**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 : Huỳnh Anh Thế Vinh 17110255**

**SVTH 2 : Ngô Thị Trang 17110241**

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

# **MỤC LỤC**

**MỤC LỤC 1**

**PHẦN MỞ ĐẦU 1**

1. Lý do chọn đề tài: 1

2. Mục đích nghiên cứu đề tài: 1

3. Yêu cầu : 1

4. Phương pháp nghiên cứu: 1

**PHẦN NỘI DUNG 2**

I. Đặc tả: 2

II. Phân công: 2

III. Thiết kế: 2

*1. Tìm hiểu về thuật toán RPN:* 2

*2. Thiết kế lớp:* 4

3. *Thiết kế giao diện*: 19

IV. Cài đặt và kiểm thử: 20

**PHẦN KẾT LUẬN 24**

1. Kết quả đạt được: 24

2. Kết luận: 24

# **PHẦN MỞ ĐẦU**

## Lý do chọn đề tài:

Các biểu thức đại số sử dụng hằng ngày được biểu diễn dưới dạng trung tố (Infix). Tuy nhiên, để máy tính tính được giá trị của một biểu thức thì cần phải biểu diễn các biểu thức đại số từ trung tố sang một dạng khác là tiền tố hoặc hậu tố. Giúp tính toán biểu thức nhanh chóng, hiệu quả và chính xác.

## Mục đích nghiên cứu đề tài:

Mục đích của đề tài là tạo ra một chương trình đơn giản được viết bằng ngôn ngữ Java thông qua phần mềm lập trình Elipse Java 2019 để có thể tính toán các biếu thức từ đơn giản đến phức tạp một cách nhanh chóng và chính xác.

## Yêu cầu :

Dùng thuật toán RPN để tính được kết quả của biểu thức được nhập vào.

Áp dụng các kiến thức của các môn đã được học trước đó để hoàn thành đề tài.

## Phương pháp nghiên cứu:

Nghiên cứu, tham khảo và tra cứu thông tin trên các tài liệu sách, báo, internet… thông qua sự cho phép của người hướng dẫn.

Hoàn thiện đề tài trong sự hướng dẫn của giáo viên.

# **PHẦN NỘI DUNG**

1. **Đặc tả:**

- Dữ liệu đầu vào của chương trình là một biểu thức được gọi là Infix (trung tố). Các số và phép tính nhập vào được gọi lần lượt là toán hạng và toán tử. Sau khi biểu thức được nhập vào chương trình sẽ phân tách chuỗi biểu thức nhập vào thành các chuỗi con chứa mỗi toán hạng và toán tử, sau đó sử dụng LinkedList để lưu và thực hiện tính toán.

- Yêu cầu của đề tài là sử dụng thuật toán RPN để tính toán nên chương trình sẽ dùng Stack để lưu toán tử. Biểu thức Postfix (hậu tố) tương tự như Infix ta cũng sử dụng LinkedList để lưu trữ sau quá trình chuyển đổi từ trung tố sang hậu tố.

- Trong chương trình sẽ có các hàm, mỗi hàm sẽ thực hiện một nhiệm vụ khác nhau như: xét độ ưu tiên của toán tử, chuyển biểu thức từ trung tố sang hậu tố, chuyển biểu thức từ hậu tố sang trung tố, tính toán kết quả của biểu thức…

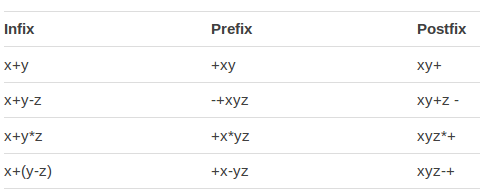
- Giao diện của chương trình được lập trình đơn giản thân thiện với người dùng.

1. **Phân công:**

|  |  |  |  |
| --- | --- | --- | --- |
| TT | Tên SV | Công việc | Phẩn trăm đóng góp |
| 1 | Huỳnh Anh Thế Vinh | Tìm hiểu thuật toán | 100% |
| 2 | Ngô Thị Trang | Tìm hiểu thuật toán, viết báo cáo | 100% |

1. **Thiết kế:**
2. ***Tìm hiểu về thuật toán RPN:***

**Reverse Polish notation** (RPN) - tạm dịch là ký hiệu Ba Lan ngược lại - là một ký hiệu toán học trong đó dấu đi theo toán hạng, giảm thiểu sử dụng bộ nhớ của máy tính và dùng ngăn xếp (*stack*) để tính những bài toán , được phát minh vào khoảng giữa thập kỷ 1950 bởi một triết học gia và nhà khoa học máy tính Charles Hamblin người Úc.

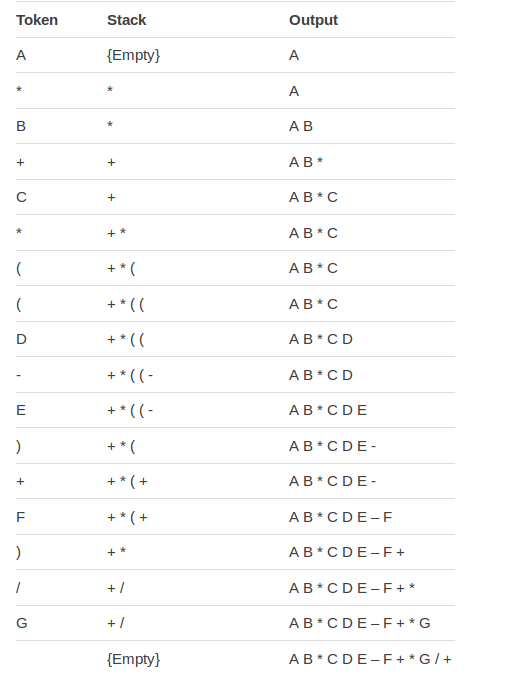


1. Ví dụ về thuật toán

**Thuật toán để chuyển một biểu thức Infix sang dạn Prefix:**

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 và 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.

