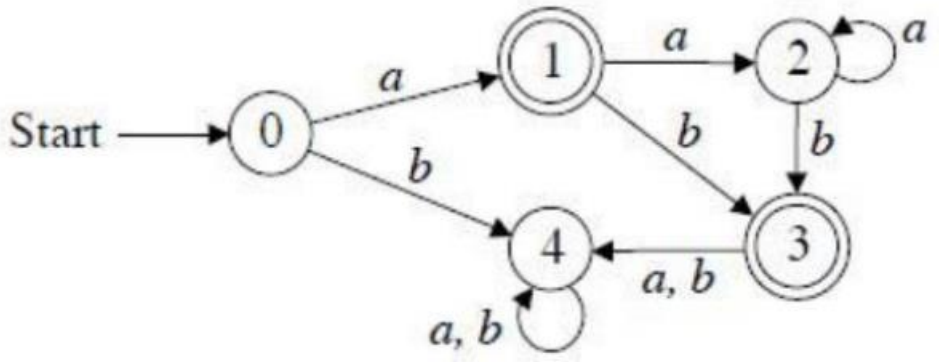
Lab -6

WAP to construct DFA for Regular Expression (a+aa*b)*.

Theory

This program implements a Deterministic Finite Automaton (DFA) to recognize strings generated by the regular expression (a+aa∗b)∗(a + aa*b)^*(a+aa∗b)∗. The DFA accepts strings that consist of sequences of 'a's followed optionally by 'b's, adhering to the specified pattern. The user inputs a string, and the program determines whether it belongs to the language defined by the regular expression.

Transition Diagram



Program:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main() {
   int table[2][2], i, l, status = 0;
   char input[100];
```

```
    printf("To implement DFA of language (a + aa*b)*, Enter Input String: ");
    table[0][0] = 1;
    table[0][1] = -1;
    table[1][0] = 1;
    table[1][1] = 0;
    scanf("%s", input);
    l = strlen(input);

    for (i = 0; i < l; i++) {
        if (input[i] != 'a' && input[i] != 'b') {
            printf("The entered value is wrong");
            exit(0);
        }
        if (input[i] == 'a')
            status = table[status][0];
        else
            status = table[status][1];
        if (status == -1) {
            printf("String not accepted");
            break;
        }
    }

    if (i == l)
        printf("String accepted");

    return 0 ;
}
```

Output:

```
To implement DFA of language (a + aa*b)*,
 Enter Input String: aabab
String accepted
```

