Navlakhi®

Stacks

Methodology and Program

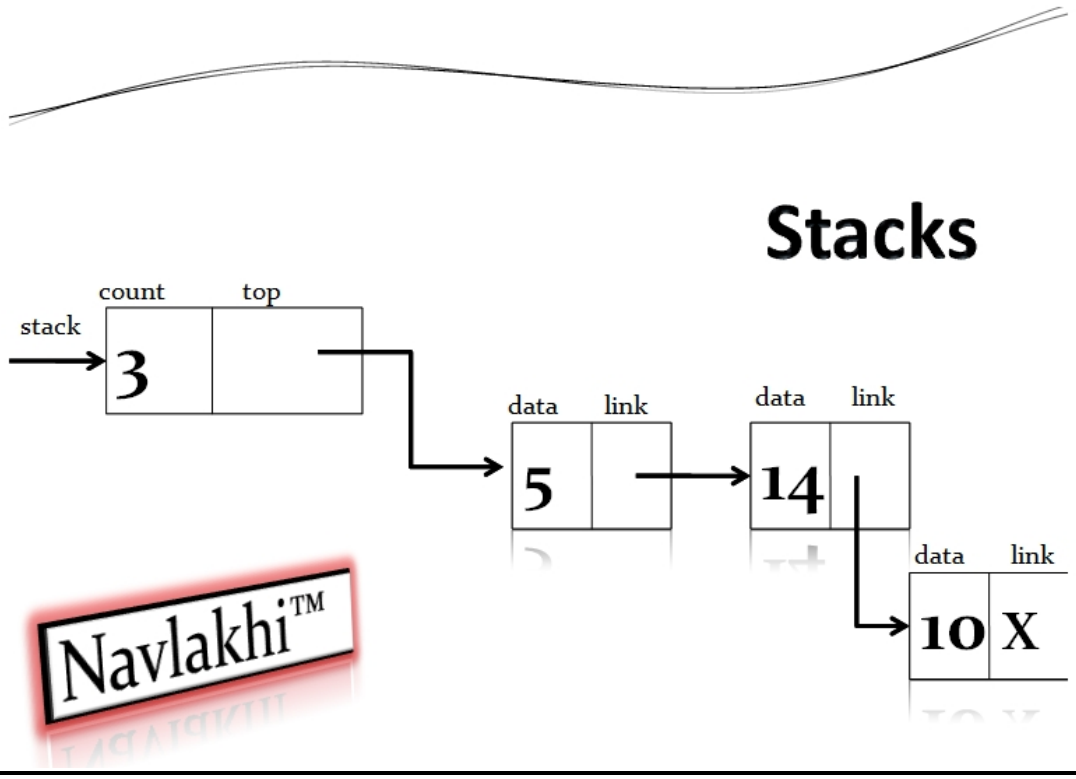
By Abhishek Navlakhi

Semester 3: Data Structures

Linked List implementation of stacks is similar in structure to the Linked List. The 'head' pointer now is called the 'top' pointer & adding a node is called 'push', removal is called 'pop'. New node is always added on top & removal of node too is done from the top. Here user does not have the choice to what data he/she can remove.

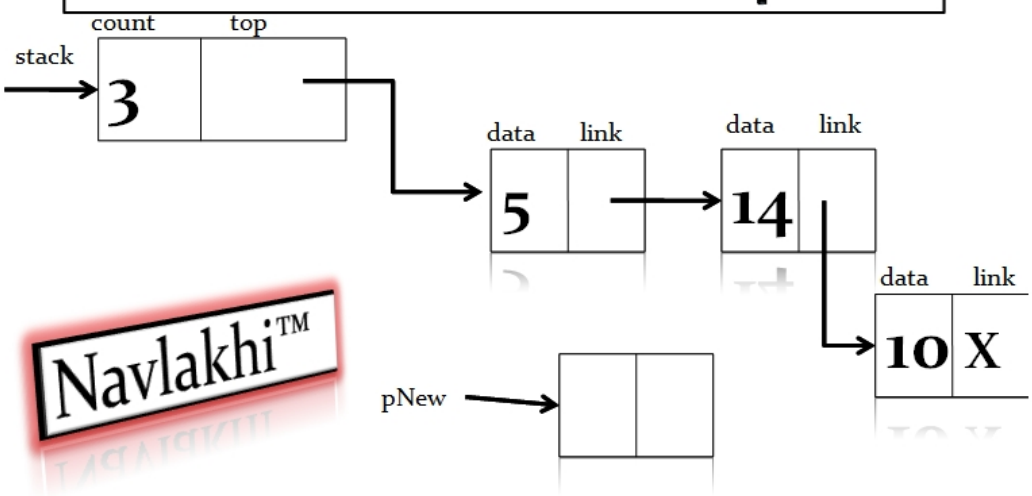
Lets lay down a few programming basics

ADDITION OF DATA (PUSH)



