

Laboratory Works for TOC

LAB 1 : Write a program to find prefixes, suffixes and substring from given string.

Code:

```
/* To find substring, prefix, suffix of a string */
#include<stdio.h>
#include<string.h>
void find_prefix(char string[]);
void find_suffix(char string[]);
void find_substring(char string[],int,int);

int main()
{
    char string[20];
    int i,j;
    printf("\n Enter a string\t");
    gets(string);

    printf("\n Prefixes:");
    find_prefix(string);
    printf("\n Suffixes");
    find_suffix(string);

    printf("\nEnter i and j for substring");
    scanf("%d%d",&i,&j);
    find_substring(string,i,j);

    return 0;
}

void find_prefix(char string[])
{
    int i,j;
    char prefix[20];
    for(i=strlen(string);i>=0;i--)
    {
        for( j = 0; j<i;j++)
        {
            prefix[j]= string[j];
        }
        prefix[j]='\0';
        printf("\n %s",prefix);
    }
}
```

```

    }
}
void find_suffix(char string[])
{
    int i,j,k;
    char suffix[20];
    for(i=0;i<=strlen(string);i++)
    {
        k = i;
        for( j = 0; j<strlen(string);j++)
        {
            suffix[j]= string[k];
            k++;
        }
        suffix[j]='\0';
        printf("\n %s",suffix);
    }
}

void find_substring(char string[],int x, int y)
{
    char substr[20];

    int k=0;
    for(int i=x-1;i<y;i++)
    {
        substr[k]=string[i];
        k++;
    }
    substr[k]='\0';
    printf("\n Substring:\n%s",substr);
}

```

OUTPUT:

```
C:\Users\c\Desktop\lab1ststring.exe
computat
computa
comput
compu
comp
com
co
c

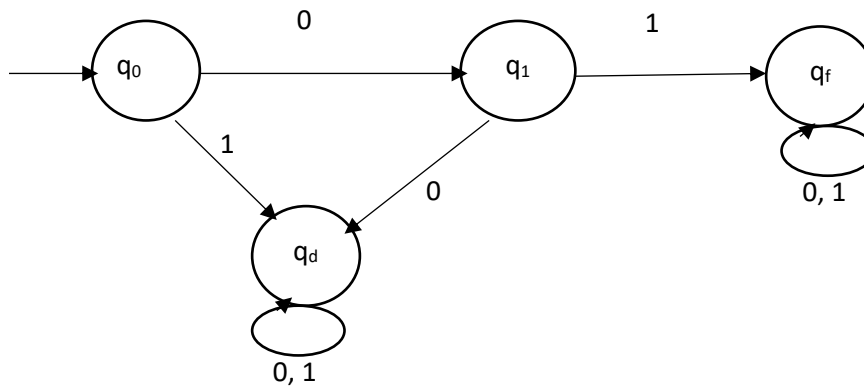
Suffixes
computation
omputation
mputation
putation
utation
tation
ation
tion
ion
on
n

Enter i and j for substring
2 5

Substring:
ompu
-----
Process exited after 61.09 seconds with return value 0
Press any key to continue . . .
```

LAB 2: Write program to implement following DFA's over alphabet $\Sigma = \{0, 1\}$.

- i. The DFA that accepts all the strings that start with 01.



Code

```
/* Implement a DFA for L = { set of all strings over {0,1} such that string start with 01 */
#include<stdio.h>
enum states { q0, q1, qf,qd};
enum states delta(enum states, char);

int main()
{
    char input[20];
    enum states curr_state = q0;
    int i =0;

    printf("\n Enter a binary string\t");
    gets(input);
    char ch = input[i];
    while( ch !='\0')
    {
        curr_state = delta(curr_state,ch);
        ch = input[++i];
    }

    if(curr_state == qf)
        printf("\n The string %s is accepted.",input);
    else
        printf("\n The string %s is not accepted.",input);
    return 0;
}
// Transition Function
enum states delta(enum states s, char ch)
{
    enum states curr_state;
    switch(s)
    {
```

```

        case q0:
            if(ch=='0')
                curr_state = q1;
            else
                curr_state = qd;
            break;
        case q1:
            if(ch=='1')
                curr_state = qf;
            else
                curr_state = qd;
            break;
        case qf:
            if(ch=='0')
                curr_state = qf;
            else
                curr_state = qf;
            break;
        case qd:
            if(ch=='0')
                curr_state = qd;
            else
                curr_state = qd;
            break;
    }
    return curr_state;
}

```

OUTPUT:

```

C:\Users\Resham\Desktop\TOC\toc lab\lab2copy.exe

Enter a binary string  01110

The string 01110 is accepted.
-----
Process exited after 6.659 seconds with return value 0
Press any key to continue . . .

