

# REPORT

## Lab 1 : Large Integer Arithmetic Expression

24120184 -Lê Quốc Hưng.

STT	Tỷ lệ hiểu	Nội dung đã hiểu	Tỷ lệ tham khảo	Nội dung tham khảo	Nguồn tham khảo
1.Hàm Negative	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
2.Hàm absValue	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
3.Hàm Addition	95%	Hiểu được cách xử lý chuỗi, để cộng 2 số nguyên lớn 1 cách hợp lí	5%	Bước xử lý cuối khi cộng 2 số nguyên lớn nếu biến nhớ vẫn còn giá trị thì thêm vào '1' cuối kết quả. (1)	ChatGPT
4.Hàm equal	70%	Hiểu được ý tưởng và cách triển khai hàm bằng code	30%	Cách thức mà toán tử == hoạt động khi so sánh 2 chuỗi. (2)	ChatGPT
5.Hàm Smaller	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
6.Hàm removeZero	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
7.Hàm Subtraction	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
8.Hàm MultiplicationByDigit	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
9.Hàm Multiplication	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
10.Hàm Division	70%	Biết được cách chia nhưng vẫn không có ý tưởng để triển khai code 1 cách mạch lạc	30%	Dựa vào code của ChatGPT cung cấp hiểu được ý tưởng và code lại theo những gì mình hiểu và không copy.(3)	ChatGPT
11.Hàm IsOperator	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X

12.Hàm IsNum	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
13.CheckError	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
14.Hàm delete_	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
15.Hàm NumToVariable	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
16.Hàm Precedence	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
17.Hàm PrecedenceBiggerOrEqual	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
18.Hàm resultMath	100%	Hiểu được ý tưởng và cách triển khai hàm bằng code	0%	X	X
19.Hàm InfixToPostFix	90%	Hiểu được cách triển khai thuật toán để chuyển từ Infix sang PostFix, nhưng ban đầu viết chỉ chuyển sang dạng chuỗi	10%	Được ChatGPT gợi ý chuyển sang dạng vector để dễ dàng xử lý hơn (được gợi ý thông qua khi đang xử lý hàm thứ 20).(4)	ChatGPT
20.HàmresultMathExpression	40%	Hiểu được cách xử lý bằng cách xét toán tử và thực hiện phép tính với 2 số trước nó (trong PostFix), nhưng không được tối ưu và dễ hiểu do ban đầu xử lý PostFix dưới dạng 1 chuỗi	60%	Được ChatGPT gợi ý sử dụng stack để thực hiện các phép toán và gợi ý chuyển PostFix sang vector để xử lý.(4)	ChatGPT
21.Hàm ReadFileInputAndPrintResult	100%	Tổng hợp từ các hàm trước và đưa ra kết quả	0%	X	X

Ở bảng này chỉ chỉ ra phần nội dung tham khảo, nội dung hiểu được , chi tiết hiểu hàm như thế nào sẽ được giải thích phía dưới.

### Những gì cần làm để hoàn thành bài tập:

#### Ý tưởng để xử lý:

- Viết các hàm trả về kết quả khi cộng, trừ, nhân, chia 2 số nguyên lớn với nhau (+ , - , \* , / ) dưới dạng chuỗi.

-Đọc file lưu biểu thức theo từng dòng nằm trong file input, lưu các số thành các biến để dễ dàng xử lý.

-Viết hàm checkError để kiểm tra biểu thức có bị lỗi hay không (vd: dư phép toán , dư dấu ngoặc ,... ).

-Sau khi kiểm tra Error thì sẽ xóa khoảng trắng có trong chuỗi để dễ dàng xử lý.

-Chuyển các số trở thành các biến dưới dạng chuỗi.(vd: 1+2+3 sẽ thành n1+n2+n3).

-Xử lý biểu thức: chuyển Infix thành Postfix.

-Viết hàm xử lý Postfix để tìm được kết quả của biểu thức.

