

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH**

**KHOA ĐÀO TẠO CHẤT LƯỢNG CAO**

**NGÀNH CÔNG NGHỆ THÔNG TIN**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



**BÁO CÁO ĐỒ ÁN** **1**

**ĐỀ TÀI**: ỨNG DỤNG PHÂN TÍCH VÀ TÍNH GIÁ TRỊ CỦA BIỂU THỨC SỬ DỤNG THUẬT TOÁN RPN

**SVTH 1 : Phạm Huỳnh Thanh Lâm**

**MSSV: 17110168**

**SVTH 2 : Chu Minh Hoàng**

**MSSV:17110139**

**TP. Hồ Chí Minh, tháng 11 năm 2019**

**MỤC LỤC**

[PHẦN MỞ ĐẦU 1](#_Toc23799947)

[1. Lý do chọn đề tài: 1](#_Toc23799948)

[2. Mục đích nghiên cứu đế tài: 1](#_Toc23799949)

[3. Yêu cầu : 1](#_Toc23799950)

[4. Phương pháp nghiên cứu: 1](#_Toc23799951)

[PHẦN NỘI DUNG 2](#_Toc23799952)

[I. Đặc tả 2](#_Toc23799953)

[II. Phân công 3](#_Toc23799954)

[III. Thiết kế 4](#_Toc23799955)

[1) Thuật toán 4](#_Toc23799956)

[2) Thiết kế lớp 6](#_Toc23799957)

[3) Thiết kế giao diện 13](#_Toc23799958)

[IV. Kiểm thử 14](#_Toc23799959)

[PHẦN KẾT LUẬN 20](#_Toc23799960)

[1. Kết quả đạt được 20](#_Toc23799961)

[2. Kết luận 20](#_Toc23799962)

[TÀI LIỆU THAM KHẢO 21](#_Toc23799963)

**DANH MỤC CÁC BẢNG**

[Bảng 1: Bảng phân công việc 4](#_Toc23800019)

[Bảng 2: Bảng các class có trong chương trình 7](#_Toc23800020)

[Bảng 3: Bảng thống kê các phương thức dùng trong class tinhtoan 8](#_Toc23800021)

[Bảng 4: Bảng thống kê các phương thức trong class Operator 13](#_Toc23800022)

**DANH MỤC CÁC HÌNH**

[Hình 1: Ví dụ chuyển infix sang postfix đơn giản (sử dụng biến x,y,z) 4](#_Toc23800062)

[Hình 2: Ví dụ chuyển đổi một biểu thức dựa theo thuật toán RPN 6](#_Toc23800063)

[Hình 3: Phương thức chuẩn hóa biểu thức 9](#_Toc23800064)

[Hình 4: Phương thức chuẩn hóa dấu cộng, trừ 10](#_Toc23800065)

[Hình 5: Phương thức chuyển đổi infix thành postfix (một phần code) 11](#_Toc23800066)

[Hình 6: Sử dụng phương thức so sánh hai operator 11](#_Toc23800067)

[Hình 7: Phương thức kiểm tra toán tử 12](#_Toc23800068)

[Hình 8: Tính toán chuỗi postfix 12](#_Toc23800069)

[Hình 9: Đưa ra độ ưu tiên của toán tử 13](#_Toc23800070)

[Hình 10: Giao diện màn hình 14](#_Toc23800071)

[Hình 11: Giao diện nhập biểu thức cần tính 14](#_Toc23800072)

[Hình 12: Kết quả của biểu thức sau khi thực hiện 14](#_Toc23800073)

[Hình 13: Kết quả sẽ ra null với kết quả nhập thiếu hay sai 14](#_Toc23800074)

[Hình 14 : Kết quả sẽ ra NULL nếu nhập chuỗi có hàm lượng giác nhưng thiếu dấu ngoặc 15](#_Toc23800075)

[Hình 15: Kết quả sẽ không tính được nếu vừa nhập biến vừa nhập số 15](#_Toc23800076)

[Hình 16: Chưa làm được hàm kiểm tra nhập liệu (nếu có các ký tự là vẫn chuyển đổi thành chuỗi postfix) 15](#_Toc23800077)

[Hình 17: Kết quả sẽ bằng Null nếu nhập liệu hàm lượng giác có chữ viết hoa đầu hoặc viết hoa toàn bộ 15](#_Toc23800078)

[Hình 18: Nhập liệu một biểu thức đơn giản và tính toán 16](#_Toc23800079)

[Hình 19: Nhập liệu một biểu thức phức tập gồm nhiều toán tử, kết quả cho ra một số vô tỷ 16](#_Toc23800080)

[Hình 20: Tính toán một hàm lượng giác cơ bản 16](#_Toc23800081)

[Hình 21: Tính toán hàm lượng giác sử dụng thêm +,-,\*,/ bên trong hàm lượng giác đó 17](#_Toc23800082)

