C code for array implementation of stack with proper explanation

```
#include <stdio.h>
#include <stdlib.h>  // For using malloc

#define MAX 5  // Define the maximum size of the stack

// Stack structure
struct Stack {
   int items[MAX];
   int top;
};

// Function to create an empty stack
struct Stack* createStack() {
   struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
   stack->top = -1;  // Stack is empty initially
   return stack;
}

// Function to check if the stack is full
int isFull(struct Stack* stack) {
   return stack->top == MAX - 1;
```

```
}

// Function to check if the stack is empty
int isEmpty(struct Stack* stack) {
    return stack->top == -1;
}

// Function to add an element to the stack (push operation)
void push(struct Stack* stack, int value) {
    if (isFull(stack)) {
        printf("Stack is full! Cannot push %d\n", value);
        return;
    }
    stack->items[++(stack->top)] = value;
    printf("Pushed %d onto the stack\n", value);
}

// Function to remove an element from the stack (pop operation)
int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty! Cannot pop\n");
        return -1;
```

```
    }
    return stack->items[(stack->top)--];
}

// Function to peek at the top element of the stack without removing it
int peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty! Nothing to peek\n");
        return -1;
    }
    return stack->items[stack->top];
}

// Function to display the elements in the stack
void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
        return;
    }
    printf("Stack elements are: ");
    for (int i = 0; i <= stack->top; i++) {
        printf("%d ", stack->items[i]);
```

```
    }
    printf("\n");
}

int main() {
    struct Stack* stack = createStack();  // Create a stack

    push(stack, 10);  // Push elements onto the stack
    push(stack, 20);
    push(stack, 30);
    push(stack, 40);
    push(stack, 50);  // This will fill the stack

    display(stack);  // Display the stack contents

    printf("Peek top element: %d\n", peek(stack));  // Peek the top element

    printf("Popped element: %d\n", pop(stack));  // Pop an element
    display(stack);  // Display the stack again

    return 0;
}
```

Explanation:

1. Stack Structure

```
struct Stack {
    int items[MAX];
    int top;
};
```

- `items[MAX]` : An array to hold the stack elements (fixed size defined by `MAX`).
- `top` : An integer to track the index of the top element in the stack. It is initialized to `-1` to indicate that the stack is empty.

2. Creating the Stack

```
struct Stack* createStack() {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->top = -1;
    return stack;
}
```

- This function dynamically allocates memory for the stack and sets the `top` to `-1` (meaning the stack is initially empty).

3. isFull and isEmpty Functions

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

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

- `isFull` : Returns true if `top` is equal to `MAX - 1` , meaning the stack is full.
- `isEmpty` : Returns true if `top` is `-1` , indicating the stack is empty.

4. Push Operation

```c
void push(struct Stack* stack, int value) {
    if (isFull(stack)) {
        printf("Stack is full! Cannot push %d\n", value);
        return;
    }
    stack->items[++(stack->top)] = value;
    printf("Pushed %d onto the stack\n", value);
}
```

- Before adding an element, it checks if the stack is full. If not, it increments the `top` and then assigns the value to `items[top]`.

5. **Pop Operation**

```c
int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty! Cannot pop\n");
        return -1;
    }
    return stack->items[(stack->top)--];
}
```

- Before removing an element, it checks if the stack is empty. If not, it returns the top element and decrements `top`.

6. **Peek Operation**

```c
int peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty! Nothing to peek\n");
        return -1;
    }
    return stack->items[stack->top];
}
```

- Returns the top element without removing it.

7. **Display Function**

```c
void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
        return;
    }
    printf("Stack elements are: ");
    for (int i = 0; i <= stack->top; i++) {
        printf("%d ", stack->items[i]);
    }
    printf("\n");
}
```

- Iterates through the stack from bottom to top and prints each element.

8. **Main Function**

- Demonstrates the usage of the stack by pushing, popping, and displaying elements.

This is a basic implementation of a stack using an array. The stack has a fixed size (`MAX`), and once it's full, no more elements can be pushed until some are popped.