Lab3 Parentheses Checker

Write a program to read a text file and print whether or not the parentheses are balanced. (use stack) The **data file**, lab3.txt should contain the following data:

```
1. (A * B) + ( C * D)
2. (A * ( B + ( C * D + E)))
3. ( A + B ) - { C + D } - [F + G]
4. ( ( A + B)
5. ) A + B ( - C
6. (A + B)) - (C + D
7. { [ A + B ] - [ ( C -D ) ]
8. (A + B } )
9. { [ A + B ) - [ ( C -D ) ] }
```

Output:

```
1. (A * B) + ( C * D)                    Valid
2. (A * ( B + ( C * D + E)))             Valid
3. ( A + B ) - { C + D } - [F + G]       Valid
4. ( ( A + B)                            Invalid (Unbalanced parentheses)
5. ) A + B ( - C                         Invalid (Unbalanced parentheses)
6. (A + B)) - (C + D                     Invalid (Unbalanced parentheses)
7. { [ A + B ] - [ ( C -D ) ]            Invalid (Unbalanced parentheses)
8. (A + B } )                            Invalid (Mismatched parentheses)
9. { [ A + B ) - [ ( C -D ) ] }          Invalid (Mismatched parentheses)
```

Algorithm 참조:

```
int check() {
  for(i=0; i<strlen(exp); i++){
      if(exp[i]=='(' || exp[i]=='{' || exp[i]=='[')      push(exp[i]);
      if(exp[i]==')' || exp[i]=='}' || exp[i]==']')
              if(stack empty) {    print("UnBalanced ");
                else { temp=pop();
                      if(!match(temp, exp[i]))    // match(a,b) { if (a==b) return true else return false}
                          print("Mismatched"    temp, "and " exp[i])
  }// end of for
  if(stack empty)    return true      else    { print("Unbalanced"); return false}

  void main() {
      open data file // check file open error
      while (infile.getline(buffer, 80)) {
          validity = check_paranthesis( );
          if (validity is true) print "valid"    else print "Invalid"
    }
```

- Extra point (**STACK – Palindrome)**

Write a program to read a text file and print whether or not a line is a palindrome.

1. palindrome: A palindrome is a string that reads the same forward and backward.

   (ex) radar 0 1 00 11 aba 1101011 등

2. **data file** : abccba abckcba abbc abbacd

3. Output:

```
abccba                  a palindrome
abckcba                 a palindrome
abbc                    not a palindrome
abbacd                  not a palindrome
```

4. Algorithm

```
while (!EOLN)) {
    length = strlen(buffer);             //string 의 길이, 글자수
    if length is EVEN {      // (len % 2) == 0,
      while (i < (len / 2))    //length의 반만큼 PUSH.  'abccba' 의 경우 push a, push b, push c.
          push(buffer[i]);
    }
    else if len is ODD {     // (len % 2) != 0,
      while (i < (len / 2))    //length의 반만큼 PUSH.  'abckcba'의 경우 push a, push b, push c.
          push(buffer[i]);    // 이젠 buffer 에 kcba  남았음
           I++;               //k를 건너 뛴다.
    }
    compare (buffer[i]  and  pop(ch))    // if same compare next,  if not SAME, then BREAK;
                                 //만일 같으면 다음 글자 비교, 다르면 error message & break
    If (buffer[i] = "empty") then print "PALINDROME"
}
```