VD: Chuyển biểu thức A\*B+C\*((D-E)+F)/G từ dạng Infix sang dạng Postfix



1. Ví dụ các bước thuật toán
2. ***Thiết kế lớp:***
3. Danh mục các lớp:

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Tên lớp | Mục đích | SV phụ trách |
| 1 | Operand | Lưa giá trị toán hạng và tính toán các phép toán 1 ngôi và 2 ngôi |  |
| 2 | InFix | Chuẩn hóa biểu thức Infix, chuyển biểu thức trung tố sang hậu tố. |  |
| 3 | PostFix | Chuẩn hóa biểu thức Postfix, chuyển biểu thức hậu tố sang trung tố. |  |
| 4 | Program | Thiết kế giao diện cho người dùng nhập biểu thức cần tính toán. |  |

1. Phương thức trong mỗi lớp:
2. Operand.java

|  |  |  |
| --- | --- | --- |
| STT | Phương thức | Mục đích |
| 1 | **private** String value; | Thuộc tính giá trị toán hạng. |
| 2 | **public** String getValue() {    **return** value;  } | Lấy giá trị toán hạng. |
| 3 | **public** **void** setValue(String value) {    **this**.value = value;  } | Gán giá trị toán hạng. |
| 4 | **public** Operand(String value) {    **this**.value = value;  } | Gọi toán hạng với giá trị tùy ý. |
| 5 | **public** **static** String evaluate(Operand o1, Operand o2, String operator)   * Input: 2 toán hạng và 1 toán tử * Output: Kết quả phép toán * Pseudo code:   //Phép toán 2 ngôi  **public** **static** String evaluate(Operand o1, Operand o2, String operator) {    **double** a,b;  **try** {    a = Double.*parseDouble*(o1.getValue());  b = Double.*parseDouble*(o2.getValue());  }**catch**(Exception e) {**return** "NaN";}    **double** result = 0;  **switch** (operator) {    **case** "+": result = a+b; **break**;  **case** "-": result = a-b; **break**;  **case** "\*": result = a\*b; **break**;  **case** "/": result = a/b; **break**;  **case** "^": result = Math.*pow*(a,b); **break**;  }  **return** Double.*toString*(result);  } | Thực hiện phép toán 2 ngôi. |
| 6 | **public** **static** String evaluate(Operand o, String operator)   * Input: 1 toán hạng và 1 toán tử * Output: Kết quả phép toán * Pseudo code:   //Phép toán 1 ngôi  **public** **static** String evaluate(Operand o, String operator) {  **double** a;  **try** {  a = Double.*parseDouble*(o.getValue());  }**catch**(Exception e) {**return** "NaN";}    **double** result = 0;  **switch** (operator) {    **case** "abs": result = Math.*abs*(a); **break**;  **case** "sqrt": result = Math.*sqrt*(a); **break**;  **case** "sin": result = Math.*sin*(a); **break**;  **case** "cos": result = Math.*cos*(a); **break**;  **case** "tan": result = Math.*tan*(a); **break**;  **case** "ln": result = Math.*log*(a); **break**;  **case** "log": result = Math.*log10*(a); **break**;  **case** "exp": result = Math.*exp*(a); **break**;  }  **return** Double.*toString*(result);  } | Thực hiện phép toán 1 ngôi. |

