

**CONFIDENTIAL**

# **C Programming Basic – week 4,5**

*For Data Structure and Algorithms*

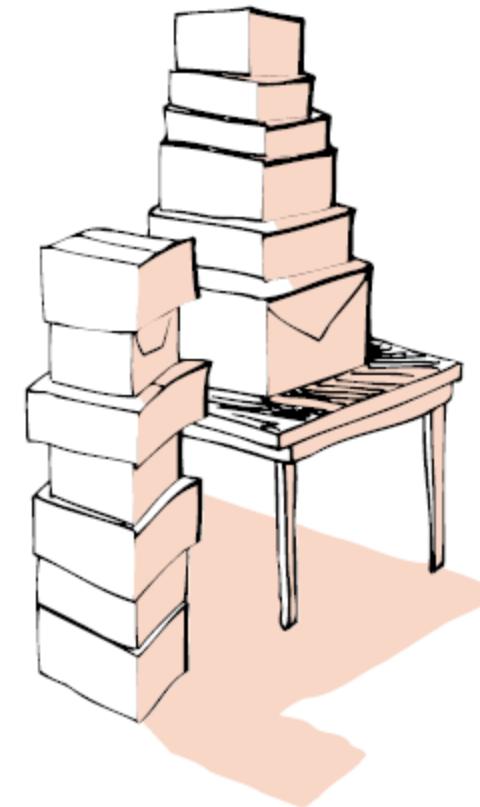
**Lecturer :**

**Do Quoc Huy**

**Dept of Computer Science  
Hanoi University of Technology**

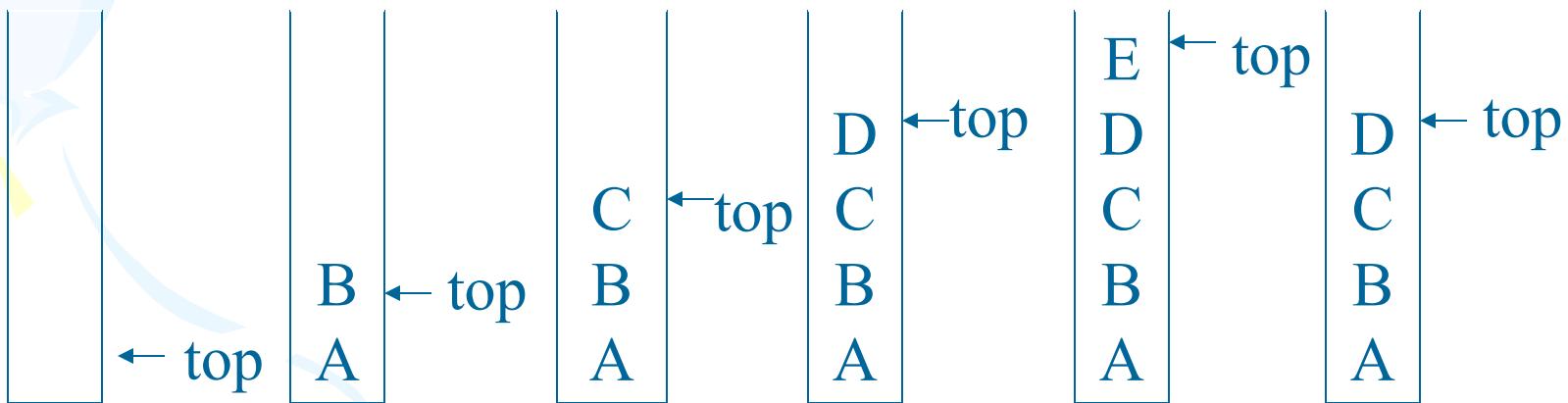
# Topics of this week

- Cấu trúc dữ liệu Stack (ngăn xếp).
  - Cài đặt ngăn xếp dùng mảng
  - Cài đặt ngăn xếp dùng danh sách liên kết
- Cấu trúc dữ liệu Queue (hàng đợi).
  - Cài đặt queue vòng sử dụng mảng
  - Cài đặt queue sử dụng danh sách liên kết
- Bài tập với Stack và Queue.



# Stack

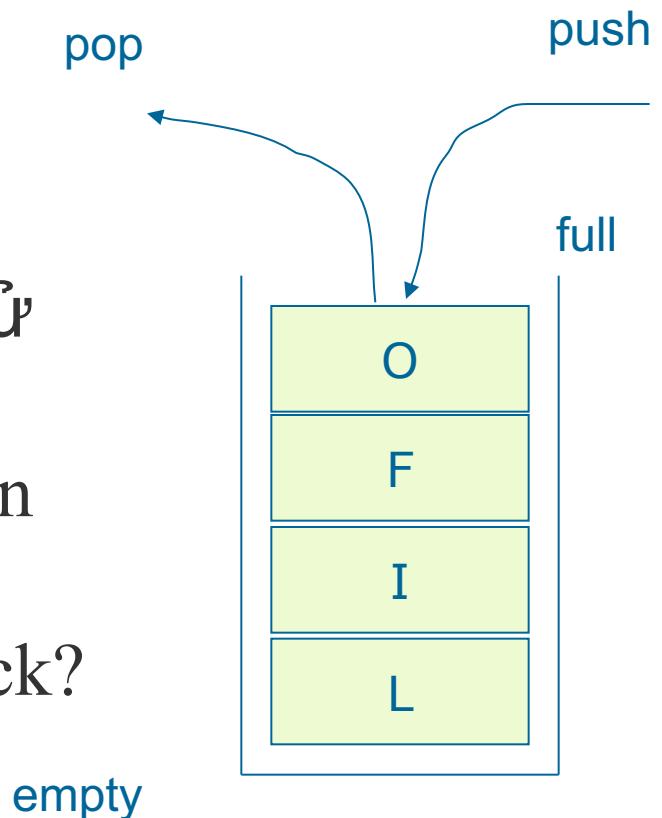
- Stack là 1 cấu trúc dữ liệu tuyến tính mà chỉ có thể truy nhập tới phần tử cuối cùng của nó để nhập và xuất dữ liệu.
- Cấu trúc LIFO(Last In First Out – vào sau ra trước)



