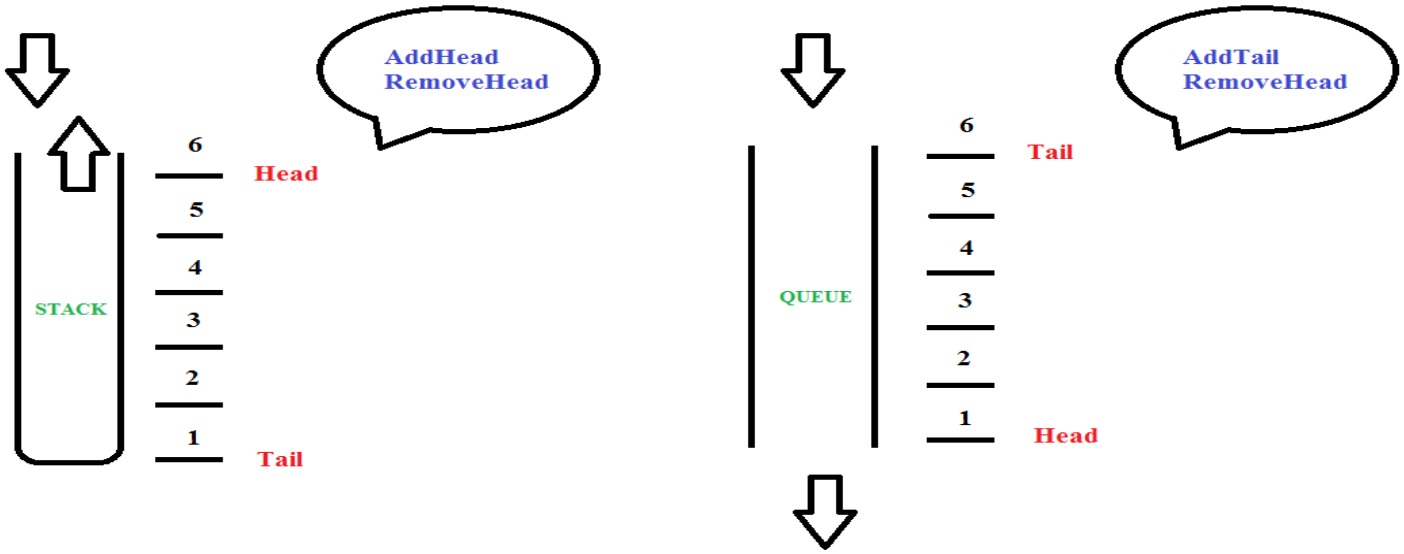
STACK - QUEUE

DATA STRUCTURES AND ALGORITHMS

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".

Các cấu trúc đặc biệt của danh sách đơn



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
- isFull(): Kiểm tra Stack có đầy hay không
- Top(): Trả về giá trị của phần tử nằm **đầu** Stack mà không hủy nó khỏi Stack.

Các thao tác trên Queue

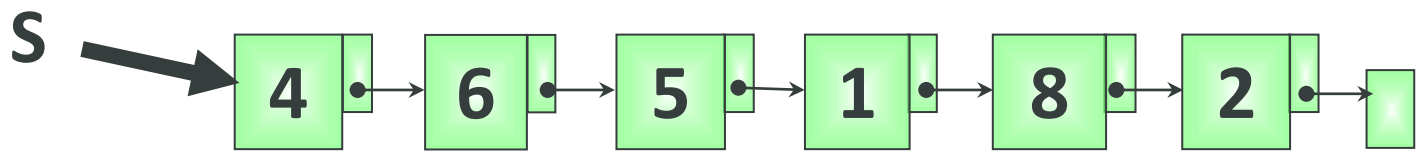
- EnQueue(O): Thêm đối tượng O vào cuối hàng đợi.
- DeQueue(): Lấy đối tượng ở đầu hàng đợi
- isEmpty(): Kiểm tra xem hàng đợi có rỗng hay không?
- Front(): Trả về giá trị của phần tử nằm **đầu** hàng đợi mà không hủy nó.
- Back(): Trả về giá trị của phần tử nằm **cuối** hàng đợi mà không hủy nó.

Cài đặt Stack

- Dùng mảng 1 chiều

Data S [N];
int t;

- Dùng danh sách liên kết đơn



List S

- Chú ý: Thêm và hủy cùng phía

Cài Stack bằng mảng 1 chiều

- Cấu trúc dữ liệu của Stack

```
struct Stack {
    int a[MAX];
    int t;
};
```

- Khởi tạo Stack

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

Kiểm tra tính rỗng và đầy của Stack

```
bool isEmpty(Stack s) { //Stack có rỗng hay không
    if (s.t == -1)
        return 1;
    return 0;
}

bool isFull(Stack s) { //Kiểm tra Stack có đầy hay không
    if (s.t >= MAX)
        return 1;
    return 0;
}
```

