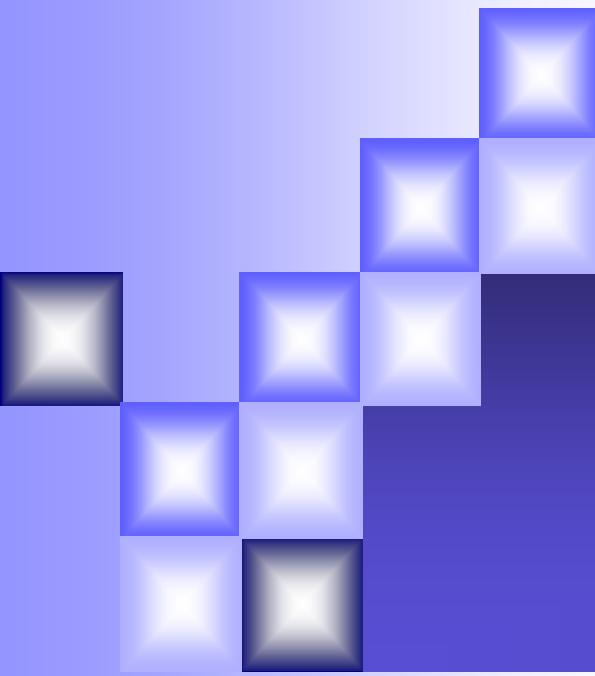


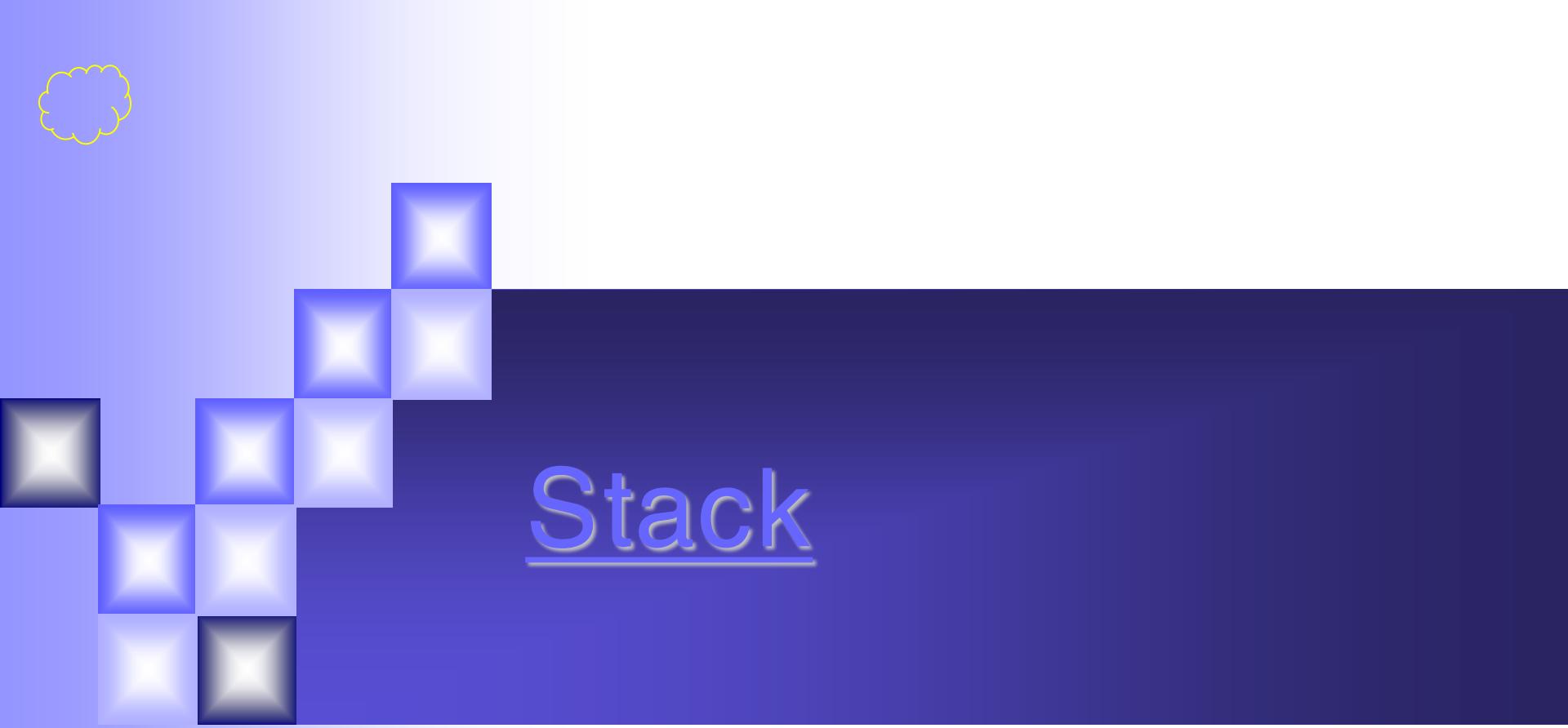
Chương 3:

Danh sách liên kết đơn (DSLK đơn)



Phần II: Các cấu trúc đặc biệt của danh sách đơn

- Stack (*Ngăn xếp*)
- Queue (*Hàng đợi*)



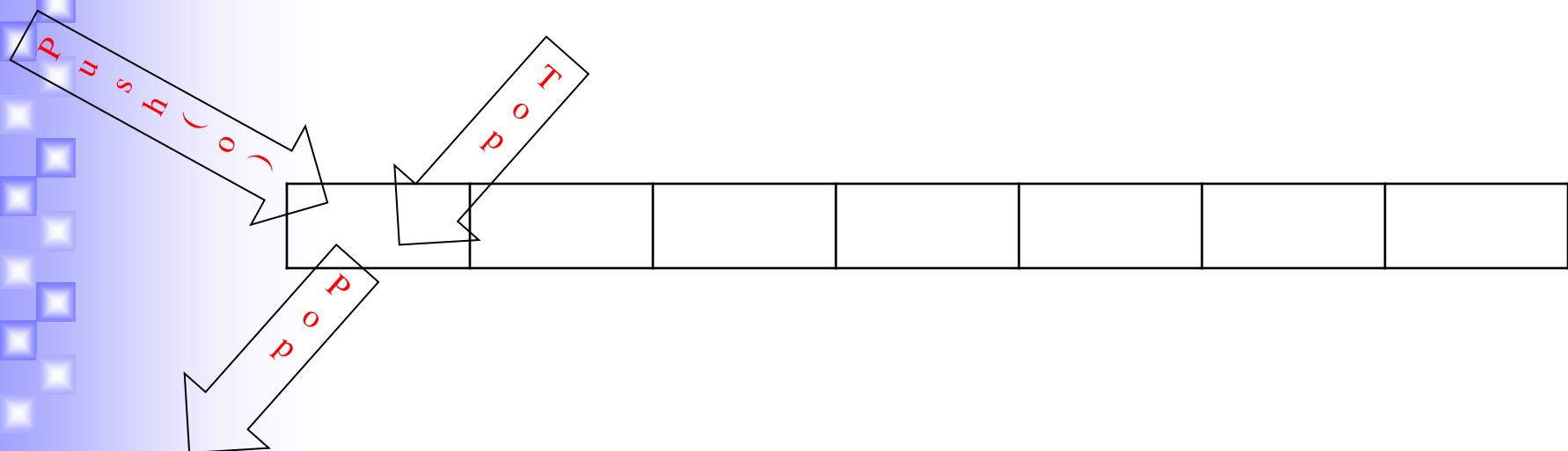
Stack

Stack

- Stack là một vật chứa (container) các đối tượng:
 - ❖ làm việc theo cơ chế LIFO (*Last In First Out, Vào sau ra trước*)
⇒ Việc thêm một đối tượng vào stack và lấy một đối tượng ra khỏi stack được thực hiện trên cùng một đầu.
 - ❖ có nhiều ứng dụng:
 - khử đệ qui
 - tổ chức lưu vết các quá trình tìm kiếm theo chiều sâu và quay lui,
 - ứng dụng trong các bài toán tính toán biểu thức, ...

Stack

- Stack là một CTDL trừu tượng hỗ trợ 2 thao tác chính:
 - **Push(o)** :Thêm đối tượng o vào đầu stack
 - **Pop()** :
 - Lấy đối tượng ở đầu stack ra khỏi stack và trả về giá trị của nó.
 - Nếu stack rỗng thì lỗi sẽ xảy ra.



- Stack cũng hỗ trợ một số thao tác khác:

- **isEmpty()**: Kiểm tra xem stack có rỗng không.
 - **Top()**: Trả về giá trị của phần tử nằm ở đầu stack mà không hủy nó khỏi stack. Nếu stack rỗng thì lỗi sẽ xảy ra.

Biểu diễn Stack:

Thường dùng:

- mảng
- danh sách liên kết đơn

Ở đây ta dùng DSLK đơn

Biểu diễn Stack dùng danh sách liên kết đơn

- Có thể tạo một stack bằng cách sử dụng một danh sách liên kết đơn (DSLK).
- DSLK có những đặc tính rất phù hợp để dùng làm stack vì mọi thao tác đều có thể thực hiện ở đầu DSLK.