### Các hàm cần viết:

- 1.bool isNegative(string n);
- 2.string absValue(string n);
- 3.string Addition(string n1,string n2);
- 4.bool equal(string n1,string n2);
- 5.bool Smaller(string n1,string n2);
- 6.string removeZero(string n);
- 7.string Subtraction(string n1, string n2);
- 8.string MultiplicationByDigit(string n1,char n2);
- 9.string Multiplication(string n1, string n2);
- 10.string Division(string n1, string n2);
- 11.bool IsOperator(char a);
- 12.bool IsNum(char a);
- 13.bool checkError(string n);
- 14.String delete\_(string t);
- 15.string NumToVariable(string temp);
- 16.int Precedence(char c);
- 17.bool PrecedenceBiggerOrEqual(char c1, char c2);
- 18.String resultMath(string n1, string n2,char Operator);
19. vector<string> InfixToPostFix(string Infix);
- 20.string resultMathExpression(const vector<string>& PostFix, const vector<string>& num);
- 21.void ReadFileInputAndPrintResult(const char\* InputFile ,const char\* OutputFile);

## Chi tiết các hàm:

Hàm `isNegative` và hàm `absValue` để xử lý số âm khi thực hiện các phép toán.

1. Hàm `bool isNegative(string n)` kiểm tra chuỗi `n` có là số âm hay không:

- Nếu kí tự đầu của chuỗi là kí tự '-' thì trả về `true`, ngược lại trả lại `false`.

2. Hàm `string absValue(string n)` trả về giá trị tuyệt đối của 1 số:

- Nếu là số âm thì trả về chuỗi khi loại bỏ kí tự đầu tiên (bằng hàm `substr`).

3. Hàm `string Addition(string n1, string n2)`, trả về kết quả khi cộng 2 số nguyên lớn:

- Vì cộng 2 số nguyên lớn nên ta cần chuyển về chuỗi để xử lý:
- Xét các trường hợp:
  - Nếu `n1` là số âm, `n2` là số dương thì ta quy về hiệu của `n2` với trị tuyệt đối của `n1`.
  - Nếu `n1` là số dương, `n2` là số âm thì ta quy về hiệu của `n1` với trị tuyệt đối của `n2`.
  - Nếu cả `n1`, `n2` đều là số âm thì ta cộng trị tuyệt đối của `n1` và `n2` sau đó thêm '-' vào đầu chuỗi kết quả khi cộng 2 chuỗi dương.
- Cộng 2 chuỗi dương:
  - Kiểm tra nhằm đảm bảo độ dài chuỗi của `n1` lớn hơn hoặc bằng `n2`, nếu nhỏ hơn thì ta swap `n1` và `n2` để xử lý tiếp tục.
  - Đảo ngược chuỗi `n1` và `n2` để dễ cộng hơn, tính từ hàng đơn vị.
  - Cộng từng cặp chữ số của `n1` và `n2`, kèm theo giá trị nhớ `temp`, lấy kết quả thu được chia lấy dư cho 10 và thêm vào chuỗi kết quả.
  - Nếu tổng  $\geq 10$ , thì nhớ `temp = 1` cho lần cộng sau.
  - Cộng phần còn lại của `n1` nếu dài hơn, cộng chữ số của `n1` với `temp` cho đến khi hết chuỗi 1
  - Lấy kết quả thu được chia lấy dư cho 10 và thêm vào chuỗi kết quả, tương tự nếu tổng  $\geq 10$ , thì nhớ `temp = 1` cho lần cộng sau.
  - Sau khi cộng hết ở chuỗi 1 nếu `temp` bằng 1 thì ta thêm '1' vào cuối chuỗi (đây là phần tham khảo), đảo ngược chuỗi để trả về kết quả đúng, nếu ban đầu `n1` và `n2` là số âm thì ta thêm dấu trừ vào đầu chuỗi và trả về kết quả phép cộng.

Hàm `equal`, `Smaller`, `removeZero` để hỗ trợ để viết hàm `Subtraction` và các hàm sau đó:

4. Hàm `bool equal(string n1, string n2)` dùng để kiểm tra giá trị số của 2 chuỗi `n1` và `n2` có bằng hay không?

- Nếu n1 bằng n2 thì trả về true ngược lại trả về false.

