BÁO CÁO BÀI LAB 1:

TÍNH TOÁN VỚI SỐ NGUYÊN LỚN

Nguyễn Đình Trung Kiên – 24120195

BẢNG CÁC NỘI DUNG THAM KHẢO:

Thứ tự	**Tỉ lệ hiểu được (trên nội dung đã tham khảo)**	**Nội dung hiểu được**	**Tỉ lệ tham khảo (so với bài làm)**	**Nội dung tham khảo**	**Nguồn tham khảo**
1	100%	Cách tạo hàm xác định độ ưu tiên toán tử một cách trực quan bằng cách trả về số nguyên rồi so sánh.	90%	Nội dung hàm xác định độ ưu tiên toán tử.	Hàm *precedence* từ link: https://github.com/BaseMax/MathematicalExpressionEvaluator/blob/03c3de606b5b7f243ff90faee69da77f7a7e00ca/MathematicalExpressionEvaluator.cpp
2	90%	Cách sử dụng stack để xử lý biểu thức và tính kết quả.	80%	Cách xử lý cụ thể với từng ký tự trong biểu thức với stack. (hàm *expressionCalculate*)	Hàm *evaluateExpression* từ link: https://github.com/BaseMax/MathematicalExpressionEvaluator/blob/03c3de606b5b7f243ff90faee69da77f7a7e00ca/MathematicalExpressionEvaluator.cpp

3	100%	Cách thực hiện phép tính chia hai số nguyên và mô phỏng lại bằng code.	70%	Cách thực hiện phép tính chia 2 số nguyên nhỏ, sử dụng vòng lặp rồi nhân ngược lại để tìm ra thương của phép chia.	chatGPT

I. Khái quát bài làm:

- Sử dụng các thư viện: *<iostream>, <stack>, <string>, <fstream>*.
- Biểu diễn số nguyên lớn bằng dữ liệu xâu ký tự (string).
- Sử dụng stack và các cách tính toán cơ bản với số nguyên thường để tính kết quả của từng biểu thức.

II. Các hàm sử dụng:

1. precedence:

- Hàm xác định độ ưu tiên của một toán tử cụ thể.
- Trả về 2 đối với toán tử có độ ưu tiên cao nhất: *, /.
- Trả về 1 đối với toán tử còn lại: +, -.
- Nếu không phải toán tử thì trả về 0.

2. higherPrecedence:

- Hàm kiểm tra độ ưu tiên của 2 toán tử.
- Trả về true nếu toán tử 1 có độ ưu tiên cao hơn hoặc bằng toán tử 2.

3. isNum:

- Hàm kiểm tra xem một ký tự có phải là chữ số hay không.
- Trả về true nếu ký tự là chữ số từ 0 đến 9.

4. isOperator:

- Hàm kiểm tra xem một ký tự có phải là toán tử hay không.
- Trả về true nếu ký tự là một trong bốn toán tử: +, -, *, /.

5. eraseZero:

- Hàm xóa hết các chữ số ‘0’ ở đầu một số, tạo điều kiện hợp lệ cho các phép so sánh hay tính toán về sau.
- Nếu chuỗi ký tự số đang ở dạng “0000...” thì cũng sẽ được chuẩn hóa về “0”.

6. checkExpression:

- Hàm kiểm tra trước khi thực hiện tính toán xem một biểu thức có hợp lệ về mặt cú pháp hay không.
- Các lỗi mà hàm sẽ kiểm tra:
  + Thứ tự toán tử và toán hạng (sau toán hạng sẽ phải là toán tử và sau toán tử phải là toán hạng hoặc một biểu thức trong ngoặc): Sử dụng biến để lưu lại thứ tự xuất hiện của toán tử và toán hạng, từ đó kiểm tra tính hợp lệ.
  + Các dấu ngoặc ở vị trí hợp lệ (ngoặc để bao quanh số, ngoặc để bao quanh biểu thức, biểu thị mức độ ưu tiên): Sử dụng stack để lưu lại các ngoặc mở và đối chiếu với ngoặc đóng.
  + Có các ký tự không hợp lệ nào hay không (các ký tự ngoài số, toán tử và ngoặc).
- Nếu có lỗi hàm sẽ dừng và biểu thức đó cũng sẽ không được thực hiện tính toán.

7. charToInt:

- Hàm chuyển đổi một ký tự chữ số (từ 0 đến 9) thành kiểu số nguyên (int) để tính toán (sử dụng thứ tự của ký tự trong bảng mã ASCII).

8. intToStr:

- Hàm chuyển đổi một số nguyên thành chuỗi ký tự.

9. isNegative:

- Hàm kiểm tra xem một số nguyên (ở dạng chuỗi ký tự) có phải là số âm hay không (ký tự đầu của số có phải là '-' hay không).

10. bigNumAbs:

- Hàm lấy giá trị tuyệt đối của chuỗi ký tự số (giữ nguyên nếu là số dương và bỏ dấu trừ nếu là số âm).