[Hình 22: Tính toán lượng giác vs một số 17](#_Toc23800083)

[Hình 23: Thực hiện tính toán vừa có biến vừa có số 17](#_Toc23800084)

[Hình 24: Chuyển hóa chuỗi gồm hai biến a và b 18](#_Toc23800085)

[Hình 25: Cộng trừ nhiều biến 18](#_Toc23800086)

[Hình 26: Tính ln của 10 18](#_Toc23800087)

[Hình 27: Tính toán biểu thức ln(10)+1 18](#_Toc23800088)

[Hình 28: Tính căn của số 19](#_Toc23800089)

[Hình 29: Tính mũ của một số 19](#_Toc23800090)

[Hình 30: Thông báo khi một số chia cho 0 19](#_Toc23800091)

# PHẦN MỞ ĐẦU

## Lý do chọn đề tài:

Nhằm củng cố kiến thức đã học ở các môn học trước, lập trình hướng đối tượng, cấu trúc dữ liệu và giải thuật nhóm thực hiện đồ án phân tích và tính giá trị biểu thức bằng thuật toán RPN.

## Mục đích nghiên cứu đế tài:

Mục đích của đề tài là tạo ra một chương trình tính toán đơn giản thông qua ngôn ngữ Java và các tính chất của một đối tượng, đồng thời củng cố kiếm thức đã học để áp dụng vào thuật toán và tìm hiểu về ngôn ngữ mới (java).

## Yêu cầu :

Áp dụng được phương pháp chuyển từ trung tố sang hậu tố bằng thuật toán RPN. Thực hiện được các phép tính đơn giản của một biểu thức bình thường ở dạng infix với các kiến thức đã học của những môn học trước.

## Phương pháp nghiên cứu:

Nghiên cứu tham khảo tra cứu thông tin và tài liệu.

Hoàn thiện đề tài với sự hướng dẫn của giảng viên.

# PHẦN NỘI DUNG

## Đặc tả

* Chương trình phân tích biểu thức và tính giá trị biểu thức đó dựa trên chuổi postfix, giống như một biểu thức được thực hiện bằng việc tính tay theo quy luật toán học.
* Dữ liệu đầu vào là một biểu thức hay còn gọi là Infix (trung tố). Sau đó được chuyển sang chuỗi Postfix (hậu tố). Các số và phép tính nhập vào được gọi lần lượt là toán hạng và toán tử.
* Chức năng: Dùng để tính một biểu thức bất kì có giá trị phép tính cộng, trừ, nhân, chia (+, -, \*, /), hàm lượng giác, căn, mũ, các đại lượng ln, log, log10. Với các số thực là: các số tự nhiên (1, 2, …, 362), số âm (-1, -2, …), số hữu tỷ (1/2, 0.12, 1.233, ... ở đây số thập phân được viết có dấu ‘.’ thay cho dấu ‘,’), các biến (là các chữ cái thường). Đồng thời tính toán các phép tính trong ngoặc tròn.
* Phân tích hướng giải quyết thuật toán : Chia làm hai giai đoạn

a.Chuyển trung tố thành hậu tố: Có 1 stack (chỉ chứa toán tử) và 1 danh sách liên kết đơn

Vấn đề được đặt ra:

* Tại sao lại sử dụng danh sách liên kết đơn?

Do chưa biết người dùng sẽ nhập bao nhiêu ký tự và danh sách liên kết đơn giải quyết được vấn đề đó (Danh sách liên kết là một cấu trúc dữ liệu có thể giữ một số lượng phần tử tùy ý và dễ dàng thay đổi kích thước, cũng như dễ dàng bỏ đi các phần tử bên trong nó)

* Tại sao không sử dụng mảng ?

Trả lời: Ta vẫn có thể dùng mảng (phải biết trước kích thước hoặc quy định) bỏ qua điều ở trên thì việc sử dụng mảng hay danh sách liên kết đều khả thi đối với thuật toán RPN.

Ngoài ra lợi ích của việc sử dụng danh sách liên kết còn được thể hiện ở việc : khi sử dụng danh sách liên kết sẽ tối ưu được bộ nhớ. Vùng nhớ sẽ được cấp phát thêm khi cần chèn thêm phần tử mới, vùng nhớ được trống khi xóa phần tử. Trong bài toán chỉ cần một nơi lưu trữ cũng như là duyệt mảng để làm nên việc sử dụng là tối ưu hơn sử dụng mảng.

b.Tính trung tố: Có 1 stack (chỉ chứa toán hạng)

* Tính toán biểu thức tính cộng, trừ, nhân, chia, căn, mũ, lượng giác,..khi hoàn thành bước a.
* Xuất kết quả.
* Giao diện: Chương trình được chạy trên Console đơn giản

