DEFINITION OF THE STACK

A stack is an ordered collection of items where new items may be inserted or deleted only at one end, called the top of the stack.

- A stack is a data structure that keeps objects in Last- In-First-Out (LIFO) order,
- Objects are added to the top of the stack,
- Only the top of the stack can be accessed.

BASIC STACK OPERATIONS

- Initialize the Stack.
- Pop an item off the top of the stack (delete an item)
- Push an item onto the top of the stack (insert an item)
- Is the Stack empty?
- Is the Stack full?
- Clear the Stack
- Determine Stack Size

ARRAY IMPLEMENTATION OF THE STACKS

The stacks can be implemented by the use of arrays and linked lists.

- One way to implement the stack is to have a data structure where a variable called top keeps the location of the elements in the stack (array)
- An array is used to store the elements in the stack
The following structure can be used to define the stack data structure:

**Alternative 1:**

```c
struct stack{
    int count; /* keeps the number of elements in the stack */
    int top; /* indicates the location of the top of the stack*/
    int items[STACKSIZE]; /*array to store stack elements*/
};
```

**Alternative 2:**

```c
typedef struct{
    int count;
    int top;
    int items[STACKSIZE]; /*stack can contain up to 10 integers*/
}STACK;
```

A Stack with 5 elements  A full stack  An empty stack

(top=4, count=5)              (top=STACKSIZE-1, count= STACKSIZE)  (top=-1, count=0)

**PUSH OPERATION:**

Before PUSH  After PUSH

(top=4, count=5) (top=5, count= 6)
**POP Operation:**

\[
\begin{array}{c|c|c|c}
\text{Before POP} & \text{After POP} \\
\hline
(\text{top}=4, \text{count}=5) & (\text{top}=3, \text{count}=4) \\
\hline
6 & 5 \\
5 & 4 \\
23 & 15 \\
15 & 3 \\
22 & 2 \\
41 & 1 \\
34 & 0 \\
\end{array}
\]

**Basic Stack Operations**

- **Initialize** the Stack.
  - You can initialize the stack by assigning -1 to the top pointer to indicate that the array based stack is empty (initialized) as follows:

  You can write following lines in the main program:
  ```c
  STACK s;
  s.top = -1;
  ```
  Alternatively you can use the following function:

  ```c
  void StackInitialize(STACK *Sptr)
  {
    Sptr->top = -1;
  }
  ```

- **Push** an item onto the top of the stack (*insert an item*)
  - The function below can be used to push an item into the stack: Note that ps can be some other type than integer.

  ```c
  void push(STACK *Sptr, int ps) /*pushes ps into stack*/
  {
    if(Sptr->top == STACKSIZE-1){
      printf("Stack is full\n");
      return; /*return back to main function*/
    }
    else {
      Sptr->top++;
      Sptr->items[Sptr->top] = ps;
      Sptr->count++;
    }
  }
  ```
**Pop** an item off the top of the stack *(delete an item)*
- The functions below can be used to pop an item from the stack. Note that exit(1) is a library function in `<stdlib.h>` which terminates the program. The stacks are assumed to contain integer elements. Different types can be used.

```
int pop(STACK *Sptr)
{
    int pp;
    if(Sptr->top == -1){
        printf("Stack is empty\n");
        exit(1); /*exit from the function*/
    } else {
        pp = Sptr->items[Sptr->top];
        Sptr->top--;
        Sptr->count--;
    }
    return pp;
}
```

```
void pop(STACK *Sptr, int *pptr)
{
    if(Sptr->top == -1){
        printf("Stack is empty\n");
        return; /*return back*/
    } else {
        *pptr = Sptr->items[Sptr->top];
        Sptr->top--;
        Sptr->count--;
    }
}
```

Ex: The following program implements stack of size 10 using array representation. The 10 elements entered by the user are pushed into the stack and then the elements are popped back and displayed.

```c
#include<stdio.h>
#include<stdlib.h>
define STACKSIZE 10
typedef struct{
    int count;
    int top;
    int items[STACKSIZE]; /*stack can contain up to 10 integers*/
}STACK;

void push(STACK *, int);
int pop(STACK *);

int main()
{
    int p, i;
    STACK s;
    s.top = -1; /*indicates that the stack is empty at the beginning*/
    s.count = 0; /* 0 items are in the stack*/
    /*reading and pushing items into the stack*/
    printf("Enter the %d stack items\n", STACKSIZE);
    for(i=0;i<= STACKSIZE-1;i++){
        scanf("%d", &p);
        push(&s, p);
    }
    /*popping and printing the items in the stack*/
    printf("\n\nStack contains the following items\n");
    for(i=0;i<= STACKSIZE-1;i++){
        p = pop(&s);
        printf("%d\t", p);
    }
    return 0;
}```
void push(STACK *Sptr, int ps) /*pushes ps into stack*/
{
    if(Sptr->top == STACKSIZE-1){
        printf("Stack is full\n");
        printf("There are %d itens in the stack\n", Sptr->count);
        return; /*return back to main function*/
    }
    else {
        Sptr->top++;
        Sptr->items[Sptr->top]= ps;
        Sptr->count++;
    }
}

int pop(STACK *Sptr)
{
    int pp;
    if(Sptr->top == -1){
        printf("Stack is empty\n");
        exit(1); /*exit from the function*/
    }
    else {
        pp = Sptr->items[Sptr->top];
        Sptr->top--;
        Sptr->count--;
    }
    return pp;
}