Chèn và xóa một phần tử trong stack

# Các thao tác trên Stack

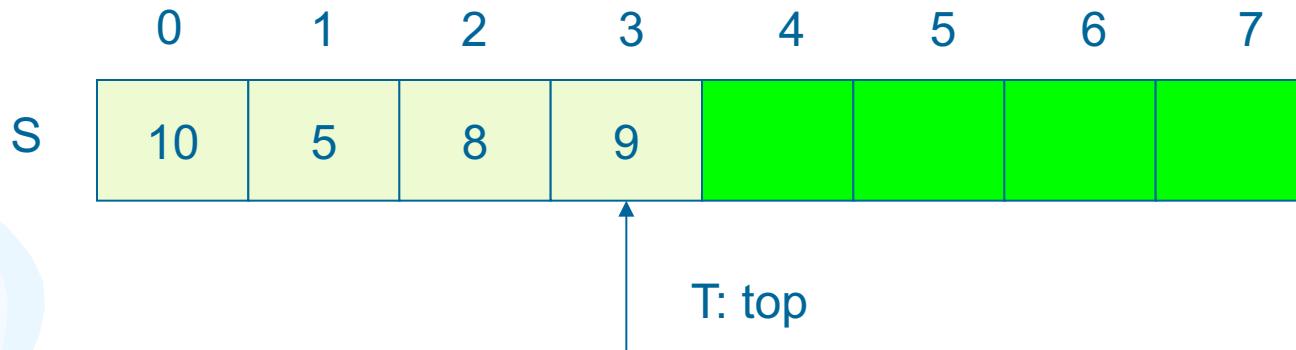
- *initialize(stack)* --- khởi tạo stack
- *empty(stack)* --- kiểm tra stack có rỗng không
- *full(stack)* --- kiểm tra stack có đầy không
- *push(el,stack)* --- thêm phần tử el vào đỉnh của stack
- *pop(stack)* --- lấy ra phần tử trên đỉnh stack
- Làm thế nào để triển khai 1 stack?



# Phân tách việc cài đặt từ đặc tả

- **INTERFACE** (giao diện): đặc tả các phép toán được cho phép
- **IMPLEMENTATION** (cài đặt/triển khai): cung cấp code cho các phép toán
- **CLIENT**: các code mà ta sử dụng.
- Có thể sử dụng **mảng** hoặc **linked list** để triển khai **stack**
- Client có thể làm việc ở mức trừu tượng cao

# Triển khai sử dụng mảng



- Mỗi phần tử của stack được lưu trữ như là 1 phần tử của mảng.
- Stack rỗng: `top=0; // top là chỉ số của phần tử đỉnh stack`
- Stack đầy: `top=Max_Element(chỉ số phần tử cuối của mảng)`.

# Đặc tả Stack (stack.h)

```
#define Max 50//số phần tử tối đa Top of stack
```

```
typedef int Eltype; //kiểu phần tử mảng  
// là int
```

```
typedef Eltype StackType[Max]; //định nghĩa  
kiểu StackType là mảng Max phần tử có kiểu  
Eltype.
```

```
int top;
```

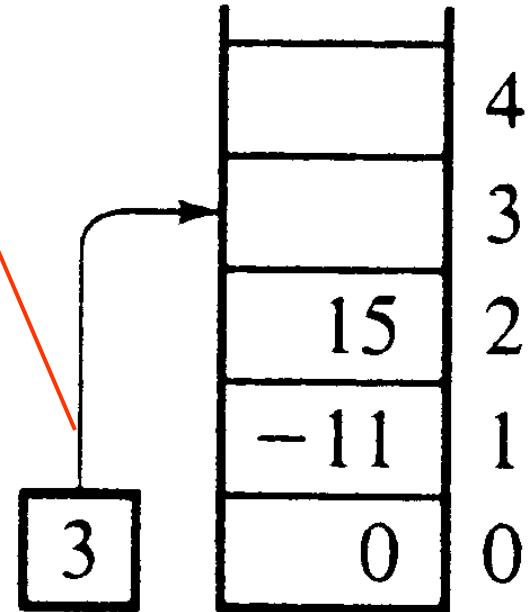
```
void Initialize(StackType stack);
```

```
int empty(StackType stack);
```

```
int full(StackType stack);
```

```
void push(Eltype el, StackType stack);
```

```
Eltype pop(StackType stack);
```



(a)

# Khai triển mảng của stack (stack.c)

```
Initialize(StackType stack)
{
    top = 0;
}

empty(StackType stack)
{
    return top == 0;
}

full(StackType stack)
{
    return top == Max;
}

push(Eltype el, StackType stack)
{
    if (full(*stack))
        printf("stack overflow");
    else stack[top++] = el;
}

Eltype pop(StackType stack)
{
    if (empty(stack))
        printf("stack underflow");
    else return stack[--top];
}
```

# Triển khai stack sử dụng cấu trúc

- Triển khai: stack được thể hiện là 1 cấu trúc có 2 trường: 1 để lưu trữ, 1 để theo dõi vị trí của phần tử trên cùng

```
#define Max 50
typedef int Eltype;
typedef struct StackRec {
    Eltype storage[Max];
    int top;
};
typedef struct StackRec StackType;
```

# Triển khai stack sử dụng cấu trúc

```
Initialize(StackType *stack)
{
    (*stack).top=0;
}
empty(StackType stack)
{
    return stack.top == 1;
}
full(StackType stack)
{
    return stack.top == Max;
}
```

```
push(Eltype el, StackType *stack)
{
    if (full(*stack))
        printf("stack tràn");
    else (*stack).storage[
        (*stack).top++]=el;
}
Eltype pop(StackType *stack)
{
    if (empty(*stack))
        printf("stack không còn để lấy ");
    else return
        (*stack).storage[--(*stack).top];
}
```

# Biên dịch file với thư viện

Bạn có **stack.h**, **stack.c** và **test.c**

Cần chèn thêm dòng sau:

