Assignment 5 Postfix Calculator

- **Assignment #5 Postfix Calculator**
  - เขียนโปรแกรมรับ Expression 1 บรรทัด แล้วคำนวนหาคำตอบ (ans) เป็นจำนวนจริง double
  - คำนวนมุมเรขาคณิตในหน่วย degree
  - case insensitive ใช้อักษรตัวใหญ่หรือตัวเล็ก พิมพ์ชื่อฟังก์ชันได้
  - สามารถใช้ค่าคงที่ต่างๆ เช่น pi , e , g (อย่างน้อย 1 ตัวคือ pi)
  - สามารถใช้ฟังก์ชันก์คำนวนตามที่มี เช่น sin(), cos(), tan(), asin(), acos(), atan(), sqrt (), exp (), log (), abs() ...
  - operator พื้นฐาน คือ + - * / ^ ( )
  - ให้มี sign operator อย่างน้อย 1 ตัวคือ "-" (หรือกำหนดใหม่เอง เช่น ! ~)
  - วนรอบเพื่อรับ Expression บรรทัดใหม่ เมื่อคำนวณเสร็จ
  - สามารถนำคำตอบจากบรรทัดที่แล้ว (ans) มาคำนวนในบรรทัดใหม่นี้
  - จะให้มีตัวแปร X1 .. X10 หรือไม่มีก็ได้ (ถ้ามีต้องเพิ่ม operator '=')
  - มีคำสั่ง help หรือ token เพื่อแสดงชื่อฟังก์ชันหรือ token ที่ใช้งานได้
  - จะแสดงค่า postfix equation ด้วยหรือไม่ก็ได้
  - มีการป้องกัน error เพื่อให้โปรแกรมทำงานเฉพาะที่ทำได้ เช่น การลำดับผิด การใส่วงเล็บไม่ถูกต้อง การหารากของจำนวนลบ การหารด้วยศูนย์ เป็นต้น ไม่ให้โปรแกรมหยุดทำงานก่อนสั่งจบโปรแกรม
  - มีคำสั่งจบโปรแกรม เช่น end / quit /exit

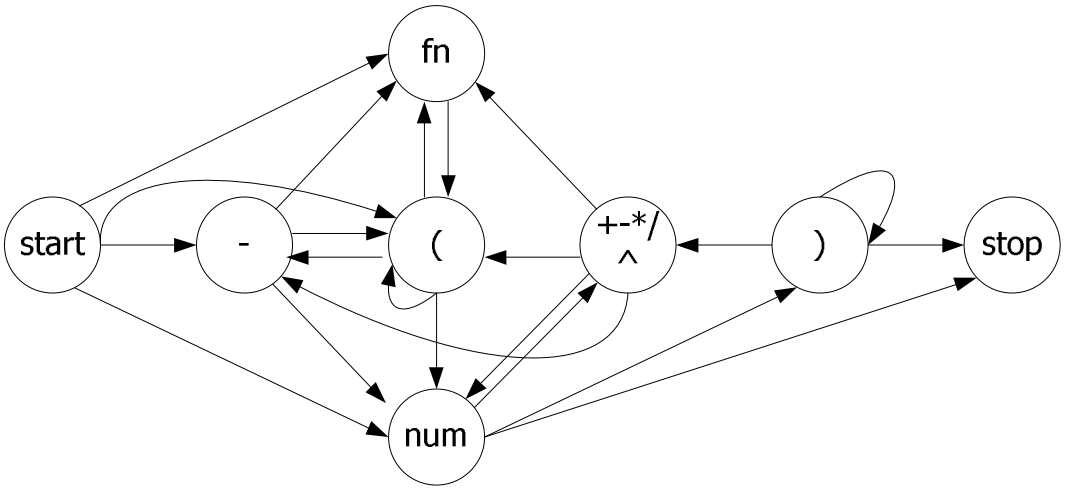
Assignment 5.1 Check Token

- โจทย์ปัญหา กำหนดฟังก์ชัน main() ที่ใช้ในการทดสอบสตริงไว้แล้ว ให้สร้างฟังก์ชันที่สอดคล้องกับการเรียกใช้ และให้ผลลัพธ์ตามที่กำหนด
- ขั้นตอน เขียนฟังก์ชันทดสอบสตริงที่สอดคล้องกับการเรียกใช้และผลลัพธ์
  - แยกสตริง ใส่ในอาร์เรย์
  - number คือตัวเลขจำนวนจริงเช่น 12.5 123
  - function ได้แก่คำว่า sin,cos,tan,asin,acos,atan,sqrt,log,exp,abs รวม 10 ตัว
  - operator ได้แก่เครื่องหมาย + - * / ^ ( ) รวม 7 ตัว
  - variable/constant ได้แก่ คำที่ตั้งชื่อไว้ล่วงหน้า เช่น ans , pi, g , e เป็นต้น
  - error ได้แก่ คำอื่นๆ ที่นอกเหนือจากที่กำหนด
- ตัวอย่างฟังก์ชันที่ควรจะสร้างขึ้น
  - สร้างฟังก์ชันแบ่ง token ใส่ในอาร์เรย์ return count //อาจต้องเพิ่ม space
  - สร้างฟังก์ชันตรวจสอบ negative sign (- , ! , ~)
  - สร้างฟังก์ชันตรวจสอบชนิดของ operator return operator group/code
  - สร้างฟังก์ชันตรวจสอบตัวเลขจำนวนจริง return success/value
  - สร้างฟังก์ชันตรวจสอบชื่อตัวแปร (ans, pi, g, e) return success/value
  - สร้างฟังก์ชันตรวจสอบชื่อเฉพาะ (function) return function group/code
  - ฟังก์ชันก์ในการตรวจสอบลำดับ syntax และ ( )
  - กรณีฟังก์ชันทำงาน return code ให้กำหนด ไม่สำเร็จให้เป็น 0 สำเร็จ >0

