**7.3. CÁCH TÍNH GIÁ TRỊ BIỂU THỨC**

Có một vấn đề cần lưu ý là khi máy tính giá trị một biểu thức số học gồm các toán tử hai ngôi (toán tử gồm hai toán hạng như +, -, \*, /) thì máy chỉ thực hiện được phép toán đó với hai toán hạng. Nếu biểu thức phức tạp thì máy phải chia nhỏ và tính riêng từng biểu thức trung gian, sau đó mới lấy giá trị tìm được để tính tiếp. Ví dụ như biểu thức 1 + 2 + 4 máy sẽ phải tính 1 + 2 trước được kết quả là 3 sau đó mới đem 3 cộng với 4 chứ không thể thực hiện phép cộng một lúc ba số được.

Khi lưu trữ biểu thức dưới dạng cây nhị phân thì ta có thể coi **mỗi nhánh con của cây đó mô tả một biểu thức trung gian** mà máy cần tính khi xử lý biểu thức lớn. Như ví dụ trên, máy sẽ phải tính hai biểu thức 6 / 2 + 3 và 7 - 4 trước khi làm phép tính nhân cuối cùng. Để tính biểu thức 6 / 2 + 3 thì máy lại phải tính biểu thức 6 / 2 trước khi đem cộng với 3.

Vậy để tính một biểu thức lưu trữ trong một nhánh cây nhị phân gốc ở nút n, máy sẽ tính gần giống như hàm đệ quy sau:

function Calculate(n): Value;

begin

if <Nút n chứa không phải là một toán tử> then

Calculate := <Giá trị chứa trong nút n>

else

begin

x := Calculate(nút con trái của n);

y := Calculate(nút con phải của n);

Calculate := x R y;

end;

end;

*(Trong trường hợp lập trình trên các hệ thống song song, việc tính giá trị biểu thức ở cây con trái và cây con phải có thể tiến hành đồng thời làm giảm đáng kể thời gian tính toán biểu thức).*

Để ý rằng khi tính toán biểu thức, máy sẽ phải quan tâm tới việc tính biểu thức ở hai nhánh con trước, rồi mới xét đến toán tử ở nút gốc. Điều đó làm ta nghĩ tới phép cây theo thứ tự sau và ký pháp hậu tố. Trong những năm đầu 1950, nhà lô-gic học người Balan Jan Lukasiewicz đã chứng minh rằng biểu thức hậu tố không cần phải có dấu ngoặc vẫn có thể tính được một cách đúng đắn bằng cách **đọc lần lượt biểu thức từ trái qua phải** và dùng một Stack để lưu các kết quả trung gian:

Bước 1: Khởi động một Stack rỗng

Bước 2: Đọc lần lượt các phần tử của biểu thức RPN từ trái qua phải (phần tử này có thể là hằng, biến hay toán tử) với mỗi phần tử đó, ta kiểm tra:

Nếu phần tử này là một toán hạng thì đẩy giá trị của nó vào Stack.

Nếu phần tử này là một toán tử , ta lấy từ Stack ra hai giá trị (y và x) sau đó áp dụng toán tử đó vào hai giá trị vừa lấy ra, đẩy kết quả tìm được (x  y) vào Stack (ra hai vào một).

Bước 3: Sau khi kết thúc bước 2 thì toàn bộ biểu thức đã được đọc xong, trong Stack chỉ còn duy nhất một phần tử, phần tử đó chính là giá trị của biểu thức.

