# Ch: 3 Elementary Data Structure

## STACKS

A Stack is linear data structure. A stack is a list of elements in which an element may be inserted or deleted only at one end, called the **top of the stack**. Stack principle is **LIFO (last in, first out)**. Which element inserted last on to the stack that element deleted first from the stack.

As the items can be added or removed only from the top i.e. the last item to be added to a stack is the first item to be removed.

Real life examples of stacks are:

Stack of books

Stack of Plates

Stack of Toys

**Operations on stack:**

The two basic operations associated with stacks are:
1. Push
2. Pop

While performing push and pop operations the following test must be conducted on the stack.

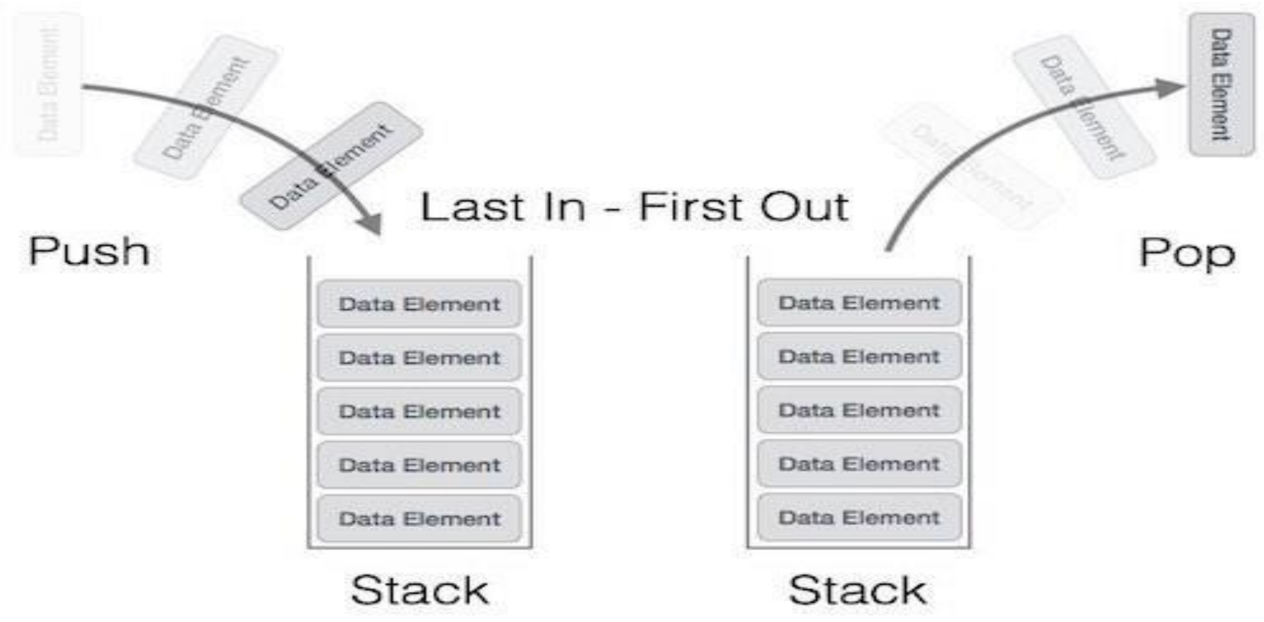
a) Stack is empty or not b) stack is full or not

**1. Push:**

Push operation is used to add new elements in to the stack. At the time of addition first check the stack is full or not. If the stack is full it generates an error message "stack overflow".

**2. Pop:**

Pop operation is used to delete elements from the stack. At the time of deletion first check the stack is empty or not. If the stack is empty it generates an error message "stack underflow". All insertions and deletions take place at the same end, so the last element added to the stack will be the first element removed from the stack. When a stack is created, the stack base remains fixed while the stack top changes as elements are added and removed. The most accessible element is the top and the least accessible element is the bottom of the stack.