Assignment 5.1

- **Syntax Group**
  - **(เครื่องหมายวงเล็บเปิด**
  - **) เครื่องหมายวงเล็บปิด**
  - **function ได้แก่ sin,cos,tan,asin,acos,atan,sqrt,log,exp,abs รวม 10 ตัว**
  - **negative sign เครื่องหมาย [-] ที่เขียนนำหน้าตัวเลข(อาจใช้ตัวอื่นแทน)**
  - **binary operator ได้แก่เครื่องหมาย [+ - * / ^] รวม 5 ตัว (3 ลำดับความสำคัญ)**
  - **number/variable ได้แก่ตัวเลขจำนวนจริง,ชื่อตัวแปร ans,ค่าคงที่ pi**
- **Balance bracket และต้องไม่มี) มาก่อน (**
- **State diagram**

Present		Next State	
0	start	number, sign, fn, (	1,5,6,7
1	number	operator,) , stop	2,8,9
2	operator	number, sign, fn, (	1,5,6,7
5	sign	number, fn, (	1,6,7
6	fn	(	7
7	(	number, sign, fn, (	1,5,6,7
8	)	operator,) , stop	2,8,9
9	stop		



Assignment 5.1 Test Case

1. expression> 2.5+pi
answer> OK
2. expression> 2.5 pi + // postfix
answer> error
3. expression> -2.54 + 1.68 (2 + 3) // ตัวเลข ตามด้วย (
answer> error
4. expression> -2.5 - sin (-30) // Negative sign
answer> OK
5. expression> sin(cos(tan(asin(acos(atan(log(sqrt(exp(abs(ans)))))))))//วงเล็บปิด 9 ตัว
answer> error
6.expression> sin(cos(tan(asin(acos(atan(log(sqrt(exp(abs(ans))))))))))//วงเล็บปิด10 ตัว
answer> OK
7. expression> 2 + 3 - ((4))
answer> OK
8. expression> 2 + 3-(4/5*) // operator ตามด้วยวงเล็บ)
answer> error
9. expression> 2 + 3-(4/5*6+()) // วงเล็บ(ตามด้วยวงเล็บ)
answer> error
10. expression> 2 + 3)+4/5*6+((7) // วงเล็บปิด) มาก่อน วงเล็บเปิด(
answer> error
11. expression> -1+2-3*4/5^6
answer> OK
12. expression> help // หรือใช้คำสั่ง token
answer> token = sin, cos, tan, asin, acos, atan, sqrt, log, exp, abs +, -, *, /, ^, (,), pi, ans
expression> end // คำสั่งจบโปรแกรม
End program

Assignment 5.2

- หาคำตอบของสมการ infix ที่ตรวจสอบจาก 5.1 แล้วว่าไม่มีข้อผิดพลาด
  - ถ้ามีความผิดพลาดจาก 5.1 ให้แจ้ง error แล้ววนรอบรับข้อมูลใหม่
- ควรแปลงสมการ infix ให้อยู่ในรูปของ postfix ก่อน
  - จองอาร์เรย์/ลิสต์ของสตริง เพื่อใช้เก็บ token ที่ได้จากการวิเคราะห์คำ
  - จองอาร์เรย์ /ลิสต์ของสตริง เพื่อใช้เป็น Operator Stack สำหรับเก็บคำสั่ง
  - จองอาร์เรย์ /ลิสต์ของสตริง เพื่อใช้เป็น postfix queue สำหรับเก็บรูปสมการ postfix
  - สร้างฟังก์ชันวนรอบเพื่อแปลง infix arrayให้อยู่ในรูป postfix array ตามอัลกอริทึม (ต้องตรวจสอบความถูกต้องของ postfix ก่อนทำขั้นตอนถัดไป)
- หาคำตอบของสมการที่อยู่ใน postfix array
  - จองอาร์เรย์ของตัวเลขจำนวนจริง เพื่อใช้เป็น Operand stack
  - วนรอบดึงข้อมูลออกจาก postfix array มาทีละตัว แล้วทำตามอัลกอริทึม

