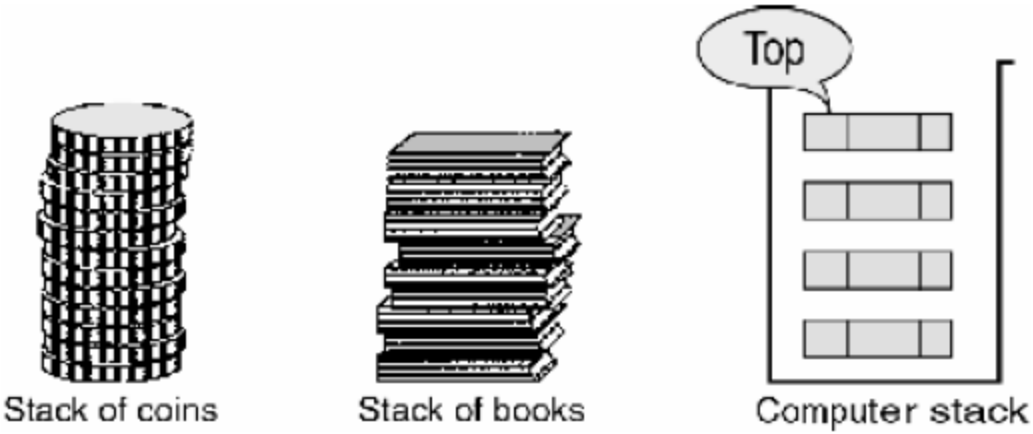
Ngăn xếp (stack)

Hàng đợi (queue)

