Ton Duc Thang University
Faculty of Information Technology

Bài tập về nhà

Môn: **Cấu trúc dữ liệu và giải thuật 1**

Đề bài: Chuyển biểu thức trung tố sang hậu tố (Infix to Postfix)

(Sinh viên đọc kỹ tất cả hướng dẫn trong đề này trước khi làm bài)

I. Giới thiệu bài toán

Với những biểu thức đơn giản thực hiện từ trái sang phải thì máy tính có thể hiểu được để thực hiện và cho ra kết quả, ví dụ 10 + 3 - 8 thì máy tính sẽ hiểu lấy 10 và 3 cộng lại rồi trừ đi 8. Tuy nhiên với ví dụ 10 + (3 - 8) thì làm sao máy tính có thể hiểu được phải thực hiện biểu thức trong ngoặc trước ? Một trong những cách để giải quyết bài toán này là sử dụng biểu thức Ba Lan ngược.

Biểu thức Ba Lan ngược (Reverse Polish notation – RPN) là biểu thức được thể hiện ở dạng hậu tố (postfix). Ví dụ biểu thức infix như sau:

```
10 + ( 3 - 8 )
```

sẽ được chuyển sang dạng postfix là:

```
10 3 8 - +
```

Ta sẽ sử dụng Stack để chuyển, cách chuyển như sau:

Ký tự từ chuỗi infix	Bước thực hiện	Stack hiện tại	Ghi chuỗi postfix
10	Số - Ghi chuỗi		10
+	Dấu - Thêm vào Stack	+	
(	Dấu "(" - Thêm vào Stack	+ (	
3	Số - Ghi chuỗi	+ (	10 3
-	Dấu - Thêm vào Stack	+ (-	

8	Số - Ghi chuỗi	+ (-	10 3 8
)	Dấu “)“ - Đưa hết phần tử trong Stack ghi vào chuỗi đến khi gặp dấu (và xóa dấu (ra khỏi Stack	+	10 3 8 -
Không còn ký tự	Ghi tất cả những dấu còn trong Stack vào chuỗi		10 3 8 - +

Trong quá trình thêm toán tử vào Stack, nếu toán tử đang thêm vào có độ ưu tiên thấp hơn toán tử ở đỉnh thì ta phải lấy hết các toán tử có độ ưu tiên cao hơn ra để ghi vào chuỗi trước rồi mới thêm toán tử đang xét vào Stack.

Ví dụ biểu thức: (2 * 4 - 3) + 6

Khi xét đến dấu “-” ta đã có sẵn dấu “*” trong Stack, lúc này vì phép “-” có độ ưu tiên thấp hơn phép “*” do đó ta phải lấy dấu “*” ra khỏi Stack để ghi vào chuỗi 2 4 *, rồi mới thêm dấu “-” vào Stack. Do đó kết quả postfix của ví dụ này là 2 4 * 3 - 6 +

Để tính được biểu thức postfix của ví dụ bên trên, người ta sẽ sử dụng Stack. Cụ thể như sau:

```
10 3 8 - +
```

Ký tự từ chuỗi postfix	Bước thực hiện	Stack hiện tại
10	Số - Thêm 10 vào Stack	10
3	Số - Thêm 3 vào Stack	10, 3
8	Số - Thêm 8 vào Stack	10, 3, 8
-	Toán tử - Lấy số kế đỉnh Stack thao tác với số ở đỉnh Stack và thêm lại kết quả vào Stack, 3 - 8 = -5	10, -5
+	Toán tử - Lấy số kế đỉnh Stack thao tác với số ở đỉnh Stack và thêm lại kết quả vào Stack, 10 + -5 = 5	5

Lúc này kết quả sẽ là số cuối cùng còn trong Stack.