5.Hàm bool Smaller(string n1,string n2) dùng để kiểm tra giá trị số của chuỗi n1 có nhỏ hơn chuỗi n2 không?

- Nếu n1 có ít chữ số hơn n2 thì trả về true, n1 có nhiều chữ số hơn n2 thì trả về false.
- Nếu có cùng độ dài thì dùng toán tử < để so sánh lần lượt theo từng kí tự.

6.Hàm removeZero(string n) xóa số 0 dư ở đầu chuỗi:

- Biến pos dùng để tìm vị trí ký tự đầu tiên khác '0'.
- Vòng lặp tăng pos đến khi gặp ký tự khác '0' hoặc hết chuỗi.
- Nếu toàn bộ chuỗi chỉ chứa số 0 (ví dụ: "0000"), thì trả về "0" duy nhất.
- Nếu có số khác 0, trả về phần chuỗi từ ký tự khác 0 đầu tiên trở đi, tức là bỏ hết các số 0 ở đầu. (bằng hàm substr).

7.Hàm string Subtraction(string n1, string n2) , trả về kết quả trừ 2 số nguyên lớn dưới dạng chuỗi.

- Xử lý số âm:
  - Nếu n1 âm và n2 dương thì ta quy về tổng của n1 và giá trị âm của n2
  - Nếu n1 dương và n2 âm thì ta quy về tổng của n1 và trị tuyệt đối của n2
  - Nếu cả n1 và n2 đều âm thì ta quy về hiệu của trị tuyệt đối của n2 với trị tuyệt đối của n1.
- Trừ 2 số dương:
  - Nếu n1 và n2 bằng nhau (hàm Equal) → trả về "0"
  - Nếu n1 < n2 (hàm Smaller) thì đổi chỗ n1, n2 để luôn trừ số lớn hơn cho số nhỏ hơn → kết quả sẽ mang dấu âm (flagNeg = true).
  - Ta đảo ngược chuỗi để dễ tính, tính từ hàng đơn vị.
  - Trừ từng cặp chữ số từ phải sang trái, lấy n1 trừ cho n2 ta quy ước n1 lớn hơn n2 (chữ số n1 trừ n2 và trừ thêm cho temp).
  - (1)-Nếu kết quả < 0 thì mượn 1, tức kết quả là cộng thêm 10 và cho temp = 1, lấy kết quả thu được chia lấy dư cho 10 và thêm vào chuỗi kết quả.
  - Tương tự nếu vẫn còn số lượng chữ số của chuỗi 1 vẫn còn dư nhưng ta chỉ trừ chữ số của n1 cho thêm, thực hiện tương tự bước (1).
  - Ta cần đảo ngược chuỗi lại để đúng thứ tự , xóa số 0 vô nghĩa đầu chuỗi bằng hàm removeZero.
  - Nếu trước đó xác định là âm (flagNeg = true) → thêm dấu '-'.

8.Hàm string MultiplicationByDigit(string n1,char n2) là hàm nhân 1 số nguyên lớn cho 1 số có 1 chữ số (hàm hỗ trợ cho hàm nhân 2 số nguyên lớn):

- Nhân lần lượt (từ chữ số cuối đến chữ số đầu) của n1 với n2 cộng thêm cho biến nhớ temp.
- Lấy kết quả thu được chia lấy dư cho 10 và thêm vào chuỗi kết quả, và cho giá trị temp chia đi 10 để cộng vào lần tiếp theo.
- Sau khi nhân đến cuối temp vẫn còn giá trị (temp khác 0) hoặc tại lần nhân cuối kết quả từ phép nhân lớn hoặc bằng 10 thì ta thêm temp vào cuối chuỗi kết quả.
- Đảo ngược chuỗi vừa thu được để đúng thứ tự và trả về kết quả.

9.Hàm string Multiplication(string n1, string n2) trả về kết quả khi nhân 2 số nguyên lớn dưới dạng chuỗi.

