Lab 4 Solutions

2 Multi-Type `min`

2.1

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

2.2

```
#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

```cpp
template<class T> class Stack;

template<class T>
Stack<T> operator+(const Stack<T> &s1, const Stack<T> &s2);

 {
    Stack<T> result = s1;

    for(unsigned i = 0; i < s1.items.size(); ++i) {
        result.items.push_back(s2.items[i]);
    }

    return result;
}

template<class T>
class Stack {
    friend Stack<T> operator+<>(const Stack<T> &s1, const Stack<T> &s2);
    vector<T> items;

public:
    bool empty() const {return items.empty();}
    void push(const T &item) {items.push_back(item);}
    T pop() {
        T last = items.back();
        items.pop_back();
        return last;
    }
};

template<class T>
Stack<T> operator+(const Stack<T> &s1, const Stack<T> &s2)
{
    Stack<T> result = s1;

    for(unsigned i = 0; i < s1.items.size(); ++i) {
        result.items.push_back(s2.items[i]);
    }

    return result;
}
```

5 Graph Representation

```cpp
class Graph {
protected:
    map<int, vector<int> > outgoing;

public:
    Graph(const vector<int> &startPoints, const vector<int> &endPoints);
    int numOutgoing(const int nodeID) const;
    const vector<int> &adjacent(const int nodeID) const;
};

// In a .cpp file...

#include <stdexcept>

Graph::Graph(const vector<int> &startPoints, const vector<int> &endPoints) {
    if(startPoints.size() != endPoints.size()) {
        throw invalid_argument("Start/end point lists differ in length");
    }

    for(unsigned i = 0; i < startPoints.size(); i++ ) {
        int start = startPoints[i], end = endPoints[i];
        outgoing[start].push_back(end);
        outgoing[end]; // Just to indicate this node exists
    }
}

int Graph::numOutgoing(const int nodeID) const {
    return adjacent(nodeID).size();
}

const vector<int> &Graph::adjacent(const int nodeID) const {
    map<int, vector<int> >::const_iterator i = outgoing.find(nodeID);
    if(i == outgoing.end()) {
        throw invalid_argument("Invalid node ID");
    }
    return i->second;
}
```