Experiment-4

Aim: Write a program to implement character stack using an array
Push
Pop functions using boundry condition
Also write paranthesis correctness in a string array.

Code:

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

#define MAX 100  // Define maximum size of stack

// Stack structure
struct Stack {
   char arr[MAX];  // Stack array to store characters
   int top;        // Stack top to keep track of the top index
};

// Function to initialize the stack
void initStack(struct Stack* stack) {
   stack->top = -1;
}

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

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

// Function to push a character onto the stack
void push(struct Stack* stack, char ch) {
   if (isFull(stack)) {
      printf("Stack overflow! Cannot push %c\n", ch);
```

```
        return;
    }
    stack->arr[++stack->top] = ch;
}

// Function to pop a character from the stack
char pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow! Cannot pop\n");
        return '\0';  // Return null character if stack is empty
    }
    return stack->arr[stack->top--];
}

// Function to check for matching parentheses
bool isMatchingPair(char left, char right) {
    if (left == '(' && right == ')') return true;
    if (left == '{' && right == '}') return true;
    if (left == '[' && right == ']') return true;
    return false;
}

// Function to check if parentheses in a string are balanced
bool checkParentheses(char str[]) {
    struct Stack stack;
    initStack(&stack);

    for (int i = 0; str[i] != '\0'; i++) {
        char ch = str[i];

        // If opening bracket, push it to stack
        if (ch == '(' || ch == '{' || ch == '[') {
            push(&stack, ch);
        }
        // If closing bracket, check for matching opening bracket
        else if (ch == ')' || ch == '}' || ch == ']') {
            if (isEmpty(&stack) || !isMatchingPair(pop(&stack), ch)) {
                return false;  // Unmatched parentheses
            }
        }
```

```
    }

    // If stack is empty, all parentheses are matched
    return isEmpty(&stack);
}

// Main function
int main() {
     struct Stack stack;
     initStack(&stack);
     for(int i=65;i<=69;i++){
        push(&stack,(char)i);
     }
     while(!isEmpty(&stack)){
        printf("%c , ",pop(&stack));
     }
    printf("\n");
    char str[MAX];
    printf("Enter a string with parentheses: ");
    scanf("%s", str);

    if (checkParentheses(str)) {
        printf("Parentheses are balanced.\n");
    } else {
        printf("Parentheses are not balanced.\n");
    }

    return 0;
}
```

Output:

```
PS C:\aditya\Programming_Languages\DTU\SE_203_DS_lab\Git_> .\exp4
E , D , C , B , A ,
Enter a string with parentheses: (){}[])
Parentheses are balanced.
PS C:\aditya\Programming_Languages\DTU\SE_203_DS_lab\Git_> .\exp4
E , D , C , B , A ,
Enter a string with parentheses: [)
Parentheses are not balanced.
PS C:\aditya\Programming_Languages\DTU\SE_203_DS_lab\Git_>
```