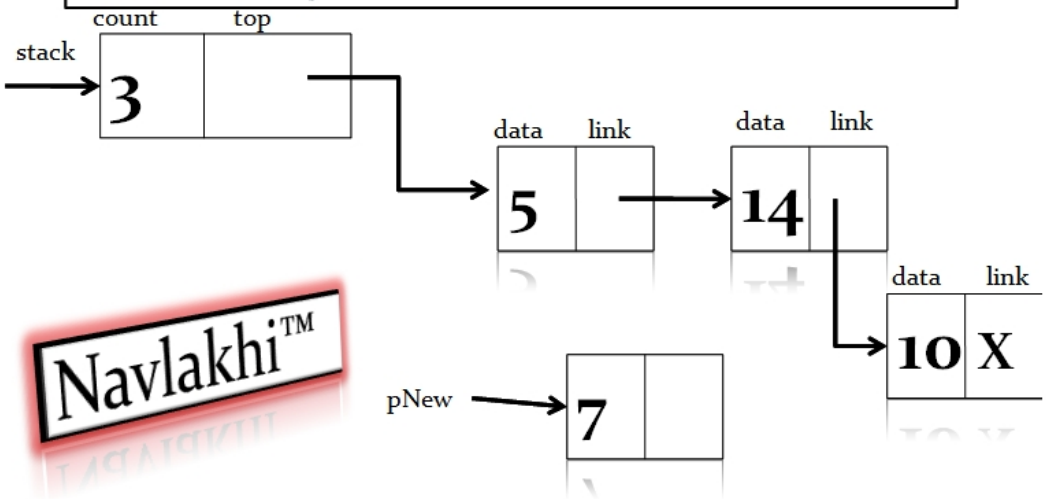
Stacks- Adding 7 PUSH

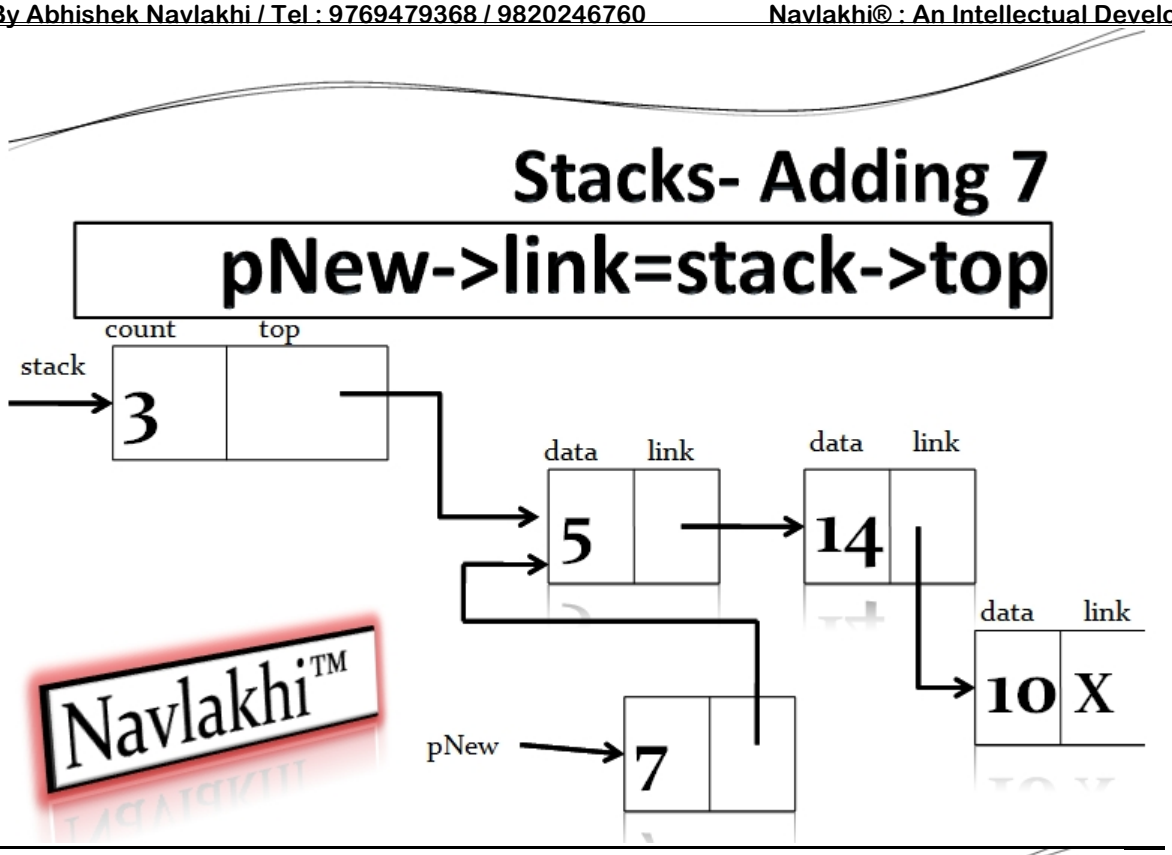