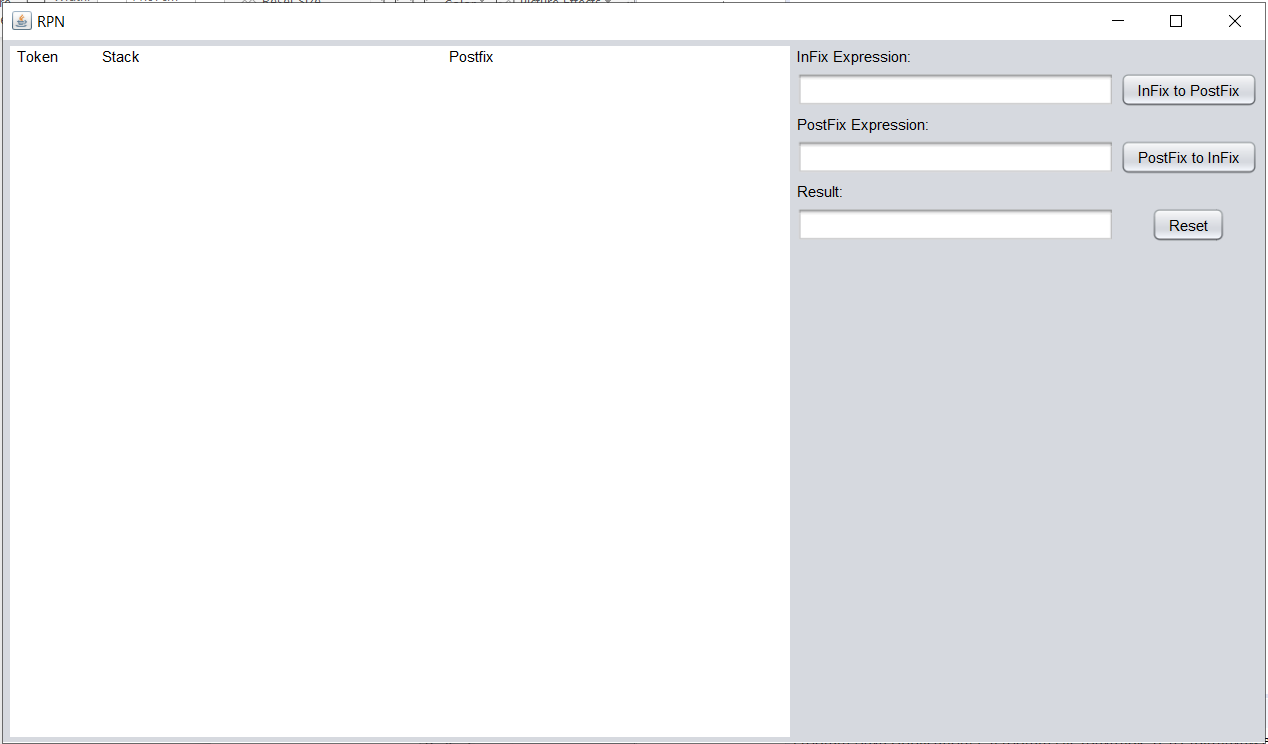
1. InFix.java

|  |  |  |
| --- | --- | --- |
| STT | Phương thức | Mục đích |
| 1 | **private** String str; | Thuộc tính str sử dụng String lưu infix với kiểu chuỗi. |
| 2 | **private** LinkedList<String> tokens; | Thuộc tính tokens sử dụng LinkedList lưu infix với kiểu List. |
| 3 | **public** String getStr() {    **return** str;  } | Lấy ra chuỗi infix. |
| 4 | **public** **void** setStr(String str) {    **this**.str = infixToStandard(str);  **this**.tokens = infixToTokens(**this**.str);  } | Gán chuỗi String cho infix |
| 5 | **public** LinkedList<String> getTokens() {    **return** tokens;  } | Lấy ra List các token của infix. |
| 6 | **public** InFix() {}  **public** InFix(String str) {    **this**.str = infixToStandard(str);  **this**.tokens = infixToTokens(**this**.str);  }  **public** InFix(PostFix postfix) {    String Str = "";  **for**(**int** i=0; i<postfixToInfix(postfix).size(); i++)  Str += postfixToInfix(postfix).get(i);  **this**.str = Str;  **this**.tokens = postfixToInfix(postfix);  } | Các kiểu truyền vào infix:  - Truyền vào rỗng.  - Truyền vào chuỗi.  - Truyền vào biểu thức hậu tố postfix.. |
| 7 | **private** String infixToStandard(String str)   * Input: Chuỗi * Output: Chuỗi được chuẩn hóa * Pseudo code:   //Chuẩn hóa chuỗi  **private** String infixToStandard(String str) {    str = str.replace(" ", "");  **for**(**int** i=0; i<str.length(); i++) {    str = str.replace("++", "+");  str = str.replace("+-", "-");  str = str.replace("-+", "-");  str = str.replace("--", "+");  str = str.replace("(+", "(");  str = str.replace("(-(", "((-1)\*(");  str = str.replace("(-abs", "((-1)\*abs");  str = str.replace("(-sqrt", "((-1)\*sqrt");  str = str.replace("(-sin", "((-1)\*sin");  str = str.replace("(-cos", "((-1)\*cos");  str = str.replace("(-tan", "((-1)\*tan");  str = str.replace("(-ln", "((-1)\*ln");  str = str.replace("(-log", "((-1)\*ln");  str = str.replace("(-exp", "((-1)\*exp");  }  **if**(str.charAt(0)=='+')  str = str.substring(1);  **else** **if**(str.charAt(0)=='-')  **if**( str.length()>1 && str.substring(1, 2).equals("(") ||  str.length()>3 && str.substring(1, 4).equals("abs") ||  str.length()>4 && str.substring(1, 5).equals("sqrt") ||  str.length()>3 && str.substring(1, 4).equals("cos") ||  str.length()>3 && str.substring(1, 4).equals("tan") ||  str.length()>3 && str.substring(1, 4).equals("abs") ||  str.length()>2 && str.substring(1, 3).equals("ln") ||  str.length()>3 && str.substring(1, 4).equals("log") ||  str.length()>3 && str.substring(1, 4).equals("exp") )  str = "(-1)\*" + str.substring(1);  **return** str;  } | Chuẩn hóa chuỗi:  - Loại bỏ tất cả khoảng trắng.  - Loại bỏ các toán tử dư thừa ++,--,+-,-+.  - Thay thế dấu âm thành (-1)\* |
| 8 | **private** **boolean** isOperator(String s)   * Input: Chuỗi * Output: True or False * Pseudo code:   //Kiểm tra kí tự toán tử  **private** **boolean** isOperator(String s) {    **return** precedence(s) > 0;  } | Kiểm tra kí tự toán tử. |
| 9 | **private** **int** precedence(String s)   * Input: Chuỗi * Output: Độ ưu tiên của toán tử * Pseudo code:   //Xét độ ưu tiên toán tử  **private** **int** precedence(String s) {  **if** (s.equals("(") || s.equals(")")) **return** 1;  **else** **if** (s.equals("+") || s.equals("-")) **return** 2;  **else** **if** (s.equals("\*") || s.equals("/") || s.equals("^")) **return** 3;  **else** **if** (s.equals("abs") ||  s.equals("sqrt") ||  s.equals("sin") || s.equals("cos") || s.equals("tan") ||  s.equals("ln") || s.equals("log") ||  s.equals("exp")) **return** 4;  **else** **return** 0;  } | Gán độ ưu tiên cho toán tử.  (, ) 🡪 1  +, - 🡪 2  \*, /, ^ 🡪 3  abs, sqrt, sin, cos, tan, ln, exp 🡪 4 |