## Phân công

|  |  |  |  |
| --- | --- | --- | --- |
| TT | Tên SV | Công việc | Phẩn trăm đóng góp |
| 1 | Phạm Huỳnh Thanh Lâm | -Tìm hiểu thuật toán và ngôn ngữ java, xác định các đối tượng cần thực hiện  -Viết hàm chuyển infix thành postfix  -Viết class với đối tượng là toán tử dùng lưu toán tử đó và độ ưa tiên của nó  -Viết hàm chuẩn âm để xử lý các dấu cộng trừ liên tiếp nhau  -Viết hàm xử lý linklist postfix để tính giá trị biểu thức  - Kiểm tra, sửa lỗi, tinh chỉnh, bổ sung chương trình (nếu có)  -Viết báo cáo đồ án với Word | 50% |
| 2 | Chu Minh Hoàng | -Tìm hiểu thuật toán và ngôn ngữ java, xác định các đối tượng cần thực hiện  -Viết hàm xuất nhập biểu thức và đưa chuỗi biểu thức đó vào danh sách liên kết (biểu thức ở đây là infix)  -Viết hàm xét phần tử trong linklist postfix có phải toán tử không  -Xử lý hàm lượng giác trong chương trình  -Viết hàm xử lí linklist postfix để tính giá trị biểu thức  - Kiểm tra, sửa lỗi, tinh chỉnh, bổ sung chương trình (nếu có)  -Viết báo cáo đồ án với Word | 50% |

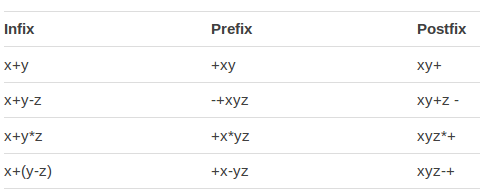
*Bảng 1: Bảng phân công việc*

## Thiết kế

### Thuật toán

Thuật toán được sử dụng cho bài toán được gọi là “ký pháp nghịch đảo Ba Lan” hoặc được viết tắt là RPN(Reverse Polish notation), được phát minh vào khoảng giữa thập kỷ 1950 bởi một triết học gia và nhà khoa học máy tính Charles Hamblin người Úc.

Dưới đây là ví dụ đơn giản cho thuật toán:

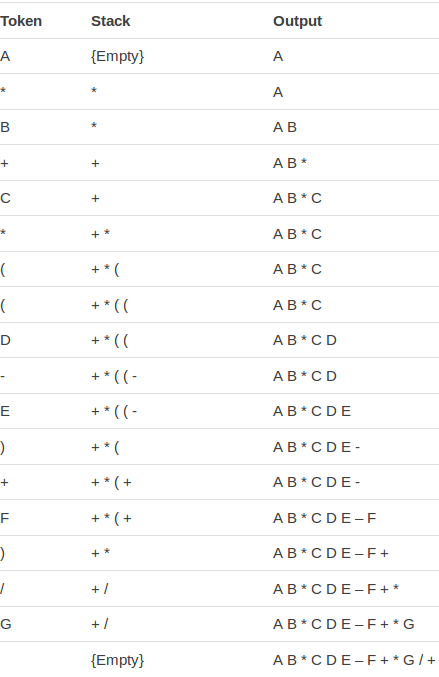


Hình 1: Ví dụ chuyển infix sang postfix đơn giản (sử dụng biến x,y,z)

Thuật toán sẽ thực hiện chuyển đổi một biểu thức infix thành postfix bằng cách đọc từng token trong biểu thức infix từ trái qua phải, với mỗi token ta thực hiện các bước sau:– Nếu là toán hạng: cho ra output.– Nếu là dấu mở ngoặc “(“: cho vào stack– Nếu là dấu đóng ngoặc “)”: lấy các toán tử trong stack ra và cho vào output cho đến khi gặp dấu mở ngoặc “(“. (Dấu mở ngoặc cũng phải được đưa ra khỏi stack)– Nếu là toán tử: +Chừng nào ở đỉnh stack là toán tử đồng thời toán tử đó có độ ưu tiên lớn hơn hoặc bằng toán tử hiện tại thì lấy toán tử đó ra khỏi stack và cho ra output. +Đưa toán tử hiện tại vào stack.

Sau khi duyệt hết biểu thức infix, nếu trong stack còn phần tử thì lấy các token trong đó ra và cho lần lượt vào output.

