Ricardo Deodutt

In this assignment, there will be a graph and a code attached. In the graph I attached, I designed it using photoshop. This graph resembles a nondeterministic pushdown accepter (npda). A nondeterministic pushdown accepter is defined by the septuple, The septuple is simply, "$M = (Q, \Sigma, \Gamma, \delta, q0, z, F)$". There are many different symbols that have their own definitions. The Q is a finite set of internal states of the control unit. The symbol Σ is the input alphabet. In the graph that I attached, there is simply 3 input alphabet which are 'a', 'b', and '*'. The next symbol Γ is a finite set of symbols called the stack alphabet. In this assignment the graph I attached follows the stack alphabet {0,1}. The next symbol $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ is a set of finite subset of $Q \times \Gamma^*$. The next symbol q0 is a subset of Q . This is the initial state. The symbol Z is simply the stack start symbol. In this assignment I assigned the stack start to be "0". The last symbol F is the final state.

In the graph I attached, this npda accepts the language $L = \{(a^n)(b^n) : n > 0\} \cup \{b, bb\}$. In the graph, it starts off at Qs and by default 0 is pushed onto the stack. There are two paths that a user can go. If a user enters the letter 'a', the number 1 is pushed onto the stack and state moves to Q1. The user can also enter a 'b' in the start state. When a 'b' is entered, a λ is pushed onto the stack. The λ symbol is basically an empty spot. Once in state Q1, the user can enter any amounts of 'a's. Everytime the user enters an 'a', 1 is pushed onto the stack and the state stays in Q1. Once a user enters 'b', the top number on the stack is popped out.

Everytime the user enters ‘b’ in the Q2 state, they stay in that state and the top number in the stack is popped out. If a user enters ‘*’ at this state the npda can either go to state Q5 or Qt. The only way the ndpa can go to state Q5, is if the top of the stack is 0. When the top of the stack is 0 at this state, that means the the npda follows the language which is L = {(a^n)(b^n) : n > 0} ∪ {b, bb}. For example if a user is at state Q5, some strings would be {ab}, {aabb}, {aaabbb}, {aaaabbbb}, etc.

If a user is in state Q2 and enters ‘*’ they can go to the trap state, Qt. Strings can be sent to the trap state if the top of the stack has a 1 or if a user enters the character ‘a’ in state Q2. If the top of the stack has a 1, that means the string did not follow the language L = {(a^n)(b^n) : n > 0} ∪ {b, bb}. There is an uneven amount of ‘a’s or ‘b’s. Once in the trap state, the user cannot exit it. If they enter a or b, they are looped back into the state and cannot exit.

In the start of the state, the user can enter ‘b’. When entering ‘b’, this will send the user to a final state. The user can then enter another ‘b’ and be sent to another final state. After state Q4, if the user enters any letter, they will automatically be sent to the trap state.

In the code attached, the stack is created and 0 is pushed onto the stack in line 11. In line 13-14, the question is asked to enter a character. Based off the input, the code will either go to line 17 or line 87. If a user enters the character ‘a’ then the program will go to line 17. Once in line 17, a “1” is pushed onto the stack. In line 20, there is a while loop so users can enter a bunch of ‘a’s. As you can see in figure 11 and 12. In figure 11 there is multiple ones that are terminated by an ‘*’. In figure 12, there is a bunch of a’s with one ‘b’. The user can enter any amount of a’s they want. Once the user enters ‘b’ then the program skips to line 32. Once in line 32, everytime the user enters ‘b’ then the program will pop the top of the stack. There is a while loop in line 37 to loop the amount of ‘b’s entered. When a user enters ‘b’, then the stack is if its empty in line 43. This is needed because if the user enters a large amount of ‘b’s then the program will just crash because it would be popping an empty stack. If the stack is empty, an error message is outputted. This means there was too many ‘b’s entered. You can see this in figure 13. There is a limit of 3 b’s in figure 13. In line 52, this code is set here because once the user enters ‘b’, then they cannot go back to the character ‘a’. Figure 4, this shows what would happen when a user enters “aba”. This avoids the input “aba”. Line 62 is when the user enters the character ‘*’. When the user enters ‘*’ that's their symbol to mean they will stop entering a string. Once ‘*’ is entered, the stack is checked to see if its empty and checked if the top number is equal to 0. If the number is equal to 0, then that means there was an even amount of a’s and b’s. There are multiple figures showing even amount of a’s and b’s. In figure 1, 2, and 3 there are even amounts of a’s and b’s. If any of letters does not match, the string has been denied. Figure 5, and 6 shows a uneven amount of a’s and b’s.

