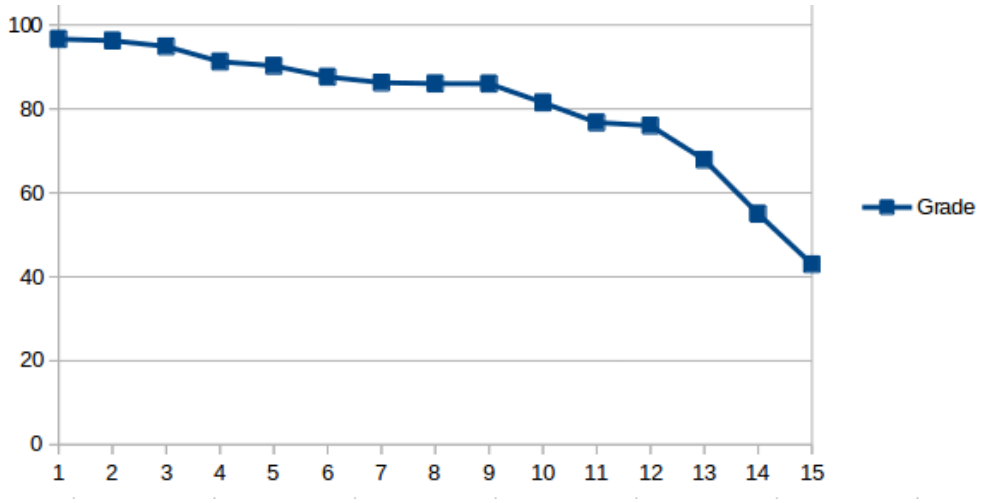
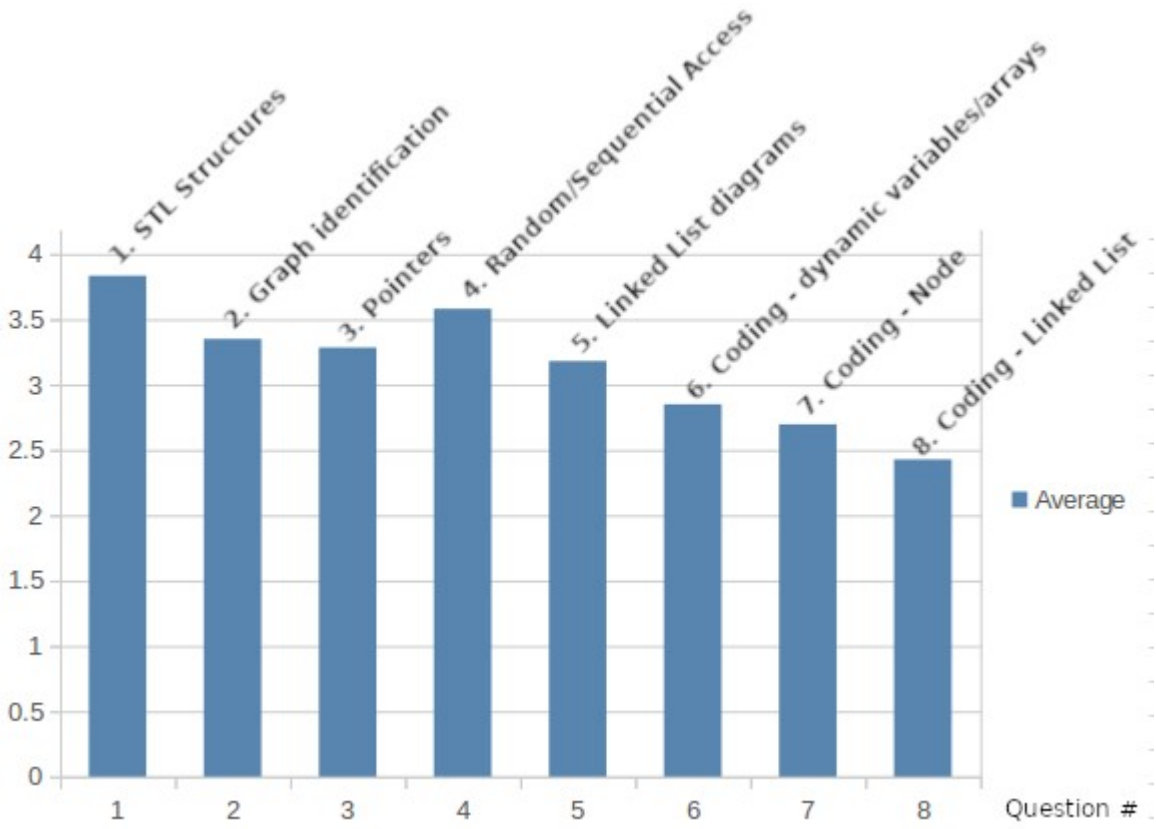
CS 250 Exam 1 stats, Fall 2018