malloc pNew

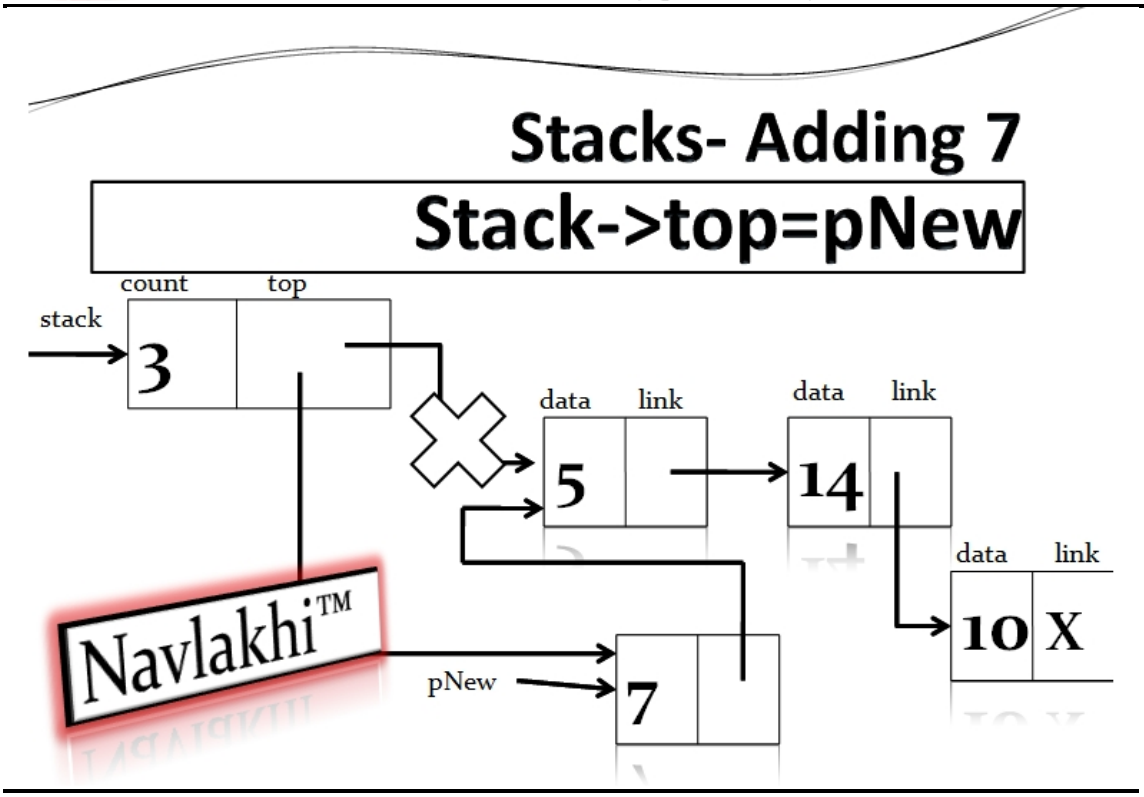


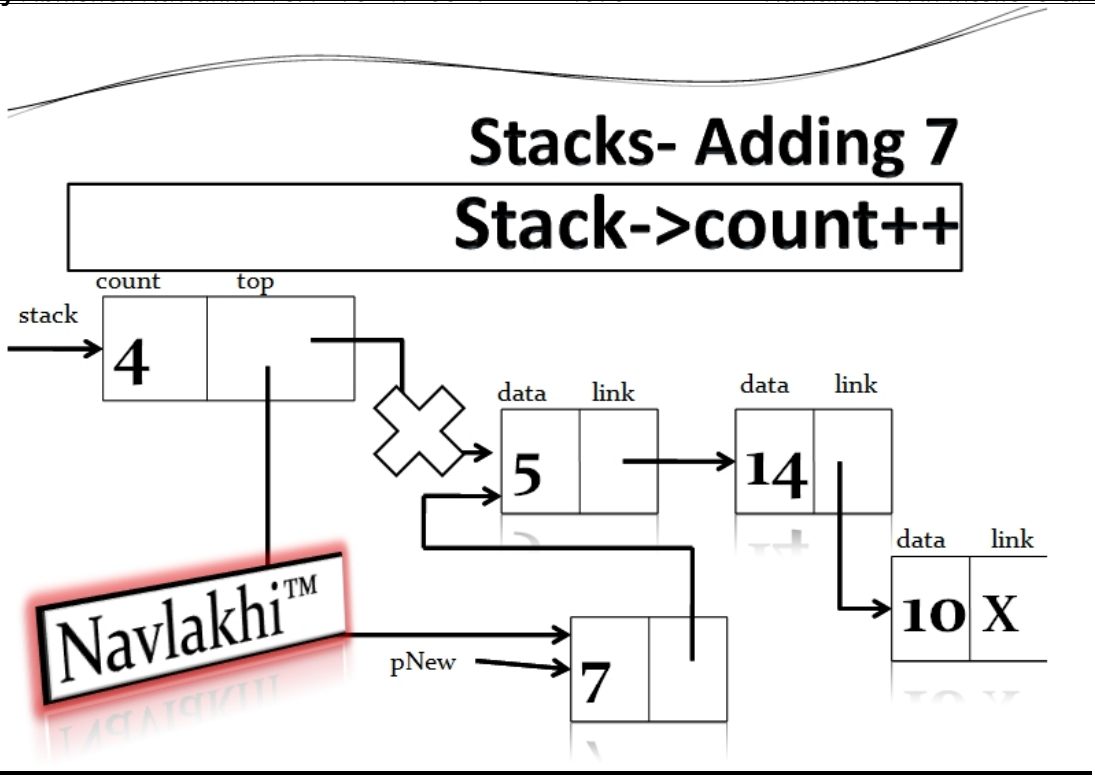
Stacks- Adding 7