```

C:\Users\Resham\Desktop\TOC\toc lab\lab2copy.exe

Enter a binary string 1101

The string 1101 is not accepted.

Process exited after 5.392 seconds with return value 0
Press any key to continue . . .

- ii. The DFA that accepts all the strings that end with 01.

$Q = \{ q_0, q_1, q_f \}$

start state = q_0 ,

Final state = q_f

Transition function, δ is defined as:

$\delta(q_0, 0) = q_1$

$\delta(q_0, 1) = q_0$

$\delta(q_1, 0) = q_1$

$\delta(q_1, 1) = q_f$

$\delta(q_f, 0) = q_1$

$\delta(q_f, 1) = q_0$

Code:

```
// Implement a DFA for L = { set of all strings over {0,1} such that string end with 01  
#include<stdio.h>
```

```
enum states { q0, q1, qf};  
enum states delta(enum states, char);
```

```
int main()  
{  
    char input[20];  
    enum states curr_state = q0;  
    int i =0;  
  
    printf("\n Enter a binary string\t");  
    gets(input);  
    char ch = input[i];  
    while( ch !='\0')  
    {  
        curr_state = delta(curr_state,ch);  
        ch = input[++i];  
    }  
}
```

```

    }

    if(curr_state == qf)
        printf("\n The string %s is accepted.",input);
    else
        printf("\n The string %s is not accepted.",input);

    return 0;
}

```

```

enum states delta(enum states s, char ch)
{
    enum states curr_state;
    switch(s)
    {
        case q0:
            if(ch=='0')
                curr_state = q1;
            else
                curr_state = q0;
            break;
        case q1:
            if(ch=='1')
                curr_state = qf;
            else
                curr_state = q1;
            break;
        case qf:
            if(ch=='0')
                curr_state = q1;
            else
                curr_state = q0;
            break;
    }
    return curr_state;
}

```

OUTPUT:

```
C:\Users\Resham\Desktop\TOC\toc lab\dfaend01.exe

Enter a binary string  11101

The string 11101 is accepted.
-----
Process exited after 4.533 seconds with return value 0
Press any key to continue . . . █
```

```
C:\Users\Resham\Desktop\TOC\toc lab\dfaend01.exe

Enter a binary string  111010

The string 111010 is not accepted.
-----
Process exited after 8.113 seconds with return value 0
Press any key to continue . . . █
```

iii. The DFA that accepts all the string that contains substring 001.

$Q = \{ q_0, q_1, q_2, q_f \}$

start state = q_0 ,

Final state = q_f

Transition function, δ is defined as:

$\delta(q_0, 0) = q_1$

$\delta(q_0, 1) = q_0$

$\delta(q_1, 0) = q_2$

$\delta(q_1, 1) = q_0$

$\delta(q_2, 0) = q_2$

$\delta(q_2, 1) = q_f$

$\delta(q_f, 0) = q_f$

$\delta(q_f, 1) = q_f$

Code:

```
// Implement DFA that accepts strings with sub string 001 over {0,1}
```

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



```

enum states { q0,q1,q2,qf};

enum states delta(enum states, char);

int main()
{
    enum states curr_state = q0;
    char string[20], ch;
    int i=0;

    printf("\n Enter a string \t");
    gets(string);

    ch = string[i];
    while(ch!='\0')
    {
        curr_state = delta(curr_state,ch);
        ch = string[++i];
    }
    if(curr_state==qf)
        printf("\n The string %s is valid.",string);
    else
        printf("\n The string %s is not valid.",string);

    return 0;
}

enum states delta(enum states s, char ch)
{
    enum states curr_state;

    switch(s)
    {
        case q0:
            if(ch=='0')
                curr_state = q1;
            else
                curr_state = q0;
            break;
        case q1:
            if(ch=='0')
                curr_state = q2;
            else
                curr_state = q0;
            break;
    }
}

```

```

        case q2:
            if(ch=='0')
                curr_state = q2;
            else
                curr_state = qf;
            break;
        case qf:
            if(ch=='0' || ch=='1')
                curr_state = qf;
            }
        return curr_state;
    }
}

```

OUTPUT

```

C:\Users\Resham\Desktop\TOC\toc lab\Labdfasub001.exe

Enter a string      1100100

The string 1100100 is valid.
-----
Process exited after 9.026 seconds with return value 0
Press any key to continue . . .

```

```

C:\Users\Resham\Desktop\TOC\toc lab\Labdfasub001.exe

Enter a string      11010

The string 11010 is not valid.
-----
Process exited after 6.237 seconds with return value 0
Press any key to continue . . .

```

LAB 3: Write a program to validate C identifiers and keywords.