- Xử lý số âm:
    - Nếu 1 trong 2 số n1 hoặc n2 là số âm thì ta thêm dấu trừ vào cuối kết quả khi nhân 2 số nguyên dương với nhau.
    - Nếu 2 số cùng âm thì quy về nhân 2 số dương.
  - Nhân 2 số dương:
    - Lấy trị tuyệt đối của n1 và n2 (qua hàm absValue).
    - Đảo ngược n2 và nhân lần lượt từng chữ số từ đầu đến cuối với n1 (MultiplicationByDigit)
    - Kết quả thu được ta thêm các chữ số '0' tương ứng sau đó cộng nó với tổng các kết quả -trước đó bằng hàm (Addition).
- Vd: (123 x 12 => Đầu tiên ta nhân 123 với 2 (2 là số đầu tiên của n2 sau khi đảo ngược chuỗi) thu được 246 (vì đây là phép nhân đầu tiên nên tổng là 246) , sau đó nhân 123 với 1 thu được 123 nhưng 1 ở đây là hàng chục nên ta thêm 1 số 0 (tương tự nếu là hàng trăm thì thêm 2 số 0) => 1230 cộng với tổng của giá trị trước đó là 246 thu được 1476).
- Ta xóa các 0 vô nghĩa bằng hàm removeZero, kiểm tra xem có rơi vào trường hợp xử lý số âm hay không có thì thêm dấu trừ vào đầu chuỗi và trả về kết quả.

10. string Division(string n1, string n2) là hàm chia 1 số nguyên lớn cho 1 số nguyên lớn khác dưới dạng chuỗi:

- Xử lý số âm:
  - Nếu 1 trong 2 số n1 hoặc n2 là số âm thì ta thêm dấu trừ vào cuối kết quả khi chia 2 số nguyên dương với nhau.
  - Nếu 2 số cùng âm thì quy về chia 2 số dương.
- Chia 2 số dương:
  - Lấy trị tuyệt đối của n1 và n2 (qua hàm absValue).
  - Kiểm tra nếu n2 là 0 thì trả về rỗng (chia 0 => vô nghĩa).

-Cho 1 biến temp, ta thêm từng số của n1 vào để tìm được số chia phù hợp:

-Xử lý các trường hợp:

+Nếu kích thước của temp (số để chia) nhỏ hơn số chia thì ta vẫn chưa thể chia, ta sẽ thêm '0' vào cuối kết quả và tiếp tục thêm n1 để tìm được temp phù hợp.

+Nếu temp là 0 thì vẫn không chia được, ta cho temp về rỗng và thêm n1 vào cuối để tìm temp phù hợp, thêm '0' vào cuối kết quả. (cho temp về rỗng vì nếu không đem về rỗng thì vd: '01' khi chia cho 1 số nào đó thì các bước sau sẽ bị xử lý sai).

+Nếu temp bằng giá trị của n2(hàm equal) thì thêm '1' vào cuối kết quả và cho temp về rỗng.

-Khi đã tìm được temp phù hợp (đảm bảo temp lớn hơn n2) thì:

+Duyệt từ 9 -> 1 để tìm chữ số lớn nhất sao cho đảm bảo tích của chữ số đó nhỏ hơn hoặc bằng temp (tính tích thông qua hàm Multiplication).

+Ta cần lưu lại chữ số phù hợp và tích khi nhân số đó cho n2(bestMul).

+Ta cần cập nhật temp bằng hiệu của temp với bestMul, thêm chữ số phù hợp tìm được vào cuối kết quả.

-Sau khi đã duyệt qua tất cả chữ số của n1 thì xóa phần 0 vô nghĩa ở đầu kết quả thông qua hàm removeZero, xét xem có rơi vào trường hợp xử lý số âm hay không, nếu có thêm dấu trừ vào đầu kết quả và trả về kết quả phù hợp.

#### 11.Hàm bool IsOperator(char a) dùng để kiểm tra kí tự đó có là 1 toán tử hay không?

- Nếu a là +, -, \*, / thì trả về true, ngược lại trả về false.

