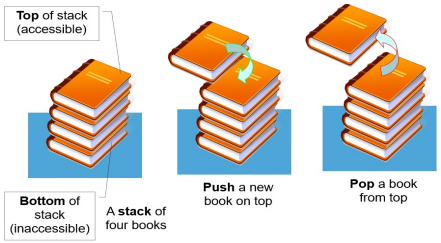
CHƯƠNG V:

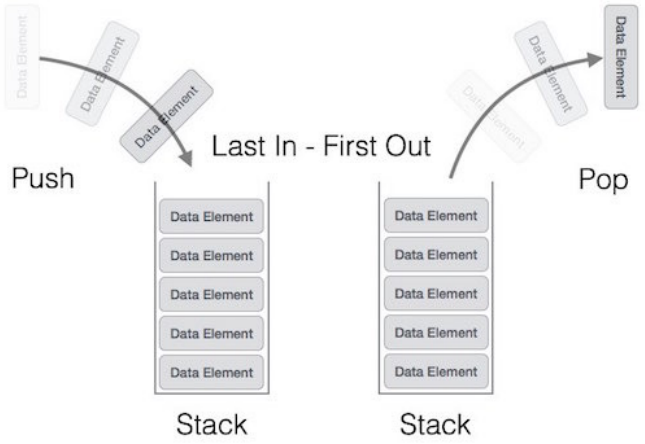
STACK - QUEUE

I. Stack – Ngăn xếp:

1. Giới thiệu:

- Một **ngăn xếp** là một cấu trúc dữ liệu trừu tượng (Abstract Data Type – viết tắt là ADT), được sử dụng trong hầu hết mọi ngôn ngữ lập trình
- **Ngăn xếp** với hai phép toán **bổ sung một phần tử** vào **đầu** danh sách và **loại bỏ một phần tử** cũng ở **đầu** của danh sách
- Trong ngăn xếp một phần tử vào **sau sẽ bị đẩy ra trước và phần tử vào trước sẽ bị đấy ra sau** ⇔ gọi là danh sách **LIFO** (*Last In First Out)*





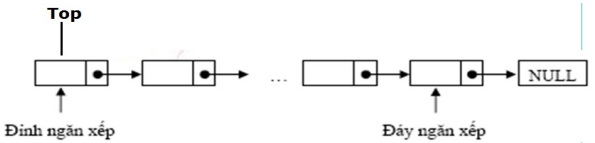
* Các thao tác cơ bản với Stack

- ***initStack*** *(Stack)*: Khởi tạo Stack rỗng
- ***isEmpty*** *(Stack)*: Kiểm tra Stack có rỗng hay không?
- ***isFull*** *(Stack):* kiểm tra danh sách đầy
- ***Push*** *(Stack, Data)*: Đẩy phần tử item vào Stack
- ***Pop*** *(Stack)*: Hủy bỏ một phần tử khỏi Stack
- ***Top*** *(Stack):* Xem nội dung của phần tử đầu tiên của Stack

2. Cài đặt Stack

- bằng mảng
  - Sử dụng mảng một chiều để chứa các phần tử
  - Cần chỉ số **top** để chỉ **đỉnh**
    - **Thêm – xóa trên vị trí Top**
  - Chỉ số đầu (0) để chỉ đáy

- bằng danh sách liên kết
  - Sử dụng một danh sách liên kết đơn
  - khai báo và định nghĩa phần tử đầu (**Top**) để chỉ **đỉnh**
    - **Thêm-xóa thực hiện tại vị trí Top**
  - Phần tử cuối là đáy danh sách



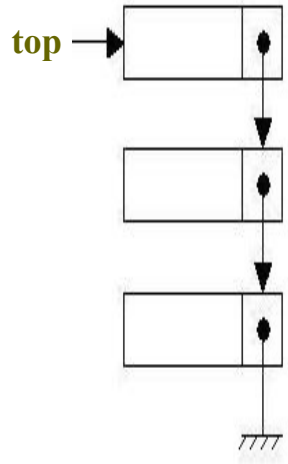
Cài đặt Stack dưới dạng liên kết đơn

- void ***initStack*** *(Stack)*: Khởi tạo Stack rỗng
- int ***isEmpty*** *(Stack)*: Kiểm tra Stack có rỗng hay không?
- ~~***int isFull*** *(Stack):* kiểm tra danh sách đầy~~
- void ***Push*** *(Stack, Data)*: Đẩy phần tử item vào Stack
- Data ***Pop*** *(Stack)*: Hủy bỏ một phần tử khỏi Stack
- Data ***Top*** *(Stack):* Xem nội dung của phần tử đầu tiên của Stack

