Full name: Mai Anh Phúc Minh

Student ID: 24120094

Report Lab 1.

No.	Percentage understood	Content understood	Percentage Referenced	Content Referenced	Referenced Soure
1. addition()	100%				
2. subtraction()	100%				
3. multiplication()	100%				
4. division()	100%				
5. changeToPostfix()	100%				
6. main()	100%				

Em xin được báo cáo về bài tập Lab 1. Với yêu cầu thiết kế chương trình dùng để tính toán các số nguyên cực lớn (>100 chữ số), em đã thiết kế **4 hàm phụ** và **5 hàm chính** để xử lí các yêu cầu.

1. Các hàm phụ.

- Hàm absolute(): hàm này được xây dựng để lấy giá trị tuyệt tối của số được đưa vào. Em triển khai bằng cách xóa đi kí tự đầu nếu nó là số âm, còn không sẽ giữ nguyên số.

- Hàm precedence(): hàm này được xây dựng để gán giá trị ưu tiên tương ứng với các toán tử. Với các toán tử + - sẽ có độ quan trọng là 1 còn * / sẽ có độ quan trọng là 2.

- Hàm compareNumber(): được xây dựng để so sánh 2 số với nhau. Em so sánh theo thứ tự a và b. Nếu độ dài của a nhỏ hơn b thì sẽ trả về 0 tương ứng với a nhỏ hơn b, khi độ dài của a lớn hơn độ dài của b thì sẽ trả về 2 tương ứng với a lớn hơn b, còn nếu độ dài a và b bằng nhau, em sẽ duyệt đồng thời lần lượt mọi chữ số của a và b, nếu gặp chữ số thứ i của a lớn hơn thứ i của b thì trả về 2, nhỏ hơn thì trả về 0, chỉ khi duyệt mọi chữ số mà chưa trả về giá trị nào thì sẽ trả về 1 tương ứng với a bằng b.

- Hàm compareOperation(): hàm này được xây dựng để so sánh độ ưu tiên của 2 toán tử. Tương tự như khi so sánh số, em so sánh theo thứ tự a và b. khi độ ưu tiên của a lớn hơn b thì sẽ trả về 2, bằng thì trả về 1 còn nhỏ hơn sẽ trả về 0.

2. Các hàm chính.

- Hàm addition(): được xây dựng để cộng 2 chuỗi số nguyên lớn. Tương tự với cộng 2 số nguyên bình thường, em sẽ cộng từ hàng đơn vị đi lên (từ phải sang trái), nếu vượt quá 1 chữ số thì sẽ đưa số ở hàng đơn vị đó vào một stack lưu, phần còn lại đưa vào biến nhớ sau đó chuyển qua cộng hàng chục và cộng thêm biến nhớ đó vào và lặp lại cho tới khi duyệt qua mọi chữ số của 2 chuỗi và biến nhớ bằng 0. Sau khi thực hiện việc cộng xong ta sẽ pop các phần tử trong stack ra thành 1 cái string và trả về string đó. Vì xử lí từ phải qua trái mà số thì được lưu từ trái qua phải nên việc sử dụng stack là tối ưu.

- Hàm subtraction(): được xây dựng để trừ 2 chuỗi số nguyên lớn. Tương tự với trừ 2 số nguyên bình thường, em thực hiện trừ từ trái qua phải, nếu giá trị nhỏ hơn thì sẽ cộng 10 vào và lưu biến nhớ để trừ. Sau đó chuyển từ stack thành string tương tự như hàm cộng.

