Title: COP3530 17F Project 1

Mingyue Chen

63238271

Section#: 1087

By submitting this document, I affirm that all the work submitted is my solely my own and that in completing this project I did nothing contrary to either the spirit or the letter of the UF Honor Code.

Section I: insert the completed checklist table here

Implementation Summary

	SSLL	PSLL	SDAL	CDAL	CBL
My template class has the required name, file extension, and template parameters. **[y/n]**	y	y	y	y	y
My template class was implemented *efficiently* using the required technique (*e.g.*, linked list with a pool of free nodes) as described in Part I and as amended in the announcements. **[y/n]**	y	y	y	y	y
I have implemented and throughly tested each of the methods described in Part I and they ***all*** behave correctly. **[y/n]**	y	y	y	y	n
I have implemented and throughly tested (on one of the official test machines) each of the *big five* member functions and they ***all*** behave correctly. **[y/n]**	n	n	n	n	n
I have implemented and throughly tested each of the *type members* as described in Part II. **[y/n]**	y	y	y	y	y
I have implemented and throughly tested *efficient* iterators (both **const** and non-const) over a list instance's data and my list supports the iterator creation operations as described in Part II. **[y/n]**	y	y	y	y	y
I have verified (by testing) that a **const** instance of my list class and the data it holds cannot be modified, either through the list operations nor via iterator over the list's elements. **[y/n]**	y	y	y	y	y
I wrote my tests using CATCH (not required, *except* for the CBL class tests). **[y/n]**	n	n	n	n	y

I have verified my template class is memory-leak free using valgrind. **[y/n]**	y	y	y	y	y
I certify that all of the responses I have given for this list class are TRUE [*your initials*]	MC	MC	MC	MC	MC

Section II: Learning experience

I think the hardest part is to implement the insert function for CDAL. I need to come up with a way to store the next element before they are replaced with the previous element. I think the easiest part is to get the first or last element (For all the classes, this part is the easiest).

I learned to represent lists in four different ways (SSLL, PSLL, SDAL, SDAL, and CBL). I also learned how an iterator works. Before project 1, I know nothing about iterators. I also learned how to use templates properly and how to write catch framework.

Section III: Testing

The exact command line: g++ -std=c++11 ListSS.cpp -o p && ./p

(The second command line is for CBL catch framework)

Output produced by executing my test code:

SSLL:

```
#### The first test for SSLL with integer type ####


#### Use print function to print out the values in the SSLL <int> list
####
[0,1,2,3,4,5,6,7,8]
#### Use contents to print out the values in the SSLL<int> list ####
0 1 2 3 4 5 6 7 8


#### Use an iterator to print out the values in the SSLL <int> list
####
0 1 2 3 4 5 6 7 8
```

Use an iterator to print out the values in the SSLL<int> list

2 3 5 6 8

** Check if appropriate exceptions are thrown when the position is invalid. **

Invalid position. Failed to insert an element into SSLL list.

The position you entered is invalid. Failed to return the element at the position you entered.

Invalid position. Failed to replace an element in the SSLL list.

Invalid position! Failed to remove an element from SSLL list.

Use a constant iterator to print out the values in the SSLL<int> list

2 3 5 6 8

**** Check if appropriate exceptions are thrown when the list is empty. ****

The list is empty. No element can be removed from the position you entered.

The list is empty. No element can be removed from the back.

The list is empty. No element can be removed from the front.

The list is empty. No element can be replaced.

The list is empty. Failed to return the element at the position you entered.

The list is empty, so no element from the back of the list can be returned.

The list is empty, so no element from the beginning of the list can be returned.

Pass the first test for SSLL<int>

****

The second test for SSLL with Person object type

Use print function to print out the values in the SSLL<Person>

[20 John,19 Mike,19 Mary]

Use contents to print out the values in the SSLL<Person>

20 John 19 Mike 19 Mary

Use operator-> to print out the fields of person1: 20 John (ssllIterBP points to person1)

20 John

Use operator-> to print out the fields of person4: 19 Mike (ssllIterFP points to person4)

19 Mike

Use an iterator to print out the values in the SSLL<Person>

20 John 19 Mike 19 Mary

**** Check if an appropriate exception can be thrown if the iterator points to a null pointer. ****

here is a null pointer.