Kiểu STACK cài đặt bằng danh sách liên kết đơn

- Giả sử đã có các định nghĩa:

```
typedef int data;  
struct tagNode  
{  
    data           info;  
    tagNode*     pNext;  
};  
// kiểu của một phần tử trong danh sách  
typedef tagNode NODE;
```

- Cài đặt kiểu STACK (Danh sách liên kết) :

```
struct STACK  
{  
    NODE* pHead;  
    NODE* pTail;  
};
```

- Tạo Stack S rỗng:

```
void CreatStack(STACK &s)
{
    s.pHead=s.pTail= NULL;
}
```

- Kiểm tra stack rỗng :

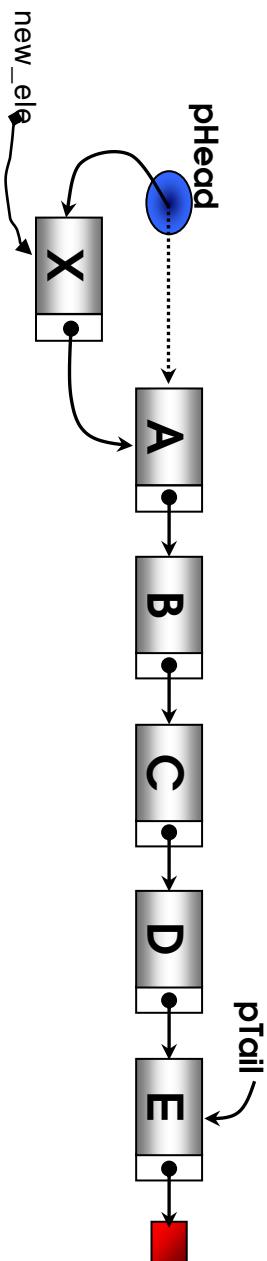
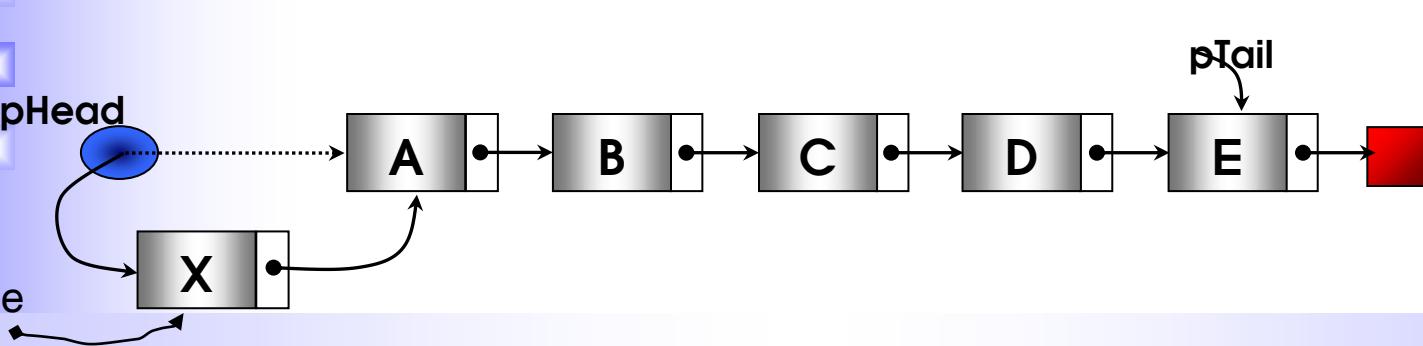
```
int IsEmpty(STACK s)
{ if (s.pHead == NULL) // stack rỗng
    return 1;
    return 0;
}
```

- Thêm một phần tử x vào đầu stack S:

```
void Push(STACK &s, data x)
{
    InsertHead(s, x);
}
```

Viết trực tiếp như sau:

```
void Push(STACK &s, data x)
{ NODE* new_ele = GetNode(x);
  if (s.pHead==NULL) //Stack rỗng
  {
    s.pHead = new_ele; s.pTail = s.pHead;
  }
  else
  {
    new_ele->pNext = s.pHead;
    s.pHead = new_ele;
  }
}
```



- Lấy thông tin và hủy phần tử ở đỉnh stack S

```
data Pop(STACK &s)
{
    data x;
    if (s.pHead==NULL)
        return NULldata;
    x = RemoveHead(s);
    return x;
}
```

//NULldata là một giá trị đặc biệt nào đó được định nghĩa trước.

