TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG HỒ CHÍ MINH

Khoa Điện tử Viễn thông

BÁO CÁO CUỐI KỲ HÌNH THỨC BÀI TẬP LỚN

Học kỳ II

Năm học 2022 – 2023

Môn: Thực hành Cấu trúc dữ liệu và giải thuật

Giảng viên phụ trách: Trần Thị Điểm

Họ và tên sinh viên: Nguyễn Minh Tâm

Mã số sinh viên: 20200333

Lớp: 20 Nhung_TH

Đề bài: Cài đặt cây nhị phân để tính 1 giá trị biểu thức

Thành phố Hồ Chí Minh, ngày 15 tháng 04 năm 2023

MỤC LỤC

1. Tổng quan

1.1. Ý tưởng tổng quan

Về tổng quan thì để thực hiện bài tập lớn này em cần có hàm chuyển từ biểu thức thành cây nhị phân, thứ hai là hàm duyệt cây nhị phân để phục vụ cho hàm thứ ba sẽ làm hàm thực hiện tính toán cho cây nhị phân đó. Ý tưởng để xây dựng cây nhị phân là em sẽ xét độ ưu tiên để truyền các cây con vô trước các cây parents vô sau. Độ ưu tiên truyền vào sẽ là nhân chia trước, nếu có nhân chia thì trái phải, nếu không có nhân chia thì cộng trừ mà phải là từ trái trước sau đó phải sau. Nếu toán tử hiện tại là cộng trừ thì cần thiết phải có toán tử ở đỉnh của stack phải là cộng trừ vì ta luôn phải cần sự so sánh, tức là phải luôn có sự so sánh giữa toán tử hiện tại và toán tử được lưu ở stack để ta ưu tiên cái nào thực hiện trước ta sẽ push vào node lá và lưu ở stack operands. Với một toán tử được pop ra từ stack operators thì em sẽ pop ra hai toán hạng ở stack operands, gồm có toán hạng phải sẽ là toán hạng đỉnh của stack operands và toán hạng trái là toán hạng kế đỉnh của stack operands. Thì hàm ưu tiên được thực thi xong thì em sẽ chuyển qua hàm không được ưu tiên, ở hàm không được ưu tiên thì còn bao nhiêu toán tử đang lưu ở stack operators sẽ được pop vào chung với hai toán hạng của stack operands. Vì với mỗi toán tử không được ưu tiên hay không được pop ra thì ta sẽ luôn lấy ra được hai toán hạng tại vị trí tương ứng nên luôn đảm bảo phép tính của ta là đúng toán hạng nhưng chỉ khác vị trí ban đầu nếu ta đưa phương trình có nhân chia sau đó cộng trừ, còn về đều là nhân chia và cộng trừ thì sẽ đúng thứ tự từ trái sang phải. Ý tưởng duyệt cây như vậy xuất phát từ ý tưởng duyệt post order mà cô đã dạy trên lớp tức là em sẽ duyệt node trái xong đến node phải và node đang làm root để hình thành node lá mới. Sau khi hình thành cây nhị phân từ ý tưởng trên thì em sẽ duyệt cây để thực hiện tính. Ý tưởng duyệt cây cũng là duyệt theo trung tố là in order tức là em sẽ duyệt node con trái xong đến node root và node con phải sau đó gọi hàm tính và trả giá trị tại node đó và thay thế node đó bằng giá trị vừa tính được. Do ưu tiên nên rõ ràng cách tính như vậy là thoả sự ưu tiên khi tính toán trong toán học. Sau khi thực hiện tính thì em sẽ duyệt cây nhị phân mình tạo ra để in ra màn hình. Cũng là cách duyệt trung tố nhưng mà không cần đến phần tử trái nhát hay phải nhất mà em sẽ bắt đầu từ node root sau đó duyệt trái in ra rồi lại duyệt phải in ra. Cứ như thế em sẽ có được một cây nhị phân tính toán. Thêm vào đó code của em còn in ra kết quả duyệt bằng cả ba phương pháp, còn có phần dịnh dạng xử lý đối với trường hợp số có nhiều chữ số. Em sử dụng kiến thức về stack, stack có kiểu Last in First Out rất thuận tiện cho bài toán này, ngoài ra là các kiến thức cây nhị phân, duyệt cây và nhập xuất cơ bản trong C++. Em đã sử dụng thư viện string và cctyype để xử lý, chi tiết về các hàm và cách em xử lý sẽ được em đề cập kỹ ở phần sau. Sau đây là code khai báo thư viện

1.2. Code khai báo thư viện

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

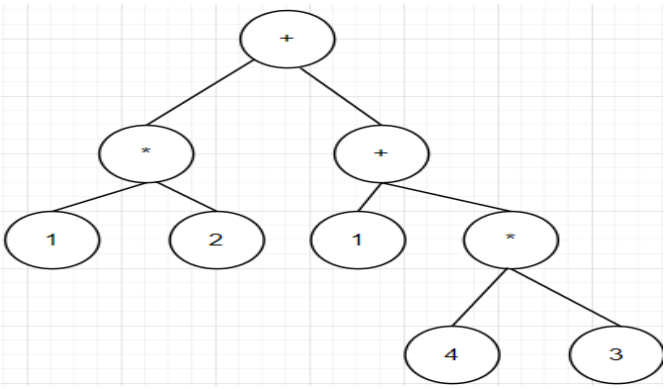
2. Cách tạo cây nhị phân

2.1. Ý tưởng

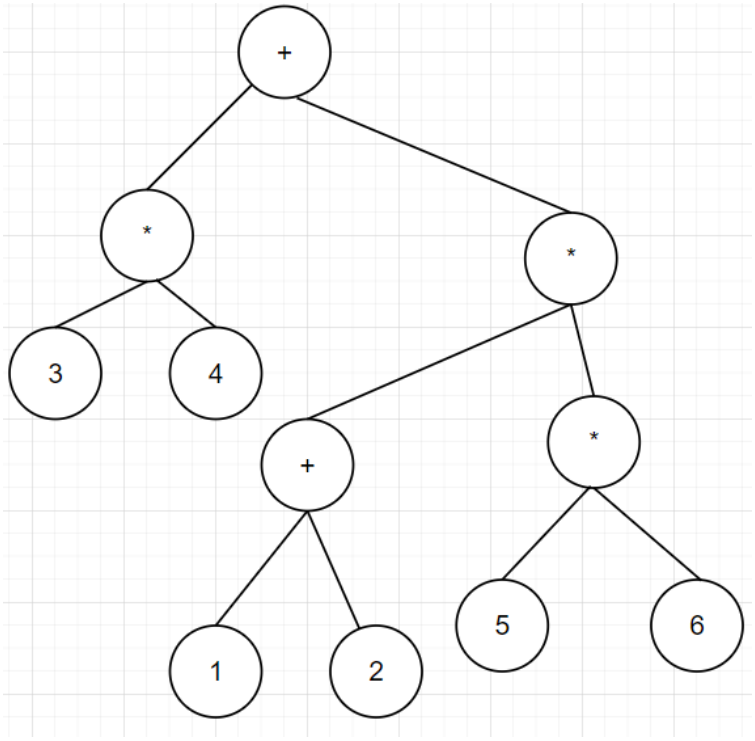
- Ý tưởng tổng quá: Duyệt biểu thức để hình thành cây thì em sẽ sử dụng cách duyệt là post order tức là với hai toán hạng đã có thì sẽ có một toán tử được thêm vào để hình thành một cây nhị phân. Và xét theo độ ưu tiên của các phép toán thì em sẽ có các cách xử lý khác nhau. Sau đây là phần trình bày các ý tưởng.
- Theo như ý tưởng em đã trình bày ở phần 1.1. thì em sẽ tạo ra một cây nhị phân bằng cách sử dụng 2 stack gồm 1 stack có tên là operators sẽ lưu toán tử, còn 1 stack nữa có tên là operands sẽ lưu lại toán hạng. Ban đầu thì khi người dùng nhập một biểu thức từ bàn phím vô thì em sẽ tiến hành duyệt từ đầu đến cuối, mỗi lần duyệt là mỗi lần em sẽ tiến hành kiểm tra điều kiện. Do dữ liệu của em là string nên em rất thoải mái thao tác khi có thể kiểm tra đến khi hết phần tử nằm trong chuỗi để có thể giúp bài toán của em linh động hơn. Việc sử dụng string kết hợp với thư viện cctype giúp em có thể kiểm soát dữ liệu đầu vào của mình có phải là số hay không bằng cách duyệt từ đầu đến cuối chuỗi, mỗi lần duyệt thì em sẽ có biến tạm tên "exp" sẽ gán bằng vị trí hiện tại của chuỗi. Em sẽ sử dụng hàm "isdigit" của thư viện cctype để kiểm tra xem ký tự hiện tại có phải là số hay không, nếu mà thoả là số thì em sẽ có biến tạm tên là num, biến này nó sẽ bằng chính giá trị "exp" lúc này đang là kiểu char sẽ trừ cho ký tự '0' thì ta có thể ra giá trị nguyên của chúng. Giải thích thì do C++ các ký tự được lưu trữ dưới dạng số nguyên, khi ta trừ ký tự cho một ký tự khác thì ta đang trừ đi giá trị nguyên của chúng. Ví dụ '0' có giá trị là 48 trong bảng mã ASCII thì khi ta có giá trị từ '1' đến '9' thì sẽ có giá trị trong bảng mã ASCII là từ 49 đến 57 khi ta thực hiện trừ ví dụ: '1' – '0' thì sẽ là 49 – 48 và bằng 1 là số nguyên 1, tương tự là 50 – 48 đối với số 2, 51 – 48 đối với số 3, 52 – 48 là số 4, 53 – 48 là số 5, 54 – 48 là số 6, 55 – 48 là số 7, 56 – 48 là số 8 và cuối cùng là 57 – 48 đối với 9. Bởi vì duyệt từng phần tử nên sau mỗi lần vậy em sẽ có được một giá trị số, nếu số chữ số có nhiều hơn 1 thì sẽ thực hiện vòng lặp là em sẽ lấy số hiện tai nhân cho 10 sau đó cộng với phần đã chuyển đổi sang số nguyên. Em sẽ có vòng lặp while là nếu còn sót ký tự là chữ số đối với chuỗi nhập thì sẽ thực hiện liên tục là nhân 10 và cộng với phần chuyển đổi nguyên đến khi nào hết ký tự là số thì kết thúc. Giải thích lý do nhân cho 10 thì ta biết rõ ràng là mỗi khi duyệt xong một vị trí ký tự là số thì đến với vị trí

tiếp theo, thì chắc chắn số đó sẽ ở vị trí nhỏ hơn số vừa xét nên nhân 10 để đảm bảo sự đúng đắn. Ví dụ ta có số 123 thì lần duyệt đầu ta sẽ tạo ra num có giá trị 1, sau đó vòng while thực hiện lệnh ta sẽ lấy ra được số 2 bằng việc "num = num*10+('2'-'0')", ta nhân 10 vì ta biết số 2 chắc chắn ở sau số 1 và khi đó số này sẽ có hai chữ số. Sau đó vẫn chưa hết ký tự là số nên ta cần nhân tiếp với 10 vì ta biết vẫn còn ký tự là số tức là số này không chỉ có hai chữ số, nhân với 10 để tạo ra số có ba chữ số và như trên ta cũng có lệnh "num = num*10+('2'-'0')" để tạo ra số 123. Sau khi có được số - tức là toán tử của biểu thức thì em sẽ đẩy vào stack và lưu ở stack operands là phần toán tử của biểu thức.