**#include "stack.h"**

Vào file **stack.c** và **test.c**

**gcc -c stack.c**

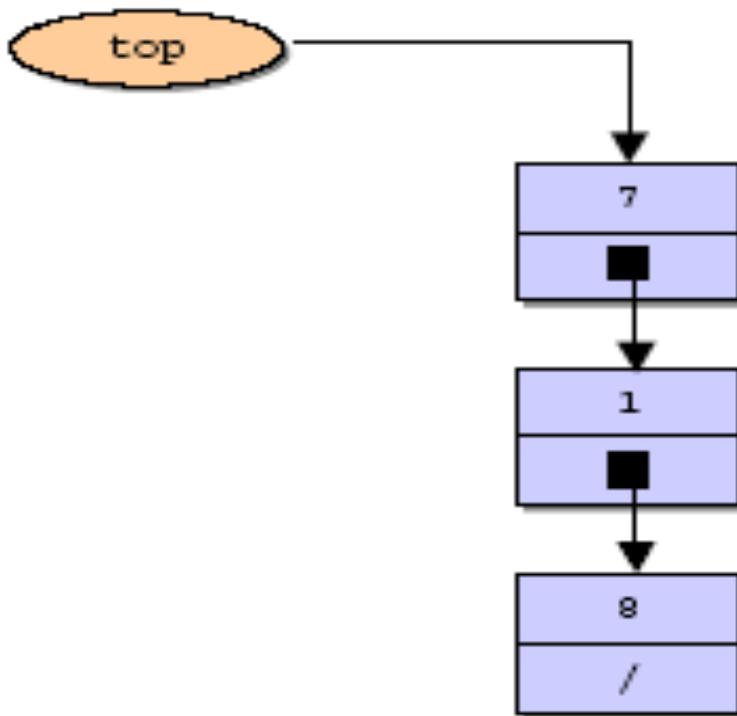
**gcc -c test.c**

**gcc -o test.out test.o stack.o**

# Triển khai sử dụng linked list

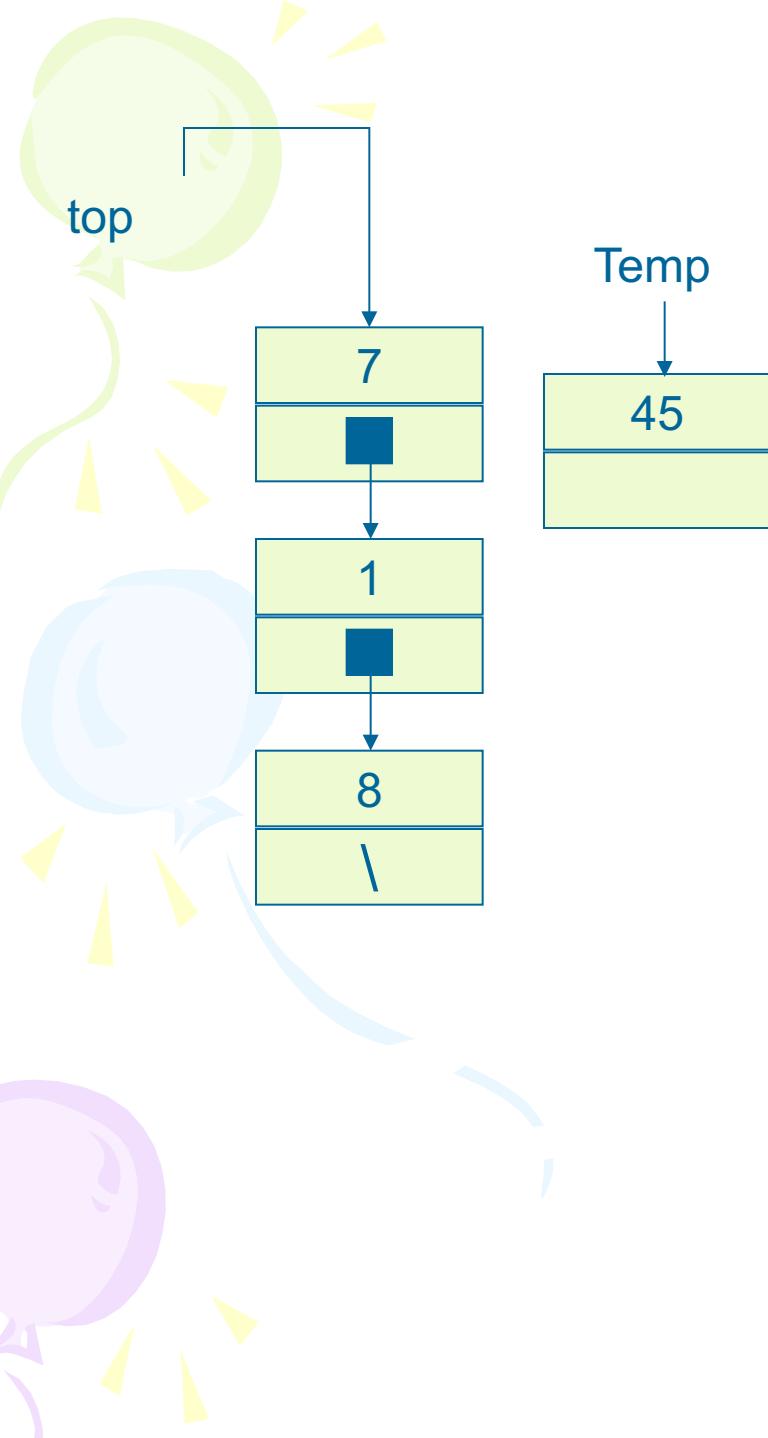
- Triển khai stack sử dụng linked list rất đơn giản.
- Sự khác nhau giữa 1 linked list bình thường và 1 stack sử dụng linked list là 1 vài thao tác của linked list **không** sử dụng được cho stack.
- Là stack thì ta chỉ có 1 thao tác chèn( insert) duy nhất là **push()**.
  - Trong nhiều trường hợp thì push giống như là phép chèn vào trước.
- Ta cũng chỉ có 1 thao tác xoá (delete) là **pop()**.
  - Thao tác này giống như là thao tác xoá từ phía trước.

# Diễn tả bằng hình ảnh của stack



```
struct node {  
    int data;  
    struct node *link;  
};
```

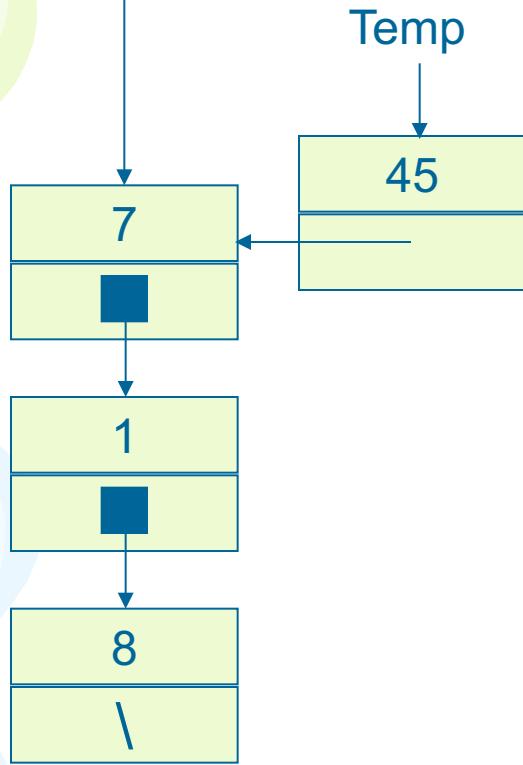
# Push



```
struct node *push(struct node *p, int value)
{
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct
node));
    if(temp==NULL) {
        printf("No Memory available Error\n");
        exit(0);
    }
    temp->data = value;
    temp->link = p;
    p = temp;
    return(p);
}
```