#### 12.Hàm bool IsNum(char a) dùng để kiểm tra kí tự đó có là 1 số hay không?

- Nếu a >= '0' và <= '9' thì trả về true ngược lại trả về false.

#### 13.Hàm bool checkError(string n) kiểm tra biểu thức có là 1 biểu thức lỗi hay không?

- Các trường hợp lỗi như:

-Thiếu hoặc dư dấu (đóng ngoặc hoặc mở ngoặc).

VD: (2 + 3, (-123 \* 45, (24 - 12))

-Dư hoặc thiếu toán tử.

VD: 2 + (3 \* 6) -, 15 + 12 \*, 6512312 3123123

-Giữa 2 số phải là 1 toán tử (không có trường hợp 2 toán tử hay trống):

VD: 2 \* - 3, 56 12, 123123 +-\* / 12

- Ý tưởng:

-Ta cần tạo 1 stack nếu là dấu mở ngoặc thì ta thêm 1 giá trị vào stack, nếu là dấu đóng ngoặc thì ta bỏ 1 giá trị ra khỏi stack (sau khi duyệt hết biểu thức nếu stack rỗng

tức là biểu thức đúng). (Nếu 1 stack đang rỗng mà gặp dấu đóng ngoặc tức là biểu thức sai)

-Ngoài ra tạo thêm 1 vector để lưu, để ý ta thấy để 1 biểu thức đúng thì số lượng số xuất hiện trong biểu thức đảm bảo lớn hơn số lượng toán tử 1 phần tử, và giữa 2 số là 1 toán tử => có tính chất so le.

-Ta triển khai như sau nếu nhận diện là 1 số (tính cả số âm và dấu trừ của dấu âm không xem là 1 toán tử) thì thêm 1 giá trị dương bất kì vào vector (lấy là 2), nếu nhận diện là toán tử và đó không là dấu trừ của số âm thì ta thêm 1 giá trị âm đối với số dương vừa này (lấy là -2) vào vector.

-Ta cần duyệt qua từng phần tử của vector:

-Có tổng là 2 thì biểu thức đúng nếu tổng khác 2 thì biểu thức sai, trong quá trình duyệt ta cần đảm bảo các phần tử có tính chất so le nếu phần tử này là âm thì phần tử sau là dương và ngược lại.

VD:

$$(1 + 2) - (-2) * 3$$

-Xét stack ta sẽ thêm và xóa tổng cộng 2 lần sau cùng stack true => biểu thức đang đúng.

-Xét vector sau khi duyệt ta thu được: 2 -2 2 -2 2 -2 2 => đảm bảo có tính chất so le và tổng bằng 2 => biểu thức đúng

$$(1 - * 2)) 12$$

-Xét đến khi duyệt đến dấu đóng ngoặc thứ 2 thì stack đang rỗng nên => Biểu thức sai.

-Xét vector sau khi duyệt ta thu được: 2 -2 -2 2 2 => không đảm bảo tính chất so le và dù tổng bằng 2 nhưng cũng dẫn đến sai.

**14. String delete\_ (string t) hàm dùng để xóa khoảng trắng trong biểu thức( vì ta đã kiểm tra hàm có bị lỗi hay không nên xóa khoảng trắng để dễ xử lý).**

- Duyệt từng kí tự của biểu thức nếu khác khoảng trắng thì thêm vào => trả về chuỗi đã xóa khoảng trắng.

