**7.4. CHUYỂN TỪ DẠNG TRUNG TỐ SANG DẠNG HẬU TỐ**

Có thể nói rằng việc tính toán biểu thức viết bằng ký pháp nghịch đảo Balan là khoa học hơn, máy móc, và đơn giản hơn việc tính toán biểu thức viết bằng ký pháp trung tố. Chỉ riêng việc không phải xử lý dấu ngoặc đã cho ta thấy ưu điểm của ký pháp RPN. Chính vì lý do này, các chương trình dịch vẫn cho phép lập trình viên viết biểu thức trên ký pháp trung tố theo thói quen, nhưng trước khi dịch ra các lệnh máy thì tất cả các biểu thức đều được chuyển về dạng RPN. Vấn đề đặt ra là phải có một thuật toán chuyển biểu thức dưới dạng trung tố về dạng RPN một cách hiệu quả, và dưới đây ta trình bày thuật toán đó:

Thuật toán sử dụng một Stack để chứa các toán tử và dấu ngoặc mở. Thủ tục Push(V) để đẩy một phần tử vào Stack, hàm Pop để lấy ra một phần tử từ Stack, hàm Get để đọc giá trị phần tử nằm ở đỉnh Stack mà không lấy phần tử đó ra. Ngoài ra mức độ ưu tiên của các toán tử được quy định bằng hàm Priority như sau: Ưu tiên cao nhất là dấu "\*" và "/" với Priority là 2, tiếp theo là dấu "+" và "-" với Priority là 1, ưu tiên thấp nhất là dấu ngoặc mở "(" với Priority là 0.

Stack := ****;

for <Phần tử T đọc được từ biểu thức infix> do

{T có thể là hằng, biến, toán tử hoặc dấu ngoặc được đọc từ biểu thức infix theo thứ tự từ trái qua phải}

case T of

'(': Push(T);

')':

repeat

x := Pop;

if x ****'(' then Output(x);

until x = '(';

'+', '-', '\*', '/':

begin

while (Stack **** ****) and (Priority(T) ****  Priority(Get)) do Output(Pop); Push(T);

end;

else Output(T);

end;

while (Stack **** ****) do Output(Pop);

Ví dụ với biểu thức trung tố (2 \* 3 + 7 / 8) \* (5 - 1)

|  |  |  |  |
| --- | --- | --- | --- |
| Đọc | Xử lý | Stack | Output |
| ( | Đẩy vào Stack | ( |  |
| 2 | Hiển thị | ( | 2 |
| \* | Phép “\*” được ưu tiên hơn phần tử ở đỉnh Stack là “(“, đẩy “\*” vào Stack | (\* |  |
| 3 | Hiển thị | (\* | 2 3 |
| + | Phép “+” ưu tiên không cao hơn phần tử ở đỉnh Stack là “\*”, lấy ra và hiển thị “\*”. So sánh tiếp, thấy phép “+” được ưu tiên cao hơn phần tử ở đỉnh Stack là “(“, đấy “+” vào Stack | (+ | 2 3 \* |
| 7 | Hiển thị | (+ | 2 3 \* 7 |
| / | phép “/” được ưu tiên hơn phần tử ở đỉnh Stack là “+”, đấy “/” vào Stack | (+/ |  |
| 8 | Hiển thị | (+/ | 2 3 \* 7 8 |
| ) | Lấy ra và hiển thị các phần tử trong Stack tới khi lấy phải dấu ngoặc mở | **** | 2 3 \* 7 8 / + |
| \* | Stack đang là rỗng, đẩy \* vào Stack | \* |  |
| ( | Đẩy vào Stack | \*( |  |
| 5 | Hiển thị | \*( | 2 3 \* 7 8 / + 5 |
| - | phép “-“ được ưu tiên hơn phần tử ở đỉnh Stack là “(“, đẩy “-“ vào Stack | \*(- |  |
| 1 | Hiển thị | \*(- | 2 3 \* 7 8 / + 5 1 |
| ) | Lấy ra và hiển thị các phần tử ở đỉnh Stack cho tới khi lấy phải dấu ngoặc mở | \* | 2 3 \* 7 8 / + 5 1 - |
| Hết | Lấy ra và hiển thị hết các phần tử còn lai trong Stack |  | 2 3 \* 7 8 / + 5 1 - \* |

Dưới đây là chương trình chuyển biểu thức viết ở dạng trung tố sang dạng RPN. Biểu thức trung tố đầu vào sẽ được hiệu chỉnh sao cho mỗi thành phần của nó được cách nhau đúng một dấu cách, và thêm một dấu cách vào cuối cho dễ tách các phần tử ra để xử lý. Vì Stack chỉ dùng để chứa các toán tử và dấu ngoặc mở nên có thể mô tả Stack dưới dạng xâu ký tự cho đơn giản.

Ví dụ về Input / Output của chương trình:

Infix: (10 \* 3 + 7 / 8) \* (5 - 1)

Refined: ( 10 \* 3 + 7 / 8 ) \* ( 5 - 1 )

RPN: 10 3 \* 7 8 / + 5 1 - \*

P\_2\_07\_2.PAS \* Chuyển biểu thức trung tố sang dạng RPN

program ConvertInfixToRPN;

const

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

var

T, Infix, Stack: String; {Stack dùng để chứa toán tử và dấu ngoặc mở nên dùng String cho tiện}

p: Integer;

{Các thao tác đối với Stack}

procedure StackInit; begin

Stack := ''; end;

procedure Push(V: Char);

begin

Stack := Stack + V;

end;

function Pop: Char;

begin

Pop := Stack[Length(Stack)];

Dec(Stack[0]);

end;

function Get: Char;

begin

Get := Stack[Length(Stack)];

end;

procedure Refine(var S: String); {Hiệu chỉnh biểu thức trung tố về khuôn dạng dễ đọc nhất}

var

i: Integer;

begin

S := S + ' ';

for i := Length(S) - 1 downto 1 do {Thêm những dấu cách trước và sau mỗi toán tử và dấu ngoặc}

if (S[i] in Opt) or (S[i + 1] in Opt) then Insert(' ', S, i + 1);

for i := Length(S) - 1 downto 1 do {Xoá những dấu cách thừa}

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

end;

function Priority(Ch: Char): Integer; {Hàm lấy mức độ ưu tiên của Ch}

begin

case ch of

'\*', '/': Priority := 2;

'+', '-': Priority := 1;

'(': Priority := 0;

end;

end;

procedure Process(T: String); {Xử lý một phần tử đọc được từ biểu thức trung tố}

var

c, x: Char;

begin

c := T[1];

if not (c in Opt) then Write(T, ' ')

else

case c of

'(': Push(c);

')': repeat

x := Pop;

if x <> '(' then Write(x, ' ');

until x = '(';

'+', '-', '\*', '/':

begin

while (Stack <> '') and (Priority(c) <= Priority(Get)) do Write(Pop, ' ');

Push(c); end;

end;

end;

begin

Write('Infix = ');

ReadLn(Infix);

Refine(Infix);

WriteLn('Refined: ', Infix);

Write('RPN: ');

T := '';

for p := 1 to Length(Infix) do

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

else

begin

Process(T);

T := '';

end;

while Stack <> '' do Write(Pop, ' ');

WriteLn;

end.