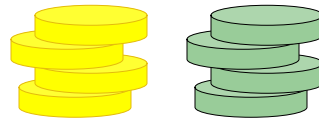


Cấu trúc dữ liệu và Giải thuật

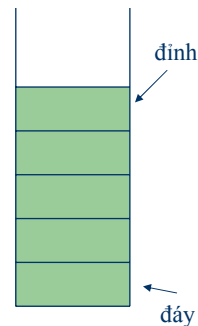
Chương III: Stack và Queue



Danh sách kiểu ngăn xếp - Stack

– Stack

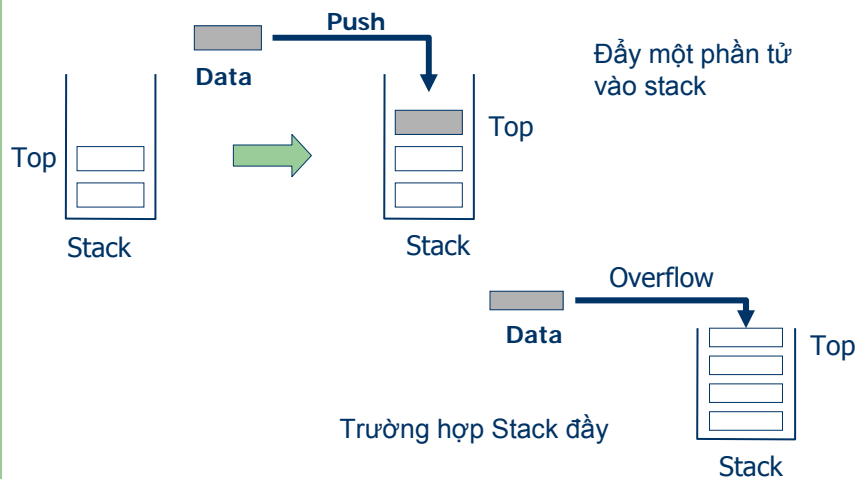
- Một kiểu danh sách tuyến tính đặc biệt
- Phép bổ sung và phép loại bỏ tuân thủ theo cơ chế “vào sau ra trước” (last in first out) , được thực hiện ở đầu đỉnh



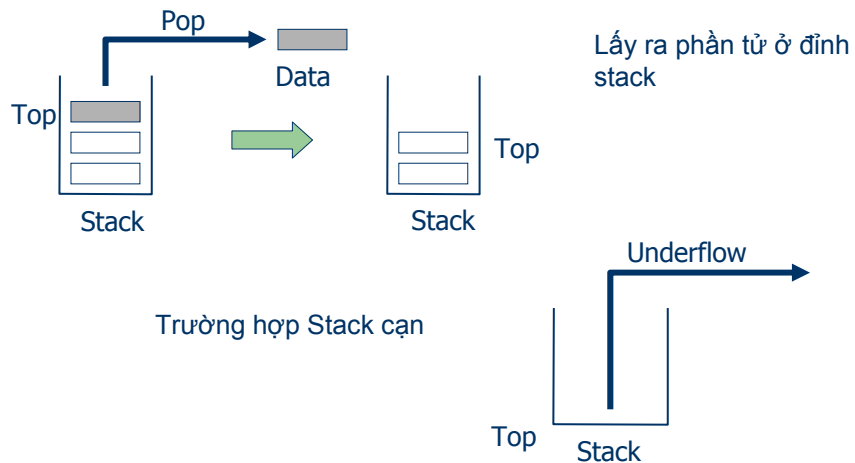
Danh sách kiểu ngăn xếp - Stack

- Hai thao tác cơ bản đối với danh sách kiểu ngăn xếp
 - push(Element e) : bổ sung phần tử vào Stack
 - Element pop(): Loại bỏ và trả ra giá trị của phần tử ở đỉnh Stack
- Các thao tác khác
 - Int size(): Trả ra số các phần tử trong Stack
 - Boolean isEmpty(): Kiểm tra xem Stack có rỗng không
 - Element top(): Trả ra giá trị của phần tử ở đỉnh Stack

Các thao tác cơ bản của Stack



Các thao tác cơ bản của Stack



Danh sách kiểu ngăn xếp

Thao tác	Output	Stack
create()	-	[]
push(5)	-	[5]
push(3)	-	[5,3]
pop()	3	[5]
push(7)	-	[5,7]
top()	7	[5,7]
pop()	7	[5]
pop()	5	[]
isEmpty()	true	[]
push(9)	-	[9]
push(8)	-	[9,8]
push(7)	-	[9,8,7]
size()	3	[9,8,7]