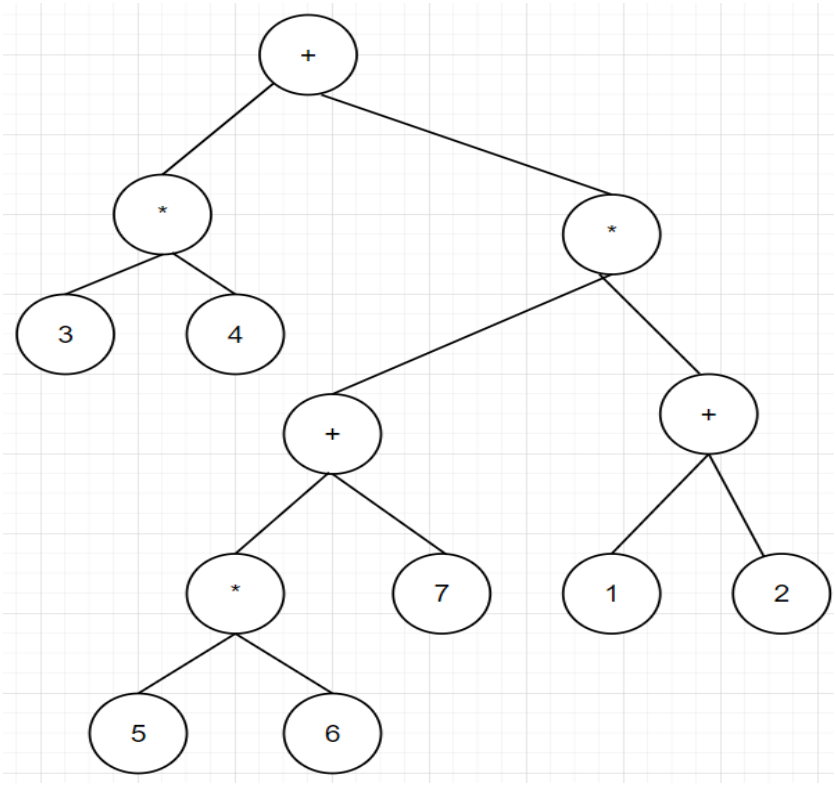
- Sau khi kiểm tra hết ký tự là số, khi đụng phải ký tự không phải số thì em sẽ tiến hành thao tác trên toán tử, tức là các phần tử + - * /. Ý tưởng ở đây em sẽ có một hàm kiểm tra xem ký tự nhập vào có phải là + - * / hay không thì mới thực thi vì trong biểu thức ta còn xét đến dấu ngoặc tròn thể hiện mức độ ưu tiên của phép toán nên để xét độ ưu tiên thì có dấu ngoặc sẽ ưu tiên cao nhất, khi có dấu ngoặc thì em sẽ không thể vội bỏ các phần tử + - * / vào stack được mà theo trình tự là bỏ dấu ngoặc vô stack sau đó mới xét đến độ ưu tiên của */ sau đó sẽ xét đến độ ưu tiên bên trái hơn bên phải. Việc ưu tiên như vậy để có thể bỏ vào cấu trúc cây một cách đúng đắn. Như ta biết cây nhị phân sẽ đi từ nốt nhỏ nhất tính lên trên tầng cao hơn cho đến khi đụng root thì ngưng. Sơ lược ý tưởng là như vậy, về chi tiết thì đầu tiên em sẽ kiểm tra xem có phải dấu ngoặc mở không, nếu đúng dấu mở ngoặc thì sẽ tiến hành bỏ dấu mở ngoặc vào stack và tiếp tục duyệt các phần tử tiếp theo cho đến khi gặp dấu đóng ngoặc thì em sẽ tiến hành thực thi là bỏ vào các số đã thêm ở stack vào bên con trỏ trái phải và các toán tử sẽ lưu vào node cha. Lý do em bỏ dấu mở ngoặc vào stack vì em muốn ưu tiên cao nhất đối với dấu mở ngoặc, khi bỏ vào dấu mở ngoặc thì em đảm bảo sẽ không thể nào có phép tính + - * / ở ngoài ngoặc diễn ra, nó sẽ được thực thi khi nào mà có dấu đóng ngoặc. Cơ chế stack là Last in First Out nên để đảm bảo luôn kiểm tra được điều kiện ưu tiên thì em sẽ kiểm tra trên cơ sở TOP của stack. Trong bài toán nếu có nhân chia sau đó mới đến dấu ngoặc thì em sẽ tiến hành nhóm dấu nhân chia cộng trừ thành một nhóm riêng, nó sẽ là left của nguyên một cây cha lớn, còn right của cây cha lớn sẽ là các phép tính của dấu ngoặc. Tức là không nhất thiết em sẽ luôn đặt các phép toán có độ ưu tiên ở bên trái mà em sẽ ưu tiên nhóm các phép tính có độ ưu tiên cao thành một nhóm nếu trong bài chúng ở sau các toán tử cũng có độ ưu tiên cao. Ví dụ nếu em có hàm sau: 1*2+(3*4+1) thì em sẽ có cấu trúc cây là:



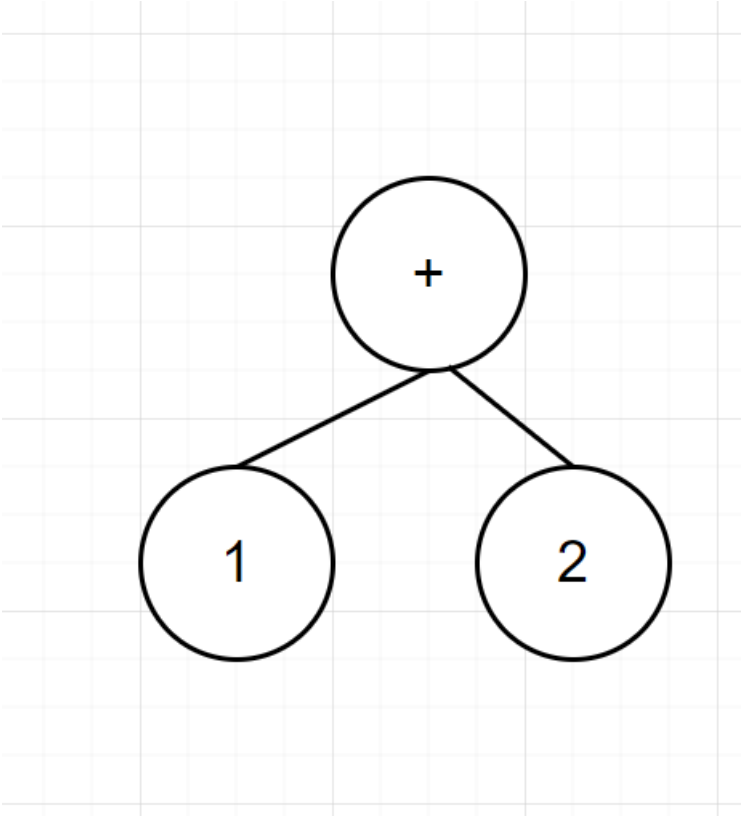
Còn trong bài toán nếu có tồn tại hai cặp ngoặc và nằm bên phải của toán tử nhân chia thì sẽ nhóm lại thành một nhánh con của nhánh lớn. Chẳng hạn em có ví dụ sau: (3*4)+(1+2)*(5*6) thì em sẽ có cấu trúc cây là:



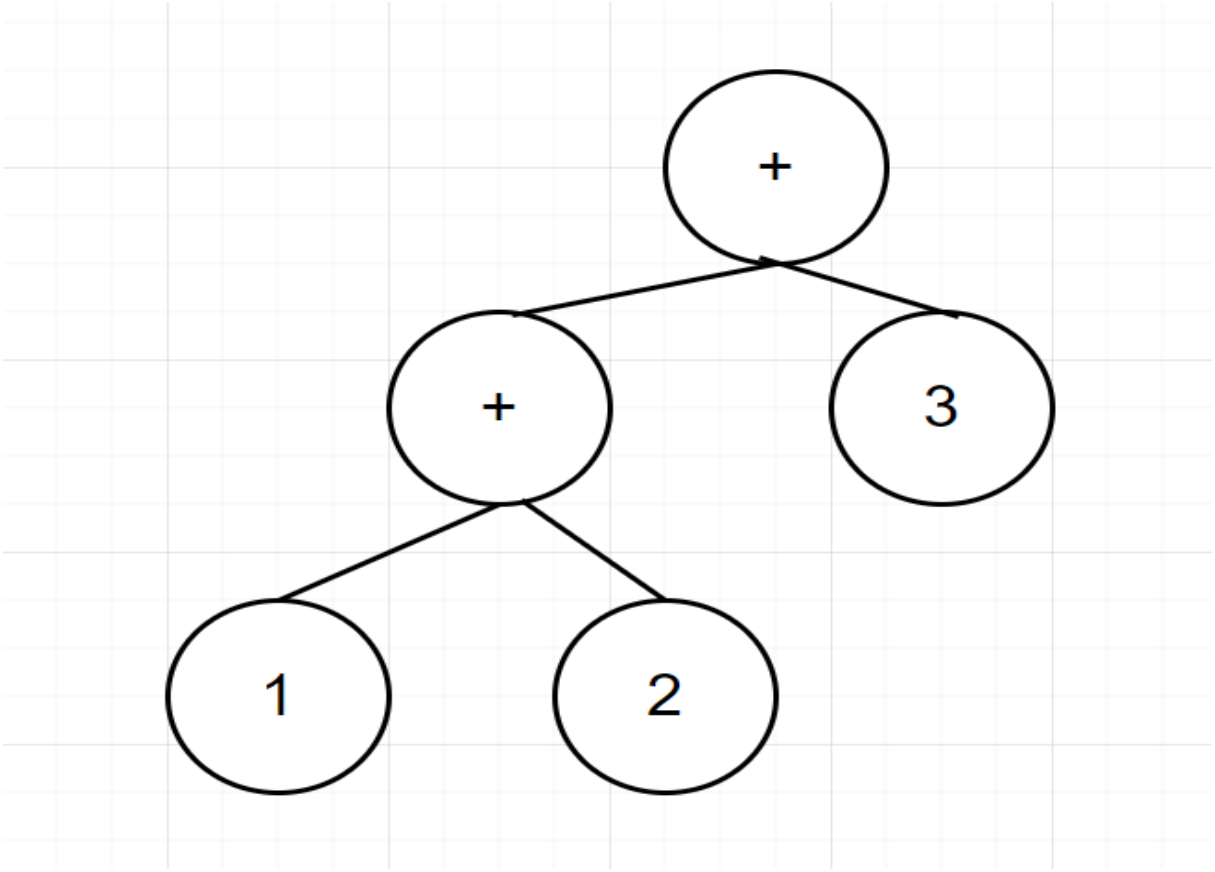
Đối với hàm phức tạp hơn thì em cũng sẽ làm tương tự, ví dụ: 3*4+(5*6+7)*(1+2) thì em sẽ có:



Tóm lại thì em sẽ có thứ tự ưu tiên là nhân chia ưu tiên cao nhất, sau đó là cộng trừ, và cuối cùng là trái phải. Nếu trong phương trình có tồn tại dấu ngoặc đơn mà chương trình quét đến trước tiên thì nó sẽ ưu tiên thực thi hàm có dấu ngoặc đơn trước nhưng nếu hàm có dấu ngoặc đơn rơi vào các nhóm mà em đã liệt kê là nằm ở sau * / và bên phải thì nó sẽ ưu tiên nhóm lại thành một nhóm, có dấu có độ ưu tiên thấp nhất làm nốt cha. Em đặt ra như vậy theo đúng quy tắc toán học là khi gặp dấu ngoặc đơn thì ta sẽ thực hiện nhóm các phép tính lại thành một nhóm thì em sẽ nhóm lại các phép tính trong dấu ngoặc đơn để thành một node con trong một node cha của phần tử có độ ưu tiên thấp trong bài toán. Ý tưởng về dấu ngoặc đơn và độ ưu tiên là như vậy. Nếu lúc đầu duyệt chuỗi đã bỏ phần tử số đầu tiên rồi mà không gặp dấu ngoặc đơn thì hàm sẽ tiến hành bỏ vào các dấu + - * / phù hợp vào stack nếu các ký tự này hợp lệ. Nếu nhập một hàm vô nghĩa thì chương trình báo lỗi và phải nhập lại. Sau khi có được toán tử đầu tiên thì ta sẽ tiếp tục bỏ vào stack operands toán hạng thứ hai và sau đó sẽ tiến hành kiểm tra xem có đúng thứ tự là phép tính hiện tại rơi vào trường hợp là + - * / hay không và kiểm tra thêm một điều kiện nữa là có phải nhân chia hay không vì nhân chia có độ ưu tiên cao hơn nên ta cần kiểm tra xem có phải nhân chia hay không. Nếu không phải * / thì ta sẽ thực hiện việc khai báo biến tạm trái có tên là temp_left để lưu toán hạng đầu tiên bỏ vào stack, tiếp theo là biến tạm phải có tên là temp_right để lưu toán hạng còn lại từ stack. Mỗi lần lấy dữ liệu ra thì ta sẽ bỏ luôn số đó ở trong stack để đảm bảo sự đúng đắn ý tưởng là last in first out của stack. Sau đó em sẽ tạo một node mới để lưu lại left và right cùng với đó là node cha của cây nhị phân đầu. Ví dụ như em có hàm sau 1+2+3 thì lần đầu duyệt hoàn tất em sẽ có cây khởi tạo sau:



Sau khi khởi tạo cây đó thì em sẽ quay lên đọc tiếp lúc này ký tự tiếp theo em đọc là ký tự của toán tử cộng, dấu cộng này có độ ưu tiên thấp nhất sẽ làm root cho cây nhị phân tính toán của em. Do operands đã rỗng nên sẽ không có bỏ vào phần tử tuy nhiên nó sẽ tạo ra một nhánh mới gồm có node bên trái và node bên phải sau đó nó sẽ push vào operands. Sau khi duyệt dấu + xong thì em sẽ duyệt tiếp đến số 3 nó không phải toán tử nên em sẽ duyệt cho đến phần cuối là bỏ vào stack như là phần tử đầu tiên, đỉnh của cây nhị phân bằng việc "return operands.top(), tức là sẽ trả về con trỏ để trỏ đến nood root của cây biểu thức, tức là trỏ dấu cộng đang làm root. Lúc này dấu cộng ở root sẽ nhân cây nhị phân ở hình trên là left của nó còn 3 sẽ là right của nó. Do vẫn chưa thực hiện tính toán nên em tạm gọi cây node bên trái của cây root là tổng 1+2, tổng ấy sẽ là nhánh trái của cây root còn 3 sẽ là nhánh phải của cây root. Từ những phân tích trên ta sẽ có cây nhị phân sau khi hoàn tất sẽ là:



- Và đó là hoàn tất quá trình biến đổi từ biểu thức thành cây nhị phân để thực hiện tính toán. Tổng quan tóm tắt lại các bước tiến hành sẽ có:
    - Bước 0: Duyệt từ đầu mảng
    - Bước 1: Duyệt kiểm tra xem có ký tự số không, sau đó tới bước 2, nếu không phải số thì ta sẽ tới bước 4
    - Bước 2: Nếu ký tự là số thì ta sẽ tiến hành chuyển đổi thành số, nếu số chữ số lớn hơn 1 thì ta sẽ thực hiện nhân 10 cộng với phần chuyển đổi sang số nguyên cho đến khi hết ký tự là số sau đó tới bước 3
    - Bước 3: Ta sẽ đẩy số vừa biến đổi được vào stack Operands

- Bước 4: Ta sẽ tiếp tục duyệt và kiểm tra xem có phải là ngoặc đơn hay không, nếu là ngoặc đơn thì ta sẽ thực hiện bước 4.1. còn không phải ngoặc đơn thì thực hiện bước 5
    - Bước 4.1: Ta sẽ lưu lại dấu mở ngoặc đơn vào stack operators để không phải thực hiện bước 5. Sau đó ta sẽ duyệt tiếp vị trí tiếp theo và thực hiện bước 1. Sau khi kết thúc bước kiểm tra thì ta sẽ đến bước 5
- Bước 5: Ta sẽ duyệt xem ký tự đã nhập vào có phải là toán tử công trừ nhân chia hay không. Nếu đúng cộng trừ nhân chia thì ta sẽ kiểm tra xem stack operators có đang trống không. Nếu stack operators trống thì ta sẽ nhét toán tử đó vào và sau đó quay lại bước 1. Còn nếu stack operators đang có toán tử thì sẽ sang bước 6
- Bước 6: Ta sẽ kiểm tra xem toán tử đỉnh có phải của stack operators có phải là nhân chia hay không. Nếu là nhân chia thì chuyển sang bước 7, còn nếu không phải nhân chia thì xét tiếp toán tử hiện tại có phải là cộng trừ không, nếu là cộng trừ thì chuyển sang bước 7 còng nếu không là cộng trừ thì ta push toán tử hiện tại vào stack operators.
- Bước 7: Tiến hành tạo một node lá mới cho cây nhị phân. Ta sẽ khởi tạo đầu tiên là biến right sẽ được pop ra từ phần tử đỉnh của operands tức là phần tử vừa được thêm vào, sau đó là pop left, là số hạng nằm kế bên số hạng ở đỉnh operands ra để tiến hành tạo thành một node lá. Sau đó ta cũng pop ra toán tử top tức toán tử vừa thêm vào ra để hình thành một node lá mới. Sau khi hình thành xong ta push vào stack operands
- Bước 8: Ta kiểm tra xem có nút đóng ngoặc đơn hay không. Vì là nhóm lại nên là nếu có mở ngoặc thì thao tác từ bước 4 đến bước 7 là ta đang nhóm các cây con, tức là sinh ra cây con mới rồi nhóm chúng lại. Dấu ngoặc đóng sẽ là báo hiệu kết thúc cho nhóm, thì ta phải cần hoàn thành bằng cách pop ra từ top một toán tử và lần lượt pop ra từ đỉnh và từ kề đỉnh của stack operands ra để tiến hành push vào stack operands hình thành nên một node lá. Ta sẽ kết thúc nếu gặp dấu mở ngoặc mà ta đã thêm vào, tức là còn dấu mở ngoặc là ta sẽ còn xử lý. Nếu không có dấu mở ngoặc thì vào bước 9
- Bước 9. Sau khi ưu tiên trái, sau khi ưu tiên nhân chia thì còn bao nhiêu toán tử và toán hạng ta sẽ tiến hành theo thứ tự đã lần lượt thêm vào. Ta sẽ lặp lại quá trình trên cho đến khi nào stack operators không còn toán tử nào thì ngưng. Do ta thêm là lần lượt từ trái sang phải nên khi lấy ra ta sẽ có thứ tự ngược lại nhưng hình thành cây sẽ hình thành đúng là các node cuối được pop ra sẽ thành các cây con bậc cao, là cây được thực hiện trước sau đó đến các cây ở phía trên.
- Bước 10: Lặp lại bước 0 cho đến khi hết chuỗi

- Từ ý tưởng và phân tích ở trên em sẽ tiến hành code cho bài toán của mình trên cơ sở sẽ code đầu tiên là khai báo một stack có kiểu dữ liệu là node có tên là operands. Đây sẽ là stack lưu trữ các thông tin của lá mỗi khi có một node lá được sinh ra. Ngoài ra stack operands này cũng sẽ được sử dụng để lưu trữ các node tạm gồm phần tử bên

