BÁO CÁO CHALLENGE
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

- Group id: 245479
- Các thành viên:
    - Trần Quốc Đông MSSV: 19120479
    - Nguyễn Hồ Diệu Dương MSSV: 19120524
    - Lê Kiệt MSSV: 19120554
- Tiến độ hoàn thành: 100%
- Option: Floating-point

1. Thuật toán kiểm tra lỗi.

- Một biểu thức luôn theo quy tắc:

- Bắt đầu bằng số. Nếu bằng ngoặc thì phải kết thúc bằng đúng ngoặc đóng của dấu ngoặc đó
- Xen kẽ giữa 2 số là một phép toán.
- Nếu trước 1 biểu thức bắt đầu bằng ngoặc phải kết thúc bằng đúng ngoặc đóng của dấu ngoặc đó.
- Sử dụng dấu ngoặc theo thứ tự hợp lệ (nghĩa là () < [] < {})

- Xây dựng một hàm kiểm tra một biểu thức có đúng với những qui tắc trên.
- Hàm nhận vào một istream và một char chứa dấu ngoặc bắt đầu.
- Xem các số và phép toán bên trong một dấu ngoặc là một biểu thức con. Sau khi kiểm tra biểu thức con thì biểu thức con sẽ được tính như một số.
- Truyền vào hàm biểu thức ban đầu để kiểm tra các quy tắc của biểu thức.
  Nếu gặp một trong những dấu ngoặc thì ta gọi đệ qui hàm kiểm tra biểu thức để kiểm tra biểu thức con đó.
- Trong quá trình kiểm tra, hàm sẽ trả về false ngay khi vi phạm bất kỳ qui tắc nào.
  VD 1: 3 ^ [3.65 * (2.12 * 8.9) + 5.01] + 6
  → Ở biểu thức đầu:
  Có 3 số : 3 và [biểu thức con 1] và 6
  Có 2 phép toán : ^ và +
  Trong quá trình đọc từ trái sang phải thì ta thấy theo đúng qui tắc → Trả về true
  → Biểu thức con 1:
  Có 3 số : 3.65 và (biểu thức con 2) và 5.01
  Có 2 phép toán : * và +
  Trong quá trình đọc từ trái sang phải thì ta thấy theo đúng qui tắc → Trả về true
  → Biểu thức con 2:
  Có 3 số : 2.12 và 8.9
  Có 2 phép toán : *
  Trong quá trình đọc từ trái sang phải thì ta thấy theo đúng qui tắc → Trả về true

  VD 2: (3 + 6) (6 + 3)
  → Ở biểu thức đầu:
  Có 2 số : (biểu thức con 1) và (biểu thức con 2)
  Có 0 phép toán :
  Trong quá trình đọc từ trái sang phải sau khi đọc xong biểu thức 1 là số liền tới một biểu thức 2 là số khác nên sai qui tắc → Trả về false
  → Biểu thức con 1:
  Có 2 số : 3 và 6
  Có 1 phép toán : +
  Trong quá trình đọc từ trái sang phải thì ta thấy theo đúng qui tắc → Trả về true

2. Thuật toán chuyển biểu thức trung tố (infix expression) sang biểu thức hậu tố (postfix expression)

- Trong biểu thức trung tố (infix), ta phải tuân theo luật ưu tiên sau (số trong cột "Thứ tự" càng bé thì càng được ưu tiên):

Thứ tự	Toán tử/ngoặc
1	Ngoặc, gồm () < [] < {}
2	Mũ (right → left),
3	Nhân, chia
4	Cộng, trừ

❖ Thuật toán chuyển đổi từ biểu thức trung tố sang biểu thức hậu tố

- Cấu trúc của 1 biểu thức hậu tố:

  <toán hạng trái><toán hạng phải><toán tử>

  ➔ 1 biểu thức trung tố muốn chuyển sang hậu tố thì phải tìm được 3 phần trên

- Ý tưởng thuật toán: Thấy rằng, nếu đang xét 1 toán tử, nếu toán tử ß có độ ưu tiên cao hơn toán tử α trước đó thì phải làm toán tử ß trước → sử dụng tính chất LIFO của stack. Còn nếu đang xét 1 toán hạng, chỉ cần append nó vào chuỗi kết quả chứ không so sánh gì
- Các bước thuật toán:

- Bước 1: Tạo 1 biến stack và biến kết quả để lưu kết quả sau khi chuyển đổi

- Bước 2: Kiểm tra phần tử đang duyệt

+ Nếu phần tử đang duyệt là toán hạng thì in vào chuỗi kết quả.

+ Nếu phần tử đang duyệt là toán tử

  - Nếu stack = empty thì push vào stack. Còn nếu độ ưu tiên của nó **lớn hơn** stack.top thì push vào stack; ngược lại thì pop và print stack.top ra chuỗi kết quả và tiếp tục kiểm tra phần tử đang được duyệt với stack.top (để khi lấy theo qui tắc LIFO, toán tử có độ ưu tiên cao nhất đi ra trước).
  - Nếu stack.top là dấu mở ngoặc thì tự động push phần tử đang xét vào stack

- Nếu phần tử đang được duyệt là dấu mở ngoặc thì push vào stack. Nếu phần tử đang được duyệt là dấu đóng ngoặc thì pop và in toán bộ toán tử trong trong stack cho đến khi tìm thấy dấu mở ngoặc tương ứng

- Bước 3: Sau khi duyệt hết biểu thức infix, còn bao nhiêu toán tử trong stack thì append hết vào chuỗi kết quả sẽ cho ra chuỗi postfix

VD 1: [0 + (1 + 2)] * 3 - 4

Phần tử đang xét	stack	postfix
[	[	
0	[	0
+	[+	0
(	[+ (	0
1	[+ (	0 1
+	[+ (+	0 1
2	[+ (+	0 1 2
)	[+ gặp dấu ‘)’ thì append tất cả các toán tử trước dấu ‘(‘ trong stack, xong xóa ‘(‘, coi như biểu thức trong cặp ngoặc tròn đã xong	0 1 2 +
]	empty gặp dấu ‘]’ thì append tất cả các toán tử trước dấu ‘[‘ trong stack, xong xóa ‘[‘, coi như biểu thức trong cặp ngoặc vuông đã xong	0 1 2 + +
*	*	0 1 2 + +
3	*	0 1 2 + + 3
-	-	0 1 2 + + 3 *
4	-	0 1 2 + + 3 * 4
Hết duyệt	Còn toán tử nào trong stack thì append nó vào postfix	0 1 2 + + 3 * 4 -

NOTE: Nếu toán tử là dấu mũ và là mũ lồng (VD: 2^3^2) thì phải ưu tiên tính từ phải qua trái, nghĩa là 2^3^2 → 2^(3^2) = 2^9 = 512. Nếu áp dụng thuật toán trên cho ra kết quả là “2 3 ^ 2 ^” – sai so với đáp án “2 3 2 ^ ^”. Chính vì thế khi số sánh độ ưu tiên của dấu mũ (‘^’), khi stack.top và phần tử đang xét cùng có ‘^’, vậy thì giữ nguyên stack.top và push toán tử ‘^’ vào stack (nghĩa là ít nhất 2 dấu ‘^’ kề nhau trong stack)

VD 2: 3.84 + (7 - 1.5) ^ (1 + 2) ^ (5 + 3)

Phần tử đang xét	stack	postfix
3.84		3.84
+	+	3.84
(	+ (	3.84
7	+ (	3.84 7
-	+ (-	3.84 7
1.5	+ (-	3.84 7 1.5