C identifiers: These are the names of variables, functions, arrays, structures and pointers etc. The first character of C identifiers must be letter or underscore and remaining characters might be letters, digits or underscore.

Keywords: These are the reserved words having predefined meaning in the language. There are 32 keywords in C. They cannot be used as identifiers.

code:

```
// to identify valid identifiers and keywords in C
#include<stdio.h>
#include<string.h>
char keyword[32][10]= { "auto","double","int","struct","break","else","long", "switch","case",
                        "enum", "register", "typedef","char", "extern", "return", "union", "const",
                        "float", "short", "unsigned","continue","for","signed","void","default",
                        "goto", "sizeof", "volatile", "do","if","static","while"} ;

enum states { q0, qf, qd};
enum states delta(enum states, char);
int iskeyword(char []);

int main()
{
    enum states curr_state = q0;
    char string[20], ch;
    int i=0;

    printf("\n Enter a string \t");
    gets(string);

    ch = string[i];
    if(iskeyword(string))
        printf("\n The string %s is keyword.",string);
    else
    {
        while(ch!='\0')
        {
            curr_state = delta(curr_state,ch);
            ch = string[++i];
        }
        if(curr_state==qf)
            printf("\n The string %s is valid identifier.",string);
        else
            printf("\n The string %s is neither keyword nor valid identifier.",string);
    }
    return 0;
} //end of the main
```

```

//transition function
enum states delta(enum states s, char ch)
{
    enum states curr_state;
    switch(s)
    {
        case q0:
            if(ch>='A' && ch<='Z' || ch>='a' && ch<='z' || ch=='_')
                curr_state = qf;
            else
                curr_state = qd;
            break;
        case qf:
            if(ch>='A' && ch<='Z' || ch>='a' && ch<='z' || ch=='_' || ch>='0' && ch<='9')
                curr_state = qf;
            else
                curr_state = qd;
            break;
        case qd:
            curr_state = qd;
    }

    return curr_state;
}

int iskeyword(char str[])
{
    for(int i=0;i<32;i++)
    {
        if(strcmp(str,keyword[i])==0)
            return 1;
    }
    return 0;
}

```

OUTPUT

```
C:\Users\Resham\Desktop\TOC\toc lab\keywordsandidentifiers.exe

Enter a string      num

The string num is valid indentifier.
-----
Process exited after 10.43 seconds with return value 0
Press any key to continue . . .
```

```
C:\Users\Resham\Desktop\TOC\toc lab\keywordsandidentifiers.exe

Enter a string      1abc

The string 1abc is neither keyword nor valid identifier.
-----
Process exited after 8.069 seconds with return value 0
Press any key to continue . . . ■
```

```
C:\Users\Resham\Desktop\TOC\toc lab\keywordsandidentifiers.exe

Enter a string      int

The string int is keyword.
-----
Process exited after 2.508 seconds with return value 0
Press any key to continue . . . ■
```

LAB 4: Write a program to implement PDA that accepts all strings over alphabet $\Sigma = \{0, 1\}$ that have equal number of 0s and 1s

i. by final state.

Description of PDA:

It starts at q_0 and pushes \$ into the empty stack and switches to new state q_1 . At q_1 , PDA pushes input symbol if stack top symbol is \$ or stack top is same as input symbol, otherwise if input is 0 and stack top is 1 or input is 1 and stack top is 0, stack top is popped off. If input is finished and stack top is \$ at state q_1 , PDA switches to final state, q_f . The figure for the PDA is given below.

