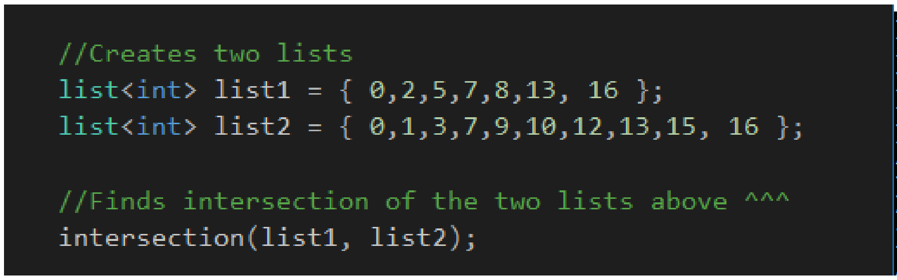
CECS 302-01 Assignment #2

Nicholas Gittings

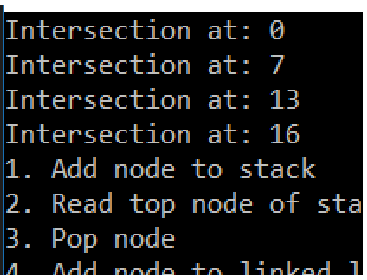
September 27, 2018

For the first problem, order the functions by growth which is 2/N, 37, $\sqrt{N}$, N, $nloglogn$, $nlogn$, $nlogn^2$, $nlog^2n$, $n^{1.5}$, n^2, n^2logn, n^3, $2^{n/2}$, 2^n. Functions that grow the same are $nlogn$ and $nlogn^2$, and $2^{n/2}$ and 2^n.

For problem two, create a function called intersection which takes two integer lists as parameters. Create two for loops that use iterators to go through both lists and if the value at iterators are equal then they are printed out.



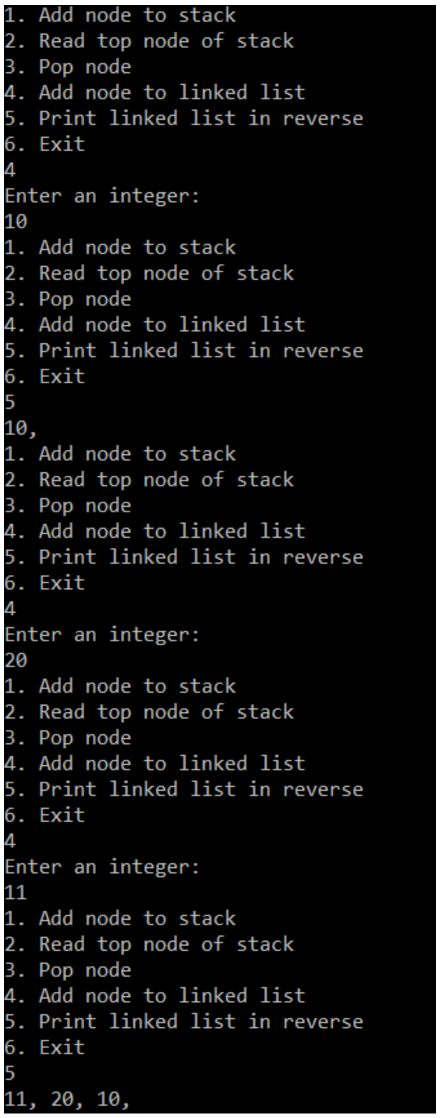
```
//Creates two lists
list<int> list1 = { 0,2,5,7,8,13, 16 };
list<int> list2 = { 0,1,3,7,9,10,12,13,15, 16 };

//Finds intersection of the two lists above ^^^
intersection(list1, list2);
```



```
Intersection at: 0
Intersection at: 7
Intersection at: 13
Intersection at: 16
1. Add node to stack
2. Read top node of sta
3. Pop node
4. Add node to linked l
```

Screenshot shows which integers the lists have in common

For problem three, create a linked list class for testing. In this class create a function to reverse print which will go through the linked list like normal but will store the values in a list using push_front(). Then use iterators to go through the beginning of the list to the end and printing the value at the iterator.

