C program for array implementation of stack

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>

// A structure to represent a stack
struct Stack {
    int top;
    unsigned capacity;
    int* array;
};

// function to create a stack of given capacity. It initializes size of
// stack as 0
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;
}

// Stack is full when top is equal to the last index
int isFull(struct Stack* stack)
{
    return stack->top == stack->capacity - 1;
}

// Stack is empty when top is equal to -1
```

```
int isEmpty(struct Stack* stack)
{
   return stack->top == -1;
}

// Function to add an item to stack. It increases top by 1
void push(struct Stack* stack, int item)
{
   if (isFull(stack))
      return;
   stack->array[++stack->top] = item;
   printf("%d pushed to stack\n", item);
}

// Function to remove an item from stack. It decreases top by 1
int pop(struct Stack* stack)
{
   if (isEmpty(stack))
      return INT_MIN;
   return stack->array[stack->top--];
}

// Function to return the top from stack without removing it
int peek(struct Stack* stack)
{
   if (isEmpty(stack))
      return INT_MIN;
   return stack->array[stack->top];
}

// Driver program to test above functions
```

```
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:

10 pushed to stack

20 pushed to stack

30 pushed to stack

30 popped from stack

// C program to implement a stack using linked list

```
#include <stdio.h>
#include <stdlib.h>

//  _________LINKED LIST UTILITY FUNCITON_____________

// Define the structure for a node of the linked list
typedef struct Node {
   int data;
   struct Node* next;
} node;
```

```
// linked list utility function
node* createNode(int data)
{
    // allocating memory
    node* newNode = (node*)malloc(sizeof(node));

    // if memory allocation is failed
    if (newNode == NULL)
        return NULL;

    // putting data in the node
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// fuction to insert data before the head node
int insertBeforeHead(node** head, int data)
{
    // creating new node
    node* newNode = createNode(data);
    // if malloc fail, return error code
    if (!newNode)
        return -1;

    // if the linked list is empty
    if (*head == NULL) {
        *head = newNode;
        return 0;
    }
```

```
    newNode->next = *head;
    *head = newNode;
    return 0;
}

// deleting head node
int deleteHead(node** head)
{
    // no need to check for empty stack as it is already
    // being checked in the caller function
    node* temp = *head;
    *head = (*head)->next;
    free(temp);
    return 0;
}

// _________STACK IMPLEMENTATION STARTS HERE_________

// Function to check if the stack is empty or not
int isEmpty(node** stack) { return *stack == NULL; }

// Function to push elements to the stack
void push(node** stack, int data)
{
    // inserting the data at the beginning of the linked
    // list stack
    // if the insertion function returns the non - zero
    // value, it is the case of stack overflow
    if (insertBeforeHead(stack, data)) {
        printf("Stack Overflow!\n");
    }
```

```
}

// Function to pop an element from  the stack
int pop(node** stack)
{
    // checking underflow condition
    if (isEmpty(stack)) {
        printf("Stack Underflow\n");
        return -1;
    }

    // deleting the head.
    deleteHead(stack);
}

// Function to return the topmost element of the stack
int peek(node** stack)
{
    // check for empty stack
    if (!isEmpty(stack))
        return (*stack)->data;
    else
        return -1;
}

// Function to print the Stack
void printStack(node** stack)
{
    node* temp = *stack;
    while (temp != NULL) {
        printf("%d-> ", temp->data);
```

```
        temp = temp->next;
    }
    printf("\n");
}

// driver code
int main()
{
    // Initialize a new stack top pointer
    node* stack = NULL;

    // Push elements into the stack
    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);
    push(&stack, 40);
    push(&stack, 50);

    // Print the stack
    printf("Stack: ");
    printStack(&stack);
    // Pop elements from the stack
    pop(&stack);
    pop(&stack);
    // Print the stack after deletion of elements
    printf("\nStack: ");
    printStack(&stack);

    return 0;
}
```

output:

Stack: 50-> 40-> 30-> 20-> 10->

Stack: 30-> 20-> 10->