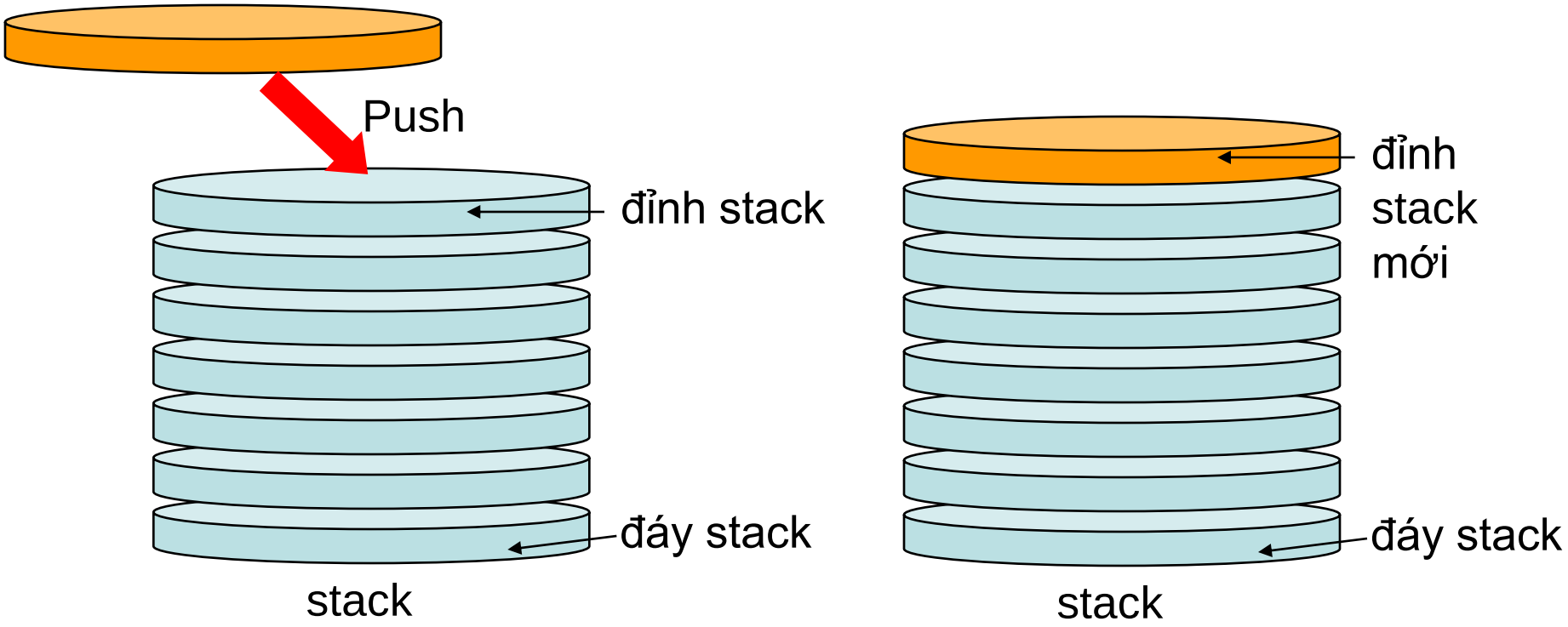
Stack {1}

- Là một vật chứa (container) các đối tượng làm việc theo cơ chế LIFO (Last In First Out) nghĩa là việc thêm một đối tượng vào stack hoặc lấy một đối tượng ra khỏi stack được thực hiện theo cơ chế “Vào sau ra trước”.
- Các đối tượng có thể được thêm vào stack bất kỳ lúc nào nhưng chỉ có đối tượng thêm vào sau cùng mới được phép lấy ra khỏi stack.



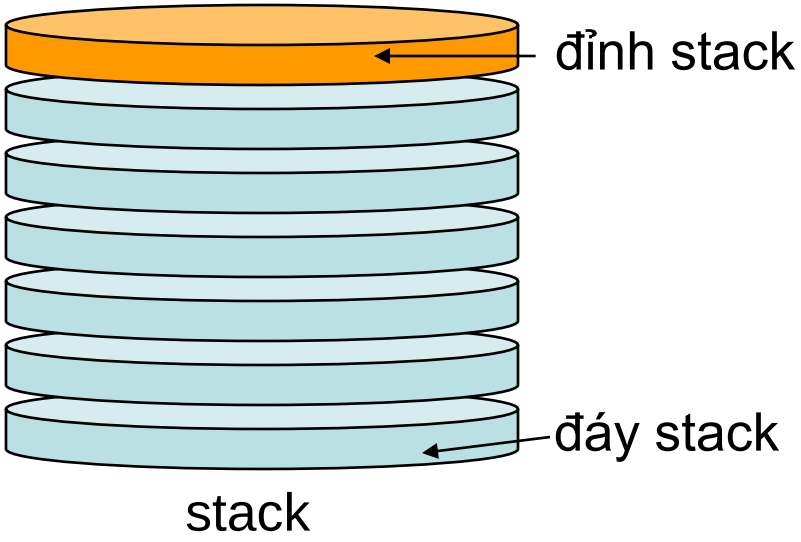
Stack {2}

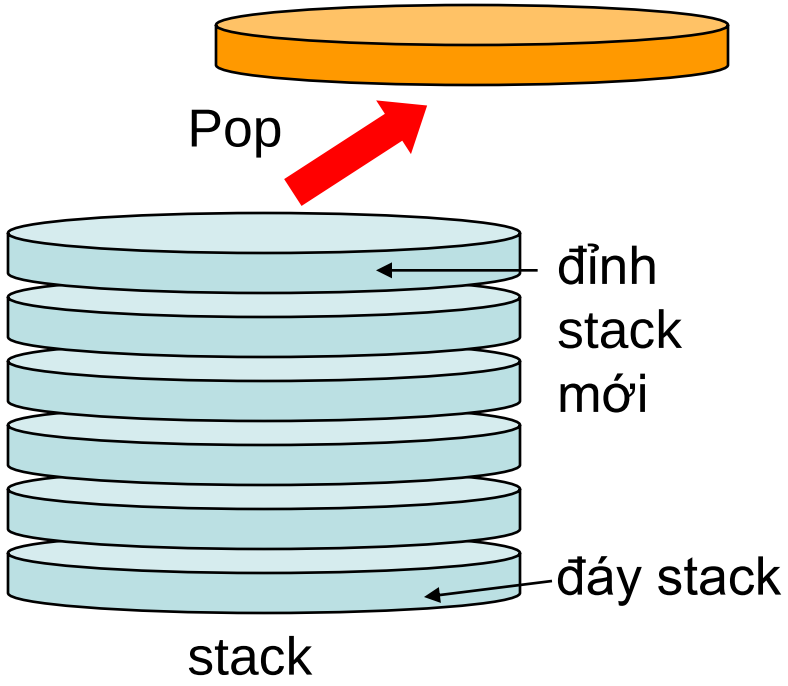
- Thao tác thêm một đối tượng vào stack gọi là “Push”



Stack {3}

- Thao tác lấy một đối tượng ra khỏi stack gọi là “Pop”





Stack {4}

- Stack có nhiều ứng dụng:
  - Khử đệ qui
  - Tổ chức lưu vết các quá trình tìm kiếm theo chiều sâu và quay lui, vét cạn.
  - Ứng dụng trong các bài toán tính toán biểu thức
  - Trình biên dịch
  - …

Stack {5}

- Các thao tác:
  - InitStack: khởi tạo stack rỗng.
  - IsEmptyStack: kiểm tra stack rỗng không.
  - IsFullStack : kiểm tra stack đầy không.
  - Push: thêm 1 phần tử vào đỉnh stack, có thể làm stack đầy.
  - Pop: lấy ra 1 phần tử từ đỉnh stack, có thể làm stack rỗng.
  - Top: kiểm tra phần tử đầu stack.
- Có 2 cách để xây dựng stack:
  - Sử dụng mảng 1 chiều.
  - Sử dụng DSLK đơn.