In line 87, the program follows a regular DFA style program. If the first letter is ‘b’ then the program ask the user for the next character. The user can then end the string and enter ‘*’. If the user enters ‘*’, then the string is accepted. In figure 7, this shows when one ‘b’ is accepted. If a user enters an ‘a’ at this point the program will deny the string. This is clearly shown in figure 8. If the user enters another ‘b’ then they are asked to enter another character. The program will accepted ‘*’ again and the string will be accepted. Figure 9 shows this case perfectly. If the user enters another ‘b’ after his point the program will terminate. There can only be a maximum of 2 ‘b’s. As you can see in figure 10, this shows what would happen if 3 b’s are entered.

Q is a finite set of states of the control unit, initial state q0 and set of final states $F \subset Q$

Σ is the input alphabet

Q is a finite set called the states

Γ is the stack alphabet, has starting symbol z.

$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ -> finite subset of $Q \times \Gamma^*$

<u>This is based off the graph attached</u>

L = {(a^n)(b^n) : n > 0} ∪ {b, bb}

Q = {Qs, Q1, Q2, Q3, Q4, Q5, Qt}

Σ = {a, b, *}

Γ = {0,1}

Z = 0 (starting)

δ (Qs,a,0) = {(Q1,10)}

δ (Qs,b,0) = {(Q3,0)}

δ (Q3,b,0) = {(Q4,0)} // acceptd

δ (Q4,a,0) = {(Qt,10)} // denied. Cannot have “bba”

δ (Q4,b,0) = {(Qt,0)} // denied too many b’s

δ (Q1,a,1) = {(Q1,11)}

δ (Q1,b,1) = {(Q2, λ)} // The value on the stack gets popped out

δ (Q1,b,1) = {(Q1,11)}

δ (Q2,a,0) = {(Qt,10)} // denied

δ (Q2,*,0) = {(Q5,0)} // accepted

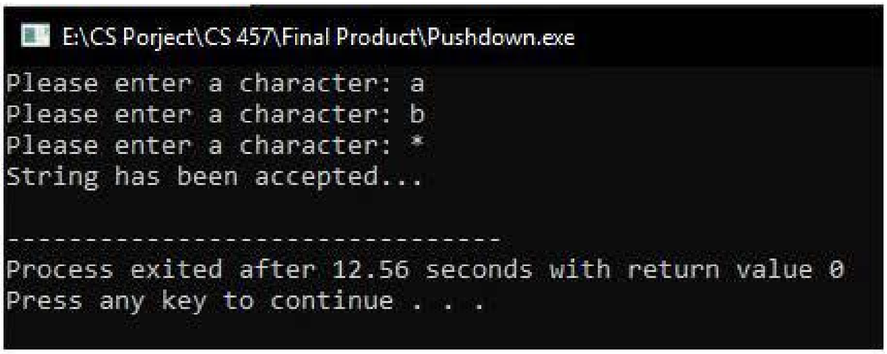
δ (Q2,*,1) = {(Qt,0)} //denied string

δ (Qt,a,1) = {(Qt,1)} // stuck in trap state

δ (Qt,b,1) = {(Qt, λ)} // stuck in trap state

Test Cases

ab*

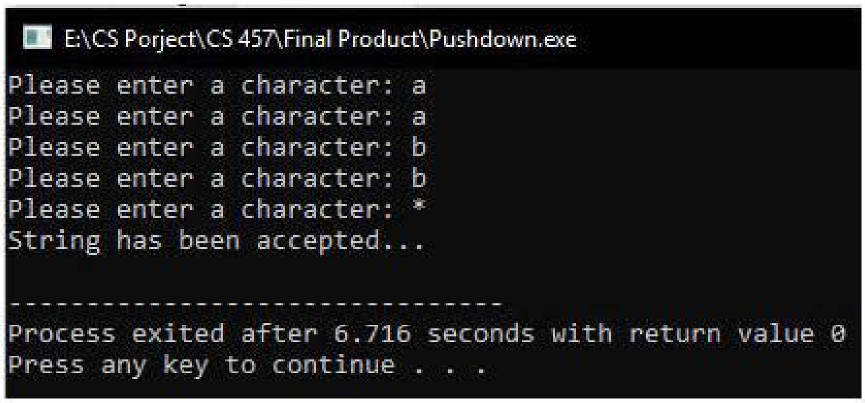


```
E:\CS Porject\CS 457\Final Product\Pushdown.exe
Please enter a character: a
Please enter a character: b
Please enter a character: *
String has been accepted...

--------------------------------
Process exited after 12.56 seconds with return value 0
Press any key to continue . . .
```