Để làm rõ thuật toán trên nhóm thực hiện đồ án tiến hành ví dụ đơn giản: Chuyển biểu thức A\*B+C\*((D-E)+F)/G từ dạng Infix sang dạng Postfix:



*Hình 2: Ví dụ chuyển đổi một biểu thức dựa theo thuật toán RPN*

### Thiết kế lớp

#### Danh mục các lớp

|  |  |  |  |
| --- | --- | --- | --- |
| TT | Tên lớp | Mục đích | SV phụ trách |
| 1 | Tinhtoan | Các phương thức để tính toán bài toán | Chu Minh Hoàng  Phạm Huỳnh Thanh Lâm |
| 2 | Operator | Đối tượng chứa các toán tử,độ ưa tiên và sự so sánh chúng với nhau | Phạm Huỳnh Thanh Lâm  Chu Minh Hoàng |
| 3 | Stack : st,sh  Giải thích: Stack được hỗ trợ trong java | Ngăn xếp dùng để chứa toán tử và toán hạng |  |

*Bảng 2: Bảng các class có trong chương trình*

Giải thêm về các class đã chọn và thực hiện được nêu ở bảng trên:

1.TinhToan

* Mục đích: Chứa các phương thức để tính toán và phân tích bài toán:
* Chuẩn âm.
* Chuẩn hóa.
* Chuyển từ infix sang postpix.
* Tính kết quả khi đã chuyển thành postfix.

2.Operater

* Đối tượng là operator chứa toán tử và độ ưu tiên của toán tử đó.
* Ngoài ra còn thực hiện so sánh các operator.

3.Stack

* Sử dụng để lưu trữ trong khi thực hiện quá trình chuyển đổi infix sang postfix. Stack ở đây lưu trữ mỗi phần tử vô là một mảng gồm 2 phần tử (toán tử, độ ưa tiên). Phần tử (toán tử, độ ưa tiên) được xét bên trong Operator, đưa mỗi operator được xét vào mảng gồm hai phần tử.
* Ngoài ra Stack còn được dùng để lưu toán hạng, để thực hiện các phép tính đưa ra kết quả cuối cùng.

#### Phương thức trong mỗi lớp

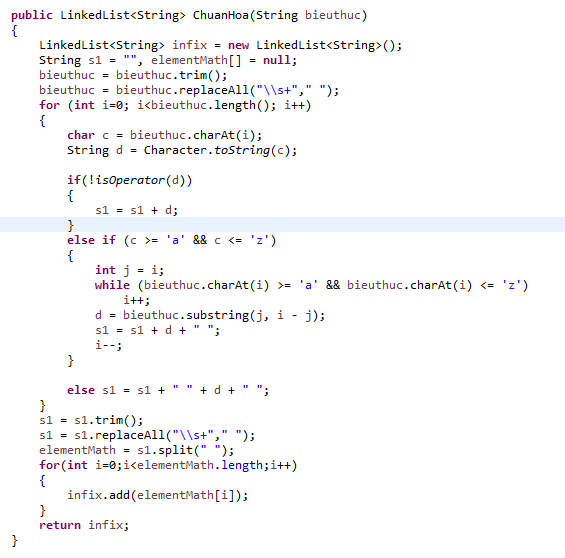
1. TinhToan.java

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TT | Phương thức | Mục đích | Tên file số thức tự dòng khai báo | SV phụ trách |
| 1 | **public** **boolean** isOperator(String c)  Input:Kí tự c  Ouput:True hoặc False  Giải thích: Sắp xếp chuỗi tăng dần với phương thức array.sort và array.binarySearch xuất vị trí của toán tử trong mảng | Kiểm tra xem kí tự c có phải là các toán tử "+", "-", "\*", "/", ")", "(" ,"sin","cos","tan","cotg","^","log","ln","sqrt" | Dòng 151 | Chu Minh Hoàng |
| 2 | **public** String Chuanam(String bieuthuc)  Input:String  Ouput:String  Giải thích: xử lý vấn đề về số âm  Ví dụ:-1+2; 1--2=>1+2; --1+2=>+1+2 | Chuẩn hoá số âm hoặc việc nhập nhiều dấu cộng trừ liên tiếp nhau | Dòng 158 | Phạm Huỳnh Thanh Lâm |
| 3 | **public** LinkedList<String> ChuanHoa(String bieuthuc)  Input:Chuỗi biểu thức <String>  Ouput:Danh sách liên kết chứa các phần tử của chuỗi trên | Chuẩn hoá đưa chuỗi lưu thành một danh sách liên kết đơn tương với từng phần tử toán hạng và toán tử | Dòng 118 | Chu Minh Hoàng |
| 4 | **public** Linked List<String> chuyendoi(LinkedList<String> elementMath)  Input: LinkedList <infix>  Ouput: LinkedList <postfix> | Chuyển từ infix sang postfix | Dòng 55 | Chu Minh Hoàng, Phạm Huỳnh Thanh Lâm |
| 5 | **public** String Xuatkq(LinkedList<String> doi)  Input: LinkedList  Ouput:String | Tính kết quả dựa trên chuỗi postfix và xuất kết quả đó ra màn hình | Dòng 7 | Chu Minh Hoàng  Phạm Huỳnh Thanh Lâm |

