# **MỤC LỤC**

[**MỤC LỤC** 1](#_Toc500794211)

[**TÌM HIỂU THUẬT TOÁN POSTFIX** 2](#_Toc500794212)

[**1.** **Lý do chọn đề tài** 2](#_Toc500794213)

[**2.** **Khái niệm** 2](#_Toc500794214)

[**3.** **Giải thích thuật toán** 2](#_Toc500794215)

* [**Ký pháp Ba Lan ngược, và ứng dụng** 3](#_Toc500794216)
* [**Phương pháp chuyển từ biểu thức trung tố sang hậu tố** 3](#_Toc500794217)

[**4.** **Ứng dụng stack vào Thuật toán** 5](#_Toc500794218)

[**5.** **Giải thích code và các bước thực thi** 5](#_Toc500794219)

[**6.** **Mức độ hoàn thành** 9](#_Toc500794226)

[**7.** **Khó khăn trong quá trình tìm hiểu** 9](#_Toc500794227)

[**8.** **Tài liệu tham khảo** 9](#_Toc500794228)

[**PHỤ LỤC** 10](#_Toc500794229)

* [**Mô tả mã giả của phương thức *postfix*** 10](#_Toc500794230)
* [**Mô tả mã giả của phương thức valueMath** 10](#_Toc500794231)

# **TÌM HIỂU THUẬT TOÁN POSTFIX**

## **Lý do chọn đề tài**

Các biểu thức đại số được sử dụng hằng ngày đều được biểu diễn dưới dạng trung tố (infix). Cách biểu diễn này rất dễ hiểu với con người vì hầu hết các toán tử (+, -, \*, /) đều là toán tử hai ngôi và chúng phân cách giữa hai toán hạng với nhau. Tuy nhiên đối với máy tính, để tính được giá trị của một biểu thức đại số theo dạng này không đơn giản như ta vẫn làm. Với các biểu thức toán học đơn giản (như a+b) thì bạn có thể tự làm bằng các phương pháp tách chuỗi “thủ công”. Nhưng để “giải quyết” các biểu thức có dấu ngoặc, ví dụ như (a+b)\*c + (d+e)\*f , thì các phương pháp tách chuỗi đơn giản đều không khả thi. Để khắc phục điều đó, máy tính cần chuyển cách biểu diễn các biểu thức đại số từ trung tố sang một dạng khác là tiền tố hoặc hậu tố. Và trong khuôn khỗ đồ án này chúng em sẽ tìm hiểu và trình bày cách thực thi một biểu thức bằng dạng hậu tố (postfix) còn được biết đến với tên Ký Pháp Nghịch Đảo Ba Lan (Reserve Polish Notation – RPN), một thuật toán “kinh điển” trong lĩnh vực trình biên dịch thông qua ngôn ngữ minh họa Java.

## **Khái niệm**

Prefix (biểu thức tiền tố) được biểu diễn bằng cách đặt các toán tử lên trước các toán hạng. Cách biểu diễn này còn được gọi là “ký pháp Ba Lan”. Tùy theo mức độ ưu tiên của các toán tử mà chúng sẽ được sắp xếp khác nhau.

Postfix (biểu thức hậu tố) ngược lại với cách Prefix, biểu thức hậu tố tức là các toán tử sẽ được đặt sau các toán hạng. Cách biểu diễn này được gọi là “ký pháp nghịch đảo Ba Lan”.

Ví dụ:

|  |  |  |
| --- | --- | --- |
| Infix | Prefix | Postfix |
| a+b | +xy | xy+ |
| a+b-c | -+abc | ab+c- |
| a+b\*c | +a\*bc | abc\*+ |
| a+(b-c) | +a-bc | abc-+ |

## **Giải thích thuật toán**

Để đơn giản cho việc minh họa, giả định rằng chuỗi biểu thức nhận được từ bàn phím chỉ bao gồm: các dấu mở ngoặc/đóng ngoặc; 4 toán tử cộng, trừ, nhân và chia (+, -, \*, /); các toán hạng đều chỉ là các con số nguyên từ 0 đến 9; không có bất kỳ khoảng trắng nào giữa các ký tự.

### **Ký pháp Ba Lan ngược, và ứng dụng**

Ký pháp nghịch đảo Ba Lan được phát minh vào khoảng giữa thập kỷ 1950 bởi Charles Hamblin - một triết học gia và khoa học gia máy tính người Úc - dựa theo công trình về ký pháp Ba Lan của nhà Toán học người Ba Lan Jan Łukasiewicz. Hamblin trình bày nghiên cứu của mình tại một hội nghị khoa học vào tháng 6 năm 1957 và chính thức công bố vào năm 1962.

