Aim: Implement Stack ADT using an array.

Theory:

What is Stack?

-To implement a stack using array theory, we utilize a one-dimensional array to store elements while adhering to the Last In First Out (LIFO) principle. This approach is straightforward and efficient for performing stack operations such as push, pop, and peek.

Procedure:

1. **Initialize an array** to represent the stack.
2. Use the **end of the array** to represent the **top of the stack**.
3. Implement **push** (add to end), **pop** (remove from the end), and **peek** (check end) operations, ensuring to handle empty and full stack conditions.

Algorithm for Each Operation

1.Push Operation

Check if the stack is full (i.e., top == SIZE - 1).

If full, print "Stack Overflow".

If not, increment top and assign the new value to stack[top].

2.Pop Operation

Check if the stack is empty (i.e., top == -1).

If empty, print "Stack Underflow".

If not, store stack[top] in a temporary variable, decrement top, and return the stored value.

3.Peek Operation

Check if the stack is empty.

If empty, print "Stack Underflow".

If not, return stack[top].

4.isEmpty Operation

Return true if top == -1, otherwise return false.

5.isFull Operation

Return true if top == SIZE - 1, otherwise return false.

```
#include <limits.h>
 #include <stdio.h>
 #include <stdlib.h>
struct Stack {
 int top;
 unsigned capacity;
 int* array;
 };
struct Stack* createStack(unsigned capacity)
 {
 struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
 stack->capacity = capacity;
 stack->top =-1;
 stack->array = (int*)malloc(stack->capacity * sizeof(int));
 return stack;
 }
 int isFull(struct Stack* stack)
 {
 return stack->top == stack->capacity-1;
 }
int isEmpty(struct Stack* stack)
 {
 return stack->top ==-1;
 }
void push(struct Stack* stack, int item)
 {
```

```
 if (isFull(stack))
 return;
 stack->array[++stack->top] = item;
 printf("%d pushed to stack\n", item);
 }
int pop(struct Stack* stack)
{
 if (isEmpty(stack))
 return INT_MIN;
 return stack->array[stack->top--];
 }
 int peek(struct Stack* stack)
 {
 if (isEmpty(stack))
 return INT_MIN;
 return stack->array[stack->top];
 }
 int main()
{
 struct Stack* stack = createStack(100);
 push(stack, 10);
 push(stack, 20);
 push(stack, 30);
 printf("%d popped from stack\n", pop(stack));
 return 0;
 }
```

Output:

```
hp@VICTUS ~/ds
$ nano stackarray.c

hp@VICTUS ~/ds
$ gcc stackarray.c -o stkary

hp@VICTUS ~/ds
$ ./stkary
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack
```

Conclusion: This experiment demonstrates the implementation of a Stack ADT using an array as the underlying data structure. It allows you to understand how stacks work, the fundamental operations (push, pop, peek), and the use of arrays for efficient storage.