

Experiment No. : 3

Title: Write a program to demonstrate the use of dynamic stack in expression conversion.

(Autonomous College Affiliated to University of Mumbai

KJSCE/IT/SY BTech/SEMIII/DS/2022-23

Batch: Roll No.: Experiment No.:3

Aim:

- a) WAP to create a stack using SLL by implementing following operations 1. Create stack, 2. Insert an element, 3. Delete an element, 4. Display top element.
- b) WAP to convert a given infix expression into equivalent postfix form using stack implemented in part (a).

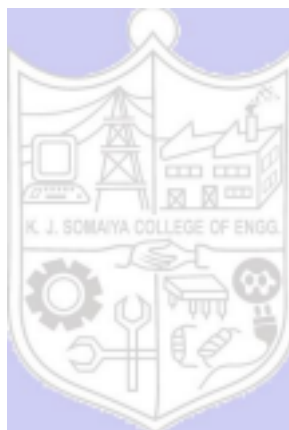
_ Resources Used: Turbo C/ C++/JAVA editor and compiler (online/offline)

_ Theory:

- a) **Stack :-** Stack can be implemented by using an array or by using a linked list. One important feature of stack, that the last element inserted into a stack is the first element deleted. Therefore stack is also called as Last in First out (LIFO) list.

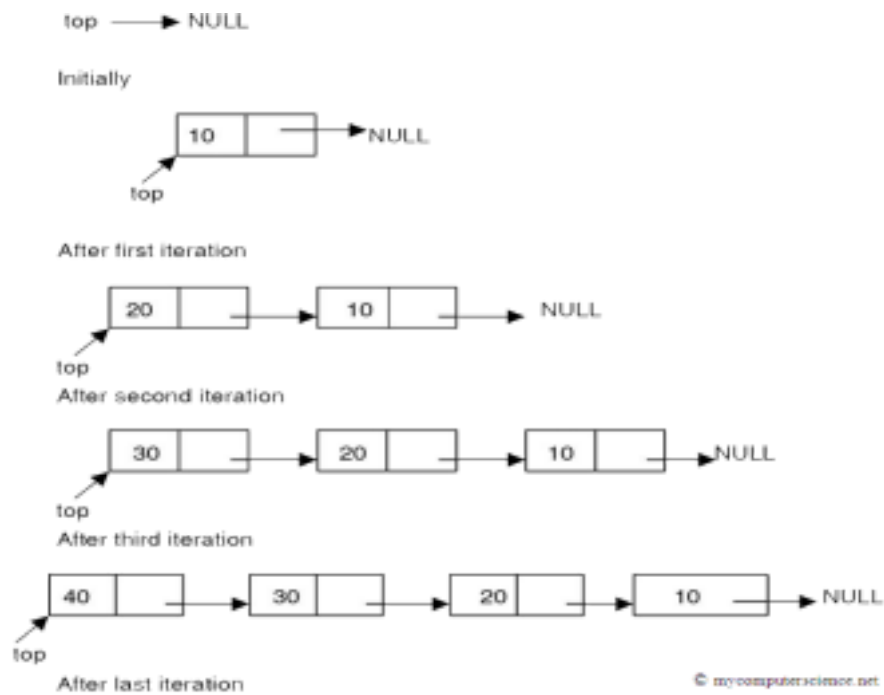
Linked List

Fig(c) shows
SLL



implementation –

the push operation working on stack using



Fig(c) Push operation using SLL

(Autonomous College Affiliated to University of Mumbai)

KJSCE/IT/SY BTech/SEMIII/DS/2022-23

b) Expression conversion

Calculators employing reverse Polish notation use a stack structure to hold values. Expressions can be represented in prefix, postfix or infix notations. Conversion from one form of the expression to another form may be accomplished using a stack. Many compilers use a stack for parsing the syntax of expressions, program blocks etc. before translating into low level code. Most of the programming languages are context-free languages allowing them to be parsed with stack based machines.