Cài đặt stack bằng mảng

```
#define MAXSTACKSIZE  100
typedef struct tagSTACK
{
        int  Top;
        DATA Array[MAXSTACKSIZE];
}STACK;
```

```
void InitStack(STACK &S)
{
        S.Top=-1;
}
int IsEmptyStack(STACK S)
{
        return (S.Top==-1);
}
int IsFullStack(STACK S)
{
        return (S.Top==MAXSTACKSIZE-1);
}
```

```
int Push(STACK &S,DATA X)
{
        if (IsFullStack(S))
                return 0;
        (S.Top)++;
        S.Array[S.Top]=X;
        return 1;
}
int Pop(STACK &S,DATA &X)
{
        if (IsEmptyStack (S))
                return 0;
        X=S.Array[S.Top];
        (S.Top)--;
        return 1;
}
int Top(STACK S,DATA &X)
{
        if (IsEmptyStack (S))
                return 0;
        X=S.Array[S.Top];
        return 1;
}
```

Ví dụ

- Chuyển số nguyên sang hệ 2

```
typedef struct tagDATA
{
        int Key;
}DATA;
```

```
void Convert(int N)
{
        STACK S;
        DATA X;
        InitStack(S);
        while (N>0)
        {
                X.Key=N%2;
                Push(S,X);
                N/=2;
        }
        while (Pop(S,X))
                printf("%d",X.Key);
}
```

Cài đặt stack bằng DSLK

```
void InitStack(LIST &S)
{
	InitList(S);
}
int IsEmptyStack(LIST S)
{
	return IsEmptyList(S);
}
int Push(LIST &S,DATA X)
{
	return (InsertHead(S,X)==NULL);
}
int Pop(LIST &S,DATA &X)
{
	PNODE P=RemoveHead(S);
	if (P==NULL)
		return 0;
	X=P->Info;
	delete P;
	return 1;
}
int Top(LIST S,DATA &X)
{
	if (S.pHead==NULL)
		return 0;
	X=S.pHead->Info;
	return 1;
}
```

Ứng dụng {1}

- Kiểm tra tính đúng đắn về cách sử dụng dấu ngoặc của biểu thức.

- Ví dụ:
  - ((a+b)*c+(a-b)*(a-c)) → đúng
  - (a+b*(c-d)-(5-8)*2*(4-8) → sai

- Thuật toán

  (1) Khởi tạo stack

  (2) Đọc lần lượt các ký tự trong biểu thức dấu ngoặc

  (a) Nếu ký tự là dấu ngoặc mở thì đẩy nó vào stack

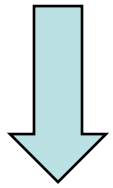
  (b) Nếu ký tự là dấu ngoặc đóng thì
  - Nếu stack rỗng thì thông báo biểu thức dấu ngoặc sai và dừng.
  - Nếu stack không rỗng thì loại dấu ngoặc mở ở đỉnh stack.

  (3) Sau khi ký tự cuối cùng trong biểu thức dấu ngoặc đã được đọc, nếu stack rỗng thì thông báo biểu thức dấu ngoặc đúng.

Ứng dụng {2}

Biểu thức	Ký tự	Stack	Thao tác
((a+b)*c+(a-b)*(a-c))	(	(	Push ‘(‘
(a+b)*c+(a-b)*(a-c))	(	((	Push ‘(‘
a+b)*c+(a-b)*(a-c))	a	((	
+b)*c+(a-b)*(a-c))	+	((	
b)*c+(a-b)*(a-c))	b	((	
)*c+(a-b)*(a-c))	)	(	Pop ‘(‘
c+(a-b)(a-c))	*	(	
c+(a-b)*(a-c))	c	(	
+(a-b)*(a-c))	+	(	
(a-b)*(a-c))	(	((	Push ‘(‘
a-b)*(a-c))	a	((	
-b)*(a-c))	-	((	
b)*(a-c))	b	((	