Sinh viên tham khảo thêm về bài toán và thuật toán giải tại:

Bài toán RPN: https://en.wikipedia.org/wiki/Reverse_Polish_notation

Thuật toán tạo chuỗi RPN: https://en.wikipedia.org/wiki/Shunting-yard_algorithm

II. Tài nguyên cung cấp

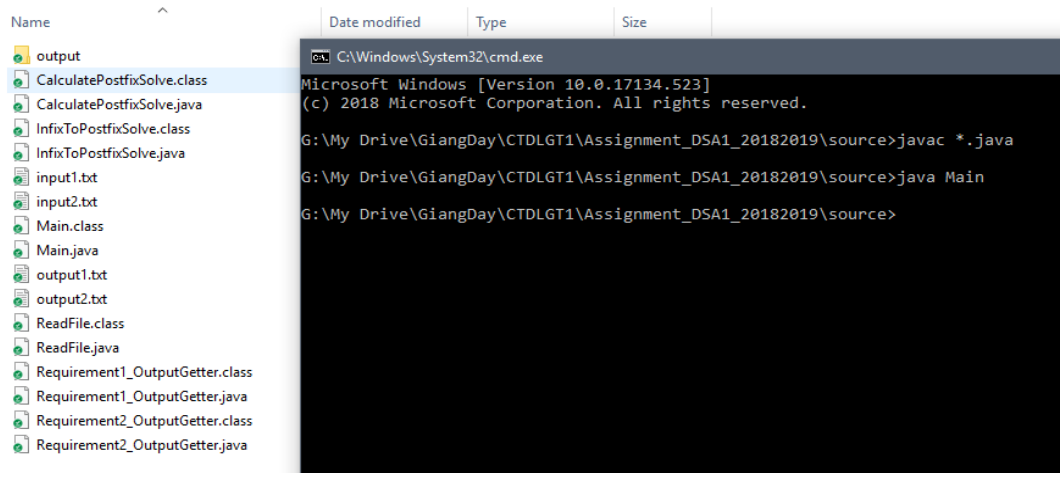
Source code được cung cấp sẵn bao gồm các file:

- File đầu vào và kết quả mong muốn:
  - *input1.txt*: chứa các biểu thức infix ví dụ cho yêu cầu 1.
  - *input2.txt*: chứa các biểu thức postfix ví dụ cho yêu cầu 2.
  - Folder *output* gồm: *output1.txt*: chứa kết quả mẫu của yêu cầu 1 và *output2.txt*: chứa kết quả mẫu của yêu cầu 2.
- File sinh viên **không được chỉnh sửa**:
  - *ReadFile.java*: class đọc file input.
  - *Main.java*: gọi đến các đối tượng theo yêu cầu của bài tập và ghi ra các file output.
  - *Requirement1_OutputGetter.java*: dùng để chấm điểm, sinh viên tuyệt đối không chỉnh sửa.
  - *Requirement2_OutputGetter.java*: dùng để chấm điểm, sinh viên tuyệt đối không chỉnh sửa.
- File sinh viên **code bài làm tại đây**:
  - *InfixToPostfixSolve.java*: class chuyển chuỗi infix thành chuỗi postfix.
  - *CalculatePostfixSolve.java*: class chuyển chuỗi postfix thành kết quả của biểu thức.

III. Cách biên dịch chương trình

Sinh viên chạy trên cửa sổ command line như đã được học trên lớp thực hành:

- Đưa đường dẫn đến thư mục đang chứa bài tập.
- Gõ lệnh *javac *.java* để biên dịch.
- Gõ lệnh *java Main* để chạy chương trình.



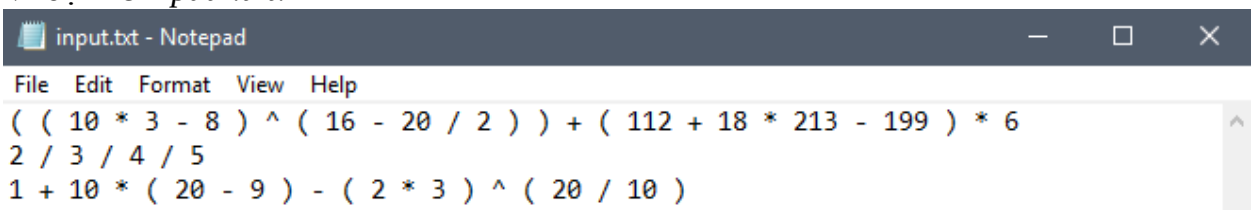
Sau khi chạy, hai file output kết quả bài làm sẽ tự sinh ra. Sinh viên thực hiện các yêu cầu ở mục dưới và so sánh kết quả output của mình với kết quả output đã được cung cấp sẵn.