pNew->data=dataIn

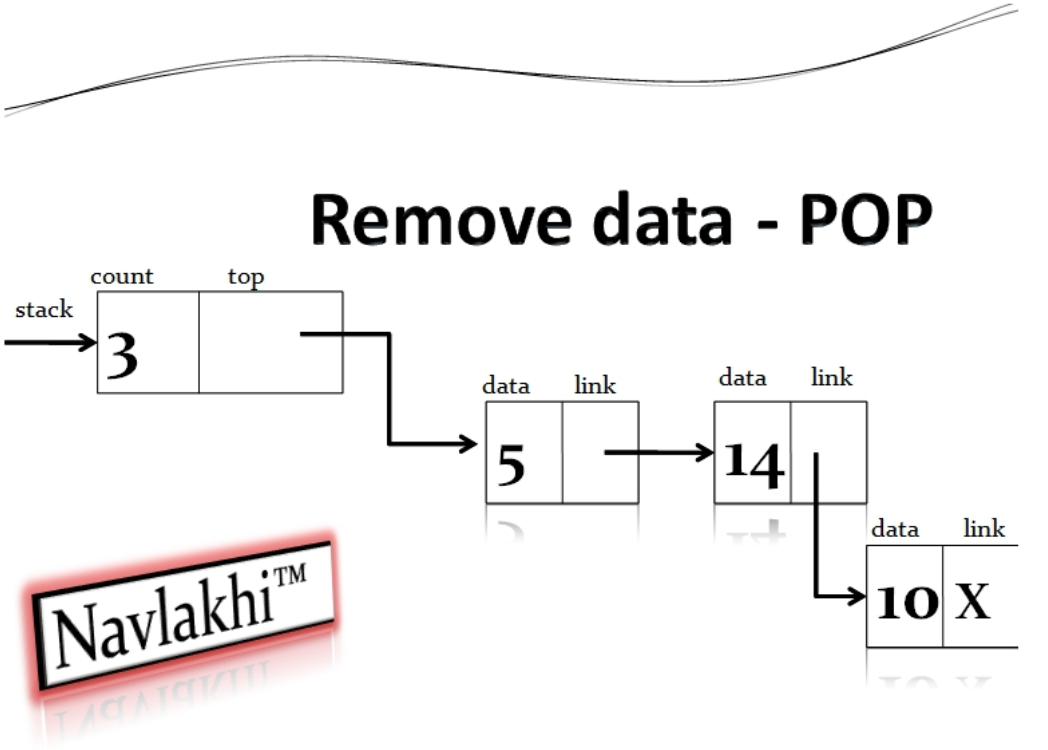


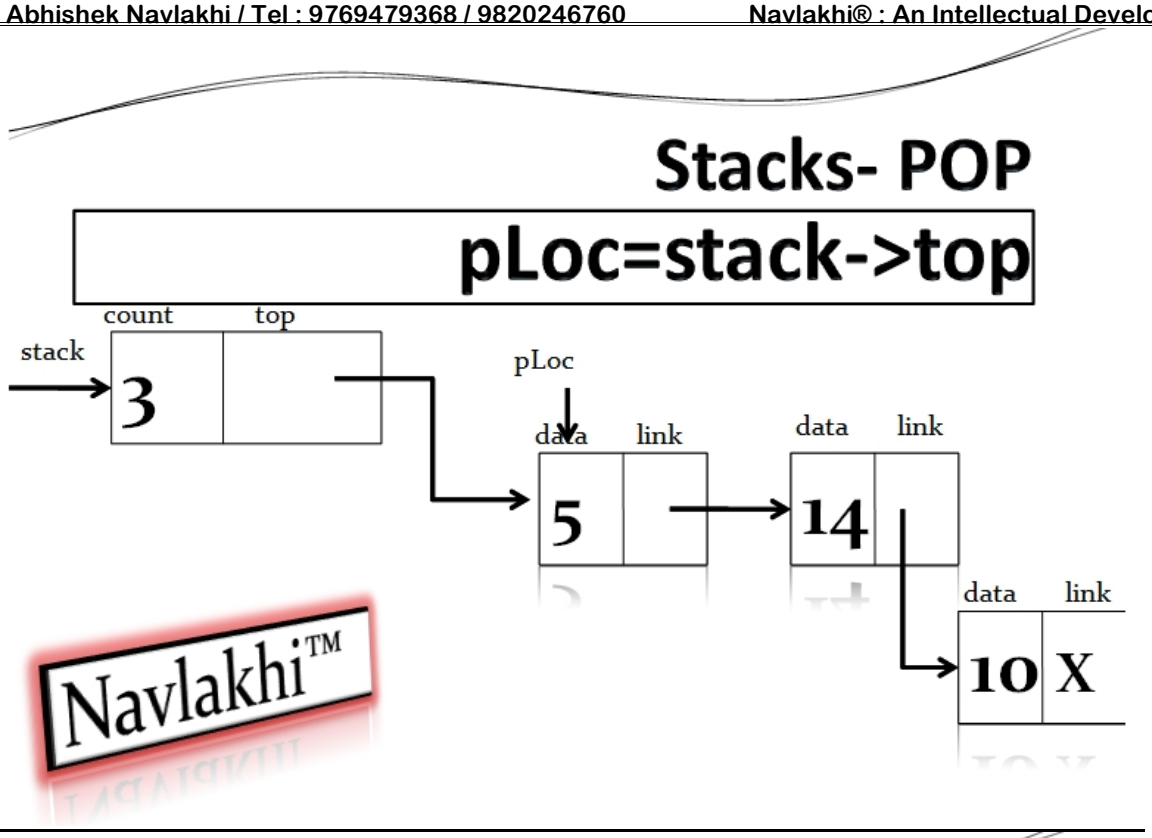

Stacks- Adding 7
pNew->link=stack->top
count
top
stack
3
data
link
5
data
link
14
data
link
10
X
pNew
7
Navlakhi™