Figure 1

aabb*

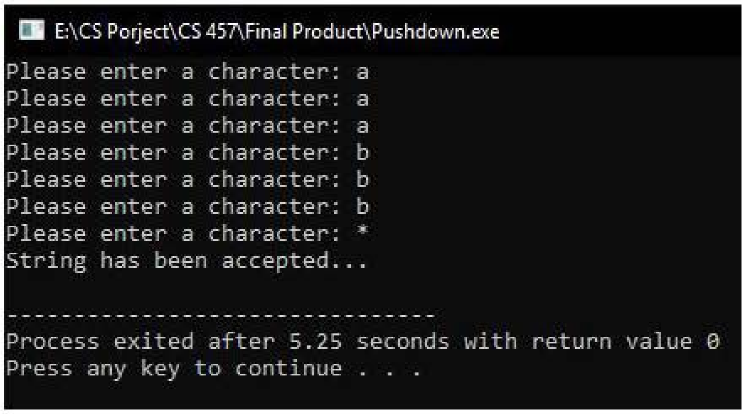


```
E:\CS Porject\CS 457\Final Product\Pushdown.exe
Please enter a character: a
Please enter a character: a
Please enter a character: b
Please enter a character: b
Please enter a character: *
String has been accepted...

--------------------------------
Process exited after 6.716 seconds with return value 0
Press any key to continue . . .
```

Figure 2

aaabbb*

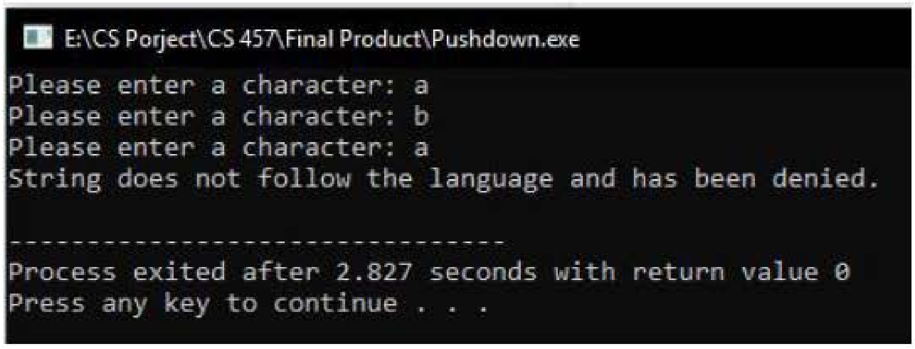


```
E:\CS Porject\CS 457\Final Product\Pushdown.exe
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: b
Please enter a character: b
Please enter a character: b
Please enter a character: *
String has been accepted...

--------------------------------
Process exited after 5.25 seconds with return value 0
Press any key to continue . . .
```

Figure 3

aba

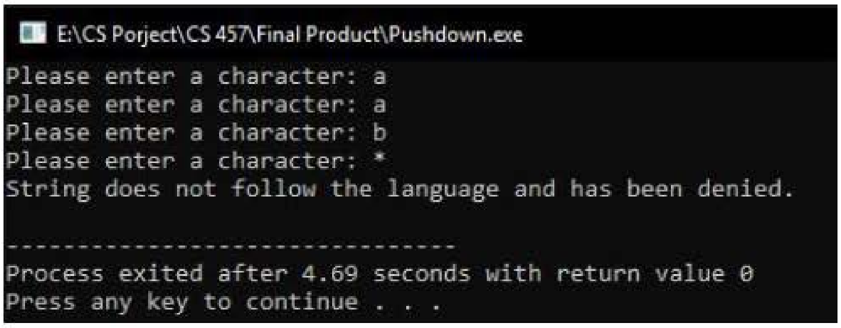


```
E:\CS Porject\CS 457\Final Product\Pushdown.exe
Please enter a character: a
Please enter a character: b
Please enter a character: a
String does not follow the language and has been denied.

--------------------------------
Process exited after 2.827 seconds with return value 0
Press any key to continue . . .
```