*Bảng 3: Bảng thống kê các phương thức dùng trong class tinhtoan*

Ở bảng phương thức tinhtoan gồm có các phương thức phục vụ cho việc chuyển đổi cũng như việc tính toán:

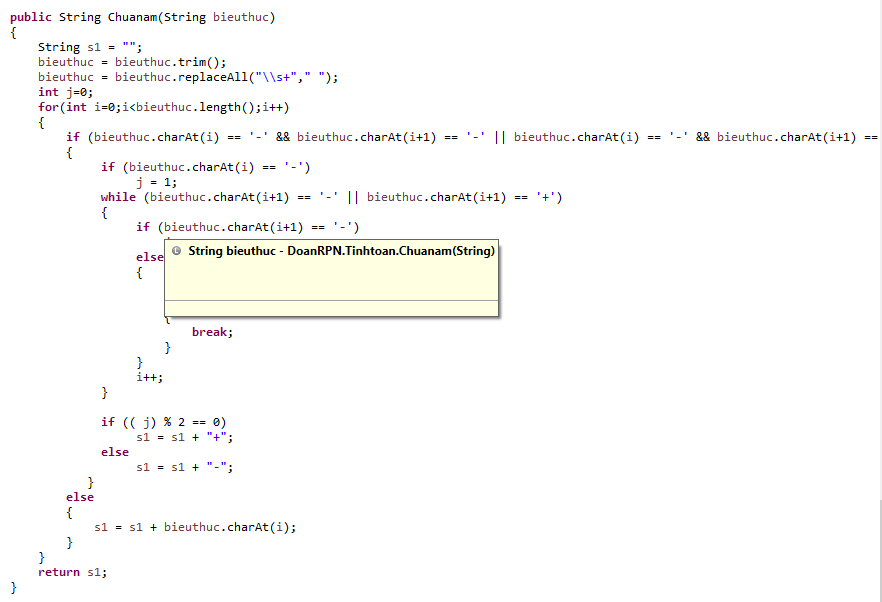
* Khi bắt đầu bài toán bắt buộc phải nhập một chuỗi để bài toán xử lý và để máy tính có thể hiểu được nhóm đồ án sử dụng phương thức: **public** LinkedList<String> ChuanHoa (String bieuthuc) để chuẩn hóa chuỗi, nhằm xóa đi các kí tự trắng không cần thiết khi nhập biểu thức, làm cho việc sử lý bài toán dễ dàng hơn.



*Hình 3: Phương thức chuẩn hóa biểu thức*

Nhóm đồ án chọn Linklist chọn để lưu dữ liệu, trong phương thức sẽ xóa dấu khoảng trắng xử lý các chữ số cũng như các biến (a,b,c) lưu vào trong linklist.

* Khó khăn gặp phải khi sử dụng phương thức chuẩn hóa là khi nhập chuỗi có các nhiều dấu cộng hay trừ thì sẽ chuyển hóa bị sai và xuất ra có thể không đúng với yêu cầu do vậy phương thức **public** String Chuanam (String bieuthuc) giải quyết vấn đề này.



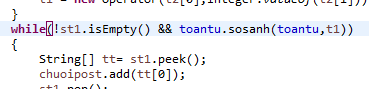
*Hình 4: Phương thức chuẩn hóa dấu cộng, trừ*

* Kết thúc phương thức chuẩn âm và chuẩn hóa ta được một chuỗi infix được lưu trong linklist, tiếp theo đến bước thực hiện chuyển infix thành postfix.



*Hình 5: Phương thức chuyển đổi infix thành postfix (một phần code)*

Ở phần này, hai đối tượng được sử dụng là Stack để lưu trữ toán tử và đối tượng Operator được sử dụng lưu trữ sẽ trả ra toán tử và độ ưu tiên, đồng thời so sánh độ ưu tiên đúng theo đối tượng ta nêu ở class Operator.



*Hình 6: Sử dụng phương thức so sánh hai operator*

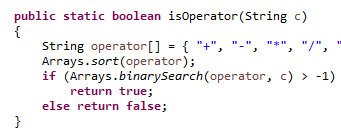
Xử lý độ ưu tiên của toán tử và việc thực hiện push, pop vào Stack trong bài toán đồng thời kết quả chuyển hóa được lưu vào một danh sách liên kết đơn.





