Stack Data Structure

What is a Stack?

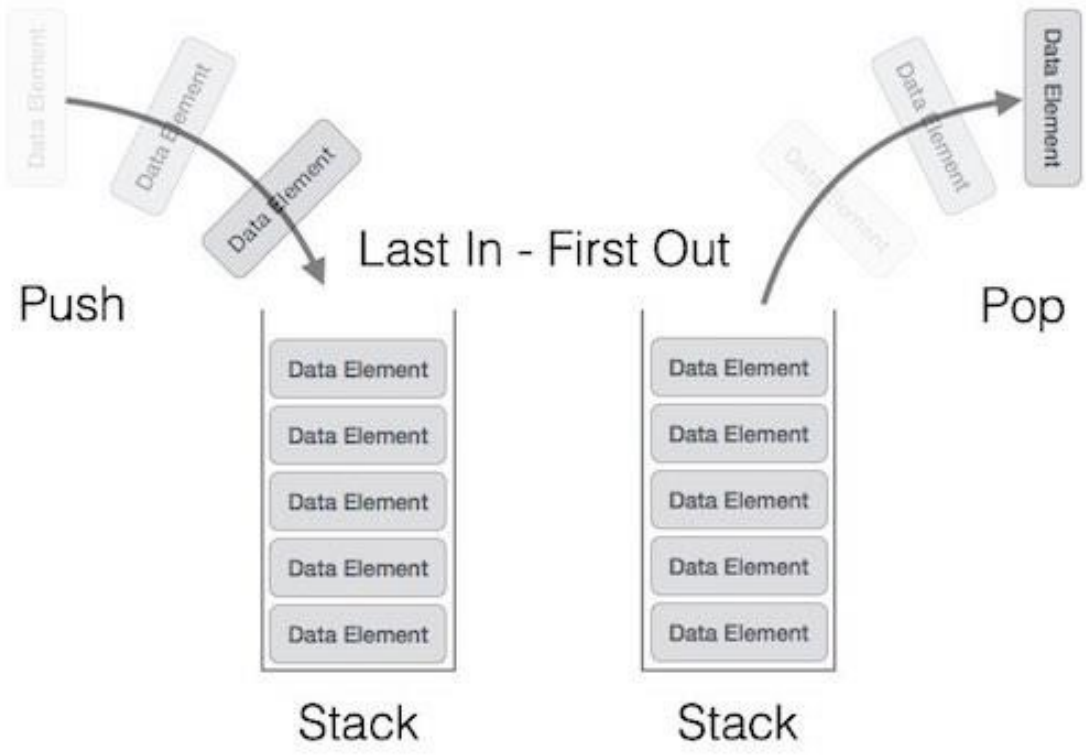
A stack is a **linear data structure** where elements are stored in the LIFO (Last In First Out) principle where the last element inserted would be the first element to be deleted. A stack is an Abstract Data Type (ADT), that is popularly used in most programming languages. It is named stack because it has the similar operations as the real-world stacks, for example – a pack of cards or a pile of plates, etc.

Stack is considered a complex data structure because it uses other data structures for implementation, such as Arrays, Linked lists, etc.

Stack Representation

A stack allows all data operations at one end only. At any given time, we can only access the top element of a stack.

The following diagram depicts a stack and its operations –



A stack can be implemented by means of Array, Structure, Pointer, and Linked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays, which makes it a fixed size stack implementation.

Basic Operations on Stacks

Stack operations are usually performed for initialization, usage and, de-initialization of the stack ADT.

The most fundamental operations in the stack ADT include: push(), pop(), peek(), isFull(), isEmpty(). These are all built-in operations to carry out data manipulation and to check the status of the stack.

Stack uses pointers that always point to the topmost element within the stack, hence called as the **top** pointer.

Stack Insertion: push()

The push() is an operation that inserts elements into the stack. The following is an algorithm that describes the push() operation in a simpler way.

Algorithm

```
1. Checks if the stack is full.
2. If the stack is full, produces an error and exit.
3. If the stack is not full, increments top to point next
    empty space.
4. Adds data element to the stack location, where top
    is pointing.
5. Returns success.
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java | Python

```c
#include <stdio.h>
int MAXSIZE = 8;
int stack[8];
int top = -1;

/* Check if the stack is full*/
int isfull(){
   if(top == MAXSIZE)
      return 1;
   else
      return 0;
}

/* Function to insert into the stack */
int push(int data){
   if(!isfull()) {
      top = top + 1;
      stack[top] = data;
   } else {
      printf("Could not insert data, Stack is full.\n");
   }
}
```

```
/* Main function */
int main(){
   int i;
   push(44);
   push(10);
   push(62);
   push(123);
   push(15);
   printf("Stack Elements: \n");

   // print stack data
   for(i = 0; i < 8; i++) {
      printf("%d ", stack[i]);
   }
   return 0;
}
```

Output

```
Stack Elements:
44 10 62 123 15 0 0 0
```

Note – In Java we have used to built-in method **push()** to perform this operation.

Stack Deletion: pop()

The pop() is a data manipulation operation which removes elements from the stack. The following pseudo code describes the pop() operation in a simpler way.

Algorithm

```
1. Checks if the stack is empty.
2. If the stack is empty, produces an error and exit.
3. If the stack is not empty, accesses the data element at
```