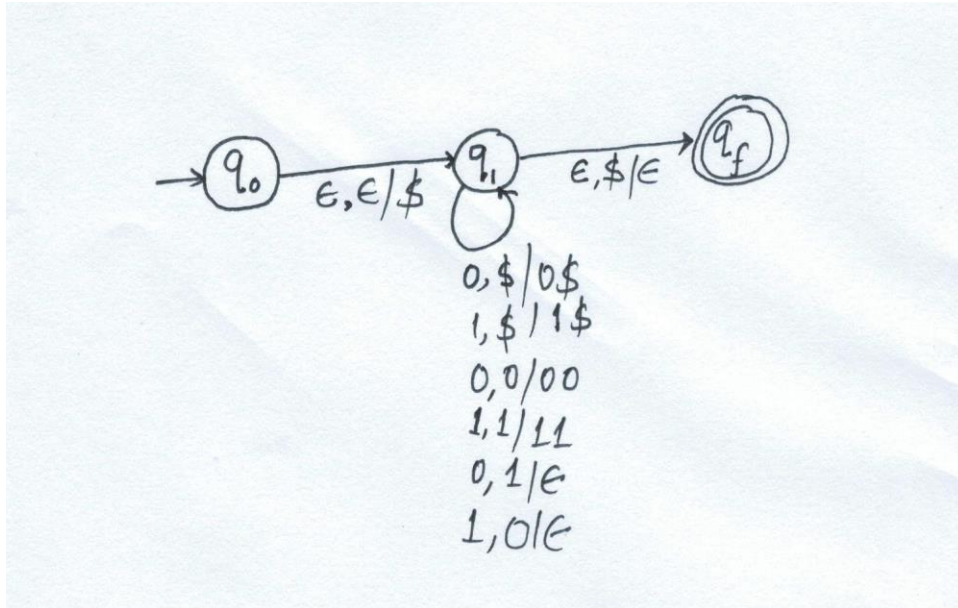


Fig. PDA accepting equal number of 0s and 1s by final state

Code for PDA accepting by Final State

/* TOC Lab : Implement a PDA for $L = \{ \text{set of all strings over } \{0,1\} \text{ such that equal number of 0s and 1s, acceptance by final state} \}$ */

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

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

```
#define MAX 100
```

```
enum states { q0, q1, qf};
```

```
void push(char ch);
```

```
void pop();
```

```
char get_stack_top();
```

```
enum states delta(enum states, char, char);
```

```
struct stack
```

```
{
```

```
    char symbols[MAX];
```

```
    int top;
```

```
};
```

```
struct stack s;
```

```
int main()
```

```
{
```

```
    char input[20];
```

```
    enum states curr_state = q0;
```

```
    s.top = -1;
```

```
    int i = 0;
```

```

char ch = 'e'; // e indicating epsilon
char st_top = 'e';
curr_state = delta(curr_state,ch,st_top);

printf("\n Enter a binary string: ");
gets(input);

ch = input[i];
st_top = get_stack_top();
int c=0;

while( c <=strlen(input))
{
    curr_state = delta(curr_state,ch,st_top);
    ch = input[++i];
    st_top=get_stack_top();
    c++;
}

if(curr_state == qf)
    printf("\n The string %s is accepted.",input);
else
    printf("\n The string %s is not accepted.",input);

return 0;
}

enum states delta(enum states s, char ch, char st_top)
{
    enum states curr_state;
    switch(s)
    {
        case q0:
            if(ch=='e' && st_top=='e')
            {
                curr_state = q1;
                push('$'); // $ is stack bottom marker
            }
            break;
        case q1:
            if(ch=='0' && (st_top=='$' || st_top=='0'))
            {
                curr_state = q1;
                push(ch);
            }
            else if(ch=='1' && (st_top=='$' || st_top=='1'))
            {

```

```

        curr_state = q1;
        push(ch);
    }
    else if(ch=='1' && st_top=='0'||ch=='0'&&st_top=='1')
    {
        curr_state = q1;
        pop();
    }
    else if(ch=='\0' && st_top=='$')
    {
        curr_state = qf;
        pop();
    }
    break;
}
return curr_state;
}

//function to get stack top symbol
char get_stack_top()
{
    return (s.symbols[s.top]);
}

//push function
void push(char ch)
{
    if(s.top<MAX-1 )
    {
        s.symbols[++s.top] = ch;
    }
    else
    {
        printf("\n Stack Full.");
    }
}

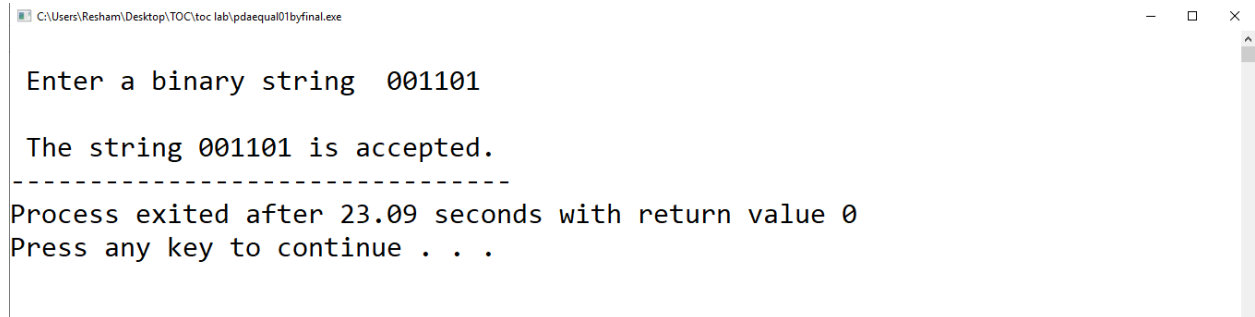
//pop function
void pop()
{
    if(s.top>=0)
    {
        s.symbols[s.top]=' ';
        s.top--;
    }
    else

