Wednesday

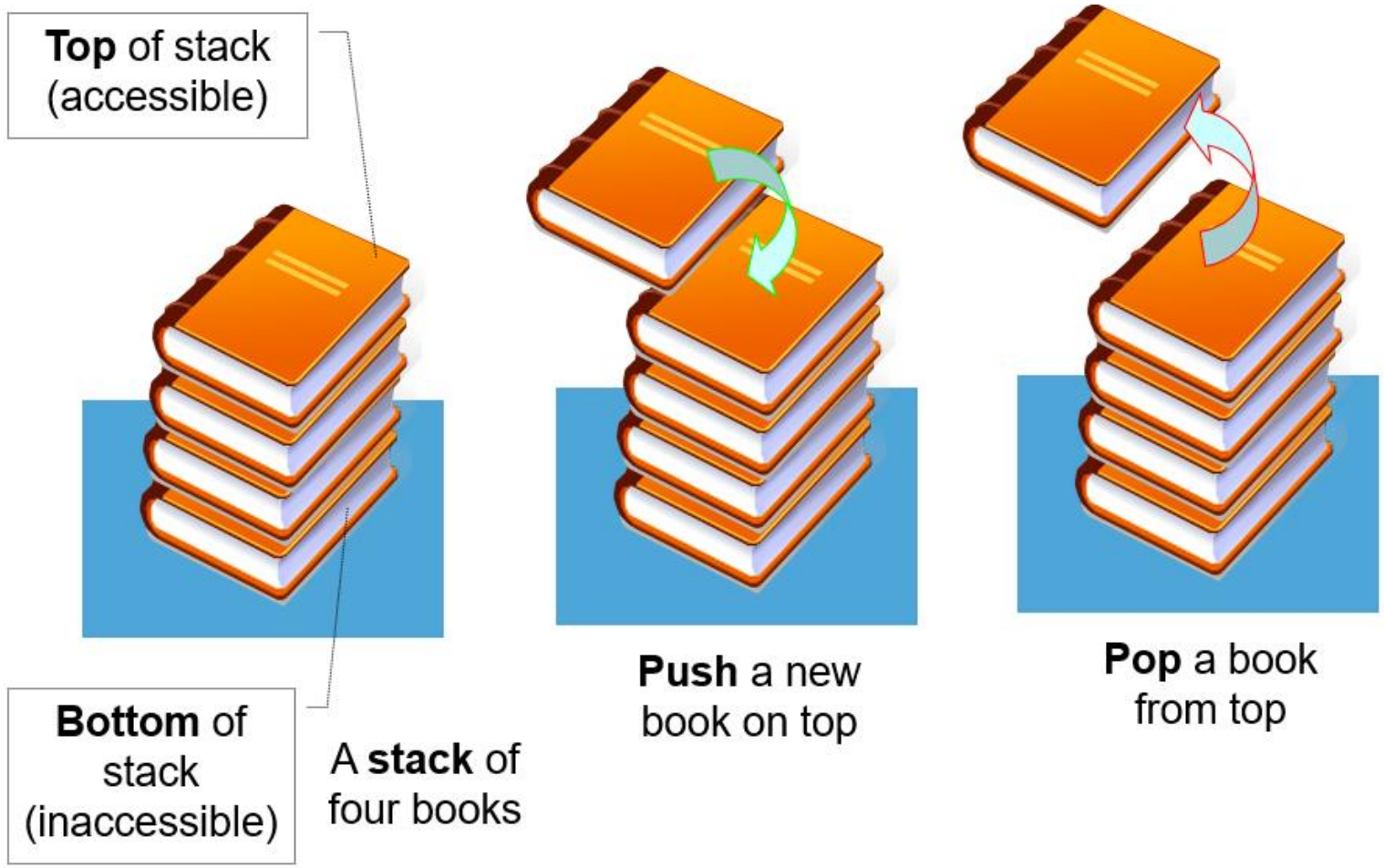
27-Oct-2021

10.30am – 12.30pm

Infix to postfix Algorithm implementation

infix to prefix conversion Algorithm and implementation

Postfix and prefix evaluation Algorithm and implementation



```
#include<iostream.h>
#include<conio.h>
```

class student

tokens

( )

(a+b)*c

0 1 2 3 4 5

```
enum precedence{lparen, rparen, plus, minus, times, divide,
                mod, eos, operand};
```

