The Stack data structure

A stack is one of the most important and useful non-primitive linear data structure in computer science. It is an ordered collection of items into which new data items may be added/inserted and from which items may be deleted at only one end, called the top of the stack. As all the addition and deletion in a stack is done from the top of the stack, the last added element will be first removed from the stack. That is why the stack is also called Last-in-First-out (LIFO). Note that the most frequently accessible element in the stack is the top most elements, whereas the least accessible element is the bottom of the stack. The operation of the stack can be illustrated as in Fig. 3.1.

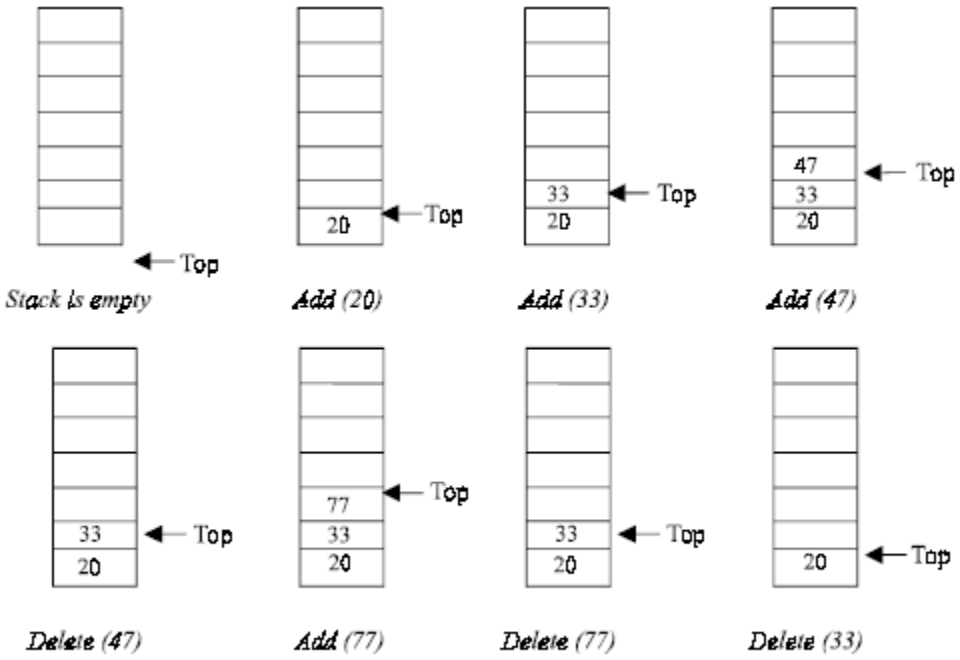

Fig. 3.1. Stack operation.

The insertion (or addition) operation is referred to as push, and the deletion (or remove) operation as pop. A stack is said to be empty or underflow, if the stack contains no elements. At this point the top of the stack is present at the bottom of the stack. And it is overflow when the stack becomes full, i.e., no other elements can be pushed onto the stack. At this point the top pointer is at the highest location of the stack.

OPERATIONS PERFORMED ON STACK

The primitive operations performed on the stack are as follows:

PUSH: The process of adding (or inserting) a new element to the top of the stack is called PUSH operation. Pushing an element to a stack will add the new element at the top. After every push operation the top is incremented by one. If the array is full and no new element can be accommodated, then the stack overflow condition occurs.

POP: The process of deleting (or removing) an element from the top of stack is called POP operation. After every pop operation the stack is decremented by one. If there is no element in the stack and the pop operation is performed then the stack underflow condition occurs.

STACK IMPLEMENTATION

Stack can be implemented in two ways:

1. Static implementation (using arrays)

2. Dynamic implementation (linked list)

Static implementation uses arrays to create stack. Static implementation using arrays is a very simple technique but is not a flexible way, as the size of the stack has to be declared during the program design, because after that, the size cannot be varied (i.e., increased or decreased). Moreover static implementation is not an efficient method when resource optimization is concerned (i.e., memory utilization). For example a stack is implemented with array size 50. That is before the stack operation begins, memory is allocated for the array of size 50. Now if there are only few elements (say 30) to be stored in the stack, then rest of the statically allocated memory (in this case 20) will be wasted, on the other hand if there are more number of elements to be stored in the stack (say 60) then we cannot change the size array to increase its capacity. The above said limitations can be overcome by dynamically implementing (is also called linked list representation) the stack using pointers.

STACK USING ARRAYS

Implementation of stack using arrays is a very simple technique. Algorithm for pushing (or add or insert) a new element at the top of the stack and popping (or delete) an element from the stack is given below.

Algorithm for push

Suppose STACK[SIZE] is a one dimensional array for implementing the stack, which will hold the data items. TOP is the pointer that points to the top most element of the stack. Let DATA is the data item to be pushed.