Biểu thức	Ký tự	Stack	Thao tác
)*(a-c))	)	(	Pop ‘(‘
*(a-c))	*	(	
(a-c))	(	((	Push ‘(‘
a-c))	a	((	
-c))	-	((	
c))	c	((	
))	)	(	Pop ‘(‘
)	)		Pop ‘(‘



Biểu thức đúng

Ứng dụng {3}

- Tính giá trị của biểu thức số học
- Các dạng của biểu thức:
  - Tiền tố (prefix): toán tử đặt trước hai toán hạng

    \+ 3 5
  - Trung tố (infix): toán tử đặt giữa hai toán hạng

    3 + 5
  - Hậu tố (postfix - ký pháp Balan): toán tử đặt sau hai toán hạng

    3 5 +
- Có thể chuyển biểu thức trung tố sang hậu tố.

  (3 + 5) * 2 → 3 5 + 2 *

  → Biểu thức hậu tố không có các dấu ngoặc

  → Việc tính các biểu thức hậu tố là rất dễ dàng

  → Các phép toán được thực hiện theo thứ tự mà chúng xuất hiện trong biểu thức kể từ trái sang phải
- Việc tính biểu thức trung tố được thực hiện qua hai giai đoạn:
  - Chuyển biểu thức trung tố thành biểu thức hậu tố.
  - Đánh giá biểu thức hậu tố.

Ứng dụng {4}

- Thuật toán chuyển biểu thức từ trung tố sang hậu tố

Input: infix

Output: postfix

(1) Khởi động: postfix rỗng, stack kiểu ký tự rỗng.

(2) Đọc 1 toán hạng hay toán tử từ infix vào token.

(3) Nếu token là toán hạng thì đưa nó vào postfix.

(4) Nếu token là toán tử (hiện thời) thì thực hiện các bước sau:

  (a) Nếu stack không rỗng thì nếu phần tử ở đỉnh stack là phép toán có độ ưu tiên cao hơn hay bằng toán tử hiện thời, thì toán tử đó được lấy khỏi stack và đưa vào postfix. Lặp lại bước này.

  (b) Nếu stack rỗng hoặc phần tử ở đỉnh stack là ‘(‘ hoặc toán tử ở đỉnh stack có độ ưu tiên thấp hơn toán tử hiện thời, thì toán tử hiện thời được đẩy vào stack.

(5) Nếu token là ‘(‘ thì đưa token vào stack.

(6) Nếu token là ‘)’, lấy nội dung ở đỉnh stack đưa vào postfix cho đến khi lấy được ‘(‘.

(7) Nếu infix còn dữ liệu thì quay lại (2), ngược lại lấy toàn bộ nội dung stack ra và đưa vào postfix .

Ứng dụng {5}

- Độ ưu tiên của các toán tử trong biểu thức trung tố:

Toán tử	Độ ưu tiên
+	0
-	0
*	1
/	1

Ứng dụng {6}

- Chuyển biểu thức (9 + 7 * 2) / 2 – 5 sang dạng postfix

Infix	Token	Stack	Postfix
(9+7*2)/2-5	(	(	
9+7*2)/2-5	9	(	9
+7*2)/2-5	+	(+	9
7*2)/2-5	7	(+	9 7
*2)/2-5	*	(+ *	9 7
2)/2-5	2	(+ *	9 7 2
)/2-5	)		9 7 2 * +
/2-5	/	/	9 7 2 * +
2-5	2	/	9 7 2 * + 2
-5	-	-	9 7 2 * + 2 /
5	5	-	9 7 2 * + 2 / 5
			9 7 2 * + 2 / 5 -

(9 + 7 * 2) / 2 – 5 → 9 7 2 * + 2 / 5 -

Ứng dụng {7}

- Thuật toán tính giá trị biểu thức hậu tố

Input: postfix

Output: kết quả của biểu thức

(1) Khởi động stack kiểu số rỗng.

(2) Đọc 1 toán hạng hay toán tử từ postfix vào token.

(3) Nếu token là toán hạng thì đưa nó vào stack.

(4) Nếu token là toán tử thì thực hiện các bước sau:

  (a) Lấy một phần tử của stack đưa vào b

  (b) Lấy một phần tử của stack đưa vào a

  (c) Thực hiện toán tử này trên hai toán hạng a và b.

  (d) Đưa kết quả vào stack

(5) Nếu posfix còn dữ liệu thì quay lại (2), ngược lại giá trị còn lại ở stack chính là giá trị của biểu thức postfix.