Viết trực tiếp như sau :

```
data Pop(STACK &s)
{ NODE *p;
  data x;
  if (s.pHead == NULL)
    return NULLDATA;
  p = s.pHead;
  x = p->info;
  s.pHead = s.pHead->pNext;
  delete p;
  if(s.pHead==NULL)
    s.pTail = NULL;
  return x;
}
```

- Trả về phần tử ở đỉnh stack S

```
data Top(STACK s)
{
    if(s.pHead == NULL)
        return NULLDATA;
    return s.pHead->info;
}
```

Ứng dụng

- Ngăn xếp có nhiều ứng dụng: *khử đệ quy, lưu vết* trong thuật toán *quay lui*, hay *tìm kiếm theo chiều sâu*, trong việc *chuyển đổi giữa các dạng kí pháp* khác nhau cũng như đánh giá các biểu thức chứa các toán tử không quá hai ngôi như biểu thức số học, lô-gic, ...
- Trong phần này ta xét các ứng dụng stack:
 - Chuyển đổi một biểu thức trung tố (toán tử đặt ở giữa hai toán hạng - đây là cách viết thông thường) sang dạng hậu tố (*toán tử đặt sau các toán hạng*).
 - Đánh giá các biểu thức số học (dạng hậu tố) : Sử dụng *ký pháp nghịch đảo Balan* (hay còn gọi là *ký pháp hậu tố RPN* - Reverse Polish Notation).

Chuyển biểu thức trung tố sang hậu tố RPN: Thuật toán POLISH

Input : sin (biểu thức trung tố, là một chuỗi . . .)

Output : sout (biểu thức hậu tố, là một chuỗi . . .)

Mô tả thuật toán :

B1. Khởi tạo stack s rỗng (dùng để chứa các toán tử);

B2. Lặp lại các việc sau cho đến khi gặp dấu kết thúc biểu thức (NULL):

Đọc phần tử (ký tự) tiếp theo (hằng, biến, toán tử, ‘(, ’)) trong biểu thức trung tố.

Nếu phần tử là:

- Toán hạng (hằng hoặc biến) : Cho ra output (sout).

- Dấu ‘(: đẩy nó vào s;

- Dấu ’): lấy các toán tử trong stack ra và cho vào output cho đến khi
gặp dấu mở ngoặc mở “(“.

(Dấu mở ngoặc cũng phải được đưa ra khỏi stack nhưng không cho vào Output)

- Toán tử:

- * Chừng nào đỉnh stack còn là toán tử và toán tử đó có độ ưu tiên **lớn hơn hoặc bằng** toán tử hiện tại thì lấy toán tử đó ra khỏi stack và cho vào output.
- * Đưa toán tử hiện tại vào stack s.

B3. *Khi gặp tín hiệu kết thúc biểu thức* thì lần lượt lấy các toán tử trong stack s ra và đưa vào output cho đến khi s rỗng;

+ Lưu ý về độ ưu tiên của các toán tử số học 2 ngôi :

Các toán tử 2 ngôi : ‘*’, ‘/’, ‘%’ có cùng độ ưu tiên và có độ ưu tiên cao hơn các toán tử sau (cùng độ ưu tiên) ‘+’, ‘-’, và qui ước rằng các toán tử trên có độ ưu tiên cao hơn toán tử () .

Minh họa: (các toán hạng là ký số)

- Input: $2*3+(6-4)$ //Biểu thức trung tố
- Output: $2\ 3\ *\ 6\ 4\ -\ +$ //Biểu thức hậu tố RPN

	Sin[i]	Stack S	Sout		Sin[i]	Stack S	Sout
			“”		5	(+()
1	2		“2”		6	6	“23*6”
2	*	*			7	-	+(-)
3	3	*	“23”		8	4	“23*64”
4	+	+	“23*”		9)	“23*64-”
					10	NULL	“23*64-+”

Vậy: $Sout = “23*64-+”$

```

// ham chuyen tu trung to sang hau to
void TrungTo_HauTo(char sin[MAX], char sout[MAX])
{
    STACK s;//stack luu cac toan tu
    char c, //Ky tu hien hanh - dang xet
    x; //luu toan tu o dinh stack trong thao tac Pop(s);
    int i; //duyet chuoi vao : sin
    sout[0] = NULL; //Khong khoi dong se bi loi : Access violation writing location
    CreatStack(s);
    for (i = 0; sin[i] != NULL; i++)
    {
        c = sin[i];//ky tu dang xet
        if (LaToanHang(c) == 1) //la ky so : toan hang
            Chen_Cuoi_Chuoi(sout, c); //chen c vao cuoi sout
        else
            if (c == '(')
                Push(s, c);//day '(' vao stack s
            else
                if (c == ')')
                    {
                        x = Pop(s);
                        while (x != '(') //khong dua '(' vao sout
                            {
                                Chen_Cuoi_Chuoi(sout, x);
                                x = Pop(s);
                            }
                    }
    }
}