trái, bên phải vì nó có kiểu dữ liệu là NODE nên cũng sẽ cùng kiểu dữ liệu với hàm của em. Tiếp theo cũng sẽ tạo ra một stack có tên là operators để lưu trữ các toán tử của em. Đây sẽ stack có kiểu dữ liệu char vì với mỗi toán tử được truyền vào thì ta sẽ chỉ xét toán tử của chúng, sau khi pop ra để thực hiện hàm thì mỗi toán tử sẽ đi kèm với các toán hạng được push vào operands sau khi hoàn tất việc thiết lập nên ta chỉ cần stack operators có kiểu dữ liệu là char. Tiếp theo thì em cần phải cho chạy vòng for từ đầu đến cuối chuỗi để tiến hành kiểm tra từng phần tử. Với ký tự đầu vào là số thì em sẽ so sánh vào if, khi thoả if thì nó sẽ thực hiện vòng while, vòng while của em sẽ kết thúc khi nó đã duyệt hết mảng hoặc ký tự tiếp theo nó đọc được không phải là ký tự số. Trong vòng while thì với mỗi số được phát hiện ra thì em sẽ có biến num sẽ nhân với 10 sau đó cộng với phần đã chuyển đổi thành số nguyên của em sau đó lưu lại vào biến num. Thì với mỗi lần lặp như vậy em sẽ thu được một số nguyên có số chữ số giống với biểu thức đầu vào. Sau khi thoát khỏi vòng while thì em sẽ tiến hành push cái số hạng vừa tìm được vào stack operands. Sau khi thu được số hạng đầu tiên thì em sẽ tiến hành kiểm tra xem biểu thức của mình có dấu cộng trừ nhân chia hay không và sau khi thoả thì tiếp tục kiểm tra điều kiện ưu tiên là có phải nhân hay chia hay không, và sau khi kiểm tra ưu tiên thì kiểm tra xem stack của operators nó có rỗng hay không. Nếu thoả các điều kiện là operators không rỗng, phần tử operators là cộng, trừ, nhân và chia, cùng với đó là toán tử rút ra là nhân hoặc chia, hoặc là nếu không phải nhân chia thì cũng sẽ có tồn tại toán tử ở trong stack thì em sẽ tiến hành việc tạo cây con theo đúng thứ tự ưu tiên là tạo cho nhân và chia trước. Nếu không thoả thì sẽ tiến hành push toán tử vào trong stack operators và lại duyệt tiếp phần tử kế tiếp. Trong code của em có đặc biệt hơn là sẽ có phần kiểm tra xem có dấu ngoặc hay không. Vì như giải thích ở trên thì đối với dấu ngoặc thì ta sẽ tiến hành nhóm các phép toán lại thành một nhóm. Thì em cũng sẽ tiến hành nhóm các phép toán lại. Bằng cách là em sẽ nhóm lại các phép toán lại một nhóm. Đầu tiên em sẽ kiểm tra xem có phải biểu thức cũng có chứa dấu ngoặc đơn hay không. Nếu có thì sẽ tiến hành push dấu ngoặc đơn đó vào trong stack operators. Sỡ dĩ ta cần push dấu ngoặc đơn đó vào trong như là cách để giữ, là cách ta thực hiện nhóm các phép toán lại. Ở đây dù là ngoặc đơn nhưng em vẫn sẽ giữ mức ưu tiên là nhân chia sẽ có mức ưu tiên cao hơn cộng trừ. Đối với phép toán bình thường em cũng đã có hàm so sánh nhân chia so với cộng trừ, tức là vòng while của em cũng sẽ kiểm tra xem operators có đang không rỗng không, rồi phần tử trong đó có phải là toán tử không, sau đó sẽ là kiểm tra xem toán tử đó có phải + - * / và so sánh xem với phần tử ở đỉnh của stack operators có thuộc * / hay không hoặc phần tử ở đỉnh của stack operators có phải là cộng trừ sẵn chưa vì ta sẽ duyệt nhân chia trước sau đó từ trái sang phải thì ta sẽ có các trường hợp là nếu toán tử đỉnh của operatos dang là thuộc dạng đúng ưu tiên thì sẽ tạo ra một node lá mới, sau đó ta sẽ lần lượt bỏ vào node phải, node trái từ stack operands. Lúc này ta sẽ giải phóng vị trí đỉnh và vị trí kế đỉnh của stack operands để tiến hành push cây con vào stack operands. Còn nếu rơi vào trường hợp đỉnh của operators không phải nhân chia thì ta sẽ đẩy các toán tử hiện tại, chắc chắn là + - vào stack operators. Còn trường hợp nữa là nếu toán tử hiện tại là * / nhưng

toán tử đỉnh đang là + - thì ta sẽ push * / vào stack operators để chuẩn bị đỉnh kế tiếp so sánh đó chính là * /. Đó là giải thích sơ nét ý tưởng cho việc code cho phần ưu tiên nhân chia so với cộng trừ. Tổng quan là hàm này sẽ xét vòng lặp while, thì nó sẽ xét bốn điều kiện gồm thứ nhất là operators phải không rỗng, thứ hai là xét tiếp xem phần tử đỉnh của operators hiện tại có phải là toán tử không, để tránh bị nhầm lẫn ký tự đặc biệt, nếu rơi ký tự đặc biệt mà không xét thì sẽ bị lỗi nên ta cần dặt điều kiện, thứ ba là xét xem toán tử hiện tại có đang là + - * / hay không và cuối cùng là phần tử đỉnh của stack operators có đang là ưu tiên hay không, vì như giải thích ở trên thì nếu toán tử hiện tại đang là * / thì ta sẽ cần bỏ vào stack operators để đến giai đoạn này ta sẽ lấy ra được một cặp toán hạng là right và left kết hợp với operators ta pop ra sẽ hình thành nên một cây. Còn nếu không phải */ thì tuỳ thuộc vào điều kiện dấu ngoặc đơn thì ta sẽ có các hàm xử lý tương ứng. Quay lại với dấu ngoặc đơn em đang nói dở thì bỏ qua dấu ngoặc đơn và sau khi truyền được dấu nhân và chia trong biểu thức vào stack và đọc ra được thêm một số thì ta sẽ hình thành một cây nhị phân mới, nó sẽ thuộc vào một lớp đối với dấu ngoặc đơn trên. Còn trường hợp không phải ưu tiên thì sẽ push vào operators, sau khi push operators thì push số hạng vào operands và lại kiểm tra độ ưu tiên và thoả thì thực hiện nhóm cây còn không thì lại thực hiện push vào operators với toán tử còn với toán hạng là operands. Cứ lặp lại quá trình cho đến gặp dấu đóng ngoặc đơn. Lúc này ta sẽ thực hiện việc hình thành các cây theo đúng theo thứ tự là phần tử ở đỉnh của stack operators sẽ được đẩy xuống làm node cha của nhánh con ở cấp thấp, chỉ hơn cấp đối với cây con thực hiện nhân chia. Và ở trong đây ta sẽ lấy ra hết tất cả toán tử mà ta đã push vào operators đồng thời cũng sẽ pop ra hết các toán hạng của stack operands. Với một toán tử được lấy ra từ stack operators ta sẽ giải phóng chúng để dồn các phần tử phía sau lên. Tương tự như thế thì mỗi lần lấy từ đỉnh của stack operators ra thì ta cũng sẽ lấy từ đỉnh của stack operands ra lần lượt là right sau đó giải phóng để dồn left lên. Ta sẽ thực hiện hàm này đến khi phần tử đỉnh là dấu mở ngoặc đơn mà ta đã cho vào stack operators mà em đã có nói ở trên. Thì đó là trình bày về dấu ngoặc đơn. Cuối cùng là hàm có ưu tiên thấp nhất, lúc này sẽ là + -. Ở đây em đã đảm bảo tất cả toán tử có dạng là */ sẽ được hình thành cây con, và ở tầng thấp nhất vì được ưu tiên tính đầu tiên. Đó là tổng quan về ý tưởng hình thành đoạn code. Sau đây sẽ là phần trình bày code của em.

2.2. Code phần tạo cây biểu thức:

```
NODE* createTree(string expression) {
      stack<NODE*> operands;
      stack<char> operators;
      for (int i = 0; i < expression.length(); i++) {
           char exp = expression[i];
           if (isdigit(exp)) { //Nếu là số thì ta thực hiện hàm
chuyển đổi số
                int num = exp - '0';
                while (i + 1 < expression.length() &&
isdigit(expression[i + 1])) { //nếu nhiều hơn một chữ số thì
thực hiện
                     num = num * 10 + (expression[i + 1] - '0');
i++;
                }
                operands.push(GetNode(num));
           }
           else if (exp == '(') {
                operators.push(exp);
           }
           else if (Operator(exp)) {
                while (!operators.empty() &&
Operator(operators.top()) && ((operators.top() == '*' ||
operators.top() == '/') || (exp == '+' || exp == '-'){
                     char op = operators.top(); // nếu là nhân
chia thì ưu tiên thực hiện hàm
                     operators.pop();
                     NODE* temp_right = operands.top();
          operands.pop();
```

```
                    NODE* node = GetNode(op);
                    node->left = temp_left;
                    node->right = temp_right;
                    operands.push(node);
               }
               operators.push(exp);
          }
else if (exp == ')') {
               while (operators.top() != '(') {
                    char op = operators.top();
                    operators.pop();
                    NODE* temp_right = operands.top();
                    operands.pop();
                    NODE* temp_left = operands.top();
                    operands.pop();
                    NODE* node = GetNode(op);
                    node->left = temp_left;
                    node->right = temp_right;
                    operands.push(node);
               }
               if (!operators.empty()) // nếu đã gặp operators
thì sẽ giải phóng dấu mở ngoặc đi
                    operators.pop();
          }
     }
while (!operators.empty()) {
          char op = operators.top(); //op sẽ là biến chứa toán
tử
          operators.pop();
```

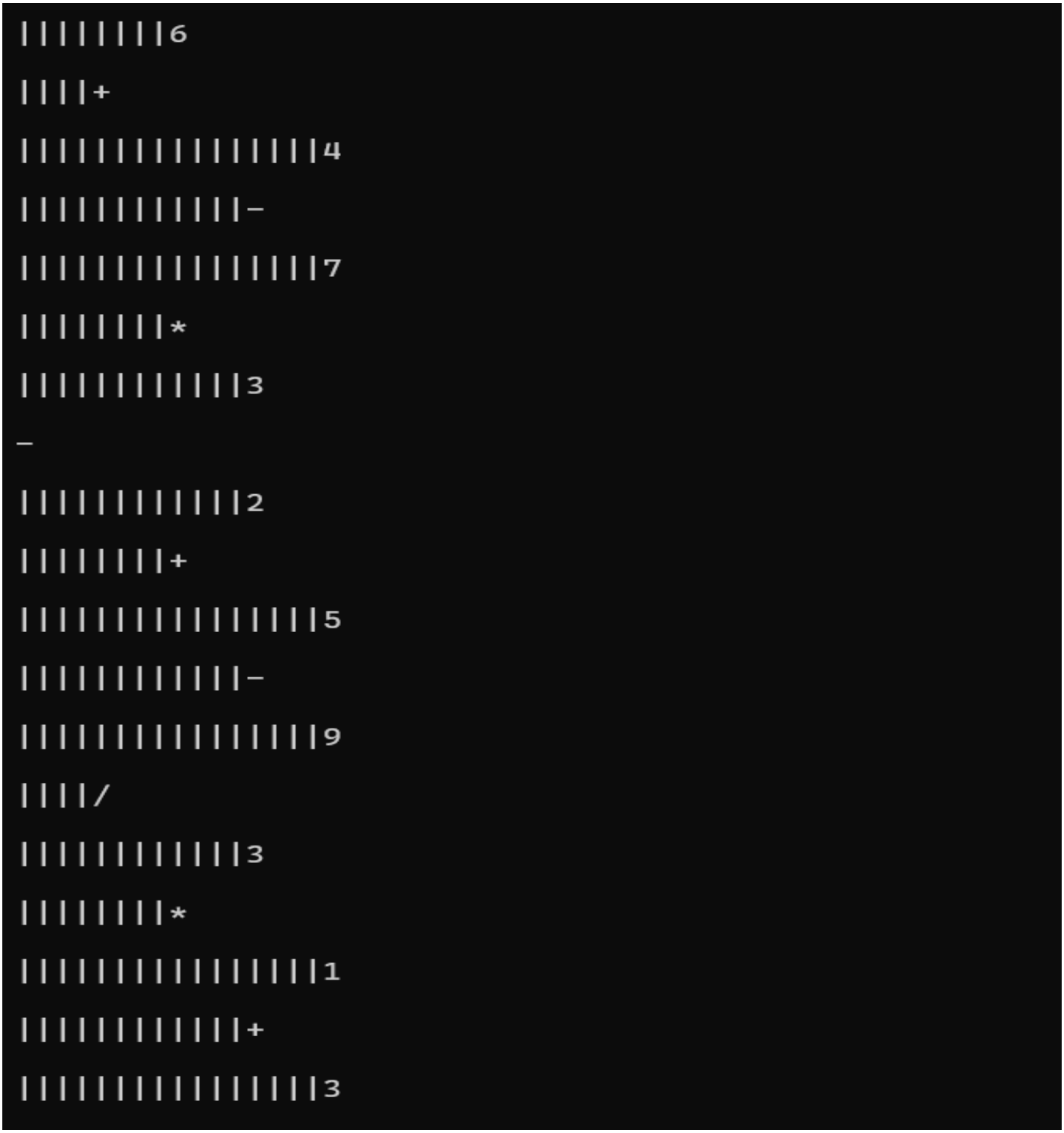
```
NODE* temp_right = operands.top();
          operands.pop();
          NODE* temp_left = operands.top();
          operands.pop();
          NODE* node = GetNode(op);
          node->left = temp_left; // Từ nút mới tạo ta sẽ gán
nhánh trái là số trái
          node->right = temp_right; // Từ nút mới tạo ta sẽ gán
nhánh phải là số trái
          operands.push(node);
      }
      return operands.top();
 }
```

