# COSC 220 - Computer Science II
# Project 3 - List, Stack, and Queue Libraries

Dr. Joe Anderson

Due: 12 May 2019

## 1 Description

You will develop generally-usable libraries that implement the List, Stack, and Queue functionality.

Recall that in Project 1, your task involved implementing the linked-list logic *twice* – once for Students, and once for Courses. This project demonstrates, due to the useful abstraction mechanisms available in C++, that was not necessary!

Your code will form not just a standalone program, but a *library* – a set of code that can be imported and used for a vast number of different applications, include the situations of both Project 1 and 2.

You will implement, in total, five classes: `SUList`, `SUQueueArr`, `SUQueueList`, `SUStackArr`, and `SUStackList`. As the names imply, they will represent a list, stacks and queues implemented with both arrays and linked lists. What makes them generally usable is that they will also be *template* classes, meaning the datatype stored in the list, stack, or queue is *completely up to the programmer using them.*

Lastly, this will optionally be a *group assignment.* You may choose to work with **one** other person on the project. If you do, however, be very sure to work closely together, and to *comment thoroughly*, including a `@author` segment to each subroutine.

## 2 Specifications

1. Your list class definition should follow:

```
template <class DataType>
class SUList{
 private:
   struct ListNode{     // The nodes of the list
    DataType data;      // The data stored in the node
    ListNode* next;     // The next node in the list
   }

   ListNode* head;      // The front of the list
   ListNode* tail;      // The last node of the list
 public:
   SUList();                        // Default ctor
   SUList(const SUList&);           // Copy ctor
   ~SUList();                       // Destructor
   DataType getFront();             // Remove & return the front
   DataType getBack();              // Remove & return the back
   void putFront(const DataType&);// Add to the front
   void putBack(const DataType&); // Add to the back
```

```
    int size() const;              // Returns the number of elements
    bool contains(const DataType&);// Tests for membership
    SUList<DataType>& operator=(const SUList<DataType>&); // Overloaded assignment
};
```

(a) The `putFront` and `putBack` functions should add *copies* of the parameter into the list. This is to ensure the List has full control of its data.

(b) The `contains` method only needs to determine if there is an element for which the `==` operator returns true, when compared with the parameter.

2. Put the `SUList` definition in `SUList.h` and the implementation in `SUList.cpp`.

3. Your stack class definitions should follow:

```
template <class DataType>
class SUStackXXX{
  private:
    /**
    * For the list-backed implementation:
    *
    * SUList<DataType> list;
    *
    * For the array-based:
    *
    * DataType* arr;        // The array of items
    * int       capacity;   // The size of the current array
    * int       top;        // The location of the top element
    */
  public:
    SUStackXXX();                    // Constructor
    SUStackXXX(const SUStackXXX &); // Copy Constructor
    ~SUStackXXX();                   // Destructor
    int size() const;                // get the number of elements in the stack
    bool isEmpty() const;            // Check if the stack is empty
    void push(const DataType&);      // Pushes an object onto the stack
    void pop(DataType&);             // Pop an object off the stack and store it
    void printStack() const;         // Prints the stack from the top, down
    SUStackXXX<DataType>& operator=(const SUStackXXX<DataType>&); // Assignment operator
}
```

where "XXX" Is replaced with either "Arr" or "List".

4. Place both the array and list definitions in a file called `SUStack.h` and all the implementation in a file called `SUStack.cpp`.

5. Your queue class definitions should follow:

```
template <class DataType>
class SUQueueXXX{
  private:
    /**
```

```
    * For the list-backed implementation:
    *
    * SUList<DataType> list;
    *
    * For the array-based:
    *
    * DataType* arr;        // The array of items
    * int       capacity;   // The size of the current array
    * int       front;      // The location of the front element
    * int       rear;       // The location of the rear element
    */
  public:
    SUQueueXXX();                    // Constructor
    SUQueueXXX(const SUQueueXXX &); // Copy Constructor
    ~SUQueueXXX();                   // Destructor
    int size() const;                // get the number of elements in the queue
    bool isEmpty() const;            // Check if the queue is empty
    void enqueue(const DataType&);  // Enqueues some data
    void dequeue(DataType&);         // Get the front element and store it
    void printQueue() const;         // Prints the queue from the front to the rear
    SUQueueXXX<DataType>& operator=(const SUQueueXXX<DataType>&); // Assignment operator
}
```

where "XXX" Is replaced with either "Arr" or "List".

6. Place both the array and list definitions in a file called `SUQueue.h` and all the implementation in a file called `SUQueue.cpp`.

7. You may include extra `private` members for convenience as you see fit, but be sure they work correctly and don't cause more bugs than they fix!

8. Any extra `public` members must be approved by the instructor.

9. Write a separate driver program to test your classes:

   (a) Use the `PayRoll` class formally specified in Lab 8 and make stacks and queues using this class as the data type. Write some tests to verify that they work as expected. Note that in order for the data to be copied into the stack or queue, one must have appropriate copy constructors, assignment operators, etc.

   (b) You may try adapting your code from Project 1 or Project 2 to use your new libraries! For example, the `HanoiStack` can now be simply replaced with `Stack<Disk>` type, with extra external functions to make sure one cannot push a larger disk onto a smaller one (pop the top disk, check if the desired push is valid; if yes, push them both. If not, push only the original one and display an error).

10. Include a `Readme` file that explains your approach to the problem, and how you solved the various issues. Be sure to mention whether you implemented the bonus features.

11. Submit a `Makefile` which compiles all the class code, as well as one or more driver programs. Make sure it functions normally if the driver programs are replaced with other, valid, test programs.

12. **Comments:**

    (a) Each class, subroutine, and class variable should have thorough documentation.

(b) Be sure to include:

i. What the inputs and outputs are.

ii. For class member functions: whether the object remembers reference arguments beyond the duration of the method call, and whether it will free them or not.

iii. If the function allocates memory that the caller must free.

iv. Whether any of the arguments can be a null pointer.

v. If there are any performance implications of how a function is used.

vi. The author of the described code.

# 3 Submission

Upload your project files (for this project only!) to the course canvas system.

If you are working as part of a group, you need to use the MyClasses system to assign yourself to a group.

Turn in (stapled) printouts of your source code, properly commented and formatted with your name, course number, and complete description of the code.

Also turn in printouts reflecting several different runs of your program (you can copy/past from the terminal). Be sure to test different situations, show how the program handles erroneous input and different edge cases.

# 4 Bonus

You may or may not add the following features for a maximum of 5 extra points each:

- Write overloaded *output* stream operators for all classes
- Write an overloaded `[]` operator for `SUList` – you may even find it useful to allow the user to traverse from the back by passing negative integers!
- Write a method to sort the `SUList` by using the comparison operator for `DataType`
- Write a small prototype of the re-implementation of Project1
  - You may omit any user-interaction – only write the `Student` and `StudentDB` classes to utilize your `SUList` instead of directly implementing the list logic. Then write a driver program to hard-code some possible user-interaction and show that it behaves as intended.
- Re-write project 2, replacing your `HanoiStack` with `SUStack<Disk>`.