Ứng dụng của Stack

- Lưu trữ các trang web đã từng được duyệt trên Web browser
- Cài đặt thao tác Undo trong các phần mềm soạn thảo
- Lưu danh sách các lời gọi hàm trong Java Virtual Machine

Lưu trữ kế tiếp của Stack

- Stack có thể được lưu trữ bởi một vector lưu trữ S , gồm n ô nhớ kế tiếp nhau
- Đỉnh stack được xác định bởi một chỉ số T
 - T sẽ được cập nhật nếu có thao tác bổ sung hay loại bỏ được thực hiện trên stack



Lưu trữ kế tiếp của Stack

- Giải thuật bổ sung một phần tử vào Stack được lưu trữ kế tiếp

Procedure PUSH(S,T,X)

Begin

{S: vector lưu trữ có n ô nhớ; T: chỉ số của phần tử đỉnh stack hiện thời; X là giá trị cần thêm vào }

- if $T \geq n$ then begin
 write('STACK TRÀN');
 return;
end;
 - $T := T + 1$;
 $S[T] := X$;
- End

Lưu trữ kế tiếp của Stack

- Giải thuật lấy ra phần tử ở đỉnh của Stack được lưu trữ kế tiếp

Procedure POP(S,T, Y)

Begin

{S: stack đang xét ; T: chỉ số của phần tử tại đỉnh stack hiện thời;
Phần tử được lấy ra sẽ được bảo lưu sử dụng biến Y }

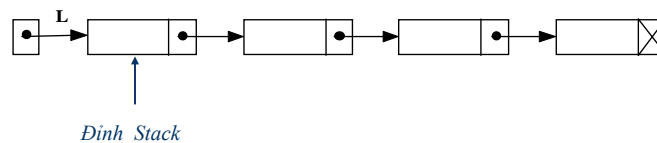
- if $T = 0$ then begin
 write('STACK CẠN'); return;
end;
 - $Y := S[T]$;
 $S[T] := \text{null}$;
 $T := T - 1$;
- End

Hiệu năng và Hạn chế

- Hiệu năng
 - n là số phần tử của stack
 - Không gian lưu trữ : $O(n)$
 - Các thao tác cơ bản có độ phức tạp $O(1)$
- Hạn chế
 - Kích thước tối đa phải được xác định trước và không được thay đổi
 - Xảy ra tràn stack

Lưu trữ móc nối đối với Stack

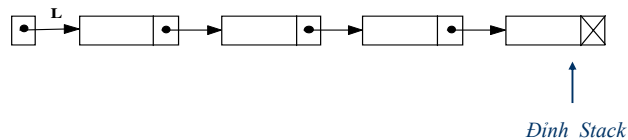
- Cách tiếp cận 1
 - Đỉnh của Stack được coi là phần tử nằm ở đầu danh sách
 - `pop()` : Lấy ra phần tử đầu tiên trong danh sách móc nối
 - `push(o)` : Bổ sung một phần tử vào đầu danh sách móc nối



Lưu trữ móc nối đối với Stack

- Cách tiếp cận 2

- Phần tử cuối cùng được coi là đỉnh stack
- pop() : Lấy ra phần tử cuối cùng trong danh sách móc nối
- push(o): Bổ sung một phần tử vào cuối danh sách móc nối



- Cách lưu trữ móc nối nào phù hợp hơn đối với cấu trúc dữ liệu Stack?

Lưu trữ móc nối đối với Stack

- Khai báo Stack móc nối trong C

```
struct stacknode {  
    int item;  
    struct stacknode *next;  
};  
typedef struct stacknode STACKNODE;  
typedef STACKNODE * STACKNODEPTR;  
STACKNODEPTR top = NULL;
```

Lưu trữ móc nối đối với Stack

– Bổ sung vào Stack

```
int push ( STACKNODEPTR *top , int value ) {  
    STACKNODEPTR newnode;  
    newnode = malloc sizeof (STACKNODE);  
    if (nut == null) { printf("\n No memory"); return 1; }  
    else {  
        newnode->item = value;  
        newnode ->next = *top;  
        *top = newnode;  
        return 0;  
    }  
}
```