```

```
        else // la toan tu
        {
            while (s.pHead != NULL)
                if (LaToanTu(s.pHead->info) == 1 &&
                    Do_UuTien_ToanTu(s.pHead->info) >= Do_UuTien_ToanTu(c))
                {
                    x = Pop(s);
                    Chen_Cuoi_Chuoi(sout, x);
                }
                else
                    break;
            //Push toan tu dang xet vao s
            Push(s, c);
        }

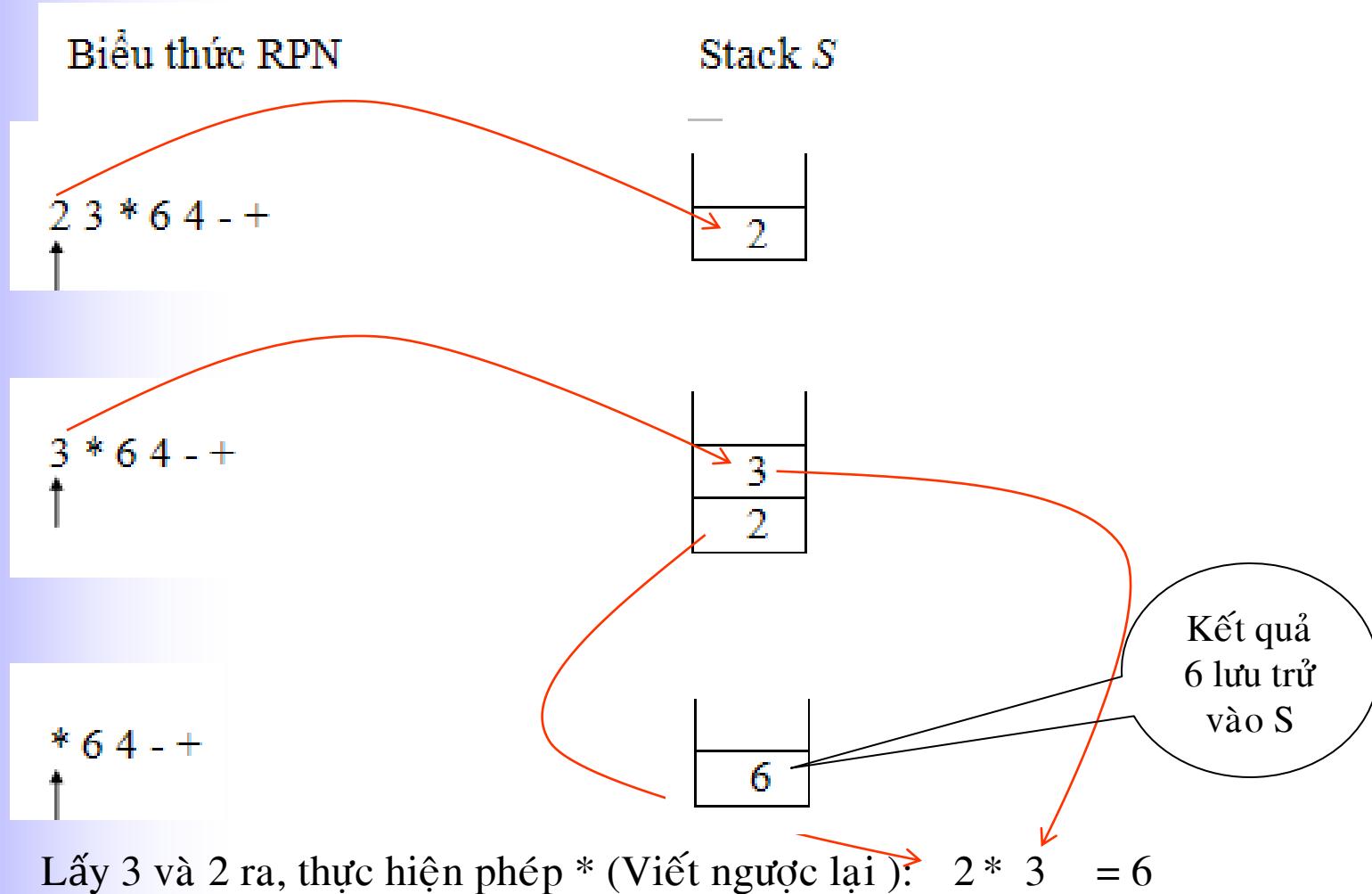
    }//da het sin
while (s.pHead != NULL) //lay cac toan tu trong s chen vao cuoi sout
{
    x = Pop(s);
    Chen_Cuoi_Chuoi(sout, x);
}
}
```