Ứng dụng {8}

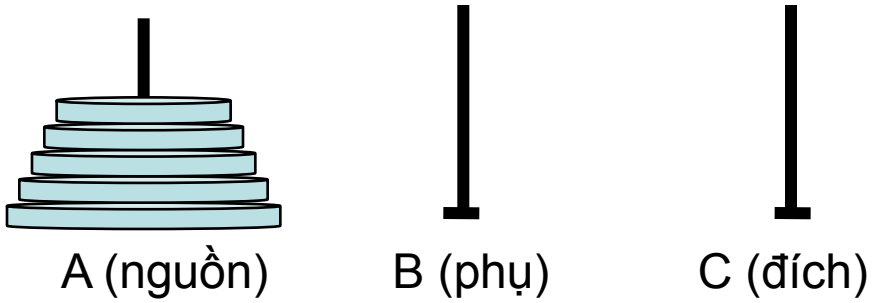
- Tính giá trị của biểu thức 9 7 2 * + 2 / 5 -

Postfix	Token	Stack	Ghi chú
9 7 2 * + 2 / 5 -	9	9	
7 2 * + 2 / 5 -	7	9 7	
2 * + 2 / 5 -	2	9 7 2	
* + 2 / 5 -	*	9 14	7 * 2 = 14
+ 2 / 5 -	+	23	9 + 14 = 23
2 / 5 -	2	23 2	
/ 5 -	/	11.5	23 / 2 = 11.5
5 -	5	11.5 5	
-	-	6.5	11.5 – 5 = 6.5

→ 9 7 2 * + 2 / 5 - = 6.5

Ứng dụng {9}

- Khử đệ quy
  - Một trong các ứng dụng quan trọng của stack là sử dụng stack để chuyển thuật toán đệ quy thành thuật toán không đệ quy.
- Bài toán tháp Hanoi




```
void HanoiTower(int N, char A, char B, char C)
{
        if (N==1)
                printf("Chuyen 1 dia tu %c sang %c\n", A, C);
        else
        {
                HanoiTower(N-1, A, C, B);
                HanoiTower(1, A, B, C);
                HanoiTower(N-1, B, A, C);
        }
}
```

Ứng dụng {10}

```
struct DATA
{
        int N;
        char A,B,C;
        DATA()
        {
        }
        DATA(int N2,char A2,
                char B2,char C2)
        {
                N=N2;
                A=A2;
                B=B2;
                C=C2;
        }
};
```

