This is a follow-up of homework 5, which suggests that you may reuse your definitions and implementations of the doubly linked list template. Make your list look like the STL list container (refer to the API). As in the last homework, you can refer to Chapter 9-8 of the reference book [Ford and Topp, 2002], where it goes through an example of building a miniList.

Write a template function prefix_sum() to compute the partial sums of this list. Function prefix_sum() shares the same interface as the STL partial_sum() algorithm. The assignment requires some readings outside of the classroom, so better start working on it as early as you can. Prefix_sum is one of the important ideas you should be familiar with as a software developer, because it can be used to solve a range of problems, especially recursive problems.

Your list container and the prefix_sum() function do not have to have the exactly same interface as their STL correspondence, but should be sufficient to support the testing you will conduct as following:

Test your code with integer value as list elements {1, 2, 3, 4, 5, 6, 7, 8, 9}. You should:
1. Run your list container with the STL algorithm partial_sum();
2. Test your template function prefix_sum() on an STL list container;

```cpp
int main() {
    list<int> stlList;
    myList<int> myList;

    partial_sum(myList.begin(), myList.end(), myList.begin());
    // output
    prefix_sum(stlList.begin(), stlList.end(), stlList.begin());
    // output
}
```

The two bold lines are the two lines I am looking for in the source code. Of course they would generate the same result; the only output I am looking for is:

Original array:
1 2 3 4 5 6 7 8 9
Result after running mylist with STL partial_sum() algorithm
1 3 6 10 15 21 28 36 45
Result after running STL list with my prefix_sum() algorithm
1 3 6 10 15 21 28 36 45
When implementing `prefix_sum()` algorithm, we need to extract the type of the data an iterator points to. This is not trivial, without help from a concept called `iterator traits`. In your own `prefix_sum()` function, you can

```cpp
typedef typename iterator_traits<T>::value_type value_type
```

then you can use `value_type` to extract the data type an iterator points to like this:

```cpp
value_type vt = *iter
```

STL uses the same mechanism in its `partial_sum()` implementation. So, you need to provide the information to make it work in your iterator class:

```cpp
typedef bidirectional_iterator_tag iterator_category;
typedef ptrdiff_t difference_type;
typedef T value_type;
typedef T * pointer;
typedef T & reference;
```

An explanation and example of using the STL `partial_sum()` algorithm can be found at the MSDN page:


Bonus points go to the first two submissions of this assignment.