```cpp
/* DSL - EXPERIMENT 9 - D26 */

#include <iostream>
using namespace std;

#define MAX 25


// Class Stack
class Stack {
    private:
        char *data;                         // Dynamic allocation or else 'char data[MAX]'
        int top;

    public:
        Stack() {                           // Constructor initialization
            data = new char[MAX];           // Allocate 'data' with 'new char[MAX]'
            top = -1;
        }
        bool isEmpty();                     // Prototypes
        bool isFull();
        bool push(char x);
        char pop();
        char peek();
        void display();
};


// Class Parenthesis
class Parenthesis {
    char expn[MAX];
    Stack stack;

    public:
        void read();
        void checkexpn();
};


// Stack class function defintions
bool Stack::isEmpty() {
    return (top == -1);
}


bool Stack::isFull() {
    return (top == (MAX - 1));
}
```

```
bool Stack::push(char x) {
    if (isFull()) {
        cout << "Stack Overflow\n";
        return false;
    }
    else {
        data[++top] = x;
        cout << x << " : Pushed into stack\n";
        return true;
    }
}


char Stack::pop() {
    char x;
    if (isEmpty()) {
        cout << "Stack Underflow\n";
        return 0;
    }
    else{
        x = data[top--];
        cout << x << " : Popped from stack\n";
        return x;
    }
}


char Stack::peek() {
    if (isEmpty()) {
        cout << "Stack is Empty\n";
        return 0;
    }
    else
        return data[top];
}

void Stack::display() {
    cout << "\nCurrent stack is: ";
    for (int i = 0; i <= top; i++) {
        cout << data[i] << " ⟶ ";
    }
    cout << "NULL";
}

// Parenthesis class function defintions
void Parenthesis::read() {
    cout << "\nEnter the expression: "; cin >> expn;
    cout << "\n";
}
```

```
void Parenthesis::checkexpn() {
    int i,flag = 0;
    char ch;

    for (i = 0; expn[i] ≠ '\0'; i++) {
        if (expn[i] == '{' || expn[i] == '[' || expn[i] == '(')
            stack.push(expn[i]);

        if(expn[i] == '}' || expn[i] == ']' || expn[i] == ')') {
            if (!stack.isEmpty()) {
                ch = stack.pop();

                if (expn[i] == '}' && ch ≠ '{') {
                    flag = 1;
                    break;
                }

                if (expn[i] == ']' && ch ≠ '[') {
                    flag = 1;
                    break;
                }

                if (expn[i] == ')' && ch ≠ '(') {
                    flag = 1;
                    break;
                }
            }
        }
    }

    stack.display();

    if (flag == 0 && stack.isEmpty())
        cout<<"\n\nExpression is in well Paranthesis";
    else
        cout<<"\n\nExpression is not in well Paranthesis";
}


int main() {
    Parenthesis expr;
    expr.read();
    expr.checkexpn();
    return 0;
}
```

```
/*

———————————————— OUTPUT ————————————————

Enter the expression: [(a+b)/{c*(d+e)}]

[ : Pushed into stack
( : Pushed into stack
( : Popped from stack
{ : Pushed into stack
( : Pushed into stack
( : Popped from stack
{ : Popped from stack
[ : Popped from stack

Current stack is: NULL

Expression is in well Paranthesis


Enter the expression: [(a+b)/{c*(d+e

[ : Pushed into stack
( : Pushed into stack
( : Popped from stack
{ : Pushed into stack
( : Pushed into stack

Current stack is: [ ⟶ { ⟶ ( ⟶ NULL

Expression is not in well Paranthesis

*/
```