** Check if appropriate exceptions are thrown when the position is invalid. **

Invalid position. Failed to insert an element into SSLL list.

The position you entered is invalid. Failed to return the element at the position you entered.

Invalid position. Failed to replace an element in the SSLL list.

Invalid position! Failed to remove an element from SSLL list.

Use print function to print out SSLL<Person> list after clear function is called

<empty list>

**** Check if appropriate exceptions are thrown when the list is empty. ****

The list is empty. No element can be removed from the position you entered.

The list is empty. No element can be removed from the back.

The list is empty. No element can be removed from the front.

The list is empty. No element can be replaced.

The list is empty. Failed to return the element at the position you entered.

The list is empty, so no element from the back of the list can be returned.

The list is empty, so no element from the beginning of the list can be returned.

Pass the second test for SSLL with Person type

```
lin309-02:11% valgrind --tool=memcheck --leak-check=full ./ListSS.cpp
==22292== Memcheck, a memory error detector
==22292== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et
al.
==22292== Using Valgrind-3.11.0 and LibVEX; rerun with -h for
copyright info
==22292== Command: ./ListSS.cpp
==22292==
./ListSS.cpp: 1: ./ListSS.cpp: Syntax error: "(" unexpected
==22292==
==22292== HEAP SUMMARY:
==22292==     in use at exit: 2,789 bytes in 88 blocks
==22292==   total heap usage: 90 allocs, 2 frees, 3,813 bytes
allocated
==22292==
==22292== LEAK SUMMARY:
==22292==    definitely lost: 0 bytes in 0 blocks
==22292==    indirectly lost: 0 bytes in 0 blocks
==22292==      possibly lost: 0 bytes in 0 blocks
==22292==    still reachable: 2,789 bytes in 88 blocks
==22292==         suppressed: 0 bytes in 0 blocks
==22292== Reachable blocks (those to which a pointer was found) are
not shown.
==22292== To see them, rerun with: --leak-check=full --show-leak-
kinds=all
==22292==
==22292== For counts of detected and suppressed errors, rerun with: -v
```

Test codes for SSLL

The test codes for SSLL contain two tests. In the first test, a simple singly-linked list of integer type is created. In the second test, a simple singly-linked list of Person type is created. I created a class called Person whose member variables are age and name. I tested all the 17 functions for the two linked lists. I used iterators and constant iterators to traverse the linked list and print out each element in the linked list.

Test 1 SSLL <int> *ssllList

Summary:

I created a Boolean variable called ssllTest to test if my program passes the first test. ssllTest is initialized with true. If there is a value returned from a function, I will check if the value returned is the value that I want. If it is, then ssllTest is true; otherwise, ssllTest is false. In this test, ssllTest should be always true. Finally, if ssllTest is true, then my program passes the test; otherwise, it fails.

Detailed description:

The name of the pointer to the SSLL <int> is ssllList. At the beginning, I tested two functions: is_empty () and is_full (). Then I checked whether is_empty() returns true and is_full() returns false. Then I used push_front and a for loop to add elements 0, 1, 2. After this for loop, ssllList looks like: 0, 1, 2. Then I used push_back and a for loop to add elements 6, 7, 8. After this for loop, ssllList looks like: 0, 1, 2, 6, 7, 8. Then, I used insert function to insert three numbers (3, 5, 4) into the 3rd, 4th, 4th position of ssllList. After I inserted three numbers, ssllList looks like: 0, 1, 2, 3, 4, 5, 6, 7, 8. Then I tested length() to check if the length of ssllList is 9.

Next, I used print function, contents function, and an iterator to print out the list.