* Khai báo cấu trúc dữ liệu

- Stack là một danh sách liên kết đơn. Được định nghĩa với cấu trúc:

```
struct StackNode
{
      Data  info;
      struct StackNode *next;
};
struct  Stack
{
      Node top;
};
```



* Khởi tạo Stack

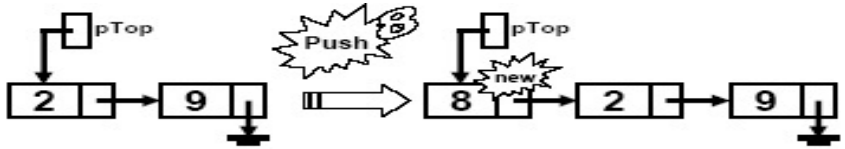
```
void InitStack ( Stack &s )
{
     s.top = NULL;
}
```


* Kiểm tra ngăn xếp rỗng

```
int isEmpty ( Stack s)
{
   if ( s.top == NULL )
      return 1;
   else
      return 0;
}
```


* Thêm phần tử vào danh sách:

```
void Push( Stack &s, Node *p)
{
   if ( s.top == NULL)
   {
      s.top = p;
   }
   else
   {
      p->next= s.top;
      s.top = p;
   }
}
```

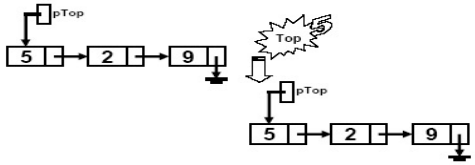


* Lấy phần tử khỏi danh sách:

```
int Pop( Stack &s)
{
   Node *p;
   if ( isEmpty(s) == 1 )
   {
      printf("Ngan xep rong");
      exit(1);
   }
   else
   {
      int tg;
      tg = s.top->info;
      p = s.top;
      s.top = s.top->next;
      delete p;
      return tg;
   }
}
```

* Lấy nội dung phần tử đầu tiên Stack:

```
int Top(Stack &s)
{
   int tg;
   if ( isEmpty(s) == 1 )
   {
      printf("Ngan xep rong");
      exit(1);
   }
   else
   {
      tg = s.top-> info;
   }
   return tg;
}
```



3. Ứng dụng ngăn xếp

- **Chuyển đổi cơ số**
- **Đảo ngược xâu ký tự**
- Tính giá trị của một biểu thức
- Chuyển biểu thức dạng trung tố sang hậu tố
- Thuật toán sắp xếp QuickSort

a. Bài toán chuyển đổi cơ số

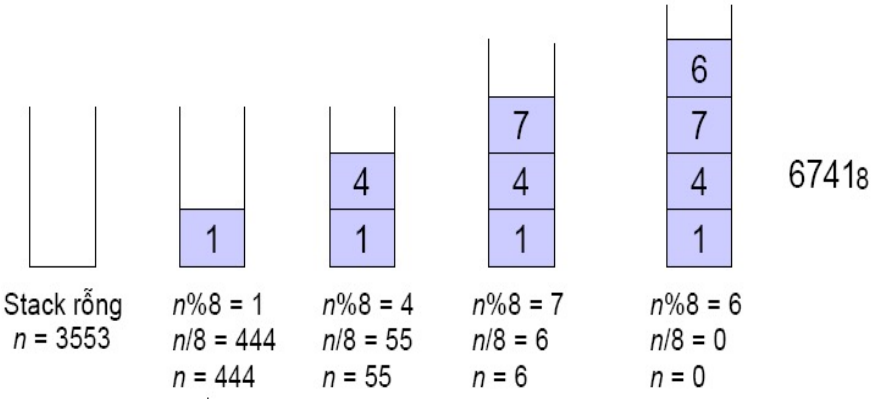
- Yêu cầu tính
  - $72_{10} = ?_4$
  - $53_{10} = ?_2$
- Kết quả:

$$\mathbf{72}_{10} = 1.4^3 + 0.4^2 + 2.4^1 + 0.4^0$$
$$= \mathbf{1020}_4$$

$$53_{10} = 1.2^5 + 1.2^4 + 0.2^3 + 1.2^2 + 0.2^1 + 1.2^0 =$$
$$= \mathbf{110101}_2$$