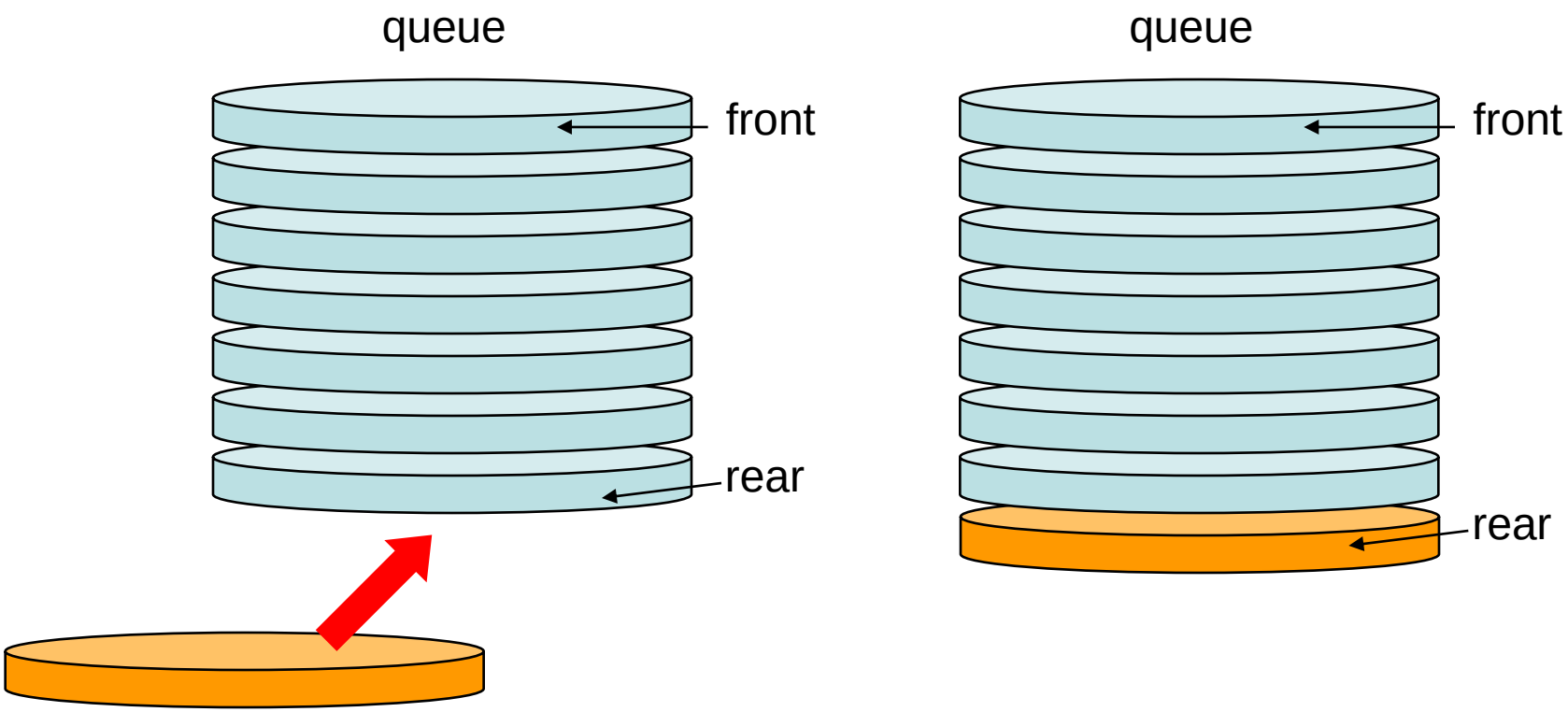
```
void HanoiTower(int N, char A, char B, char C)
{
        STACK S;
        InitStack(S);
        DATA X;
        Push(S,DATA(N,A,B,C));
        while (Pop(S,X))
        {
                if (X.N==1)
                        printf("Chuyen 1 dia tu %c sang %c\n",
                                X.A,X.C);
                else
                {
                        Push(S,DATA(X.N-1,X.B,X.A,X.C));
                        Push(S,DATA(1,X.A,X.B,X.C));
                        Push(S,DATA(X.N-1,X.A,X.C,X.B));
                }
        }
}
```

Queue {1}

- Là một vật chứa (container) các đối tượng làm việc theo cơ chế FIFO (First In First Out) nghĩa là việc thêm một đối tượng vào stack hoặc lấy một đối tượng ra khỏi queue được thực hiện theo cơ chế “Vào trước ra trước”.
- Thao tác thêm một đối tượng vào hàng đợi và lấy một đối tượng ra khỏi hàng đợi lần lượt được gọi là “enqueue” và “dequeue”.
- Việc thêm một đối tượng vào hàng đợi luôn diễn ra ở cuối hàng đợi (rear) và một phần tử luôn được lấy ra từ đầu hàng đợi (front).

