NỘI DUNG

DANH SÁCH LIÊN KẾT ĐƠN (LIST)

Các cấu trúc đặc biệt của danh sách đơn

- Stack (ngăn xếp): Là 1 vật chứa các đối tượng làm việc theo cơ chế LIFO (Last In First Out), tức việc thêm 1 đối tượng vào Stack hoặc lấy 1 đối tượng ra khỏi Stack được thực hiện theo cơ chế "vào sau ra trước"
- Queue (hàng đợi): Là 1 vật chứa các đối tượng làm việc theo cơ chế FIFO (First In First Out), tức việc thêm 1 đối tượng vào hàng đợi hay lấy 1 đối tượng ra khỏi hàng đợi thực hiện theo cơ chế "vào trước ra trước".

Ứng dụng Stack và Queue

- Stack:
  - Trình biên dịch
  - Khử đệ qui đuôi
  - Lưu vết các quá trình quay lui, vét cạn
- Queue:
  - Tổ chức lưu vết các quá trình tìm kiếm theo chiều rộng, và quay lui vét cạn,
  - Tổ chức quản lý và phân phối tiến trình trong các hệ điều hành,
  - Tổ chức bộ đệm bàn phím, …

Các thao tác trên Stack

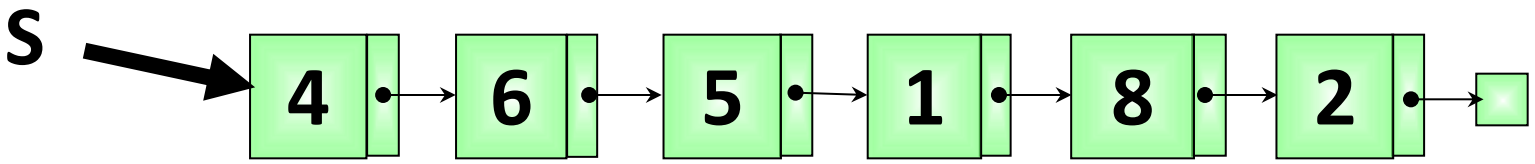
- Push(o): Thêm đối tượng o vào Stack
- Pop(): Lấy đối tượng từ Stack
- isEmpty(): Kiểm tra Stack có rỗng hay không
- Top(): Trả về giá trị của phần tử nằm đầu Stack mà không hủy nó khỏi Stack.

Cài đặt Stack

- Dùng mảng 1 chiều


```
Data   S [N];
int    t;
```

- Dùng danh sách liên kết đơn



List S

- **Thêm và hủy cùng phía**

Cài Stack bằng mảng 1 chiều

- *Cấu trúc dữ liệu của Stack*

```
typedef struct tagStack
{
   int a[max];
   int t;
}Stack;
```

- *Khởi tạo Stack:*

```
void CreateStack(Stack &s)
{
   s.t=-1;
}
```


Kiểm tra tính rỗng và đầy của Stack

```
int IsEmpty(Stack s)//Stack có rỗng hay không
{
    if(s.t==-1)
        return 1;
    else
        return 0;
}
int IsFull(Stack s) //Kiểm tra Stack có đầy hay không
{
    if(s.t>=max)
        return 1;
    else
        return 0;
}
```


Thêm 1 phần tử vào Stack

```
int Push(Stack &s, int x)
{
   if(IsFull(s)==0)
   {
        s.t++;
        s.a[s.t]=x;
        return 1;
   }
   else
        return 0;
}
```


Lấy 1 phần tử từ Stack

```
int Pop(Stack &s, int &x)
{
    if(IsEmpty(s)==0)
    {
        x=s.a[s.t];
        s.t--;
        return 1;
    }
    else
        return 0;
}
```


Cài Stack bằng danh sách liên kết

- Kiểm tra tính rỗng của Stack

```
int IsEmpty(List &s)
{
    if(s.pHead==NULL)//Stack rong
        return 1;
    else
        return 0;
}
```


Thêm 1 phần tử vào Stack

```
void Push(List &s,Node *Tam)
{
   if(s.pHead==NULL)
   {
        s.pHead=Tam;
        s.pTail=Tam;
   }
   else
   {
        Tam->pNext=s.pHead;
        s.pHead=Tam;
   }
}
```


Lấy 1 phần tử từ Stack

```
int Pop(List &s,int &trave)
{   Node *p;
    if(IsEmpty(s)!=1)
    {
        if(s.pHead!=NULL)
        {       p=s.pHead;
                trave=p->Info;
                s.pHead=s.pHead->Next;
                if(s.pHead==NULL)
                        s.Tail=NULL;
                return 1;
                delete p;
        }
    }
    return 0;
}
```