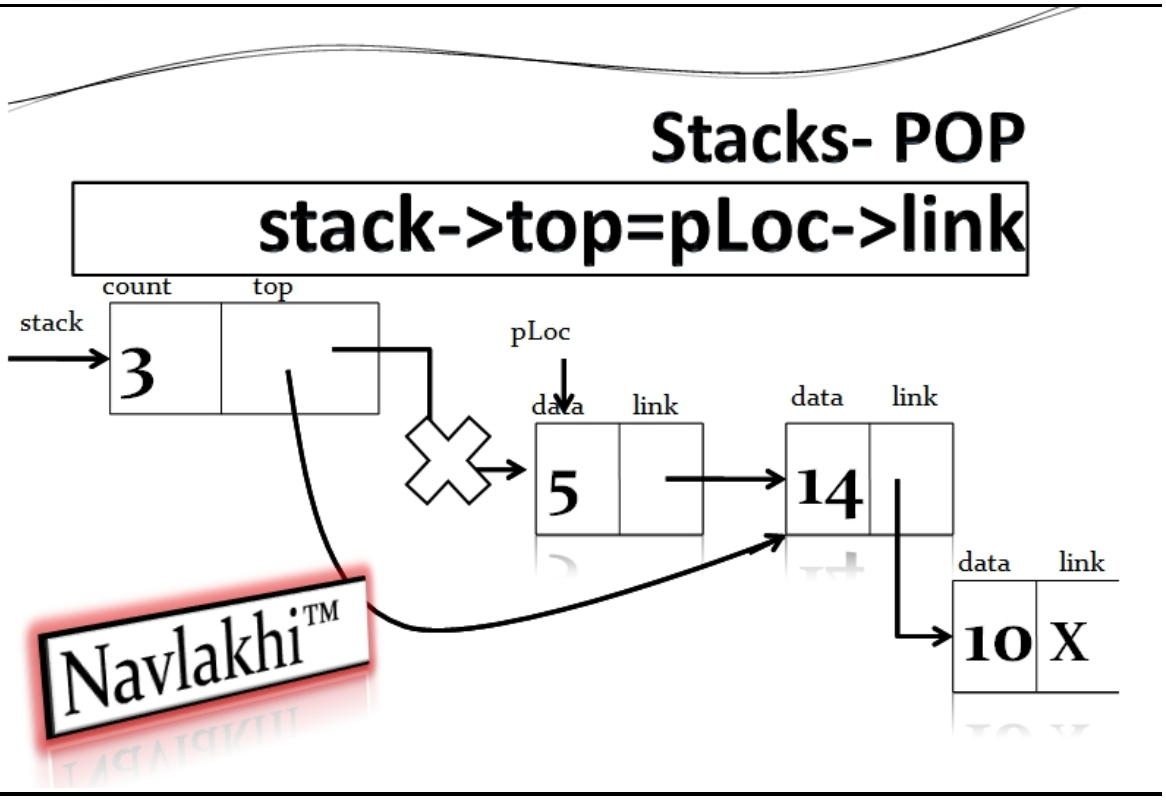

Stacks- Adding 7
Stack->top=pNew
count
top
stack
3
data
link
5
data
link
14
data
link
10
X
pNew
7
Navlakhi™