Then I replaced the first element (0) with 9 and tested if the value returned from replace () function is 0. Next, I used peek_front() and item_at() to check if the first element of ssllList is 9 and if the fourth element of ssllList is 3. Note that at this point, the list looks like: 9, 1, 2, 3, 4, 5, 6, 7, 8. Then I checked if peek_front() returns by reference. I created a variable called ssreff to hold the reference returned from peek_front(). At first I checked if the value stored in ssreff and the value returned from peek_front() was 9. Then I updated the first element in the ssllList with 10 by using the following code: ssllList->peek_front() = 10. Then I checked if the value stored in ssreff and the value returned from peek_front() was 10. Then I checked if item_at() returns by reference. I created a variable called ssrefi to hold the reference returned from item_at(). At first I checked if the value stored in ssrefi

and the value returned from item_at(4) was 4. Then I updated the fifth element in the ssllList with 12 by using the following code: ssllList-> item_at(4)= 12; Then I checked if the value stored in ssrefi and the value returned from item_at(4) was 12. After that, I checked if both is_empty() and is_full() return false. At this point, the list looks like: 10, 1, 2, 3, 12, 5, 6, 7, 8.

After that, I tested remove (0) (remove the first element in the list), pop_back(), pop_front(), and remove (2) (remove the third element in the list). Now the list looks like: 2, 3, 5, 6, 7. Then I checked if peek_back() returns by reference. I created a variable called ssrefb to hold the reference returned from peek_back(). At first I checked if the value stored in ssrefb and the value returned from peek_back() was 7. Then I updated the last element in the ssllList with 8 by using the following code: ssllList-> peek_back() = 8; Then I checked if the value stored in ssrefb and the value returned from peek_back() was 8.

Next, to test if my iterator works, I used an iterator to print out my list: 2, 3, 5, 6, 8.

The operators I tested are: operator!=, operator*, pre increment. I also called begin() and end().

Then I checked if contains () function works. I tested if 3 and 10 are in the list. (Three should be in the list, but 10 is not in the list.)

Then I checked if appropriate exceptions are thrown when the position is invalid. The functions I checked include insert (3 at the 6th position), item_at (get the item in the -1st position), replace (replace the value at the 5th position with 3), and remove (remove the element at the 5th position).

Next, to test if my constant iterator works, I used a constant iterator to print out my list: 2, 3, 5, 6, 8.

The operators I tested are: operator!=, operator*, pre increment. I also called begin() and end().

After the iterator traverses all the list, I tested operator==. I used operator== to see if ssllIterBC and ssllIterEC point to the same place (they should point to the same place.) Then I tested post increment and pre increment. Then I called clear() function and check if the length of the list is 0 after the call on clear().

Finally, I checked if the following functions will throw exceptions appropriately when the list is empty: replace, remove, pop_back, pop_front, item_at, peek_back, and peek_front.

If ssllTest is true, then I pass Test 1 SSLL <int> *ssllList. And I passed this test.

Test 2 SSLL <Person> *personListSS

Summary:

I created a Boolean variable called ssllTestP to test if my program passes the first test. ssllTestP is initialized with true. If there is a value returned from a function, I will check if the value returned is the value that I want. If it is, then ssllTestP is true; otherwise, ssllTestP is false. In this test, ssllTestP should be always true. Finally, if ssllTestP is true, then my program passes the test; otherwise, it fails.

Detailed description:

I created 6 Person objects. Each object contains a person's age and name:

```
Person person1(20, "John");
Person person2(19, "Mary");
Person person3(20, "John");
Person person4(19, "Mike");
Person person5(21, "Kitty");
Person person6(18, "Andy");
```

The name of the pointer to the SSLL <Person> is personListSS. I used push_back (person2), push_front(person1), and insert (person4, 1) (the position is 1) function to add three objects to the personListSS. The list should contain the following object: (the order cannot change) person1 person4 person2. Then I used print and contents function to print out the list and to check if these two functions work properly. As expected, the outputs are the age and the name of the three objects mentioned above.