Lưu trữ móc nối đối với Stack

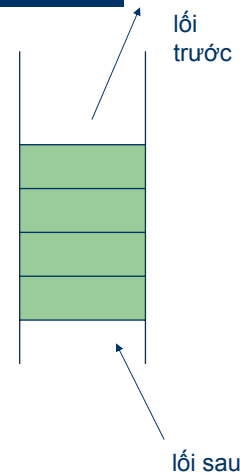
– Loại bỏ nút

```
int pop ( STACKNODEPTR *top) {  
    int item; STACKNODEPTR temp;  
    temp = *top;  
    item = (*top)->item;  
    *top = (*top)->next;  
    free (temp);  
    return item;  
}
```


Danh sách kiểu hàng đợi - Queue

- Queue

- Queue (Hàng đợi) là một kiểu danh sách tuyến tính đặc biệt
- Phép bổ sung và loại bỏ hoạt động theo cơ chế “vào trước ra trước” (first in first out) ; bổ sung ở một đầu thì loại bỏ ở đầu kia



Danh sách kiểu hàng đợi - Queue

- Hai hàm cơ bản đối với danh sách kiểu hàng đợi
 - enqueue(Element e)
 - Element dequeue()
- Các hàm khác
 - create():
 - size() :
 - isEmpty():
 - Element front()

Danh sách kiểu hàng đợi – Queue

Thao tác	Output	Queue
create()	-	[]
enqueue(5)	-	[5]
enqueue(3)	-	[5,3]
dequeue()	5	[3]
enqueue(7)	-	[3,7]
front()	3	[3,7]
dequeue()	3	[7]
dequeue()	7	[]
isEmpty()	true	[]
enqueue(9)	-	[9]
enqueue(8)	-	[9,8]
enqueue(7)	-	[9,8,7]
size()	3	[9,8,7]

Ứng dụng của Queue

- Hàng đợi trong các phòng bán vé
- Truy nhập vào các thiết bị dùng chung tại văn phòng (ví dụ máy in)

Lưu trữ kế tiếp đối với Queue

- Sử dụng một vector lưu trữ Q gồm n ô nhớ kế tiếp nhau để biểu diễn một Queue
- Cần nắm được hai chỉ số
 - R : Chỉ số của phần tử nằm ở lồi sau của Q
 - F : Chỉ số của phần tử ở lồi trước của Q

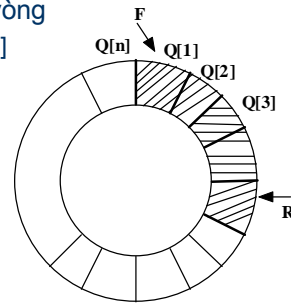


Lưu trữ kế tiếp đối với Queue

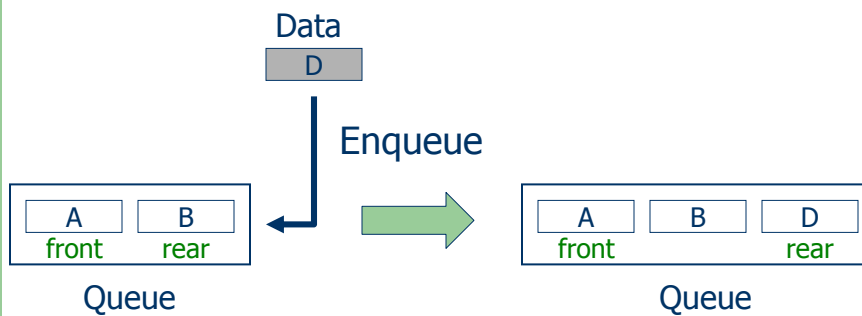
- Khi Queue rỗng thì $F = R = 0$
- Khi bổ sung thêm một phần tử vào Queue thì R tăng lên 1
- Khi lấy ra một phần tử trong Queue thì F tăng lên 1
- Nhược điểm của cách tổ chức lưu trữ này
 - Các phần tử trong Queue sẽ dịch chuyển khắp không gian nhớ nếu liên tục thực hiện bổ sung rồi loại bỏ
 - Hiện tượng TRÀN vẫn xảy ra khi vector lưu trữ Q vẫn còn chỗ nhưng $R = n$