# Push

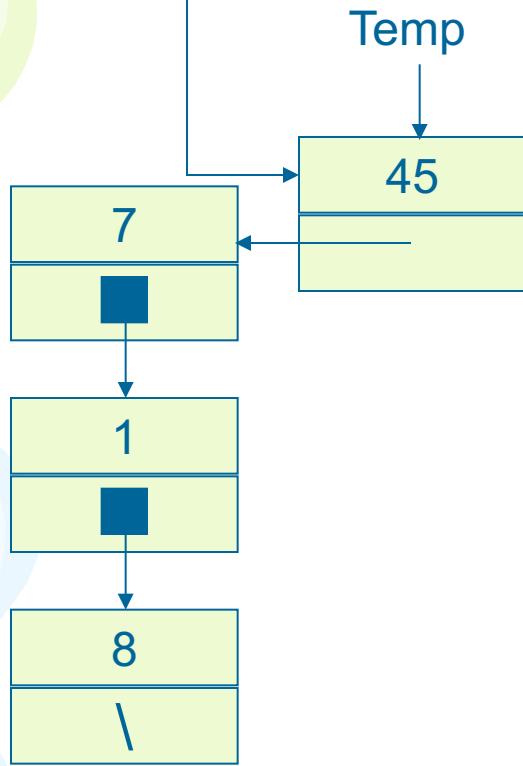
top



```
struct node *push(struct node *p, int value)
{
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct
node));
    if(temp==NULL) {
        printf("No Memory available Error\n");
        exit(0);
    }
    temp->data = value;
    temp->link = p;
    p = temp;
    return(p);
}
```

# Push

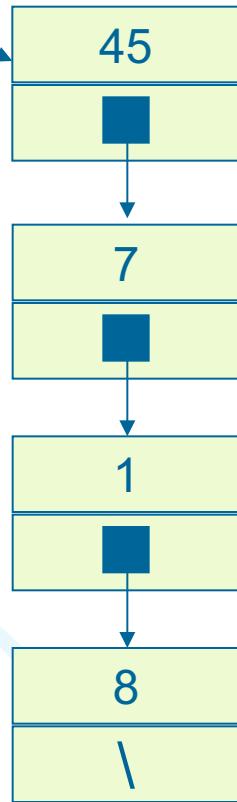
top



```
struct node *push(struct node *p, int value)
{
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node));
    if(temp==NULL) {
        printf("No Memory available Error\n");
        exit(0);
    }
    temp->data = value;
    temp->link = p;
    p = temp;
    return(p);
}
```

# Pop (linked list)

top



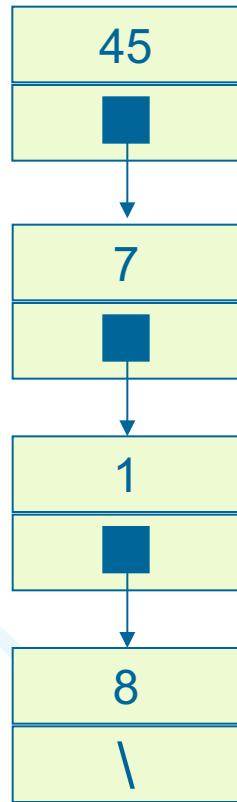
Temp

```
struct node *pop(struct node *p, int *value)
{
    struct node *temp;
    if (p==NULL)
    {
        printf(" The stack is empty can
               not pop Error\n");
        exit(0);
    }
    *value = p->data;
    temp = p;
    p = p->link;
    free(temp);
    return (p);
}
```

Giá trị tại phần tử đầu tiên cần  
phải được lưu giữ trước  
khi thực hiện theo tác pop

# Pop (linked list)

top



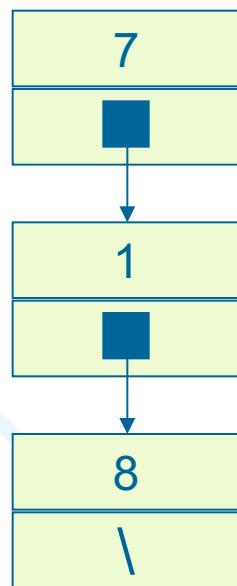
Temp

```
struct node *pop(struct node *p, int *value)
{
    struct node *temp;
    if(p==NULL)
    {
        printf(" The stack is empty can
               not pop Error\n");
        exit(0);
    }
    *value = p->data;
    temp = p;
    p = p->link;
    free(temp);
    return(p);
}
```

# Pop (linked list)

top

Temp

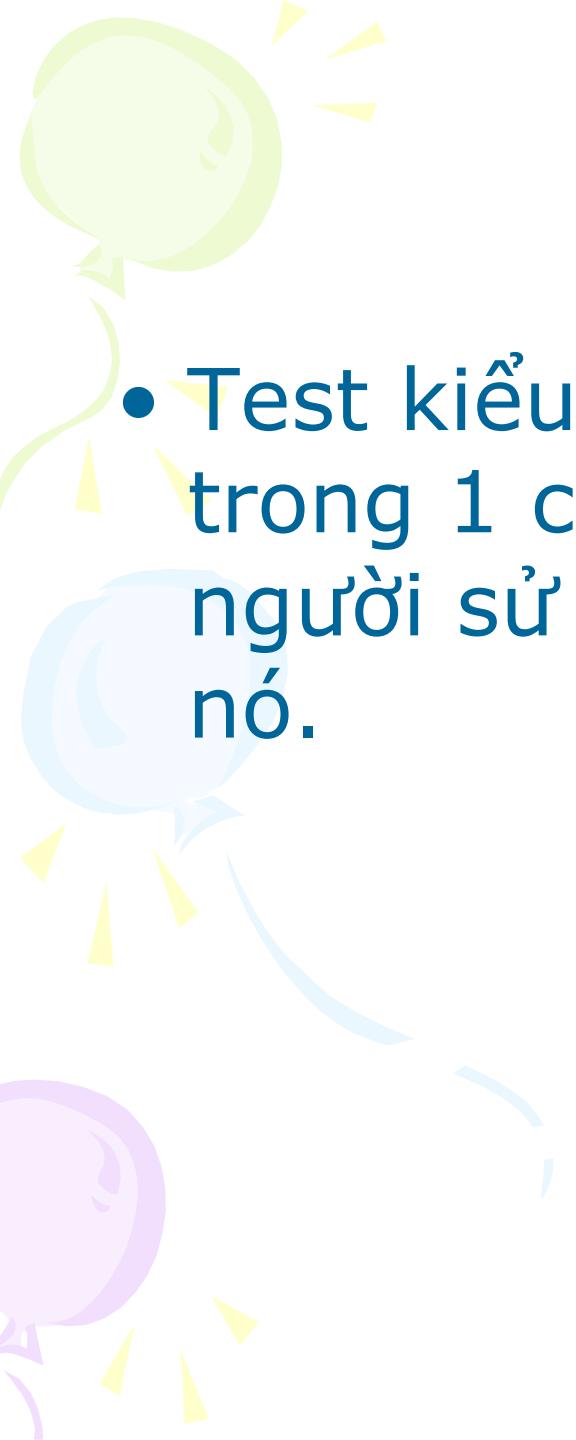


```
struct node *pop(struct node *p, int *value)
{
    struct node *temp;
    if(p==NULL)
    {
        printf(" The stack is empty can
               not pop Error\n");
        exit(0);
    }
    *value = p->data;
    temp = p;
    p = p->link;
    free(temp);
    return(p);
}
```