6 7 8

User defined datatype

end of stack

```
int icp[] = {20, 19, 12, 12, 13, 13, 13, 0};
int isp[] = {0, 19, 12, 12, 13, 13, 13, 0};
```

icp[0] = 20

icp[lparen] = 20

icp: incoming predence, isp: instack precedence

```
char Stack::topele(){ return(a[top]); }

precedance get_token(char c)
{
    switch(c)
    {
        case '(': return lparen;
        case ')': return rparen;
        case '+': return plus;
        case '-': return minus;
        case '*': return times;
        case '/': return divide;
        case '%': return mod;
        default: return operand;
    }
}
```

operators

a+b*c

Character → named constant

Conversion

```
void infix_postfix(char infix[])
{
  precedence temp;
   int i=0;
  stack S;
  while(infix[i] != '\0')
  {     temp= get_token(i fix[i]);
     ① if(temp== operand) cout<<infix[i];
     ② else if(temp == rparen)
       {
          while(get_token(S.topele()) != lparen)
                cout<< S.pop();
          char C= S.pop(); //pop lparen & ignore
       }
}
```

Input	o/p of get_token
a	operand
+	plus
-	minus

abc*+

a+b*c

token	stack	o/p
a		a
+	+	
b	+	ab
*	* +	
c	* +	abc
)		abc*+

③ else { // if infix[i] is operator

{

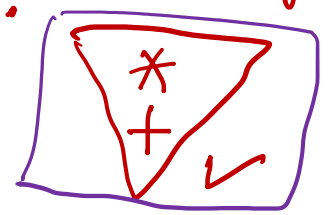
if(s.isempty()) s.push(infix[i]);

else // stack is not empty, other operators are present inside // stack

{

while(icp[temp] <= isp[get_token(s.topele())] && !s.isempty())

cout << s.pop();



s.push(infix[i]);

