TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN 2
MÔN KTMT&HN

PHAN NHẬT VINH – 1712914
LÊ VĂN VŨ – 1712919
PHẠM THỊ TUYẾT VY – 1712927

PHÂN CÔNG CÔNG VIỆC

MSSV	HỌ TÊN	CÔNG VIỆC
1712914	PHAN NHẬT VINH	• Đọc file input, lưu biểu thức vào chuỗi để xử lí. • Chuyển biểu thức sang dạng postfix, prefix và các hàm hỗ trợ xử lí chuyển đổi.
1712919	LÊ VĂN VŨ	• Viết hàm hỗ trợ đảo ngược chuỗi, đếm độ dài chuỗi, chuyển chuỗi sang số int. • Tính giá trị biểu thức dựa trên biểu thức dạng postfix.
1712927	PHẠM THỊ TUYẾT VY	• In các chuỗi biểu thức dạng postfix, prefix và kết quả biểu thức ra 3 file tương ứng. • Viết báo cáo.

I. Ý TƯỞNG THỰC HIỆN ĐỒ ÁN VÀ CÁC BƯỚC THỰC HIỆN

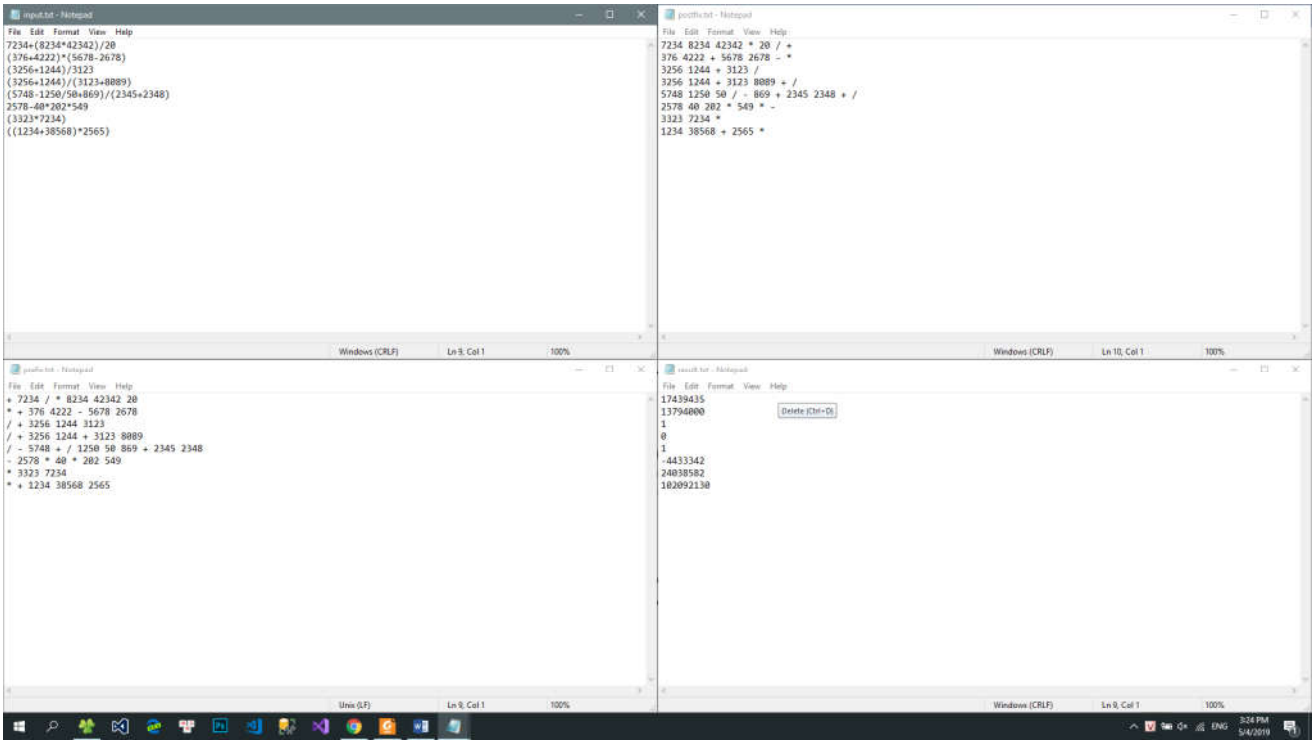
- Đồ án thực hiện xử lí và tính toán dựa trên thuật toán chuyển đổi biểu thức từ dạng trung tố (Infix) sang hậu tố (Postfix) và tiền tố (Prefix) và thuật toán tính giá trị biểu thức dựa trên biểu thức hậu tố (Postfix) được trình bày trong file mô tả đồ án 2 môn KTMT&HN – khoa CNTT, trường ĐH KHTN.
- Đầu tiên, chương trình sẽ mở các file input.txt để đọc, postfix.txt, prefix.txt, result.txt để ghi kết quả. Mỗi file descriptor sẽ lưu lại trong 1 thanh ghi để gọi syscall sau này.
- Bắt đầu với thủ tục ReadAllLines, thủ tục này chạy vòng lặp thủ tục ReadLine (đọc từng dòng file) cho tới khi hết file.
- Trong thủ tục ReadLine, ta tiến hành đọc từng kí tự một và lưu vào 1 vùng nhớ gọi là buff, khi gặp dấu xuống dòng thì thoát khỏi vòng lặp, lúc này buff chứa chuỗi biểu thức đầu vào.
- Việc đầu tiên là chuyển đổi chuỗi biểu thức này sang dạng postfix, với stack là một vùng nhớ được lưu địa chỉ tại thanh ghi $t0, vùng nhớ postfix chứa kết quả. Sau khi lưu các thanh ghi vào $sp, ta nhảy tới thủ tục toPostFix. Từ đây chương trình thực hiện từng bước như trong thuật toán trong file hướng dẫn đồ án. Lúc này ta chỉ quan tâm tới từng kí tự của biểu thức nên chưa đề cập đến việc chuyển đổi chuỗi sang số hay số sang chuỗi.