# Sử dụng stack trong chương trình

```
# include <stdio.h>
# include <stdlib.h>
void main()
{
    struct node *top = NULL;
    int n,value;
    do
    {
        do
        {
            printf("Enter the element
to be pushed\n");
            scanf("%d",&value);
            top = push(top,value);
            printf("Enter 1 to
continue\n");
            scanf("%d",&n);
        } while(n == 1);
    }
```

```
printf("Enter 1 to pop an element\n");
scanf("%d",&n);
while( n == 1)
{
    top = pop(top,&value);
    printf("The value poped is
%d\n",value);
    printf("Enter 1 to pop an element\n");
    scanf("%d",&n);
}
printf("Enter 1 to continue\n");
scanf("%d",&n);
} while(n == 1);
}
```

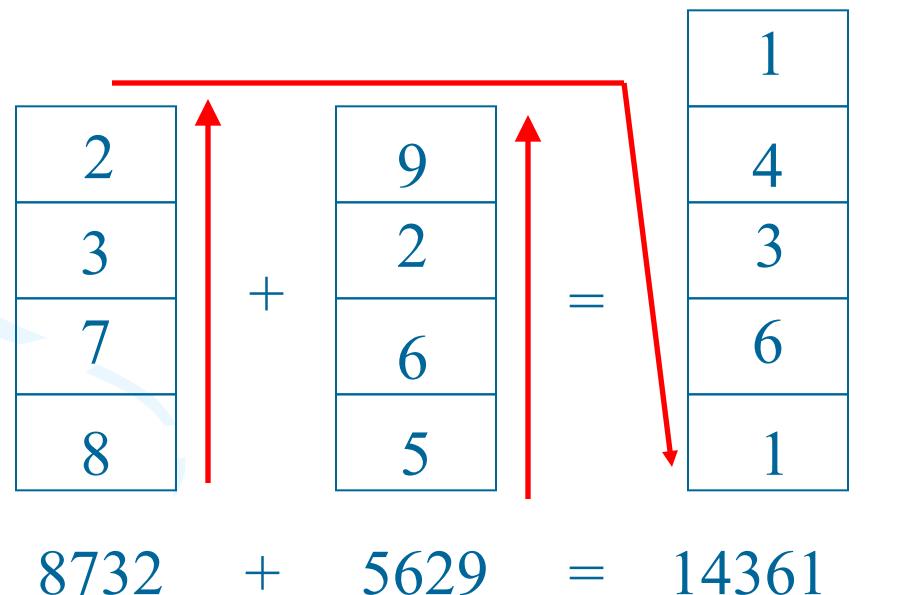


# Exercises

- Test kiểu stack bạn đã định nghĩa trong 1 chương trình mà nhận từ người sử dụng 1 xâu, và đảo ngược nó.

# Cộng những số cực lớn

- Xử lý những số này như là 1 xâu chữ số, lưu trữ những số tương ứng với các chữ số này trong 2 stacks. Sau đó thực hiện phép cộng bằng cách lấy các số (pop()) trong stack ra.



# Cộng những số cực lớn: Thuật toán chi tiết

- Đọc các chữ số của số đầu tiên và lưu trữ các số tương ứng của chúng vào **stack**.
- Đọc các chữ số của số thứ 2 và trữ các số tương ứng của chúng vào **stack khác**.
- **result=0;**
- Trong khi còn ít nhất 1 **Stack** không rỗng:
  - **Pop** 1 số từ mỗi **stack** không rỗng ra và cộng chúng.
  - **Push** tổng (trừ đi 10 nếu cần thiết) vào **stack kết quả** .
  - Trữ số nhớ ở trong **result**.
- Push số nhớ vào **stack kết quả** nếu nó khác 0.
- **Pop** các số từ **stack kết quả** ra và trình diễn chúng ra màn hình.

# Exercise 4.1 Stack sử dụng mảng

- Giả thiết rằng bạn lập 1 quyển danh bạ điện thoại.
- Khai báo 1 cấu trúc “address” gồm ít nhất là "name", "telephone number" and "e-mail address".
- Viết 1 chương trình copy dữ liệu của 1 address book từ 1 file vào 1 file khác sử dụng stack. Đầu tiên đọc dữ liệu của 1 address book rồi push vào 1 stack.Rồi pop dữ liệu từ stack ghi ra 1 file khác theo thứ tự mà bạn pop ra.(tức là theo thứ tự ngược với thứ tự đầu).

## Exercise 4-2: Sự chuyển đổi sang kí pháp Balan ngược sử dụng stack

- Viết 1 chương trình chuyển đổi từ biểu thức kí pháp giữa sang biểu thức theo kí pháp Balan ngược(bíểu thức hậu tố).
  - 1 biểu thức bao gồm các chữ số dương (1->9) và 4 phép toán (+ - \* /).
  - Đọc 1 biểu thức theo kí pháp giữa từ 1 đầu vào chuẩn, chuyển sang kí pháp Balan ngược và đưa ra dưới dạng đầu ra chuẩn
  - .Tham khảo sách giao khoa để có thêm chi tiết về kí pháp Balan ngược.
- Ví dụ  
3+5\*4  
là dữ liệu vào. Dữ liệu ra như sau

3 5 4 \* +



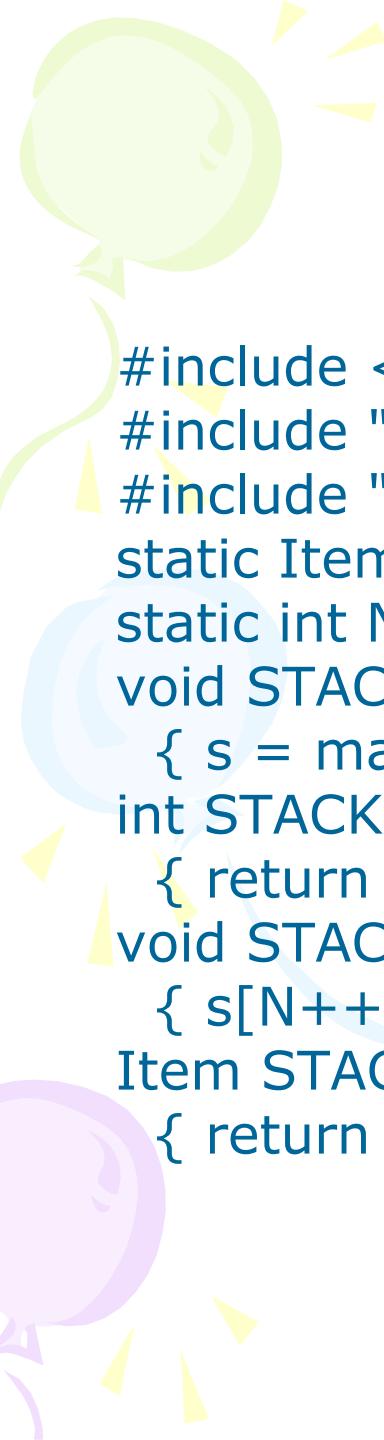
# STACK.h

void STACKinit(int);

int STACKempty();

void STACKpush(Item);