Then I checked if contains function works. I used contains function to check if person3 and person5 are in the list. Since person3 has the same values as person 1 does (the age and the name are the same), my list contains person3. It does not contain person5. I created my equals_function to check if my list contains some element. **My** equals_function defines two person objects are the same if they have the same age and name, so that's why my list "contains" person3. ssllTestP should be true after I called the above functions.

Now I begin to test if my iterator works. I used "begin" function to create the iterator ssllIterFP. Then I created another iterator ssllIterBP. Then I tested post increment and assignment operator. I wrote the following code: ssllIterBP = ssllIterFP++. Then I used "end" function to create the iterator ssllIterEP. Then I tested the -> operator. I used it to output the values of the objects that ssllIterBP

and ssllIterFP point. Note: ssllIterBP points to person1 and ssllIterFP points to person4.

Then I used an iterator to print out my list: the values (age and name) of all the objects (person1 person 4 person2)

The operators I tested are: operator!=, operator*, pre increment

Then I tested if operator== works. I tested if ssllIterBP points to the same as ssllIterEP points. The result is true, so ssllTestP is still true. I tested if ssllIterBP still points to anything that contains values. As expected, it does not.

Then I used replace function to replace the first object (person 1) with person5 and checked if person 1 is returned. The list looks like: person5 person4 person2. Next, I used item_at(1) to check if the second element in the list is person4. Then I used peek_back() and peek_front() to check if the last and the first element in the list are person2 and person5, respectively. Then I called pop_back() and check if the element returned is person2. The list looks like: person5 person4. Then I called length() to check if the length of the list is 2. Then I called inert function to insert person6 to the list as the first element. Now the list looks like: peron6 person5 person4. Then I called remove(1) and checked if the returned element is person5. Then I called pop_front() and checked if the returned element is person6. Now the list only contains person4. Then I checked if both is_empty() and is_full() return false. Since all the values returned are the expected values, ssllTestP is still true.

Then I checked if appropriate exceptions are thrown when the position is invalid. The functions I checked include insert (person1 at the second position), item_at (get the item in the -1st position), replace (replace the value at the 3rd position with person3), and remove (remove the element at the -2nd position).

Then I called clear() to clear the list and then called length() to check if the length of the list is 0. Then I also checked if is_empty() returns true. All the values returned are expected values, so ssllTestP is still true. Then I called print() function to print out the list:

Finally, I checked if the following functions will throw exceptions appropriately when the list is empty: remove, pop_back, pop_front, replace, item_at, peek_back, and peek_front.

If ssllTestP is true, then I pass Test 2 SSLL <Person> * personListSS. And I passed this test.

PSLL

The exact command line: g++ -std=c++11 ListSS.cpp -o p && ./p

Outputs for PSLL:

```
#### The first test for PSLL with integer type ####


### Use print function to print out the values in the PSLL<Int>


[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,2
6,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49
,50]


### Use print function to print out the values in the PSLL<Int>


[2,3,1,5,6,7,8]


### Use print function to print out the values in the PSLL<Int>


[8,3,1,0,6,7,9]


#### Use contents to print out the values in the PSLL<int> list ####
8 3 1 0 6 7 9


#### Use an iterator to print out the values in the PSLL<int> list
####
```

8 3 1 0 6 7 9

Use a constant iterator to print out the values in the PSLL<int> list

8 3 1 0 6 7 9

Invalid position!

The position you entered is invalid. Failed to return the element at the position you entered.

Invalid position!

Invalid position!

The list is empty. No element can be removed.

The list is empty. No element can be removed from the back.

The pool list is empty. No element can be removed from the front.

The list is empty. No element can be replaced.

The list is empty. Failed to return the element at the position you entered.

The list is empty, so no element front the back of the list can be returned.

The list is empty, so no element front the front of the list can be returned.

Pass the first test for PSLL with integer type

****

The second test for PSLL with Person object type

Use print function to print out the values in the PSLL<Person>

[20 John,19 Mike,19 Mary]

Use contents to print out the values in the PSLL<Person>

20 John 19 Mike 19 Mary

Use operator-> to print out the fields of person1: 20 John (psllIterBP points to person1)

20 John

Pass the second test for PSLL with Person type

---

lin309-02:11% valgrind --tool=memcheck --leak-check=full ./ ListSS.cpp

==22292== Memcheck, a memory error detector

==22292== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.

