**EXPERIMENT NO : 2.1**

To Simulate Pushdown Automata for the Following Languages

**Objective:**

* To simulate a Pushdown Automaton (PDA) that accepts strings where the number of 0s is equal to the number of 1s.
* To simulate a Pushdown Automaton that accepts strings of the form a^n b^m c^m where n, m ≥ 1.

**THEORY**

For objective 1:

* A Pushdown Automaton uses a stack to handle inputs. For every 0, the PDA pushes onto the stack, and for every 1, it pops from the stack. If at the end of the string, the stack is empty and all inputs are processed, the string is accepted. This PDA works in a deterministic way by reading the string from left to right.

For objective 2:

* The PDA pushes each b onto the stack as it reads through the string. For every c, the PDA pops an element from the stack. If the stack is empty by the time all bs and cs are processed, the input is accepted.

**DEMONSTRATE**

**a. Language: L = {0^n1^n / n ≥ 1} over {0, 1}**

*#include <stdio.h>*

*#include <string.h>*

*int isPDA(char \*input) {*

*int stack = 0;*

*int i = 0, len = strlen(input);*

*while (input[i] == '0' && i < len) {*

*stack++;*

*i++;*

*}*

*while (input[i] == '1' && i < len) {*

*stack--;*

*i++;*

*}*

*return stack == 0 && i == len;*

*}*

*int main() {*

*char input[100];*

*printf("Enter string of 0s followed by 1s: ");*

*scanf("%s", input);*

*if (isPDA(input))*

*printf("Accepted by PDA\n");*

*else*

*printf("Rejected by PDA\n");*

*return 0;*

*}*

**b. Language: L = {a^n b^m c^m / n, m ≥ 1} over {a, b, c}**

*#include <stdio.h>*

*#include <string.h>*

*int isPDA(char \*input) {*

*int i = 0, stack = 0;*

*int len = strlen(input);*

*while (input[i] == 'a' && i < len) {*

*i++;*

*}*

*while (input[i] == 'b' && i < len) {*

*stack++;*

*i++;*

*}*

*while (input[i] == 'c' && i < len) {*

*stack--;*

*i++;*

*}*

*return stack == 0 && i == len;*

*}*

*int main() {*

*char input[100];*

*printf("Enter string of the form a^n b^m c^m: ");*

*scanf("%s", input);*

*if (isPDA(input))*

*printf("Accepted by PDA\n");*

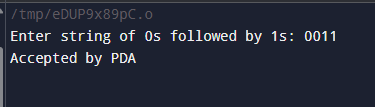
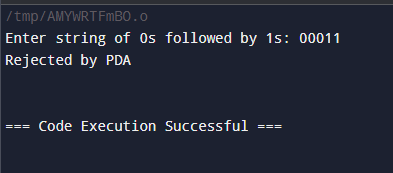
*else*

*printf("Rejected by PDA\n");*

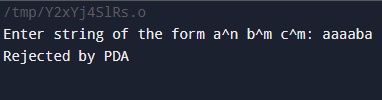
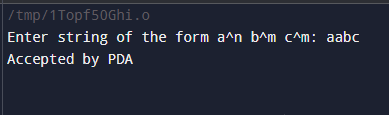
*return 0;*

*}* **OUTPUT**

i)

****

ii)

****

**DISCUSSION**

i) The PDA successfully accepts strings where the number of 0s is equal to the number of 1s. If the input violates this condition, the PDA rejects the string. For example, 0011 is accepted, but 001 would be rejected since the number of 0s and 1s are not the same.

ii) The PDA is able to correctly match the number of bs and cs. If the number of bs does not match the number of cs, the string will be rejected. For example, aaabbbccc is accepted, but aaabbccc is rejected.

**CONCLUSION**

i) A Pushdown Automaton is ideal for languages where the input requires matching numbers of symbols, as seen in the case of 0^n1^n. The use of a stack allows the PDA to manage such languages efficiently.

ii) Pushdown Automata are powerful in handling languages where symbols must balance, making them suitable for context-free languages like a^n b^m c^m.