)	+	3.84 7 1.5 -
^	+ ^	3.84 7 1.5 -
(	+ ^ (	3.84 7 1.5 -
1	+ ^ (	3.84 7 1.5 - 1
+	+ ^ (+	3.84 7 1.5 - 1
2	+ ^ (+	3.84 7 1.5 – 1 2
)	+ ^	3.84 7 1.5 – 1 2 +
^	+ ^ ^ stack.top & phần tử đang xét đều cùng là '^' nên giữ nguyên stack.top và push phần tử đang xét (là '^') vào stack	3.84 7 1.5 – 1 2 +
(	+ ^ ^ (	3.84 7 1.5 – 1 2 +
5	+ ^ ^ (	3.84 7 1.5 – 1 2 + 5
+	+ ^ ^ (+	3.84 7 1.5 – 1 2 + 5
3	+ ^ ^ (+	3.84 7 1.5 – 1 2 + 5 3
)	+ ^ ^	3.84 7 1.5 – 1 2 + 5 3 +
Hết duyệt	empty	3.84 7 1.5 – 1 2 + 5 3 + ^ ^ +

❖ Lập trình:

- Tham khảo:
  +) Trên Github: https://gist.github.com/mycodeschool/7867739
  +) Youtube: https://bom.to/3A7Mo9x
- Code:

```
1.  string InfixToPostfix(string s) {
2.      string postfix = "";
3.      stack<char> st;
4.
5.      for (int i = 0; i < s.length(); i++) {
6.          //if the scanned character is an operand
7.          while (IsOperand(s[i])) {
8.              postfix += s[i];
9.              i++;
10.         }
11.
12.         //If the scanned character is an operator
13.         if (IsOperator(s[i])) {
14.         //if the scanned character
15.           while (!st.empty() && !IsOpenParen(st.top()) && IsHigherPrecedence(st.top(), s[i]
    )) {
16.                 postfix += st.top();
17.                 st.pop();
18.             }
19.
20.         //now st.top has lower prec than scanned operator
21.         st.push(s[i]);
22.         }
23.
24.         //if the scanned character is an opening bracket
```

```
25.         else if (IsOpenParen(s[i]))
26.             st.push(s[i]);
27.
28. //if the scanned character is a closing bracket, append all the operators left in stack b/w
     open-close brackets into postfix
29.         else if (IsCloseParen(s[i])) {
30.             while (!st.empty() && !IsOpenParen(st.top())) {
31.                 postfix += st.top();
32.                 st.pop();
33.             }
34.             st.pop(); // remove opening bracket from stack
35.         }
36.     }
37.
38.     //if we get to the end of array but stack is not empty -
    > append all the operators left to postfix
39.     while (!st.empty()) {
40.         postfix += st.top();
41.         st.pop();
42.     }
43.
44.     return postfix;
45. }
```

3. Thuật toán chuyển biểu thức trung tố (infix expression) sang biểu thức tiền tố (prefix expression)

- Cấu trúc của 1 biểu thức tiền tố:
  <toán tử><toán hạng trái><toán hạng phải>
  ➔ 1 biểu thức trung tố muốn chuyển sang hậu tố thì phải tìm được 3 phần trên

❖ Thuật toán chuyển đổi từ biểu thức trung tố sang biểu thức tiền tố:

- Các bước thuật toán: đảo ngược chuỗi và sử dụng stack để lưu toán tử

- Bước 1: Đảo ngược biểu thức infix đầu vào và swap "dấu mở ngoặc" với "dấu đóng ngược tương ứng". Sau đó duyệt biểu thức (đã đảo ngược) từ trái sang phải.

- Bước 2: Kiểm tra phần tử được duyệt

+ Nếu phần tử đang được duyệt là toán hạng thì in vào chuỗi kết quả.