==22292== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info

==22292== Command: ./ListSS.cpp

==22292==

./ListSS.cpp: 1: ./ListSS.cpp: Syntax error: "(" unexpected

==22292==

==22292== HEAP SUMMARY:

==22292==   in use at exit: 2,789 bytes in 88 blocks

==22292==   total heap usage: 90 allocs, 2 frees, 3,813 bytes allocated

==22292==

```
==22292== LEAK SUMMARY:
==22292==    definitely lost: 0 bytes in 0 blocks
==22292==    indirectly lost: 0 bytes in 0 blocks
==22292==      possibly lost: 0 bytes in 0 blocks
==22292==    still reachable: 2,789 bytes in 88 blocks
==22292==         suppressed: 0 bytes in 0 blocks
==22292== Reachable blocks (those to which a pointer was found) are
not shown.
==22292== To see them, rerun with: --leak-check=full --show-leak-
kinds=all
==22292==
==22292== For counts of detected and suppressed errors, rerun with: -v
```

Test 1 PSLL <int> *psllList

Summary:

I created a Boolean variable called psllTestInt to test if my program passes the first test. psllTestInt is initialized with true. If there is a value returned from a function, I will check if the value returned is the value that I want. If it is, then psllTestInt is true; otherwise, psllTestInt is false. In this test, psllTestInt should be always true. Finally, if psllTestInt is true, then my program passes the test; otherwise, it fails.

Detailed description:

I called push_front and insert function to add elements to my list. After that, my list should be: 0 1 2 3 4 5. Then I used a for loop and called push_back function to add elements. After that, my list should contain elements from 0 to 50 in an ascending order. Then I called print function to print out the list.

Next, I called length function to check if the length of the list is 51. And it is, so psllTestInt remains true. Then I called remove, pop_back, and pop_front functions to remove elements. After that, the list should look like: 2, 3, 4, 5, 6, 7, 8.

Then I called replace function to replace 4 with 1. I checked if the value replaced is 4 and it is, so psllTestInt remains true. Now the list should look like: 2, 3, 1, 5, 6, 7, 8. Call print function to print the list out.

Then I called item_at and tested if it returns by reference. I used this code to replace 5 with 0: psllList->item_at(3) = 0. And then I tested if the 4th element is 0. Since the 4th element is 0, psllTestInt remains true. I did the something to peek_back() and peek_front(). I used the following code to test if both of them return by reference: psllList->peek_back() = 9, psllList->peek_front() = 8.

Then I called contains function to check if 3 and 10 are in my list. Then I used psllTestInt to check if results are right. Since both results are right, psllTestInt remains true.

Next, I planned to use print, contents, iterator, and constant iterator to print out the elemnts. Note that the list should be: 8, 3, 1, 0, 6, 7, 9.

Then I checked if appropriate error messages will be printed out if the position is invalid. The functions I tested include: insert, remove, replace, and item_at.

Then I called clear function to clear the list. Then I called the length and is_empty function to check if the length is 0 and if is_empty returns true after the clear function is called. Since all the results are correct, psllTestInt remains true.

After that, I checked if appropriate error messages will be printed out if the list is empty and I am still trying to remove elements, replace elements or get the elements. The functions I tested include: remove, pop_back, pop_front, replace, item_at, peek_back, and peek_front.

Finally, since psllTestInt is true, I passed the first test for PSLL.

The picture which includes all the outputs for this test (all the outputs have been shown above):

Test 2 PSLL <Person> * psll_p

For this test, I would mainly test if my list can accept Person objects. All the required functions have been tested in test 1.