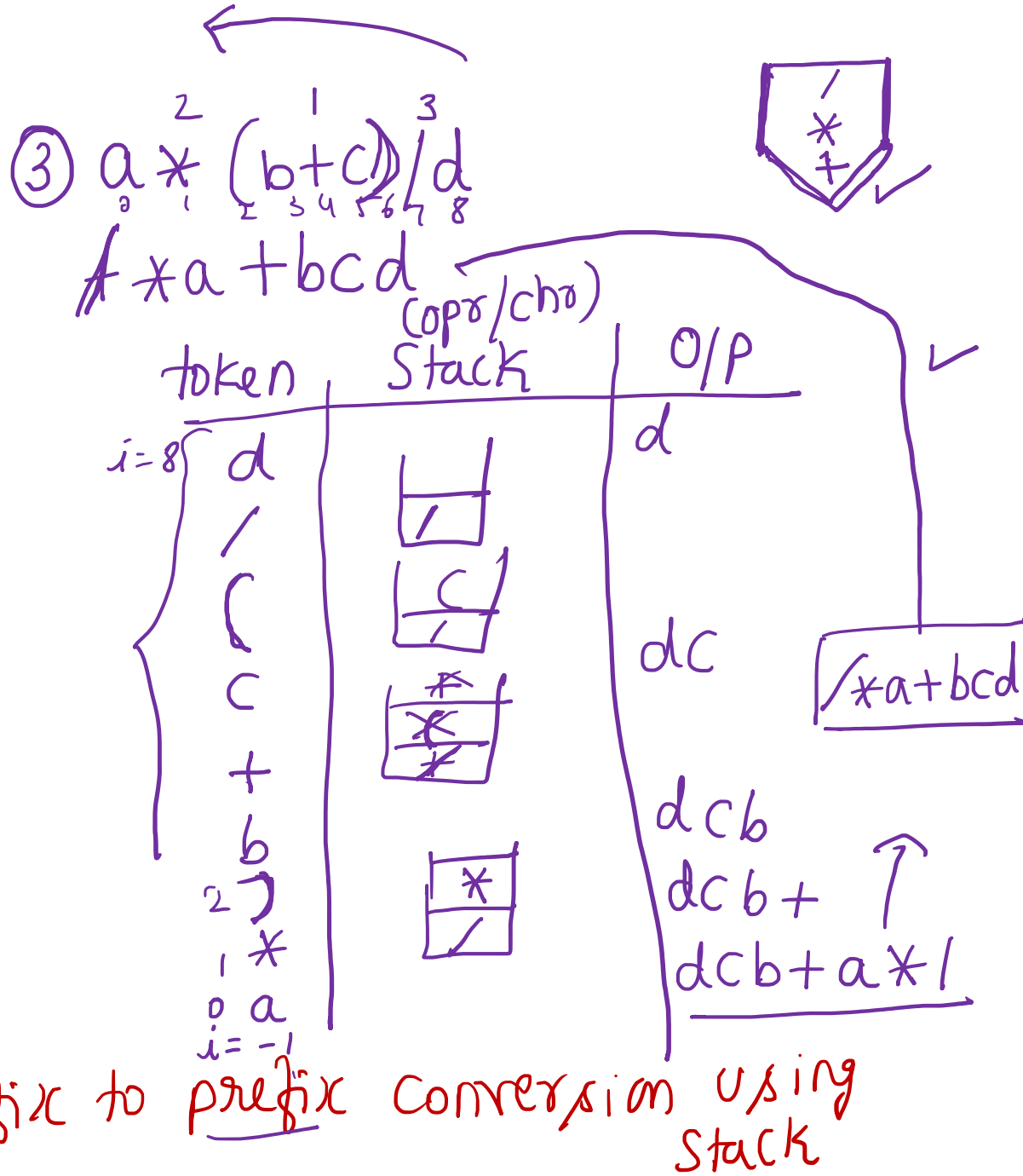
}

}

i++;

}// while(infix != '\0') s.display(); }

Sl. No.	Infix	Prefix
1	a+b*c-d	
2	a*b+c/d	
3	a*(b+c)/d	
4	a*(b+c/d)	
5	(a*b)+(c/d)	
6	(a*(b+c))/d	
7	a+b*c/d-e	
8	(a+b)*c/d-e	
9	(a+b)*(c-d)/(e+f)	
10	(a+b)*(c-d)/((e-f)*(g+h))	
11	a+b*c-d/e*f	
12	a/b^c+d*e-f*g	



do it for all 12 expressions ← Infix to prefix conversion using Stack

infix to prefix conversion alg

$A * (B+C) * D$
2 1 3

$**A+BCD$

token	stack	o/p
D	*	D
*	(*	
(	+ (*	DC
C	(*	
+		DCB
B	*	DCB+
)	* *	
*		DCB+A
A		DCB+A**
i=-1		

reverse infix expression

'(' ↔ ')', ')' ↔ '('

↓

infix to postfix alg ✓

↓ o/p

reverse this o/p & print

$**A+BCD$

Infix to Prefix conversion code 1/5

```
enum Boolean{FALSE,TRUE};
int icp[]={20,19,12,12,13,13,13,0};
int isp[]={0,19,12,12,13,13,13,0};
enum precedence{lparen,rparen,plus,minus,times,divide,mod,eos,operand};
class stack
{           int top, maxsize;          char a[10];

public: stack()
            { maxsize=10;top=-1;}
            Boolean Isfull()
            {           if(top==maxsize-1)  return TRUE;
                        return FALSE;
            }
            Boolean Isempty()
            {           if(top==-1)             return TRUE;
                        return FALSE;
            }
            void push(char x);
            char  pop();
            void display();
            char topele();
};
```

operators are placed inside stack

Infix to Prefix conversion code 2/5

```
void stack::push(char x)
{      if(Isfull())
                        cout<<"Stack is full \n";
              else
                        a[++top]=x;
}
char stack::topele()
{             return(a[top]);
}
char  stack::pop()
{
              if(Isempty())
                        return(-9999);
              else
                          return(a[top--]);

}
void  stack::display()
{       if(Isempty())
                        cout<<"Stack is empty";
              else
              for(int i=top;i>-1;i--)
                        cout<<a[i];
}
```

Infix to Prefix conversion code 3/5

```
precedence get_token(char c)
{
        switch(c)
        {
        case '(':return lparen;
        case ')':return rparen;
        case '+':return plus;
        case '-':return minus;
        case '*':return times;
        case '/':return divide;
        case '%':return mod;
        default:return operand;
        }
}
```