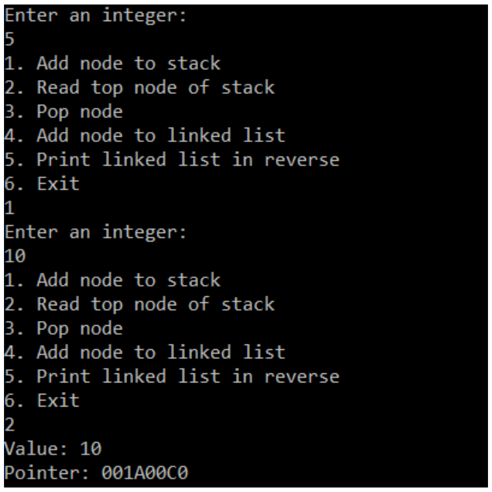


```
1. Add node to stack
2. Read top node of stack
3. Pop node
4. Add node to linked list
5. Print linked list in reverse
6. Exit
4
Enter an integer:
10
1. Add node to stack
2. Read top node of stack
3. Pop node
4. Add node to linked list
5. Print linked list in reverse
6. Exit
5
10,
1. Add node to stack
2. Read top node of stack
3. Pop node
4. Add node to linked list
5. Print linked list in reverse
6. Exit
4
Enter an integer:
20
1. Add node to stack
2. Read top node of stack
3. Pop node
4. Add node to linked list
5. Print linked list in reverse
6. Exit
4
Enter an integer:
11
1. Add node to stack
2. Read top node of stack
3. Pop node
4. Add node to linked list
5. Print linked list in reverse
6. Exit
5
11, 20, 10,
```

Inputting integers in a list moves them to the end of the list, but are printed in reverse using a regular list

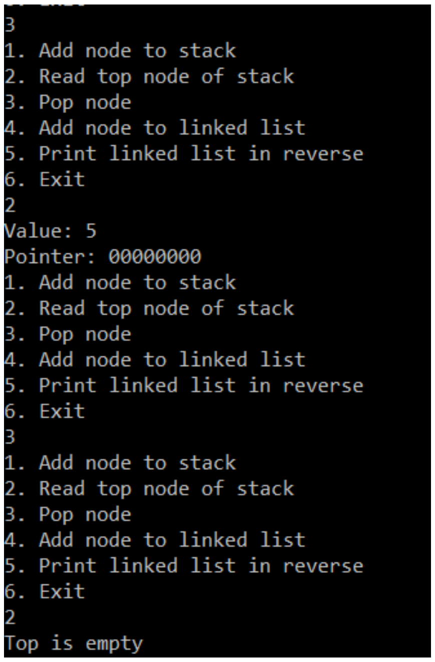
For problem four, inserting or removing elements will cause the iterator to be invalid because the iterator goes through the list from beginning to end and calling during the process will not reach the new insert or will go to an empty element and cause an error.

For problem five, create a class called stack which will use nodes without header or tail. Create a pointer node called top which will be the top of the stack which is only visible. Create three functions push(), pop(), and read(). The push function creates a node which will accept an integer parameter and set the new node to top of the stack. The pop function removes the top node and replaces it with the one underneath. The read function prints out the top node's value.



```
Enter an integer:
5
1. Add node to stack
2. Read top node of stack
3. Pop node
4. Add node to linked list
5. Print linked list in reverse
6. Exit
1
Enter an integer:
10
1. Add node to stack
2. Read top node of stack
3. Pop node
4. Add node to linked list
5. Print linked list in reverse
6. Exit
2
Value: 10
Pointer: 001A00C0
```

Entering integers into stack and prints out value of top of stack



```
3
1. Add node to stack
2. Read top node of stack
3. Pop node
4. Add node to linked list
5. Print linked list in reverse
6. Exit
2
Value: 5
Pointer: 00000000
1. Add node to stack
2. Read top node of stack
3. Pop node
4. Add node to linked list
5. Print linked list in reverse
6. Exit
3
1. Add node to stack
2. Read top node of stack
3. Pop node
4. Add node to linked list
5. Print linked list in reverse
6. Exit
2
Top is empty
```

Popping values bring the previously entered integers to the top and are displayed (if empty it displays the top is empty)

For problem six, the running time of the programs for a single for loop like for A and G depends on the number of iterations in the loop. For the rest of the problems they have nested for loops which means the running time is the product of the size of the loops and the statement inside of the nested group. Function A, B, C, D, E, F, G takes less than a second to process at an N of 100. A is a O(N), B is O(N^2), C is O((N*N)^2), D is O(N*N), E is O(N^3), F is O(N^3*2), and G is O(N). With a big N, functions E and F would take the longest to compute.