# CSCI-1200 Data Structures — Spring 2023
# Lab 6 — Reversing Data 8 Ways:
# STL Vectors vs. STL Lists vs. Homemade Linked Lists

## Checkpoint 1A: Reverse with STL Vector Swaps *estimate: TBD*

Download this starter code:
http://www.cs.rpi.edu/academics/courses/spring23/csci1200/labs/06_lists_iterators/checkpoint1.cpp

Complete the function `reverse` that reverses the contents of an STL `vector` of integers. For example, if the contents of the vector are in increasing order before the call to `reverse_vector`, then they will be in decreasing order afterwards. For this checkpoint, use **indexing/subscripting/`[]`** on the vector, not iterators (or pointers). *You may not use a second vector or array or list.*

The trick is to step through the vector one location at a time, swapping values between the first half of the vector and the second half. As examples, the value at location 0 and the value at location `size()-1` must be swapped, and the value at location 1 and the value at location `size()-2` must be swapped.

Make sure your code works with even and odd length vectors. Also add a few more tests to the main function to make sure your code will work for the special cases of an empty vector and vectors of size 1 and 2.

## Checkpoint 1B: Reverse with STL List Swaps *estimate: TBD*

Copy your code from Checkpoint 1A to a new file. Then, convert this code to use STL Lists instead of STL vectors. Start by replacing '`vector`' with '`list`' everywhere. And you'll need to replace your subscripting with iterators.

You may want to use a straightforward concept we did not discuss in lecture: a reverse iterator. A reverse iterator is designed to step through a list from the back to the front. An example will make the main properties clear:

```
std::list<int> a;
unsigned int i;
for ( i=1; i<10; ++i ) a.push_back( i*i );

std::list<int>::reverse_iterator ri;
for( ri = a.rbegin(); ri != a.rend(); ++ri )
  cout << *ri << endl;
```

This code will print out the values $81, 64, 49, \ldots, 1$, in order, on separate lines. Observe the type for the reverse iterator, the use of the functions `rbegin` and `rend` to provide iterators that delimit the bounds on the reverse iterator, and the use of the `++` operator to take one step backwards through the list. It is very important to realize that `rbegin` and `end` are NOT the same thing! One of the challenges here will be determining when to stop (when you've reached the halfway point in the list). You may use an integer counter variable to help you do this.

For this checkpoint you should not use `erase`, or `insert`, or the `push` or `pop` functions.

Note, you'll probably need to add the keyword `typename` in front of your templated iterator types to unconfuse the compiler.

```
typename std::list<T>::iterator itr = data.begin();
```

**To complete this checkpoint,** show a TA your debugged functions to reverse STL vectors and STL lists by element swapping. Be sure to ask your TA/mentors any questions you have about regular vs. reverse iterators for lists and vectors.

### Checkpoint 2: Reverse with STL List Using Insert/Erase/Push/Pop *estimate: TBD*

Form a team of 4. *You may form a team of 2 or 3 only with approval from your graduate lab TA.*

Each student should make a copy of their solution file for Checkpoint 1B. And then, each student should rewrite their STL list `reverse` function:

- **STUDENT 1** Using only `front()`, `pop_front()`, and `insert()`.
- **STUDENT 2** Using only `back()`, `pop_back()`, `insert()`, and iterator increment.
- **STUDENT 3** Using only `erase()`, `push_front()`, and iterator dereference.
- **STUDENT 4** Using only `erase()`, `push_back()`, iterator dereference, and iterator decrement.

Each of these solutions is allowed to use a `for` or `while` loop and a temporary variable to store a single element, but should not use any auxiliary data structures (no array, no vector, no additional list, etc.)

Note that these solutions are quite different than the algorithms that reverse a vector or list by swapping values. If the algorithm is paused and printed midway, the vector contents will be different than the algorithms for Checkpoint 1. (Go ahead, add more calls to the print function in the middle of your reverse function.) Test and debug your own code before helping your teammates. Discuss the similarities and differences between the solutions to each version of the `reverse` function.

**To complete this checkpoint,** as a team, present your debugged solutions to a TA or mentor.

### Checkpoint 3: Reversing a Homemade Linked List *estimate: TBD*

This checkpoint is an individual checkpoint. (But you can ask your teammates questions if you get stuck.)

Pull out some paper. Following the conventions from lecture, draw a picture of a "homemade" singly-linked list that stores the values 1, 2, 3, and 4. Make a variable on the stack named `my_list` of type `Node*` that points to the first node in the chain (the node storing the value 1). The 4 node objects should be separate blobs of memory dynamically-allocated on the heap.

Now, *modify this diagram* to reverse the list – you can do this by only changing pointers! You should use the existing node objects. Don't copy the entire diagram or make any new nodes. You should not change the values inside of any node – *don't swap values like we did for Checkpoint 1.*

Then, write pseudo-code to reverse this list, just changing the pointers as you diagrammed above. You may use helper variables (of type `Node*`) but no other data structures or variables. *Remember that when we directly manipulate homemade linked lists we don't use iterators.*

Finally, download this starter code:
http://www.cs.rpi.edu/academics/courses/spring23/csci1200/labs/06_lists_iterators/checkpoint3.cpp

And complete the `reverse` function using your diagram and pseudocode as a guide. Test and debug the code. Add a few additional test cases to the main function to ensure your code works with an empty list, and lists with one or two values. Also add a test or two of a node chain with something other than `int`s.

If you have time, write 2 versions of this function, one version should be iterative (using a `for` or `while` loop) and one version should be recursive.

**To complete this checkpoint,** show a TA or mentor your diagram and your debugged function(s) to reverse a homemade singly-linked list.