```

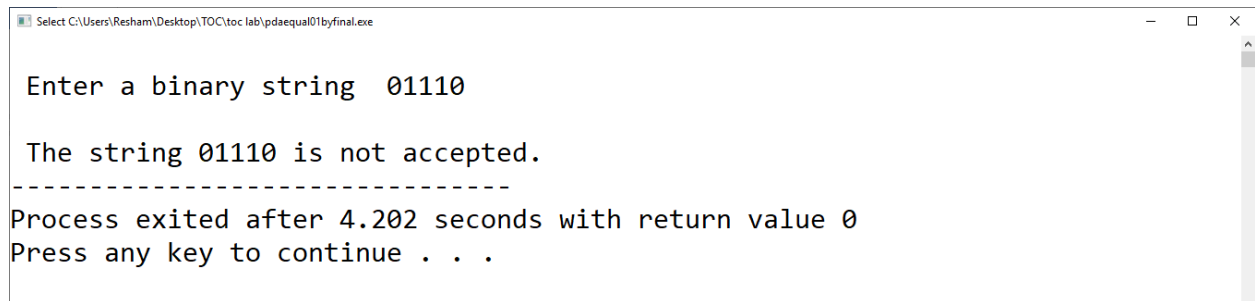


```
        printf("\n Stack Empty.");  
    }
```

OUTPUT:



```
C:\Users\Resham\Desktop\TOC\toc lab\pdarequal01byfinal.exe  
Enter a binary string  001101  
The string 001101 is accepted.  
-----  
Process exited after 23.09 seconds with return value 0  
Press any key to continue . . .
```

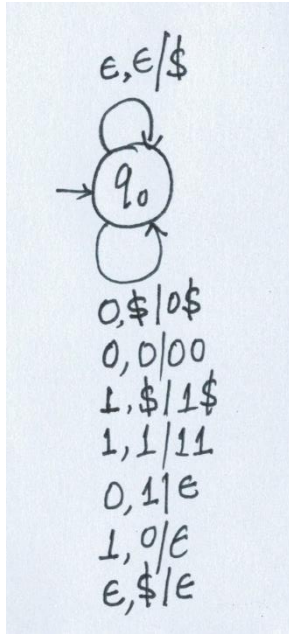


```
Select C:\Users\Resham\Desktop\TOC\toc lab\pdarequal01byfinal.exe  
Enter a binary string  01110  
The string 01110 is not accepted.  
-----  
Process exited after 4.202 seconds with return value 0  
Press any key to continue . . .
```

ii. PDA accepting equal number of 0s and 1s with empty stack

PDA description:

It starts at q_0 and pushes $\$$ into the empty stack. At the same state q_0 , PDA pushes input symbol if stack top symbol is $\$$ or stack top is same as input symbol, otherwise if input is 0 and stack top is 1 or input is 1 and stack top is 0, stack top is popped off. If input is finished and stack top is $\$$, PDA accepts the input string. The figure for the PDA is given below.



Code for PDA Accepting by Empty Stack

/* TOC Lab : Implement a PDA for $L = \{ \text{set of all strings over } \{0,1\} \text{ such that equal number of 0s and 1s, acceptance by empty stack} \}$ */

```
#include<stdio.h>
#include<string.h>
#define MAX 100

enum states { q0};
void push(char ch);
void pop();
char get_stack_top();
enum states delta(enum states, char, char);

struct stack
{
    char symbols[MAX];
    int top;
};

struct stack s;

int main()
{
    char input[20];
```

```

enum states curr_state = q0;
s.top = -1;
int i = 0;
char ch = 'e';
char st_top = 'e';
curr_state = delta(curr_state, ch, st_top);

printf("\n Enter a binary string: ");
gets(input);

ch = input[i];
st_top = get_stack_top();
int c = 0;

while( c <= strlen(input))
{
    curr_state = delta(curr_state, ch, st_top);
    ch = input[++i];
    st_top = get_stack_top();
    c++;
}

if(s.symbols[s.top] == '$')
    printf("\n The string %s is accepted.", input);
else
    printf("\n The string %s is not accepted.", input);

return 0;
}

enum states delta(enum states s, char ch, char st_top)
{
    enum states curr_state;
    switch(s)
    {
        case q0:
            if(ch == 'e' && st_top == 'e')
            {
                curr_state = q0;
                push('$');
            }
            else if(ch == '0' && (st_top == '$' || st_top == '0'))
            {
                curr_state = q0;
                push(ch);
            }
            else if(ch == '1' && (st_top == '$' || st_top == '1'))