2.3. Code phần kiểm tra toán tử và khởi tạo node mới

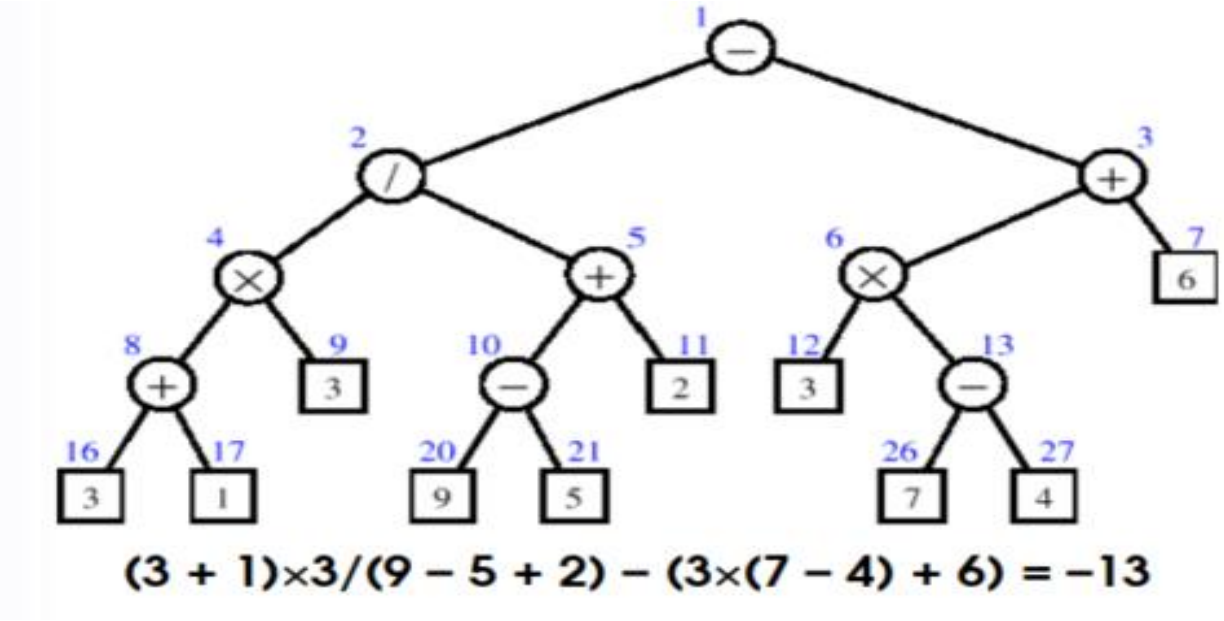
```
struct NODE {
      long value;
      NODE* left;
      NODE* right;
};
 bool Operator(char c) {
      return (c == '+' || c == '-' || c == '*' || c == '/');
}
NODE* GetNode(int x) {
      NODE* node = new NODE;
      node->value = x;
      node->left = node->right = nullptr;
      return node;
}
```

2.4. Kết quả đoạn code

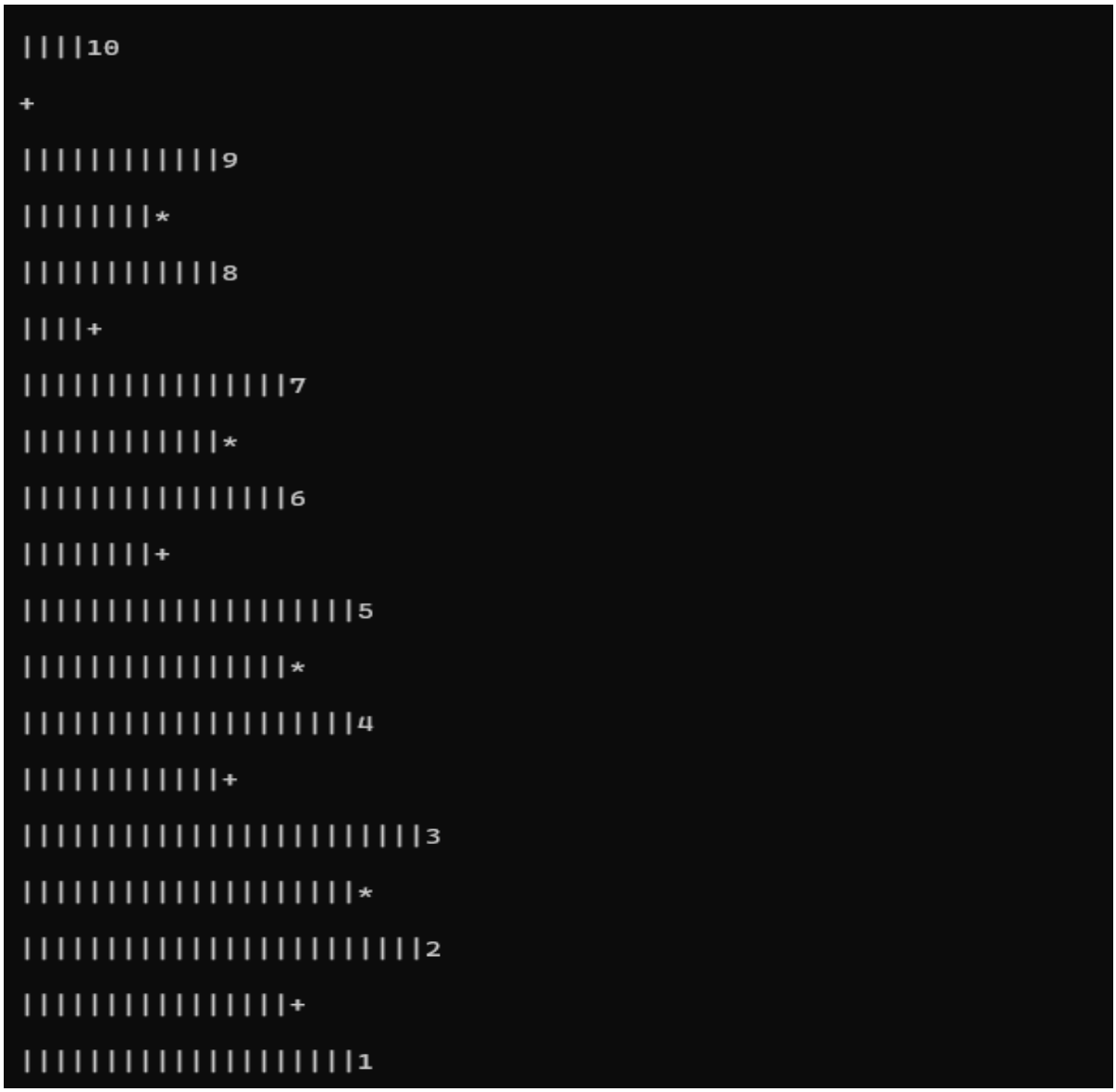
- Sau khi đoạn code thực thi thì chắc chắn em sẽ không có được cây như là hình vẽ nhưng ta có được các cụm NODE nó sẽ được lưu vào trong operands. Khi lấy ra thì tuỳ thuộc độ ưu tiên, độ nhóm lại của các phép tính ta sẽ có cây gồm nhánh trái nhánh phải khác nhau. Sau đây em sẽ ví dụ để chứng minh cây nhị phân của mình bằng biểu thức mà cô đã có trình bày trên lớp: (3+1)*3/(9-5+2)-(3*(7-4)+6):