Figure 4

aab*

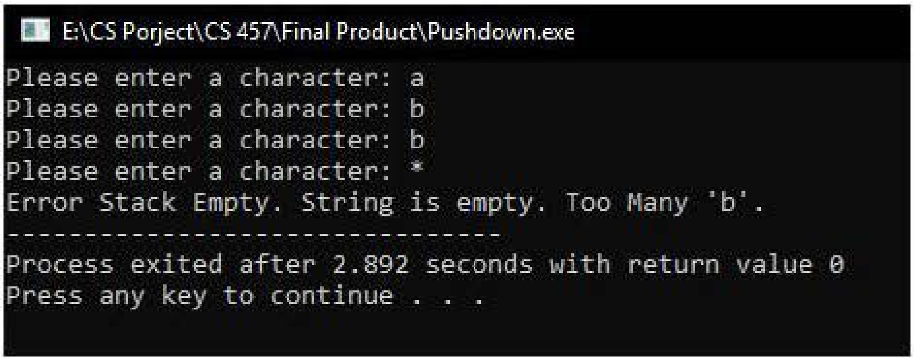


```
E:\CS Porject\CS 457\Final Product\Pushdown.exe
Please enter a character: a
Please enter a character: a
Please enter a character: b
Please enter a character: *
String does not follow the language and has been denied.

--------------------------------
Process exited after 4.69 seconds with return value 0
Press any key to continue . . .
```

Figure 5

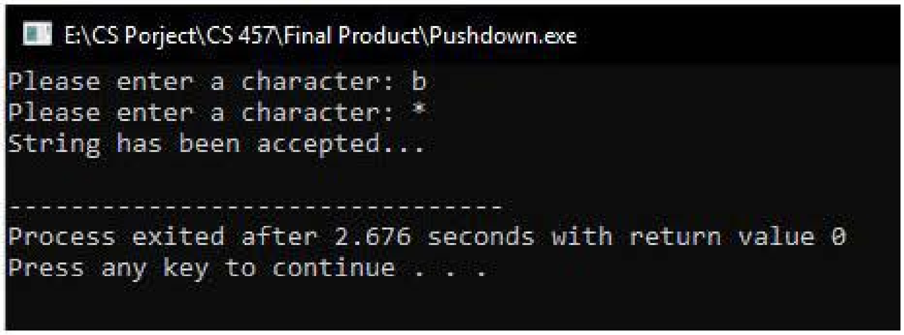
abb*



```
E:\CS Porject\CS 457\Final Product\Pushdown.exe
Please enter a character: a
Please enter a character: b
Please enter a character: b
Please enter a character: *
Error Stack Empty. String is empty. Too Many 'b'.
--------------------------------
Process exited after 2.892 seconds with return value 0
Press any key to continue . . .
```

Figure 6

b*

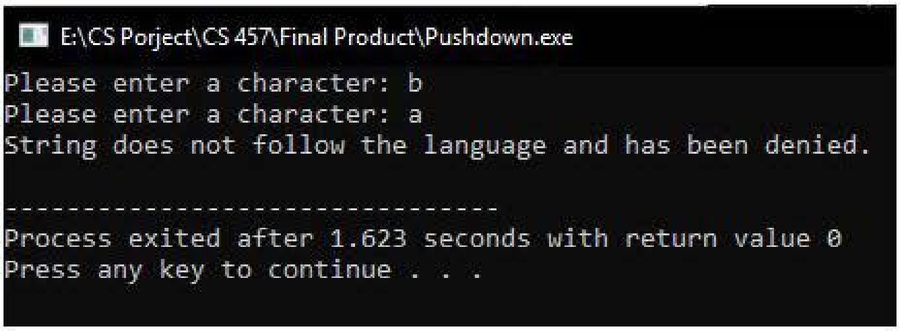


```
E:\CS Porject\CS 457\Final Product\Pushdown.exe
Please enter a character: b
Please enter a character: *
String has been accepted...

--------------------------------
Process exited after 2.676 seconds with return value 0
Press any key to continue . . .
```