Thêm 1 phần tử vào Stack

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

Lấy 1 phần tử từ Stack

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

Thêm 1 phần tử vào Stack

```
void Push(LIST &s, NODE *p) { // AddHead
    if (s.pHead == NULL) {
        s.pHead = p;
        s.pTail = p;
    }
    else {
        p->pNext = s.pHead;
        s.pHead = p;
    }
}
```

Lấy 1 phần tử từ Stack

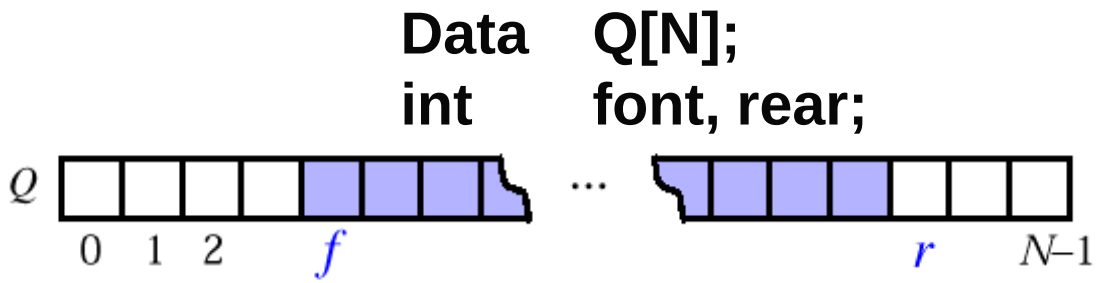
```
bool Pop(LIST &s, int &x) {
    NODE *p;
    if (isEmpty(s) != 1) {
        if (s.pHead != NULL) {
            p = s.pHead;
            x = p->info;
            s.pHead = s.pHead->pNext;
            if (s.pHead == NULL)
                s.pTail = NULL;
            return 1;
        }
    }
    return 0;
}
```

Các thao tác trên Queue

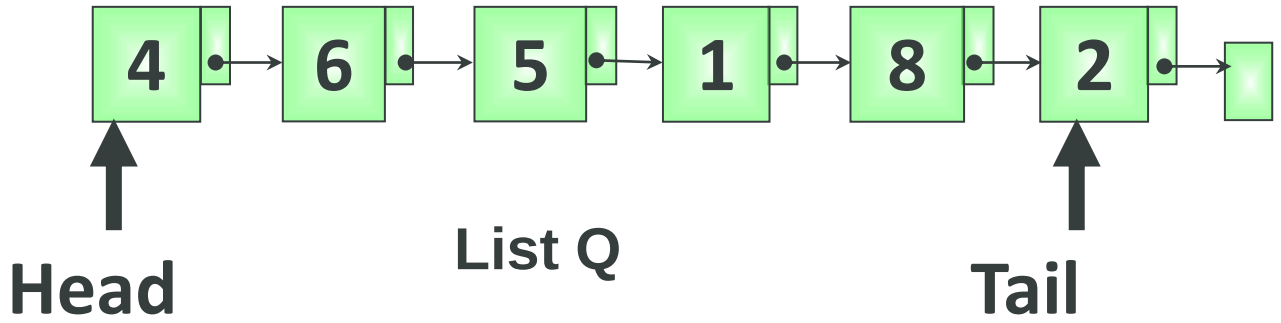
- EnQueue(O): Thêm đối tượng O vào cuối hàng đợi.
- DeQueue(): Lấy đối tượng ở đầu hàng đợi
- isEmpty(): Kiểm tra xem hàng đợi có rỗng hay không?
- Front(): Trả về giá trị của phần tử nằm đầu hàng đợi mà không hủy nó.

Cài đặt Queue

- Dùng mảng 1 chiều



- Dùng danh sách liên kết đơn



- Chú ý: **Thêm và hủy Khác phía**

Cài đặt Queue bằng mảng 1 chiều

- *Cấu trúc dữ liệu:*

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

- *Khởi tạo Queue rỗng*

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

Kiểm tra tính rỗng và đầy của Queue

```
bool isEmpty(Queue q) { // Queue có rỗng?
  if (q.Front == -1)
     return 1;
  return 0;
}

bool isFull(Queue q) { // Kiểm tra Queue có đầy?
  if (q.Rear - q.Front + 1 == MAX)
     return 1;
  return 0;
}
```

Thêm 1 phần tử vào Queue

```
void EnQueue(Queue &q, int x) {
    int f, r;
    if (isFull(q)) //queue bị đầy => không thêm được nữa
        printf("queue day roi khong the them vao duoc nua");
    else {
        if (q.Front == -1) {
            q.Front = 0;
            q.Rear = -1;
        }
        if (q.Rear == MAX - 1) { //Queue đầy ảo
            f = q.Front;
            r = q.Rear;
            for (int 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;
    }
}
```

Lấy 1 phần tử từ Queue

```
bool DeQueue(Queue &q, int &x) {
    if (isEmpty(q)==0) { //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;
    }
    //queue trong
    printf("Queue rong");
    return 0;
}
```

Cài đặt Queue bằng List

- Kiểm tra Queue có rỗng?

```
bool isEmpty(LIST &Q) {
    if (Q.pHead == NULL) //Queue rỗng
          return 1;
    return 0;
}
```

