CẤU TRÚC CHƯƠNG TRÌNH

- Cách tổ chức dữ liệu:

- Xem các toán tử, toán hạng, dấu ngoặc là token có cấu trúc:

```
struct Token
{
	int Kieu;
	double GiaTri;
};
```

- Stack và Queue là các danh sách liên kết đơn, với mỗi phần tử có dạng:

```
struct Nod
{
	Token key;
	Nod *next;
};

struct Stack
{
	Nod * stackTop;
};

struct Queue
{
	Nod *head;
	Nod *tail;
};
```

- Các hàm sử dụng:

```
bool IsEmpty(Stack stack);                  // Kiểm tra stack có rỗng hay không
bool IsEmpty(Queue queue);                  // Kiểm tra queue có rỗng hay không
void InitStack(Stack &stack);               // Khởi tạo stack
void InitQueue(Queue &queue);               // Khởi tạo queue
void PushStack(Token a, Stack &stack);      // Thêm phần tử vào đỉnh Stack
void PushQueue(Token a, Queue &queue);      // Thêm phần tử vào cuối Queue
Token PopStack(Stack &stack) ;              // Lấy phần tử ở đỉnh khỏi Stack
Token PopQueue(Queue &queue);               // Lấy phần tử ở đỉnh ra khỏi Queue
Token TopStack(Stack stack);                // Xem giá trị phần tử ở đỉnh stack
```

void checkString(string &s);

// Xử lí chuỗi trước khi tạo token, thêm số 0 vào trước toán tử "-" một ngôi

bool CreateToken(string s, Token a[], int &n);

// Tạo token và kiểm tra độ chính xác của biểu thức
+ Với số: Tạo token với Kieu la 0. Đọc các chữ số liên tiếp nhau, cứ nhân 10 rồi cộng dồn vào. Đối với số thực thì dùng một biến đếm t để đếm số chữ số sau dấu ".", rồi lấy số ban đầu chia cho 10^t. Đối với số âm, thì dùng một biến bool để kiểm tra (Lưu ý: dấu âm phải được viết liền với số, ví dụ "-95")
+ Với dấu đóng, mở ngoặc: Tạo Token với Kieu la -1. Dùng biến đếm lưu số ngoặc mở và đóng. Nếu chúng không bằng nhau thì trả về biểu thức không hợp lệ.
+ Với toán tử "+" và "-": Tạo Token với Kieu la 1.
+ Với toán tử "*" và "/": Tạo Token với Kieu la 2.
+ Với toán tử "^" và "!": Tạo Token voi Kieu la 3.
+ Nếu có các kí tự khác các kí tự quy định thì trả về biểu thức sai.

void InFixToPostFix(Token a[], int n, Stack &stack, Queue &queue);
// Chuyển từ dạng trung tố sang hậu tố
+ Khởi động Stack rỗng
+ Khởi động Queue rỗng
+ Lặp khi chưa số Token đã tạo
{
- Đọc một Token
- Nếu Token là:
o Ngoặc trái: Push (Token, Stack).
o Ngoặc phải: lặp lại thao tác Push(Pop(Stack), Queue) cho đến khi gặp ngoặc trái (pop cả ngoặc trái nhưng không push nó vào Queue).
o Toán tử: (Kieu = 1, 2, 3) Xét độ ưu tiên dựa trên Kieu để đẩy vào Stack hay Queue

```
if (!IsEmpty(stack))
{
        while (a[i].Kieu <= TopStack(stack).Kieu)
                Push(Pop(Stack), Queue);
}
Push(Token, Stack)
```

o Toán hạng : (Kieu = 0) Push(Token, Queue).
Đẩy hết các Token còn lại trong stack vào Queue

```
while (!isEmpty (Stack))
        Push(Pop(Stack), Queue))
```

double PostFixToInFix(Stack &stack, Queue &queue, bool &kt);
- Khởi động Stack rỗng.

```
while (!isEmpty(Queue))
{
        x = Pop(Queue)
```

Nếu x là :
- Toán hạng: Push(X, Stack)
- Toán tử:
+ Một ngôi thì lấy ra 1 Token, hai ngôi lấy 2 Token. Nếu không còn Token để lấy thì trả về biểu thức sai.
U = Pop(Stack)
V = Pop(Stack)
+ Tính toán sau đó đẩy kết quả lại vào stack
Y = X(V, U) // Thực hiện toán tử X cho 2 toán hạng trong Stack.
Push(Y, Stack)
}
- Trả về giá trị cuối cùng trong stack;