COMP6771 19T2 (https://webcms3.cse.unsw.edu.au/COMP6771/19T2)

Week 07 Tutorial Sample Answers

Advanced C++ Programming (https://webcms3.cse.unsw.edu.au/COMP6771/19T2)

1. Have you selected your group for ass3 yet?

2. Do you have any questions about the assignment?

3. How many instantiations of max are generated?

```
#include <iostream>

template <typename T>
T max(T a, T b) {
  return a < b ? b : a;
}

int main() {
  int result = 7;
  std::cout << max(1, 2) << "\n";
  std::cout << max(1.1, 2.2) << "\n";
  std::cout << max(1.0, 2.0) << "\n";
  std::cout << max('a', 'z') << "\n";
  std::cout << max(7, result) << "\n";
  std::cout << max("cat", "dog") << "\n";
}
```

4

4. What is wrong with this implementation?

```
#ifndef LECTURES_WEEK7_STACK_H_
#define LECTURES_WEEK7_STACK_H_

#include <vector>

template <typename T>
class Stack {
 public:
  void push(T);
  T pop();
 private:
  std::vector<T> stack_;
};

#endif  // LECTURES_WEEK7_STACK_H_
```

```
#include <stack.h>

template <typename T>
void Stack<T>::push(T t) {
  stack_.push_back(t);
}

template <typename T>
T Stack<T>::pop() {
  return stack_.pop_back();
}
```

```
#include <iostream>

#include "stack.h"

int main() {
  Stack<int> is1;
  is1.push(5);
  std::cout << is1.pop() << "\n";
}
```

Compiled with

```
g++ stack_use.cpp stack.cpp -o stack
```

Template definitions must be in the .h file because files using templates must be able to generate the code at compile time (as opposed to link time)

5. Convert the following class to a generic (templated) type.

```
#ifndef STRINGQUEUE_H_
#define STRINGQUEUE_H_

#include <string>
#include <memory>

class StringQueue {
 public:
  StringQueue(int size) : size_curr_{0} size_max_{size}, queue_{std::make_unique<std::string[]>(size)}
  void enque(std::string s);
  std::string deque();
  int size() const;
 private:
  int size_curr_;
  int size_max_;
  std::unique_ptr<std::string[]> queue_;
}

#endif  // STRINGQUEUE_H_
```

After converting it to a generic type:

- Complete all incomplete implementations.
- Add an appropriate default constructor
- Add an appropriate copy constructor

```
#ifndef STRINGQUEUE_H_
#define STRINGQUEUE_H_

#include <string>
#include <memory>
#include <stdexcept>

template <typename T>
class Queue {
 public:
  explicit Queue() : Queue(5);
  Queue(int size) : size_{size}, queue_{std::make_unique<T[]>(size)} {}
  Queue(const Queue& q) {
        size_max_ = q.size_max_;
        size_curr_ = q.size_curr_;
        queue_ = std::make_unique<T[]>(size_max_);
        for (int i = 0; i < size_curr_; ++i) {
          queue_[i] = q.queue_[i];
        }
  }
  void enque(T s);
  T deque();
  int size() const;
 private:
  int size_curr_;
  int size_max_;
  std::unique_ptr<T[]> queue_;
}

template <typename T>
void Queue<T>::enque(T s) {
  if (size_curr_ + 1 >= size_max_) {
        throw std::overflow_error{"Queue is full"};
  }
  queue_[size_curr_] = s;
  size_curr_ += 1;
}

template <typename T>
T Queue<T>::deque() {
  if (size_curr_ == 0) {
        throw std::logic_error{"No items in queue"};
  }
  T item = queue_[0];
  for (auto i = 1; i <= size_curr_; ++i) {
        queue_[i - 1] = queue_[i];
  }

  size_curr_--;
  return item;
}

template <typename T>
int Queue<T>::size() const {
  return size_curr_;
}

#endif  // STRINGQUEUE_H_
```

Finally, make the size of this queue a non-type parameter.

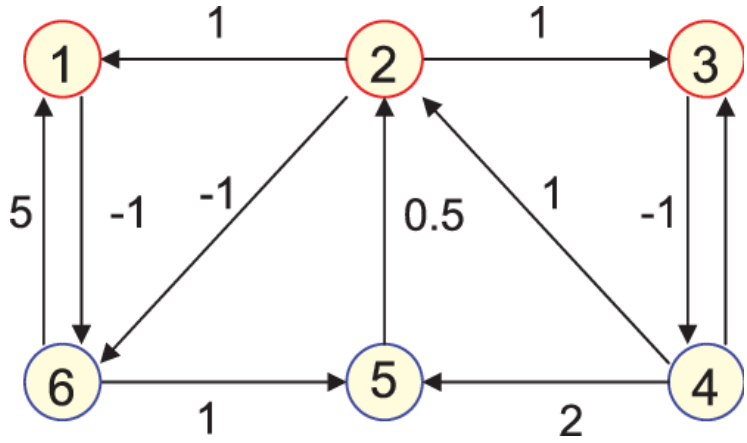
```
#ifndef STRINGQUEUE_H_
#define STRINGQUEUE_H_

#include <string>
#include <memory>
#include <stdexcept>

template <typename T, int max_size>
class Queue {
 public:
  explicit Queue() : queue_{std::make_unique<T[]>(max_size)} {}
  Queue(const Queue& q) {
        size_curr_ = q.size_curr_;
        queue_ = std::make_unique<T[]>(max_size);
        for (int i = 0; i < size_curr_; ++i) {
          queue_[i] = q.queue_[i];
        }
  }
  void enque(T s);
  T deque();
  int size() const;
 private:
  int size_curr_;
  std::unique_ptr<T[]> queue_;
}

template <typename T>
void Queue<T>::enque(T s) {
  if (size_curr_ + 1 >= max_size) {
        throw std::overflow_error{"Queue is full"};
  }
  queue_[size_curr_] = s;
  size_curr_ += 1;
}

template <typename T>
T Queue<T>::deque() {
  if (size_curr_ == 0) {
        throw std::logic_error{"No items in queue"};
  }
  T item = queue_[0];
  for (auto i = 1; i <= size_curr_; ++i) {
        queue_[i - 1] = queue_[i];
  }

  size_curr_--;
  return item;
}

template <typename T>
int Queue<T>::size() const {
  return size_curr_;
}

#endif  // STRINGQUEUE_H_
```

What are the advantages and disadvantages of the non-type parameter?

Advantage: Improved performance. Size of a queue is determined at compile time requiring less to be done / processed at runtime.

Disadvantage: (1) Code explosion - instantiation created for a queue of every size. (2) Unable to copy construct easily

6. Which part of this graph are:
   - Edges
   - Nodes
   - Weightings


1
2
3
6
5
4
1
1
5
-1
-1
0.5
1
-1
1
2