+ Nếu phần tử đang được duyệt là toán tử

- Nếu stack = empty thì push vào stack. Còn nếu độ ưu tiên của nó **lớn hơn hoặc bằng** stack.top thì push vào stack; ngược lại thì pop và print stack.top ra chuỗi kết quả và tiếp tục kiểm tra phần tử đang duyệt với stack.top.
- Nếu phần tử đang được duyệt và stack.top cùng là dấu mũ "^" thì pop & append stack.top vào chuỗi kết quả, tiếp tục kiểm tra stack.top với phần tử đang được duyệt (dấu mũ). Nếu stack.top không cùng độ ưu tiên với phần tử đang được duyệt thì push phần tử đang duyệt (là dấu '&') vào stack.

- Nếu stack.top là dấu đóng ngoặc thì tự động push phần tử đang xét vào stack

- Nếu phần tử đang được duyệt là dấu đóng ngoặc (do đảo ngược chuỗi gốc) thì push vào stack. Nếu phần tử đang được duyệt là dấu mở ngoặc thì pop stack.top cho đến khi tìm thấy dấu đóng ngoặc tương ứng

- Bước 3: Sau khi duyệt hết biểu thức infix, còn bao nhiêu toán tử trong stack thì append hết vào chuỗi kết quả. Lật ngược chuỗi kết quả sẽ cho ra biểu thức prefix.

VD: "4 / 2 / 1 + 2 ^ (7 + 2 ^ 3) ^ 5

- Bước 1: đảo chuỗi → "5 ^) 3 ^ 2 + 7(^ 2 + 1 / 2 / 4"
- Buốc 2: xử lý

Phần tử đang xét	Stack	Prefix
5		5
^	^	5
)	^)	5
3		5 3
^	^) ^	
2		5 3 2
+	^) + Do '+' có độ ưu tiên bé hơn stack.top = '^' nên pop stack.top và append nó vào prefix, đồng thời push '+' vào stack	5 3 2 ^
7		5 3 2 ^ 7
(	^ Do gặp dấu '(' nên pop tất cả toán tử trong stack tới khi gặp dấu ')'	5 3 2 ^ 7 +
^	^ tử đang được duyệt và stack.top cùng là dấu mũ "^" nên pop stack.top và append vào prefix	5 3 2 ^ 7 + ^
2		5 3 2 ^ 7 + ^ 2
+	+	5 3 2 ^ 7 + ^ 2 ^
1		5 3 2 ^ 7 + ^ 2 ^ 1
/	+ /	
2		5 3 2 ^ 7 + ^ 2 ^ 1 2
/	+ / /	
4		5 3 2 ^ 7 + ^ 2 ^ 1 2 4
Hết duyệt	Empty Trong stack còn bao nhiêu toán tử thì append hết vào prefix	5 3 2 ^ 7 + ^ 2 ^ 1 2 4 / / +

→ Chuỗi prefix (đảo ngược lại): "+ / / 4 2 1 ^ 2 ^ + 7 ^ 2 3 5"

- Tham khảo: https://bom.to/YvIIGE7

4. Thuật toán chuyển biểu thức tiền tố (prefix expression) sang biểu thức hậu tố (postfix expression)

❖ Thuật toán chuyển đổi từ biểu thức tiền tố sang biểu thức hậu tố:

Bước 1: Tạo 1 biến stack S, tạo 1 biến lưu kết quả postfix sau khi được convert

Bước 2: duyệt chuỗi perfix từ phải sang trái

LOOP (i = (prefix.length – 1) to 0)

Bước 2.1: Nếu gặp toán hạng

Push vào stack S

Bước 2.2: Nếu gặp toán tử

Bước 2.2.1: Pop ra 2 toán hạng trong stack (gọi lần lượt là op1 và op2)

Bước 2.2.2: Tạo biến tạm X = op1 + op2 + prefix[i];