Examples

Infix expression $(2 * 5 - 1 * 2) / (11 - 9)$

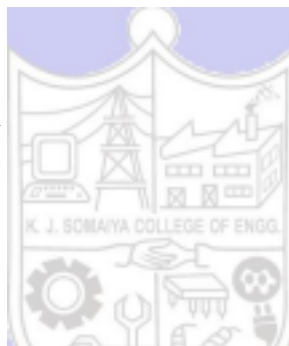
Postfix Expression $2\ 5\ *\ 1\ 2\ *\ -\ 11\ 9\ -\ /\$

Algorithm :

Infix String : $a+b*c-d$

1. Read an
2. If item is an
3. If item is

item from input infix expression
operand append it to postfix string
“(“ push it on the stack



4. If the item is an operator and top of the stack(tos) is also operator
 1. If the operator has higher precedence than the one on tos then push it onto the operator stack
 2. If the operator has lower or equal precedence than the one on tos then 1. pop the operator on tos and append it to postfix string(repeat if tos is again an operator)
 2. push lower precedence operator onto the stack
5. If item is “)” pop all operators from tos one-by-one and append it to postfix string, until a “(“ is encountered on stack. Remove “(“ from tos and discard it. 6. If end of infix string, pop the items from tos one-by-one and append to postfix string. If other than operator anything is encountered on the stack at this step, then declare input as invalid input.

Infix String : a+b*c-d

Postfix String : abc*+d

Activity :

a) Implement “dynamic stack” with following functions

1. **void createStack(void)** : Create and initializes the top to NULL, creating the empty stack.
2. **void push(char x)** : Creates a node with value 'x' and inserts on top of the stack.
3. **char pop(void)** : Deletes a node from top of the stack and returns the deleted value.
4. **boolean isEmpty(void)** : Returns “1” for stack empty; “0” otherwise.
5. **char peek(void)** : Return current stack top element.

(Autonomous College Affiliated to University of Mumbai

KJSCE/IT/SY BTech/SEMIII/DS/2022-23

NOTE : Map appropriate methods of SLL with the methods of STACK and use.

b) Implement expression conversion

1. Implement the above algorithm given for INFIX to POSTFIX expression conversion.

Results: A program depicting the correct behaviour of stack in conversion of expression and capable of handling all possible exceptional conditions and the same is reflecting clearly in the output.

Program and Output:

Program 1

```
#include<stdio.h>
// Code by Arya Nair
struct Node
```

```

{
    int data;
    struct Node* next;
};
struct Node* head;

int isEmpty(struct Node* temp){
    return !temp;
}

void createSLL()
{
    head=(struct Node*)malloc(sizeof(struct Node));
    head->next=NULL;
}

void push(struct Node* head,int val){
    while(head->next!=NULL){
        head=head->next;
    }
    struct Node* temp =(struct Node*)malloc(sizeof(struct Node));
    temp->data=val;
    temp->next=NULL;
    head->next=temp;
}

void display(struct Node* temp)
{
    while (!isEmpty(temp)){
        printf("%d ",temp->data);
        temp=temp->next;
    }
    printf("\n");
}

void pop(){
    if(isEmpty(head)){
        printf("\nstack is empty nothing to pop");
        return;
    }
    else{
        struct Node* temp=head;
        head=head->next;
        free(temp);
    }
}

```

```
void top(){
    printf("%d\n",head->data);
}

int main(){
    int choice=0;

    while(choice!=6){
        printf("1.Create a Stack\n2.Push\n3.Pop\n4.Display\n5.Get top
element\n6.Exit\nEnter a choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1:{
                createSLL();
                printf("enter a value for first element: ");
                scanf("%d",&head->data);
                break;
            }

            case 2:{
                int data=0;
                printf("enter a value to push: ");
                scanf("%d",&data);
                push(head,data);
                break;
            }

            case 3:{
                pop();
                break;
            }

            case 4:{
                display(head);
                break;
            }

            case 5:
            {
                top();
                break;
            }

        }
    }
}
```

```
    return 0;  
}
```

C:\Users\kjsce_it114\Desktop\Untitled1.exe

```
4.Display  
5.Get top element  
6.Exit  
Enter a choice: 1  
enter a value for first element: 5  
1.Create a Stack  
2.Push  
3.Pop  
4.Display  
5.Get top element  
6.Exit  
Enter a choice: 2  
enter a value to push: 50  
1.Create a Stack  
2.Push  
3.Pop  
4.Display  
5.Get top element  
6.Exit  
Enter a choice: 4  
5 50  
1.Create a Stack  
2.Push  
3.Pop  
4.Display  
5.Get top element  
6.Exit  
Enter a choice: 5  
5  
1.Create a Stack  
2.Push  
3.Pop  
4.Display  
5.Get top element  
6.Exit  
Enter a choice: 3  
1.Create a Stack  
2.Push  
3.Pop  
4.Display  
5.Get top element  
6.Exit  
Enter a choice: 4  
50  
1.Create a Stack  
2.Push  
3.Pop  
4.Display  
5.Get top element  
6.Exit  
Enter a choice: _
```