Item STACKpop();



# STACK.c

```
#include <stdlib.h>
#include "Item.h"
#include "STACK.h"
static Item *s;
static int N;
void STACKinit(int maxN)
{ s = malloc(maxN*sizeof(Item)); N = 0; }
int STACKempty()
{ return N == 0; }
void STACKpush(Item item)
{ s[N++] = item; }
Item STACKpop()
{ return s[--N]; }
```

# Tính giá trị biểu thức tiền tố

- Viết một chương trình đọc bất kỳ biểu thức có chứa phép nhân và cộng số nguyên.
- Ví dụ
- `./posteval 5 4 + 6 * => 54`



# STACK.H

void STACKinit(int);

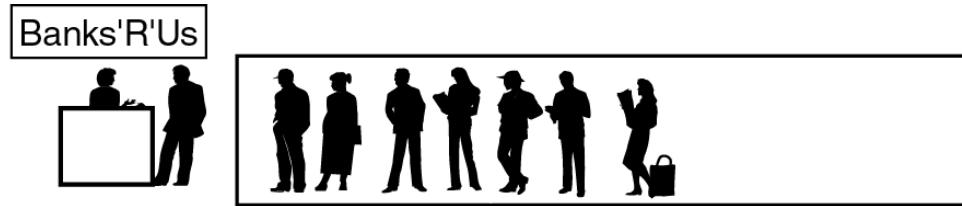
int STACKempty();

void STACKpush(Item);

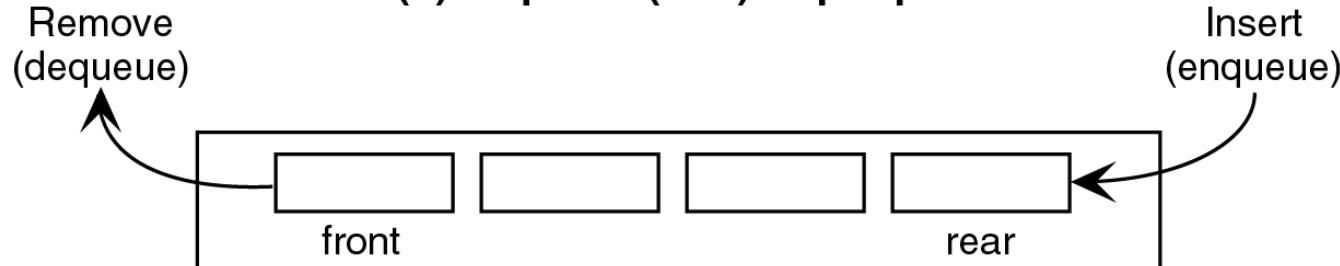
Item STACKpop();

# Queue

- Queue là 1 hàng đợi
- Cả 2 đầu đều được sử dụng: 1 để thêm fàn tử, 1 để lấy ra fàn tử.
- 1 fàn tử được thêm vào (enqueue) ở cuối hàng (rear) và lấy ra (dequeue) ở đầu hàng (front).



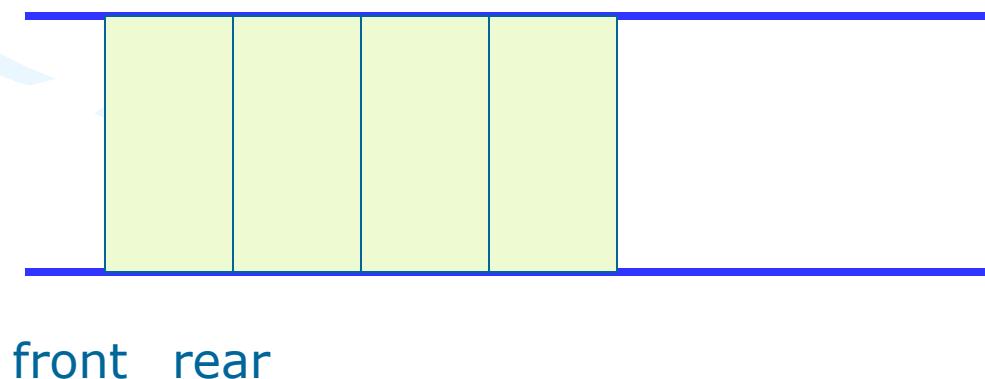
(a) A queue (line) of people

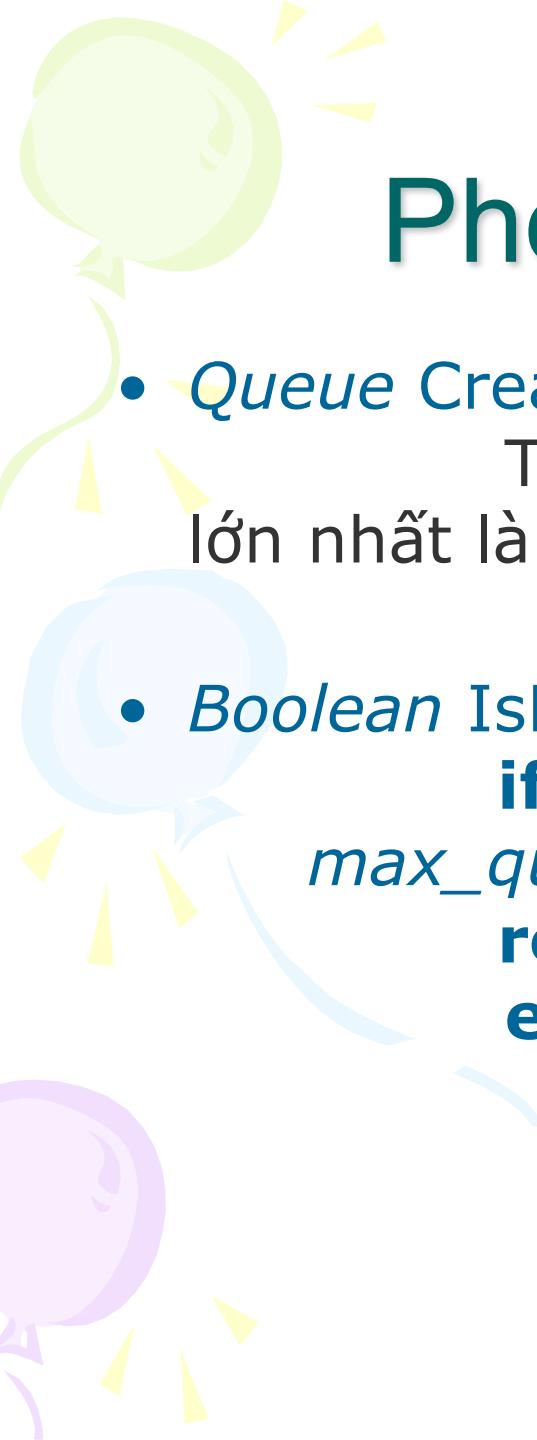


(b) A computer queue

# Cấu trúc dữ liệu FIFO

- Phần tử hàng đợi được loại bỏ đi theo đúng thứ tự mà chúng được thêm vào.
  - Cấu trúc FIFO: Vào trước, Ra trước