```
   which top is pointing.
4. Decreases the value of top by 1.
5. Returns success.
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java | Python

```c
#include <stdio.h>
int MAXSIZE = 8;
int stack[8];
int top = -1;

/* Check if the stack is empty */
int isempty(){
   if(top == -1)
      return 1;
   else
      return 0;
}

/* Check if the stack is full*/
int isfull(){
   if(top == MAXSIZE)
      return 1;
   else
      return 0;
}

/* Function to delete from the stack */
int pop(){
   int data;
   if(!isempty()) {
      data = stack[top];
      top = top - 1;
```

```
      return data;
   } else {
      printf("Could not retrieve data, Stack is empty.\n");
   }
}

/* Function to insert into the stack */
int push(int data){
   if(!isfull()) {
      top = top + 1;
      stack[top] = data;
   } else {
      printf("Could not insert data, Stack is full.\n");
   }
}

/* Main function */
int main(){
   int i;
   push(44);
   push(10);
   push(62);
   push(123);
   push(15);
   printf("Stack Elements: \n");

   // print stack data
   for(i = 0; i < 8; i++) {
      printf("%d ", stack[i]);
   }
   /*printf("Element at top of the stack: %d\n" ,peek());*/
   printf("\nElements popped: \n");

   // print stack data
   while(!isempty()) {
      int data = pop();
      printf("%d ",data);
   }
```

```
   return 0;
}
```

Output

```
Stack Elements:
44 10 62 123 15 0 0 0
Elements popped:
15 123 62 10 44
```

Note – In Java we are using the built-in method pop().

Retrieving topmost Element from Stack: peek()

The peek() is an operation retrieves the topmost element within the stack, without deleting it. This operation is used to check the status of the stack with the help of the top pointer.

Algorithm

```
1. START
2. return the element at the top of the stack
3. END
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java | Python

```
#include <stdio.h>
int MAXSIZE = 8;
int stack[8];
int top = -1;
```

```
/* Check if the stack is full */
int isfull(){
   if(top == MAXSIZE)
      return 1;
   else
      return 0;
}

/* Function to return the topmost element in the stack */
int peek(){
   return stack[top];
}

/* Function to insert into the stack */
int push(int data){
   if(!isfull()) {
      top = top + 1;
      stack[top] = data;
   } else {
      printf("Could not insert data, Stack is full.\n");
   }
}

/* Main function */
int main(){
   int i;
   push(44);
   push(10);
   push(62);
   push(123);
   push(15);
   printf("Stack Elements: \n");

   // print stack data
   for(i = 0; i < 8; i++) {
      printf("%d ", stack[i]);
   }
   printf("\nElement at top of the stack: %d\n" ,peek());
```

```
   return 0;
}
```

Output

```
Stack Elements:
44 10 62 123 15 0 0 0
Element at top of the stack: 15
```

Verifying whether the Stack is full: isFull()

The isFull() operation checks whether the stack is full. This operation is used to check the status of the stack with the help of top pointer.

Algorithm

```
1. START
2. If the size of the stack is equal to the top position of the stac
   the stack is full. Return 1.
3. Otherwise, return 0.
4. END
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java | Python

```
#include <stdio.h>
int MAXSIZE = 8;
int stack[8];
int top = -1;

/* Check if the stack is full */
```

```
int isfull(){
   if(top == MAXSIZE)
      return 1;
   else
      return 0;
}

/* Main function */
int main(){
   printf("Stack full: %s\n" , isfull()?"true":"false");
   return 0;
}
```

Output

```
Stack full: false
```

Verifying whether the Stack is empty: isEmpty()

The isEmpty() operation verifies whether the stack is empty. This operation is used to check the status of the stack with the help of top pointer.

Algorithm

```
1. START
2. If the top value is -1, the stack is empty. Return 1.
3. Otherwise, return 0.
4. END
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java | Python

```c
#include <stdio.h>
int MAXSIZE = 8;
int stack[8];
int top = -1;

/* Check if the stack is empty */
int isempty() {
   if(top == -1)
      return 1;
   else
      return 0;
}

/* Main function */
int main() {
   printf("Stack empty: %s\n" , isempty()?"true":"false");
   return 0;
}
```

Output

```
Stack empty: true
```

Stack Complete implementation

Following are the complete implementations of Stack in various programming languages –

C | C++ | Java | Python

```c
#include <stdio.h>
int MAXSIZE = 8;
int stack[8];
int top = -1;
/* Check if the stack is empty */
int isempty(){
```

```
   if(top == -1)
      return 1;
   else
      return 0;
}
/* Check if the stack is full */
int isfull(){
   if(top == MAXSIZE)
      return 1;
   else
      return 0;
}

/* Function to return the topmost element in the stack */
int peek(){
   return stack[top];
}

/* Function to delete from the stack */
int pop(){
   int data;
   if(!isempty()) {
      data = stack[top];
      top = top - 1;
      return data;
   } else {
      printf("Could not retrieve data, Stack is empty.\n");
   }
}

/* Function to insert into the stack */
int push(int data){
   if(!isfull()) {
      top = top + 1;
      stack[top] = data;
   } else {
      printf("Could not insert data, Stack is full.\n");
   }
}
```

```
/* Main function */
int main(){
   push(44);
   push(10);
   push(62);
   push(123);
   push(15);
   printf("Element at top of the stack: %d\n" ,peek());
   printf("Elements: \n");
   // print stack data
   while(!isempty()) {
      int data = pop();
      printf("%d\n",data);
   }
   printf("Stack full: %s\n" , isfull()?"true":"false");
   printf("Stack empty: %s\n" , isempty()?"true":"false");
   return 0;
}
```

Output

```
Element at top of the stack: 15
Elements:
15123
62
10
44
Stack full: false
Stack empty: true
```

Stack Implementation in C

Click to check the implementation of Stack Program using C