Từ tên hậu tố cũng đoán ra là theo cách biểu diễn này, các toán tử sẽ được đặt sau các toán hạng. Cụ thể là biểu thức trung tố: 4+5 sẽ được biểu diễn thành 4 5 +.

Quá trình tính toán giá trị của biểu thức hậu tố khá tự nhiên đối với máy tính. Ý tưởng là đọc biểu thức từ trái sang phải, nếu gặp một toán hạng (con số hoặc biến) thì push toán hạng này vào ngăn xếp; nếu gặp toán tử, lấy hai toán hạng ra khỏi ngăn xếp (stack), tính kết quả, đẩy kết quả trở lại ngăn xếp. Khi quá trình kết thúc thì con số cuối cùng còn lại trong ngăn xếp chính là giá trị của biểu thức đó.

### **Phương pháp chuyển từ biểu thức trung tố sang hậu tố**

* Độ ưu tiên của các toán tử

Một trong những điều quan trọng trước khi bắt đầu là phải tính toán được độ ưu tiên của các toán tử trong biểu thức nhập vào. Để đơn giản chỉ xét các toán tử hai ngôi và thường dùng bao gồm: multiply (+),subtract (-), multiply (\*), divide (/),modulo (%). Theo đó các toán tử “\*, /, %” có cùng độ ưu tiên và cao hơn hai toán tử “+, -”. Như vậy phương thức lấy độ ưu tiên toán tử như sau:

1 Xác định mức ưu tiên toán tử nằm trong chuỗi biểu thức (toantu)

2 {

3 if (toantu := "\*" OR toantu := "/" OR toantu == "%")

4 return 2;

5 if (toantu := " " OR toantu == "-")

6 return 1;

7 return 0;

8 }

* Chuẩn hóa biểu thức Infix trước khi chuyển đổi

Các biểu thức Infix khi nhập vào có thể dư thừa các khoảng trắng, các kí tự không phù hợp hoặc viết sai cú pháp và cần bước chuẩn hóa để loại bỏ điều đó.

Ngoài ra còn phải ghép các chữ số liền nhau thành số (toán hạng), tách các toán tử, phân cách với nhau bằng một khoảng trắng. Các phần tử này được gọi là một token.

* Kiểm tra toán tử và toán hạng

Trong thuật toán chuyển đổi này, cần có các phương thức kiểm tra xem một thành phần của chuỗi có phải là toán tử hoặc toán hạng không. Thay vì sử dụng các cấu trúc if hoặc switch dài dòng và bất tiện khi phát triển, dùng Regex để kiểm tra.

Ngoài ra vì chuỗi nhập vào là một biểu thức đại số, nên các toán hạng sẽ xét không chỉ là các chữ số mà còn có chữ cái từ a-z và A-Z.

Có một quy tắc nữa là khi dùng chữ cái thì chỉ cho phép duy nhất một chữ cái đại diện cho một toán hạng, còn khi dùng chữ số thì có thể nhiều chữ số ghép thành một toán hạng. Ví dụ viết ab phải hiểu là a\*b, viết 123 không được hiểu là 1\*2\*3.

1 Chương\_trình\_con\_Nhận\_dạng\_toán\_tử\_tên\_là IsOperator(string str)

2 {

3 Trả\_về\_đúng\_khi\_str\_thuộc\_một\_trong\_(str, @" |-|\*|/|%");

4 }

5 Chương\_trình\_con\_nhận\_dạng\_toán\_hạng\_là\_chữ IsOperand(string str)

6 {

7 Trả\_về\_đúng\_khi\_str\_chứa\_ký\_tự\_đầu\_tiên\_trong\_(str, @"^d $|^([a-z]|[A-Z])$");

8 }

Ví dụ: biểu thức trung tố :

được biểu diễn lại dưới dạng hậu tố là (ta sẽ bàn về thuật toán chuyển đổi từ trung tố sang hậu tố sau):

5 1 2 + 4 \* + 3 +

|  |  |  |  |
| --- | --- | --- | --- |
| **Ký tự** | **Thao tác** | **Stack** | **Chuỗi hậu tố** |
| 3 | Ghi 3 vào kết quả |  | 3 |
| + | Push + | + |  |
| 4 | Ghi 4 vào kết quả |  | 3 4 |
| \* | Push \* | +\* |  |
| 2 | Ghi 2 vào kết quả |  | 3 4 2 |
| / | Lấy \* ra khỏi stack, ghi vào kết quả, push / | +/ | 3 4 2 \* |
| ( | Push ( | +/( | 3 4 2 \* |
| 1 | Ghi 1 vào kết quả | +/( | 3 4 2 \* 1 |
| - | Push - | +/(- | 3 4 2 \* 1 |
| 5 | Ghi 5 vào kết quả | +/(- | 3 4 2 \* 1 5 |
| ) | Pop cho đến khi lấy được (, ghi các toán tử pop được ra kết quả | +/ | 3 4 2 \*1 5 - |
| 2 | Ghi 2 ra kết quả | +/ | 3 4 2 \*1 5 - 2 |
|  | Pop tất cả các toán tử ra khỏi ngăn xếp và ghi vào kết quả |  | 3 4 2 \*1 5 – 2 / + |