Lưu trữ kế tiếp đối với Queue

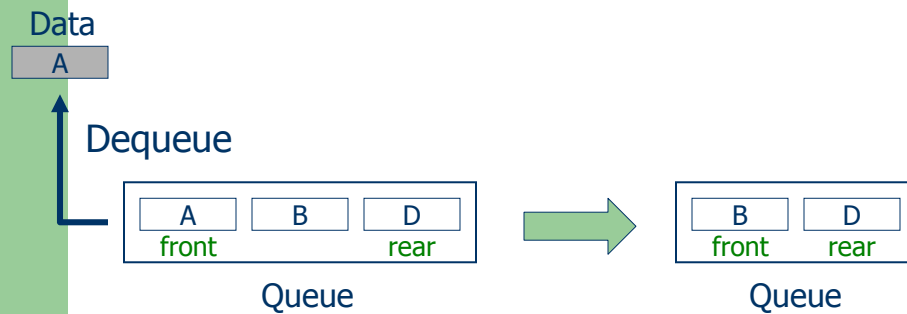
- Khắc phục các vấn đề bằng cách coi vector lưu trữ Queue được tổ chức dưới dạng vòng
 - $Q[1]$ được coi như đứng sau $Q[n]$



Các thao tác cơ bản của Queue



Các thao tác cơ bản của Queue



Lưu trữ kế tiếp đối với Queue

- Giải thuật bổ sung vào Queue được lưu trữ trong vector Q gồm n phần tử và được tổ chức dưới dạng thường

```
Procedure ENQUEUE(Q,F,R,X)
Begin
  1. if (R >= n) then begin
        write('QUEUE TRÀN');
        return;
      end;
  2. {Q rỗng} if F = 0 then F:= R:= 1;
  3. else R:= R+ 1;
  4. Q[R] := X;
End
```

Lưu trữ kế tiếp đối với Queue

- Giải thuật lấy ra (loại bỏ) khỏi Queue

Procedure DEQUEUE(Q,F,R, Y)

Begin

{ Y là biến lưu trữ phần tử được lấy ra }

1. if F = 0 then begin

 write('QUEUE CHẶN');

 return;

end;

2. Y:= Q[F]; {lưu giá trị của phần tử cần lấy}

3. if F = R=1 then F:= R:= 0; { Queue chỉ còn một phần tử}

4. else F:= F+ 1;

End

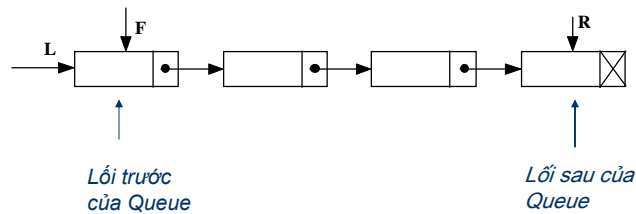
Lưu trữ kế tiếp đối với Queue

- Bài tập: Hãy viết giải thuật thực hiện bổ sung và loại bỏ trên Queue lưu trữ kế tiếp dưới dạng vòng

Lưu trữ móc nối đối với Queue

– Cách tiếp cận 1: Sử dụng danh sách nối đơn

- Lối trước của Queue là đầu danh sách
- enqueue(o): bổ sung phần tử vào cuối danh sách
- dequeue() : loại bỏ phần tử ở đầu danh sách

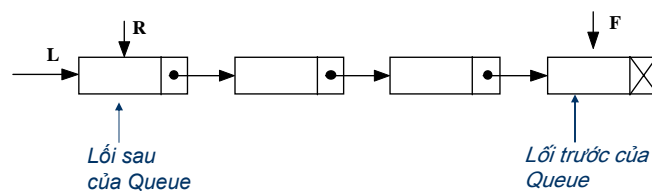


- Luôn nắm giữ hai con trỏ F trỏ tới phần tử ở lối trước của queue, R trỏ tới phần tử ở lối sau của queue

Lưu trữ móc nối đối với Queue

– Cách tiếp cận 2:

- Lối sau của Queue là đầu danh sách
- enqueue(o): bổ sung phần tử vào đầu danh sách
- dequeue() : loại bỏ phần tử ở cuối danh sách



Lưu trữ móc nối đối với Queue

- Giải thuật bổ sung một phần tử vào Queue lưu trữ trong danh sách móc nối – Bổ sung vào cuối danh sách

Procedure ENQUEUE(F,R,X)