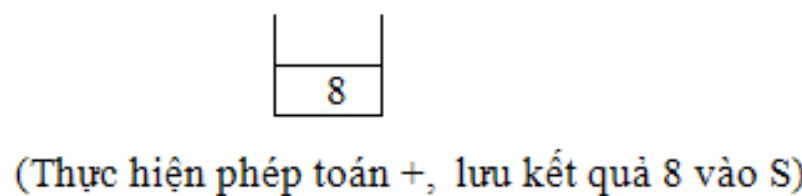
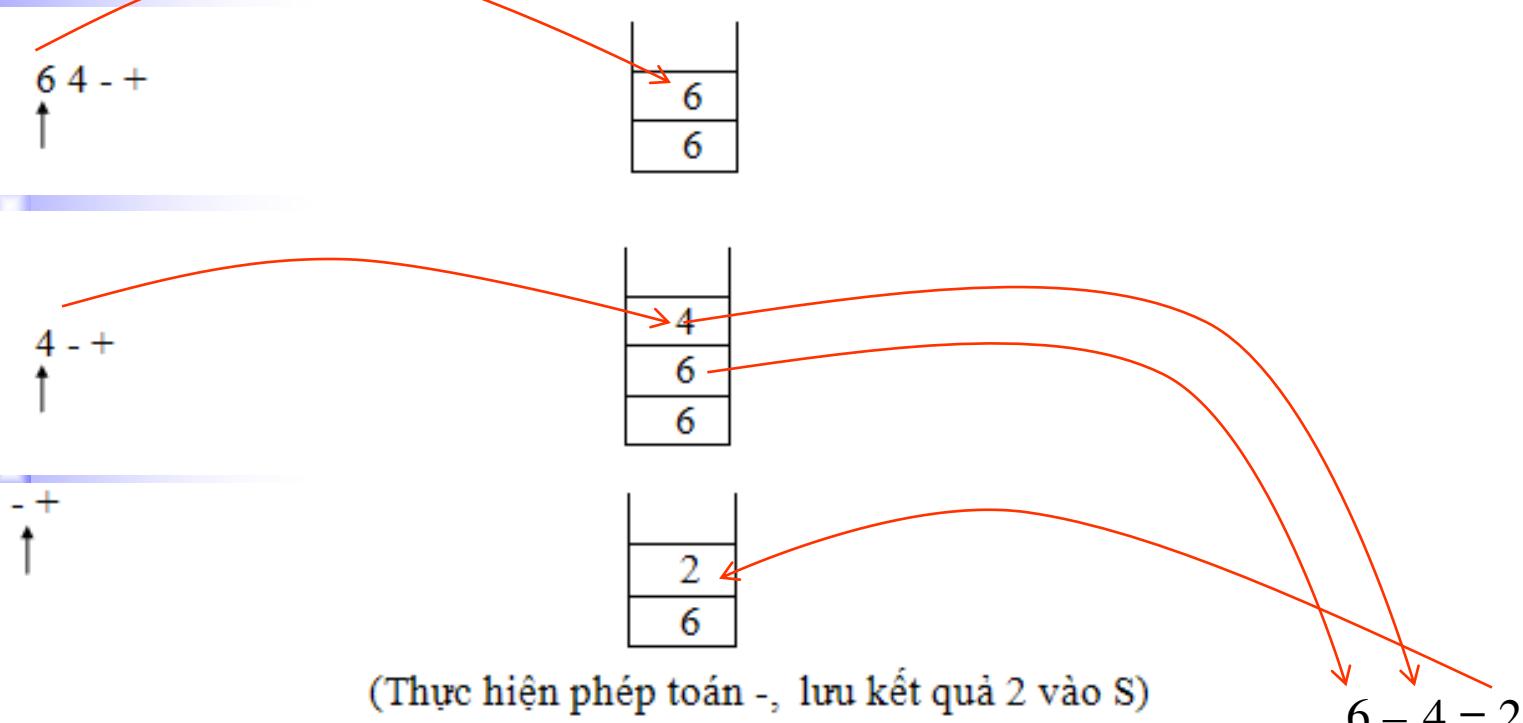
Thuật toán đánh giá biểu thức hậu tố RPN