Program 2:

```
#include<stdio.h>

struct node
{
    char data;
    struct node *next;
};
typedef struct node node;

node *top;

node *top1;

void createSSL()
{
    top = NULL;
}

void push(char value)
{
    node *tmp;
    tmp = malloc(sizeof(node));
    tmp -> data = value;
    tmp -> next = top;
    top = tmp;
}

char pop()
{
    node *tmp;
    char n;
    tmp = top;
    n = tmp->data;
    top = top->next;
    free(tmp);
    return n;
}

char peek()
```



```
{
    return top->data;
}
int isEmpty()
{
    if(top==NULL){
        return 1;
    }
    else{
        return 0;
    }
}

void display()
{
    top1=top;
    if(isEmpty())
    {
        printf("stack is empty: ");
        return;
    }
    while(!isEmpty())
    {
        printf("%c",top1->data);
        top1=top1->next;
    }
}

int isOperand(char ch)
{
    return (ch>='a' && ch<='z')||(ch>='A' && ch<='Z');
}

int Prec(char ch)
{
    switch(ch)
    {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
    }
}
```

```

        case '^':
            return 3;
    }
}

int infixToPostfix(char exp[100])
{
    int i,k;
    createSSL();

    for(i=0,k=-1;exp[i];++i)
    {
        if(isOperand(exp[i]))
        {
            return exp[++k]=exp[i];
        }
        else if(exp[i]=='(')
        {
            push(exp[i]);
        }
        else if(exp[i]==')')
        {
            while(!isEmpty() && peek()!='(')
            {
                exp[++k]=pop();
            }
            if(!isEmpty() && peek()!='(')
            {
                return -1;
            }
            else{
                pop();
            }
        }
        else
        {
            while(isEmpty()&& Prec(exp[i])<=Prec(peek()))
            {
                exp[++k]=pop();
            }
            push(exp[i]);
        }
    }
}

```

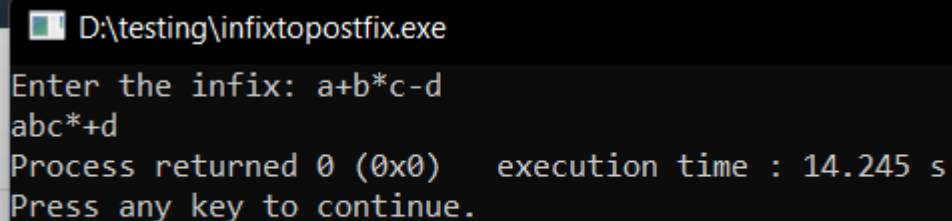
```

    }
}
while(!isEmpty())
{
    exp[++k]=pop();
}
exp[++k]='\0';
printf("%s",exp);
}

int main()
{
    char infix[100];
    printf("Enter the infix: ");
    scanf("%s",infix);
    infixToPostfix(infix);

}

```



D:\testing\infixtopostfix.exe

Enter the infix: a+b*c-d

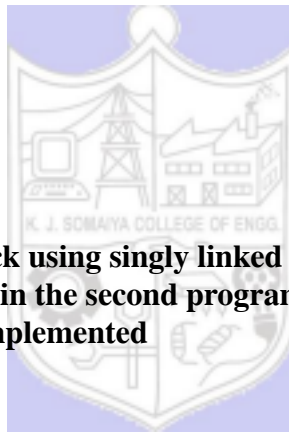
abc*+d

Process returned 0 (0x0) execution time : 14.245 s

Press any key to continue.

Outcomes:

CO2: Apply linear and non-linear data structure in application development.



Conclusion: We implemented Stack using singly linked list by creating functions such as push, pop, display and peek. Then in the second program, we converted an infix string to postfix string using the stack we implemented

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

- Y. Langsam, M. Augenstein and A. Tannenbaum, “Data Structures using C”, Pearson Education Asia, 1st Edition, 2002