Infix to Prefix conversion code 4/5

```
void prefix(char infix[])
{       precedence temp;   char prefix[20];
        int i=0,p=0;       stack s;
        while(i>=0)//while(i>-1)
        {       temp=get_token(infix[i]);
                if(temp==operand)                       prefix[p++]=infix[i];
                else if(temp==lparen)
                {       while(get_token(s.topele())!=rparen)
                                prefix[p++]=s.pop();
                        char c=s.pop();
                }
                else
                {       if(s.Isempty()==TRUE)   s.push(infix[i]);
                         else
                         {      while(icp[temp]<isp[get_token(s.topele())]&&s.Isempty()==FALSE)
                                                prefix[p++]=s.pop();
                          s.push(infix[i]);
                          }
                }
                i--;
        }
        while(!s.Isempty()) prefix[p++]=s.pop();
        prefix[p]='\0';
        cout<<strrev(prefix);
}
```

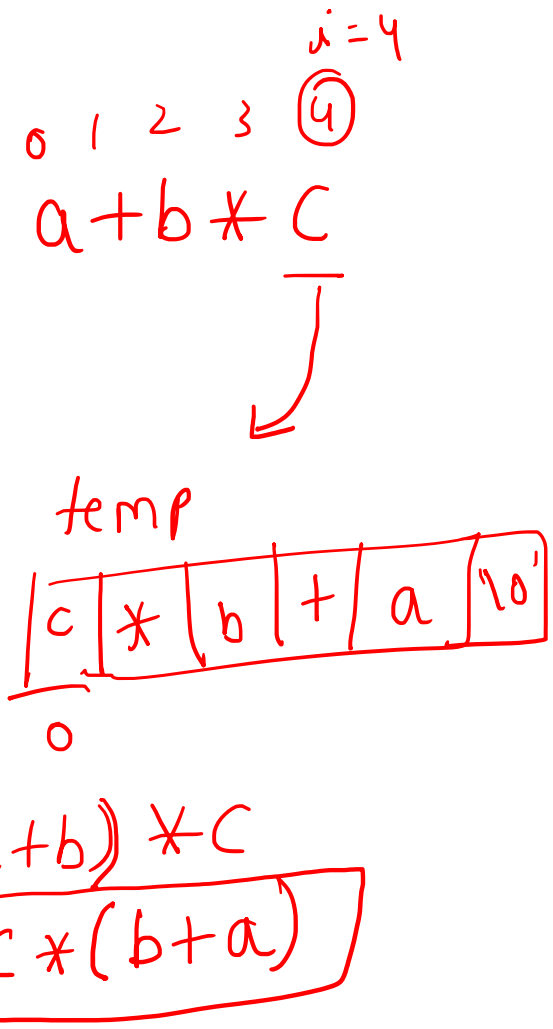
```
int i=0;
while(infix[i]!='\0')
        i++;
i=i-1;
```

```
char[] strrev(char pre[])
{       char a[50]; int i=0;
        while(pre[i]!='\0')
```

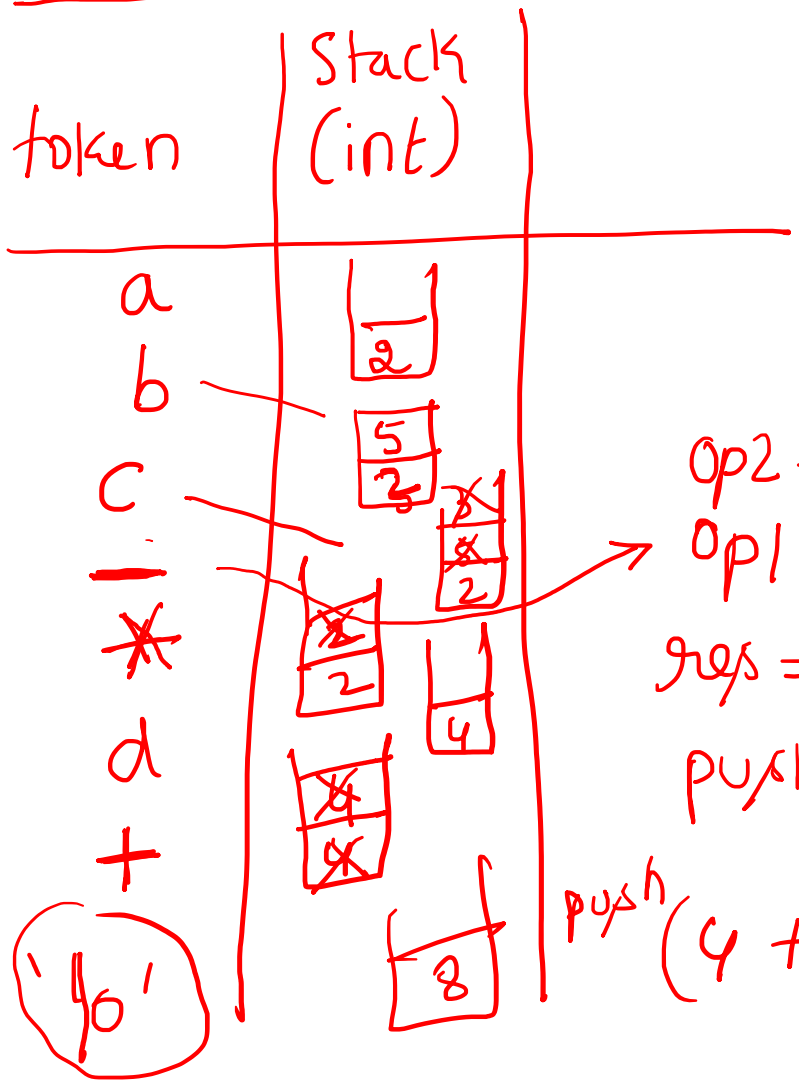
Infix to Prefix conversion code 5/5

```
void main()
{
        char infix[20],temp[20];
        int j=0;
        cout<<"enter infix expression\n";
        cin>>infix;
        for(int i=strlen(infix)-1;i>=0;i--)
        {
                if(infix[i]=='(')       temp[j++]=')';
                else if(infix[i]==')')  temp[j++]='(';
                else                    temp[j++]=infix[i];
        }
        temp[j]='\0';

        prefix(temp);
        getch();
}
```