1. If TOP = SIZE – 1, then:
   (*a*) Display "The stack is in overflow condition"
   (*b*) Exit
2. TOP = TOP + 1
3. STACK [TOP] = ITEM
4. Exit

```
void push(void)
{
        int x;
        if(top==max-1)     // Condition for checking If Stack is Full
        {
                cout<<"\nstack overflow\n";
                return;
        }
        cout<<"enter a no:  ";
        cin>>x;
        a[++top]=x;    //increment the top and inserting element
        cout<< "\nsucc. pushed:  " << x << endl << endl;
             return;
}
```

Algorithm for pop

Suppose STACK[SIZE] is a one dimensional array for implementing the stack, which will hold the data items. TOP is the pointer that points to the top most element of the stack. DATA is the popped (or deleted) data item from the top of the stack.

1. If TOP is equal to -1, then
   (a) Display "The Stack is empty"
   (b) Exit
2. DATA = STACK[TOP]
3. TOP = TOP – 1
4. Exit

```
void pop(void)
{
        int y;
        if(top==-1)            // Condition for checking If Stack is Empty
        {
                cout <<"stack underflow\n";
                return;
        }
        y=a[top];
        a[top--]=0;     //insert 0 at place of removing element and decrement the top
        cout <<"\n succ.poped\n\n";
        return;
}
```

Algorithm for display

1. If TOP is equal to -1, then
   (a) Display "the stack is empty"
   (b) exit
2. i ←TOP to 0
   (a) display Array[i]
3. Exit

```
void display(void)
{
        if(top==-1)
        {
               cout <<"stack is empty\n";
               return;
        }
        cout<<"\nelements of Stack are : ";
        for(int i=0;i<=top;i++)
        {
                cout << a[i] << "  ";
         }
             cout << endl << endl;
         return;
}
```

STACK USING LINKED LIST

we have discussed the implementation of stack using array, i.e., static memory allocation. Implementation issues of the stack (Last In First Out - LIFO) using linked list is illustrated in following figures.

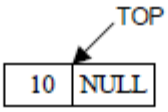

Fig. 5.11. push (10)

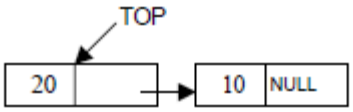

Fig. 5.12. push (20)

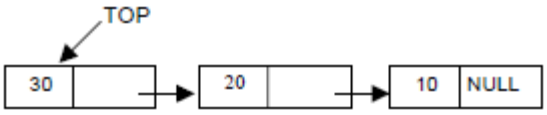

Fig. 5.13. push (30)

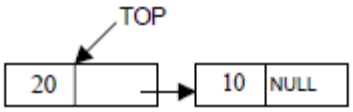

Fig. 5.14. X = pop() (ie; X = 30)

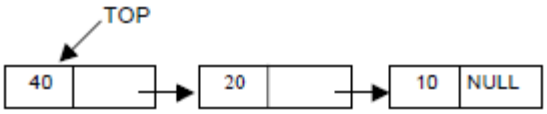

Fig. 5.15. push (40)

Algorithm for push operation

Suppose TOP is a pointer, which is pointing towards the topmost element of the stack. TOP is NULL when the stack is empty. DATA is the data item to be pushed.

1. Input the DATA to be pushed
2. Creat a New Node
3. NewNode → DATA = DATA
4. NewNode → Next = TOP
5. TOP = NewNode
6. Exit

```
void push()
{
     int item;
     Node *NewNode;
     NewNode = new Node;
     cout<<"\ninput the new value to be pushed on the stack: ";
     cin>>item;
     NewNode->info=item;
     NewNode->link=top;
     top=NewNode;
}
```

Algorithm for pop operation

Suppose TOP is a pointer, which is pointing towards the topmost element of the stack. TOP is NULL when the stack is empty. TEMP is pointer variable to hold any nodes address. DATA is the information on the node which is just deleted.

1. if (TOP is equal to NULL)
   (a) Display "The stack is empty"
2. Else
   (a) TEMP = TOP
   (b) Display "The popped element TOP → DATA"
   (c) TOP = TOP → Next
   (d) TEMP → Next = NULL
   (e) Free the TEMP node
3. Exit

```
void pop(){
if (top==NULL)
    cout<<"the stack is empty\n";
else
    {
       Node *temp=top;
       cout<<"the poped element is: "<<top->info;
       top=top->link;
       temp->link=NULL;
       delete temp;
    }
}
```

Algorithm for display operation

1. if (TOP is equal to NULL)
   (a) display "the stack is empty"
   (b) exit
2. else
   (a) temp = top
   (b) while temp is not equal to null
      (b.1) display temp->info
      (b.2) temp = temp->link
3. exit

```
void display(){
if(top==NULL)
    cout<<"the stack is empty"<<endl;
else
{
    cout<<"\nthe stack elements are: "<<endl;
    Node *temp=top;
    while(temp!=NULL)
    {
       cout<<temp->info;
       temp=temp->link;
    }
}
}
```