Stacks

4.1 INTRODUCTION

A stack is a non-primitive linear data structure. It is an ordered list in which addition of new data item and deletion of already existing data item is done from only one end, known as *top of stack* (TOS). As all the deletion and insertion in a stack is done from top of the stack, the last added element will be the first to be removed from the stack. That is the reason why stack is also called **Last-in-First-out (LIFO)** type of list. Note that the most frequently accessible element in the stack is the top most element, whereas the least accessible element is the bottom of the stack.

Example 1 : A common model of a stack is plates in a marriage a party or coin stacker. Fresh plates are "pushed" onto to the top and "popped" off the top.

Example 2 : Some of you may eat biscuits (or poppins). If you assume only one side of the cover is torn and biscuits are taken off one by one. This is what is called poping and similarly, if you want to preserve some biscuits for some time later, you will put them back into the pack through the same torn end called pushing.

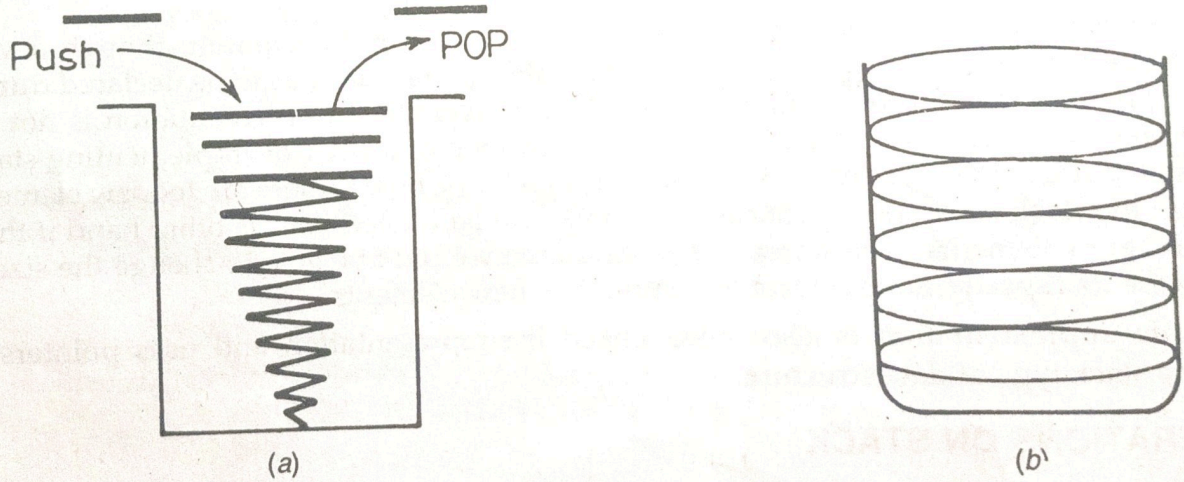

Fig. 4.1.

Whenever a stack is created, the stack base remains fixed, as a new element is added to the stack from the top, the top goes on increasing, conversely as the top most element of the stack is removed the stack top is decrementing. For example, Fig. 4.2 shows various stages of stack top, during insertion and during deletion.

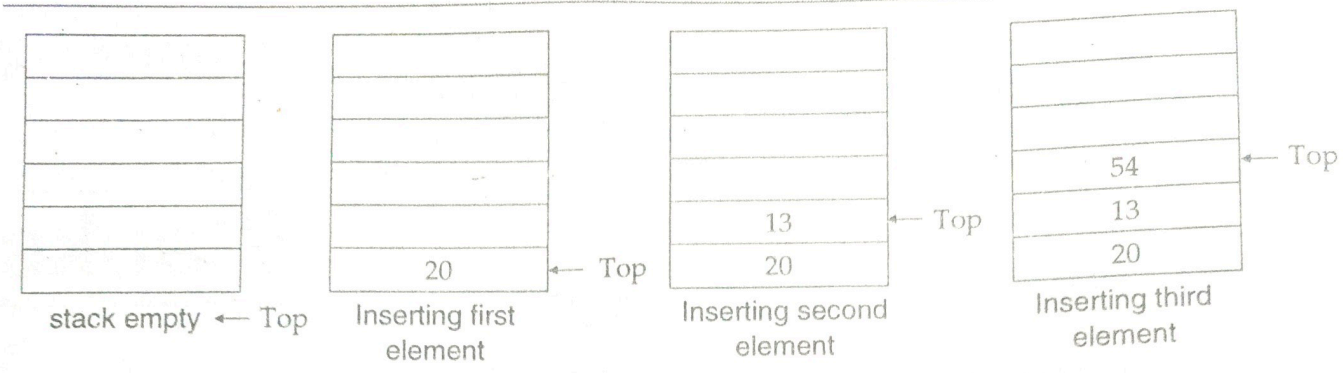

Fig. 4.2. (*a*) Stack top increases during insertion.