```
To implement DFA of language (a + aa*b)*,
 Enter Input String: abbababa
String not accepted
```

Conclusion:

The program constructs a Deterministic Finite Automaton to recognize strings according to the regular expression (a+ aa∗b)*.

Lab-7:

Design a Push Down Automata (PDA) that accepts all string having equal number of 0's and 1's over input symbol {0, 1} for a language 0^n1^n where n >= 1.

THEORY

As per the AIM, set of valid strings that can be generated by given language is represented in set A:

A = {01, 0011, 000111, ...}

means all string having n number of 0's followed by n numbers of 1's where n can be any number greater than equal tp and count of 0's must be equal to count of 1's. Block diagram of push down automata is shown in Figure below.

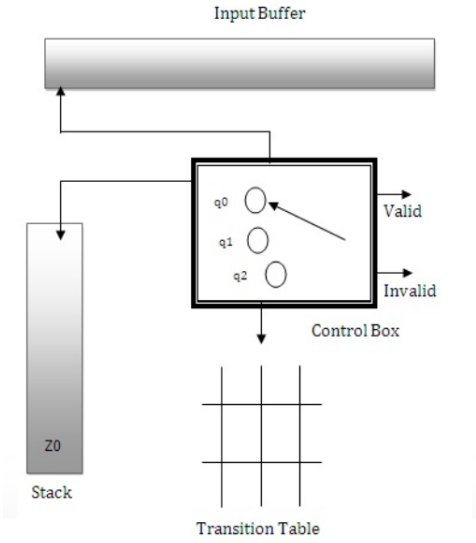
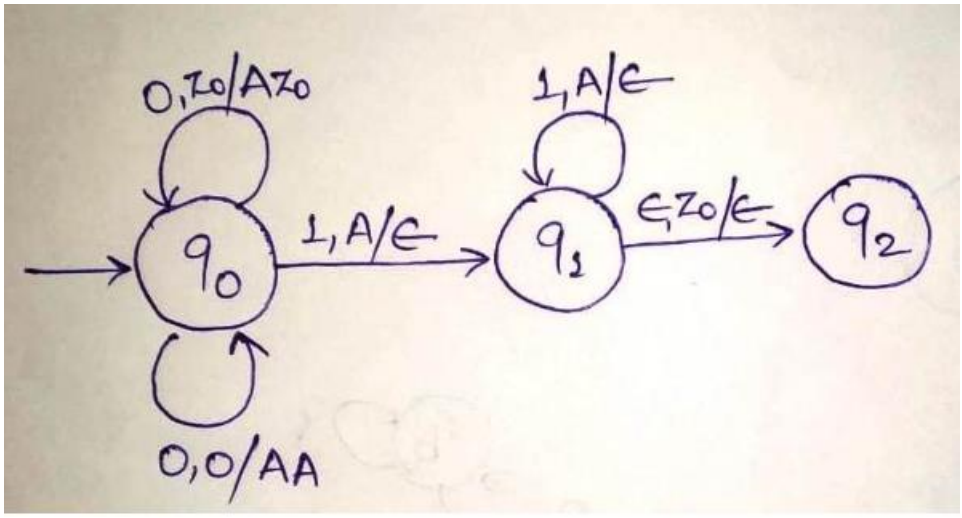


Figure 1: Block Diagram of Push Down Automata.

Transition Diagram



Transition Diagram

ID

$$\delta(q0, 0, Z0) \rightarrow (q0, AZ0)$$
$$\delta(q0, 0, 0) \rightarrow (q0, 00)$$
$$\delta(q0, 1, 0) \rightarrow (q1, \varepsilon)$$
$$\delta(q1, 1, 0) \rightarrow (q1, \varepsilon)$$
$$\delta(q1, \varepsilon, Z0) \rightarrow (q2, \varepsilon)$$

PROGRAM

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main() {
    string input;
    vector<char> stack;
    stack.push_back('Z');

    cout << "Enter binary string to validate (input string should be of 0 and 1): ";
    cin >> input;

    size_t i = 0;
    while (i < input.length()) {
        char currentChar = input[i];

        if (currentChar == '0') {
            if (stack.back() == 'Z' || stack.back() == 'A') {
                stack.push_back('A');
            } else {
                cout << "\nOutput: Invalid String\n";
                return 0;
            }
        } else if (currentChar == '1') {
            if (stack.back() == 'A') {
                stack.pop_back();
            } else {
```

```
            cout << "\nOutput: Invalid String\n";
            return 0;
        }
    } else {
        cout << "\nOutput: Invalid String\n";
        return 0;
    }
    i++;
  }

  if (stack.size() == 1 && stack.back() == 'Z') {
    cout << "\nOutput: Valid String\n";
  } else {
    cout << "\nOutput: Invalid String\n";
  }

  return 0;
}
```

Output:

```
Enter binary string to validate (input string should be of 0 and 1): 000111

Output: Valid String
```

```
Enter binary string to validate (input string should be of 0 and 1): 101000

Output: Invalid String
```

Conclusion:

The program successfully validates binary strings that consist of any number of '0's followed by an equal number of '1's, demonstrating the principles of finite automata and stack operations in string validation.

Lab-8

Design a PDA to accept WCW^R where w is any binary string and W^R is reverse of that string and C is a special symbol.

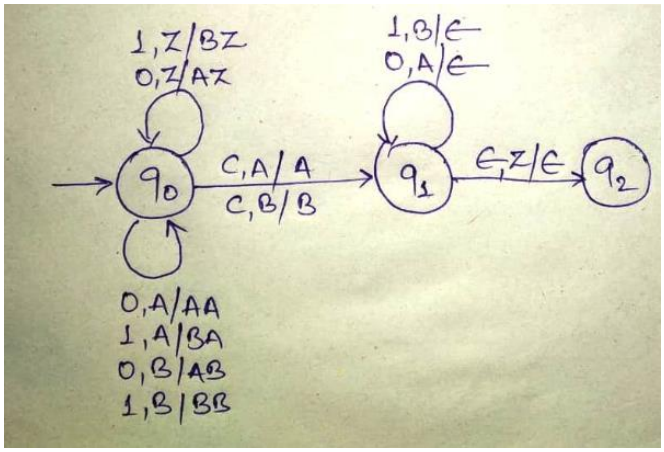
Theory

As per the AIM, set of valid strings that can be generated by given language is represented in set A:
A = {0C0, 1C1, 011000110C011000110, 101011C110101, ...}

Input string can be valid or invalid, valid if the input string follow set A (define above). PDA has to determine whether the input string is according to the language or not.