| 10 | **private** LinkedList<String> infixToTokens(String str)   * Input: Chuỗi * Output: LinkedList với các phần tử là các token * Pseudo code:   //Sử dụng LinkedList để lưu từng toán tử và toán hạng thành các token  **private** LinkedList<String> infixToTokens(String str) {    LinkedList<String> tokens = **new** LinkedList<String>();  String tempStr = "";  **for**(**int** i=0; i<str.length(); i++) {  //Kiểm tra phát hiện toán tử 1 ngôi  String s = "";  **if**(i < str.length()-3 && str.substring(i, i+3).equals("abs")) {  //i nhỏ hơn str.length()-3 vì chuỗi "abs" kiểm tra có 3 kí tự tránh lỗi từ str.substring()  s += "abs";  i=i+2;  }  **else** **if**(i < str.length()-4 && str.substring(i, i+4).equals("sqrt")) {    s += "sqrt";  i=i+3;  }  **else** **if**(i < str.length()-3 && str.substring(i, i+3).equals("sin")) {    s += "sin";  i=i+2;  }  **else** **if**(i < str.length()-3 && str.substring(i, i+3).equals("cos")) {    s += "cos";  i=i+2;  }  **else** **if**(i < str.length()-3 && str.substring(i, i+3).equals("tan")) {    s += "tan";  i=i+2;  }  **else** **if**(i < str.length()-2 && str.substring(i, i+2).equals("ln")) {    s += "ln";  i=i+1;  }  **else** **if**(i < str.length()-3 && str.substring(i, i+3).equals("log")) {    s += "log";  i=i+2;  }  **else** **if**(i < str.length()-3 && str.substring(i, i+3).equals("exp")) {    s += "exp";  i=i+2;  }  **else** s += str.charAt(i);  //Kiểm tra nếu là toán hạng thì thêm vào chuỗi tạm thời  **if**(!isOperator(s) || s.equals("-") && i==0 || s.equals("-") && str.charAt(i-1)=='(') {    tempStr += s;  }  //Nếu là toán tử thì  **else** {  //Kiểm tra chuỗi tạm thời tồn tại toán hạng thì thêm vào List  **if**(!tempStr.isEmpty()) {    tokens.add(tempStr);  }  tempStr = "";  //Thêm toán tử vào List  tokens.add(s);  }  //Thêm toán hạng cuối vào List  **if**(i==str.length()-1)  **if** (!tempStr.isEmpty())  tokens.add(tempStr);  }  **return** tokens;  } | Chuyển đổi chuỗi infix lưu thành các token trong LinkedList. |
| 11 | **private** LinkedList<String> postfixToInfix(PostFix postfix)   * Input: Biểu thức postfix * Output: Biểu thức infix lưu trong LinkedList * Pseudo code:   //Sử dụng LinkedList để lưu infix sau khi chuyển từ postfix  **private** LinkedList<String> postfixToInfix(PostFix postfix) {    LinkedList<String> infix = **new** LinkedList<String>();  String tempStr = "";  **for** (**int** i = 0; i < postfix.getTokens().size(); i++) {    tempStr = postfix.getTokens().get(i);  //Nếu là toán hạng thì thêm vào List  **if** (precedence(tempStr) == 0) {    infix.add(tempStr);  }  //Trường hợp toán tử 1 ngôi  **else** **if** (precedence(tempStr) == 4) {    String b ="" + infix.getLast(); infix.removeLast();  infix.add(tempStr + "(" + b + ")");  }  //Trường hợp toán tử 2 ngôi  **else** {    String b ="" + infix.getLast(); infix.removeLast();  String a ="" + infix.getLast(); infix.removeLast();  infix.add("(" + a + tempStr + b + ")");  }  tempStr = "";  }  **return** infix;  } | Chuyển biểu thức từ hậu tố (postfix) sang trung tố (infix). |