*Thuật toán

- **Cho số nguyên n trong hệ thập phân và cơ số cần chuyển sang là b**
- **Bước 1**: Khởi tạo Stack rỗng
- **Bước 2**: Tạo Stack
  - Tính kết quả = *n* % *b*.
  - Đẩy kết quản n%b vào Stack.
  - Thay ***n = n / b*** (để tìm các số tiếp theo).
  - Quá trình lặp cho đến khi **n == 0**.
- **Bước 3**: In kết quả (Đọc từ Stack)
  - Lấy lần lượt các chữ số lưu trong Stack,
  - Chuyển sang dạng ký tự tương ứng với hệ cơ số và in ra màn hình



Cài đặt Chuyển đổi cơ số (n,b)

- Nhập vào số nguyên n ở hệ cơ số thập phân
- Nhập vào hệ cơ số b tùy ý
- Yêu cầu: Cho biết kết quả chuyển số nguyên n sang hệ cơ số b tương ứng

Cài đặt chuyển đổi cơ số

```
//Cài đặt các thao tác của Stack
        - Khai báo CTDL dạng Stack
        - Đ/n hàm khởi tạo ⇔ initStack
        - Đ/n hàm kiểm tra rỗng ⇔ isEmpty
        - Đ/n hàm bổ sung phần tử ⇔ Push
        - Đ/n hàm lấy và xóa phần tử ⇔ Pop

//Cài đặt hàm main thực hiện thuật toán
        int main()
        {
                … //gọi lần lượt các hàm của Stack để áp dụng
        }
```


b. Bài toán đảo ngược xâu ký tự

- **Bài toán**: *cho một chuỗi ký tự, yêu cầu đảo ngược chuỗi ký tự đã có* (ứng dụng Stack)

- **Thuật toán**
  - **Bước 1**: Tạo ngăn xếp
    - Duyệt từ đầu xâu đến cuối xâu
    - Lần lượt cho các ký tự vào ngăn xếp – *cho hết các ký tự vào ngăn xếp*
  - **Bước 2**: Lấy từ ngăn xếp ra
    - Lần lượt lấy các phần tử từ ngăn xếp và in ra

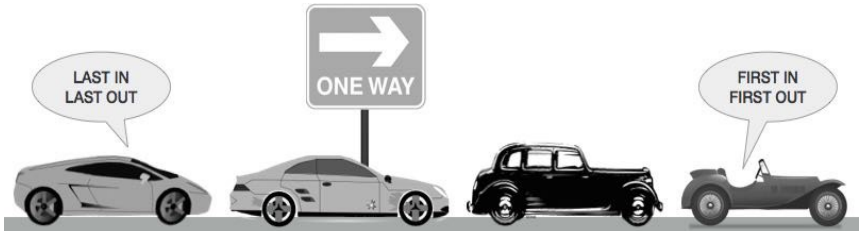
* Cài đặt

```
int main()
{
    char st[100];
    int i;
    Stack s;
    Node *p;

    InitStack(s);
    printf("Nhap chuoi ky tu:"); gets(st);
    printf("\n Chuoi da nhap:%s", st);

    for (int i=0; i<strlen(st); i++)
    {
        p = getNode(st[i]);
        Push( s, p);
    }

    printf("\n Xau dao nguoc:");
    while (!isEmpty(s))
        printf("%c", Pop(s));
    getch();
}
```


