VIVEKANANDA COLLEGE

THAKURPUKUR

KOLKATA-700063

NAAC ACCREDITED 'A' GRADE

Topic: Array and linked representation of stack

Course Title: Data Structure

Paper: CMS-A-CC-2-3-TH

Unit: Stack

Semester: 2nd Semester

Name of the Teacher: Amitava Biswas

Name of the Department: Computer Science

Definition:

A stack is a linear data structure in which all the insertion and deletion of data or you can say its values are done at one end only, rather than in the middle. Stacks can be implemented by using arrays of type linear.

The stack is mostly used in converting and evaluating expressions in Polish notations, i.e.:

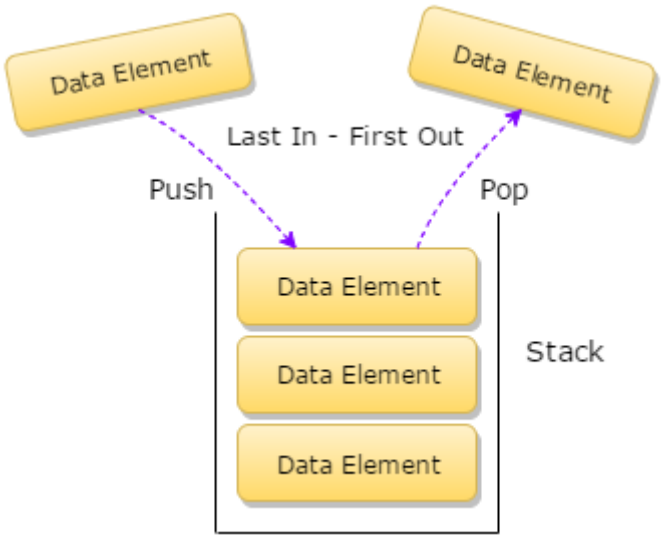
- **Infix**
- **Prefix**
- **Postfix**

In case of arrays and linked lists, these two allows programmers to insert and delete elements from any place within the list, i.e., from the beginning or the end or even from the middle also. But in computer programming and development, there may arise some situations where insertion and deletion require only at one end wither at the beginning or end of the list. The stack is a linear data structure, and all the insertion and deletion of its values are done in the same end which is called the top of the stack. Let us suppose take the real-life example of a stack of plates or a pile of books etc. As the item in this form of data structure can be removed or added from the top only which means the last item to be added to the stack is the first item to be removed. So you can say that the stack follows the Last In First Out (LIFO) structure.

Stack as ADT:

Stacks of elements of any particular type is a finite sequence of elements of that type together with the following operations:

- Initialize the stack to be empty
- Determine whether the stack is empty or not
- Check whether the stack is full or not
- If the stack is not full, add or insert a new node at the top of the stack. This operation is termed as Push Operation
- If the stack is not empty, then retrieve the node at its top
- If the stack is not empty, the delete the node at its top. This operation is called as Pop operation



C Examples on Stack Implementation

A Stack is a data structure which is used to store data in a particular order. Two operations that can be performed on a Stack are: Push operation which inserts an element into the stack. Pop operation which removes the last element that was added into the stack. It follows Last In First Out(LIFO) Order. The C programs in this section dealing with Stack. The section deals with various implementations of Stacks, to reverse a Stack using recursion and without using recursion, to implement two Stacks using a single array and check for Overflow and Underflow conditions and implementing a Stack using linked list. A stack overflow is an undesirable condition in which the program tries to use more memory space than the call stack has available. If a Stack is empty and yet a Pop operation is attempted, then it results in Stack Underflow condition.

Problem Description

This program implements the stack operation.

Problem Solution

1. Use three functions for three operations like push, pop and display.
2. Use switch statement to access these functions.
3. Exit.