Figure 7

ba*

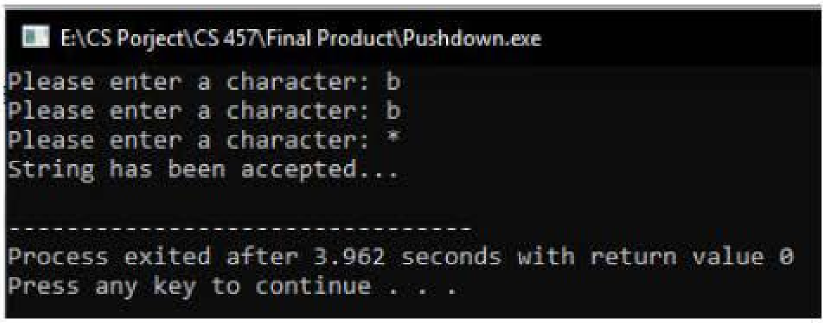


```
E:\CS Porject\CS 457\Final Product\Pushdown.exe
Please enter a character: b
Please enter a character: a
String does not follow the language and has been denied.

--------------------------------
Process exited after 1.623 seconds with return value 0
Press any key to continue . . .
```

Figure 8

bb*

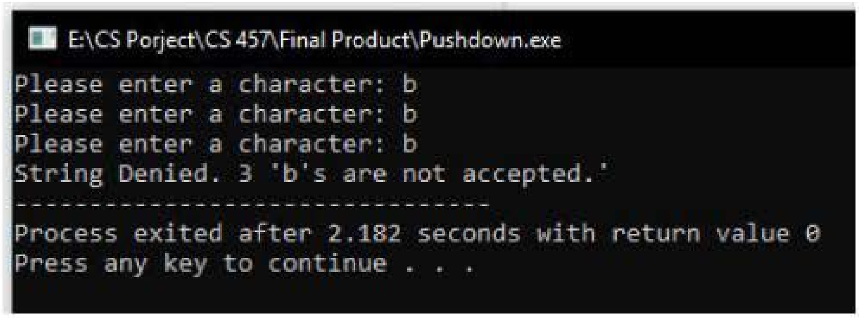


```
E:\CS Porject\CS 457\Final Product\Pushdown.exe
Please enter a character: b
Please enter a character: b
Please enter a character: *
String has been accepted...

--------------------------------
Process exited after 3.962 seconds with return value 0
Press any key to continue . . .
```

Figure 9

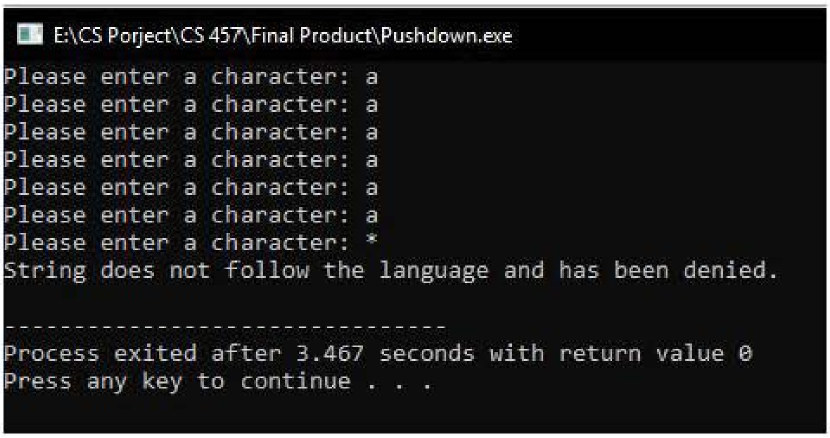
bbb



```
E:\CS Porject\CS 457\Final Product\Pushdown.exe
Please enter a character: b
Please enter a character: b
Please enter a character: b
String Denied. 3 'b's are not accepted.'
--------------------------------
Process exited after 2.182 seconds with return value 0
Press any key to continue . . .
```