Begin

1. {Khởi tạo nút mới} Call New(p);
 INFO(p) := X; LINK(p) := Null;
2. {Danh sách đã cho rỗng} if F = Null then F:= R:= p;
3. else LINK(R) := p; R:= p;

End

Lưu trữ móc nối đối với Queue

- Giải thuật loại bỏ phần tử khỏi Queue – Loại bỏ phần tử đầu tiên trong danh sách

Procedure DEQUEUE(F,R, Y)

Begin

{ Y là biến lưu trữ phần tử được lấy ra }

1. p:= F; Y:= INFO(p);
2. {Danh sách ban đầu chỉ có một phần tử}
 if (F = R) and (F <> Null) then F:= R:= Null;
2. else F:= LINK(p);
3. Call Dispose(p) ;

End

Hàng đợi hai đầu - DeQueue

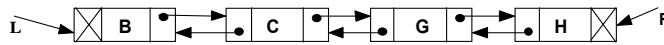
- DeQueue
 - Hàng đợi hai đầu là một cấu trúc dữ liệu dạng hàng đợi nhưng nó hỗ trợ phép bổ sung và loại bỏ ở cả đầu và cuối
- Các hàm cơ sở của hàng đợi hai đầu D
 - insertFirst(o)
 - insertLast(o) :
 - removeFirst()
 - removeLast()
- Các hàm khác
 - first()
 - last()
 - size()
 - isEmpty()
 - create()

Hàng đợi hai đầu - DeQueue

Thao tác	Output	DeQueue
create()	-	[]
insertFirst(5)	-	[5]
insertFirst(3)	-	[3,5]
removeFirst()	3	[5]
insertLast(7)	-	[5,7]
removeFirst()	3	[7]
removeLast()	7	[]
removeLast()	error	[]
isEmpty()	true	[]
insertLast(9)	-	[9]
insertFirst(8)	-	[8,9]
insertLast(7)	-	[8,9,7]
size()	3	[8,9,7]

Lưu trữ móc nối với DeQueue

- DeQueue được lưu trữ sử dụng cấu trúc danh sách móc nối kép (Doubly Linked – List)
 - Mỗi nút trong danh sách ngoài trường INFO chứa dữ liệu còn có 2 trường con trỏ
 - PREV
 - NEXT
 - Cần nắm được hai con trỏ, con trỏ L trỏ tới nút cực trái, con trỏ R trỏ tới nút cực phải của danh sách
 - Với danh sách rỗng, $L = R = \text{NULL}$



Lưu trữ móc nối đối với DeQueue

- Giải thuật bổ sung phần tử vào đầu một DeQueue lưu trữ trong một danh sách nối kép
- Giải thuật loại bỏ phần tử đầu một DeQueue lưu trữ trong một danh sách nối kép

Bài toán đổi số cơ số

– Bài toán: Viết một số trong hệ thập phân thành số trong hệ cơ số b bất kỳ

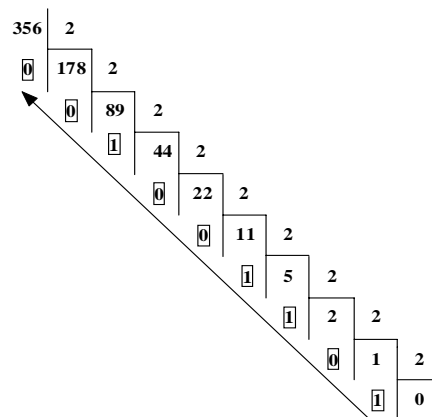
• Ví dụ:

- $(356)_{10} = (101100100)_2$
- $(356)_{10} = (544)_8$
- $(356)_{10} = (164)_{16}$

Bài toán đổi cơ số sử dụng Stack

• Ví dụ:

- $(356)_{10} = (101100100)_2$

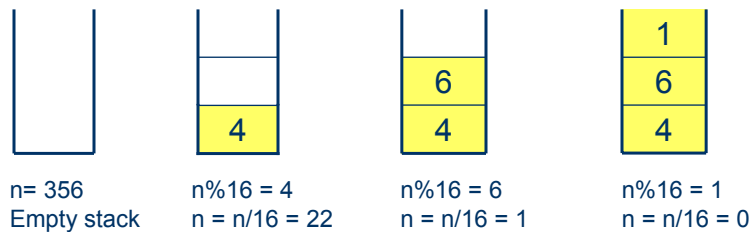


Bài toán đổi cơ số sử dụng Stack

– Thuật toán:

- Đầu vào: Số n trong hệ thập phân
- Đầu ra: Số tương ứng với n trong hệ đếm cơ số b
- Thực hiện
 1. Lấy chữ số tạo bởi $n \% b$. Đẩy vào Stack
 2. Thay n bằng n/b để tiếp tục lấy các chữ số tiếp theo trong kết quả
 3. Lặp lại bước 1 và 2 cho đến khi kết quả của phép chia là 0
 4. Lần lượt lấy các chữ số ra khỏi Stack và in chúng ra kết quả

Bài toán đổi cơ số sử dụng Stack



Bài toán đổi cơ số sử dụng Stack

Procedure CONVERT(n, b)

Begin

1. $m = N$;

2. { Tính số dư và nạp vào stack S }

while $m \neq 0$ do begin

$R := m \bmod b$; call PUSH(S, T, R);

$m := m \div b$; {thay m bằng thương của phép chia m cho b}

end;

3. {Hiện thị từng chữ số nhị phân trong mã số biểu diễn N}

while $T \neq 0$ do begin call POP(S,T,X); {lấy số ra khỏi stack} write(X);

end

End

Bài toán kiểm tra cặp ngoặc

– Kiểm tra cặp ngoặc

Mỗi dấu “(”, “{”, or “[” đều phải có một dấu đóng tương ứng “)”, “}”, or “]”

- Đúng: () (()) { ([]) }
- Đúng : ((() (()) { ([]) }
- Sai:) (() { ([]) }
- Sai: ({ [] }
- Sai: (

– Viết giải thuật nhận một xâu đầu vào gồm các ký tự mở , đóng ngoặc. Kiểm tra xâu có hợp lệ không

Function ParenMatch(X, n):

{ X là một chuỗi bao gồm n ký tự mở, đóng ngoặc. Giải thuật trả ra giá trị **true** nếu chuỗi X chứa một số hợp lệ cặp ngoặc, nếu không trả ra giá trị false }

Khởi tạo S là một stack rỗng

for $i = 1$ to n **do begin**

if $X[i]$ là một ngoặc mở **then**

$S.push(X[i])$

else if $X[i]$ là một ngoặc đóng **then begin**

if $S.isEmpty()$ **then**

return false /không có ngoặc mở tương ứng/

if $S.pop()$ không hợp kiểu với $X[i]$ **then**

return false /cặp ngoặc đóng mở khác kiểu/

end

end

if $S.isEmpty()$ **then**

return true /tất cả cặp ngoặc hợp lệ/

else

return false /vẫn tồn tại một số ngoặc mở mà không tìm thấy ngoặc đóng tương ứng/

Biểu thức số học với ký pháp Balan

- Thông thường, một biểu thức số học được biểu diễn theo ký pháp trung tố (infix notation)
 - Dấu phép toán (toán tử) nằm giữa 2 toán hạng
 - $A+B*C$
 - Thứ tự thực hiện các phép toán được xác định sử dụng các cặp dấu ngoặc hoặc quy định một thứ tự ưu tiên giữa các phép toán
 - Biểu thức $A * B^2 - C/D + E$ được thực hiện theo thứ tự sau
$$B^2 \rightarrow A*(B^2) \rightarrow C/D \rightarrow (A*(B^2)) - (C/D) \rightarrow ((A*(B^2)) - (C/D)) + E$$
 - Tính toán giá trị biểu thức sẽ khá phức tạp

Biểu thức số học với ký pháp Balan

- Có thể biểu diễn các biểu thức mà không dùng đến dấu ngoặc sử dụng ký pháp tiền tố (prefix notation) hoặc ký pháp hậu tố (postfix notation)
- Biểu thức dạng tiền tố và hậu tố
 - Trong ký pháp dạng tiền tố: Toán tử luôn được đặt trước 2 toán hạng

Toán tử

Toán hạng 1

Toán hạng 2

- Trong ký pháp dạng hậu tố: Toán tử luôn đặt sau 2 toán hạng

Toán hạng 1

Toán hạng 2

Toán tử

Biểu thức số học với ký pháp Balan

Dạng trung tố	Dạng tiền tố	Dạng hậu tố
$A+B$	$+ A B$	$A B +$
$(A+B) * C$	$* + A B C$	$A B + C *$
$A + B * C$	$+ A * B C$	$A B C * +$
$(A + B) / (C-D)$	$/ + A B - C D$	$A B + C D - /$
$A + B / C - D$	$- + A / B C D$	$A B C / + D -$