Let M be the PDA machine for above AIM, hence it can be define as M(Q, Σ, Γ, δ, q0, Z0, F) where
Q: set of states: {q0, q1, q2}
Σ: set of input symbols: {0, 1, C}
Γ: Set of stack symbols: {A, B, Z}
q0: initial state (q0)
Z0: initial stack symbol (Z)
F: set of Final states:
δ: Transition Function:

transition diagram



ID

$\delta(q0, 0, Z) \rightarrow (q0, AZ)$
$\delta(q0, 1, Z) \rightarrow (q0, BZ)$
$\delta(q0, 0, A) \rightarrow (q0, AA)$
$\delta(q0, 0, B) \rightarrow (q0, AB)$
$\delta(q0, 1, A) \rightarrow (q0, BA)$
$\delta(q0, 1, B) \rightarrow (q0, BB)$
$\delta(q0, C, A) \rightarrow (q1, A)$
$\delta(q0, C, B) \rightarrow (q1, B)$
$\delta(q1, 0, A) \rightarrow (q1, \varepsilon)$
$\delta(q1, 1, B) \rightarrow (q1, \varepsilon)$
$\delta(q1, \varepsilon, Z) \rightarrow (q2, \varepsilon)$

PROGRAM:

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
   char Input[100];
   char stack[100];
   int Top = -1;

   cout << "Enter string to be validated (format: wCwR):\n";
   cin.getline(Input, 100);
   stack[++Top] = 'Z';
   int i = 0;

   while (Input[i] != '\0')
   {
      if (Input[i] == '0' || Input[i] == '1')
      {
         stack[++Top] = Input[i];
      }
      else if (Input[i] == 'C')
      {
         break;
      }
      else
      {
         cout << "\nOutput: Invalid String";
         return 0;
      }
      i++;
   }

   i++;
   while (Input[i] != '\0')
   {
      if (Top == -1)
      {
         cout << "\nOutput: Invalid String";
         return 0;
      }
      if (Input[i] == stack[Top])
      {
         Top--;
      }
      else
      {
         cout << "\nOutput: Invalid String";
         return 0;
      }
```

```
        i++;
    }

    if (Top == 0 && stack[Top] == 'Z')
    {
        cout << "\nOutput: Valid String";
    }
    else
    {
        cout << "\nOutput: Invalid String";
    }

    return 0;
}
```

Output:

```
Enter string to be validated (format: wCwR):
0110C0110

Output: Valid String
```

```
Enter string to be validated (format: wCwR):
0011C00

Output: Invalid String
```

Conclusion:

The program validates strings in the format WCW^R ensuring that the string before C is the same as the reverse of the string after C.

Lab - 9

Design a Turing Machine that calculate 2's complement of given binary string.

THEORY

As per the AIM, Input can be any binary string and we have to design a turing machine that can calculate its two's complement.

For example (as shown in Figure 1) if the input binary string is 1011010000 then its two's complement will be 0100110000. For constructing a turing machine if we look in the logic then if we read the input string from right to left then the output string will remain exactly the same until the first 1 is found and as the first 1 encounter, complement of rest string appear in the output.

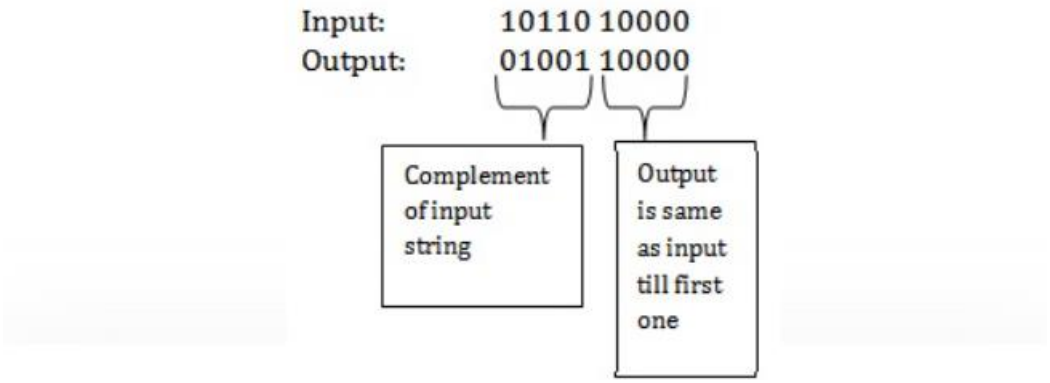


Figure 1: Input outpur relationship of 2's complement of binary string while reading right to left.

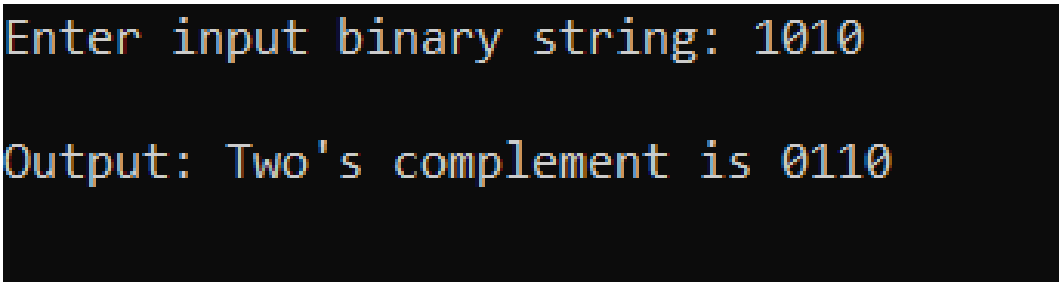
PROGRAM

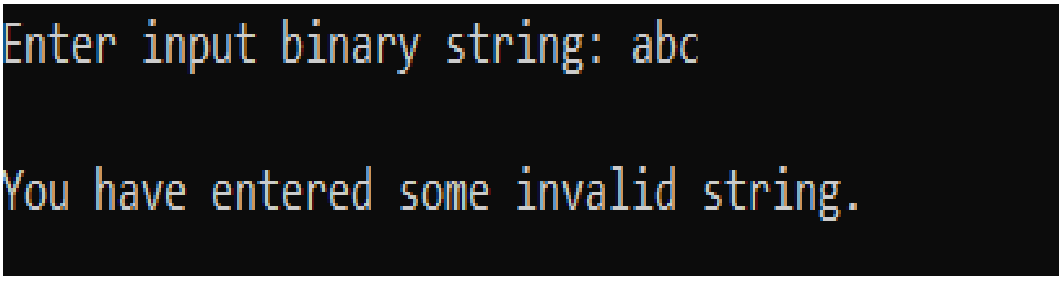
```
#include <iostream>
#include <conio.h>
#include <cstring>
using namespace std;