11. biggerNum:

- Hàm so sánh 2 số nguyên ở dạng chuỗi ký tự.
- Dựa vào dấu của 2 số và số lượng ký tự, nếu cả 2 đều không đưa ra kết quả được thì so sánh từng ký tự từ chữ số quan trọng nhất đến cuối.
- Trả về 1 nếu số nguyên đầu tiên lớn hơn số nguyên thứ 2, 0 nếu bằng nhau và -1 nếu nhỏ hơn.

12. bigNumAdd:

- Hàm cộng 2 số nguyên và trả về kết quả ở dạng chuỗi ký tự (2 số nguyên đã được chuẩn hóa).
- Hàm được thiết kế để lấy tổng 2 số dương, vì vậy ở các trường hợp có số âm đều được chuyển về phép trừ hoặc biến đổi lại thành phép cộng.
- Sử dụng phương pháp cộng lần lượt từng chữ số theo từng hàng đơn vị, hàng chục,... cho đến hết cả 2 số. Đồng thời sử dụng biến *carry* để lưu lại phép "nhớ" nếu ở một hàng nào đó có kết quả phép cộng tạo thành số có 2 chữ số.
- Kết quả ở từng phép cộng theo từng chữ số được nối vào đầu của xâu kết quả, cộng đến khi đã duyệt hết tất cả các chữ số.

13. bigNumSubstract:

- Hàm lấy phép trừ 2 số nguyên (số thứ nhất trừ số thứ hai, 2 số đã được chuẩn hóa trước khi tính toán).
- Hàm được thiết kế để tính phép trừ của 2 số dương, số lớn trừ số bé. Vì vậy các trường hợp khác trên đều được biến đổi lại để tạo thành phép trừ phù hợp với thuật toán của hàm.
- Thuật toán gần như tương đồng với phép cộng 2 số nguyên: Duyệt các chữ số theo từng hàng rồi lấy trừ nhau (cùng với biến nhớ nếu có). Nếu trừ ra số dương thì đưa hẳn vào xâu kết quả, ngược lại nếu là số âm thì cộng thêm cho 10 để đưa vào xâu kết quả, đồng thời sử dụng biến nhớ *carry* để lưu lại cho phép trừ ở hàng tiếp theo.
- Chuẩn hóa lại kết quả trước khi trả về.

14. isZero:

- Kiểm tra một số nguyên dưới dạng xâu ký tự có phải là số 0 hay không (cho các trường hợp như "00000..." khi chưa được chuẩn hóa).

15. addZero:

- Hàm dùng để thêm n số 0 vào sau một số nguyên (dùng cho các phép tính toán theo hàng đơn vị, trăm, chục... tương ứng ở các thuật toán sau).

16. changeNumSign:

- Hàm dùng để đổi dấu một số nguyên (nếu là âm thì thành dương và ngược lại).

17. bigNumMultiply:

- Hàm nhân 2 số nguyên lớn với yêu cầu số lượng chữ số của số nguyên thứ nhất phải dài hơn hoặc bằng số lượng chữ số của số nguyên thứ 2.
- Vì vậy trường hợp ngược lại cũng sẽ được biến đổi cho hợp lệ.
- Các toán tử cũng đã được chuẩn hóa trước khi thực hiện phép tính.
- Sử dụng biến *changeSign* để lưu lại dấu của 2 số nguyên khi nhân vào với nhau. Đồng thời lấy giá trị tuyệt đối của 2 toán tử để dễ dàng thực hiện phép tính với 2 số dương.
- Thuật toán được sử dụng là nhân từng chữ số của thừa số thứ 2 với thừa số thứ 1, sau đó cộng lại theo từng hàng (ở hàng đơn vị ở thừa số 2 thì cộng theo hàng đơn vị, ở hàng chục thì phải đẩy thêm một số 0 vào hàng đơn vị, ở hàng trăm thì đẩy thêm hai số 0... để khi cộng các tổng theo từng hàng lại với nhau bằng hàm tổng thì ta có được phép cộng hợp lệ, thêm các số 0 vào cuối cũng sẽ không ảnh hưởng đến kết quả). Ở các bước nhân từng chữ số, ta cũng thực hiện thuật toán tương tự là nhân chữ số đang xét ở thừa số 2 với từng chữ số của thừa số 1 và lấy tổng qua từng bước.
- Trả về xâu kết quả là phép tổng qua từng bước duyệt và nhân, sử dụng biến *changeSign* để biết dấu của kết quả rồi trả về.