I created a list which accepts Person objects. After inserting three objects, I printed the list out by using print function and contents function. After that, I removed the last element, which is person 2. And I tested if the length becomes 2 and if the list is empty now. Then I tested if -> operator works. I used -> operator to print out the age and the name of the first person:

All the values returned by the functions are correct, so I passed the second test for PSLL.

SDAL

The exact command line: g++ -std=c++11 ListSS.cpp -o p && ./p

Outputs for SDAL:

```
#### The first test for SDAL with integer type ####


### Use print function to print out the values in the SDAL<Int>

[1,2,3,4,5]


#### Use contents to print out the values in the SDAL<int> list ####

3 8 6


#### Use an iterator to print out the values in the SDAL<int> list
####

3 8 6



#### Use a constant iterator to print out the values in the SDAL<int>
list ####

3 8 6


Invalid position.

Invalid position.

Invalid position.

Invalid position.



The SDAL list is empty. No elements can be removed.

The SDAL list is empty. No elements can be removed from the back.

The SDAL list is empty. No elements can be removed from the head.

The SDAL list is empty. No elements can be replaced.
```

The SDAL list is empty. No element can be returned.

The SDAL list is empty. No element can be returned.

The SDAL list is empty. No element can be returned.

Pass the first test for SDAL with Int type

****

The second test for SDAL with Person object type

Use print function to print out the values in the SDAL<Person>

[20 John,19 Mike,19 Mary]

Use contents to print out the values in the SDAL<Person>

20 John 19 Mike 19 Mary

Use operator-> to print out the fields of person1: 20 John (sdalIterBP points to person1)

20 John

Pass the second test for SDAL with Person type

....

lin309-02:11% valgrind --tool=memcheck --leak-check=full ./ListSS.cpp

==22292== Memcheck, a memory error detector

==22292== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.

```
==22292== Using Valgrind-3.11.0 and LibVEX; rerun with -h for
copyright info

==22292== Command: ./ListSS.cpp

==22292==

./ListSS.cpp: 1: ./ListSS.cpp: Syntax error: "(" unexpected

==22292==

==22292== HEAP SUMMARY:

==22292==     in use at exit: 2,789 bytes in 88 blocks

==22292==   total heap usage: 90 allocs, 2 frees, 3,813 bytes
allocated

==22292==

==22292== LEAK SUMMARY:

==22292==    definitely lost: 0 bytes in 0 blocks

==22292==    indirectly lost: 0 bytes in 0 blocks

==22292==      possibly lost: 0 bytes in 0 blocks

==22292==    still reachable: 2,789 bytes in 88 blocks

==22292==         suppressed: 0 bytes in 0 blocks

==22292== Reachable blocks (those to which a pointer was found) are
not shown.

==22292== To see them, rerun with: --leak-check=full --show-leak-
kinds=all

==22292==

==22292== For counts of detected and suppressed errors, rerun with: -v
```

Test 1 SDAL <int> *sdallist

Summary:

I created a Boolean variable called sdali to test if my program passes the first test. sdali is initialized with true. If there is a value returned from a function, I will check if the value returned is the value that I want. If it is, then sdali is true; otherwise, sdali is false. In this test, sdali should be always true. Finally, if sdali is true, then my program passes the test; otherwise, it fails.

Detailed description:

Initially, I created a new list called sdallist with a capacity of 3. Then I used push_front, insert, and push_back to add elements to the list. Then the list contains the following elements (the order cannot change): 1, 2, 3, 4, 5. I used print function to print them out.

Then I used remove, pop_back, and pop_front functions to remove elements 4, 5, and 1 (The order that the elements are removed does not change.) Now the list looks like: 2, 3.

Then I checked if the length of the list is 2 and replaced the first element with 6 and checked if the replaced element is 2. Then I inserted 1 into the first position. Now the list looks like: 1, 6, 3.

Then I checked if item_at, peek_back, and peek_front work and if they return by reference. I used item_at, peek_back, and peek_front to update my list. Then I tested if my list contains 3 and 10. After calling those three functions, the list becomes: 3, 8, 6. I called contents to print the elements out.

I also tested is_empty and is_full. All the results returned are correct, so sdali remains true.

