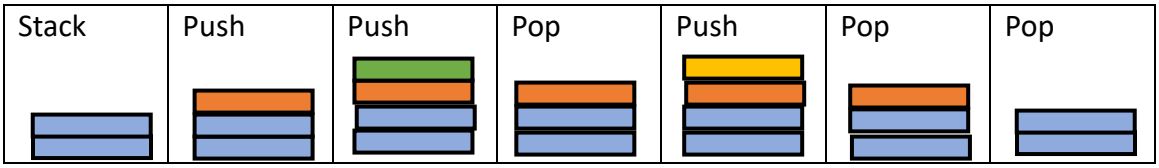
Assignment 1 Report - STACK

Code: Minh Pham

Report: Joseph Levesque

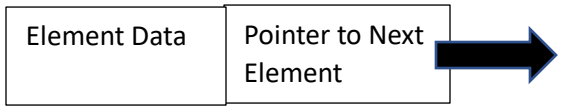
1. A stack is a data structure that holds information using functions that operate on a LIFO model. This means that as data is entered into a stack, it will be removed in the reverse order. The function that adds elements to the top of a stack is called a push and the function that removes elements from the top of the stack is called a pop. For example, if elements 3, 62, 7, 23, and 4 are added to the stack (push) in that order, they will be removed (pop) in this order: 4, 23, 7, 62, 3. The stack data structure is useful when a program needs to go back to the most recent elements added to the stack. For example, the forwards and backwards buttons on an internet browser are a form of a stack because they take you to the most recent places visited (last visited, first directed towards is a form of LIFO).
   Diagram:



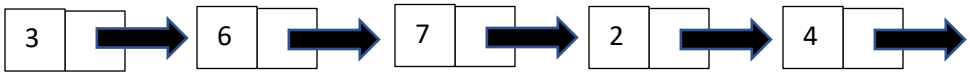
   (Operations in diagram go from left to right)

2. A linked list is a tool that can be used by programmers to dynamically allocate data into memory. It is similar to an array in that it is a list of information, however in a regular array, you need to manually allocate memory for the array at the start and it is difficult to modify and find the chunk of memory in storage to do so. With a linked list, information can be stored all over memory, linked together by pointers that are stored with the data of each list element. This is done by storing the data in nodes which hold two things: the data for the element and the pointer variable to the next node (element). This also allows for the appending and deletion of nodes without memory problems because the data is not all stored in one big chunk. You can allocate new memory if you want to add elements and you can free memory to delete elements.
   Diagram:

   Node:



   Linked List:



(Each note can be stored in its own separate location in memory)

3. Before explaining how an input is processed, let me first explain the classes that we used to build our stack program. We started with a node class, which contains a character value and a pointer that points to another node. We then used this node format to build a linked list class, which contains the functions append, removeEnd, clean, and getEnd. These functions append a node to the end of the linked list, remove a node from the end of the linked list, delete the entire linked list, and get the last data value of the linked list, respectively. These functions were programmed to complete the functions of a stack, however, to make it more clear and to keep track of the size of a stack, we created another class called Stack which contains a linked list, a variable to hold the size of the list, and functions for push, pop, noop (no operation), top, and clean. These functions use the functions defined in the linked list class to do their respective jobs, as well as keep track of the stack size appropriately. The last function defined in the stack class is called isValidString. This is the main function used to determine the output desired in this assignment, and it relies upon the classes and functions described above to do this. It's logic is works like this:

   (Function parameter takes in a string)

   The function then loops through every character in the string and behaves as follows:

   If the character is not an opening character ('(' or '[' or '{') or a closing character (')' or ']' or '}'), the function does nothing and it continues with the next character (no-op).

   If the character is an opening parentheses ('(' or '[' or '{'), then push the character onto the stack. This needs to be stored because in order for the test to return true, the string will need to first find a closing character to match the opening character.

   If the character is a closing character (')' or ']' or '}'), the function checks the top character in the stack. If the top character in the stack (opening character) matches with this closing character, the opening character in the stack is popped because it has now been closed and the process of checking the string resumes. In any other case, this closing character is trying to close something that has not been opened so the function stops and returns false immediately. If at the end of the string the stack size is 0, this means all the opening characters were closed and it should return true.

Now, with an understanding of how the program is built, we can analyze how the test cases are tested. In the main function, a stack is created called "st". Then, we printed the results of the

isValidString function being called for a bunch of test cases, which prints either true or false to the console according to the result of the function.

Example:	**Execution Time (Custom Code)**	**Execution Time (STL library)**
{{{{}}}}[][](())	1.909e-06 ms	7e-07 ms
(%TT{>><]]]%%%%%%)	3.2e-07 ms	2.51e-07 ms
((((((((((){}{}{}[][][][[]]]	1.3e-06 ms	6.49e-07 ms
{{{@23+6767}}}*<45>(a+b)[]	6.11e-07 ms	4.4e-07 ms
(((((a+b %%%%%####3422222222222))))))[]{}[]	7.11e-07 ms	4.8e-07 ms
[[[{{()}}]]]	6.8e-07 ms	4e-07 ms
((((((}}}}}}	3.29e-07 ms	2.2e-07 ms
Aaaaaaaa+bbbb({}{}{}[][][]ttttt + %^$)	8.1e-07 ms	5.5e-07 ms
@@@@@@@@@@@@@@@@@@@@@@@@@$$ %%T$%$%$$%{}()%%%%%%%%%%%%%%%%%%%%%%%%%%%	5.6e-07 ms	5.5e-07 ms
@@@@@@@@@@@@@@@@@@@@@@@@@$$ %%%%%%%####3422222222222))))	2.1e-07 ms	3.4e-07 ms
{{{{}}}}[][sdfsdg3345{[[[[]]]}((())	1.02e-06 ms	4.3e-07 ms
34235]]]}}})_{{sa{{}}}[][](())	1.39e-07 ms	3.49e-07 ms
{{dsafaswwwwwwwwwwwwwwwwwwwwwwwwwwwwww{{}}}[][]((***)())	1.051e-06 ms	5.51e-07 ms
((((((((((){[[]]{}}{}{}[][][][[]]]	1.369e-06 ms	5.09e-07 ms
{{{{}}}}[355----++++][]((()))	7.2e-07 ms	3.89e-07 ms

As shown in this table, there were 2 occasions where our stack was actually faster, maybe because we stopped checking once there was a closing character before an opening character. However in most cases, the library stack functions worked faster than ours.