**15. string NumToVariable(string temp) ở hàm này ta sẽ chuyển các số thành biến:**

Vd:  $1+(2*3) \Rightarrow n0+(n1*n2)$  ,  $(-2)+3+4 \Rightarrow n0+n1+n2$

- Sau khi khóa khoảng trắng và kiểm tra lỗi thì để xác định là 1 số âm thì nếu kí tự đầu là dấu đóng ngoặc thì kí tự tiếp theo chính là dấu trừ và ta chỉ thêm biến vào các phần tử.( sử dụng flagNeg để xác định số âm không thêm dấu đóng hoặc mở ngoặc vào chuỗi kết quả)
- Ta cần duyệt và giữ lại các phần tử phù hợp như dấu đóng ngoặc , mở ngoặc , các toán tử (chuyển đúng các số âm trở thành biến)



- Sử dụng cờ flagNum để nhận diện trong quá trình tạo thành 1 số nên sẽ không thêm kí tự số vào vào chuỗi, khi gặp các kí tự kết thúc (như toán tử, dấu đóng ngoặc) thì sẽ thêm biến vào chuỗi mới và trả flagNum về false.

#### 16. Hàm int Precedence(char c) xác định độ ưu tiên toán tử:

- Nếu toán tử là nhân hoặc chia trả về 2, toán tử là cộng hoặc trừ trả về 1.
- Không là toán tử thì trả về 0.

#### 17. Hàm bool PrecedenceBiggerOrEqual(char c1, char c2) kiểm tra toán tử c1 có được ưu tiên hơn hoặc cùng cấp độ so với c2 hay không?

- Trả về Precedence(c1) >= Precedence(c2).

#### 18. Hàm String resultMath(string n1, string n2, char Operator)

- Dựa vào toán tử trả về kết quả phù hợp.
- Nếu toán tử là '+' thì trả về Addition(n1,n2).
- Nếu toán tử là '-' thì trả về Subtraction(n1,n2).
- Nếu toán tử là '\*' thì trả về Multiplication(n1,n2).
- Nếu toán tử là '/' thì trả về Division(n1,n2).
- Không phải là toán tử thì trả về rỗng.

#### 19. Hàm vector<string> InfixToPostFix(string Infix) chuyển từ biểu thức trung tố (Infix) dạng chuỗi sang biểu thức hậu tố (PostFix) lưu dưới dạng vector.

VD: Infix :  $n0+n1*n2 \rightarrow$  Postfix : [ n0 , n1 , n2 , \* , + ]

Infix :  $((n0+n1)*n3-n4)/n5 \rightarrow$  Postfix : [ n1 , n2 , + , n3 , \* , n4 , - , n5 , / ]

- Ta duyệt từng kí tự trong Infix:
  - Khi gặp kí tự n (n đại diện cho biến dùng để lưu 1 số) tức là 1 số ta đọc số nguyên ở sau nó để biết được index, dùng hàm IsNum để đọc các chữ số sau n, sau khi đọc sẽ đẩy chuỗi số -vào hậu tố (Postfix) tức là thêm 1 số vào Postfix.
  - Khi gặp dấu mở ngoặc thì ta sẽ đưa vào stack.
  - Khi gặp dấu đóng ngoặc, rút các toán tử đã thêm vào stack và đưa chúng vào hậu tố (Postfix) cho đến khi đỉnh của stack là dấu mở ngoặc, sau đó ta loại bỏ dấu mở ngoặc trong stack.
  - Khi gặp toán tử (IsOperator) ,đầu tiên đảm bảo trong stack không rỗng để kiểm tra s.top() (!s.empty), sau đó ta so sánh độ ưu tiên toán tử đang xét hiện tại với toán tử ở đỉnh của stack, nếu toán tử trên stack có độ ưu tiên cao hơn hoặc bằng toán tử đang xét thì lấy toán tử đó ra khỏi stack và thêm vào hậu tố Postfix, ngoài ra nếu đỉnh của stack là dấu mở ngoặc thì ta không thể lấy dấu mở ngoặc ra khỏi stack vì dấu mở ngoặc chỉ bị lấy khi gặp dấu đóng ngoặc.
- Sau khi duyệt xong ta đẩy nốt các toán tử còn lại trong stack vào hậu tố.