```

```

        {
            curr_state = q0;
            push(ch);
        }
    else if(ch=='1' && st_top=='0' || ch=='0' && st_top=='1')
    {
        curr_state = q0;
        pop();
    }
    else if(ch=='\0' && st_top=='$')
    {
        curr_state = q0;
        //pop();
    }
    break;
}
return curr_state;
}

```

```

char get_stack_top()
{
    return (s.symbols[s.top]);
}

```

```

void push(char ch)
{
    if(s.top<MAX-1 )
    {
        s.symbols[++s.top] = ch;
    }
    else
    {
        printf("\n Stack Full.");
    }
}

```

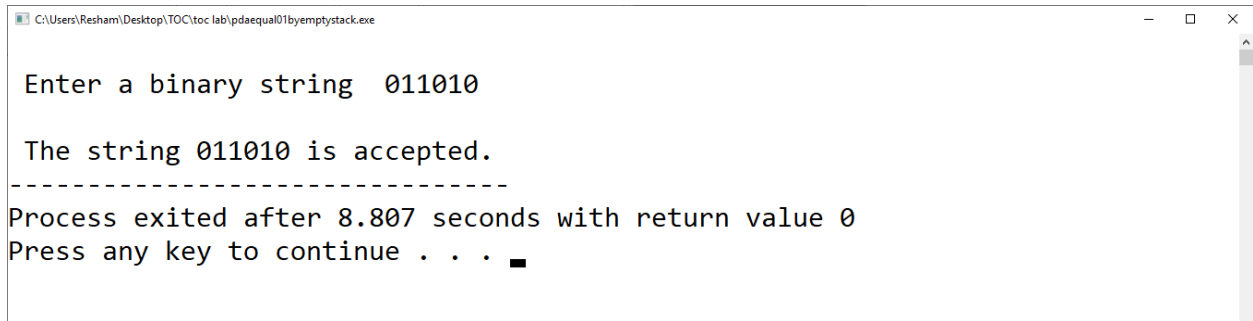
```

void pop()
{
    if(s.top>=0)
    {
        s.symbols[s.top]=' ';
        s.top--;
    }
    else
        printf("\n Stack Empty.");
}

```

```
}
```

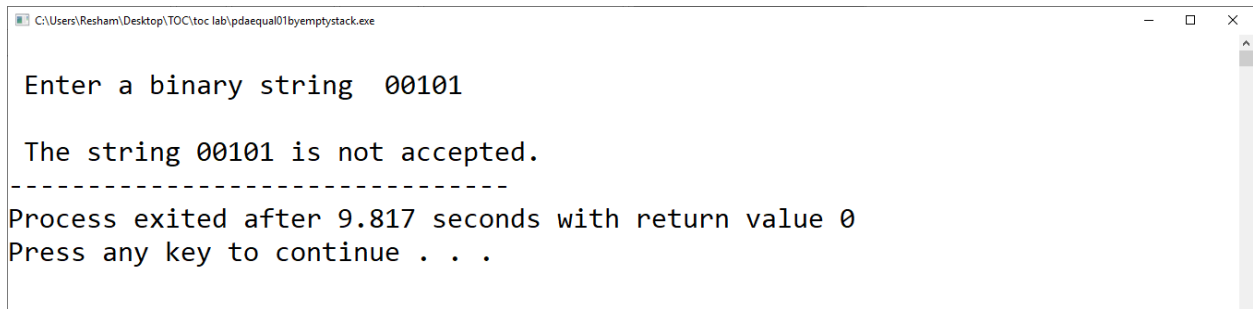
OUTPUT



```
C:\Users\Resham\Desktop\TOC\toc lab\pdaequal01byemptystack.exe

Enter a binary string 011010

The string 011010 is accepted.
-----
Process exited after 8.807 seconds with return value 0
Press any key to continue . . . █
```



```
C:\Users\Resham\Desktop\TOC\toc lab\pdaequal01byemptystack.exe

Enter a binary string 00101

The string 00101 is not accepted.
-----
Process exited after 9.817 seconds with return value 0
Press any key to continue . . . █
```

LAB 5:

Write a program to implement PDA that accepts all strings over alphabet $\Sigma = \{0,1\}$ that have contain number of 0's followed by equal number of 1s by final state.

PDA Description:

The PDA starts at q0 state. At this state, it simply pushes \$ into the empty stack and switches to next state q1. At state q1, PDA pushes input 0 seen on the top of \$ or 0 on the stack and if it sees 1 when stack top is 0, it pops the stack top switching to next state q2. At q2, it pops 0's for each 1 seen on the input. If no input symbol is present and \$ on the top of stack, PDA switches to final state qf.