- **toPostFix:** load từng kí tự trong chuỗi biểu thức (được lưu trong vùng nhớ buff) và nhảy tới thủ tục token.Compare.
- **token.Compare:** xem xét kí tự vừa được load là kí tự gì, toán tử hay số, nếu là số thì save vào biểu thức kết quả. Nếu là toán tử thì xem xét nó là toán tử gì rồi nhảy tới hàm xác định độ ưu tiên của toán tử. Nếu là dấu ngoặc mở thì độ ưu tiên là 0, dấu +, - là 1, dấu *, / là 2. Sau đó nhảy đến thủ tục tokenChecker.push.
- **tokenChecker.push:** so sánh độ ưu tiên của toán tử ở trên với độ ưu tiên của toán tử ở đỉnh stack, nếu lớn hơn thì push vào stack và nhảy về toPostFix để xử lí kí tự tiếp theo, nếu không thì nhảy tới thủ tục stack.popAndPush.
- **stack.popAndPush:** pop toán tử ở đỉnh stack ra và save nó vào chuỗi kết quả, sau đó tiếp tục nhảy về tokenChecker.push để so sánh độ ưu tiên của toán tử kế sau đỉnh stack.
- Nếu **tokenChecker** đã đọc hết chuỗi (đọc đến kí tự xuống dòng) thì nhảy đến **tokenChecker.exit**).
- **tokenChecker.exit:** kiểm tra xem stack đã hết chưa, nếu còn thì lần lượt pop stack và lưu vào biểu thức kết quả. Nếu đã hết thì nhảy về $ra, kết thúc hàm (nhảy về $ra thông qua **tokenChecker.exitJump**).