# Phép toán trên Queue

- *Queue CreateQ(max\_queue\_size) ::=*  
    Tạo 1 hàng đợi rỗng trong có kích thước  
    lớn nhất là *max\_queue\_size*
- *Boolean IsFullQ(queue, max\_queue\_size) ::=*  
    **if**(nếu số fân tử trong hàng đợi ==  
*max\_queue\_size*)  
    **return TRUE**  
    **else return FALSE**



# Phép toán trên Queue

- *Queue* EnQ(*queue*, *item*) ::=  
**if** (IsFullQ(*queue*)) *queue\_full*  
**else** insert *item* at rear of *queue* and  
return *queue*
- *Boolean* IsEmptyQ(*queue*) ::=  
**if** (*queue* == CreateQ(*max\_queue\_size*))  
**return** *TRUE*  
**else return** *FALSE*
- *Element* DeQ(*queue*) ::=  
**if** (IsEmptyQ(*queue*)) **return**  
**else** remove and return the *item* at  
front of *queue*.

# Cài đặt Queue bằng mảng và cấu trúc

```
#define MaxLength 100
typedef ... ElementType;
typedef struct {
    ElementType Elements[MaxLength];
    //Store the elements
    int Front, Rear;
} Queue;
```

# Khởi tạo và kiểm tra các trạng thái

```
void MakeNull_Queue(Queue *Q) {  
    Q->Front=-1;  
    Q->Rear=-1;  
}  
  
int Empty_Queue(Queue Q) {  
    return Q.Front== -1;  
}  
  
int Full_Queue(Queue Q) {  
    return (Q.Rear-Q.Front+1)==MaxLength;  
}
```

# Enqueue

```
void EnQueue(ElementType X, Queue *Q) {  
    if (!Full_Queue(*Q)) {  
        if (Empty_Queue(*Q)) Q->Front=0;  
        Q->Rear=Q->Rear+1;  
        Q->Element[Q->Rear]=X;  
    }  
    else printf("Queue đã đầy!");  
}
```

# Dequeue (huỷ queue)

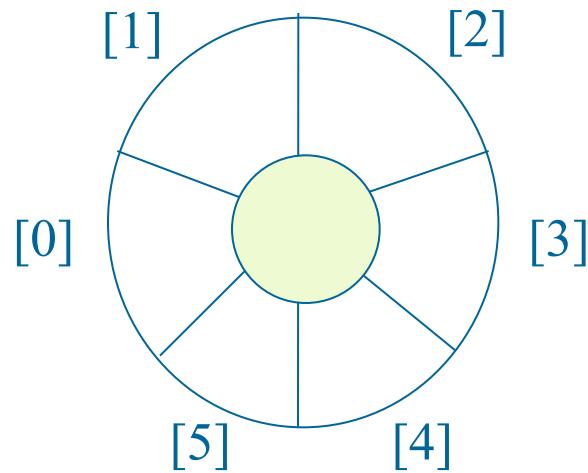
```
void DeQueue (Queue *Q) {  
    if (!Empty_Queue (*Q)) {  
        Q->Front=Q->Front+1;  
        if (Q->Front > Q->Rear)  
            MakeNull_Queue (Q);  
        // Queue trở thành rỗng  
    }  
    else printf ("Queue bị rỗng!");  
}
```

## Cách cài đặt 2: xem một mảng như là queue vòng tròn

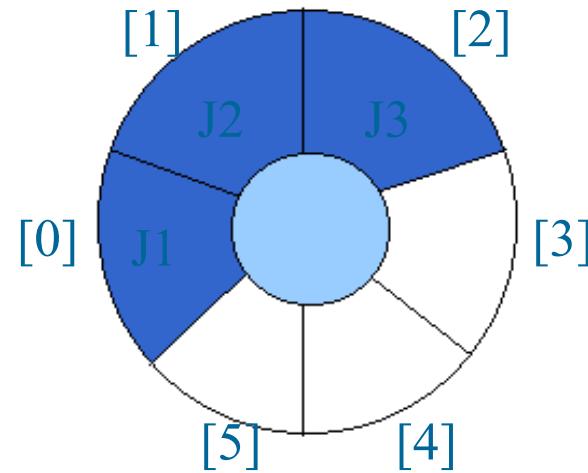
front: 1 vị trí ngược chiều kim đồng hồ từ phần tử đầu tiên

rear: vị trí cuối hiện tại

EMPTY QUEUE



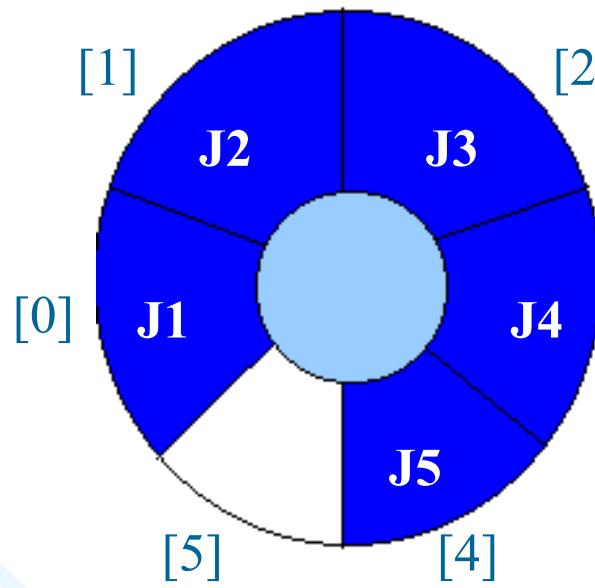
**front = 0**  
**rear = 0**



**front = 0**  
**rear = 3**

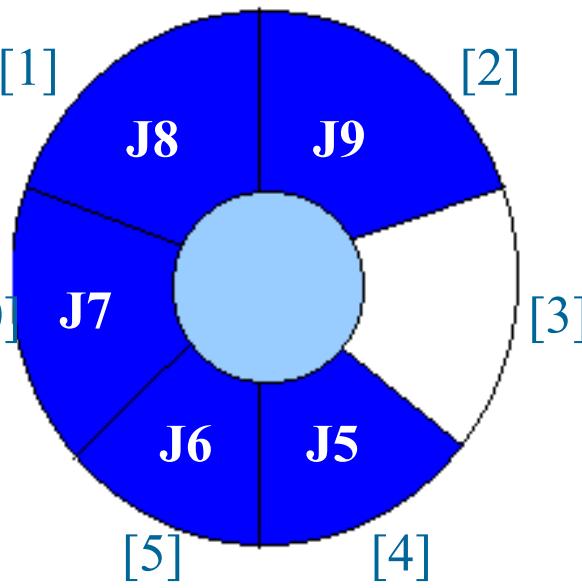
# Vấn đề: Còn một vị trí khi Queue bị đầy