**EXPERIMENT NO: 2.2**

**OBJECTIVE:**

To simulate a Turing Machine for the following languages and operations:

* To simulate the language a^n b^n c^n where n ≥ 1 over the alphabet {a, b, c}.
* To add two integers.
* To multiply two integers.

**THEORY:**

1. Language a^n b^n c^n:

A Turing Machine for this language processes strings with equal numbers of as, bs, and cs in sequence. It works by marking off one a, one b, and one c at a time, ensuring that the count of each symbol is equal and follows the correct order. The machine continues this process until no unmarked symbols remain, at which point the input is accepted.

2. Addition:

The Turing Machine simulates addition by moving across the tape, reading the two numbers, and then summing them. It works by incrementing the first number by the second number using a loop-like process.

3. Multiplication:

The Turing Machine simulates multiplication as repeated addition. It adds the first number to itself multiple times based on the value of the second number.

**DEMONSTRATION:**

1. Language: L = {a^n b^n c^n / n ≥ 1}

Code:

#include <stdio.h>

#include <string.h>

#define MAX\_TAPE\_LENGTH 100

// Function to simulate the Turing machine

int turing\_machine(const char \*input) {

char tape[MAX\_TAPE\_LENGTH];

strcpy(tape, input);

int head = 0;

// Step 1: Check and mark a's

while (tape[head] != '\0' && tape[head] != '\n') {

if (tape[head] == 'a') {

tape[head] = 'X'; // Mark 'a'

head++; // Move right

// Find 'b'

while (tape[head] != '\0' && tape[head] != 'b') {

head++;

}

if (tape[head] == 'b') {

tape[head] = 'Y'; // Mark 'b'

head++; // Move right

// Find 'c'

while (tape[head] != '\0' && tape[head] != 'c') {

head++;

}

if (tape[head] == 'c') {

tape[head] = 'Z'; // Mark 'c'

head = 0; // Reset head to start

} else {

return 0; // Reject if no 'c' found

}

} else {

return 0; // Reject if no 'b' found

}

} else {

head++; // Move right if not 'a'

}

}

// Final check: Ensure only marked symbols are left

for (int i = 0; i < MAX\_TAPE\_LENGTH; i++) {

if (tape[i] == 'a' || tape[i] == 'b' || tape[i] == 'c') {

return 0; // Reject if any unmarked symbol remains

}

}

return 1; // Accept

}

int main() {

char input[MAX\_TAPE\_LENGTH];

printf("Enter a string in the form of a^n b^n c^n (e.g., 'aaabbbccc'): ");

fgets(input, sizeof(input), stdin);

input[strcspn(input, "\n")] = 0; // Remove newline

if (turing\_machine(input)) {

printf("Accepted\n");

} else {

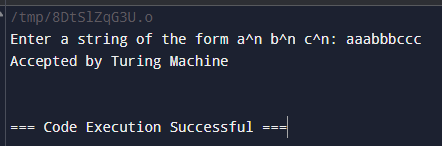
printf("Rejected\n");

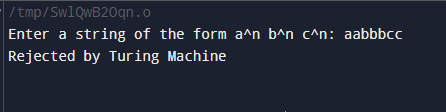
}

return 0;