1. PostFix.java

|  |  |  |
| --- | --- | --- |
| STT | Phương thức | Mục đích |
| 1 | **private** String str; | Thuộc tính str sử dụng String lưu postfix với kiểu chuỗi. |
| 2 | **private** LinkedList<String> tokens; | Thuộc tính tokens sử dụng LinkedList lưu postfix với kiểu List. |
| 3 | **private** String result; | Thuộc tính kết quả biểu thức result. |
| 4 | **public** String getStr() {    **return** str;  } | Lấy ra chuỗi postfix. |
| 5 | **public** **void** setStr(String str) {    **this**.str = str;  **this**.tokens = postfixToTokens(str);  **this**.result = result();  } | Gán chuỗi String cho postfix. |
| 6 | **public** LinkedList<String> getTokens() {    **return** tokens;  } | Lấy ra List các token của postfix. |
| 7 | **public** String getResult() {    **return** result;  } | Lấy ra kết quả biểu thức postfix. |
| 8 | **public** PostFix() {}  **public** PostFix(String str) {    **this**.str = str;  **this**.tokens = postfixToTokens(str);  **this**.result = result();  }  **public** PostFix(InFix infix) {    String Str = "";  **for**(**int** i=0; i<infixToPostfix(infix).size(); i++)  Str += infixToPostfix(infix).get(i) + " ";  **this**.str = Str;  **this**.tokens = infixToPostfix(infix);  **this**.result = result();  } | Các kiểu truyền vào postfix:  - Truyền vào rỗng.  - Truyền vào chuỗi.  - Truyền vào biểu thức trung tố infix.. |
| 9 | **private** **boolean** isOperator(String s) | Tương tự phương thức isOperator của class InFix |
| 10 | **private** **int** precedence(String s) | Tương tự phương thức precedence của class InFix |
| 11 | **private** LinkedList<String> postfixToTokens(String str)   * Input: Chuỗi * Output: LinkedList chứa các token của postfix * Pseudo code:   //Sử dụng LinkedList để lưu từng toán tử và toán hạng thành các token  **private** LinkedList<String> postfixToTokens(String str) {    LinkedList<String> tokens = **new** LinkedList<String>();  String tempStr = "";  //Các token postfix cách nhau bởi khoảng trắng  **for**(**int** i=0; i<str.length(); i++) {    **char** c = str.charAt(i);    **if** (c != ' ') {    tempStr += c;  }  **else** {    **if**(!tempStr.isEmpty()) {    tokens.add(tempStr);  }  tempStr = "";  }  **if**(i==str.length()-1)  **if** (!tempStr.isEmpty())  tokens.add(tempStr);  }  **return** tokens;  } | Chuyển đổi chuỗi postfix lưu thành các token trong LinkedList. |
| 12 | **private** LinkedList<String> infixToPostfix(InFix infix)   * Input: Biểu thức infix * Output: Biểu thức postfix lưu thành LinkedList * Pseudo code:   //Sử dụng LinkedList để lưu postfix sau khi chuyển từ infix  **private** LinkedList<String> infixToPostfix(InFix infix) {    LinkedList<String> postfix = **new** LinkedList<String>();  Stack<String> operator = **new** Stack<String>();  String popped;  **for** (**int** i = 0; i < infix.getTokens().size(); i++) {  String s = infix.getTokens().get(i);  //Nếu là toán hạng thì thêm vào cuối postfix  **if** (!isOperator(s))  postfix.add(s);  //Nếu là dấu đóng ngoặc “)”:  //lấy các toán tử trong stack ra và cho vào cuối postfix cho đến khi gặp dấu mở ngoặc “(“  //(Dấu đóng ngoặc cũng phải được đưa ra khỏi stack)  **else** **if** (s.equals(")"))  **while** (!(popped = operator.pop()).equals("("))  postfix.add(popped);  //Nếu là toán tử:  //-Chừng nào ở đỉnh stack là toán tử đó có độ ưu tiên lớn hơn toán tử hiện tại thì lấy toán tử đó ra khỏi stack và cho ra postfix  //-Đưa toán tử hiện tại vào stack  **else** {  **while** (!operator.isEmpty() && !s.equals("(") && precedence(operator.peek()) >= precedence(s))  postfix.add(operator.pop());  operator.push(s);  }  }  //Nếu trong stack còn toán tử thì lấy các token trong đó ra và cho lần lượt vào postfix  **while** (!operator.isEmpty())  postfix.add(operator.pop());  **return** postfix;  } | Chuyển biểu thức từ trung tố (infix) sang hậu tố (postfix). |
| 13 | **private** String result()   * Output: Kết quả biểu thức postfix * Pseudo code:   **private** String result() {    LinkedList<String> result = **new** LinkedList<String>();  **for** (**int** i = 0; i < tokens.size(); i++) {    String s = tokens.get(i);  //Nếu toán tử là phép toán 2 ngôi  **if** (precedence(s) == 2 || precedence(s) == 3) {    **if**(result.size()<2) **break**;  Operand b = **new** Operand(result.getLast());  result.removeLast();  Operand a = **new** Operand(result.getLast());  result.removeLast();  result.add(Operand.*evaluate*(a, b, s));  }  //Nếu toán tử là phép toán 1 ngôi  **else** **if** (precedence(s) == 4) {    **if**(result.size()<1) **break**;  Operand b = **new** Operand(result.getLast());  result.removeLast();  result.add(Operand.*evaluate*(b, s));  }  **else** result.add(s);  }  **try** {    **return** result.getLast();  }**catch**(Exception e) {**return** "NaN";}  } | Kết quả biểu thức postfix. |

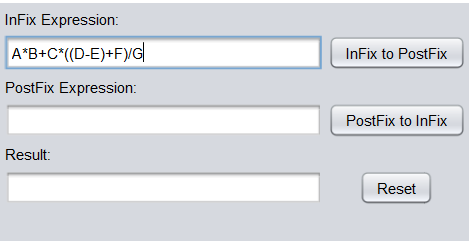
1. ***Thiết kế giao diện***:



1. *Giao diện chương trình*

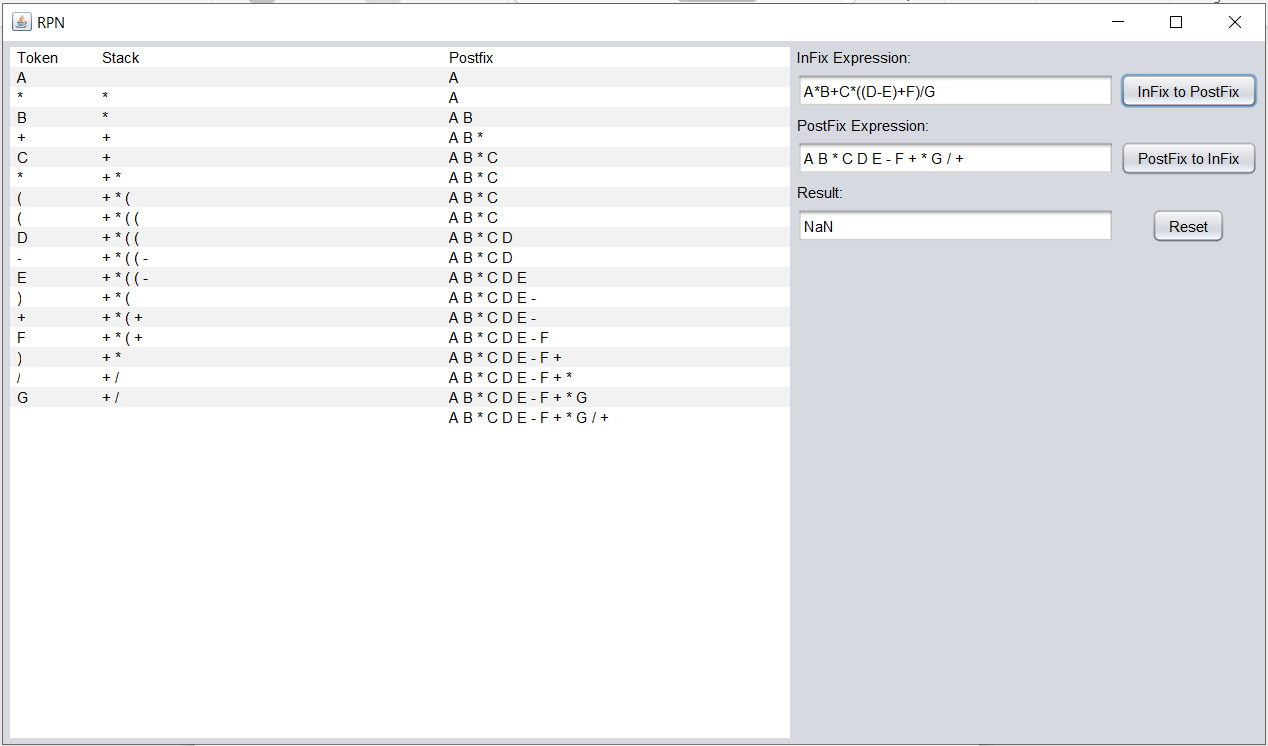
* InFix Expression: để nhập biểu thức trung tố cần tính.
* PostFix Expression: để nhập biểu thức hậu tố cần tính.
* Result: trả kết quả của phép tính.
* Table bảng: để hiện thị stack theo từng bước chạy.

1. **Cài đặt và kiểm thử:**
2. Nhập vào chuỗi Infix với toán hạng là các chữ cái

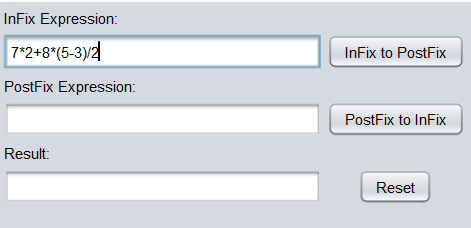


1. *Tình huống 1*

* Kết quả trả về:

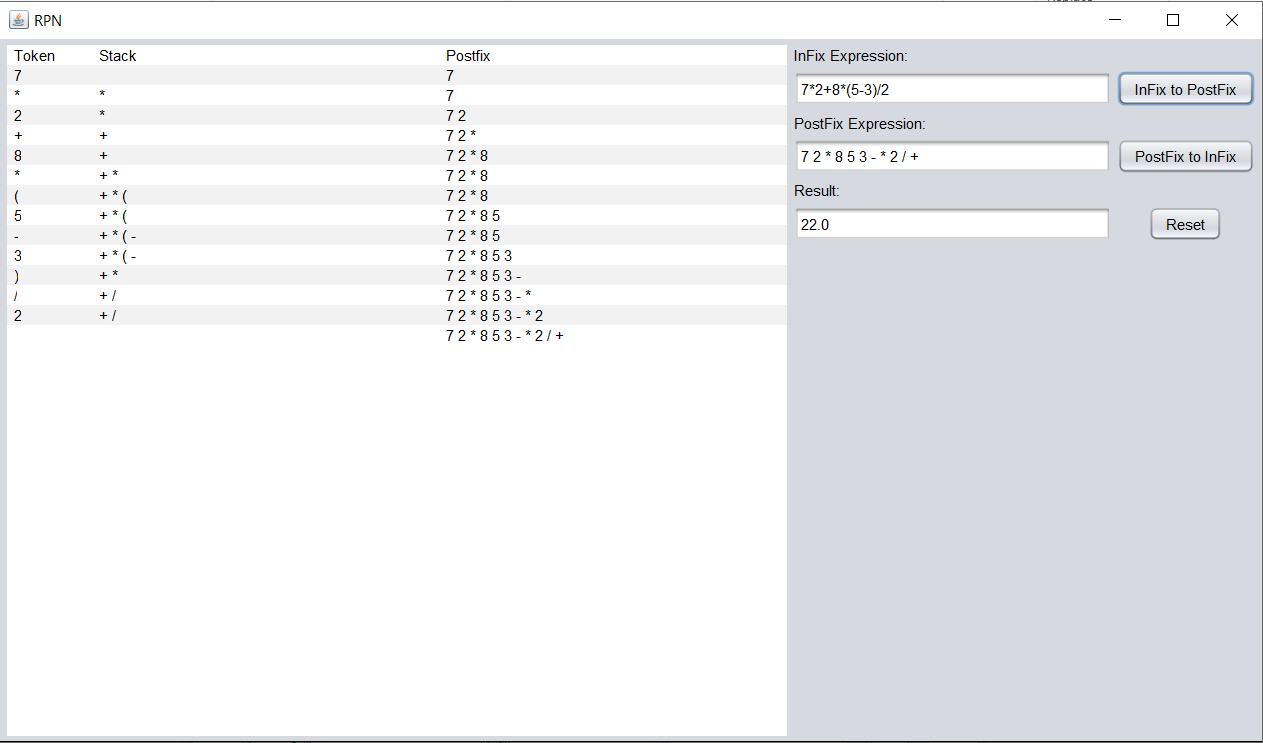


1. *Kết quả tình huống 1*
2. Nhập vào chuỗi Infix với toán hạng là các con số

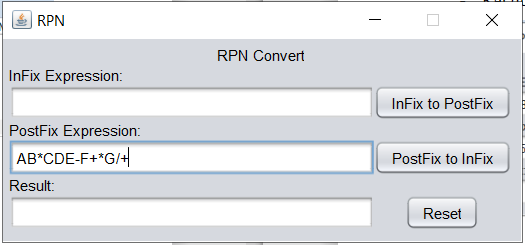


1. *Tình huống 2*

* Kết quả trả về:

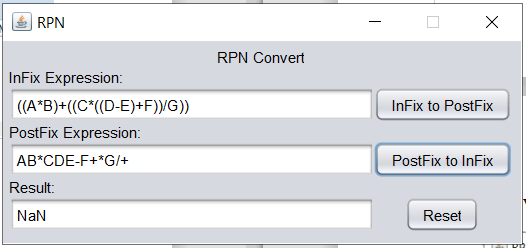


1. *Kết quả tình huống 2*
2. Nhập vào chuỗi Postfix

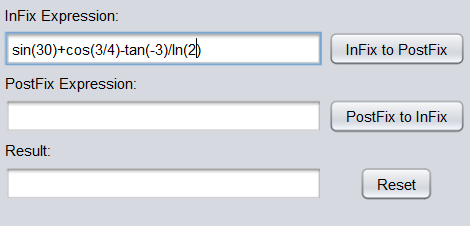


1. *Tình huống 3*

* Kết quả trả về:

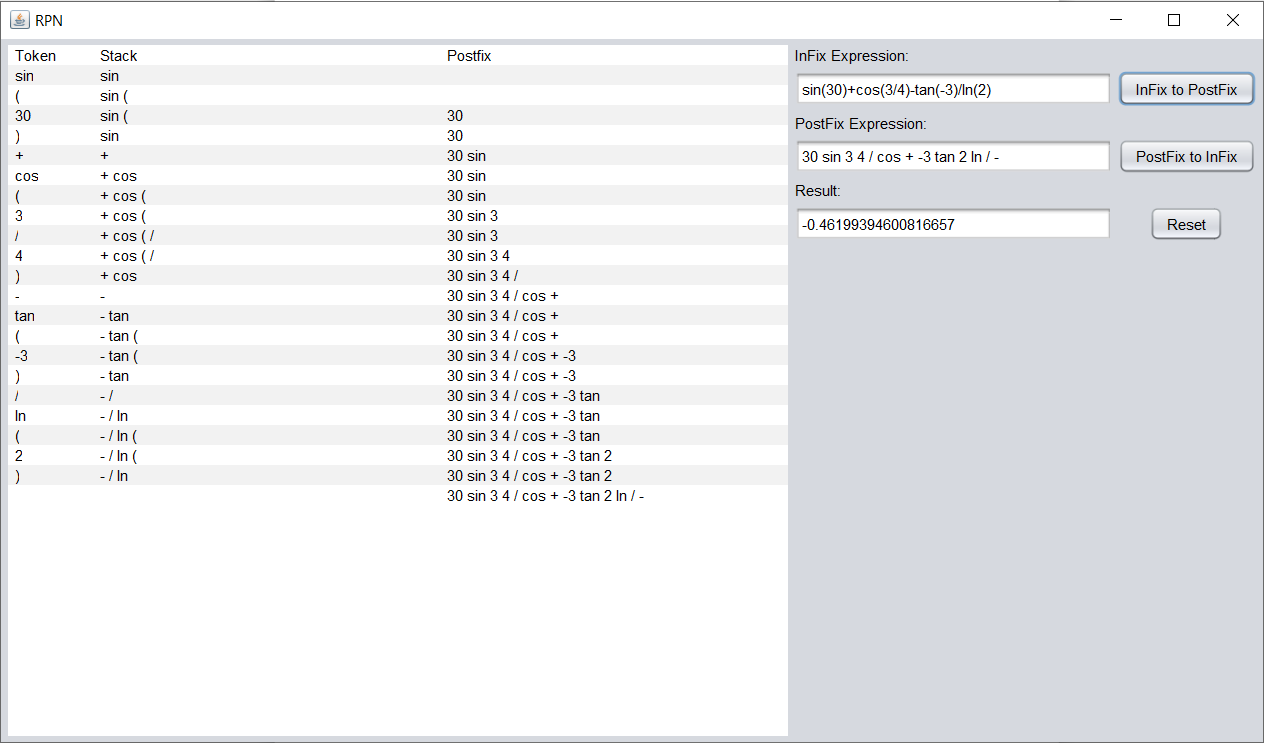


1. *Kết quả tình huống 3*
2. Nhập vào chuỗi các hàm sin, cos, tan

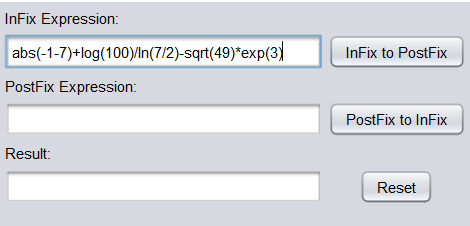


1. *Tình huống 4*

* Kết quả trả về:

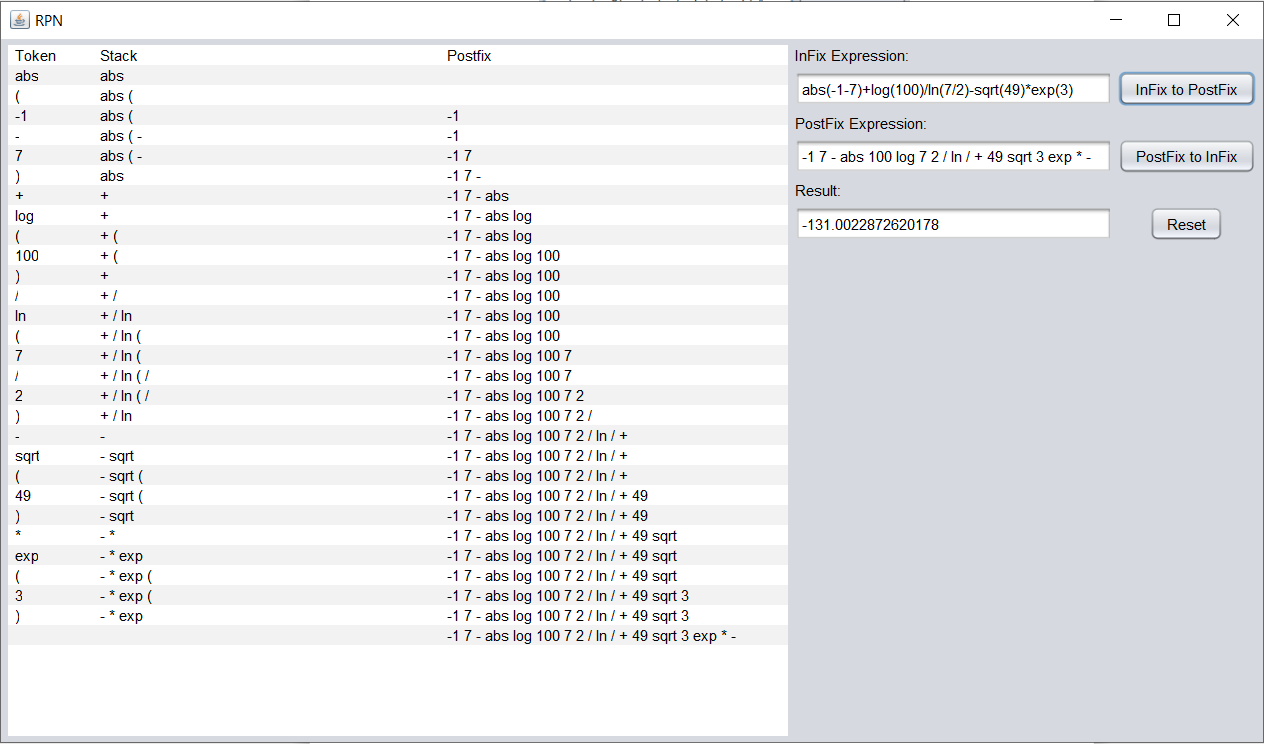


1. *Kết quả tình huống 4*
2. Nhập vào chuỗi Infix các hàm abs, log, ln, sqrt,exp



1. *Tình huống 5*

* Kết quả trả về:



1. *Kết quả tình huống 5*

# **PHẦN KẾT LUẬN**

1. **Kết quả đạt được:**

Có một máy tính tính toán biểu thức một cách đơn giản và nhanh chóng. Hiểu sâu hơn về các kiến thức đã học và tác dụng của chúng đối với thực tiễn.

1. **Kết luận:**
2. **Ưu điểm:**

Đề tài này đã giúp chúng tôi hiểu rõ hơn về cách thức tính toán của máy tính. Nắm rõ hơn các kiến thức đã học và biết thêm về một ngôn ngữ lập trình nữa là Java.

Thông qua quá trình làm đề tài này chúng tôi đã có thêm cho mình những kĩ năng như làm việc nhóm và giao tiếp ứng xử. Ngoài ra còn giúp chúng tôi nhận ra những thiếu sót của bản thân cần phải hoàn thiện.

1. **Nhược điểm:**

Ngoài những ưu điểm trên chương trình của chúng tôi vẫn còn nhược điểm là chưa tính toán được những biểu thức phức tạp như các máy tính hiện đại bây giờ.