EECS 268 2021 Spring Midterm Exam

Rules:

- This exam is worth zero points
- You DO NOT submit this to anyone
- We will go over the exam in lecture

Conceptual

1. When does the type variable of a templated class (e.g. the T in template <typename T>) become well defined? Choose one.

A. When you include the header file in another file
B. It is never defined
C. When an object of that type is created (having its constructor called)
D. None of the above

Answer:	

2. Why does a node based implementation of a Stack's peek method need the option of throwing an exception, but isEmpty does not? Choose one.
    A. Because peek has a return type but isEmpty is a void function
    B. Because isEmpty does not traverse to the end of the stack
    C. Because peek is templated
    D. Because peek requires a value to exist to return but a stack's emptiness is always determinable

Answer:	

3. Fill in the blanks: A Queue is a _____ in _____ out data structure.

4. Fill in the blanks: A Stack is a _____ in _____ out data structure.

5. If a function named *func* declares an integer locally, when is that integer deallocated? (Choose one)
    A. When the function returns
    B. When main returns
    C. When delete is called on the variable
    D. None of the above

Answer:	

6. If a function named *func* declares a Circle* named *ptr*, when is that pointer (not anything it's pointing to, but the pointer) deallocated?
    A. When main returns
    B. When the function returns
    C. When delete is called on a pointer to the object
    D. None of the above

Answer:	

7. Why shouldn't getEntry be in charge of catching and handling the exception it throws? Choose one.
    A. Because the operating system catches exceptions, not our code
    B. Because if getEntry catches the exception it throws, it still needs to return a value. We need the throw to take control of the code back to the caller who passed in the bad parameters.
    C. Because Exceptions are automatically thrown when memory errors occur

Answer:	

8. What happens if a thrown exception is never handled? Choose one.
    A. Nothing
    B. Memory error
    C. Program ends due to unhandled exception

Answer:	

9. What is the difference between a deep copy and shallow copy of a data structure? Choose one.
    A. Deep copy will permanently save the information to file
    B. Shallow copy is stack allocated but deep copies are heap allocated
    C. Nothing
    D. Shallow copies set pointers to the same objects, but deep copies create separate copies.

Answer:	

10. By default, assignment operators and copy constructors create what kind of copy (deep or shallow)?
    A. Shallow
    B. Deep
    C. None of the Above

Answer:	

11. When is a copy constructor called?
    A. When you save an object to file
    B. When you declare an array of objects
    C. When an object is passed by value

Answer:	

12. The ability of a base type pointer or reference to run a method created in a derived type is called: (Choose one)
    A. Derivation
    B. Polymorphism
    C. Inheritance

Answer:	

13. A Queue always adds a new value where? (Choose one)
    A. Back
    B. Front
    C. Top

Answer:	

14. A Stack always puts a new value where?
    A. Front
    B. Top
    C. Back

Answer:	

Class Hierarchies

Below are the descriptions of how various classes are related to each other.

Descriptions:

- Shark is a Fish
- Corgi is a Dog
- Bird is an Animal
- Tabby is a Cat
- Dog is an Animal
- Duck is a Bird
- Cat is an Animal
- Fish is an Animal

Assume a class hierarchy of these classes is implemented. Now, answer the questions below regarding whether or not each operation would be legal or illegal.

Action	**Legal or illegal?**
A Shark pointer pointing to a Cat object	
A Corgi pointer pointing to a Dog object	
A function that takes Sharks being passed a Fish object	
An Animal pointer to an Animal object	
Having a Dog pointer to a Dog object	
Passing an Animal to a function that takes Dogs by reference	
Passing a Tabby to a function that takes Animals by reference	
Dog having its own definition for a method defined in the Animal	
Using a Fish object to call a Cat method	
Animal pointer pointing to a Shark object	

Code Writing

Assume you are in Queue.cpp. The Queue class has an m_front and m_back both of which are of type Node<T>* and are set to nullptr in the constructor. Below you will implement a modification of dequeue called *dequeueIf*. It works just like dequeue, but will only perform the dequeue if the value you are removing matches the value passed in. There are no other member variables, but you may assume all other Queue methods are implemented EXCEPT for dequeue (i.e. you may NOT call dequeue in this method). A std:runtime_error with the message "Invalid dequeue" should be thrown when trying to dequeueIf an empty queue or if the value passed in doesn't match.

```
template <typename T>
void Queue<T>::dequeueIf(T value) {
      //your code below
```

You may refer to this interface for the following coding question.

```
template <typename T>
class ListInterface
{
    public:
    virtual ~ListInterface() {}
    virtual int length() const = 0;

    /**
    * @pre The index is valid
    * @post The entry is added to the list at the index, increasing
length by 1
    * @param index, position to insert at (1 to length+1)
    * @param entry, value/object to add to the list
    * @throw std::runtime_error if the index is invalid
    */
    virtual void insert(int index, T entry) = 0;

    /**
    * @pre The index is valid
    * @post The entry at given position is removed, reducing length by 1
    * @param index, position to insert at (1 to length)
    * @throw std::runtime_error if the index is invalid
    */
    virtual void remove(int index) = 0;

    /**
    * @pre The index is valid
    * @post None
    * @param index, position to insert at (1 to length)
    * @throw std::runtime_error if the index is invalid
    */
    virtual T getEntry(int index) const = 0;

    /**
    * @post List is empty
    */
    virtual void clear() = 0;

    /**
    * @pre The index is valid (must already exist)
    * @post Given entry overrides the entry at the given index
    * @param index, position to override at (1 to length)
    * @param entry, value/object to place at given index
    * @throw std::runtime_error if the index is invalid
    */
    virtual void setEntry(int index, T entry) = 0;

};
```

Code Writing

Assume you are in main.cpp define a function that is NOT SCOPED to any class. It is NOT a member of the ListInterface nor LinkedList classes. Your function should accept two lists. You will remove all values from the first list that are present in the second list. Example:

Lists before call:	Lists after call:
target: A, B, C, A, D, C, E, F, B, C, A	target: A, A, D, E, F, A
toRemove: C, B, Q, Z	toRemove: C, B, Q, Z

```
template <typename T>
void removeHelper(ListInterface<T>& target, const ListInterface<T>& toRemove)
{
      //your code below
```