Overall grades:



Question scores:



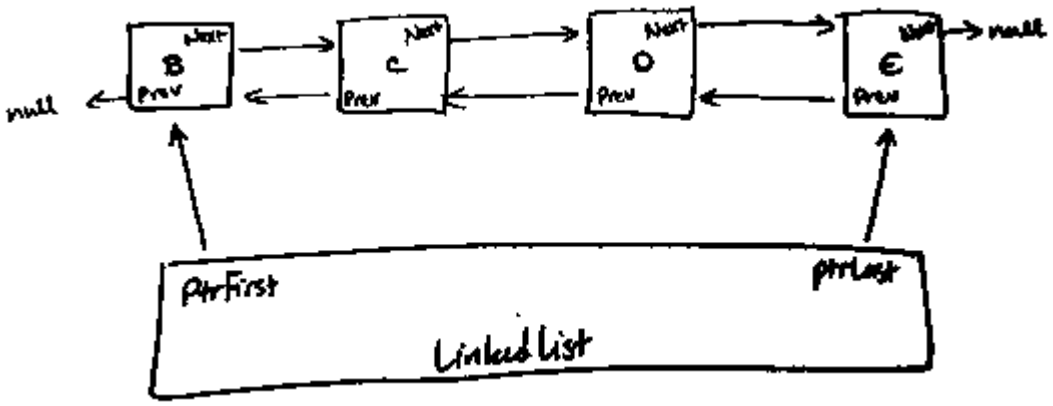
CS 250 Exam 1 stats, Fall 2018

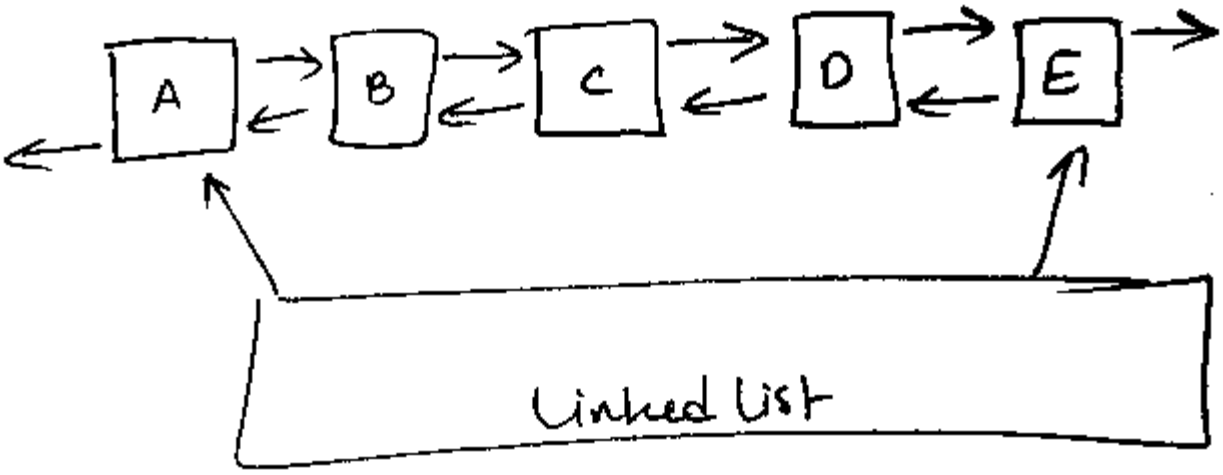
Solution:

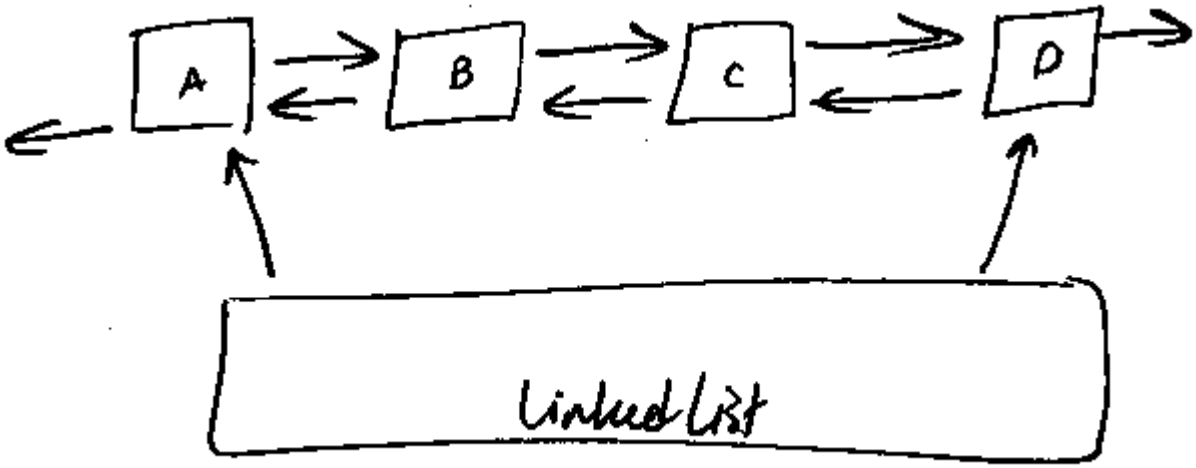
1.
   a. The STL vector is implemented as a dynamic array
   b. The STL List is implemented as a linked list
2. Figure 1 is a linked list, Figure 2 is a dynamic array
3.
   a. When a pointer is not being used, it should be initialized to nullptr.
      This is so that you can identify when a pointer is not currently being used, so that you don't accidentally de-reference a bad address, or so you know when you can overwrite the pointer's value.
   b. Pointers are initialized to garbage; an indeterminate, seemingly random value.
4. Array-based structures provide random access. Link-based structures require traversal.
5. See next pages
6. `Node * n;` and `int * arr;` are already declared…
   a. Allocate space for variable: `n = new Node;`
      Free space for n: `delete n;`
   b. Allocate space for array: `arr = new int[20];`
      Free space for arr: `delete [] arr;`
7. ```
   class Node {
    T data;    // Could put int, string, float, etc. for credit.
    Node* ptrNext;
    Node* ptrPrev;
   };
   ```
8. See next pages
   ```

CS 250 Exam 1 stats, Fall 2018

5.







CS 250 Exam 1 stats, Fall 2018

8.

```
template <typename T>
void LinkedList<T>::PushFront( T newData )
{
    Node<T>* newNode = new Node<T>;
    newNode->m_data = newData;

    if ( IsEmpty() )
    {
        m_ptrFirst = newNode;
        m_ptrLast = newNode;
    }
    else
    {
        newNode->m_ptrNext = m_ptrFirst;
        m_ptrFirst->m_ptrPrev = newNode;
        m_ptrFirst = newNode;
    }
    m_itemCount++;
}
```

```
template <typename T>
void LinkedList<T>::PopBack() noexcept
{
    if ( IsEmpty() )
    {
        // ignore
    }
    else if ( m_ptrFirst == m_ptrLast )
    {
        delete m_ptrFirst;
        m_ptrFirst = nullptr;
        m_ptrLast = nullptr;
        m_itemCount--;
    }
    else
    {
        Node<T>* ptrSecond = m_ptrLast->m_ptrPrev;
        ptrSecond->m_ptrNext = nullptr;
        delete m_ptrLast;
        m_ptrLast = ptrSecond;
        m_itemCount--;
    }
}
```

```
template <typename T>
T& LinkedList<T>::operator[]( const int index )
{
    if ( IsEmpty() )
    {
        throw out_of_range( "Cannot GET an item from an empty list!" );
    }
    else if ( IsInvalidIndex( index ) )
    {
        throw out_of_range( "Invalid index passed into [ ]" );
    }

    int counter = 0;
    Node<T>* current = m_ptrFirst;

    while ( current != nullptr && counter < index )
    {
        current = current->m_ptrNext;
        counter++;
    }

    return current->m_data;
}
```