FULL QUEUE



**front = 0**  
**rear = 5**

FULL QUEUE



**front = 4**  
**rear = 3**

# Queue có đầy hay không?

```
int Full_Queue(Queue Q) {  
    return (Q.Rear-Q.Front+1) %  
        MaxLength==0;  
}
```

# Dequeue

```
void DeQueue(Queue *Q) {  
    if (!Empty_Queue(*Q)) {  
        //Nếu queue chỉ chứa một phần tử  
        if (Q->Front==Q->Rear)  
            MakeNull_Queue(Q);  
        else Q->Front=(Q->Front+1) % MaxLength;  
    }  
    else printf("Queue rỗng!");  
}
```

# Enqueue

```
void EnQueue(ElementType X, Queue *Q) {  
    if (!Full_Queue(*Q)) {  
        if (Empty_Queue(*Q)) Q->Front=0;  
        Q->Rear=(Q->Rear+1) % MaxLength;  
        Q->Elements[Q->Rear]=X;  
    } else printf("Queue đầy!");  
}
```

# Cài đặt Queue sử dụng danh sách

- Exercise: Queue là một danh sách cụ thể. Cài đặt các thao tác trên queue bằng cách sử dụng lại các thao tác trên danh sách.

# Cài đặt Queue sử dụng danh sách

```
typedef ... ElementType;  
typedef struct Node{  
    ElementType Element;  
    Node* Next; //pointer to next element  
};  
typedef Node* Position;  
typedef struct{  
    Position Front, Rear;  
} Queue;
```

# Khởi tạo một queue rỗng

```
void MakeNullQueue (Queue *Q) {  
    Position Header;  
    Header=(Node*) malloc (sizeof (Node)) ;  
    // Allocation Header  
    Header->Next=NULL;  
    Q->Front=Header;  
    Q->Rear=Header;  
}
```

# Is-Empty

```
int EmptyQueue(Queue Q) {  
    return (Q.Front==Q.Rear);  
}
```

# EnQueue

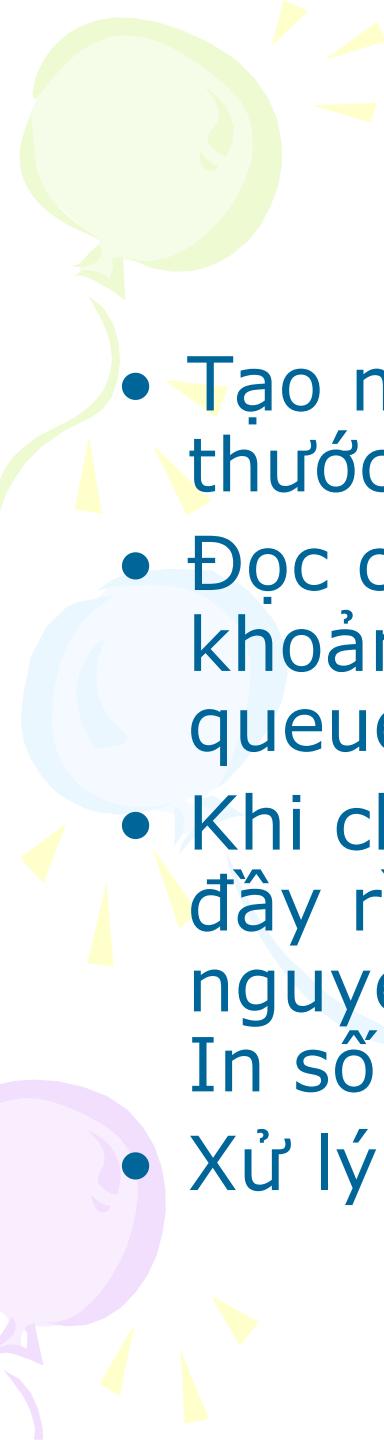
```
void EnQueue(ElementType X, Queue *Q) {  
    Q->Rear->Next=  
        (Node*) malloc(sizeof(Node));  
    Q->Rear=Q->Rear->Next;  
    Q->Rear->Element=X;  
    Q->Rear->Next=NULL;  
}
```

# DeQueue

```
void DeQueue (Queue *Q) {  
    if (!Empty_Queue (Q) ) {  
        Position T;  
        T=Q->Front;  
        Q->Front=Q->Front->Next;  
        free (T) ;  
    }  
    else printf ("Error: Queue is empty." );  
}
```

# Exercise 4-3: Queues sử dụng Lists

- Giả định rằng bạn viết một danh bạ điện thoại.
- Khai báo cấu trúc "Address" chứa các trường "name", "telephone number" và "e-mail address".
- Viết một chương trình sao chép dữ liệu của một danh bạ từ file tới file khác, sử dụng queue.
  - Trước tiên, đọc dữ liệu của danh bạ từ file và thêm chúng vào queue.
  - Sau đó lấy dữ liệu từ queue và ghi ra file theo thứ tự lấy được. Nói cách khác: dữ liệu đọc vào trước sẽ được đọc ra trước; dữ liệu đọc vào sau sẽ được lấy ra sau.



# Exercises

- Tạo một queue chứa các số nguyên. Kích thước của queue cố định là 10.
- Đọc các số nguyên được chia cách bởi khoảng trống từ bàn phím và thêm vào queue.
- Khi chương trình đọc số thứ 11, queue đã đầy rồi. Do đó chương trình sẽ xoá số nguyên đầu tiên và thêm số thứ 11 vào. In số nguyên bị xoá ra màn hình
- Xử lý tất cả số nguyên theo cách trên.

# Exercise: To Do List (danh sách việc cần làm)

- Bằng cách dùng queue, viết chương trình quản lý To Do List với 1 menu để thêm, sửa, xoá các phần tử trong danh sách.
- Một mục trong danh sách có các trường sau:
  - Time
  - Place
  - People
  - Description.
- Trường time có thể là system time (thời gian hiện tại của hệ thống) tại thời điểm nhập dữ liệu.