*Bảng 1. Mô tả quá trình thuật toán diễn ra*

## **Ứng dụng stack vào Thuật toán**

* Lý do:

Việc dùng Stack phổ biến hơn có ưu điểm là dễ cài đặt, đơn giản còn dùng Expression Tree sẽ giúp việc chuyển đổi được dễ hiểu và trực quan hơn tuy nhiên lại mất thời gian cài đặt. Trong bài viết này sẽ trình bày kĩ thuật sử dụng Stack.

* Stack trong Java

Stack là một cấu trúc dữ liệu lưu trữ nhiều phần tử dữ liệu. Stack hoạt động theo cơ chế vào sau ra trước Last In/First Out (LIFO).

Trong Stack có các thao tác cơ bản:

+ Push : thêm 1 phần tử vào đỉnh Stack

+ Pop : lấy 1 phần tử từ đỉnh Stack

+ Peek: trả về phần tử đầu Stack mà không loại bỏ nó ra khỏi Stack

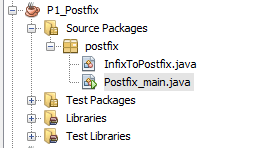
+ isEmpty: Kiểm tra Stack có rỗng không?

+ Search: trả về vị trí phần tử trong Stack tính từ đỉnh stack nếu ko thấy trả về -1.

## **Giải thích code và các bước thực thi**

Tiếp theo đây sẽ hiện thực lại thuật toán bằng ngôn ngữ Java với một số bước thực thi và giải thích chúng.

Trong ngôn ngữ Java hay một số ngôn ngữ khác cũng vậy, để không code không qua dài trong một class, nên impliment các method tại nhiều class để việc tìm kiểm và sửa lỗi được thuận tiện.

Vì vậy để thực hiện class *Postfix\_main.java* (đây là class chứa void main), trước tiên nên tạo một class mới *(InfixToPostfix.java)* và nó được chứa trong source *postfix*:

*Hình 1. Class InfixConverter.java chứa trong postfix*

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên lớp** | **Tên sinh viên thực hiện** | **Mục đích của lớp** |
| 1 | Postfix\_main | Phạm Thu Thảo, nguồn code từ trang web: <https://nguyenvanquan7826.com/2013/07/07/thuat-toan-java-chuyen-bieu-thuc-trung-to-sang-hau-to-java-converts-infix-to-postfix/> | * Cho phép nhập vào biểu thức cần chuyển đổi. * Gọi hàm tách biểu thức thành các phần tử từ class *InfixToPostfix.* * Gọi phương thức thực thi thuật toán được định nghĩa tại lớp InfixToPostfix. * Xuất trả về kết quả đã chuyển đổi biểu thức từ Infix sang Postfix. |
| 2 | InfixToPostfix | Phạm Thu Thảo, nguồn code từ trang web: <https://nguyenvanquan7826.com/2013/07/07/thuat-toan-java-chuyen-bieu-thuc-trung-to-sang-hau-to-java-converts-infix-to-postfix/> | Thực thi các phương thức hỗ trợ cho thuật toán chuyển biểu thức từ infix sang postfix:   * Phương thức *priority:* tính toán độ ưu tiên của toán tử * Phương thức *processString:* tách chuỗi biểu thức nhập vào thành các phần tử. * Phương thức *isOperator:* xét mỗi phần tử có phải toán tử không. * Phương thức *postfix:* thực hiện chuyển đổi biểu thức từ dạng infix sang dạng postfix. * Phương thức *valueMath:* thực hiện tính giá trị biểu thức khi ở dạng postfix. |