postfix evaluation



2 * 2

4

op2 = pop(); op2 3

op1 = pop(); 5

res = op1 opr op2

push(res)

push (4 + 4) → 8

res = 8

a * (b - c) + d

2 5 3 4

2 * 2

4 + 4

= 8

a b c - * d +

(5 - 3) → res = 2

push(2)

a *(b-c)+d

a = 2

b = 55

c = 40

d = 10

postfix

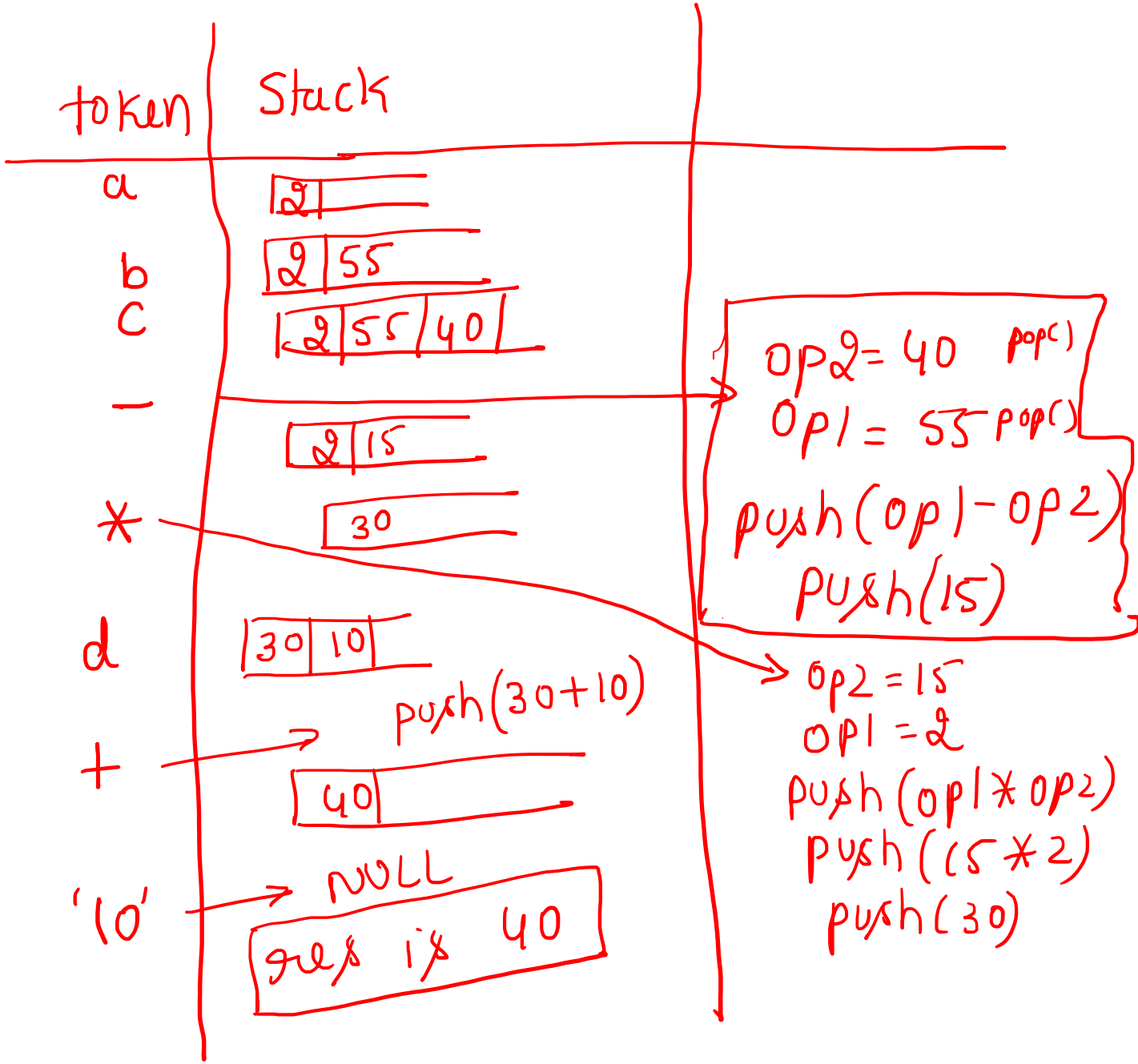
a bc-*d+

2*(55-40)+10

2*15+10

30+10

ans = 40



postfix evaluation algorithm

Step 1:

```
for(i=0; postfix[i] != '\0'; i++)
{
    if (postfix[i] is an operand)
        push(postfix[i])
    else // postfix[i] is an operator
    {
        op2 = pop();
        op1 = pop();
        find the value of op1 operator op2
        push the value into the stack
    }
}
```