Các thao tác trên Queue

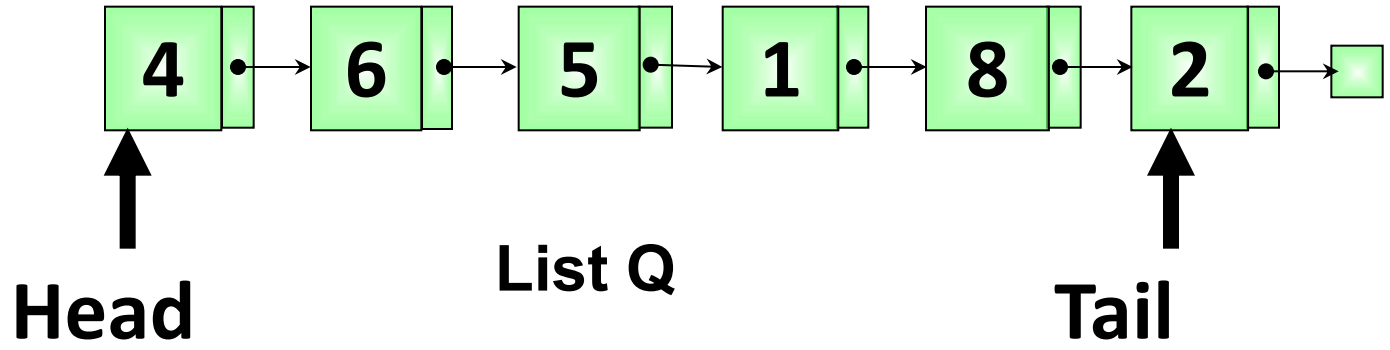
- EnQueue(O): Thêm đối tượng O vào cuối hàng đợi.
- DeQueue(): Lấy đối tượng ở đầu hàng đợi
- isEmpty(): Kiểm tra xem hàng đợi có rỗng hay không?
- Front(): Trả về giá trị của phần tử nằm đầu hàng đợi mà không hủy nó.

Cài đặt Queue

- Dùng mảng 1 chiều


```
Data   S [N];
int    f,r;
```

- Dùng danh sách liên kết đơn



* **Thêm và hủy Khác phía**

Cài đặt Queue bằng mảng 1 chiều

- *Cấu trúc dữ liệu:*

```
typedef struct tagQueue
{
    int a[100];
    int Front; //chỉ số của phần tử đầu trong Queue
    int Rear; //chỉ số của phầ tử cuối trong Queue
}Queue;
```

- *Khởi tạo Queue rỗng*

```
void CreateQueue(Queue &q)
{  q.Front=-1;
   q.Rear=-1;
}
```


Lấy 1 phần tử từ Queue

```
int DeQueue(Queue &q,int &x)
{
    if(q.Front!=-1)          //queue khong rong
    {
        x=q.a[q.Front];
        q.Front++;
        if(q.Front>q.Rear)//truong hop co mot phan tu
        {
            q.Front=-1;
            q.Rear=-1;
        }
        return 1;
    }
    else //queue trong
    {
        printf("Queue rong");
        return 0;
    }
}
```


Thêm 1 phần tử vào Queue

```
void EnQueue(Queue &q,int x)
{
	int i;
	int f,r;
	if(q.Rear-q.Front+1==N)//queue bi day khong the them vao duoc nua
		printf("queue day roi khong the them vao duoc nua");
	else
	{
		if(q.Front==-1)
		{
			q.Front=0;
			q.Rear=-1;
		}
		if(q.Rear==N-1)//Queue đầy ảo
		{
			f=q.Front;
			r=q.Rear;
			for(i=f;i<=r;i++)
				q.a[i-f]=q.a[i];
			q.Front=0;
			q.Rear=r-f;
		}
		q.Rear++;
		q.a[q.Rear]=x;
	}
}
```

Cài đặt Queue bằng List

- Kiểm tra Queue có rỗng?

```
int IsEmpty(List &Q)
{
        if(Q.pHead==NULL)//Queue rỗng
                return 1;
        else
                return 0;
}
```


Thêm 1 phần tử vào Queue

```
void EnQueue(List &Q, Node *Tam)
{
    if(Q.pHead==NULL)
    {
        Q.pHead=Tam;
        Q.pTail=Tam;
    }
    else
    {
        Q.pTail->Next=tam;
        Q.pTail=tam;
    }
}
```


Lấy 1 phần tử từ Queue

```
int DeQueue(List &Q,int &trave)
{   Node *p;
    if(IsEmpty(Q)!=1)
    {
        if(Q.pHead!=NULL)
        {       p=Q.pHead;
                trave=p->Info;
                Q.pHead=Q.pHead->Next;
                if(Q.pHead==NULL)
                        Q.pTail=NULL;
                return 1;
                delete p;
        }
    }
    return 0;
}
```