDelim = pInput.find_first_of(",:;|#");
  int lastDelim = pInput.find_last_of(",:;|#");
  int lastSpace = pInput.rfind(" ");
  string prodID = pInput.substr(0, firstDelim);
  string vol = pInput.substr(lastDelim+1, lastSpace-lastDelim);
  string wt = pInput.substr(lastSpace+1, string::npos);

  (istringstream(prodID)) >> _productID;
  (istringstream(vol)) >> _volume;
  (istringstream(wt)) >> _weight;
}
```

Question 3 (a).

Solution (second part).

```
string str() { // return the formatted record
  ostringstream oss;
  oss << "|" << setw(8) << _productID
      << "|" << setw(7) << _volume
      << "|" << setw(4) << _weight << "|";
  return oss.str();
}
```

Question 3 (b).

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

```
// for example, the following should work
int main () {
  Product someProduct(
    "1234567:Wheel bearing|Yamaha XJ900s|Front:9000 50"
    );

  cout << someProduct << endl; // referring to this part
}
```

Question 3 (b).

Solution: operator overloading.

```
// this is done outside the class
ostream& operator<<(ostream& os, Product& prod) {
  os << prod.str();
  return os;
}
// now the following should work
int main () {
  Product someProduct(
    "1234567:Wheel bearing|Yamaha XJ900s|Front:9000 50"
    );

  cout << someProduct << endl; // referring to this part
}
```

End of Tutorial Discussion

Note: Detailed solutions (i.e. the file `T3_ans.pdf`) will be released soon at

`http://www.comp.nus.edu.sg/~stevenha/cs1020e.html`

Let's take a short break!

Preparation for Practical Exam

Problem

`https://open.kattis.com/problems/simonsays`

OOP Idea: Matrices?

2×2 `Matrix` class which supports operations $+, -, \times$.

Exercise 1 & 2

Some comments and tips...

- This is a rather tedious lab assignment. Watch out for bugs (e.g. output formatting, corner cases)!
- Only supposed to submit `distributor.cpp`, `product.cpp` and `store.cpp`. No modification of header files allowed.
- Read the sample test data if you are unsure of the problem description/requirements. This should give you a better understanding of the problem as well.

Any Questions?

See you next week!