20.Hàm `string resultMathExpression(const vector<string>& PostFix, const vector<string>& num)` :

- Từ vector PostFix hậu tố, vector num dùng để lưu trữ các số của biểu thức ban đầu (vector num lưu các số của biểu thức trong hàm thứ 21)
- Dùng stack để tính các biểu thức hậu tố, duyệt qua từng phần tử của vector PostFix nếu là biến ta lưu giá trị index và đẩy vào stack.
- Khi gặp toán tử ta lấy 2 phần tử ngoài của stack thực hiện phép toán với nhau qua hàm resultMath tính toán được kết quả của phép toán, nếu kết quả là chuỗi rỗng tức là đã gặp trường hợp chia cho 0 (vô lý) nên hàm trả về chuỗi rỗng, nếu không là chuỗi rỗng ta đẩy vào stack và duyệt cho đến hết phần tử của PostFix.
- Sau cùng trả về phần tử đỉnh của stack tức là kết quả sau cùng.

21.Hàm `void ReadFileInputAndPrintResult(const char* InputFile ,const char* OutputFile)` ở hàm này ta thực hiện đọc file, lưu các số vào vector num để thực hiện các phép tính , từ các hàm trước đó thu được kết quả cuối cùng của biểu thức và xuất ra file output và cả terminal:

- Mở file output và file input
- Đọc xử lý lần lượt các biểu thức:
  - Kiểm tra biểu thức có lỗi hay không thông qua hàm `checkError(t)`, nếu lỗi thì xuất ra -terminal và file Output Error rồi tiếp tục xử lý biểu thức tiếp theo.
  - Sau khi kiểm tra ta xóa khoảng trắng của biểu thức bằng hàm `delete_` để dễ xử lý hơn.
  - Vì đã xóa khoảng trắng nên nhận diện số âm bằng cách sau dấu ngoặc mở '(' là dấu âm '-' từ đó ta đọc được số âm, cùng với đó là các số dương, khi gặp ')', toán tử thì kết thúc đọc 1 số, các số thu được lưu vào vector num dùng để sử dụng khi tính toán biểu thức.
  - Tạo chuỗi Infix bằng hàm `NumToVariable` để chuyển các số trở thành biến để xử lý.
  - Tạo vector PostFix có kiểu dữ liệu string bằng hàm `InfixToPostFix` để thu được PostFix.
  - Sau cùng ta thu được kết quả của biểu thức thông qua hàm `resultMathExpression` với 2 tham số truyền vào là vector PostFix và vector Num lưu các số.
  - Nếu kết quả là chuỗi rỗng tức là xảy ra trường hợp chia cho 0 (vô lý) in ra Error ở terminal và file Output.
  - Nếu kết quả không là chuỗi rỗng in kết quả thu được ở terminal và fileOutput.
- Tiếp tục xử lý cho đến hết các biểu thức nằm trong file input, đóng file Input và file Output.

Hình ảnh tham khảo:

(1)

❌ Vấn đề tiềm ẩn hoặc điểm cần cải thiện:

1. Chưa kiểm tra `temp` sau vòng lặp cuối

Nếu sau vòng lặp mà `temp == 1` (vẫn còn nhớ), thì cần thêm `'1'` vào đầu kết quả.

✅ Cách sửa:

cppSao chépChỉnh sửa

```
if (temp == 1) {
    sum.push_back('1');
}
```

→ Câu này phải thêm trước khi `reverse(sum.begin(), sum.end())`.

(2)

Vậy là các phép so sánh đều đúng so với chuỗi

Đúng vậy, trong C++, các phép so sánh chuỗi (`<`, `>`, `==`, `!=`, `<=`, `>=`) hoạt động theo thứ tự từ điển (lexicographical order), nghĩa là:

- So sánh từng ký tự từ trái sang phải.
- Ký tự nào có mã ASCII lớn hơn thì lớn hơn.
- Nếu hai chuỗi có cùng độ dài và không có ký tự khác nhau, thì chúng bằng nhau.

Với các chuỗi số (không có số 0 ở đầu), thứ tự từ điển giống với thứ tự số học.