Bài toán tính giá trị của biểu thức dạng hậu tố

- Tính giá trị của một biểu thức dạng hậu tố sử dụng Stack
 - Đầu vào : Xâu ký tự biểu diễn biểu thức dạng hậu tố
 - $A B + C - D E * /$
 - Các giá trị của các biến số
- Các bước chính trong giải thuật
 - Đọc biểu thức dạng hậu tố từ trái qua phải
 - Nếu ký tự được đọc là một toán hạng thì lưu giá trị vào stack
 - Nếu ký tự được đọc là một toán tử X thì lần lượt lấy từ stack ra 2 giá trị, thực hiện phép toán X với 2 giá trị đó, nạp kết quả vào stack
 - Thực hiện các bước trên đến khi toàn bộ biểu thức đã được đọc

Bài toán tính giá trị của biểu thức dạng hậu tố

Procedure EVALUATE (P, VAL)

Begin { P là biểu thức dạng hậu tố cần tính, VAL là biến sẽ lưu giá trị tính được }

1. Ghi thêm dấu ‘)’ vào cuối P để đánh dấu điểm kết thúc

2. repeat

 Đọc ký tự X trong P (lần lượt từ trái sang phải) ;

 if X là một toán hạng then call PUSH(S, T, X) ;

 else begin

 call POP(S, T, Y) ; call POP(S, T, Z);

 Thực hiện phép toán với hai toán hạng Z,Y kết quả là W;

 call PUSH (S, T, W) ;

 end;

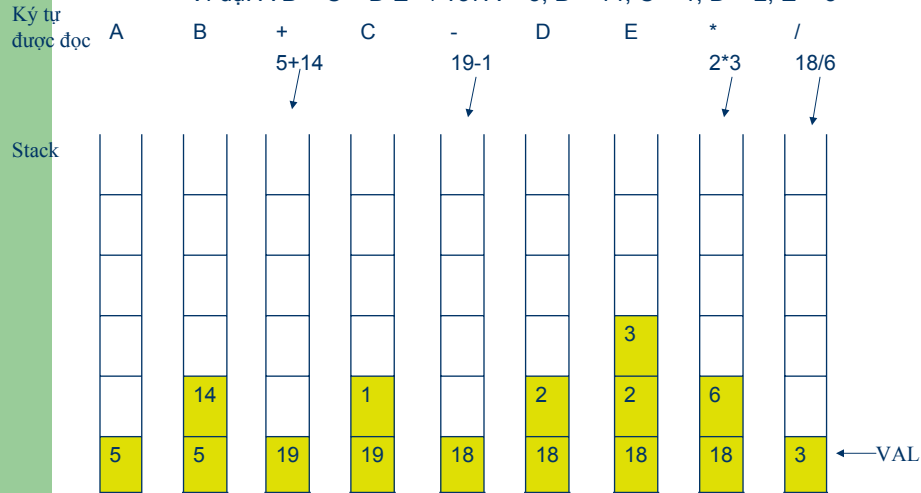
until Gặp dấu kết thúc xâu ‘)’ ;

3. call POP (S,T, VAL);

End

Bài toán tính giá trị của biểu thức dạng hậu tố

- Ví dụ: $A B + C - D E * /$ với $A = 5, B = 14, C = 1, D = 2, E = 3$



Chuyển biểu thức dạng trung tố sang dạng hậu tố

- Bài toán
 - Xét biểu thức số học dạng trung tố gồm các phép toán cộng, trừ, nhân, chia, lũy thừa và các dấu ngoặc
 - Viết biểu thức dạng hậu tố tương ứng với biểu thức trung tố đầu vào
- Để thực hiện, trong biểu thức trung tố cần biết
 - Thứ tự ưu tiên của các phép toán : Lũy thừa → Nhân, Chia → Cộng, Trừ
 - Quy tắc kết hợp: Nếu có hai phép toán cùng thứ tự ưu tiên
 - Lũy thừa: Phải trước, trái sau. $2^3^4 = 2^{(3^4)}$
 - Các phép toán khác : Trái trước, phải sau
 - Dấu ngoặc : Ưu tiên nhất