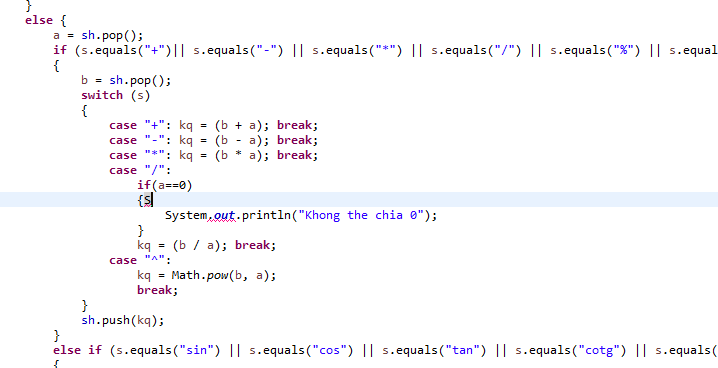


**public** **boolean** isOperator(String c) là một phương thức đóng vai trò vô cùng quan trọng trong bài toán, phương thức dùng để xác định trong phần tử trong danh sách liên kết là toán tử hay toán hạng được sử dụng xuyên suốt trong quá trình thực hiện bài toán.



*Hình 7: Phương thức kiểm tra toán tử*

* Khi đã chuyển thành postfix, bước tiếp theo là làm sao từ postfix ra được kết quả( tính toán từ chuỗi postfix)



*Hình 8: Tính toán chuỗi postfix*

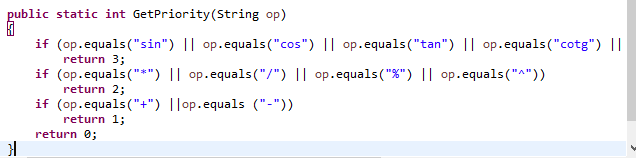
1. Operate.java

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TT | Phương thức | Mục đích | Tên file số thức tự dòng khai báo | SV phụ trách |
| 1 | **public char** getkitu()  **public int** getdouutien()  **public** Operator(**char** kitu, **int** douutien)  **public** Operator() | Khai báo operate gồm 2 thành phần toán tử và độ ưa tiên | Dòng 9 tới 29 | Phạm Huỳnh Thanh Lâm |
| 2 | **public void** Chuoi(**char** elementMath)  Input: toán tử | Đưa toán tử vào đối tượng operator | Dòng 31 | Phạm Huỳnh Thanh Lâm |
| 3 | **public** Character[] Xetdouutien(**char** elementMath)  Input: toán tử  Ouput: một mảng 2 phần tử toán tử và độ ưu tiên | Đưa toán tử và độ ưu tiên vào mảng để đưa vào stack | Dòng 38 | Chu Minh Hoàng |
| 4 | **public boolean** sosanh(Operator a, Operator b)  Input: Operate a và b  Ouput: True hay False | So sánh 2 toán tử với nhau coi toán tử a có “<=” toán tử b hay không | Dòng 47 | Phạm Huỳnh Thanh Lâm |
| 5 | **public static int** GetPriority(**char** op)  Input: toán tử char  Ouput: độ ưu tiên int | Gán độ ưu tiên cho toán tử | Dòng 54 | Phạm Huỳnh Thanh Lâm |

*Bảng 4: Bảng thống kê các phương thức trong class Operator*

Dưới đây là một vài giải thích cho các phương thức bên trong class Operator:

* Ở đây ta có phương thức **public boolean** sosanh (Operator a, Operator b) nhằm so sánh các operator là một đối tượng gồm toán tử và độ ưa tiên nhằm xác định đối tượng nào lớn nhất dựa vào độ ưa tiên.
* Ngoài ra ta còn có phương thức:

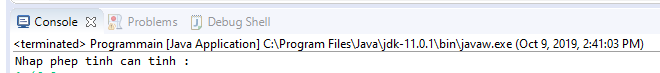


*Hình 9: Đưa ra độ ưu tiên của toán tử*

Ta dùng tính chất so sánh hai chuỗi của String trong java là phương thức equals, là phương thức được java hỗ trợ. Ở đây phương thức equals sẽ so sánh một chuỗi với một Object. Kết quả là true nếu và chỉ nếu tham số là không null và là một đối tượng String mà biểu diễn cùng dãy ký tự như đối tượng đó. Ngoài ở đây nếu ta dùng kí hiệu so sánh ‘= =’ sẽ làm cho phần tử op sẽ không nhận được bắt kì chuỗi cần so sánh.

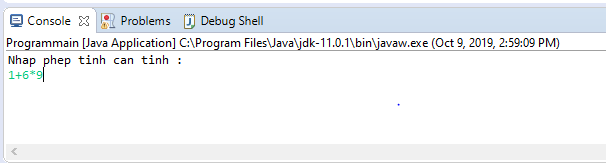
### Thiết kế giao diện

* Dòng in thông báo nhập biểu thức



Hình 10: Giao diện màn hình

* Dòng nhập biểu thức



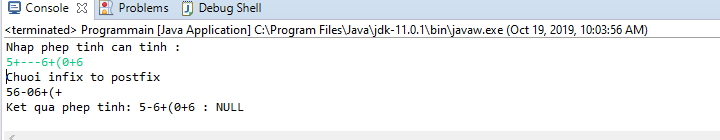
Hình 11: Giao diện nhập biểu thức cần tính

