OOP Homework Assignment № 1

Frank (Shuxin) Chen, Tsinghua University 08/22/2016

Problem 1

For this problem, we need two constructors, one for default construction and another for overloaded construction. Since the problem requires the information be constants, we should use () operators instead of = operators (= is not defined for constants).

Listing 1: Construtors for class Member.

```
1  Member::Member() : name("?"), age(0)
2  Member::Member(std::string _name, int _age) : name(_name), age(_age)
```

Another key to this problem is the overload of operators like « and []. For the operator «, we should add friend property to it since the variables we output may be private. For the operator [], it is easier to implement. In addition, in order to query the age from the given name, I use std::map<std::string, int> to maintain the information. If there exists the member for the given name, it returns the member's age. Otherwise it returns "No Found!".

Listing 2: Overload of operator.

```
1  std::string MemberList::operator [] (const std::string &name)
2  {
3      auto iter = members.find(name);
4      if (iter == members.end())
5      {
6          return "No Found!";
7      }
8      else
9      {
10         int age = iter->second;
11         std::string ret = "";
12         while (age)
13         {
14             ret = static_cast<char>(age % 10 + 48) + ret;
15             age /= 10;
16         }
17         return ret;
18     }
19 }
```

Problem 2

This problem requires us to implement a virtual class Shape. Three actual classes inherit the virtual class. We should design the function getarea() in class Shape as a pure virtual function since the implements should be written in the classes which inherit it. For the constant value pi which is necessary for calculating the area of a circle, we can store this value in a static variable in class Circle. The code for class Shape is displayed as follows.

Listing 3: Code for class Shape.

```
class Shape // virtual
{
public:
    Shape();
    virtual ~Shape();

public:
    virtual double getarea() = 0;
};
```

Problem 3

This problem is relatively easy. The key to this problem is implementing a template. As the problem only requires comparisons between integers, doubles and strings, we can just use the > operator to compare them.

Listing 4: A simple template.

```
template<class T> T Max(const T& a, const T& b)
{
    if (a > b) return a;
    return b;
}
```