Ví dụ:

cppSao chépChỉnh sửa

```
"123" < "124"    // đúng
"9" < "10"       // sai vì '9' > '1' theo mã ASCII (nên cần xử lý độ dài)
"0009" < "10"    // sai, nên bỏ số 0 đầu nếu muốn so sánh đúng số học
```

**Kết luận:**

- Nếu hai chuỗi có cùng độ dài và đại diện cho số không âm, thì phép so sánh chuỗi trong C++ cho kết quả đúng như so sánh số nguyên lớn.
- Nếu chuỗi không cùng độ dài, cần so sánh độ dài trước, hoặc thêm logic loại bỏ số 0 đầu chuỗi.

Bạn có muốn một hàm chuẩn hóa chuỗi số (loại bỏ số 0 đầu) trước khi so sánh không?

(3)

```
string Divisor(string n1, string n2)
{
    if (Smaller(n1, n2)) {
        return "0";
    }

    string result;
    string current;

    int idx = 0;

    while (idx < n1.size()) {
        current.push_back(n1[idx++]);

        // Xóa số 0 đầu nếu có
        while (current.size() > 1 && current[0] == '0') {
            current.erase(0, 1);
        }

        if (Smaller(current, n2)) {
            if (!result.empty()) result.push_back('0');
            continue;
        }

        char digit = DivisorStep(current, n2, n1.size() - idx);
        result.push_back(digit);
    }

    // Xóa số 0 đầu nếu có
    while (result.size() > 1 && result[0] == '0') {
        result.erase(0, 1);
    }

    return result;
}
```

(4)

## 8. Thay thế toán hạng trong biểu thức:

```
cpp
string indexAdds = to_string(indexAdd);
NewPostFix = PostFix1.substr(0, head) + 't' + indexAdds + PostFix1.substr(tail);
PostFix1 = NewPostFix;
```

→ Tức là: thay phần toán hạng + toán tử đã xử lý bằng kết quả trung gian `tX`.

## ✗ Tổng hợp lỗi cần sửa

Vấn đề	Dòng code lỗi	Cách sửa
Không gán giá trị cho <code>PostFix1</code>	<code>string PostFix1;</code>	<code>PostFix1 = PostFix;</code>
Sai điều kiện vòng lặp	<code>while (PostFix1.size() == 2)</code>	<code>while (PostFix1.length() &gt; 1)</code>
Sai điều kiện while khi tìm 'n' hoặc 't'	<code>!= 'n'</code>	

## ✓ Gợi ý đơn giản hóa

Bạn nên chuyển postfix thành vector token (chuỗi `n0`, `t1`, `+`, `*`, ...) và xử lý bằng stack sẽ rõ ràng, dễ debug hơn nhiều.

### ✓ Hàm đầy đủ:

```
cpp
#include <iostream>
#include <vector>
#include <string>
#include <stack>
using namespace std;

// Giả định có sẵn hàm này để xử lý phép tính chuỗi lớn
string resultMath(const string& a, const string& b, char op) {
    // Thay bằng logic xử lý chuỗi số lớn tương ứng
    if (op == '+') return to_string(stoll(a) + stoll(b));
    if (op == '-') return to_string(stoll(a) - stoll(b));
    if (op == '*') return to_string(stoll(a) * stoll(b));
    if (op == '/') return to_string(stoll(a) / stoll(b));
    return "0";
}

// Hàm tính toán biểu thức hậu tố từ vector token
string resultMathExpression(const vector<string>& PostFix, const vector<string>& num) {
    vector<string> temp;
    stack<string> stk;

    for (const string& token : PostFix) {
        if (token[0] == 'n' || token[0] == 't') {
            // Toán hạng
            if (token[0] == 'n') {
                int index = stoi(token.substr(1));
                stk.push(num[index]);
            } else { // 't'
                int index = stoi(token.substr(1));
                stk.push(temp[index]);
            }
        } else {
            // Toán tử
            string b = stk.top(); stk.pop();
            string a = stk.top(); stk.pop();

            string res = resultMath(a, b, token[0]);
            temp.push_back(res);

            // Push lại kết quả dạng trung gian để dùng tiếp
            stk.push(res);
        }
    }

    return stk.top(); // Kết quả cuối cùng
}
```