*Bảng 1. Mô tả các lớp trong chương trình*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STT** | **Tên phương thức** | **Mục đích của phương thức trong chương trình** | **Input** | **Output** | **Vị trí gọi phương thức** |
| 1 | *priority* | * **Thiết lập mức độ ưu tiên của toán tử.** * **Để đơn giản ta chỉ xét các toán tử hai ngôi và thường dùng bao gồm: multiply (+),subtract (-), multiply (\*), divide (/). Theo đó các toán tử “\*, /” có cùng độ ưu tiên và cao hơn hai toán tử “+, -”.** | Toán tử hoặc toán hạng cần xét (kiểu char) | -1: toán tử +, -.  - 2: toán tử \*,/.  -0: dấu ngoặc (,). | Phương thức được gọi trong hàm *postfix* tại class InfixToPostfix: *P1\_Postfix\src\postfix/* *InfixToPostfix* |
| 2 | *isOperator* | * Kiểm tra kí tự input có phải là một toản tử hay không, vì chuỗi nhập vào là một biểu thức đại số, nên các toán hạng sẽ xét không chỉ là các chữ số mà còn có chữ cái từ a-z và A-Z. | Toán tử hoặc toán hạng cần xét (kiểu char) | -true: nếu là toán tử  -false: nếu không là toán tử | Phương thức được gọi trong hàm *postfix* tại class InfixToPostfix: *P1\_Postfix\src\postfix/* *InfixToPostfix* |
| 3 | *processString* | Chuẩn hóa biểu thức Infix trước khi chuyển đổi, vì:   * Các biểu thức Infix khi nhập vào có thể dư thừa các khoảng trắng, các kí tự không phù hợp hoặc viết sai cú pháp. * Ngoài ra phải ghép các chữ số liền nhau thành số (toán hạng), tách các toán tử, phân cách với nhau bằng một khoảng trắng. Các phần tử này được gọi là một token. | Chuỗi nhập vào từ người dùng | Chuỗi đã được chuẩn hóa, nghĩa là mỗi phần tử cách nhau một khoảng trắng | Sau khi nhận được chuỗi của người dùng. |
| 4 | *postfix* | Đây là phương thức chính của chương trình. Phương thức này cho phép chuyển đổi dạng biểu thức từ infix sang postfix thông qua việc sử dụng các hàm kiểm tra và đặc biệt là stack.  (Mã giả được thể hiện tại phụ lục trang 11) | Chuỗi chữa phần tử đã được chuẩn hóa | Mảng các element sau khi đã được chuyển từ infix sang postfix. | Phương thức được gọi trong hàm *postfix* tại class InfixToPostfix: *P1\_Postfix\src\postfix/* *InfixToPostfix* |
| 4 | *valueMath* | Thực hiện tính giá trị biểu thức khi ở dạng postfix.  (Mã giả được thể hiện tại phụ lục trang 11) | Chuỗi biểu thức dạng postfix | Giá trị biểu thức | Phương thức được gọi trong hàm *postfix* tại class InfixToPostfix: *P1\_Postfix\src\postfix/* *InfixToPostfix* |

*Bảng 2. Giải thích các phương thức*

## **Mức độ hoàn thành**

* Trong phạm vi đề tài chỉ tìm hiểu được cách hoạt động của stack và ứng dụng vào đề tài của mình.
* Thực tế các biểu thức toán học rất phức tạp, nhưng mức độ tìm hiểu của đề tài chỉ dừng lại ở những phép toán cơ bản – cộng trừ, nhân chia.
* Đề tài hoàn thành đúng thời hạn.

## **Khó khăn trong quá trình tìm hiểu**

* Do khả năng có hạn, không thể tìm hiểu sâu một số thuật toán phức tạp.
* Những trang thông tin tiếng Việt quá sơ sài, phải tìm hiểu những trang nước ngoài và mất thời gian để dịch và hiểu.

## **Tài liệu tham khảo**

* Tài liệu từ Internet

<http://voer.edu.vn/c/danh-sach-tuyen-tinh-ngan-xep-stack/60bbf7d3/a208ce0f>

<https://nguyenvanquan7826.com/2013/07/07/thuat-toan-java-chuyen-bieu-thuc-trung-to-sang-hau-to-java-converts-infix-to-postfix/>

<http://khmt.123.st/t20-topic>

# **PHỤ LỤC**

## **Mô tả mã giả của phương thức *postfix***

BEGIN PROGRAM with elementMath

INIT EMPTY String s1

INIT EMPTY Stack S

LOOP element in elementMath

IF element not Operator:

s1 += element

END IF

ELSE:

IF element ==’(‘

Push element into S

END IF ELSE:

IF element ==’)’:

UNTIL S.Peek!=’(‘:

s1 += S.Pop()

S.Pop()

END IF

ELSE:

UNTIL S not EMPTY and S.Peek has Priovity bigger then element

s1+= S.Pop()

END UNTIL

S.Push(element)

END LOOP

UNTIL S not EMPTY:

s1 += S.Pop()

END UNTIL

RETURN s1

END PROGRAM

## **Mô tả mã giả của phương thức valueMath**

BEGIN PROGRAM with elementMath

INIT EMPTY STACK S

LOOP element in elementMath

IF not isOperator(element):

S.push(element);

END IF

ELSE

num = 0.0

num1 = S.pop();

num2 = S.pop();

SWITCH element:

CASE +: num = num2 + num1; BREAK;

CASE -: num = num2 – num1; BREAK;

CASE \*: num = num2 \* num1; BREAK;

CASE /: num = num2 / num1; BREAK;

END SWITCH

S.PUSH(num);

END ELSE

RETURN S.POP()

END PROGRAM