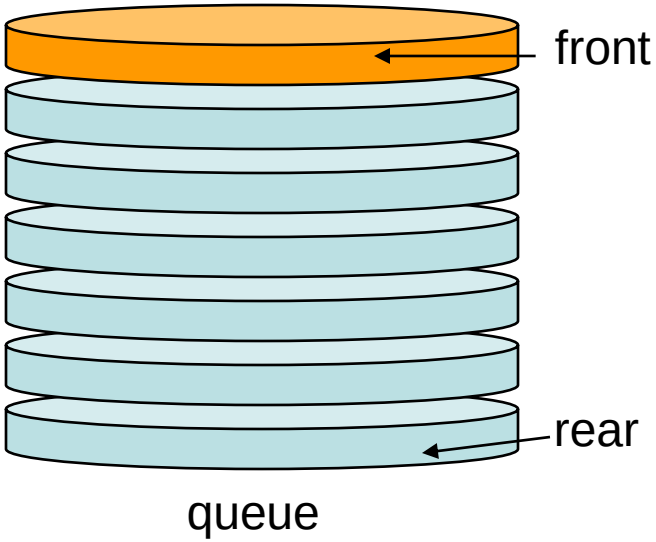
Queue {2}

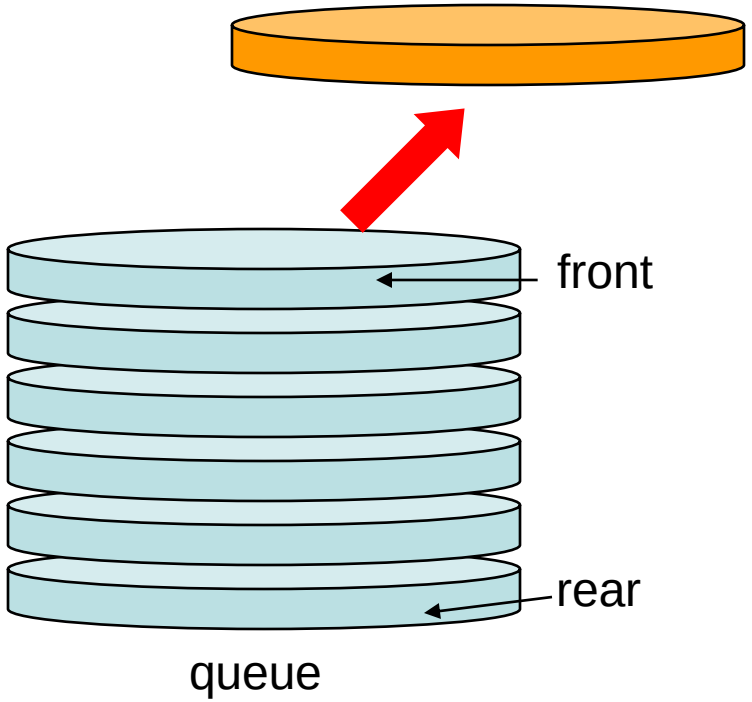
- Thao tác “enqueue”



Queue {3}

- Thao tác “dequeue”





Queue {4}

- Ứng dụng:
  - Bài toán ‘sản xuất và tiêu thụ’ (ứng dụng trong các hệ điều hành song song).
  - Bộ đệm (Nhấn phím → Bộ đệm → CPU xử lý).
  - Xử lý các lệnh trong máy tính (ứng dụng trong HĐH, trình biên dịch), hàng đợi các tiến trình chờ được xử lý, …
- Các thao tác:
  - InitQueue: khởi tạo queue rỗng.
  - IsEmptyQueue : kiểm tra queue rỗng không.
  - IsFullQueue : kiểm tra queue đầy không.
  - Enqueue: thêm 1 phần tử vào cuối queue , có thể làm queue đầy.
  - Dequeue: lấy ra 1 phần tử ở đầu queue , có thể làm queue rỗng.
  - Front: kiểm tra phần tử ở đầu queue.
- Có 2 cách để xây dựng queue:
  - Sử dụng mảng 1 chiều.
  - Sử dụng DSLK đơn.

Cài đặt queue bằng mảng {1}

- Khi thêm vào ở rear và lấy ra ở front sẽ làm cho phần sử dụng của queue có khuynh hướng di chuyển về phía dưới.
  - ⇒ Đến lúc nào đó sẽ không thêm được phần tử mới do số phần tử tối đa là cố định (gọi là MAXQUEUESIZE).
  - ⇒ Queue bị tràn (còn chỗ trống nhưng không thêm được)
  - Queue đầy khi: front=0 và rear=MAXQUEUESIZE-1
  - Queue bị tràn khi: front≠0 và rear=MAXQUEUESIZE-1
- Khắc phục hiện tường bị tràn:
  - Di chuyển tịnh tiến: di chuyển toàn bộ phần tử lên front vị trí.
    - front<rear
    - Số phần tử hiện tại: (rear-front+1)
  - Di chuyển vòng: queue đầy khi:
    - rear-front+1=0
    - rear-front+1=MAXQUEUESIZE

Cài đặt queue bằng mảng {2}

```
typedef struct tagQUEUE
{
        int    Front,Rear;
        DATA Array[MAXQUEUESIZE];
}QUEUE;
```

```
void InitQueue(QUEUE & Q)
{
        Q.Front=Q.Rear=-1;
}
int IsEmptyQueue (QUEUE Q)
{
        return (Q.Front==-1);
}
//Phuong phap di chuyen tinh tien
int IsFullQueue1(QUEUE Q)
{
        return (Q.Rear-Q.Front+1)==MAXQUEUESIZE;
}
//Phuong phap di chuyen vong
int IsFullQueue2(QUEUE Q)
{
        return  (Q.Rear-Q.Front+1==MAXQUEUESIZE)||(Q.Rear-Q.Front+1==0);
}
```