- Tiếp đến ta thực hiện in chuỗi vừa chuyển sang postfix ra file postfix.txt, rồi chuyển sang bước tiếp theo: tính giá trị biểu thức dựa trên chuỗi postfix vừa rồi. Lúc này ta có calstack làm nhiệm vụ là stack cho việc tính toán, tempcalbuff là vùng nhớ tạm để xử lí chuỗi, result chứa kết quả.
    - Ta nhảy tới thủ tục **equationCal**. Thực hiện đọc từng kí tự, nếu là số hay toán tử thì thêm vào tempcalbuff, nếu là dấu khoảng cách (mã ASCII 32) thì nhảy tới **tempcalbuff.check.**
    - **tempcalbuff.check:** kiểm tra dữ liệu được lưu trong tempcalbuff là toán tử hay toán hạng, nếu là toán tử thì kí tự cuối cùng của chuỗi sẽ có mã ASCII <48, ngược lại thì là toán tử. Nếu nó là toán tử thì nhảy đến **calStack.poptwo**, còn là toán hạng thì nhảy đến thủ tục **atoi** để chuyển từ chuỗi sang số và push vào stack, empty tempcalbuff và nhảy về **equationCal** đọc kí tự tiếp theo.
    - **calStack.poptwo:** pop 2 toán hạng ở đỉnh calstack, lưu nó vào 2 thanh ghi, và xét toán tử là gì thì nhảy đến thủ tục tính toán phù hợp và lưu kết quả vào calstack.
    - Khi **equationCal** đã đọc hết chuỗi biểu thức thì nhảy đến **equationCal.exit,** tiến hành pop hết calstack và tính toán, rồi lưu kết quả vào calstack. Sau cùng, ta lấy phần tử ở đỉnh calstack ra, nhảy đến **intToString** để chuyển sang chuỗi rồi lưu vào result, nhảy về $ra.
- Ta tiếp tục thực hiện in result ra file result.txt, rồi đến bước chuẩn bị tiếp theo: đảo ngược chuỗi biểu thức ban đầu đọc từ file, kết quả được lưu trong reversedbuff:

- Ta nhảy đến **toReversedEquation** để chuyển, lưu các biến cần thiết rồi nhảy đến **reversedEquation**. Bắt đầu đọc từng kí tự trong chuỗi từ phải qua trái và lưu dần vào chuỗi tạm tempreverse, nếu là các dấu ngoặc thì đổi chiều dấu ngoặc rồi mới lưu. Đến khi đọc đến toán tử hay dấu ngoặc thì nhảy đến **prefix.Operator.**
- **prefix.Operator:** trước hết sẽ nhảy tới **reverse** để đảo ngược chuỗi số trong tempreverse (để không làm đảo ngược giá trị của toán hạng khi đọc ngược từ cuối lên đầu chuỗi) rồi nhảy tới **copyToReversedString.** Kết quả của chuỗi số đảo ngược được lưu trong tempreverse2
- **copyToReversedString:** copy số đã đảo ngược từ tempreverse2 vào chuỗi kết quả, rồi sau đó copy toán tử vào rồi nhảy đến **copyToReversedString.exit.**
- **copyToReversedString.exit:** empty các vùng nhớ tempverse, tempverse2 cho lần chạy tiếp theo, rồi nhảy về **reversedEquation** để đọc tiếp cho hết chuỗi.
- Nếu đã đọc hết chuỗi, nhảy đến **reversedEquation.exit1** để reverse số lần cuối, nhảy đến **reversedEquation.exit2** để lưu vào chuỗi kết quả, rồi nhảy đến **reversedEquation.exitConfrim** để nhảy về $ra.

- Lúc này chuỗi biểu thức đảo ngược đã được lưu trong reversedbuff, ta tiến hành chuyển chuỗi này sang dạng postfix theo các bước như trên, kết quả sẽ được lưu trong vùng nhớ prefix.
- Sau đó, ta tiến hành đảo ngược lại chuỗi prefix như các bước ở trên 1 lần nữa, kết quả cuối cùng là chuỗi biểu thức dạng prefix được lưu trong reversedbuff. Ta tiến hành in chuỗi biểu thức này ra file prefix.txt.



II. CHỨC NĂNG LÀM ĐƯỢC VÀ CHƯA LÀM ĐƯỢC:

a. Làm được:

Làm được hết các yêu cầu của đồ án.

b. Chưa làm được:

Chưa đảm bảo bao quát hết tất cả các trường hợp đặc biệt, chưa tối ưu hóa code (việc sử dụng thanh ghi còn hơi "bừa bộn", cấu trúc thủ tục của chương trình cũng chưa được rõ ràng, mạch lạc). Cần thêm thời gian để tối ưu vấn đề này.

c. Mức độ hoàn thành: 90%

* Các nguồn tài liệu tham khảo:

- Slide bài giảng môn KTMT&HN của cô Chung Thùy Linh.
- Các trang Stackoverflow, Geeksforgeeks,…