- **Application of Stacks**

- Stack is used in the mobile. Message sent by one user and another message sent by same user after some time, in that case the message that was sent last by the user arrives in the inbox first.
- Consider a stack of plates placed on counter in a restaurant, During dinner time, plates are taken from the top of place(stack) and waiter puts the washed plates on the top of position which is known as TOS.
- The most important application of stack is recursion.
- It is also used in memory management and in operating systems

- **Types of stack/implementation of stack**

- ✓ Static Stack/Stack(array)
- ✓ Dynamic Stack(linked list)

## Static stack:

### 1. Algorithm for PUSH information static stack

- To insert an element on the stack we are using push operation.
- This operation inserts the element only on the TOS
- Algorithm (we have to define size first)

**Step-1**: First we have to check for stack overflow

```
If(tos>=size)
        Stack is full
```

**Step-2:** **Tos=tos+1**(increment the pointer value by 1)
**Step-3:** Stack[tos]=ele; where ele is an element that is to be inserted
**Step-4:** End.

## 2. Algorithm for POP information using static stack :

- To delete an element from the stack we are using pop operation.
- This operation removes or deletes the top most element from stack.

```
Step-1: First we have to check for stack underflow
        If(tos<0)
            Stack is empty.
Step – 2 : [Remove the Tos information]
        tos=tos-1.
Step – 3 :[Return the former information of the stack]    mean
        Print stack[tos]
Step-4:  End.
```

## 3.Algorithm to PEEP information using static stack

- If we want to access some information stored at some location in stack then peep operation is required.
- The index value is subtracted from tos
- Algorithm
- **Step-1:** If tos<0---stack is empty
- **Step-2:**Input the position of the element that you want to read
- **Step-3:**If(element<0 || element>tos+1)

  Out of range
- **Step-4:** Else Print peeped element

  Stack[tos-pos+1]
- **Step-5:** End

## 4) Algorithm to DISPLAY information using static stack:

[Display element]

- **Step-1:** If(tos<0)—stack is empty
- **Step-2:** else

```
for(i=tos;i>=0;i--)
        print element that is stack[i]
```

- **Step-3:** End

**Algorithm to UPDATE information using static Stack:**

```
Step -1 :[ input location to update]
          i=value

Step – 2 : [Chack for the stack underflow]
           If Tos-i+1<0 then
                Output "Stack underflow"
                exit
Step – 2 : [Chang the element]

           Stack[Tos-i+1] = value

Step – 3 :  return(stack[Tos-i+1])
```