Program/Source Code

Here is source code of the C program to implement a stack. The C program is successfully compiled and run on a Linux system. The program output is also shown below.

```
/*
 * C program to implement stack. Stack is a LIFO data structure.
```

```
 * Stack operations: PUSH(insert operation), POP(Delete operation)
 * and Display stack.
 */
#include <stdio.h>
#define MAXSIZE 5

struct stack
{
   int stk[MAXSIZE];
   int top;
};
typedef struct stack STACK;
STACK s;

void push(void);
int  pop(void);
void display(void);

void main ()
{
   int choice;
   int option = 1;
   s.top = -1;

   printf ("STACK OPERATION\n");
   while (option)
   {
      printf ("----------------------------------------\n");
      printf ("      1    -->    PUSH              \n");
      printf ("      2    -->    POP             \n");
      printf ("      3    -->    DISPLAY              \n");
      printf ("      4    -->    EXIT          \n");
      printf ("----------------------------------------\n");

      printf ("Enter your choice\n");
      scanf    ("%d", &choice);
      switch (choice)
```

```
        {
        case 1:
            push();
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            return;
        }
        fflush (stdin);
        printf ("Do you want to continue(Type 0 or 1)?\n");
        scanf    ("%d", &option);
    }
}
/*  Function to add an element to the stack */
void push ()
{
    int num;
    if (s.top == (MAXSIZE - 1))
    {
        printf ("Stack is Full\n");
        return;
    }
    else
    {
        printf ("Enter the element to be pushed\n");
        scanf ("%d", &num);
        s.top = s.top + 1;
        s.stk[s.top] = num;
    }
    return;
}
/*  Function to delete an element from the stack */
```

```
int pop ()
{
    int num;
    if (s.top == - 1)
    {
        printf ("Stack is Empty\n");
        return (s.top);
    }
    else
    {
        num = s.stk[s.top];
        printf ("poped element is = %dn", s.stk[s.top]);
        s.top = s.top - 1;
    }
    return(num);
}
/*  Function to display the status of the stack */
void display ()
{
    int i;
    if (s.top == -1)
    {
        printf ("Stack is empty\n");
        return;
    }
    else
    {
        printf ("\n The status of the stack is \n");
        for (i = s.top; i >= 0; i--)
        {
            printf ("%d\n", s.stk[i]);
        }
    }
    printf ("\n");
}
```

Program Explanation

1. Ask the user for the operation like push, pop, display and exit. Use the variable top to represent the top of the stack.

2. According to the option entered, access its respective function using switch statement.

3. In the function push(), firstly check if the stack is full. If it is, then print the output as “Stack is Full”. Otherwise take the number to be inserted as input and store it in the variable num. Copy the number to the array stk[] and increment the variable top by 1.

4. In the function pop(), firstly check if the stack is empty. If it is, then print the output as “Stack is Empty”. Otherwise print the top most element of the array stk[] and decrement the variable top by 1.

5. In the function display(), using for loop print all the elements of the array.

6. Exit.

Runtime Test Cases

```
STACK OPERATION
----------------------------------------
    1   -->   PUSH
    2   -->   POP
    3   -->   DISPLAY
    4   -->   EXIT
----------------------------------------
Enter your choice
1
Enter the element to be pushed
34
Do you want to continue(Type 0 or 1)?
0
$ a.out
STACK OPERATION
----------------------------------------
    1   -->   PUSH
    2   -->   POP
    3   -->   DISPLAY
    4   -->   EXIT
----------------------------------------
Enter your choice
1
Enter the element to be pushed
34
```

```
Do you want to continue(Type 0 or 1)?
1
-----------------------------------------
    1   -->   PUSH
    2   -->   POP
    3   -->   DISPLAY
    4   -->   EXIT
-----------------------------------------
Enter your choice
2
poped element is = 34
Do you want to continue(Type 0 or 1)?
1
-----------------------------------------
    1   -->   PUSH
    2   -->   POP
    3   -->   DISPLAY
    4   -->   EXIT
-----------------------------------------
Enter your choice
3
Stack is empty
Do you want to continue(Type 0 or 1)?
1
-----------------------------------------
    1   -->   PUSH
    2   -->   POP
    3   -->   DISPLAY
    4   -->   EXIT
-----------------------------------------
Enter your choice
1
Enter the element to be pushed
50
Do you want to continue(Type 0 or 1)?
1
-----------------------------------------
```

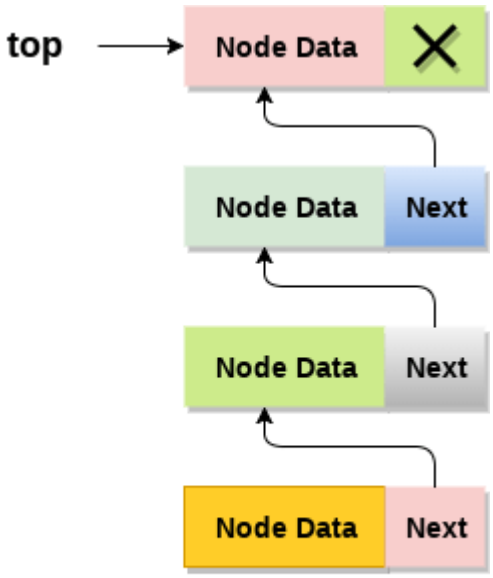
```
    1   -->   PUSH
    2   -->   POP
    3   -->   DISPLAY
    4   -->   EXIT
----------------------------------------
Enter your choice
1
Enter the element to be pushed
60
Do you want to continue(Type 0 or 1)?
1
----------------------------------------
    1   -->   PUSH
    2   -->   POP
    3   -->   DISPLAY
    4   -->   EXIT
----------------------------------------
Enter your choice
3

The status of the stack is
60
50

Do you want to continue(Type 0 or 1)?
1
----------------------------------------
    1   -->   PUSH
    2   -->   POP
    3   -->   DISPLAY
    4   -->   EXIT
----------------------------------------
Enter your choice
4
```