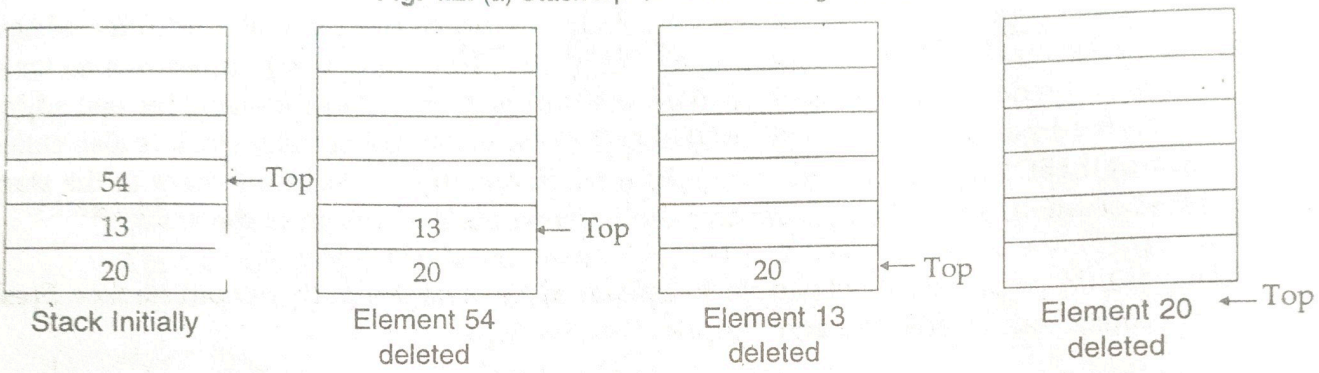

Fig. 4.2. (*b*) Stack top decreases during insertion.

4.2 STACK IMPLEMENTATION

Stack can be implemented in two ways :

(*i*) Static implementation. (*ii*) Dynamic implementation.

Static implementation uses arrays to create stack. Static implementation though a very simple technique but is not a flexible way of creation, as the size of stack has to be declared during program design, after that the size cannot be varied. Moreover static implementation is not too efficient with respect to memory utilization. As the declaration of array (for implementing stack) is done before the start of the operation (at program design time), now if there are too few elements to be stored in the stack the statically allocated memory will be wasted, on the other hand if there are more number of elements to be stored in the stack then we can't be able to change the size of array to increase its capacity, so that it can accommodate new elements.

Dynamic implementation is also called linked list representation and uses pointers to implement the stack type of data structure.

4.3 OPERATIONS ON STACK

The basic operations that can be performed on stack are as follows :

1. *PUSH.* The process of adding a new element to the top of stack is called PUSH operation. Pushing an element in the stack invoke adding of element, as the new element will be inserted at the top after every push operation the top is incremented by one. In case the array is full and no new element can be accommodated, it is called STACK-FULL condition. This condition is called STACK OVERFLOW.

2. *POP.* The process of deleting 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 on the stack and the pop is performed then this will result into stack underfow condition.

4.4 STACK TERMINOLOGY

1. *Context.* The environment in which a function executes : includes argument values, local variables and global variables. All the context except the global variables is stored in a stack frame.

2. *Stack frames.* The data structure containing all the data (arguments, local variables, return address, *etc*) needed each time a proccdure or function is called.

3. *MAXSIZE.* This term is not a standard one, we use this term to refer the maximum size of stack.

4. *TOP.* This term refers to the top of stack (TOS). The stack top is used to check stack overflow or underflow conditions. Initially Top stores – 1. This assumption is taken so that whenever an element is added to the stack the Top is first incremented and then the item is inserted into the location currently indicated by the Top.

5. *Stack.* It is an array of size MAXSIZE.

6. *Stack empty or underflow.* This is the situation when the stack contains no element. At this point the top of stack is present at the bottom of the stack.

7. *Stack overflow.* This is the situation when the stack becomes full, and no more elements can be pushed onto the stack. At this point the stack top is present at the highest location of the stack.

4.5 ALGORITHMS FOR PUSH AND POP

Algorithm

(*i*) *Algorithm for Inserting an item into the stack (PUSH)*

Let stack[MaxSize] is an array for implementing the stack.
1. [Check for stack overflow ?]
   If Top = Maxsize – 1, then : print : overflow and exit.
2. Set TOP = TOP + 1 [Increase TOP by 1]
3. Set STACK [TOP] := ITEM [Inserts Item in new TOP position]
4. Exit

The function for the Stack push operation in C is as follow :

```
void push( )
{
    int item ;
    if (top = = MAXSIZE – 1)
    {   printf ("\n The Stack Is Full") ;
        getch( ) ;
        exit( 0 ) ;
    }
    else
    {   printf ("Enter the element to be inserted") ;
        scanf("%d", &item) ;
        top = top + 1 ;
        stack [top] = item ;
    }
}
```

(ii) Algorithm for Deleting an element from the stack

This algorithm deletes the top element of stack and assign it to a variable Item.

1. [Check for the stack underflow]
   If Top < 0, then
   Print "Stack Underflow" and exit.
   Else [Remove the Top element]
   Set item = stack [Top].
2. Decrement the stack top
   Set Top = Top – 1
3. Return the deleted item from the stack
4. Exit.

The function of Pop operation is given as :

```
int pop( )
{
    int item ;
    if (top = = –1)
    {    printf ("The stack is Empty") ;
         getch( ) ;
         exit(0) ;
    }
    else
    {
         item = stack[Top] ;
         top = top – 1 ;
    }
    return (item) ;
}
```