Assignment 5.2

- การแปลงสมการ infix ให้อยู่ในรูป postfix
  1. ใส่ "(" และ ")" คร่อม infix expression //ไม่ใส่ก็ได้ แต่ต้องเช็คstack(4) ตอนหมดข้อมูล
  2. ดึงข้อมูล (TOKEN) ออกจาก string ทีละชุด จากซ้ายไปขวา
     - 2.1 ถ้าเป็น "(" ให้ PUSH ใส่ใน STACK (STACK ของ CHAR/STRING)
     - 2.2 ถ้าเป็น ตัวเลข number ให้ส่งไปยัง output //อาจเป็น Queue หรือสตริง
     - 2.3 ถ้าเป็น operator ให้พิจารณาดังนี้
       - 2.3.1 POP operators ทุกตัวที่มีลำดับความสำคัญมากกว่าหรือเท่ากับ operator ที่พบใน 2.3 ส่งไปยัง postfix output
       - 2.3.2 PUSH operator ที่เจอใน 2.3 ใส่เข้าไปใน STACK แทน
     - 2.4 ถ้าเป็น ")" ให้ POP operators ทุกตัวจาก STACK ส่งไปยัง postfix output จนกว่าจะเจอ "(" // pop "(" ออกมา แต่ไม่ต้องส่ง ไป output
  3. ทำซ้ำในข้อ 2 จนกว่าข้อมูลจะหมด // ข้อมูลจะหมดที่ ")" ที่เพิ่มเข้าไป
  4. ถ้ามีข้อมูลเหลืออยู่ใน stack ให้ POP ข้อมูลทั้งหมดออกมาด้วย //กรณีที่ไม่ได้เพิ่ม ")"
- ฟังก์ชันที่แนะนำ
  - ฟังก์ชันเพิ่ม (และ) เข้าไปในสตริง หรืออาร์เรย์ของ token ที่เช็คแล้ว
  - ฟังก์ชัน peek เพื่อดูข้อมูล operator ตัวบนสุดของ stack
  - ฟังก์ชันเพื่อแปลง operator +- , */, ^, !, fn, (,) เป็นเลขตามลำดับความสำคัญ
  - ฟังก์ชันเกี่ยวกับการจัดการ stack ของคำสั่ง(Operator) เช่น
    - pop_operator(token) ทำหน้าที่ดึง token ตัวบนสุดออกจาก stack (2.3.1)
    - push_operator(token) ทำหน้าที่ push token ใส่เข้าใน stack
  - ฟังก์ชันที่ใช้ add token ไปใส่ไว้ใน postfix array

Assignment 5.2

- การหาคำตอบของสมการ postfix
  1. ดึงข้อมูล (TOKEN) ออกจากสมการ(string/queue) ทีละตัวจากซ้ายไปขวา
     - 1.1 ถ้าเป็น ตัวเลข/ค่าคงที่ pi/ตัวแปร ans ให้ PUSH ใส่ใน STACK (ของเลขจำนวนจริง)
     - 1.2 ถ้าเป็น unary operator (Negative, function) ให้ POP ตัวเลข 1 ตัว จาก STACK มาคำนวณตามคำสั่ง แล้วนำคำตอบที่ได้ PUSH ใส่กลับเข้าไปใน STACK

       (กำหนดให้การคำนวณมุมทางเรขาคณิตอยู่ในโหมดของ degree ดังนั้นจะต้องมีการแปลงค่าในคำสั่ง sin, cos, tan, asin, acos, atan ให้ถูกต้องด้วย)
     - 1.3 ถ้าเป็น binary operator (+, -, *, /, ^) ให้ POP ตัวเลขออกจาก STACK 2 ตัวมาคำนวณตามคำสั่ง (POPตัวหลัง operate POPตัวแรก) แล้วนำคำตอบที่ได้ PUSH ใส่กลับเข้าไปใน STACK
  2. ทำซ้ำ ในข้อ 1 จนกว่าข้อมูลจะหมด
  3. POP ตัวเลขจาก STACK (เหลือตัวเดียว) มาเป็นคำตอบ
- ฟังก์ชันที่แนะนำ
  - ฟังก์ชันที่ใช้ตรวจ token ว่าเป็นตัวเลข (ต้องแปลงเป็นจำนวนจริงด้วย)
  - ฟังก์ชันการคำนวณตามรหัสของแต่ละ operator หรือฟังก์ชัน
  - ฟังก์ชันที่แยกรหัสของ operator แต่ละตัว
  - ฟังก์ชัน push ตัวเลขไปยัง stack
  - ฟังก์ชัน pop ตัวเลขที่อยู่ใน stack เพื่อนำมาคำนวณหาคำตอบ

Assignment 5.2 Test Case

```
expression> 1.23  4.56 +
answer> error
expression>sin 90
answer> error
expression> 0/1
answer> 0
expression> 1/sin(0)
answer> error
expression> sin(0)/sin(0)
answer> error
expression> -2^-2-2
answer>-1.75
13expression> -1+2^3/(4-5*6)+pi
answer> 1.8339
14expression> sqrt(log(10^2)+exp(3))
answer> 4.69953
15expression>sqrt(3^2+4^2)
answer> 5
17expression> ans^-2
answer> 0.04
16expression> asin(1-cos(0)+sin(90))
answer> 90
18expression>sin(30)^2+cos(30)^2
answer> 1
expression> end      // คำสั่งจบโปรแกรม
End program
```