Code:

```
/* TOC Lab : Implement a PDA for L = { set of all strings over {0,1} such that 0^n1^n,
acceptance by final state */
```

```
#include<stdio.h>
#include<string.h>
#define MAX 100

enum states { q0, q1,q2,qf,qr};
void push(char ch);
void pop();
char get_stack_top();
enum states delta(enum states, char, char);

struct stack
{
    char symbols[MAX];
    int top;
};

struct stack s;

int main()
{
    char input[20];
    enum states curr_state = q0;
    s.top = -1;
    int i =0;
    char ch = 'e';
    char st_top = 'e';
    curr_state = delta(curr_state,ch,st_top);

    printf("\n Enter a binary string: ");
```

```

    gets(input);

    ch = input[i];
    st_top = get_stack_top();
    int c=0;

    while( c <=strlen(input))
    {
        curr_state = delta(curr_state,ch,st_top);
        ch = input[++i];
        st_top=get_stack_top();
        c++;
    }
    if(curr_state==qf)
        printf("\n The string %s is accepted.",input);
    else
        printf("\n The string %s is not accepted.",input);

    return 0;
}

enum states delta(enum states s, char ch, char st_top)
{
    enum states curr_state = qr;
    switch(s)
    {
        case q0:
            if(ch=='e' && st_top=='e')
            {
                curr_state = q1;
                push('$');
            }
            break;
        case q1:
            if(ch=='0' && (st_top=='$' ||st_top=='0'))
            {
                curr_state = q1;
                push(ch);
            }
            else if(ch=='1' && st_top=='0')
            {
                curr_state = q2;
                pop();
            }
            else
                curr_state = qr; // qr for undefined transition
    }
}

```

```

        break;
    case q2:
        if(ch=='1' && st_top=='0')
        {
            curr_state = q2;
            pop();
        }
        else if(ch=='\0' and st_top=='$')
        {
            curr_state = qf;
            pop();
        }
        else
            curr_state = qr;

        break;
    }
    return curr_state;
}

```

```

char get_stack_top()
{
    return (s.symbols[s.top]);
}

```

```

void push(char ch)
{
    if(s.top<MAX-1 )
    {
        s.symbols[++s.top] = ch;
    }
    else
    {
        printf("\n Stack Full.");
    }
}

```

```

void pop()
{
    if(s.top>=0)
    {
        s.symbols[s.top]=' ';
        s.top--;
    }
}

```



```

    else
        printf("\n Stack Empty.");
}

```

Output:

```

Select C:\Users\c\Downloads\pda0n1nbyfinal.exe

Enter a binary string  000111
The string 000111 is accepted.
-----
Process exited after 4.313 seconds with return value 0
Press any key to continue . . .

```

```

C:\Users\c\Downloads\pda0n1nbyfinal.exe

Enter a binary string  00011
The string 00011 is not accepted.
-----
Process exited after 8.545 seconds with return value 0
Press any key to continue . . .

```

Lab 6: Implement the TM accepting the language $\{ 0^n 1^n / n \geq 1 \}$ over alphabet, $\Sigma = \{ 0, 1 \}$.

Turing Machine Description:

Given finite sequence of 0's and 1's on tape and followed by blanks. The TM starts at state q0 and changes 0 to an X and moves to the right changing its state to q1.

At state q1, TM expects 1 and changes a 1 to Y and moves to the left changing the state to q2. If any number of 0s and Ys are seen, it remains on the state q1 and leaving these symbols unchanged and moving the head position to the right.

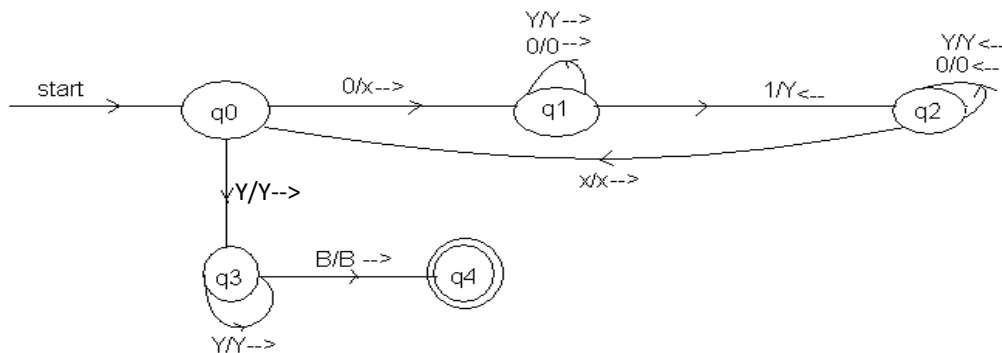
At state q2, if 0s or Y's are seen, it leaves them as it is and moves to the left staying at the same state q2. If it sees X at state q2, the tape symbol is left unchanged moves to right switching its state to q0.

