In [1]:

```
from dsLecture3 import ArrayStack
```

```
Algorithm ParenMatch(X,n):

Input: An array X of n tokens, each of which is either a grouping symbol, a variable, an
arithmetic operator, or a number

Output: true if and only if all the grouping symbols in X match

=======================================
Let S be an empty stack
for i=0 to n-1 do
    if X[i] is an opening grouping symbol then
        S.push(X[i])
    else if X[i] is a closing grouping symbol then
        if S.is_empty() then
            return false {nothing to match with}
        if S.pop() does not match the type of X[i] then
            return false {wrong type}
if S.is_empty() then
    return true {every symbol matched}
else return false {some symbols were never matched}
```

In [2]:

```
def ParenMatch(X):
    lefty = '({['
    righty = ')}]'
    S = ArrayStack()
    for symbol in X:
        if symbol in lefty:
            S.push(symbol)
        elif symbol in righty:
            if S.is_empty():
                return False
            if righty.index(symbol) != lefty.index(S.pop()):
                return False
    return S.is_empty()
```

In [3]:

```
print(ParenMatch(list('()(()){([()])}')))
print(ParenMatch(list('((()(()){([()])}))')))
print(ParenMatch(list(')(()){([()])}')))
print(ParenMatch(list('({[])}')))
print(ParenMatch(list('(')))
```

```
True
True
False
False
False
```

In [4]:

```
def ParenMatch_explained(X):
    print(f'Processing {X}')
    lefty = '({['
    righty = ')}]'
    S = ArrayStack()
    index = 0
    for symbol in X:
        print(f'#{index} - Processing {symbol}')
        index += 1

        if symbol in lefty:
            S.push(symbol)
            print(f'Push {symbol} into the stack')
        elif symbol in righty:
            if S.is_empty():
                print('The stack is empty, but there comes a righty')
                return False
            popped = S.pop()
            print(f'Pop {popped} from the stack')
            if righty.index(symbol) != lefty.index(popped):
                print(f'The righty {symbol} is not matching to {popped}')
                return False
            print(f'The popped symbol {popped} is matched to {symbol}')
        S.display()
    return S.is_empty()
```

In [5]:

```
print(ParenMatch_explained(list('()(()){([()])}')))
```

```
Processing ['(', ')', '(', '(', ')', ')', '{', '(', '[', '(', ')', ']', ')', '}']
#0 - Processing (
Push ( into the stack
STACK: B|(|T
#1 - Processing )
Pop ( from the stack
The popped symbol ( is matched to )
STACK: B||T
#2 - Processing (
Push ( into the stack
STACK: B|(|T
#3 - Processing (
Push ( into the stack
STACK: B|((|T
#4 - Processing )
Pop ( from the stack
The popped symbol ( is matched to )
STACK: B|(|T
#5 - Processing )
Pop ( from the stack
The popped symbol ( is matched to )
STACK: B||T
#6 - Processing {
Push { into the stack
STACK: B|{|T
#7 - Processing (
Push ( into the stack
STACK: B|{(|T
#8 - Processing [
Push [ into the stack
STACK: B|{([|T
#9 - Processing (
Push ( into the stack
STACK: B|{([(|T
#10 - Processing )
Pop ( from the stack
The popped symbol ( is matched to )
STACK: B|{([|T
#11 - Processing ]
Pop [ from the stack
The popped symbol [ is matched to ]
STACK: B|{(|T
#12 - Processing )
Pop ( from the stack
The popped symbol ( is matched to )
STACK: B|{|T
#13 - Processing }
Pop { from the stack
The popped symbol { is matched to }
STACK: B||T
True
```

In [6]:

```
print(ParenMatch_explained(list('((()(()){([()])}))')))
```

```
Processing ['(', '(', '(', ')', '(', '(', ')', ')', '{', '(', '[', '(', ')', ']', ')', '}', ')', '
)']
#0 - Processing (
Push ( into the stack
STACK: B|(|T
#1 - Processing (
Push ( into the stack
STACK: B|((|T
#2 - Processing (
Push ( into the stack
STACK: B|(((|T
#3 - Processing )
Pop ( from the stack
The popped symbol ( is matched to )
STACK: B|((|T
#4 - Processing (
Push ( into the stack
STACK: B|(((|T
#5 - Processing (
Push ( into the stack
STACK: B|((((|T
#6 - Processing )
Pop ( from the stack
The popped symbol ( is matched to )
STACK: B|(((|T
#7 - Processing )
Pop ( from the stack
The popped symbol ( is matched to )
STACK: B|((|T
#8 - Processing {
Push { into the stack
STACK: B|(({|T
#9 - Processing (
Push ( into the stack
STACK: B|(({(|T
#10 - Processing [
Push [ into the stack
STACK: B|(({([|T
#11 - Processing (
Push ( into the stack
STACK: B|(({([(|T
#12 - Processing )
Pop ( from the stack
The popped symbol ( is matched to )
STACK: B|(({([|T
#13 - Processing ]
Pop [ from the stack
The popped symbol [ is matched to ]
STACK: B|(({(|T
#14 - Processing )
Pop ( from the stack
The popped symbol ( is matched to )
STACK: B|(({|T
#15 - Processing }
Pop { from the stack
The popped symbol { is matched to }
STACK: B|((|T
#16 - Processing )
Pop ( from the stack
The popped symbol ( is matched to )
STACK: B|(|T
#17 - Processing )
Pop ( from the stack
The popped symbol ( is matched to )
STACK: B||T
True
```

In [7]:

```
print(ParenMatch_explained(list(')(()){([()])}')))
```

```
Processing [')', '(', '(', ')', ')', '{', '(', '[', '(', ')', ']', ')', '}']
#0 - Processing )
The stack is empty, but there comes a righty
False
```

In [8]:

```
print(ParenMatch_explained(list('({[])}')))
```

```
Processing ['(', '{', '[', ']', ')', '}']
#0 - Processing (
Push ( into the stack
STACK: B|(|T
#1 - Processing {
Push { into the stack
STACK: B|({|T
#2 - Processing [
Push [ into the stack
STACK: B|({[|T
#3 - Processing ]
Pop [ from the stack
The popped symbol [ is matched to ]
STACK: B|({|T
#4 - Processing )
Pop { from the stack
The righty ) is not matching to {
False
```

In [9]:

```
print(ParenMatch_explained(list('(')))
```

```
Processing ['(']
#0 - Processing (
Push ( into the stack
STACK: B|(|T
False
```

In [ ]:

```

```

In [ ]:

```

```