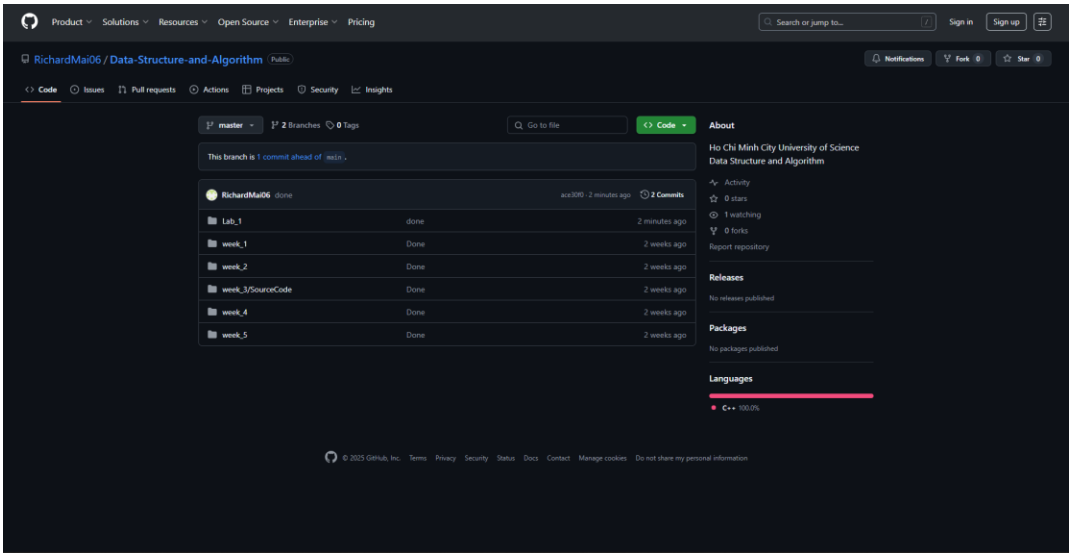
- Hàm multiplication(): được xây dựng để nhân 2 chuỗi số nguyên lớn. Em duyệt tuần tự từng chữ số của chuỗi đầu và nhân nó cho mọi chữ số của số thứ 2, sau đó cộng dồn vào string kết quả và nhích sang trái 1 đơn vị tương tự như khi nhân 2 số nhân bình thường.

- Hàm division(): được xây dựng để chia 2 chuỗi số nguyên lớn. Nếu nhận thấy chuỗi thứ 2 là số 0 hoặc chuỗi 1 bé hơn chuỗi 2 thì sẽ trả về các lỗi tương ứng. Ta sẽ duyệt từng phần tử của chuỗi 1, nếu nó nhỏ hơn chuỗi 2 thì thêm số 0 vào chuỗi kết quả. Đến khi chuỗi phụ đủ lớn để chia cho chuỗi 2 thì ta sẽ tìm kiếm nhị phân được thương số lớn nhất và lưu vào string kết quả, sau đó nhân chuỗi 2 cho thương và lấy chuỗi phụ trừ đi phép nhân đó. Nếu sau khi duyệt hết phần tử của chuỗi 1 mà chuỗi phụ không bằng 0 thì ta xuất lỗi chia số thực, còn không thì trả về chuỗi kết quả.

- Hàm changeToPostfix(): được xây dựng để chuyển hóa một biểu thức infix thành postfix. Vận dụng queue và stack, em có 2 stack operator và bracket, một cái để chứa tạm các toán tử, một cái để lưu trữ và kiểm tra xem có bị thừa các dấu ngoặc không. Khi ta duyệt qua các phần tử trong biểu thức, nếu là số thì ta đẩy thẳng vào queue postfix, nếu là toán tử, ta xem xét rằng stack chứa toán tử có rỗng không, nếu có thì đẩy thẳng vào stack, hoặc toán tử trên cùng của stack có độ ưu tiên nhỏ hơn thì ta push vào stack. Còn nếu bằng hoặc nhỏ hơn thì ta sẽ pop trong stack ra và thêm vào trong queue cho tới khi nào thỏa điều kiện trước hoặc stack rỗng. Tới khi duyệt hết phần tử trong biểu thức mà stack không rỗng thì ta chỉ việc pop hết vào trong queue.

- Hàm main(): nhận 2 tham số là input path và output path. Sau đó ta đọc các biểu thức trong input file, xủ lí rồi ghi kết quả vào output file. Ta lặp lại cho tới khi duyệt hết biểu thức trong input file. Trong quá trình xử lí, ta chuyển biểu thức thành postfix, sau đó pop các phần tử trong postfix ra. Nếu nó là số thì push vào stack thực thi, đến khi gặp toán tử thì ta pop 2 phần tử trên cùng trong stack ra xử lí rồi push kết quả vào stack lại rồi duyệt queue tiếp cho đến khi hết phần tử trong queue. Nếu khi có toán tử để xử lí mà trong stack chỉ có 1 toán tử thì vẫn sẽ xuất ra lỗi và chuyển sang biểu thức tiếp theo.

3. Minh chứng Github.



Link Github: https://github.com/RichardMai06/Data-Structure-and-Algorithm.git