IV. Yêu cầu của bài tập

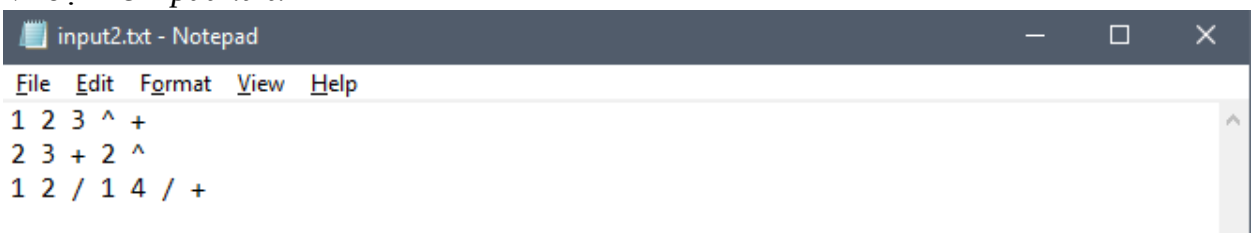
Lưu ý trước khi làm bài:

- Bài tập này chỉ yêu cầu thao tác với biểu thức chứa số nguyên (int) và các toán tử cộng "+", trừ "-", nhân "*", chia "/" và lũy thừa "^". File *input1.txt* và *input2.txt* chứa biểu thức có định dạng chuẩn mỗi ký tự cách nhau bằng một dấu khoảng trắng. Tất cả biểu thức dùng để chấm bài đều có nghĩa và đúng hoàn toàn với định dạng trên.

  Ví dụ file *input1.txt*:



  Ví dụ file *input2.txt*:



- Trong bài tập này sinh viên không cần tự định nghĩa lại Stack mà được phép sử dụng thư viện Stack có sẵn trong Java. Các file code cung cấp cho sinh viên đã được thêm sẵn thư viện Stack.

- Sinh viên xem kỹ các file input và output trước khi làm. File *input1.txt* có bao nhiêu dòng thì *output1.txt* sẽ có bấy nhiêu dòng kết quả tương ứng, tương tự với *input2.txt* có bao nhiêu dòng thì *output2.txt* sẽ có bấy nhiêu dòng kết quả tương ứng.
- Sinh viên có thể tự thêm testcase vào các file input để thử nhiều trường hợp khác nhau nhưng lưu ý thêm dữ liệu phải đúng định dạng đã được nên bên trên.
- Sinh viên nên đọc kỹ hàm **main** để xác định hướng hiện thực các class được yêu cầu. Hàm main có chức năng gọi đến *Readfile.java* để ghi file *output1.txt* và *output2.txt*.
- Bài làm của sinh viên **phải chạy được trên hàm main đã cung cấp sẵn**. Sinh viên không chỉnh sửa file **Readfile.java**, file **Main.java** và **2 file interface**.
- Sinh viên hiện thực **YÊU CẦU 1** trên file *InfixToPostfixSolve.java* và **YÊU CẦU 2** trên file *CalculatePostfixSolve.java*. Một số phương thức trong 2 file này sinh viên tuyệt đối không được chỉnh sửa tên và code bên trong phương thức – cụ thể sẽ được đề cập trong phần yêu cầu bên dưới.
- Sinh viên có thể thêm hàm mới để phục vụ bài làm tuy nhiên bài làm của sinh viên phải chạy được với file Main.java đã được cung cấp sẵn và không được phép sửa đổi những chỗ đã được nêu ở lưu ý bên trên.