Figure 10

Bunch of a's

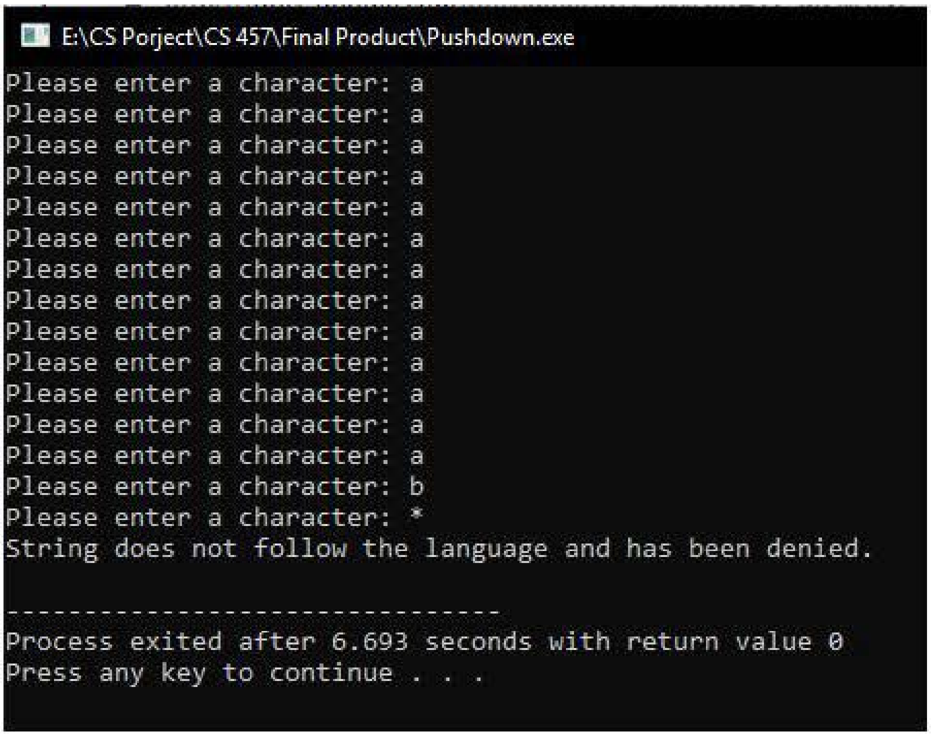


```
E:\CS Porject\CS 457\Final Product\Pushdown.exe
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: *
String does not follow the language and has been denied.

--------------------------------
Process exited after 3.467 seconds with return value 0
Press any key to continue . . .
```

Figure 11

Bunch of a's and one b



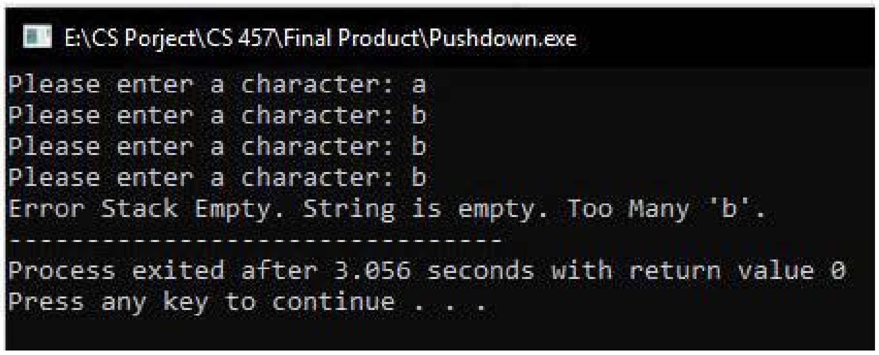
E:\CS Porject\CS 457\Final Product\Pushdown.exe

```
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: a
Please enter a character: b
Please enter a character: *
String does not follow the language and has been denied.

--------------------------------
Process exited after 6.693 seconds with return value 0
Press any key to continue . . .
```

Figure 12

One 'a' and multiple b's



E:\CS Porject\CS 457\Final Product\Pushdown.exe

```
Please enter a character: a
Please enter a character: b
Please enter a character: b
Please enter a character: b
Error Stack Empty. String is empty. Too Many 'b'.
--------------------------------
Process exited after 3.056 seconds with return value 0
Press any key to continue . . .
```

Figure 13

Overall this project was really enjoyable for me. I felt really inexperienced with using stacks. I was not understanding the concept of this topic but I figured it out after research. The website I listed below saved me so much time and was very clear in their explanation. It showed me a basic example of how stacks is used in c++ and defined all the terms they used. While doing this program, I found some new ways I could make the code shorter. I used while loops instead of nested loops. I could have put a limit to how many characters would be entered, like my professor did in his example, but I felt my code would be more efficient if there was no limit. My code is barebones and very simple to use. The program follows all the instructions based off the assignment. If I were to redo this assignment in the future, I would do this in linked list. I have never attempted that style of coding and would be a new challenge for me, just like how I learned how to use stacks in this assignment.

Works Cited

"Stack in C++ STL." GeeksforGeeks, 2 Nov. 2018, www.geeksforgeeks.org/stack-in-cpp-stl/.