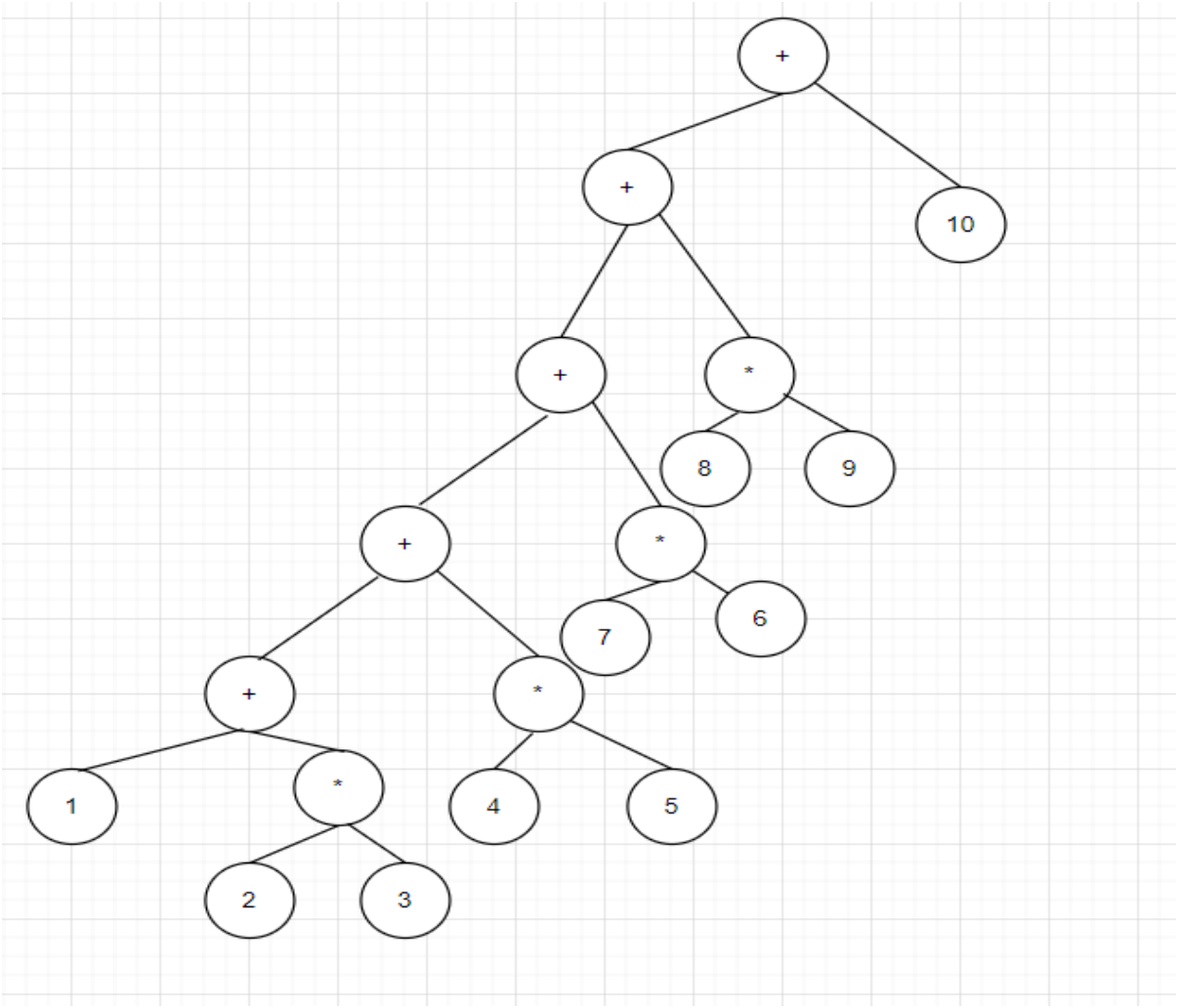
- Ta thấy với biểu thức có ngoặc thì đoạn code sẽ ưu tiên nhóm thành một nhóm và có xu hướng sẽ thành một nhánh con trái hay phải tuỳ thuộc vào mình đã truyền toán tử và toán hạng. Ở đây theo đúng những gì em trình bày thì dấu mở ngoặc đã mở, thì các toán hạng sẽ tập trung xung quanh các phép tính trong dấu mở ngoặc đó thay vì tách ra và nằm ở các node lá. Do cộng thì sẽ được giữ chứ không có bung ra thành các lá, còn đối với nhân thì sẽ được hình thành cây con nhưng nếu nhân với biểu thức nào thì phép nhân đó lại sinh ra các cây con nhỏ hơn nữa, nên ta có thể kết luận rằng đối với phép nhân chia thì chúng có xu hướng trở thành các cây con, và biểu thức có dấu ngoặc thì có xu hướng sẽ nhóm lại thành một cây nhỏ, tức là cây nhị phân nhỏ thuộc vào nhánh bên trái hay phải thì tuỳ. Và so sánh với cây nhị phân cô đã trình bày trên lớp thì em nhân định rằng cây nhị phân biểu thức của em là chính xác.



- Theo cô yêu cầu thì phải là cây nhị phân ít nhất 5 bậc, thì em sẽ có cây con sau có 5 bậc, phương trình là 1+2*3+4*5+6*7+8*9+10



- Được trình bày bằng hình vẽ trực quan sẽ là:



2.5. Nhận xét

- Nhận xét: Đối với em việc đã hình thành cây nhị phân từ biểu thức là phần khó nhất và phức tạp nhất đối với bài toán này. Việc xác định độ ưu tiên xong rồi lại xác định biểu thức có ngoặc hay không để tiến hành nhóm các biểu thức lại rồi lại thêm vào stack, pop stack hay lại push các nhánh lá sau khi được hình thành bởi các node. Đối với em khó nhất của phần này là việc xác định độ ưu tiên nhân chia so với cộng trừ, em đã thử rất nhiều cách trước khi nghĩ ra cách là lấy toán tử hiện tại so sánh với đỉnh của stack operators. Tức là việc ta luôn có 1 toán tử được lưu ở đỉnh của stack operators và một toán tử hiện tại thì để đảm bảo rằng sẽ luôn có những sự so sánh cũng như là cơ sở để em thực hiện đúng quy tắc phép toán là nhân chia trước cộng trừ sau và làm phải từ bên trái sang bên phải vì theo cơ chế stack là phần toán tử nào thêm vào sau thì sẽ được lấy ra đầu. Thì đối với các toán tử bên phải sẽ được thêm vào sau, sau đó lấy ra đầu để kết hợp với toán hạng hình thành node lá sau đó được thêm vào stack operands đầu tiên nên các node này có xu hướng trở thành các node lá bậc thấp, còn các toán tử được thêm đầu sẽ được ra cuối và khi truyền cũng như lấy ra từ stack operands thì sẽ có xu hướng trở thành node bậc cao, tức nằm ở tầng thấp và trở thành các phép toán được ưu tiên thực hiện trước tiên, theo đúng quy tắc

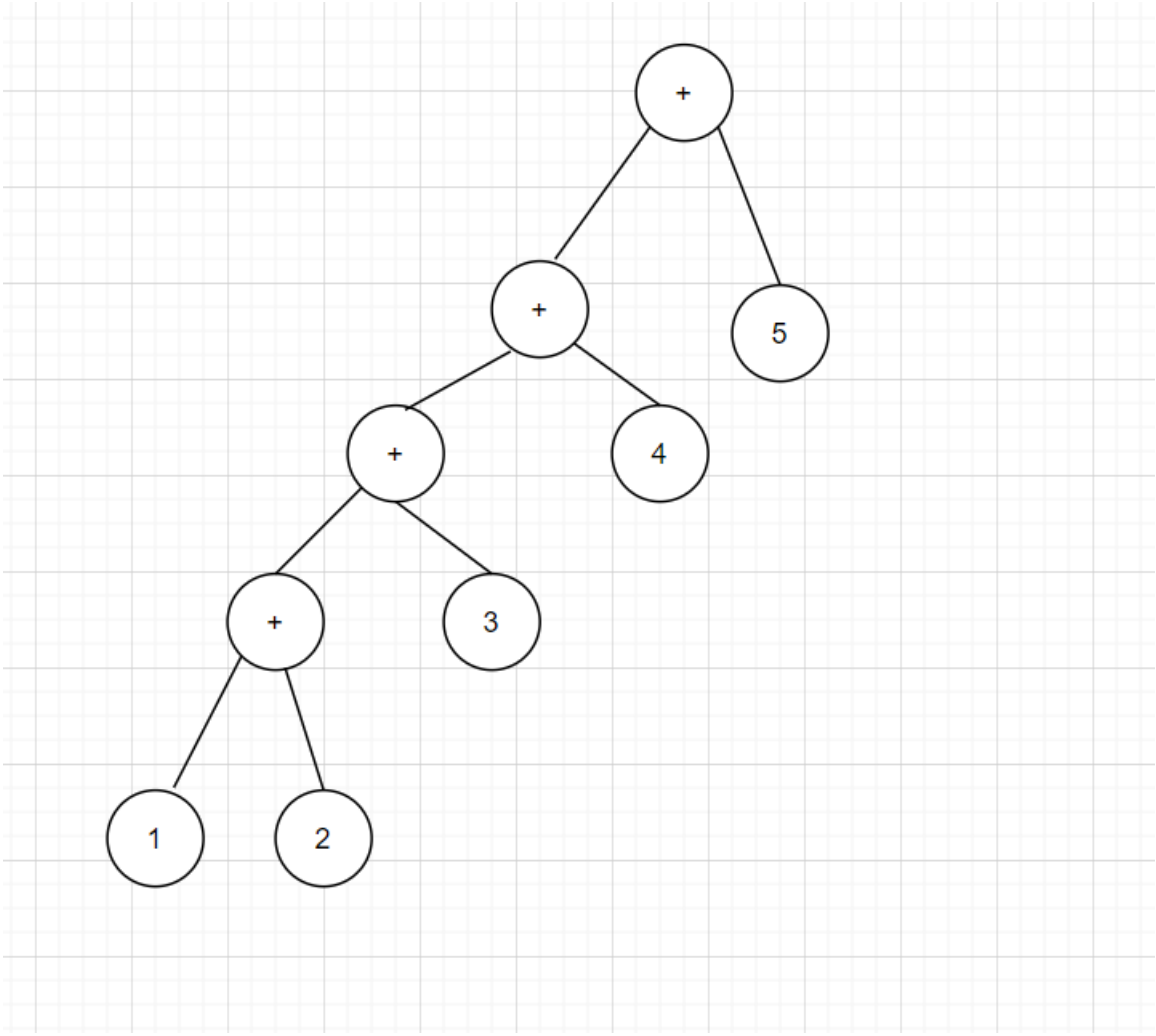
toán học. Việc hình thành cây nhị phân trên phải yêu cầu ta hiểu cấu trúc cây và hiểu về quy tắc hoạt động của stack, ngoài ra là còn về xét độ phức tạp nữa khi phải so sánh toán tử hiện tại với toán tử nằm ở đỉnh của stack operators. Sau khi xét độ ưu tiên thì ta cần hình thành node lá mới sau đó truyền lần lượt phải và trái để khi lấy ra thì biểu thức có số bên trái đi với toán tử và số bên phải.

- Việc sử dụng một stack operands để lưu cả các toán hạng và các node lá vì em muốn tiết kiệm không gian lưu trữ. Về quy tắc thì với mỗi toán hạng được sinh ra thì ta sẽ luôn sinh ra một node mới nên ta phải khai báo stack operands có kiểu NODE, lý do cho việc phải khai báo node thì để lưu trữ trong chương tình.