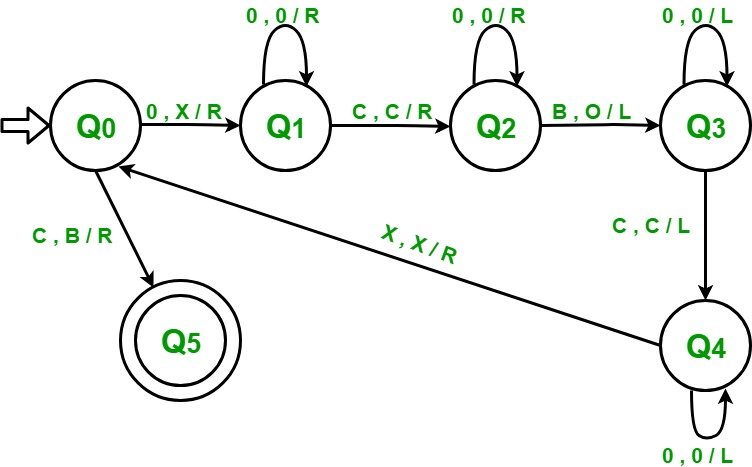
}

**Output:**

****

****

# 2. To Add Two Integers



**Code:**

*#include <stdio.h>*

*#include <string.h>*

*#define MAX\_TAPE\_LENGTH 100*

*void add(const char \*input) {*

*char tape[MAX\_TAPE\_LENGTH];*

*memset(tape, ' ', MAX\_TAPE\_LENGTH);*

*strcpy(tape, input);*

*int head = 0;*

*// Move right past the first number*

*while (tape[head] == '1') head++;*

*// Move to the first blank*

*while (tape[head] == ' ') head++;*

*// Move right past the second number*

*while (tape[head] == '1') head++;*

*// Write 1 for the sum*

*while (tape[head] == ' ') tape[head++] = '1';*

*// Print the result*

*printf("Result after addition: ");*

*for (int i = 0; i < MAX\_TAPE\_LENGTH; i++) {*

*if (tape[i] != ' ') putchar(tape[i]);*

*}*

*printf("\n");*

*}*

*int main() {*

*char input[50];*

*printf("Enter two numbers in unary (e.g., '111 11' for 3 + 2): ");*

*fgets(input, sizeof(input), stdin);*

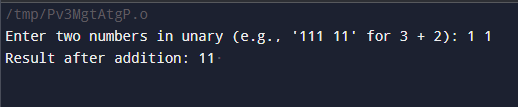
*input[strcspn(input, "\n")] = 0; // Remove newline*