C Program to Implement a Stack using Linked List:

This C Program implement a stack using linked list. Stack is a type of queue that in practice is implemented as an area of memory that holds all local variables and parameters used by any function, and remembers the order in which functions are called so that function returns occur correctly. Each time a function is called, its local variables and parameters are “pushed onto” the stack. When the function returns,these locals and parameters are “popped.” Because of this, the size of a program’s stack fluctuates constantly as the program is running, but it has some maximum size. stack is as a last in, first out (LIFO) abstract data type and linear data structure. Linked list is a data structure consisting of a group of nodes which together represent a sequence. Here we need to apply the application of linkedlist to perform basic operations of stack.

Instead of using array, we can also use linked list to implement stack. Linked list allocates the memory dynamically. However, time complexity in both the scenario is same for all the operations i.e. push, pop and peek.

In linked list implementation of stack, the nodes are maintained non-contiguously in the memory. Each node contains a pointer to its immediate successor node in the stack. Stack is said to be overflown if the space left in the memory heap is not enough to create a node.



Stack

Here is source code of the C Program to implement a stack using linked list. The C program is successfully compiled and run on a Linux system. The program output is also shown below.

```
/*
 * C Program to Implement a Stack using Linked List
 */
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int info;
    struct node *ptr;
}*top,*top1,*temp;

int topelement();
void push(int data);
void pop();
```

```
void empty();
void display();
void destroy();
void stack_count();
void create();

int count = 0;

void main()
{
    int no, ch, e;

    printf("\n 1 - Push");
    printf("\n 2 - Pop");
    printf("\n 3 - Top");
    printf("\n 4 - Empty");
    printf("\n 5 - Exit");
    printf("\n 6 - Dipslay");
    printf("\n 7 - Stack Count");
    printf("\n 8 - Destroy stack");

    create();

    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);

        switch (ch)
        {
        case 1:
            printf("Enter data : ");
            scanf("%d", &no);
```

```
            push(no);
            break;
        case 2:
            pop();
            break;
        case 3:
            if (top == NULL)
                printf("No elements in stack");
            else
            {
                e = topelement();
                printf("\n Top element : %d", e);
            }
            break;
        case 4:
            empty();
            break;
        case 5:
            exit(0);
        case 6:
            display();
            break;
        case 7:
            stack_count();
            break;
        case 8:
            destroy();
            break;
        default :
            printf(" Wrong choice, Please enter correct choice  ");
            break;
        }
    }
```

```
}

/* Create empty stack */
void create()
{
    top = NULL;
}

/* Count stack elements */
void stack_count()
{
    printf("\n No. of elements in stack : %d", count);
}

/* Push data into stack */
void push(int data)
{
    if (top == NULL)
    {
        top =(struct node *)malloc(1*sizeof(struct node));
        top->ptr = NULL;
        top->info = data;
    }
    else
    {
        temp =(struct node *)malloc(1*sizeof(struct node));
        temp->ptr = top;
        temp->info = data;
        top = temp;
    }
    count++;
}
```

```
/* Display stack elements */
void display()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("Stack is empty");
        return;
    }

    while (top1 != NULL)
    {
        printf("%d ", top1->info);
        top1 = top1->ptr;
    }
 }

/* Pop Operation on stack */
void pop()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("\n Error : Trying to pop from empty stack");
        return;
    }
    else
        top1 = top1->ptr;
    printf("\n Popped value : %d", top->info);
    free(top);
    top = top1;
```

```
    count--;
}

/* Return top element */
int topelement()
{
    return(top->info);
}

/* Check if stack is empty or not */
void empty()
{
    if (top == NULL)
        printf("\n Stack is empty");
    else
        printf("\n Stack is not empty with %d elements", count);
}

/* Destroy entire stack */
void destroy()
{
    top1 = top;

    while (top1 != NULL)
    {
        top1 = top->ptr;
        free(top);
        top = top1;
        top1 = top1->ptr;
    }
    free(top1);
    top = NULL;
```

```
    printf("\n All stack elements destroyed");
    count = 0;
}
$ cc pgm2.c
$ a.out

1 - Push
2 - Pop
3 - Top
4 - Empty
5 - Exit
6 - Dipslay
7 - Stack Count
8 - Destroy stack
Enter choice : 1
Enter data : 56

Enter choice : 1
Enter data : 80

Enter choice : 2

Popped value : 80
Enter choice : 3

Top element : 56
Enter choice : 1
Enter data : 78

Enter choice : 1
Enter data : 90

Enter choice : 6
```

90 78 56
Enter choice : 7

No. of elements in stack : 3
Enter choice : 8

All stack elements destroyed
Enter choice : 4

Stack is empty
Enter choice : 5