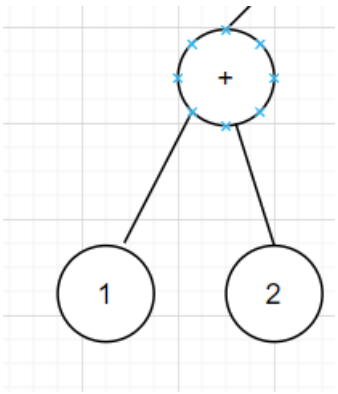
3. Hàm duyệt và tính giá trị biểu thức

3.1. Ý tưởng

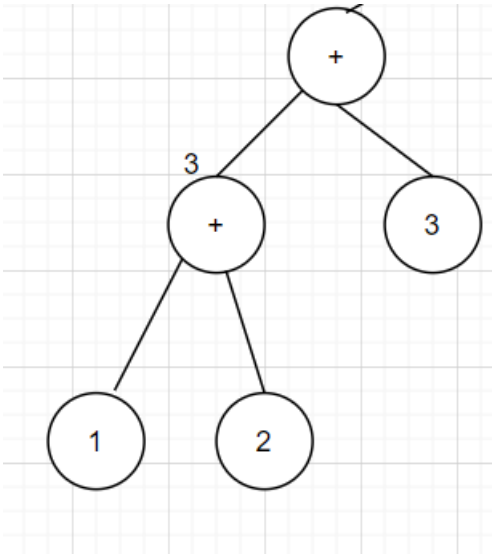
- Đã hoàn tất phần khó nhất của bài toán là hình thành nên cây nhị phân, thì em sẽ tiến hành duyệt cây nhị phân đó để thực hiện tính giá trị của một biểu thức.
- Ý tưởng ở đây thì em sẽ gọi đệ quy hàm duyệt cây để cho nó duyệt đến cây con trái nhất của cây nhị phân này sau đó tại cây con trái nhất em sẽ có hàm trả về giá trị duyệt trái và sau đó duyệt phải và sau đó sẽ gọi hàm tính toán thì với lệnh gọi hàm tính toán ta sẽ truyền vào ba thông số là left tức là node lá bên trái của cây mà ta vừa duyệt, right tức là node lá bên phải của cây mà ta duyệt và kèm theo đó ta sẽ trả về node gốc tức là giá trị tại root.
- Từ ý tưởng đó em sẽ hình thành đoạn code trên cơ sở là đầu tiên em sẽ phải đảm bảo chắc chắn là có cây nhị phân nên đầu tiên sẽ kiểm tra xem nốt root gốc, tức là nốt parent của toàn bộ cây nhị phân có NULL hay không, nếu có thì ngưng chương trình và báo lỗi còn nếu không sao thì sẽ thực thi lệnh duyệt cây. Lý do em phải có dòng trên vì khi không có dòng lệnh đó thì máy của em liên tục báo lỗi “nulptr”, nên em sẽ có điều kiện nếu root == nulptr thì ngưng thực hiện. Tiếp theo thì em sẽ xét xem node lá trái và node lá phải có phải là null chưa vì ta sẽ duyệt từ phần tử trái nhất qua phải nhất rồi lại trái rồi lại phải ta sẽ duyệt đến khi nào cây nhị phân hoàn tất tính xonh. Tức là sau mỗi lần tính ta sẽ có giá trị và khi đó ta sẽ dịch con trỏ lên. Nếu tính tại nhánh trái ta sẽ có giá trị tại nhánh trái sau đó dịch con trỏ tính nhánh phải sau đó lại dịch con trỏ lên khi đó ta lại thực hiện tính và trả về kết quả, nếu cây con lớn hơn ta sẽ lại dịch kết quả vừa tính lên nhánh trái và lại đưa con trỏ phải lên. Cứ như vậy đến khi ta hết khả năng dịch lên nữa thì ngưng. Khả năng dịch lên ngưng khi ta chạm vào root, khi đó sẽ không còn node phải hay trái nào cho ta dịch nữa thì ta ngưng việc dịch. Hàm của em sẽ trả về kết quả ra biến “result” và in ra màn hình. Để ví dụ thì em sẽ có phép tính sau: 1+2+3+4+5, chuyển thành cây nhị phân là:



- Để thực hiện thì con trỏ sẽ dịch đến vị trí trái nhất của nhánh cây con bên trái đầu tiên thì em có đó là cây con sau:



Sau đó do gọi đệ quy thì left của em sẽ đặt tại số 1, sau đó right sẽ đặt tại số 2, root của em đang là cộng thì sẽ thực thi phép tính là 2+1 = 3, kết quả 3 này sẽ được xem như thay thế cho toàn bộ cây đó. Và ta lại kiểm tra xem nó null không, và con trỏ quay lên nhánh cây ở trên. Khi đó ta có cây sau:



Ta thấy nhánh cây con trái nhất đã được thay bằng số 3 và ta đã có cây tiếp theo với left là 3, right là 3 và root sẽ là dấu cộng. Theo ý tưởng đó thì mỗi lần hoàn thành tính thì root sẽ dịch lên, dịch lên từng bậc, tức là root sẽ luôn duy trì ở đó cho đến khi left và right thực hiện xong phép tính của mình trả về giá trị tại left và right của nhánh con có root để chúng thực hiện phép tính để trả về kết quả thì mới dịch sang root kế tiếp. Từ ý tưởng đó em sẽ có đoạn code sau, đoạn code của em gồm hai hàm, một hàm calculate và hàm duyệt cây. Đối với hàm calculate em sẽ thiết lập nó trả về kiểu long double để nếu cô có minh hoạ bằng số có 10 chữ số hay lớn hơn thì em vẫn đảm bảo em sẽ in ra màn hình đầy đủ kết quả, không có sai lệch. Cũng trong hàm calculate em cũng sẽ để có ba dữ liệu input gồm left, right và op với op tức là viết tắt của operator- em không thể đặt operator vì khi dịch chương trình thì visual studio báo lỗi khi nó bị nhầm lẫn với lệnh khác nên em đã để là op. Với left và right cùng có kiểu dữ liệu sẽ là long double. Kiểu dữ liệu long khác long double theo em biết là long thông thường sẽ chỉ biểu diễn số nguyên còn long double sẽ biểu diễn cả nguyên và thực nên dù cây nhị phân đang có kiểu dữ liệu là long nhưng để in ra số thực nếu xảy ra thì em sẽ khai báo hai biến left và right có cùng kiểu dữ liệu là long double, còn root thì sẽ luôn có kiểu char rồi nên em sẽ không xét đến. Và kiểu dữ liệu mà calculate và hàm duyệt cây sẽ luôn trả về kiểu long double theo đúng những gì mà kiểu dữ liệu left và right có. Ý tưởng hàm calculate sẽ là dùng cấu trúc switch case, với mỗi biến char đưa vào thì nếu rơi vào trường hợp " +" thì sẽ return left + right ra hàm calculate và trả cho hàm duyệt cây để tiếp tục duyệt, nếu rơi vào trường hợp " – " thì sẽ return kết quả của phép tính left – right và tương tự là " * " thì sẽ là left*right còn " / " thì sẽ là right / left. Bên cạnh đó em cũng sẽ thêm vào các đoạn code về cách duyệt cây theo thứ tự tức là hàm duyệt cây để tính sẽ là duyệt theo trung tố tức In order nhưng để có sự so sánh thì em sẽ viết riêng ra ba hàm về duyệt theo tiền tố - Pre order hay còn gọi là Node Left Right (NLR), hàm về duyệt theo trung tố - In order hay còn gọi là Left Node Right (LNR) và cuối cùng là hàm về duyệt theo hậu tố - post order hay còn gọi là Left Right Node (LRN). Ở đây thì em chỉ duyệt và in màn hình còn về việc duyệt theo hậu tố để hình thành cây nhị phân từ biểu thức thì đã được em trình

bày ở trên, còn về việc duyệt theo trung tố để thực hiện việc tính toán thì cũng đã được em đề cặp ở phần này rồi. Phần duyệt của 3 kiểu duyệt sẽ chỉ hỗ trợ cho em trong việc so sánh kết quả, có cái nhìn trực quan hơn về cách mà đoạn code duyệt. Về mặt ý tưởng hình thành đoạn code như trên, em tiến hành viết code cho việc duyệt cây và tính toán biểu thức và duyệt cây để in ra màn hình.

3.2. Code phần thực hiện tính toán cho các nhánh cây con

```
long double calculate(long double left, long double  right, char
op) {
      switch (op) {
      case '+': return left + right;
      case '-': return left - right;
      case '*': return left * right;
      case '/': return left/right;
      }
 }
```

3.3. Code phần thực hiện duyệt cây theo trung tố và thực hiện gọi hàm tính

```
long double evaluateTree(NODE* root) {
      if (root == nullptr) return 0;
      if (root->left == nullptr && root->right == nullptr) {
           return root->value;
      }
      long double left = evaluateTree(root->left);
      long double right = evaluateTree(root->right);
      return calculate(left, right, root->value);
 }
```

3.4. Code phần duyệt cây in ra màn hình

3.4.1. Duyệt theo tiền tố (Pre Order – NLR)

```
void PreOrder(NODE* root) {
      if (root != nullptr) {
           if (Operator(root->value))cout << char(root->value) << ' ';
           else cout << root->value << ' ';
           PreOrder(root->left);
          PreOrder(root->right);
      }else return;
 }
```

3.4.2. Duyệt theo trung tố (In order – LNR):

```
void InOrder(NODE* root) {
      if (root != nullptr) {
           InOrder(root->left);
           if (Operator(root->value)) cout << char(root->value) << ' ';
           else cout << root->value << ' ';
           InOrder(root->right);
      }
      else return;
 }
```

3.4.3. Duyệt theo hậu tố (Post order – LRN):

```
void PostOrder(NODE* root) {
      if (root != nullptr) {
           PostOrder(root->left);
           PostOrder(root->right);
           if (Operator(root->value)) cout << char(root->value) << ' ';
           else cout << root->value << ' ';
      }else return;
  }
```

3.5. Kết quả chạy đoạn code:

- Để ví dụ về minh hoạ kết quả chạy code thì em sẽ chạy đoạn phương trình mà cô có để trong slide cô dạy trên lớp là: (3 + 1)*3/(9 – 5 + 2) – (3*(7 – 4) + 6) theo kết quả là bằng -13, em sẽ kiểm lại đoạn code bằng cách chạy phương trình trên. Kết quả là:

```
Nhap mot bieu thuc: (3+1)*3/(9-5+2)-(3*(7-4)+6)
Ket qua la = -13.000000
Duyet Pre-order (NLR):  - / * + 3 1 3 + - 9 5 2 + * 3 - 7 4 6
Duyet In-order (LNR): 3 + 1 * 3 / 9 - 5 + 2 - 3 * 7 - 4 + 6
Duyet Post-order (LRN): 3 1 + 3 * 9 5 - 2 + / 3 7 4 - * 6 + -
```

Về cấu trúc cây thì em đã trình bày ở phần trên, ở đây em sẽ quan tâm về cách duyệt của cây và cả kết quả. Thì em cũng đã minh hoạ trực quan cách duyệt của hàm và cả kết quả, ta thấy ra đúng -13. Duyệt In order là thứ tự thực hiện phép tính ở cây nhị phân còn duyệt post order là để thực hiện từ biểu thức ta sẽ chuyển thành cấu trúc cây

