

Lab 4 Solutions

2 Multi-Type min

2.1

```
1 template<typename T>
2 T min(const T t1, const T t2) {
3     return t1 < t2 ? t1 : t2;
4 }
```

2.2

```
1 #define min(x, y) (x < y ? x : y)
```

3 Casting

3.1

```
static_cast<Triangle *>(p)
    or
reinterpret_cast<Triangle *>(p)
```

3.2

```
dynamic_cast<Triangle *>(p)
```

4 Templated Stack

4.1

```
1 template<class T> class Stack;
2
3 template<class T>
4 Stack<T> operator+(const Stack<T> &s1, const Stack<T> &s2);
5
6 {
7     Stack<T> result = s1;
8
9     for(unsigned i = 0; i < s1.items.size(); ++i) {
10         result.items.push_back(s2.items[i]);
11     }
12
13     return result;
14 }
15
16 template<class T>
17 class Stack {
18     friend Stack<T> operator+<>(const Stack<T> &s1, const Stack<T> &
19         s2);
20     vector<T> items;
21 public:
22     bool empty() const {return items.empty();}
23     void push(const T &item) {items.push_back(item);}
24     T pop() {
25         T last = items.back();
26         items.pop_back();
27         return last;
28     }
29 };
30
31 template<class T>
32 Stack<T> operator+(const Stack<T> &s1, const Stack<T> &s2)
33 {
34     Stack<T> result = s1;
35
36     for(unsigned i = 0; i < s1.items.size(); ++i) {
37         result.items.push_back(s2.items[i]);
38     }
39
40     return result;
41 }
```

5 Graph Representation

```
1 class Graph {
2 protected:
3     map<int, vector<int> > outgoing;
4
5 public:
6     Graph(const vector<int> &startPoints, const vector<int> &
           endPoint);
7     int numOutgoing(const int nodeID) const;
8     const vector<int> &adjacent(const int nodeID) const;
9 };
10
11 // In a .cpp file...
12
13 #include <stdexcept>
14
15 Graph::Graph(const vector<int> &startPoints, const vector<int> &
           endPoint) {
16     if(startPoints.size() != endPoint.size()) {
17         throw invalid_argument("Start/end point lists differ in
           length");
18     }
19
20     for(unsigned i = 0; i < startPoints.size(); i++ ) {
21         int start = startPoints[i], end = endPoint[i];
22         outgoing[start].push_back(end);
23         outgoing[end]; // Just to indicate this node exists
24     }
25 }
26
27 int Graph::numOutgoing(const int nodeID) const {
28     return adjacent(nodeID).size();
29 }
30
31 const vector<int> &Graph::adjacent(const int nodeID) const {
32     map<int, vector<int> >::const_iterator i = outgoing.find(nodeID)
           ;
33     if(i == outgoing.end()) {
34         throw invalid_argument("Invalid node ID");
35     }
36     return i->second;
37 }
```

MIT OpenCourseWare

<http://ocw.mit.edu>

6.096 Introduction to C++

January (IAP) 2011

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.