Step 2:
Display the stack content as result

Solve $a*(b+c/d)$

$a = 4$

$b = 5$

$c = 15$

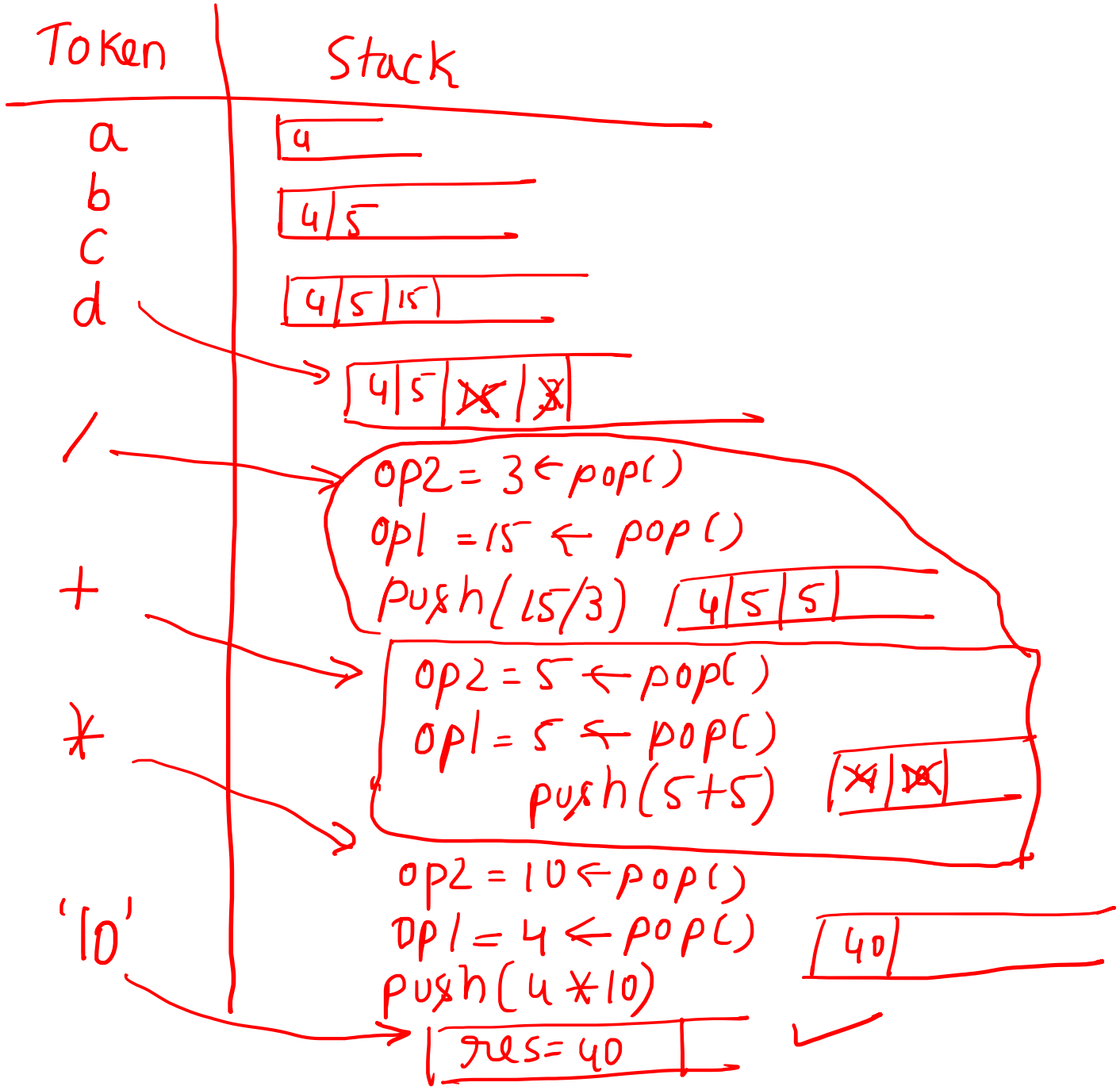
$d = 3$

$5+5$

$4*10$

res = 40

abcd/+*



```
void postfix_evaluation()
{
    Stack s;
    int digit, op1, op2, res; char token, postfix[50];
    cout<<"Enter a postfix expression"; gets(postfix);
    for(int i=0; postfix[i]!='\0'; i++)
    {
        token = postfix[i];
        if( token >= 48 && token <= 57)
        { digit = token - 48; s.push(digit); }
        else if( token == '+' || token == '-' || token == '*'
              || token == '/' || token == '%' )
        // token is an operator
```

ascii value	digit
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9

token is a digit

12*3+

ab*c+

```
{ //token is an operator
        op2 = s.pop();
        op1 = s.pop();
switch(token)
{  case '%' : res = op1 % op2;
              break;
   case '/' : res = op1 / op2;
              break;
   case '*' : res = op1 * op2;
              break;
   case '+' : res = op1 + op2;
              break;
   case '-' : res = op1 - op2;
              break;
  } s.push(res);
 }//else if
else //token is an operand
{
    int value;
cout << "Enter numerical value of: "
    << token;
 cin >> value;
  s.push(value);
  }// else
}//for
cout << "result: " << s.pop();