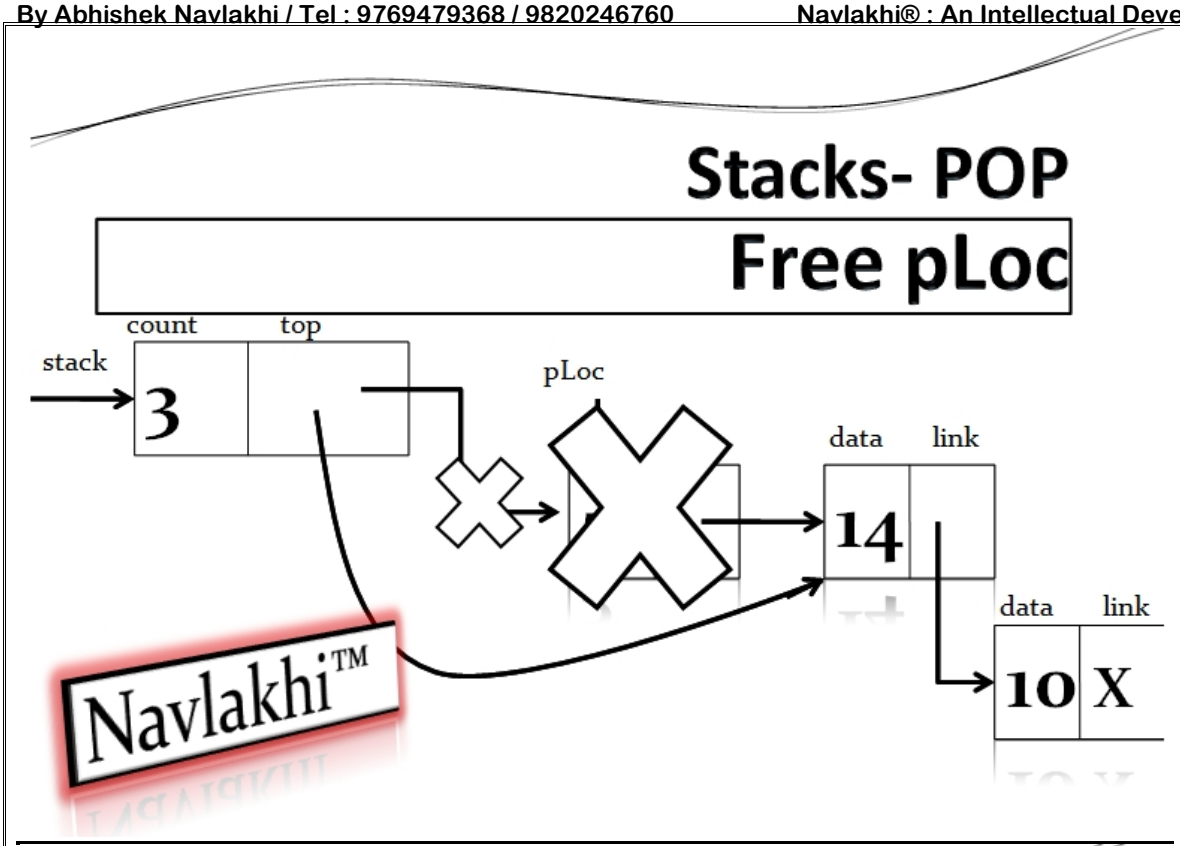


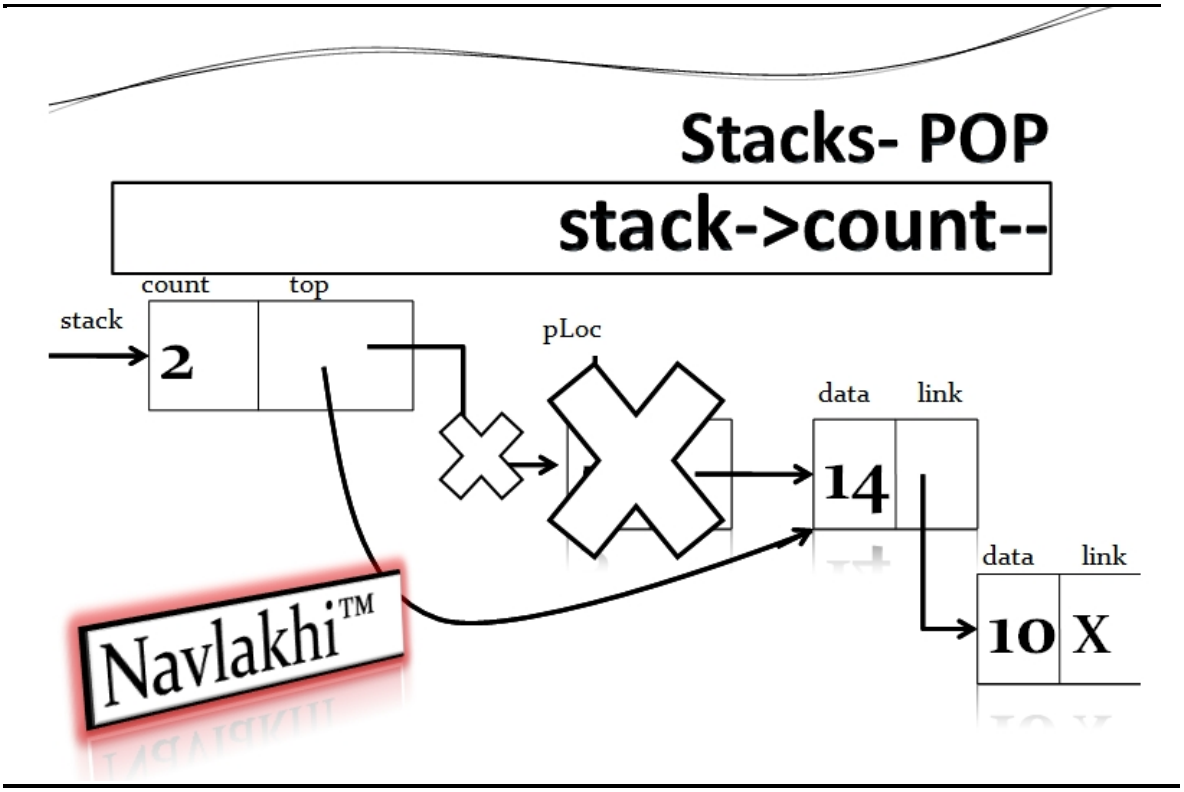
DELETING A NODE




Stacks- POP
pLoc=stack->top
count
top
stack
3
pLoc
data
link
5
data
link
14
data
link
10
X
Navlakhi™


Stacks- POP
stack->top=pLoc->link
count
top
stack
3
pLoc
data
link
5
data
link
14
data
link
10
X
Navlakhi™


Stacks- POP
Free pLoc
count
top
stack
3
pLoc
data
link
14
data
link
10
X
Navlakhi™


Stacks- POP
stack->count--
count
top
stack
2
pLoc
data
link
14
data
link
10
X
Navlakhi™

Program

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

struct node
{
int data;
struct node *link;
};

struct STACK
{
int count;
struct node *top;
}*stack;


void destroyStack( )
{
        struct node *pLoc;
        while(stack->count!=0)
        {
                pLoc=stack->top;
                stack->top=pLoc->link; /*OR stack->top=stack->top->link*/
                stack->count-= 1;
                free(pLoc);
        }
        free(stack);
}
```

```
int stackTop(int *dataIn)
{
if (stack->count!=0)
{
        *dataIn=stack->top->data;
        return 1;
}

else return 0;
}

int popStack(int *dataPtr)
{
 struct node *pLoc;

 if(stack->count==0)  /* OR stack->top==NULL*/
        return 0;

 pLoc=stack->top;
 *dataPtr=pLoc->data;
 stack->top=pLoc->link; /* OR stack->top=stack->top->link*/
 free(pLoc);
 stack->count-=1;
 return 1;
}

int pushStack(int dataIn)
{
struct node *pNew;
pNew=(struct node *)malloc(sizeof(struct node));
if (pNew==NULL) return 0;

pNew->data=dataIn;
pNew->link=stack->top;
stack->top=pNew;