* Dòng in kết quả



Hình 12: Kết quả của biểu thức sau khi thực hiện

* Với kết quả sai

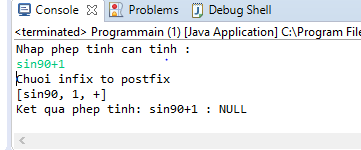


Hình 13: Kết quả sẽ ra null với kết quả nhập thiếu hay sai

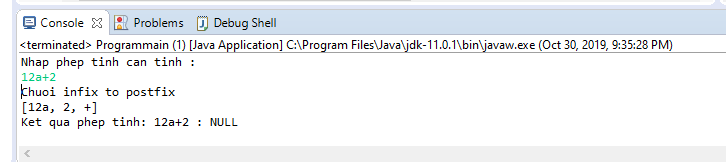
## Kiểm thử

Ở đây nhóm thực hiện đồ án xét hai trường hợp khi nhập đúng và khi nhập sai:

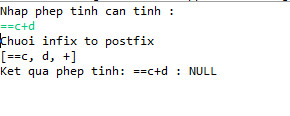
* Các trường hợp sai khi nhập liệu:



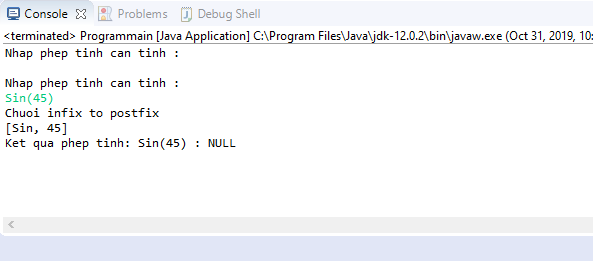
Hình 14 : Kết quả sẽ ra NULL nếu nhập chuỗi có hàm lượng giác nhưng thiếu dấu ngoặc



Hình 15: Kết quả sẽ không tính được nếu vừa nhập biến vừa nhập số

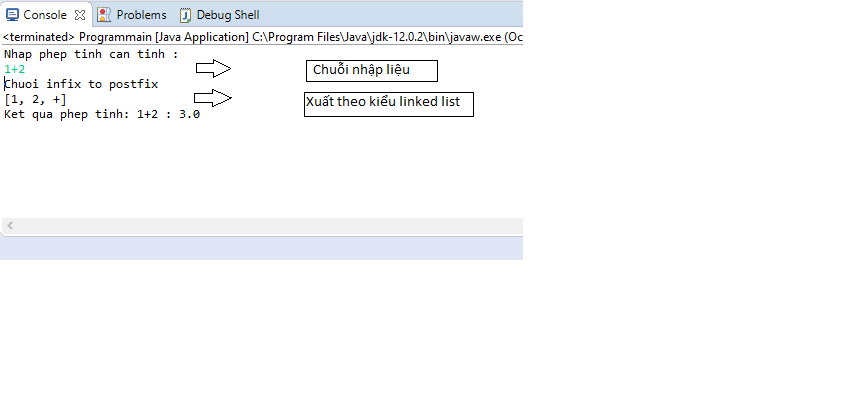


Hình 16: Chưa làm được hàm kiểm tra nhập liệu (nếu có các ký tự là vẫn chuyển đổi thành chuỗi postfix)

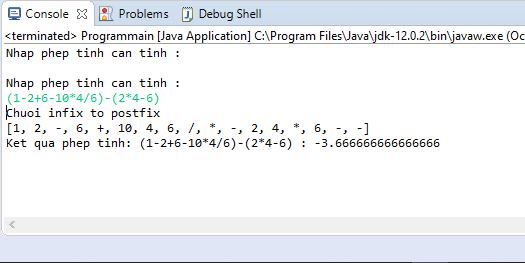


Hình 17: Kết quả sẽ bằng Null nếu nhập liệu hàm lượng giác có chữ viết hoa đầu hoặc viết hoa toàn bộ

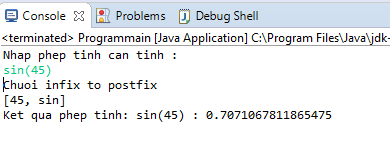
* Kiểm thử các trường hợp nhập đúng



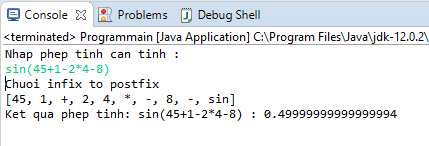
Hình 18: Nhập liệu một biểu thức đơn giản và tính toán



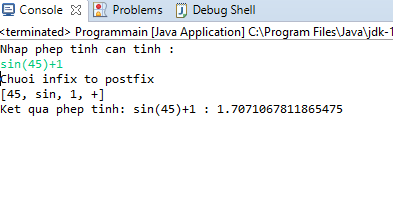
Hình 19: Nhập liệu một biểu thức phức tập gồm nhiều toán tử, kết quả cho ra một số vô tỷ