int main()
{
   char Input[100];
   clrscr();
   cout << "Enter input binary string: ";
```

```
    gets(Input);

    int i = 0;
q0:
    if (Input[i] == '0' || Input[i] == '1')
    {
        i++;
        goto q0;
    }
    else if (Input[i] == '\0')
    {
        i--;
        goto q1;
    }
    else
    {
        goto Invalid;
    }
q1:
    if (Input[i] == '0')
    {
        i--;
        goto q1;
    }
    else if (Input[i] == '1')
    {
        i--;
        goto q2;
    }
    else
    {
        goto Invalid;
    }
q2:
    if (Input[i] == '0')
    {
        Input[i] = '1';
        i--;
        goto q2;
    }
    else if (Input[i] == '1')
    {
        Input[i] = '0';
        i--;
        goto q2;
    }
    else if (i == -1)
    {
        goto q3;
    }
    else
    {
```

```
        goto Invalid;
    }
q3:
    cout << "\nOutput: Two's complement is " << Input << endl;
    goto exit;
Invalid:
    cout << "\nYou have entered some invalid string." << endl;
    goto exit;
exit:
    getch();
    return 0;
}
```

Output:





Conclusion:

The program successfully calculates the two's complement of a given binary string, ensuring the input is valid.

LAB – 10

Design a Turing Machine that's accepts the following language $a^n b^n c^n$ where n>0.

THEORY

A Turing Machine (TM) is a theoretical computational model that manipulates symbols on an infinite tape according to a set of rules. It is capable of simulating any algorithm and is fundamental in understanding the limits of computability. This lab focuses on constructing a Turing Machine that recognizes strings consisting of equal numbers of aaa's, bbb's, and ccc's, adhering to the order where all aaa's precede bbb's, which in turn precede ccc's.

PROGRAM

```
#include <iostream>
#include <cstring>
using namespace std;

bool checkLanguage(const char* input) {
   int countA = 0, countB = 0, countC = 0;
   int len = strlen(input);


   for (int i = 0; i < len; i++) {
      if (input[i] == 'a') {
         countA++;
      } else if (input[i] == 'b') {
         // Stop counting a's when we hit b's
         break;
      }
   }


   for (int i = countA; i < len; i++) {
      if (input[i] == 'b') {
         countB++;
      } else if (input[i] == 'c') {
         // Stop counting b's when we hit c's
         break;
      }
   }
```

```
    for (int i = countA + countB; i < len; i++) {
        if (input[i] == 'c') {
            countC++;
        }
    }


    return (countA == countB && countB == countC && countA > 0);
}

int main() {
    char input[100];
    cout << "Enter a string of the form a^n b^n c^n: ";
    cin >> input;

    if (checkLanguage(input)) {
        cout << "Output: Valid String" << endl;
    } else {
        cout << "Output: Invalid String" << endl;
    }

    return 0;
}
```

OUTPUT

```
Enter a string of the form a^n b^n c^n: aabbcc
Output: Valid String
```

```
Enter a string of the form a^n b^n c^n: aaabbbcc
Output: Invalid String
```

Conclusion:

This program effectively validates whether the input string is of the form $a^n b^n c^n$ for n>0 by ensuring that all character counts are equal and positive.