stack->count+=1;
return 1;
}
```

```
void createStack( )
{
stack=(struct STACK *)malloc(sizeof(struct STACK));
if (stack==NULL)
{
        printf("Insufficient Memory.... Exiting Program\n");
        exit(1);
}

/* Memory creation for head node successful */
stack ->top=NULL;
stack->count=0;

}



int menu( )
{
int choice;
printf("\n\n");
printf("1. Push Data onto the Stack\n");
printf("2. Pop Data from the Stack\n");
printf("3. Check number of data items on Stack\n");
printf("4. Check the stack top element\n");
printf("5. Exit\n\n");

printf("Feed in your choice: ");
scanf("%d",&choice);
return choice;
}
```

```
void main( )
{
int choice,data,success;

createStack();

do
{
choice=menu();

switch (choice)
{
case 1: printf("feed in the data to insert onto the Stack: ");
        scanf("%d",&data);
        success=pushStack(data);
        if (success) printf("Data Inserted Successfully\n");
        else printf("Memory Overflow\n");
        break;

case 2:         success=popStack(&data);
        if (success) printf("Data pop'ed of Stack is %d\n",data);
        else printf("Underflow... no data to pop\n");
        break;

case 3:         printf("Stack count= %d\n",stack->count);
        break;

case 4: success= stackTop(&data);
        if (success==1) printf("The stack top element is %d\n",data);
        else printf("Stack is empty\n");
        break;
}
}while (choice !=5);
destroyStack();
}
```

HOME OF EDUCATION

Navlakhi®

www.navlakhi.com
Home of Education

DATA Structures@ Navlakhi's