YÊU CẦU 1 (5 điểm)

Giải thích class InfixToPostfixSolve (phần tên class và implements sinh viên không được chỉnh sửa):

Thuộc tính:

- inputString: chuỗi infix đầu vào (sinh viên không được chỉnh sửa tên thuộc tính)
- outputString: chuỗi postfix đầu ra (sinh viên không được chỉnh sửa tên thuộc tính)

Phương thức:

- Các hàm khởi tạo và set thuộc tính: sinh viên không được chỉnh sửa các phương thức này.
- String infixToPostfix(): dùng để chuyển chuỗi inputString infix thành chuỗi postfix.
- String[] stringTokenizer: dùng để tách khoảng trắng, kết quả trả về là mảng các chuỗi ký tự.
- int priorityOfOperator(String op): trả về mức độ ưu tiên của toán tử
- String getOutputString(): trả kết quả outputString, hàm main() sẽ gọi phương thức này để lấy kết quả. (sinh viên không được chỉnh sửa phương thức này)

- private boolean isNum(String c): phương thức kiểm tra chuỗi c truyền vào có phải là số hay không.

Sinh viên hiện thực các phương thức sao cho **inputString** là chuỗi infix đầu vào và **outputString** là chuỗi postfix trả về. Kết quả sẽ được hàm main ghi ra file "output1.txt".

YÊU CẦU 2 (5 điểm)

Giải thích class CalculatePostfixSolve (phần tên class và implements sinh viên không được chỉnh sửa):

Thuộc tính:

- String inputString: chuỗi postfix đầu vào (sinh viên không được chỉnh sửa tên thuộc tính)
- double resultOfExpression: kết quả của biểu thức (sinh viên không được chỉnh sửa tên thuộc tính)

Phương thức:

- Các hàm khởi tạo và set thuộc tính: sinh viên không được chỉnh sửa các phương thức này.
- double calculatePostfix(): dùng để chuyển chuỗi inputString postfix thành kết quả của biểu thức.
- String[] stringTokenizer: dùng để tách khoảng trắng, kết quả trả về là mảng các chuỗi ký tự.
- double getResultOfExpression (): trả kết quả của biểu thức, hàm main() sẽ gọi phương thức này để lấy kết quả. (sinh viên không được chỉnh sửa phương thức này)
- private boolean isNum(String c): phương thức kiểm tra chuỗi c truyền vào có phải là số hay không.

Sinh viên hiện thực các phương thức để **inputString** là chuỗi postfix đầu vào và **resultOfExpression** là kết quả của biểu thức. Trong quá trình tính toán nếu gặp trường hợp 0 / 0, sinh viên dừng bài toán và trả về kết quả của dòng biểu thức đang xét là 0.

Kết quả của yêu cầu được hàm main ghi ra file "output2.txt".

V. Hướng dẫn nộp bài

- Khi nộp bài sinh viên chỉ nộp lại file *InfixToPostfixSolve.java* và file *CalculatePostfixSolve.java*, **không nộp kèm bất cứ file nào khác và tuyệt đối không được sửa tên 2 file này.**

- **Sinh viên đặt 2 file bài làm vào thư mục MSSV_HoTen** và nén lại với định dạng **.zip** nộp lên Sakai vào mục DSA1_Assignment01.
- Trường hợp làm sai yêu cầu nộp bài (đặt tên thư mục sai, không để bài làm vào thư mục khi nộp, nộp dư file, …) thì bài làm của sinh viên sẽ bị **0 điểm**.

VI. Đánh giá và quy định

- Bài làm sẽ được chấm tự động thông qua testcase (file input và output có định dạng như mẫu đã gửi kèm) do đó sinh viên tự chịu trách nhiệm nếu không thực hiện đúng theo Hướng dẫn nộp bài hoặc tự ý sửa các phương thức đã ghi chú không được chỉnh sửa dẫn đến bài làm không biên dịch được khi chấm.
- Số lượng testcase sử dụng để chấm bài là 20 testcase, với 10 testcase input1.txt để chấm YÊU CẦU 1 và 10 testcase input2.txt để chấm YÊU CẦU 2.
- Nếu bài làm của sinh viên có file của yêu cầu nào khi biên dịch bị lỗi thì yêu cầu đó **0 điểm**.
- **Mọi hành vi sao chép code trên mạng, chép bài bạn hoặc cho bạn chép bài nếu bị phát hiện đều sẽ bị điểm 0.**
- Nếu bài làm của sinh viên có dấu hiệu sao chép trên mạng hoặc sao chép nhau, sinh viên sẽ được gọi lên phỏng vấn code để chứng minh bài làm là của mình.
- **Hạn chót nộp bài: 23h00 ngày 19/04/2019.**

-- HẾT --