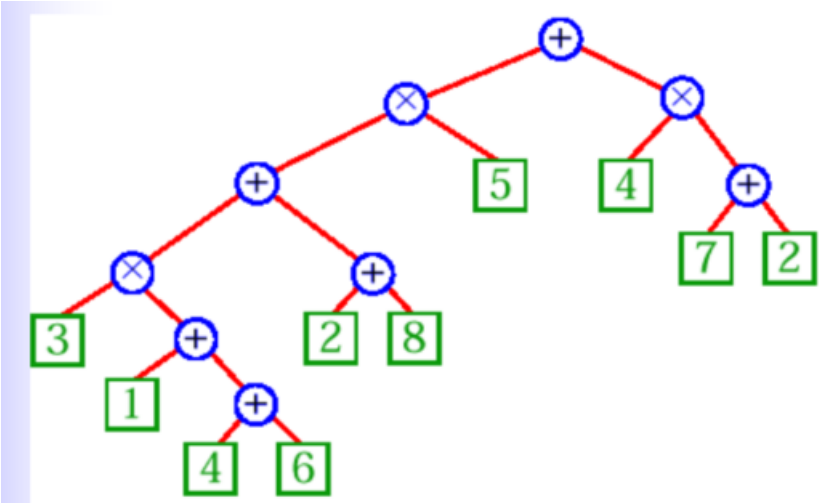
- Cô yêu cầu cài đặt cây nhị phân tối thiểu 5 bậc thì em cũng sẽ mô phỏng một cây nhị phân có 5 bậc là hàm sau, cũng là phương trình cô có dạy trên lớp: ((((3*(1+(4+6))) + (2+8))*5) + (4*(7+2))). Kết quả là:

```
Nhap mot bieu thuc: ((((3*(1+(4+6)))+(2+8))*5)+(4*(7+2)))
Ket qua la = 251.000000
Duyet Pre-order (NLR):  + * + * 3 + 1 + 4 6 + 2 8 5 * 4 + 7 2
Duyet In-order (LNR): 3 * 1 + 4 + 6 + 2 + 8 * 5 + 4 * 7 + 2
Duyet Post-order (LRN): 3 1 4 6 + + * 2 8 + + 5 * 4 7 2 + * +
```

Và cây nhị phân hình thành là:

```
|||||||||||2
|||||||+
|||||||||||7
||||*
|||||||4
+
|||||||5
||||*
||||||||||||||8
||||||||||+
||||||||||||||2
|||||||+
||||||||||||||||||||||6
||||||||||||||||||+
||||||||||||||||||||||4
||||||||||||||+
||||||||||||||||||1
||||||||||*
||||||||||||||3
```

So sánh với cây nhị phân cô trình bày trong slide:



Dù hơi khó nhìn vì không trực quan như cô nhưng vẫn thấy cây thấp nhất là 6+4 rồi +1 cao hơn rồi cây cao hơn là *3. Nhìn chung vẫn ra đúng theo cây mà cô đã in

trong slide. Và cây nhị phân này có 5 bậc đúng theo yêu cầu của cô là tối thiểu 5 bậc.

3.6. Nhận xét

- Đoạn code của em đã hoàn thành nhiệm vụ là duyệt cây và thực hiện tính toán trên cơ sở là hình thành cây từ biểu thức và từ cây đó ta sẽ duyệt in order để có thể thực hiện tính toán và trả về kể quả. Em thấy rằng đoạn code của em là tối ưu cả số nguyên có phạm vi tính toán rộng, tức là với số nguyên có 9 chữ số thì đoạn code của em vẫn thực hiện tính toán một cách ổn thoả. Sau đây là ví dụ với việc nhập vào số nguyên có trên 8 chữ số, biểu thức là: ((((31234567*(12345678+(412345678+612345678)))+(212345678))*512345678)+(412345678*(712345678+212345678))), em sẽ có kết quả như sau:

```
Nhap mot bieu thuc: ((((31234567*(12345678+(412345678+612345678)))+(212345678))*512345678)+(412345
678*(712345678+212345678)))
Ket qua la = 165955956780128958907678 72.000000
Duyet Pre-order (NLR):  + * + * 31234567 + 12345678 + 412345678 612345678 212345678 512345678 * 41
2345678 + 712345678 212345678
Duyet In-order (LNR): 31234567 * 12345678 + 412345678 + 612345678 + 212345678 * 512345678 + 412345
678 * 712345678 + 212345678
Duyet Post-order (LRN): 31234567 12345678 412345678 612345678 + + * 212345678 + 512345678 * 412345
678 712345678 212345678 + * +
```

- Nhận xét: Đoạn code em chạy tốt trong các trường hợp tuy nhiên nếu số có rất nhiều chữ số, cụ thể là từ 10 chữ số trở lên thì ta sẽ sử dụng kiểu long long, nhưng em không thể xây dụng đối với số có 10 chữ số trở lên vì trong thực tế khi thực hiện với số có 9 chữ số thì ta đã thấy kết quả cực kỳ lớn rồi, nên thật khó để yêu cầu hơn, vì thật tế không ai tính biểu thức với số lớn như vậy. Em đã hoàn tất việc tính toán biểu thức bằng cách duyệt cây nhị phân, cây nhị phân thì được tạo bằng cách duyệt từng phần tử của biểu thức input từ bàn phím.

4. Phần chính và các hàm hỗ trợ

4.1. Tổng quan

- Ở đây các ý tưởng khó nhất đã được em trình bày ở trên rồi, nên đối với hàm chính thì ngoài việc gọi các hàm con ra, ta cần khai báo các kiểu dữ liệu để tiến hành truyền thông số để bắt đầu thực hiện hàm. Các hàm hỗ trợ sẽ là in các node lá của cây nhị phân này, ý tưởng cho hàm in node lá thì cũng giống với hàm thực hiện tính toán đó là sẽ duyệt từ đầu đến cuối nhưng khác là nó không thực hiện tính mà mỗi lần duyệt qua thì sẽ in ra các toán tử toán hạng và sau đó kèm theo là các dấu "|" để thể hiện các bậc của cây nhị phân. Dấu "|" nhiều hơn là bậc cao hơn. Do gọi đệ quy trái phải nên hàm này sẽ duyệt lần lượt từng node của node lá sinh ra từ hàm tạo cây. Như ta biết hàm tạo cây thì các node lá cuối cùng được nhét vào thì sẽ được sinh ra làm parent nên ta sẽ in nó ra đầu tiên vì nó là bậc thấp nhất, sau đó sẽ lần lượt in ra các cây con của nó ra, và mỗi lần in thì sẽ đi kèm với dấu "|", mỗi lần có

một root và toán hạng trái phải thì cách nhau 4 lần “|” , và giữa các node root tức là các bậc của cây nhị phân cũng cách nhau 4 lần “|”. Do khoảng cách tới 4 lần nên nhìn chung ta vẫn nhìn rõ mối quan hệ parent and children của các cây nhị phân. Ý tưởng là với ta sẽ cho chạy lệnh in, gọi đệ quy hàm printleaf, với mỗi lần chạy thì dịch sang phải tức là dịch sang vị trí tiếp theo của cây bên phải, và sau đó sẽ dịch bên trái để in cây con bên trái. Ta cứ dịch phải rồi trái saun đó in ra, nếu cây con phải hết phần tử để in thì sẽ dịch trái đến khi hết phần tử để in ra thì kết thúc. Theo đúng quy tắc mà ta đã khai báo là right đã là lưu lại của con trỏ trỏ đến right còn left sẽ là con trỏ trỏ đến left. Dịch đến khi nào mà nó hết tức là rơi vào nullptr thì ta ngưng, hoàn tất việc in ra màn hình. Sau đây là đoạn code.

4.2. Code hàm in cây nhị phân

```
void printtree(NODE* q, int space) {
      if (q != nullptr) {
            space += 4;
            printtree(q->right, space);
            cout << endl;
            for (int i = 4; i < space; i++) {
                  cout << "|" ;
            }
            if (Operator(q->value)) cout << char(q->value) << endl;
            else cout << q->value << endl;
            printtree(q->left, space);
      }
      else return;
 }
```

4.3. Code hàm chính

```
int main() {
      string expression;
      cout << "Nhap mot bieu thuc: ";
      cin >> expression;
      NODE* root = createTree(expression);
      long double result = evaluateTree(root);
      printf("Ket qua la = %Lf", result);
      cout << endl;
      std::cout << "Duyet Pre-order (NLR):  ";
      PreOrder(root);
      std::cout << std::endl;

      std::cout << "Duyet In-order (LNR): ";
      InOrder(root);
      std::cout << std::endl;

      std::cout << "Duyet Post-order (LRN): ";
      PostOrder(root);
      std::cout << std::endl;
      cout << "Trinh tu cay nhi phan thuc hien tinh toan: \n";
      printleaf(root,0);
 }
```

5. Tổng kết

- Em đã hoàn thành việc tạo ra cây nhị phân từ biểu thức nhập từ bàn phím. Cơ sở để tạo cây nhị phân là duyệt theo hậu tố tức duyệt trái xong duyệt phải và đến duyệt node. Để mà có thể ưu tiên xem cái nào sẽ được duyệt trước thì em đã xét độ ưu tiên. Ý tưởng về độ ưu tiên là phần phức tạp nhất của bài toán này, khi độ ưu tiên được xét đến sẽ là nhân chia trước, cộng trừ sau khi cung cấp là trái trước phải sau và ưu tiên nữa sẽ là có ngoặc đơn. Loại ưu tiên có ngoặc đơn sẽ thuộc vào trường hợp nhóm các phần tử lại thành một nhóm và xem nguyên một nhánh đó là một

nhánh con trái hay phải gì đó thì tuỳ bài toán. Sau khi ưu tiên hết rồi thì còn bao nhiêu em sẽ đẩy hết vào stack với cơ sở duyệt là trái phải và node thì sau khi hoàn tất thì stack operands sẽ không còn lưu toán hạng nữa mà sẽ chuyển thành lưu các node lá của cây nhị phân biểu thức này.

- Có được cây nhị phân biểu thức thì em sẽ duyệt cây theo trung tố để thực hiện tính. Em sẽ gọi đệ quy để dịch đến node trái nhất và thực hiện sau đó cứ dịch lên đến khi nào thực hiện phép toán ở root xong thì kết thúc
- Em cũng áp dụng cách duyệt left node right để duyệt và in cây nhị phân.
- Điểm yếu của chương trình của em là do nhập vào là chuỗi nên nó hiểu là chuỗi và khoảng trắng cũng xem là một ký tự trong chuỗi đó nên khi nhập khoảng trắng nó sẽ hiểu là ký tự rỗng và báo lỗi
- Em hoàn thành những yêu cầu cô cho thông qua những phần trên là áp dụng cây nhị phân tính giá trị biểu thức tối thiểu 5 bậc. Sau đây là chương trình sau khi được nhập biểu thức và chạy:

```
Nhap mot bieu thuc: ((((3*(1+(4+6)))+(2+8))*5)+(4*(7+2)))
Ket qua la = 251.000000
Duyet Pre-order (NLR):  + * + * 3 + 1 + 4 6 + 2 8 5 * 4 + 7 2
Duyet In-order (LNR): 3 * 1 + 4 + 6 + 2 + 8 * 5 + 4 * 7 + 2
Duyet Post-order (LRN): 3 1 4 6 + + * 2 8 + + 5 * 4 7 2 + * +
Trinh tu cay nhi phan thuc hien tinh toan:

||||||||||2

|||||||+

||||||||||7

||||*

|||||||4

+

|||||||5

||||*

||||||||||||||8

||||||||||+

||||||||||||||2

|||||||+

||||||||||||||||||||||6

||||||||||||||||||+

||||||||||||||||||||||4

||||||||||||||+

||||||||||||||||||1

||||||||||*

||||||||||||||3
```