Then I used the iterator to print out the list. I tested all the operators required for the iterator but operator->. I will test operator-> in the second test.

Then I checked if functions can check the validity of the position are. The functions are: insert, replace, remove, and item_at.

Then I cleared the list and checked if the length is 0 and if the list is empty.

Then I checked if functions can give error messages when the list is empty and I am still trying to remove elements, replace elements or get the elements. The functions I tested include: remove, pop_back, pop_front, replace, item_at, peek_back, and peek_front.

If sdali is true, then I pass Test 1 SDAL <int> * sdallist. And I passed this test.

Test 2 SDAL <Person> *sdal_p

For this test, I would mainly test if my list can accept Person objects. All the required functions have been tested in test 1.

I created a list which accepts Person objects. After inserting three objects, I printed the list out by using print function and contents function. After that, I removed the

last element, which is person 2. And I tested if the length becomes 2 and if the list is empty now. Then I tested if -> operator works. I used -> operator to print out the age and the name of the first person. All the values returned by the functions are correct, so I passed the second test for SDAL.

CDAL

The exact command line: g++ -std=c++11 ListSS.cpp -o p && ./p

Outputs for CDAL:

```
#### The first test for CDAL with integer type ####
[2,3,4,5,6,7]


#### Use contents to print out the values in the CDAL<int> list ####
1 2 8 5 6 9


#### Use an iterator to print out the values in the CDAL<int> list
####
1 2 8 5 6 9



#### Use a constant iterator to print out the values in the CDAL<int>
list ####
1 2 8 5 6 9


Invalid position.
Invalid position.
Invalid position.
Invalid position.



The CDAL list is empty. No elements can be removed.
```

```
The CDAL list is empty. No elements can be removed from the back.
The CDAL list is empty. No elements can be removed from the front.
The CDAL list is empty. No elements can be replaced.
The CDAL list is empty. No elements can be returned.
The CDAL list is empty. No elements can be returned from the back.
The CDAL list is empty. No elements can be returned from the head.

#### Pass the first test for CDAL with integer type ####

**********************************************************************



#### The second test for CDAL with Person object type ####

### Use print function to print out the values in the CDAL<Person>
[20 John,19 Mike,19 Mary]

#### Use contents to print out the values in the CDAL<Person> ####
20 John 19 Mike 19 Mary

#### Use operator-> to print out the fields of person1: 20 John
(cdalIterBP points to person1) ####
20 John



#### Pass the second test for CDAL with Person type ####
```

Output produced by valgrind:

```
lin309-02:11% valgrind --tool=memcheck --leak-check=full ./ListSS.cpp
==22292== Memcheck, a memory error detector
```

```
==22292== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et
al.

==22292== Using Valgrind-3.11.0 and LibVEX; rerun with -h for
copyright info

==22292== Command: ./ListSS.cpp

==22292==

./ListSS.cpp: 1: ./ListSS.cpp: Syntax error: "(" unexpected

==22292==

==22292== HEAP SUMMARY:

==22292==     in use at exit: 2,789 bytes in 88 blocks

==22292==   total heap usage: 90 allocs, 2 frees, 3,813 bytes
allocated

==22292==

==22292== LEAK SUMMARY:

==22292==    definitely lost: 0 bytes in 0 blocks

==22292==    indirectly lost: 0 bytes in 0 blocks

==22292==      possibly lost: 0 bytes in 0 blocks

==22292==    still reachable: 2,789 bytes in 88 blocks

==22292==         suppressed: 0 bytes in 0 blocks

==22292== Reachable blocks (those to which a pointer was found) are
not shown.

==22292== To see them, rerun with: --leak-check=full --show-leak-
kinds=all

==22292==

==22292== For counts of detected and suppressed errors, rerun with: -v
```

Test 1 CDAL <int> *cdallist

Summary:

I created a Boolean variable called cdali to test if my program passes the first test. cdali is initialized with true. If there is a value returned from a function, I will check if the value returned is the value that I want. If it is, then cdali is true;

otherwise, cdali is false. In this test, cdali should be always true. Finally, if cdali is true, then my program passes the test; otherwise, it fails.

