ClassTemplates

August 8, 2020

1 Class Templates

1.1 Table of Contents

1.2 header includes used in this notebook

[1]:
```
#include <iostream>
#include <string>

using namespace std;
```

1.3 Class Templates

- just like function templates, we can also create class templates
  - class templates allow to generalize data types of class members
- class templates let us define a different implementation for a template when a specific type is passed as template argument (template specialization)
- e.g., Rectangle class below only works on length and width values of different types!

[2]:
```
// Class Templates
// for template class, class interface and definition must be in
// the same file if header (.h) file is used!
template<class T>
class Rectangle {
    private:
        T length, width;
    public:
        // default and overloaded constructor
        Rectangle(T length=0, T width=0) {
            this->setLength(length);
            this->setWidth(width);
        };

        void setLength(T length) {
            if (length < 0)
                length = 0;
            this->length = length;
        }

        void setWidth(T width) {
            if (width == 0)
                width = 0;
            this->width = width;
        }

        T findArea() const {
            return this->length*this->width;
        }

        T getPerimeter() const {
            return 2*(this->length + this->width);
        }

        // overload + operator
        Rectangle<T> operator+(const Rectangle<T>& rhs) {
            Rectangle temp;
            temp.length = this->length + rhs.length;
            temp.width = this->width + rhs.width;
            return temp;
        }

        void print() {
            cout << "length = " << length;
            cout << " width = " << width;
            cout << " area= " << findArea();
        }
};
```

[3]:
```
Rectangle<int> r1 = {20, 10};
Rectangle<int> r2 = {10, 5};
```

[4]:
```
Rectangle<int> r = r1 + r2;
```

[5]:
```
r.print();
```

```
length = 30 width = 15 area= 450
```

[6]:
```
#include <iostream>
using namespace std;
```

[7]:
```
template<class T>
class MyPair {
    T values[2];
    public:
        MyPair(T first, T second) {
            values[0] = first;
            values[1] = second;
        }
        void print() {
            cout << "first: " << values[0] << " second: " << values[1] << endl;
        }
};
```

[8]:
```
MyPair<int> p1(10, 100);
MyPair<float> p2(10, 100.99);
MyPair<string> p3("hello", "john");
MyPair<char>p4('A', 'a');
```

[9]:
```
p1.print();
p2.print();
p3.print();
p4.print();
```

```
first: 10 second: 100
first: 10 second: 100.99
first: hello second: john
first: A second: a
```

[10]:
```
template<class T1, class T2>
class Pair {
    T1 first;
    T2 second;
    public:
        Pair(T1 first, T2 second) {
            this->first = first;
            this->second = second;
        }
        void print() {
            cout << "first: " << this->first << " second: " << this->second <<␣
 ↪endl;
        }
};
```

[11]:
```
Pair<int, int> p5(10, 100);
Pair<int, float> p6(10, 100.99);
Pair<string, double> p7("hello", 4.5);
Pair<char, unsigned int>p8('A', 200);
```

[12]:
```
p5.print();
p6.print();
```

```
p7.print();
p8.print();
```

```
first: 10 second: 100
first: 10 second: 100.99
first: hello second: 4.5
first: A second: 200
```

[ ]:

```
```