II. Queue – Hàng đợi

1. Giới thiệu:

- ✓ **Hàng đợi (Queue)** là một kiểu danh sách trong đó có phép toán **bổ sung một phần tử vào cuối danh sách** và **loại bỏ một phần tử ở đầu danh sách.**
- ✓ Trong hàng đợi một phần tử vào trước sẽ bị đẩy ra trước và phần tử vào sau sẽ bị đẩy ra sau ⇔ gọi là danh sách **FIFO** (***First In First Out***)

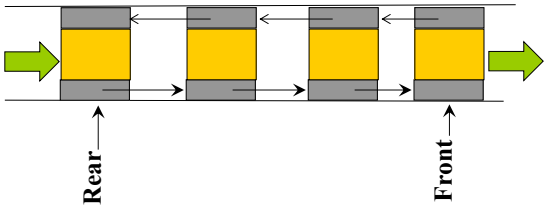


* Các thao tác cơ bản với Queue

- ***initQueue*** *(Queue)*: Khởi tạo Queue rỗng
- ***isEmpty*** *(Queue)*: Kiểm tra Queue có rỗng hay không?
- ***isFull*** *(Queue):* kiểm tra danh sách đầy
- ***Put*** *(Queue, Data)*: Đẩy một phần tử vào Queue
- ***Get*** *(Queue)*: Hủy bỏ một phần tử khỏi Queue

2. Biểu diễn Queue:

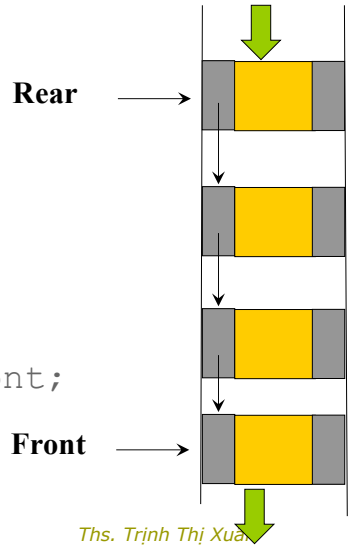
- **Mảng**
  - Dùng hai chỉ số **Rear** và **Front** để lưu giữ điểm đầu và điểm cuối hàng đợi
- **Danh sách liên kết**
  - Dùng **DSLK đôi** với điểm đầu **Rear** và điểm cuối **Front**
    - Thêm vào **Rear**
    - Lấy ra từ **Front**



*Cài đặt Queue bằng Danh sách liên kết

✓Khai báo cấu trúc:

```
struct NodeQueue
{
   int info;
   NodeQueue *next, *pre;
};

struct  Queue
{
   NodeQueue  *Rear, *Front;
   int count;
};
```



- **Khởi tạo**

```
void  InitQueue( Queue &q )
{
   q.Rear = NULL;
   q.Front = NULL;
   q.Count =0;
}
```

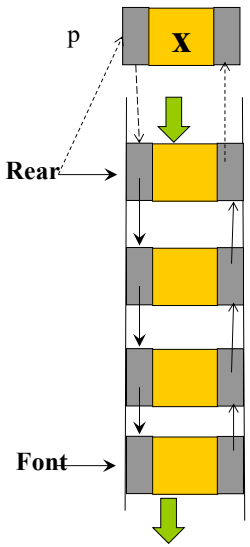

Kiểm tra hàng đợi rỗng

```
int  isEmpty( Queue q )
{
  if ( q.Rear == NULL )
    return 1;
  else
    return 0;
}
```

* Thêm phần tử vào Queue:

*Thêm phần tử vào vị trí **Rear***

```
void  Put( Queue &q, NodeQueue *p )
{
   if ( q.Rear == NULL )
   {
      q.Rear = p;
      q.Front = p;
   }
   else
   {
      p->next = q.Rear;
      q.Rear->prev = p;
      q.Rear = p;
      //q.count ++;
   }
}
```



* Hủy bỏ phần tử khỏi danh sách:

*Hủy bỏ phần tử tại vị trí **Front** của DS*

```
int Get( Queue &q)
{
    int x;
    if ( isEmpty(q) == 1)
        cout<<"\nHang doi rong\n";
    else
    {
        NodeQueue *p;
        x = q.Front->info;
        p = q.Front;
        q.Front = q.Front->next;
              //q.count--;
        delete p;
    }
        return x;
}
```