Ví dụ: Tính biểu thức 10 2 / 3 + 7 4 - \* (tương ứng với biểu thức (10 / 2 + 3) \* (7 - 4)

|  |  |  |
| --- | --- | --- |
| Đọc | Xử Lý | Stack |
| 10 | Đẩy vào stack | 10 |
| 2 | Đẩy vào stack | 10, 2 |
| / | Lấy 2 và 10 khỏi Stack, tính được 10/2=5, đẩy 5 vào Stack | 5 |
| 3 | Đẩy vào Stack | 5, 3 |
| + | Lấy 3 và 5 khỏi Stack, tính được 5+3=8, đẩy 8 vào Stack | 8 |
| 7 | Đẩy vào Stack | 8, 7 |
| 4 | Đẩy vào Stack | 8, 7, 4 |
| - | Lấy 4 và 7 khỏi Stack, tính được 7-4=3, đẩy 3 vào Stack | 8, 3 |
| \* | Lấy 3 và 8 khỏi Stack, tính được 8\*3=24, đẩy 24 vào Stack | 24 |

Ta được kết quả là 24

Dưới đây ta sẽ viết một chương trình đơn giản tính giá trị biểu thức RPN. Chương trình sẽ nhận Input là biểu thức RPN gồm các số thực và các toán tử + - \* / và cho Output là kết quả biểu thức đó.

Quy định khuôn dạng bắt buộc là hai số liền nhau trong biểu thức RPN phải viết cách nhau ít nhất một dấu cách. Để quá trình đọc một phần tử trong biểu thức RPN được dễ dàng hơn, sau bước nhập liệu, ta có thể hiệu chỉnh đôi chút biểu thức RPN về khuôn dạng dễ đọc nhất. Chẳng hạn như thêm và bớt một số dấu cách trong Input để mỗi phần tử (toán hạng, toán tử) đều cách nhau đúng một dấu cách, thêm một dấu cách vào cuối biểu thức RPN. Khi đó quá trình đọc lần lượt các phần tử trong biểu thức RPN có thể làm như sau:

T := '';

for p := 1 to Length(RPN) do

if RPN[p] **** ' ' then T := T + RPN[p]

else

begin

<Xử lý phần tử T>

T := '';

end;

Để đơn giản, chương trình không kiểm tra lỗi viết sai biểu thức RPN, việc đó chỉ là thao tác tỉ mỉ chứ không phức tạp lắm, chỉ cần xem lại thuật toán và cài thêm các mô-đun bắt lỗi tại mỗi bước.

Ví dụ về Input / Output của chương trình:

Enter RPN Expression: 10 2 / 3 + 4 7 - \*

10 2 / 3 + 4 7 - \* = 24.0000

P\_2\_07\_1.PAS \* Tính giá trị biểu thức RPN

{$N+,E+}

program CalculateRPNExpression;

const

Opt = ['+', '-', '\*', '/'];

var

T, RPN: String;

Stack: array[1..255] of Extended;

p, Last: Integer;

procedure StackInit;

begin

Last := 0;

end;

procedure Push(V: Extended);

begin

Inc(Last);

Stack[Last] := V;

end;

function Pop: Extended;

begin

Pop := Stack[Last];

Dec(Last);

end;

procedure Refine(var S: String);

var

i: Integer;

begin

S := S + ' ';

for i := Length(S) - 1 downto 1 do

if (S[i] in Opt) or (S[i + 1] in Opt) then

Insert(' ', S, i + 1);

for i := Length(S) - 1 downto 1 do

if (S[i] = ' ') and (S[i + 1] = ' ') then

Delete(S, i + 1, 1);

end;

procedure Process(T: String);

var

x, y: Extended; e: Integer;

begin

if not (T[1] in Opt) then

begin

Val(T, x, e); Push(x);

end

else

begin

y := Pop;

x := Pop;

case T[1] of

'+': x := x + y;

'-': x := x - y;

'\*': x := x \* y;

'/': x := x / y;

end;

Push(x); {Vào một}

end;

end;

begin

Write('Enter RPN Expression: ');

ReadLn(RPN);

Refine(RPN);

StackInit;

T := '';

for p := 1 to Length(RPN) do

if RPN[p] <> ' ' then T := T + RPN[p]

else

begin

Process(T);

T := ''; {Đặt lại T để chuẩn bị đọc phần tử mới}

end;

WriteLn(RPN, ' = ', Pop:0:4); {In giá trị biểu thức RPN được lưu trong Stack}

end.