- Input: Biểu thức hậu tố
- Output: Giá trị của biểu thức trung tố tương ứng
- Mô tả thuật toán:
 1. Khởi tạo ngăn xếp S rỗng;
 2. Lặp lại các việc sau cho đến khi dấu kết thúc biểu thức được đọc:
 - Đọc phần tử (toán hạng, toán tử) tiếp theo trong biểu thức;
 - Nếu phần tử là toán hạng: đẩy nó vào S ;
 - Ngược lại: // phần tử là toán tử
 - ❖Lấy từ đỉnh S hai toán hạng;
 - ❖Áp dụng toán tử đó vào 2 toán hạng (theo thứ tự ngược);
 - ❖Đẩy kết quả vừa tính trở lại S ;
 - 3. Khi gặp dấu kết thúc biểu thức (NULL), giá trị của biểu thức chính là giá trị ở đỉnh S ;

Minh họa:

- Input: $2 \ 3 \ * \ 6 \ 4 \ - \ +$ //Biểu thức hậu tố RPN
- Output: Giá trị biểu thức trung tố $2*3+(6-4)$





Kết thúc: Giá trị biểu thức= 8

```

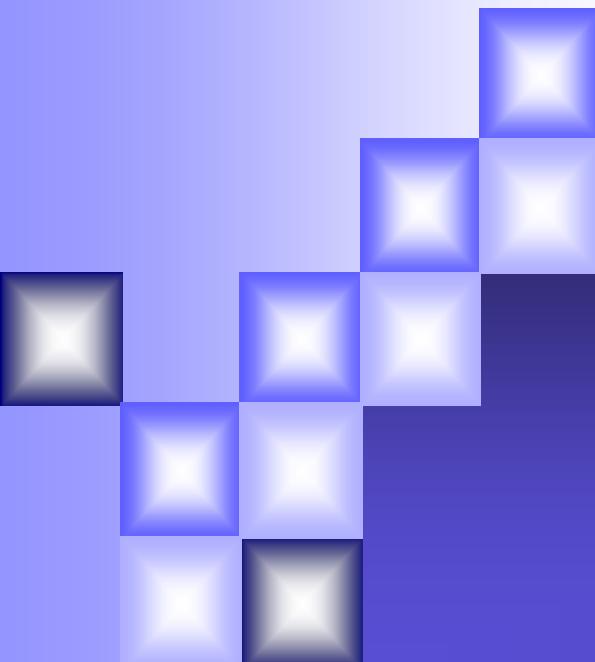
int Tinh_BT_HauTo(char a[MAX])
{
    int i;
    char c;
    data x,y;
    STACK s;
    CreatStack(s);
    for (i = 0; a[i] != NULL; i++)
    {
        c = a[i];
        if (LaKySo(c) == 1)
        {
            x = So(c);
            Push(s, x);
        }
        else //toan tu
        {
            x = Pop(s);
            y = Pop(s);
            switch (c)
            {

```

```

                case '+':
                    Push(s,y + x);
                    break;
                case '-':
                    Push(s, y-x);
                    break;
                case '*':
                    Push(s, y*x);
                    break;
                case '/':
                    Push(s, y/x);
                    break;
                case '%':
                    Push(s, y % x);
                    break;
            }
        }
        return s.pHead->info;
    }

```



Hàng đợi (Queue)

Hàng đợi (Queue)

- Hàng đợi là một vật chứa (container) các đối tượng:
 - ❖ Làm việc theo cơ chế FIFO (*First In First Out*)
 - ❖ ⇒ Thêm một đối tượng vào hàng đợi và lấy một đối tượng ra khỏi hàng đợi thực hiện theo cơ chế “*Vào trước ra trước*” :
 - ✓ Thêm một phần tử vào hàng đợi: luôn diễn ra ở cuối hàng đợi
 - ✓ Lấy một phần tử ra khỏi hàng đợi: luôn diễn ra ở đầu hàng đợi.
 - ❖ có nhiều ứng dụng:
 - ✓ khử đệ qui
 - ✓ tổ chức lưu vết các quá trình tìm kiếm theo chiều rộng và quay lui.
 - ✓ tổ chức quản lý và phân phối tiến trình trong các hệ điều hành, tổ chức bộ đệm bàn phím, ...

- Hàng đợi có các thao tác chính:
 - **EnQueue(o)**: Thêm đối tượng o vào cuối hàng đợi
 - **DeQueue()**: Lấy đối tượng ở đầu queue ra khỏi hàng đợi và trả về giá trị của nó. Nếu hàng đợi rỗng thì lỗi sẽ xảy ra.
- Hàng đợi còn hỗ trợ các thao tác:
 - **IsEmpty()**: Kiểm tra xem hàng đợi có rỗng không.
 - **Front()**: Trả về giá trị của phần tử nằm ở đầu hàng đợi mà không hủy nó. Nếu hàng đợi rỗng thì lỗi sẽ xảy ra.