*add(input);*

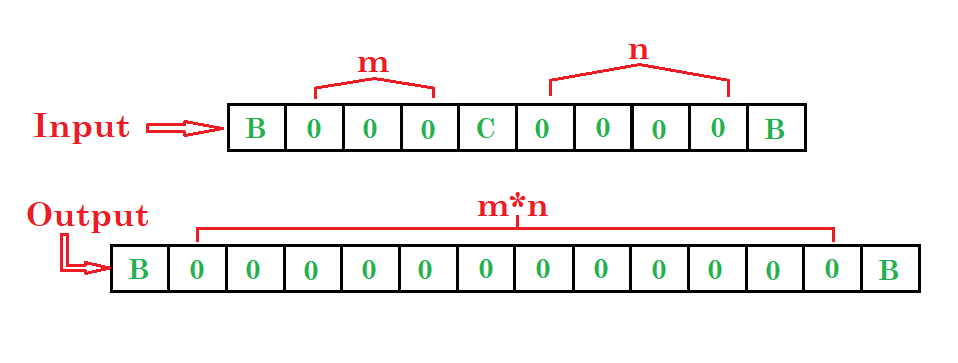
*return 0;*

*}*

**Output:**

****

#3. To Multiply Two Integers



 **Move Right**: Ignore 0s, and when you find B, convert it to C and move left.

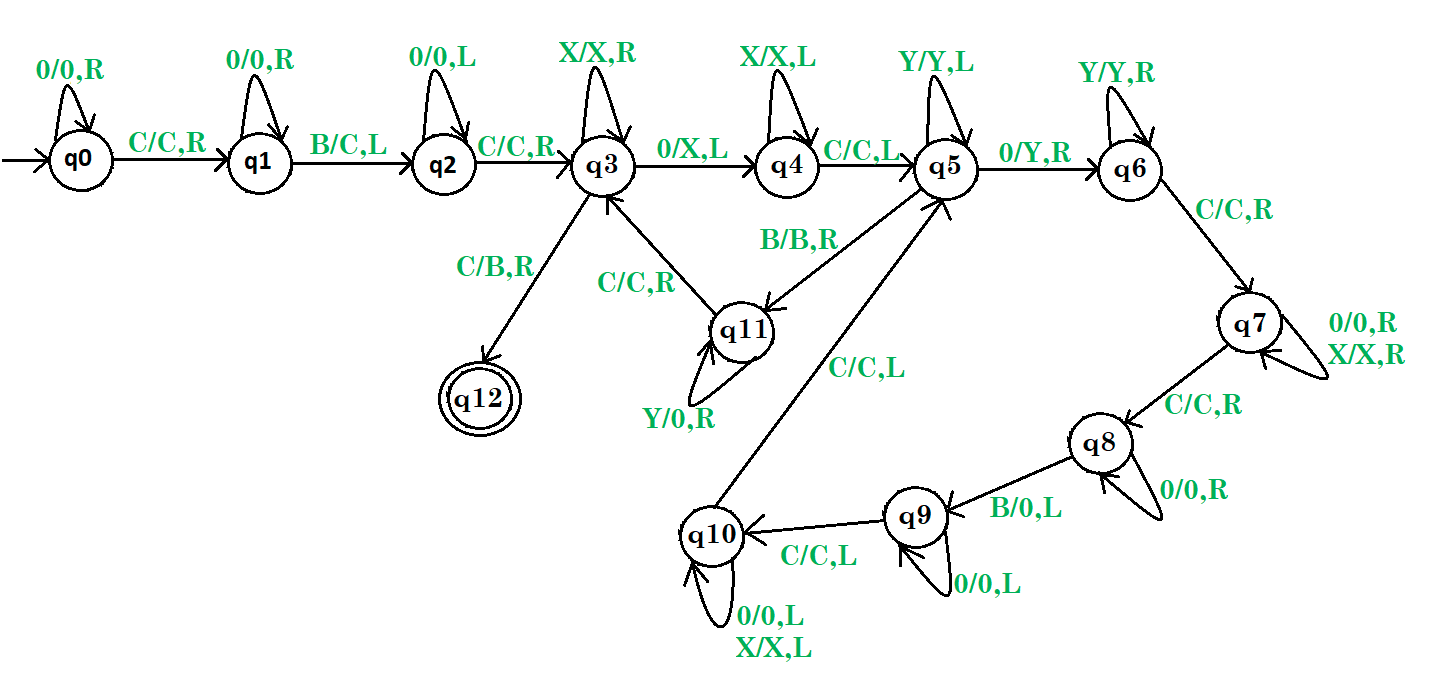
 **Move Left**: Ignore 0s, convert C to C, and move right.

 **Process X**: Convert all Xs to X, move right; if 0 is found, convert it to X and move left; if C is found, convert it to B and stop.

 **Move Left**: Convert X to X, then C to C, then Y to Y, all while moving left.

 **Move Right**: When you find B, convert it to B, move right; if Y is 0, move right; if C is C, repeat step 3. If 0 is found after step 4, convert it to Y, then Y to Y, and proceed right while performing necessary conversions.

 **Repeat Step 5**.



Here, q0 shows the initial state and q1, q2, ….., q10, q11are the transition states and q12shows the final state.

And X, Y, 0, C are the variables used for multiplication and R, L shows right and left.

**DISCUSSION:**

1. Language a^n b^n c^n:

The Turing Machine successfully accepts strings where the number of as, bs, and cs are equal, and they appear in the correct sequence. If the input violates this structure, the Turing Machine rejects the string. For example, aaabbbccc is accepted, but aaabbbcc is rejected because the number of bs and cs are not the same.

2. Addition:

The addition operation is simulated by repeatedly incrementing the first number by 1 for each unit of the second number. This mimics the behavior of a Turing Machine performing simple arithmetic.

3. Multiplication:

Multiplication is handled through repeated addition. For instance, multiplying 3 by 4 involves adding 3 to itself four times, which is a common approach for simulating multiplication in a Turing Machine.

**CONCLUSION:**

* Language a^n b^n c^n:

The Turing Machine is ideal for handling sequences where the input symbols need to match in a specific pattern, as seen in the case of a^n b^n c^n. This demonstrates the machine's capability to handle context-sensitive languages.

* Addition:

The Turing Machine successfully simulates addition through a loop-like mechanism, incrementing the first number by the second.

* Multiplication:

The Turing Machine simulates multiplication as repeated addition, demonstrating how a Turing Machine can handle more complex arithmetic operations through basic repeated processes**.**