18. getQuotion:

- Hàm sử dụng để tính thương của (op1 / op2) khi 2 số khá lớn và thương của 2 số chỉ nằm trong khoảng từ 0 tới 9.
- Sử dụng vòng lặp cùng biến *mul* kiểm tra từ 0 đến 9, trả về nếu phép nhân thương với biến *mul + 1* vượt quá op1, có nghĩa là *mul* chính là thương để op1 / op2 = mul (vì khi *(mul + 1) * op2* vượt quá op1 thì *mul* chính là số lớn nhất để *op1 / op2* tạo ra thương lớn hơn 1, nghĩa là *mul* chính là thương của phép chia *op1 / op2*).
- Đồng thời sử dụng biến *res* truyền vào tham chiếu để có được thương đó, hàm sẽ trả về phép nhân op2 và *mul* phục vụ cho hàm chia chính của 2 số nguyên.

19. bigNumDivide:

- Hàm sử dụng để chia 2 số nguyên và lấy phần nguyên làm tròn xuống.
- Sử dụng biến *changeSign* để xác định dấu của kết quả như hàm nhân. Phép chia số dương lớn cho số dương bé nên các trường hợp chưa thỏa sẽ được biến đổi cho về đúng dạng (số bé chia số lớn có kết quả là “0”).

- Hàm sẽ kiểm tra trường hợp vô định là số bị chia là “0” và trả về “Error” nếu phát hiện, đồng thời các trường hợp đặc biệt khác như số chia là “0” hay số bị chia là “1”.
- Bước đầu tiên của thuật toán là tìm ra chuỗi con tính từ đầu số chia có thể chia được cho số bị chia (lớn hơn hoặc bằng số bị chia) rồi dùng hàm *getQuotion* để tìm ra thương của phép chia để đưa vào xâu kết quả cũng như là phép nhân ngược lại của thương với số bị chia để tìm phần dư *remainder*.
- Biến *remainder* để lưu lại phần dư của phép trừ *tmpDiv – tmpDivided* (là chuỗi con được chọn ra để chia cho số bị chia và phép nhân ngược lại của thương và số chia).
- Ở mỗi bước tiếp theo, ta hạ tiếp một chữ số kế tiếp từ số chia xuống đặt ngay sau *remainder* rồi lặp lại quá trình dùng hàm *getQuotion* để có được thương, thêm vào xâu kết quả rồi lại lấy *remainder* để lưu lại phép trừ giữa phần dư cũ và phép nhân mới (thương với số bị chia).
- Lặp lại đến khi không còn chữ số nào của số chia có thể hạ xuống được nữa thì ta có được xâu kết quả hoàn chỉnh.
- Kiểm tra dấu của kết quả rồi trả về.

20. numbersCalculate:

- Hàm tính kết quả của một biểu thức cơ bản mà không cần xử lý xâu biểu thức.
- Sử dụng các hàm trước đó để trả về kết quả giữa *n1, n2* với toán tử *op* được truyền vào.

21. expressionCalculate:

- Hàm xử lý biểu thức infix để tính ra kết quả từ các hàm phép tính cơ bản.
- Kiểm tra tính hợp lệ của biểu thức trước khi xử lý, nếu biểu thức có lỗi thì dừng hàm và không cần làm gì thêm, trả về “Error”.
- Sử dụng 2 stack là *numbers* và *operators* để lưu lại các giá trị số, kết quả tính được và toán tử.
- Lần lượt xét qua các ký tự trong biểu thức, bỏ qua các ký tự dấu cách:
  + Nếu là dấu ngoặc mở: ta push vào *operators* để lưu lại thứ tự ưu tiên.
  + Nếu là dấu đóng ngoặc: ta pop từ stack *operators* đến khi tìm được dấu mở ngoặc. Ở mỗi bước pop được toán tử nào, ta sẽ pop thêm 2 giá

trị từ stack *numbers* rồi tính bằng hàm *numbersCalculate* với giá trị 2 tính với giá trị 1, sau đó push lại vào stack *numbers.*

+ Nếu là toán tử và ở đầu stack *operators* đang có toán tử có độ ưu tiên cao hơn hoặc bằng, ta pop liên tục toán tử ra đến khi toán tử ở đầu stack có độ ưu tiên thấp hơn hoặc đã pop hết toán tử. Với mỗi toán tử pop ra được, ta thực hiện phép tính như thao tác khi ta duyệt tới dấu ngoặc đóng.

+ Nếu là số, ta thực hiện đọc hết số từ biểu thức rồi push vào stack *operators*.

- Duyệt đến khi hết biểu thức và stack *operators* trở thành stack rỗng, ở mỗi toán tử còn được pop ra, ta cũng thực hiện tính toán với các toán hạng ở *numbers* như bước duyệt tới dấu ngoặc đóng.
- Phần tử duy nhất còn lại ở stack *numbers* chính là kết quả của phép tính.

22. Hàm main() với tham số đọc từ *command line:*

- Ta đọc được 3 tham số từ *terminal* với *command line* để chạy chương trình với tên file input ở tham số thứ 2 và output ở tham số thứ 3.
- Ta đọc file với 2 tham số trên, xử lý từng dòng ở file input rồi in ra ở file output, đồng thời in ra màn hình *terminal* kết quả của các biểu thức.