Chuyển biểu thức dạng trung tố sang dạng hậu tố

– Thực hiện

- Duyệt biểu thức trung tố từ trái sang phải
 - Gặp toán hạng → Đưa ra
 - Gặp phép toán → Cần đợi toán hạng số 2 → Phải lưu lại
 - Sử dụng Stack để lưu trữ
- Cần xác định các trường hợp để đưa phép toán ra cho chính xác

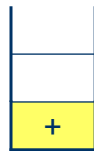
Chuyển biểu thức dạng trung tố sang dạng hậu tố

- Tình huống:
 - Có các phép toán đang được lưu trữ trong Stack
 - Khi duyệt biểu thức dạng trung tố, đang gặp phải một phép toán khác
 - Xác định các trường hợp để đưa phép toán từ Stack ra cho chính xác với biểu thức dạng hậu tố
- Giải pháp: So sánh phép toán đang xét với phép toán được lưu ở đỉnh Stack về thứ tự ưu tiên, qui tắc kết hợp

Chuyển biểu thức dạng trung tố sang dạng hậu tố

– Quy tắc đưa các phép toán khỏi Stack:

- Quy tắc cơ bản: Nếu phép toán đang xét có thứ tự ưu tiên **thấp hơn** phép toán ở đỉnh Stack → Đưa phép toán ở đỉnh Stack ra



$1+2 * 3$
Đã đưa ra 1 2
Đang xét +
Không đưa + ra



$1*2 + 3$
Đã đưa ra 1 2
Đang xét +
Đưa * ra

Chuyển biểu thức dạng trung tố sang dạng hậu tố

– Quy tắc khi có hai phép toán cùng mức ưu tiên

- Nếu phép toán có tính chất **kết hợp trái** (cộng, trừ, nhân, chia) thì đưa phép toán ở đỉnh ngăn xếp ra



$1 - 2 + 3$
Đang xét +
Đưa - ra
Hậu tố tương ứng 1 2 - 3 +

- Nếu phép toán có tính chất kết hợp phải (lũy thừa) thì không đưa phép toán ở đỉnh ngăn xếp ra



$2 ^ 3 ^ 4$
Đang xét ^ thứ 2
Không đưa ^ ở đỉnh Stack ra
Hậu tố tương ứng 2 3 4 ^ ^

Chuyển biểu thức dạng trung tố sang dạng hậu tố

- Trường hợp hàng loạt phép toán dờn ngăn xếp

	1 + 2 * 3 - 4
*	Đang xét -
+	Phải đưa ra * rồi +
	Hậu tố 1 2 3 * + 4 -

- Dấu ngoặc:
 - Khi đọc mở ngoặc, đẩy vào Stack , không đưa bất kỳ phép toán nào ra
 - Khi dấu mở ngoặc trong Stack sẽ không đưa nó ra khi đang xét các phép toán
 - Khi đọc đóng ngoặc, đưa tất cả các phép toán trong Stack ra cho đến khi gặp dấu mở ngoặc
 - Các dấu ngoặc không đưa vào trong biểu thức hậu tố

Chuyển biểu thức dạng trung tố sang dạng hậu tố

- Các bước trong giải thuật
 - Duyệt biểu thức trung tố từ trái sang phải
 - 1. Gặp toán hạng đưa ra ngay
 - 2. Gặp dấu (→ Đưa vào ngăn xếp
 - 3. Gặp dấu) → đưa các phần tử ra khỏi ngăn xếp cho đến khi gặp dấu (, dấu (được lấy ra khỏi ngăn xếp
 - 4. Nếu gặp dấu phép toán : so sánh độ ưu tiên của phép toán đang xét và phép toán ở đỉnh của stack
 - Nếu phép toán đang xét có độ ưu tiên lớn hơn phép toán ở đỉnh stack hoặc phép toán đang xét có độ ưu tiên bằng phép toán ở đỉnh và đó là phép toán ^, đẩy phép toán đang xét vào đỉnh stack
 - Nếu không, đưa lần lượt phép toán ở đỉnh stack ra cho đến khi thấy một phép toán tại đỉnh stack có độ ưu tiên thấp hơn phép toán đang xét.
 - Đẩy phép toán đang xét vào đỉnh stack
 - 5. Kết thúc biểu thức → Đưa nốt các phép toán trong ngăn xếp ra