**Program  for Static Stack:**

```
#include <stdio.h>

#include <stdlib.h>


#define size 5


int stack[size];

int tos = -1, flag = 0;


void push(int x);

int pop();

int peep();

int update();

void display();


int main() {

   int choice = 1, data;


   do {

      printf("\n\t==================");

      printf("\n\t\t Menu");

      printf("\n\t==================");

      printf("\n\t\t1. Push");

      printf("\n\t\t2. Pop");

      printf("\n\t\t3. Peep");
      printf("\n\t\t4. Update");
      printf("\n\t\t5. Display");
      printf("\n\t\t6. Exit");
      printf("\nEnter Your Choice: ");
      scanf("%d", &choice);

      switch (choice) {
          case 1: {
              printf("\n Push Operation Selected");
              printf("\nEnter Value to be Pushed: ");
              scanf("%d", &data);
              push(data);
              if (flag == 0) {
                  printf("\nStack Is Overflow");
              } else {
                  display();
                  if (tos == size - 1) {
                      printf("\nStack Full");
                  }
              }
              break;
          }
          case 2: {
              printf("\n Pop Operation Selected");
              data = pop();
              if (flag == 0) {
                  printf("\nStack Underflow");
              } else {
                  printf("\nPopped Element = %d", data);
                  if (tos == -1)
                      printf("\nStack is Empty");
              }
              break;
          }
          case 3: {
              printf("\n Peep Operation Selected");
              data = peep();
              if (flag) {
                  printf("\nPeeped Element: %d", data);
                  printf("\nStack is as follows:");
                  display();
              } else {
                  printf("\nStack Underflow");
              }
              break;
          }
          case 4: {
              printf("\n Update Operation Selected");
              data = update();
              if (flag) {
                  printf("\nUpdated Information: %d", data);
                  printf("\nStack is as follows:");
                  display();
              } else {
                  printf("\nStack Underflow");
              }
              break;
          }
          case 5: {
              printf("\n Display Selected");
              if (tos == -1) {
                  printf("\nStack is Empty");
              } else {
                  display();
              }
              break;
          }
          case 6: {
              printf("\nEnd of Program\n");
              exit(0);
          }
      }
  } while (choice != 6);

  printf("\nEnd Of Program\n");
```

```
    return 0;
}


void push(int x) {
    if (tos == size - 1) {
        printf("\nStack Overflow");
        flag = 0;
    } else {
        tos++;
        stack[tos] = x;
        flag = 1;
    }
}


int pop() {
    int popped_ele;
    if (tos == -1) {
        printf("\nStack Underflow");
        flag = 0;
        popped_ele = 0;
    } else {
        popped_ele = stack[tos];
        tos--;
        flag = 1;
    }
    return popped_ele;
}

int peep() {
    int i, peeped_element;
    printf("\nEnter the location of value which you want to peep: ");
    scanf("%d", &i);
    if (tos - i + 1 < 0) {
        printf("\nStack Underflow");
        flag = 0;
        peeped_element = 0;
    } else {
        flag = 1;
        peeped_element = stack[tos - i + 1];
    }
    return peeped_element;
}

int update() {
    int i, updated_element;
    printf("\nEnter the location of value which you want to update: ");
    scanf("%d", &i);
    if (tos - i + 1 < 0) {
        printf("\nStack Underflow");
        flag = 0;
        updated_element = 0;
    } else {
        flag = 1;
        printf("\nUpdation Possible");
        printf("\nEnter Value to Be Updated: ");
        scanf("%d", &updated_element);
        stack[tos - i + 1] = updated_element;
    }
    return updated_element;
}

void display() {
    int i;
    printf("\nStack Values:");
    for (i = tos; i >= 0; i--) {
        printf("\n\t\t%d", stack[i]);
    }
}
```

**OR**

**Program:**

```
#include<stdio.h>
#include<conio.h>
#define size 100
int tos=-1;
int stack[size];
void push(int);
void display();
void peep();
void pop();
void update();
void main()
{
    clrscr();
    push(10);
    push(11);
    push(12);
    display();
    peep();
    update();
    display();
    pop();
    display();
    getch();
}
void push(int ele)
{
    if(tos>=size)
```

```
                printf("\nStack is full");
        else
                tos++;
                stack[tos]=ele;
}
void display()
{
        int i;
        if(tos<0)
                printf("\nstack is empty");
        else
        {
                for(i=tos;i>=0;i--)
printf("\nElement at position %d is %d",i,stack[i]);
        }
}
void peep()
{
        int pos;
        if(tos<0)
                printf("\nstack is empty");
        else
                printf("\nEnter value of pos:");
                scanf("%d",&pos);
                if(pos<0 || pos>tos+1)
                printf("\nout of range");
                else
printf("\nPeeped element is %d",stack[tos-pos+1]);
}
void pop()
{
        if(tos<0)
                printf("\nStack is empty");
        else
                printf("\nelement deleted");
                tos--;
```

```
}
void update()
{
    int pos;
    if(tos<0)
            printf("\nstack is empty");
    else
    {
            printf("\nEnter  postion:");
            scanf("%d",&pos);
            if(pos<=0 || pos>tos+1)
            printf("\nout of range");
            else
            {
    printf("\nenter new value:");
    scanf("%d",&stack[tos-pos+1]);
    //printf("\nelement=%d",stack[tos-pos+1]);
            }
    }
}
```