STACK IMPLEMENTATION USING ARRAY

```
#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

=== Code Execution Successful ===

STACK IMPLEMENTATION USING LINKED LIST

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int info;
   struct node *ptr;
}*top,*top1,*temp;

int count = 0;
// Push() operation on a  stack
void push(int data) {
    if (top == NULL)
   {
      top =(struct node *)malloc(1*sizeof(struct node));
      top->ptr = NULL;
      top->info = data;
   }
   else
   {
      temp =(struct node *)malloc(1*sizeof(struct node));
      temp->ptr = top;
      temp->info = data;
      top = temp;
   }
   count++;
   printf("Node is Inserted\n\n");
}
```

```
int pop() {
     top1 = top;


    if (top1 == NULL)
    {
        printf("\nStack Underflow\n");
        return -1;
    }
    else
        top1 = top1->ptr;
    int popped = top->info;
    free(top);
    top = top1;
    count--;
    return popped;
}


void display() {
    // Display the elements of the stack
    top1 = top;


    if (top1 == NULL)
    {
        printf("\nStack Underflow\n");
        return;
    }
```

```
    printf("The stack is \n");
    while (top1 != NULL)
    {
        printf("%d--->", top1->info);
        top1 = top1->ptr;
    }
    printf("NULL\n\n");

}

int main() {
    int choice, value;
    printf("\nImplementation of Stack using Linked List\n");
    while (1) {
        printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice) {
        case 1:
            printf("\nEnter the value to insert: ");
            scanf("%d", &value);
            push(value);
            break;
        case 2:
            printf("Popped element is :%d\n", pop());
            break;
```

```
        case 3:
            display();
            break;
        case 4:
            exit(0);
            break;
        default:
            printf("\nWrong Choice\n");
        }
    }
}
```

OUTPUT

Implementation of Stack using Linked List

1. Push

2. Pop

3. Display

4. Exit

Enter your choice : 1

Enter the value to insert: 2 3 5

Node is Inserted