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

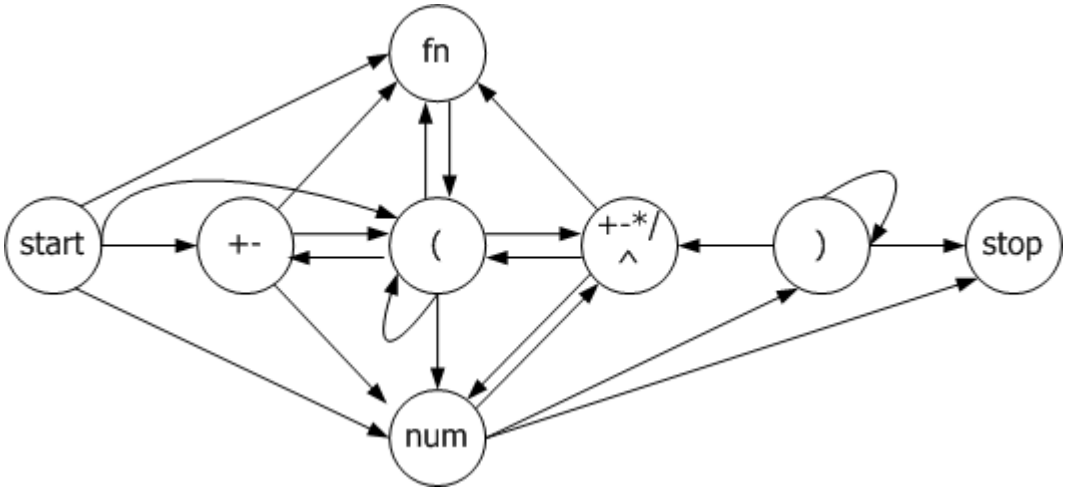
Assignment 5.1 Check Token

- 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**

0	start	(, fn, number, sign
1	number	operator,) , stop
2	operator	number, fn, (
3	sign	number, fn, (
4	fn	(
5	(	(, fn, number, unary
6	)	operator,) , stop
7	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

ตัวอย่าง...ken

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
int main()
{  char str[500],token[50][20];
   int i,count=0,code;
   double  num;
   do
   {  printf("Command> ");
      gets(str);
      strlwr(str);
      add_space(str);
      count=split(str,token);
      change_sign_operator(token,count);
      i = check_state(token,count);
      if (i==-1||i==2||i==4||i==5)
         printf("Syntax Error\n");
      else printf("Syntax OK\n");
   }  while(strcmp(str,"end")!=0);
   return 0;
}
```

```
int check_state(char token[][20],int count)
{  int state = 0,next=0,b=0,i;
   for (i=0; i<count&&state>=0; i++)
   {  state = next;
      next = check_group(token[i]);
      if (next==0) state = -1;
      if (next==5) b++;
      else if (next==6) b--;
      if (b<0)
         state=-1;
      else if (state==0&&(next==6||next==2))
         state = -1;
      else if (state==1&&(next==1||next==4||next==5))
         state = -1;
      else if (state==2&&(next==2||next==6))
         state = -1;
      else if (state==3&&(next==2||next==6))
         state = -1;
      else if (state==4&&(next!=5))
         state = -1;
      if (state==5&&(next==6||next==2))
         state = -1;
   }
   if (b!=0||state==-1)
      return -1;
   else
      return next;
}
```

```
int check_group(char token[])
{  double num;
   int b=0;
   if (check_number(token,&num)>0)
      return 1;
   else if (check_operator(token)>0)
   {  if (strchr("+-",token[0])!=NULL)
         return 2;
      else if (strchr("*/^",token[0])!=NULL)
         return 2;
      else if (strcmp(token,"!")==0)
         return 3;
      else if (strcmp(token,"(")==0)
         return 5;
      else if (strcmp(token,")")==0)
         return 6;
   }
   else if (check_function(token)>0)
      return 4;
   else
      return 0;
}
```

```
void change_sign_operator(char token[][20],int count)
{ int i;
   if (strcmp(token[0],"-")==0)
      strcpy(token[0],"!");
  for(i=1;i<count;i++)
      if ((strcmp(token[i+1],"-")==0)&&(strstr("(+-*/^",token[i])!=NULL))
         strcpy(token[i+1],"!");
}
```

เฉลย Check Token C

```
void add_space(char *str)
{ char buff[255]="",old[255]="";
   int i,j;
   for (i=0,j=strlen(str);i<j;i++)
       { if (strchr("+-*/^()",str[i])!=NULL)
            sprintf(buff,"%s %c ",old,str[i]);
          else
            sprintf(buff,"%s%c",old,str[i]);
          strcpy(old,buff);
        }
  strcpy(str,buff);
}
```

```
int check_identifier(char *str)
{ int flag=1, i;
   int len=strlen(str);
   if ((str[0]>='a'&&str[0]<='z')||str[0]=='_')
     {  for(i=0;i<len;i++)
           if (!(isalpha(str[i])||isdigit(str[i])||str[i]=='_'))
             flag=0;
        return flag;
     }
   else
      return 0;
}
```

```
int check_number(char *str, double *n)
{ char *end;
   *n=strtod(str, &end);
   if (strcmp(end,"")!=0)
     return 0;
   else
     return 1;
}
```

```
int split(char *str, char token[][20])
{  int count=0;
   char *tok;
   tok=strtok(str," ");
   while(tok!=NULL)
   {  strcpy(token[count++],tok);
      tok=strtok(NULL," ");
   }
   return count;
}
```

```
int check_operator(char *st)
{  if (strcmp(st,"+")==0) return 1;
   else if (strcmp(st,"-")==0)  return 2;
   else if (strcmp(st,"*")==0)  return 3;
   else if (strcmp(st,"/")==0)  return 4;
   else if (strcmp(st,"^")==0)  return 5;
   else if (strcmp(st,"!")==0)  return 6;
   else if (strcmp(st,"(")==0)  return 7;
   else if (strcmp(st,")")==0)  return 8;
   else return 0;
}
```

```
int check_function(char *st)
{  if (strcmp(st,"sin")==0) return 11;
   else if (strcmp(st,"cos")==0)   return 12;
   else if (strcmp(st,"tan")==0)   return 13;
   else if (strcmp(st,"asin")==0)  return 14;
   else if (strcmp(st,"acos")==0)  return 15;
   else if (strcmp(st,"atan")==0)  return 16;
   else if (strcmp(st,"sqrt")==0) return 17;
   else if (strcmp(st,"log")==0)   return 18;
   else if (strcmp(st,"exp")==0)   return 19;
   else if (strcmp(st,"abs")==0)   return 20;
   else return 0;
}
```