**Stack Algorith for insertion:**

**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.**

public class Demo{

final static int MAXSIZE = 8;

static int stack[] = new int[MAXSIZE];

static int top = -1;

public static int isfull(){

if(top == MAXSIZE)

return 1;

else

return 0;

}

public static int push(int data){

if(isfull() != 1) {

top = top + 1;

stack[top] = data;

} else {

System.out.print("Could not insert data, Stack is full.\n");

}

return data;

}

public static void main(String[] args){

int i;

push(44);

push(10);

push(62);

push(123);

push(15);

System.out.print("\nStack Elements: ");

// print stack data

for(i = 0; i < MAXSIZE; i++) {

System.out.print(stack[i] + " ");

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**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.**

public class Demo{

final static int MAXSIZE = 8;

public static int stack[] = new int[MAXSIZE];

public static int top = -1;

/\* Check if the stack is empty \*/

public static int isempty(){

if(top == -1)

return 1;

else

return 0;

}

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

public static int isfull(){

if(top == MAXSIZE)

return 1;

else

return 0;

}

/\* Function to delete from the stack \*/

public static int pop(){

int data = 0;

if(isempty() != 1) {

data = stack[top];

top = top - 1;

return data;

} else {

System.out.print("Could not retrieve data, Stack is empty.");

}

return data;

}

/\* Function to insert into the stack \*/

public static int push(int data){

if(isfull() != 1) {

top = top + 1;

stack[top] = data;

} else {

System.out.print("\nCould not insert data, Stack is full.\n");

}

return data;

}

/\* Main function \*/

public static void main(String[] args){

push(44);

push(10);

push(62);

push(123);

push(15);

System.out.print("Stack Elements: ");

// print stack data

for(int i = 0; i < MAXSIZE; i++) {

System.out.print(stack[i] + " ");

}

System.out.print("\nElements popped: ");

// print stack data

while(isempty() != 1) {

int data = pop();

System.out.print(data + " ");

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Retrieving topmost Element: 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**

public class Demo{

final static int MAXSIZE = 8;

public static int stack[] = new int[MAXSIZE];

public static int top = -1;

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

public static int isfull(){

if(top == MAXSIZE)

return 1;

else

return 0;

}

/\* Function to return the topmost element in the stack \*/

public static int peek(){

return stack[top];

}

/\* Function to insert into the stack \*/

public static int push(int data){

if(isfull() != 1) {

top = top + 1;

stack[top] = data;

} else {

System.out.print("Could not insert data, Stack is full.");

}

return data;

}

/\* Main function \*/

public static void main(String[] args){

push(44);

push(10);

push(62);

push(123);

push(15);

System.out.print("Stack Elements: ");

// print stack data

for(int i = 0; i < MAXSIZE; i++) {

System.out.print(stack[i] + " ");

}

System.out.print("\nElement at top of the stack: " + peek());

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Verifying whether the Stack is full: isFull()**

**Algorithm:**

**1. START**

**2. If the size of the stack is equal to the top position of the stack,**

**the stack is full. Return 1.**

**3. Otherwise, return 0.**

**4. END**

import java.io.\*;

public class StackExample {

private int arr[];

private int top;

private int capacity;

StackExample(int size) {

arr = new int[size];

capacity = size;

top = -1;

}

public boolean isEmpty() {

return top == -1;

}

public boolean isFull() {

return top == capacity - 1;

}

public void push(int key) {

if (isFull()) {

System.out.println("Stack is Full\n");

return;

}

arr[++top] = key;

}

public static void main (String[] args) {

StackExample stk = new StackExample(5);

stk.push(1); // inserting 1 in the stack

stk.push(2);

stk.push(3);

stk.push(4);

stk.push(5);

System.out.println("Stack full: " + stk.isFull());

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Verifying whether the Stack is empty: isEmpty()**

**Algorithm:**

**1. START**

**2. If the top value is -1, the stack is empty. Return 1.**

**3. Otherwise, return 0.**

**4. END**

public class Demo{

final static int MAXSIZE = 8;

static int stack[] = new int[MAXSIZE];

static int top = -1;

/\* Check if the stack is empty \*/

public static int isempty(){

if(top == -1)

return 1;

else

return 0;

}

/\* Main function \*/

public static void main(String[] args){

boolean res = isempty() == 1 ? true : false;

System.out.print("Stack empty: " + res);

}

}