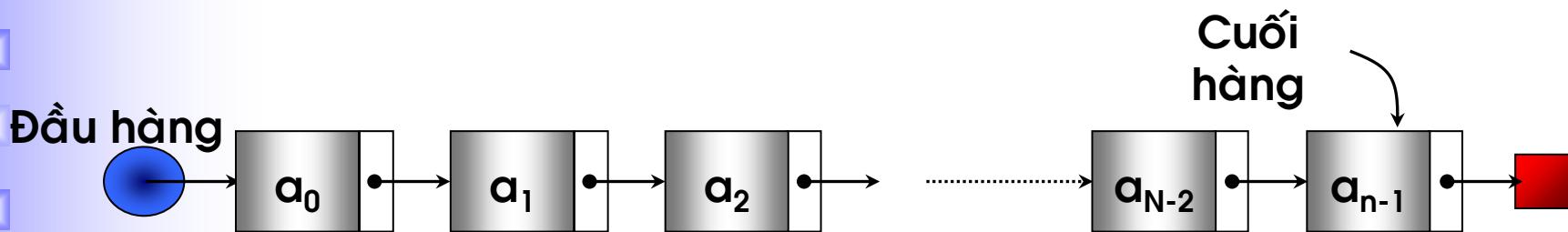
Biểu diễn hàng đợi (queue):

- dùng mảng
- dùng danh sách liên kết đơn

**Biểu diễn hàng đợi
bằng danh sách liên kết đơn**

Biểu diễn hàng đợi dùng danh sách liên kết

- Có thể tạo một hàng đợi sử dụng một DSKL đơn.
- Phần tử đầu DSKL (head) sẽ là phần tử đầu hàng đợi, phần tử cuối DSKL (tail) sẽ là phần tử cuối hàng đợi.



Kiểu QUEUE cài đặt bằng danh sách liên kết đơn

- Giả sử đã có các định nghĩa:

```
typedef int data;  
struct tagNode  
{  
    data           info;  
    tagNode*     pNext;  
};  
// kiểu của một phần tử trong danh sách  
typedef tagNode NODE;
```

- Cài đặt kiểu QUEUE (Danh sách liên kết , STACK) :

```
struct QUEUE  
{  
    NODE* pHead;  
    NODE* pTail;  
};
```

- Tạo hàng đợi rỗng:

```
void CreatQ(QUEUE &q)
{
    q.pHead=q.pTail= NULL;
}
```

- Kiểm tra hàng đợi rỗng :

```
int IsEmpty(QUEUE q)
{
    if (q.pHead == NULL) // hàng đợi rỗng
        return 1;
    return 0;
}
```

- Ghi chú :

Câu lệnh trực tiếp kiểm tra hàng đợi khác rỗng :

```
if (q.pHead != NULL)
```

//Tạo nút có dữ liệu là x, chèn nút này vào cuối hàng đợi.

```
void EnQueue(QUEUE &q, data x) //InserTai(q,x)
```

```
{
```

```
    NODE* new_ele = GetNode(x);
```

```
    if (new_ele ==NULL)
```

```
        return ; //exit(1);
```

```
    if (q.pHead==NULL)
```

```
{
```

```
        q.pHead = new_ele;
```

```
        q.pTail = q.pHead;
```

```
}
```

```
else
```

```
{
```

```
    q.pTail->Next = new_ele;
```

```
    q.pTail = new_ele;
```

```
}
```

```
}
```

```
//Hủy nút đầu hàng đợi
data DeQueue(QUEUE &q) //RemoveHead(q)
{
    NODE *p;
    data x;
    if (q.pHead == NULL)
        return NULLDATA;
    p = q.pHead;
    x = p->info;
    q.pHead = q.pHead->pNext;
    delete p;
    if(q.pHead==NULL)
        q.pTail = NULL;
    return x;
}
```

- Trả về phần tử ở đầu hàng đợi

```
data Front(QUEUE q)
{
    if (q.pHead == NULL)
        return NULLDATA;
    return
        q.pHead->info;
}
```



Nhận xét:

- Các thao tác trên hàng đợi biểu diễn bằng danh sách liên kết làm việc với chi phí $O(1)$.
- Nếu không quản lý phần tử cuối DSLK, thao tác **Dequeue** sẽ có độ phức tạp $O(n)$.