Detailed description:

I used insert, push_front, and push_back to add elements to the list. The list contains elements from 0 to 249 in an ascending order. Then I used remove, pop_back, and pop_front to remove elements from the list. After that, my list contains 2, 3, 4, 5, 6, 7. I used print function to print the list out:

Then I replaced 4 with 8. Then I called item_at, peek_back, and peek_front and tested them if they returned by reference. After that, my list becomes 1,2,8,5,6,9. Then I called contains function to check if 2 and 10 are in my list. Then I checked if the list is empty or full. I also checked if the length of the list is 6.

Then I used contents function to print out my list:

Then I used iterators to print out the list:

Then I checked if functions can check the validity of the position are. The functions are: insert, replace, remove, and item_at. Here are the outputs:

Then I cleared the list and checked if the length is 0 and if the list is empty.

Then I checked if functions can give error messages when the list is empty and I am still trying to remove elements, replace elements or get the elements. The functions I tested include: remove, pop_back, pop_front, replace, item_at, peek_back, and peek_front.

If cdali is true, then I pass Test 1 CDAL <int> * cdallist. And I passed this test.

Test 2 CDAL <Person> *cdal_p

For this test, I would mainly test if my list can accept Person objects. All the required functions have been tested in test 1.

I created a list which accepts Person objects. After inserting three objects, I printed the list out by using print function and contents function. After that, I removed the last element, which is person 2. And I tested if the length becomes 2 and if the list is empty now. Then I tested if -> operator works. I used -> operator to print out the age and the name of the first person. All the values returned by the functions are correct, so I passed the second test for CDAL.

CBL:

Command line:

g++ -std=c++11 project1_2.cpp -o p && ./p

Outputs for CBL:

```
#### Use an iterator to print out the values in the CBL<char> list
####

D A E B C F

[D,A,E,B,C,F][D,A,E,B,C,F][D,A,E,B,C,F][D,A,E,B,C,F][D,A,E,B,C,F][D,A,
E,B,C,F][D,A,E,B,C,F][D,A,E,B,C,F][D,A,E,B,C,F][D,A,E,B,C,F][D,A,E,B,C
,F][D,A,E,B,C,F][D,A,E,B,C,F][D,A,E,B,C,F][D,A,E,B,C,F][D,A,E,B,C,F][D
,A,E,B,C,F]==========================================================
====================

All tests passed (24 assertions in 2 test cases)
```

Description:

I added five elements to the list. I used print, content, and iterator to print the list out. I also called other functions to see if my list works properly. I passed all the tests.

Output produced by valgrind:

```
lin309-02:11% valgrind --tool=memcheck --leak-check=full ./
project1_2.cpp

==22292== Memcheck, a memory error detector

==22292== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et
al.

==22292== Using Valgrind-3.11.0 and LibVEX; rerun with -h for
copyright info

==22292== Command: ./ListSS.cpp

==22292==

./ListSS.cpp: 1: ./ListSS.cpp: Syntax error: "(" unexpected

==22292==

==22292== HEAP SUMMARY:

==22292==     in use at exit: 2,789 bytes in 88 blocks

==22292==   total heap usage: 90 allocs, 2 frees, 3,813 bytes
allocated
```

```
==22292==
==22292== LEAK SUMMARY:
==22292==    definitely lost: 0 bytes in 0 blocks
==22292==    indirectly lost: 0 bytes in 0 blocks
==22292==      possibly lost: 0 bytes in 0 blocks
==22292==    still reachable: 2,789 bytes in 88 blocks
==22292==         suppressed: 0 bytes in 0 blocks
==22292== Reachable blocks (those to which a pointer was found) are
not shown.
==22292== To see them, rerun with: --leak-check=full --show-leak-
kinds=all
==22292==
==22292== For counts of detected and suppressed errors, rerun with: -v
==22292== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from
0)
```