
# Data Structures and its Applications

**Dinesh Singh**
Department of Computer Science & Engineering

# DATA STRUCTURES AND ITS APPLICATIONS

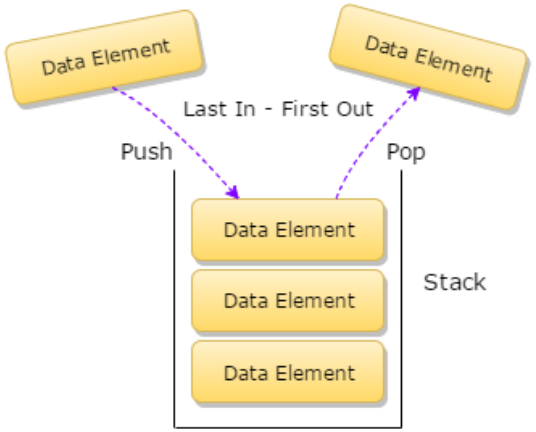
## Stacks

**Dinesh Singh**
Department of Computer Science & Engineering

# Data Structures and its Applications

## Stacks - Definition

- A Stack is a data Structure in which all the insertions and deletions of entries are at one end. This end is called the TOP of the stack.
- When an item is added to a stack it is called push into the stack
- When an item is removed it is called pop from the stack.
- The Last item pushed onto a stack is always the first that will be popped from the stack.
- This property is called the *last in, first out* or LIFO for short



## Stacks – Representation in C

A stack in C is declared as a structure containing two objects :

- An array to hold the elements of the stack
- An Integer to indicate the position of the current stack top within the array
- Stack of integers can be done by the following declaration

```
#define STACKSIZE 100
struct stack
{
  int top;
  int items[STACKSIZE]
};
```

Once this is done, actual stack can be declared by

```
struct stack s;
```

# Data Structures and its Applications

## Stacks – Representation in C


Items need not be restricted to integers, items can be of any type.

A stack can contain items of different types by using C unions.

```
#define STACKSIZE 100
#define INT 1
#define FLOAT 2
#define STRING 3
struct stackelement {
    int etype;
    union{
        int ival;
        float fval;
        char *pavl; //pointer to string
    } element;
};
```

## Stacks – Representation in C

```
struct stack
{
  int top;
  struct stackelement items[STACKSIZE];
};
```

- The above declaration defines a stack whose items can either be integers, floating point numbers or string depending on the value of etype (previous slide).

# Data Structures and its Applications

## Stacks – Implementation of operations of stack

Operations on stack

- Inserting an element on to the stack : push
- Deleting an element from the stack : pop
- Checking the top element : peep
- Checking if the stack is empty : empty
- Checking if the stack is full : overflow

Representation of stack will be as follows

```
#define STACKSIZE 100
struct stack
{
  int top;
  int items[STACKSIZE]
};
```

# Data Structures and its Applications

## Stacks – Implementation of operations of stack

```
void push(struct stack *ps, int x)
/*ps is pointer to the structure representing stack, x is integer to be inserted
top is integer that indicates the position of the current stack top within the
      array, items is an integer array that represents stack, STACK_SIZE is the
      maximum size of the stack */
{
   if (ps->top == STACKSIZE -1) //check if the stack is full
         printf("STACK FULL Cannot insert..");
  else
   {
      ++(ps->top); //increment top
      ps->items[ps->top]=x; //insert the element at a location top
    }
}
```

# Data Structures and its Applications

## Stacks – Implementation of operations of stack

```
int pop(struct stack *ps )
/*ps is pointer to the structure representing stack, top is integer that indicates
      the position of the current stack top within the array , items is an integer
      array that represents stack, STACK_SIZE is the maximum size of the stack */
  {
   if (ps->top == -1) // check if the stack is the empty
          printf("STACK EMPTY Cannot DELETE..");
  else
   {
      x=ps->items[ps->top]; //delete the element
      --(ps->top); //decrement top
     return x;
    }
}
```

# Data Structures and its Applications

## Stacks – Implementation of operations of stack

```
int display(struct stack *ps )
/*ps is pointer to the structure representing stack, top is integer that indicates
      the position of the current stack top within the array , items is an integer
      array that represents stack, STACK_SIZE is the maximum size of the stack */
  {
   if (ps->top == -1) // check if the stack is the empty
          printf("STACK EMPTY ");
  else
   {
      for (i=ps->top;i>=0;i++) // displays the elements from top
         printf("%d",ps->items[i]);
 }
}
```

## Stacks – Implementation of operations of stack

```
int peep(struct stack *ps )
{
   if (ps->top == -1)
          printf("STACK EMPTY ..");
  else
   {
      x=ps->items[ps->top]; //get the element
      return x;
    }
}
```

## Stacks – Implementation of operations of stack

```
int empty(struct stack *ps )
{
    if (ps->top == -1)
            return 1;
   return 0;
}
int overflow(struct stack *ps)
{
    if (ps->top==STACKSIZE-1)
      return 1;
  return 0;
}
```

## Stacks – Implementation of operations of stack


**implementation of stack operations where the items array and top are separate variables (Not part of structure)**

```
void push(int *s, int *top, int x)
{
  if(*top==STACKSIZE-1)//check if the stack is empty
  {
    printf("stack overflow..cannot insert");
   return 0;
 }
 else
  {
    ++*top; //increment the top
    s[*top]=x; //insert the element
  }
 return 1;
}
```

## Stacks – Implementation of operations of stack

- Implementation of stack operations where the items and the top are separate variables (Not part of structure)

```
int pop(int *s, int *top)
{
  if(*top==-1)//check if the stack is empty
  {
    printf("Stack empty .. Cannot delete");
   return -1;
 }
 else
  {
    x=s[*top]; //insert the element
    --*top; //decrement top
   return x; // return the deleted element
  }
}
```

## Stacks – Implementation of operations of stack

- Implementation of stack operations where the items and the top are separate variables (Not part of structure)

```
display pop(int *s, int *top)
{
   if(*top==-1)
    printf("Empty stack");
 else
  {
   for(i=*top;i>=0;i--) // display the elements from the top
     printf("%d",s[i]);
  }
}
```

## Stacks – Application of Stack

- Write an algorithm to determine if an input character string is of the form x C y where x is a string consisting of the letters ‘A’ and ‘B’ and where y is the reverse of x. At each point you may read only the character of the string

```
 int check(t)
//the function returns 1 if string t is of the form x C y, else returns 0
//uses stack s and its operations push and pop
 {
    i=0;
    while(t[i]!=‘C’) //push all the characters of the string into the stack
until C  is encountered
    {
      push(&s, t[i]);
      i=i+1;
    }
```

```
      //pop the contents of the stack and compare with t[i] until the end of  the string
while(t[i]!='\0')
{
   x=pop(&s);
   if(t[i]!=x) //if the character popped out is not equal to the character read from the string
      return 0; // not of the form xCy
   i=i+1;
 }
 return 1; // string of the form
 }
```


# THANK YOU

**Dinesh Singh**
Department of Computer Science & Engineering

**dineshs@pes.edu**

+91 8088654402