Cài đặt queue bằng mảng {3}

```
//Them mot phan tu vao hang,dung phuong phap di chuyen tinh tien
int EnQueue1(QUEUE &Q,DATA X)
{
	if (!IsFullQueue1(Q))
		return 0;
	if (Q.Front==-1) //Hang rong
		Q.Front=0;
	if (Q.Rear==MAXQUEUESIZE-1) //Hang bi tran,doi hang len Front vi tri
	{
		for(int I=Q.Front;I<=Q.Rear;++I)
			Q.Array[I-Q.Front]=Q.Array[I];
		Q.Rear=MAXQUEUESIZE-1-Q.Front;
		Q.Front=0;
	}
	Q.Rear++;
	Q.Array[Q.Rear]=X;
	return 1;
}
```

Cài đặt queue bằng mảng {4}

```
//Them mot phan tu vao hang,dung phuong phap di chuyen vong
int EnQueue2(QUEUE &Q,DATA X)
{
	if (IsFullQueue2(Q))
		return 0;
	if (Q.Front==-1) //Hang rong
		Q.Front=0;
	if (Q.Rear==MAXQUEUESIZE-1)
		Q.Rear=-1;
	Q.Rear++;
	Q.Array[Q.Rear]=X;
	return 1;
}
```

Cài đặt queue bằng mảng {5}

```
//Phuong phap di chuyen tinh tien
int DeQueue1(QUEUE &Q,DATA &X)
{
	if (IsEmptyQueue(Q))
		return FALSE;
	X=Q.Array[Q.Front];
	Q.Front++;
	if (Q.Front>Q.Rear)
		Q.Front=Q.Rear=-1;
	return TRUE;
}
//Phuong phap di chuyen vong
int DeQueue2(QUEUE &Q,DATA &X)
{
	if (IsEmptyQueue(Q))
		return FALSE;
	X=Q.Array[Q.Front];
	if (Q.Front==Q.Rear)
		Q.Front=Q.Rear=-1;
	else
	{
		Q.Front++;
		if (Q.Front==MAXQUEUESIZE)
			Q.Front=0;
	}
	return TRUE;
}
```

```
int Front(QUEUE Q,DATA &X)
{
	if (IsEmptyQueue(Q))
		return FALSE;
	X=Q.Array[Q.Front];
	return TRUE;
}
```

Cài đặt queue bằng DSLK

```
void InitQueue(LIST &Q)
{
	InitList(Q);
}
int IsEmptyQueue(LIST Q)
{
	return IsEmptyList(Q);
}
int EnQueue(LIST &Q,DATA X)
{
	return InsertTail(Q,X)==NULL;
}
int DeQueue(LIST &Q,DATA &X)
{
	PNODE P=RemoveHead(Q);
	if (P==NULL)
		return 0;
	X=P->Info;
	delete P;
	return 1;
}
int Front(LIST Q,DATA &X)
{
	if (Q.pHead==NULL)
		return 0;
	X=Q.pHead->Info;
	return 1;
}
```

Ví dụ {1}

- Đổi từ hệ 10 sang hệ 2 đối với phần lẻ

$0.6875_{10} = 0.1011_2$

```
typedef struct tagDATA
{
        int Key;
}DATA;
```

```
#define  MAX  10
void Convert(float  N)
{
        QUEUE Q;
        DATA X;
        InitQueue(Q);
        int C=0;
        while ((N!=0)&&(C<MAX))
        {
                N*=2;
                X.Key=(int)N;
                N-=X.Key;
                ++C;
                EnQueue(Q,X);
        }
        printf("0.");
        while (!IsEmptyQueue(Q))
        {
                DeQueue(Q,X);
                printf("%d",X.Key);
        }
}
```

Ví dụ {2}

- Kiểm tra một chuỗi là palindrome?
  - Một palindrome là một chuỗi đọc xuôi ngược đều giống như nhau.
  - Ví dụ:

    madam

    radar

    Able was I ere I saw Elba

```
typedef struct tagDATA
{
        char Key;
}DATA;
```

```
printf("Nhap vao mot chuoi:");
gets(St);
strupr(St);
LIST S,Q;
DATA X,Y;
InitStack(S);
InitQueue(Q);
for(int I=0;I<strlen(St);++I)
{
        X.Key=St[I];
        Push(S,X);
        EnQueue(Q,X);
}
```

```
int Palindrome=1;
while ((!IsEmptyStack(S)) &&(!IsEmptyQueue(Q)))
{
        Pop(S,X);
        DeQueue(Q,Y);
        if (X.Key!=Y.Key)
        {
                Palindrome=0;
                break;
        }
}
if (Palindrome)
        printf("Day la mot palindrome\n");
else
        printf("Day khong phai la mot palindrome\n");
```

Q & A