  }//postfix - evaluation
```

alphabet

prefix evaluation

2 55 -40 10

a * (b - c) + d

res = 40

+ * a - b c d

←

token	Stack (int)
d	10
c	10 | 40
b	10 | ~~40~~ | ~~55~~
—	10 | 15
a	10 | ~~15~~ | ~~2~~
*	10 | 30
+	40
'10'	Res = 40

op1 = 55 ← pop()
op2 = 40 ← pop()
push (55 - 40)

op1 = 2 ← pop()
op2 = 15 ← pop()
push (2 * 15)

op1 = 30 ← pop()
op2 = 10 ← pop()
push (30 + 10)

```
void prefix_evaluation()
{
    Stack s; int i=0;
    int digit, op1, op2, res;
    char token, prefix[50];
    cout<<"Enter a prefix expression: ";
    gets(prefix);
    while(prefix[i]!='\0')  i++;
    i=i-1;
    while(i>=0)
    {
        token=prefix[i];
```

```
if(token >= 48  && token <= 57)
{      digit = token - 48;  S.push(digit);  }
else if(token == '+' || token == '-' || token == '*' || token == '/'
          || token == '%')
{   op1 = S.pop();   op2 = S.pop();
    switch(token)
    {   case '%': S.push(op1 % op2); break;
        case '/': S.push(op1 / op2); break;
        case '*': S.push(op1 * op2); break;
        case '+': S.push(op1 + op2); break;
        case '-': S.push(op1 - op2); break;
    }
}//else if
```

```
else // token is an operand
{
    int val;
    cout<<"Enter value for: "<<token;
    cin>>val;
    s.push(val);
}
{ i=i-1;
}// for while
cout<<" Result is "<<s.pop();
}// prefix_evaluation
```