At state q0, if it sees Y then the symbol is left unchanged and head is moved right changing to the state q3.

At state q3, if Y is seen, it is left unchanged and head is moved to the right. If BLANK (here, '\0') is seen at state q3, the string is accepted by switching the state to q4.

At any state, if the machine seen other than the defined symbols, it rejects the string.

The state transition diagram is shown in the figure below.



Code:

```
/* TOC Lab: Implement a TM for L = { set of all strings over {0,1} such that the string have  
number of 0s followed by same number of 1s. */
```

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

```
enum states { q0, q1, q2,q3,q4,qr};
```

```
int main()
```

```
{
```

```
    char input[100];
```

```
    enum states curr_state = q0;
```

```
    int i;
```

```
    for(i=0;i<100;i++)
```

```
        input[i] = '\0';
```

```

printf("\n Enter a binary string: ");
gets(input);
i=0;
while(1)
{
    switch(curr_state)
    {
        case q0:
            if(input[i]=='0')
            {
                curr_state = q1;
                input[i]='x';
                i++;
            }
            else if(input[i]=='y')
            {
                curr_state=q3;
                i++;
            }
            else
                curr_state = qr; //for invalid transition
            break;
        case q1:
            if(input[i]=='0')
            {
                curr_state = q1;
                i++;
            }
            else if(input[i]=='y')
            {
                curr_state = q1;
                i++;
            }
            else if(input[i]=='1')
            {
                curr_state = q2;
                input[i]='y';
                i--;
            }
            else
                curr_state = qr;
            break;
        case q2:

```

```

        if(input[i]=='0')
        {
            curr_state = q2;
            i--;
        }
        else if(input[i]=='y')
        {
            curr_state = q2;
            i--;
        }
        else if(input[i]=='x')
        {
            curr_state = q0;
            i++;
        }
        else
            curr_state = qr;
        break;
    case q3:
        if(input[i]=='y')
        {
            curr_state = q3;
            i++;
        }
        else if(input[i]=='\0')
        {
            curr_state=q4;
        }
        else
            curr_state = qr;
        break;
    }//end of switch
    if(curr_state == qr || curr_state==q4)
        break;
//end of while loop

    if(curr_state == q4)
        printf("\n The string is accepted.");
    else
        printf("\n The string is not accepted.");

    return 0;
}

```

OUTPUT

```
C:\Users\Resham\Desktop\tm0n1n.exe

Enter a binary string  000111

The string is accepted.
-----
Process exited after 7.121 seconds with return value 0
Press any key to continue . . .
```

```
C:\Users\Resham\Desktop\tm0n1n.exe

Enter a binary string  00110

The string is not accepted.
-----
Process exited after 4.432 seconds with return value 0
Press any key to continue . . . ■
```

Q) Implement the TM: $L = \{a^n b^n c^n / n \geq 0\}$ by yourself...

(By Arjun using Chat GPT)

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

```
#define MAX 1000
```

```
// Function to check if the input is of the form  $a^n b^n c^n$ 
```

```
int checkTM(char tape[]) {
    int i = 0, n = strlen(tape);
```

```
// While there are unmarked 'a's
```

```
while (1) {
```

```
    // Move to the leftmost 'a'
```

```
    while (tape[i] != 'a' && i < n) i++;
```

```
    if (i >= n) break; // No more 'a's, move to check end
```

```

    // Mark 'a' as X
    tape[i] = 'X';

    // Move right to find the first 'b'
    while (tape[i] != 'b' && i < n) i++;
    if (i >= n) return 0; // Mismatch: no 'b' found, reject

    // Mark 'b' as Y
    tape[i] = 'Y';

    // Move right to find the first 'c'
    while (tape[i] != 'c' && i < n) i++;
    if (i >= n) return 0; // Mismatch: no 'c' found, reject

    // Mark 'c' as Z
    tape[i] = 'Z';

    // Reset to the beginning of the tape
    i = 0;
}

// After marking, check if there are any remaining 'a', 'b', or 'c'
for (i = 0; i < n; i++) {
    if (tape[i] == 'a' || tape[i] == 'b' || tape[i] == 'c')
        return 0; // Reject if any unmarked 'a', 'b', or 'c' remains
}
return 1; // Accept if all 'a', 'b', and 'c' are marked
}

int main() {
    char tape[MAX];

    printf("Enter a string over the alphabet {a, b, c}: ");
    scanf("%s", tape);

    if (checkTM(tape))
        printf("The string is accepted by the Turing Machine.\n");
    else
        printf("The string is rejected by the Turing Machine.\n");

    return 0;
}

```