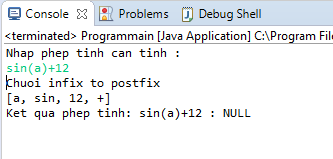
Hình 20: Tính toán một hàm lượng giác cơ bản



Hình 21: Tính toán hàm lượng giác sử dụng thêm +,-,\*,/ bên trong hàm lượng giác đó

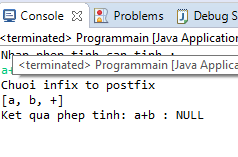


Hình 22: Tính toán lượng giác vs một số

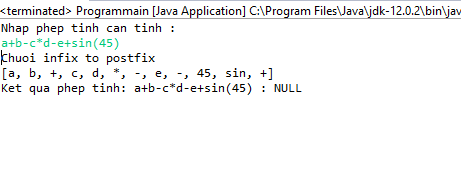


Hình 23: Thực hiện tính toán vừa có biến vừa có số

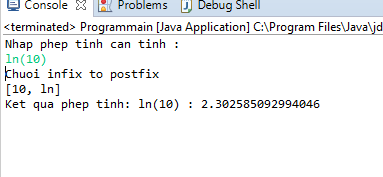
(biểu thức nhập không thể tính toán được, từ biểu thức trên chỉ có thể chuyển hóa thành chuỗi postfix)



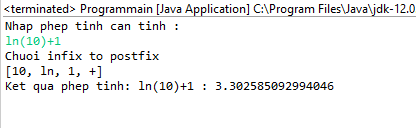
Hình 24: Chuyển hóa chuỗi gồm hai biến a và b



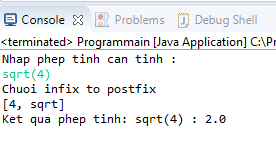
Hình 25: Cộng trừ nhiều biến



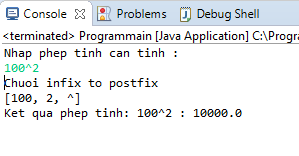
Hình 26: Tính ln của 10



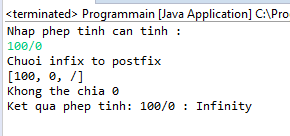
Hình 27: Tính toán biểu thức ln(10)+1



Hình 28: Tính căn của số



Hình 29: Tính mũ của một số



*Hình 30: Thông báo khi một số chia cho 0*

# 

# PHẦN KẾT LUẬN

## Kết quả đạt được

Sau quá trình thực hiện đồ án “Ứng dụng phân tích và tính giá trị biểu thức bằng thuật toán RPN” nhóm đã hoàn thành được các mục tiêu:

* Tính toán biểu đơn giản với các phép tính toán học.
* Hiểu rõ được phương pháp chuyển trung tố sang hậu tố (infix => postfix).

Tuy nhiên còn một vài chức năng về phần xuất nhập còn hạn chế, vẫn chưa tối ưu hóa hoàn toàn code

## Kết luận

1. **Ưu điểm**

Sau khi hoàn thành đồ án, nhóm thực hiện đồ án tự nhận thấy đã hoàn thành tốt phần nào những yêu cầu cầu của đề tài đã đặt ra, và cả những mục tiêu mà nhóm đã tự đặt ra:

* Có thể tính toán được các biểu thức khác nhau.
* Tính toán được các biểu thức với nhiều lớp căn một cách chính xác.
* Phân tích chuỗi biểu thức thành chuỗi postfix bất kì cả biên và số.

1. **Nhược điểm**

Tuy hoàn thành được đồ án nhưng chưa hoàn thiện vì còn một số sai sót:

* Không tính được hàm lượng giác nếu không có ngoặc và không tính được hàm lượng giác nếu viết hoa các chữ cái.
* Nếu nhập thiếu một dấu ngoặc hay sai, chương trình sẽ không tính được và ra Null.
* Không tính được nếu nhập sai chữ kí hiệu toán học.
* Giao diện kém bắt mắt, đơn giản, tính tương tác với người dùng không cao do người dùng chỉ có thể thao tác bằng phím.

# TÀI LIỆU THAM KHẢO

[1] <https://cachhoc.net/2013/07/07/thuat-toan-java-chuyen-bieu-thuc-trung-to-sang-hau-to-java-converts-infix-to-postfix/>

[2] <https://yinyangit.wordpress.com/2011/01/26/algorithm-chuy%E1%BB%83n-bi%E1%BB%83u-th%E1%BB%A9c-trung-t%E1%BB%91-sang-ti%E1%BB%81n-t%E1%BB%91-va-h%E1%BA%ADu-t%E1%BB%91-b%E1%BA%B1ng-stack/>