

Templates C++

Thursday, October 10, 2019 7:14 PM

1. There are 2 kinds of templates:

Function Templates	<p>1. Allows us to create a function template whose functionality can be adapted to more than one type without repeating the code for each type</p> <pre>6 template<class T> 7 T giveMax(T first, T second) { 8 9 if(first >= second) return first; 10 11 return second; 12 } 13 14 15 int main() { 16 17 int maxInt = giveMax(2, 10); 18 cout << "Max Integer = " << maxInt << endl; 19 20 21 double maxDouble = giveMax(5.00, 10.43); 22 cout << "Max Double = " << maxDouble << endl; 23 24 return 0; 25 }</pre> <p>2.</p> <pre>\$g++ -std=c++11 -o main *.cpp \$main Max Integer = 10 Max Double = 10.43</pre>
Class Templates	<p>1. We may even write class templates, where a class can have members that use template parameters as types</p>

a.

```

6  template<class T>
7  class myPair {
8
9      public:
10     T giveMax(T first, T second) {
11
12         if(first >= second) return first;
13
14         return second;
15     }
16
17 };
18
2. 19
20 int main() {
21
22
23     myPair<int> intPair;
24     int maxInt = intPair.giveMax(11, 10);
25     cout << "Max Integer = " << maxInt << endl;
26
27     myPair<double> doublePair;
28     double maxDouble = doublePair.giveMax(9.00, 12.43);
29     cout << "Max Double = " << maxDouble << endl;
30
31     return 0;
32 }

```

```
$g++ -std=c++11 -o main *.cpp
```

```
$main
```

```
Max Integer = 11
```

```
Max Double = 12.43
```

2. There are mainly 2 things to worry about in template writing:

Declaration	<pre>template<class T> class myPair {</pre>
Usage	<p>1. Every time we use the class, we have to specify the template parameters separately</p> <pre>1. myPair<int> intPair;</pre>

a.

2. Or, we can use something like typedef to avoid repetitive usages:

```
typedef myPair<int> IntPair;
```

3. Scope and writing a function outside of its class:

4.

```
41 template<class T>
42 int Stack<T>::size() {
43
44     return myVect.size();
45 }
```

3. Example Linked List code using Templates:

a. Linked List Node using templates:

b.

```
7  template <class T>
8  class Node{
9
10     public:
11
12         T value;
13         Node * next;
14
15         Node() {
16
17             next = nullptr;
18         }
19
20         Node(T val) {
21
22             value = val;
23             next = nullptr;
24         }
25 };
26
```

Using the above node as a template for further use in the Linked List

```

28 template<class U>
29 class LinkedList {
30
31     typedef Node<U> ListNode;
32
33     private:
34         ListNode * dummy;
35
36     public:
37
38         LinkedList() {
39
40             dummy = new ListNode();
41         }
42
43
44         void addNode(U val) {
45
46             ListNode * front = dummy->next;
47             ListNode * back = dummy;
48
49             while(front != nullptr) {
50
51                 front = front->next;
52                 back = back->next;
53             }
54
55             back->next = new ListNode(val);
56         }
57
58

```

```

59 - void removeNode(U val) {
60
61     ListNode * front = dummy->next;
62     ListNode * back = dummy;
63
64     while(front != nullptr) {
65
66         if(front->value == val) {
67
68             back->next = front->next;
69             delete front;
70             return;
71         }
72
73         front = front->next;
74         back = back->next;
75     }
76 }
77
78 void traverse() {
79
80     ListNode * front = dummy->next;
81     while(front != nullptr) {
82
83         cout << front->value << ", ";
84         front = front->next;
85     }
86
87     cout << endl;
88 }
89

```

```

90 ~LinkedList() {
91
92     ListNode * back = dummy;
93     ListNode * front = dummy->next;
94
95     while(front != nullptr) {
96
97         delete back;
98         back = front;
99         front = front->next;
100     }
101     delete back;
102 }
103
104 };
105

```

```

110     LinkedList<double> doubleList;
111     doubleList.addNode(1.00);
112     doubleList.addNode(2.32);
113     doubleList.addNode(3.22);
114     doubleList.traverse();
115     doubleList.removeNode(2.32);
116     doubleList.traverse();
117
118     LinkedList<int> intList;
119     intList.addNode(100);
120     intList.addNode(200);
121     intList.addNode(300);
122     intList.traverse();
123

```

```
$g++ -std=c++11 -o main *.cpp
```

```
$main
```

```

1, 2.32, 3.22,
1, 3.22,
100, 200, 300,

```

4. General Template Stack using Vectors:

```

6
7  template<class T>
8  class Stack {
9
10     private:
11         vector<T> myVect;
12
13     public:
14
15         void push(T val);
16         T pop();
17         T top();
18         int size();
19 };

```

```

21  template<class T>
22  void Stack<T>::push(T val) {
23
24      myVect.push_back(val);
25  }
26
27  template<class T>
28  T Stack<T>::pop() {
29
30      T val = myVect.back();
31      myVect.pop_back();
32      return val;
33  }
34
35  template<class T>
36  T Stack<T>::top() {
37
38      return myVect.back();
39  }
40
41  template<class T>
42  int Stack<T>::size() {
43
44      return myVect.size();
45  }

```

```

48  int main() {
49
50      Stack<string> myStack;
51      myStack.push("vaibhav");
52      myStack.push("raheja");
53      cout << "Top = " << myStack.top() << endl;
54
55      return 0;
56  }

```

```
$g++ -std=c++11 -o main *.cpp
```

```
$main
```

```
Top = raheja
```