Hacettepe University
Computer Science and Engineering Department

Name and Surname	**:** Ahmet Eren Akbaş
Identity Number	**:** 21945757
Course	**:** BBM203
Experiment	**:** Assignment 3
Subject	**:** Stack
Due Date	**:** 17.12.2021
Advisors	**:** Ahmet Alkılıç, Hayriye Çelikbilek, Merve Özdeş
e-mail	**:** b21945757@cs.hacettepe.edu.tr
Main Program	**:** Computer Science

2. Software Using Documentation

2.1. Software Usage

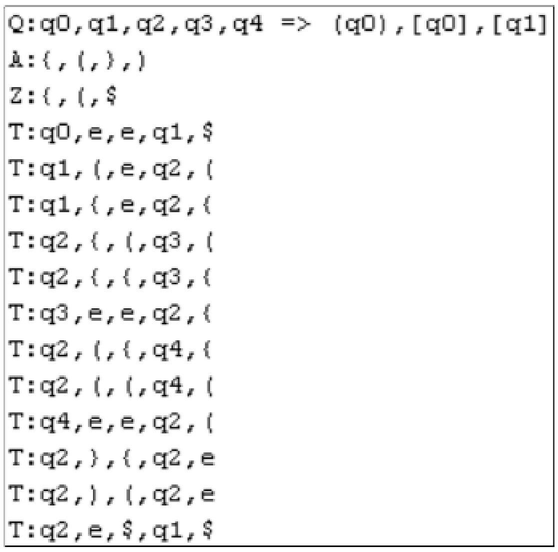
Software runs by compiling the code and giving the parameters of the program.

Here is an example:

```
> g++ -o Main *.cpp -std=c++11
>./Main dpda1.txt input1.txt output1.txt
```

The dpda1.txt file contains the information about the machine. It defines states, initial and final states, alphabet of the input, alphabet of the output and finally transitions.

Here is an example of dpda file:



```
Q:q0,q1,q2,q3,q4 => (q0),[q0],[q1]
A:{,(,},)
Z:{,(,$
T:q0,e,e,q1,$
T:q1,(,e,q2,(
T:q1,{,e,q2,{
T:q2,{,(,q3,(
T:q2,{,{,q3,{
T:q3,e,e,q2,{
T:q2,(,{,q4,{
T:q2,(,(,q4,(
T:q4,e,e,q2,(
T:q2,},{,q2,e
T:q2,),(,q2,e
T:q2,e,$,q1,$
```

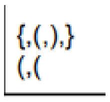
Firstly line starts with Q states are defined and after the arrow (=>) the state with the () parantheses is initial state and the states with the [] parantheses are final states.

Line starts with A defines the alphabet of input.

Line starts with Z defines the alphabet that will be used on stack.

Lines start with T defines the transitions. First parameter is initial state, second parameter is item to read in from input, third parameter is item to pop from stack, fourth parameter is next state, fifth parameter is item to push on stack.

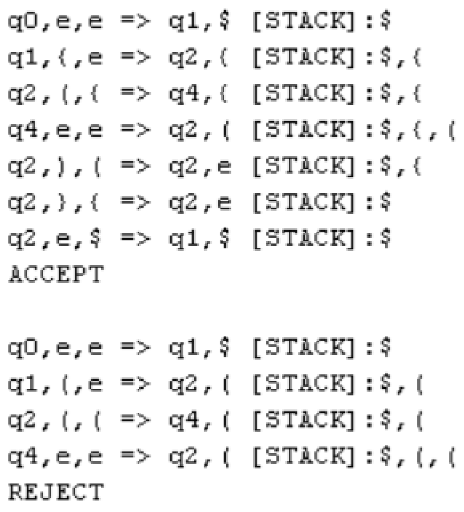
Here is an example of input1.txt:



```
{,(,),}
(,(
```

Each line is a writing to be checked whether it’s an acceptable writing.

Here is an example of output1.txt:



```
q0,e,e => q1,$ [STACK]:$
q1,{,e => q2,{ [STACK]:$,{
q2,(,{ => q4,{ [STACK]:$,{
q4,e,e => q2,( [STACK]:$,{,(
q2,),( => q2,e [STACK]:$,{
q2,},{ => q2,e [STACK]:$
q2,e,$ => q1,$ [STACK]:$
ACCEPT

q0,e,e => q1,$ [STACK]:$
q1,(,e => q2,( [STACK]:$,(
q2,(,( => q4,( [STACK]:$,(
q4,e,e => q2,( [STACK]:$,(,(
REJECT
```

It outputs each step and shows the stack. In the end it confirms the string or rejects the string.

2.3 Error Messages

There is only one message:

Error message: “Error [1]:DPDA description is invalid!”

Error conditions: Is there any transition that it’s parameters doesn’t match with the input alphabet, output alphabet or states that are defined in the beginning.

Solutions to overcome: Creating a correct dpda file.

3.Software Design Notes

3.1. Desctiption of the program

3.1.1. Problem

Writing a program that checkes whether a writing is correct or incorrect.
The alphabet of that writing and conditions will be independent, in other words user will define it.

3.1.2.Solution

I found two solutions to overcome this problem.

First one is:
Creating a transition class that contains initial state, item to read, item to pop, next state, item to push.
In this solution we traverse the input and create the transitions. After that we start to read input.txt and execute it.
Disadvantages: We have to traverse all of the transitions to find correct transition every time without any logic. For me it seems a little simple, non-sustainable.
Advantages: There is only one class called transitions and it's easier to implement.

Second one is:
Like the first one creating a transition class but this time without initial state. Creating a state class that contains transitions.
In this solution we do the same things but in creating the states we attach the transition to the related state. After that we start to read input.txt and execute it.
Disadvantages: In creation part time complexity is a little more than the first solution because in each transition we find the related state and attach it there is a search.
Advantages: In execution part we don't need to traverse all of the transitions to find correct transition, we just iterate through the current state's transition. It has a better time complexity in execution.

I chose the second one because we only have disadvantage in creation part. After that if there are more writings to check, it will work in much better performance than the first one.

3.2. Algorithm

1. Make initialisation.

   1.1. Initialize these structures to hold values.:

```
vector<State> states;
string initState;
vector<string> finalStates;
vector <string> inputAlphabet;
vector <string> outputAlphabet;
```

   1.2. Open input: ‘dpda1.txt” and fill the datas in 1.1. If there is a problem then open Output: ‘output.txt” and write error message.

2. Create necessary functions to avoid repeating in the code.

3. For every line in ‘input.txt’
   2.1 Create the stack.
   2.2. Start from current state=initial state.
   2.3. Iterate through state’s transition and find the related transition.
   2.4. Process the transition on stack.
   2.5. Print the step.
   2.6. Go to transition’s next state by updating the current state.
   2.7. If no transition found then print reject.
   2.8. If the input is processed and the last state we end up with is in final states then print ACCEPT if not then print REJECT.

4. Close file.

3.5. Special Design Properties

I don’t have a new approach to the problem.

4. SOFTWARE TESTING NOTES :

4.1. Bugs and Software Reliability

In my software there is no precedence in transitions so it can lead to different output maybe. For example: If there is “(“ character on top of stack and there is two related transitions one of them contains “e” for item to read and other one have “(“ this, my program can select the first one and lead to different output.

4.2. Comments

In my opinion the assignment was hard to understand the problem at first but it was very taughtful. I learnt a lot of things.

REFERENCES

https://bilgisayarkavramlari.com/2009/09/15/push-down-automata/