Bước 2.2.3: Push X vào trong stack S

END LOOP

Bước 3: Pop và in ra toàn bộ biểu thức Postfix từ stack S

Ví dụ: / + 3 ^ 5 ^ 2 * 7 3 8

LOOP

➔ / + 3 ^ 5 ^ 2 X1 8 X1 = 7 3 *

➔ / + 3 ^ 5 X2 8 X2 = 2 X1 ^

➔ / + 3 X3 8 X3 = 5 X2 ^

➔ / X4 8 X4 = 3 X3 +

➔ X5 X5 = X4 8 /

END LOOP

POP AND PRINT postfix: 3 5 2 7 3 * ^ ^ + 8 /

- Tham khảo: https://bom.to/Htm7NE8

5. Thuật toán tính toán

- Ta xem mỗi con số và bên trong dấu ngoặc () [] {} là một đại lượng.
- Một biểu thức khi đó bắt đầu bằng một đại lượng, xem kẽ giữa hai đại lượng luôn là một trong những phép toán + - * / ^
- Xây dựng một struct Priority gồm:
  - Một biến int biểu thị tính ưu tiên cao nhất mà thuật toán hiện tại có.
  Qui ước: 2 cho ^, 1 cho * / và 0 cho + -
  - Một vector size = 3, chứa số lượng phép toán, phép toán có độ ưu tiên là 2 sẽ cộng vào Vector[2] của vector, tương tự cho 1 và 0.

- Xây dựng một hàm đọc biểu thức bao gồm.
  - struct Priority : Ưu tiên của thuật toán và lượng thuật toán còn lại
  - vector<double> : Chứa các số đọc vào
  - string : Chứa các phép toán đọc vào
- Bắt đầu đọc từ trái qua phải theo trình tự: đại lượng (Số và ngoặc) đến phép toán rồi lại đại lượng . . .
- Nếu gặp ngoặc, gọi đệ qui hàm đọc biểu thức để đọc biểu thức bên trong ngoặc và trả về double
- Sau khi đọc hết biểu thức ta có được:
  struct Priority, vector<double>, string
- (*) Xây dựng một hàm tính toán với biểu thức chỉ có số và phép toán nhận vào các tham số trên:

- Nếu Priority có độ ưu tiên là 2, tồn tại ^ trong string, cho biến chạy từ phải qua trái để phù hợp với thự tự tính toán.
- Nếu Priority có độ tiên thấp hơn 2, tồn tại + - * / trong string, cho biến chạy từ trái sang phải để phù hợp với thứ tự tính toán.
- Trong khi chạy trên string chưa các phép toán, nếu tại vị trí i có một phép toán có độ ưu tiên bằng với priority hiện tại thì ta sẽ thực hiện phép toán trên vector<double> chứa dấu theo cách:

→ Vector[i] = Vector[i] + Vector[i+1]

→ Xóa Vector[i+1], xóa thuật toán khỏi string

giảm 1 đơn vị của vector chứa số lượng thuật toán đó trong Priority, break ngay nếu số lượng thuật toán đó trong chuỗi còn bằng 0, giảm độ ưu tiên 1 đơn vị và chạy lại qui trình nếu độ ưu tiên còn hơn 0.

- Cứ như vậy cho đến khi vector<double> chỉ còn một số khi đó trả về số đó chính là kết quả của bài toán.

- VD: 3 ^ [4 * (5 * 6) + 7] + 8

  → Hàm chủ: Nhận vào 3 ^ [4 * (5 * 6) + 7] + 8

  Độ ưu tiên: 2

  Vector chứa số : 3 [Đệ qui hàm con 1] 8

  String phép toán: ^ +

  Tính toán bằng hàm tính toán (*) trả về kết quả bài toán.

  → Hàm con 1: Nhận vào 4 * (5 * 6) + 7

  Độ ưu tiên: 1

  Vector chứa số : 4 [Đệ qui hàm con 2] 7

  String phép toán: * +

  Tính toán bằng hàm tính toán (*) trả về cho [Đệ qui hàm con 1]

→ Hàm con 2: 5 * 6
Độ ưu tiên: 1
Vector chứa số : 5   6
String phép toán: *
Tính toán bằng hàm tính toán (*) trả về cho [Đệ qui hàm con 2]