Thêm 1 phần tử vào Queue

```
void EnQueue(LIST &Q, NODE *p) {
    if (Q.pHead == NULL) {
        Q.pHead = p;
        Q.pTail = p;
    }
    else {
        Q.pTail->pNext = p;
        Q.pTail = p;
    }
}
```

Lấy 1 phần tử từ Queue

```
int DeQueue(LIST &Q, int &x) {
    NODE *p;
    if (isEmpty(Q) != 1) {
        if (Q.pHead != NULL) {
            p = Q.pHead;
            x = p->info;
            Q.pHead = Q.pHead->pNext;
            if (Q.pHead == NULL)
                Q.pTail = NULL;
            return 1;
        }
    }
    return 0;
}
```

Ứng dụng Stack - Queue

Stack

- Đảo mảng
- Đảo chuỗi
- Chuyển đổi hệ cơ số
- Bracket Matching
- Balancing Act

Queue

- Palindromes
- Demerging
- Tính giá trị của biểu thức

Đảo mảng

Đảo mảng 1 chiều A gồm n phần tử thuộc kiểu int

A[10]={0,1,2,3,4,5,6,7,8,9} → A[10]={9,8,7,6,5,4,3,2,1,0}

1. Khởi tạo một Stack rỗng, có kiểu số.

2. Với n phần tử của mảng, lần lượt đưa vào Stack thông qua hàm Push: **Push** a[i] into **Stack**.

3. Lần lượt lấy ra từ Stack n phần tử và đưa vào trở lại mảng ban đầu: **Pop** a item from **Stack** in to a[i].

4. Kết thúc giải thuật.

Đảo chuỗi

Đảo chuỗi S="Lan di hoc" → S="hoc di Lan"

1. Tạo một Stack rỗng, có kiểu chuỗi
2. Với mỗi từ mWord đọc được từ chuỗi S, đưa từ đó vào Stack:

   Push mWord into Stack.
3. Đọc từ Stack cho đến hết, thực hiện:

   Pop a word from **Stack** into mWord.
4. Kết thúc giải thuật.

Lưu ý: cần xây dựng hàm tách mWord từ chuỗi S

Chuyển đổi hệ cơ số

Chuyển đổi 1 số nguyên từ hệ thập phân sang hệ nhị phân

N(10)=8 → N(2)=1000

1. Khởi tạo một Stack rỗng.
2. Chuyển đổi số từ dạng thập phân sang nhị phân bằng phép **div** và **mod** cho 2.
3. Kết quả của phép mod được đưa vào **Stack**.
4. Đọc từ **Stack** cho đến hết, kết quả được nối với nhau để tạo thành chuỗi.
5. Kết thúc giải thuật.

Bracket Matching

Một biểu thức sử dụng dấu ngoặc (Bracket) đúng nếu:
- Với mỗi dấu ngoặc trái sẽ có 1 dấu ngoặc phải gần nhất khớp với nó;
- Với mỗi dấu ngoặc phải sẽ có 1 dấu ngoặc trái gần nhất (bên trái) khớp với nó;
- Một đoạn biểu thức nằm giữa một cặp dấu ngoặc trái, phải được gọi là sử dụng đúng dấu ngoặc;

s * (s – a) * (s – b) * (s – c)	→ Well
(– b + (b2 – 4*a*c)^0.5) / 2*a	→ Well
s * (s – a) * (s – b * (s – c)	→ ???
s * (s – a) * s – b) * (s – c)	→ ???
(– b + (b^2 – 4*a*c)^(0.5/ 2*a))	→ ???

Bracket Matching

1. Tạo một Stack rỗng (Stack chứa dấu ngoặc).

2. Với mỗi ký hiệu sym trong đoạn (từ trái sang phải) :

    2.1. Nếu **sym** là dấu **ngoặc trái**:

        2.1.1. Đưa sym vào Stack.

    2.2. Nếu **sym** là dấu **ngoặc phải**:

        2.2.1. Nếu **Stack** rỗng, return **false**.

        2.2.2. Ngược lại, lấy dấu **ngoặc trái** ở stack ra khỏi stack để triệt tiêu với dấu **ngoặc phải** của sym.

3. Kết thúc giải thuật, trả về **True** nếu **Stack** rỗng, hoặc **False** với các trường hợp khác.

Balancing Act

- Với phương pháp trước mới chỉ đảm bảo được khớp dấu ngoặc '**(**' và '**)**'**.**
- Trong thực tế, c.n có nhiều dấu ngoặc khác cần khớp như: '**(**' và '**)**'**;** '**[**' và '**]**'**;** '**{**' và '**}**'**.**
- Như vậy, trong giải thuật trên, cần lưu ý thêm quá trình **push** và **pop** vào **Stack**:
  - Nếu quá trình push vào **Stack** có dạng: **(, [, {.**
  - Quá trình **pop** từ **Stack** cần đúng thứ tự: **},],).**
  - Nếu đã duyệt xong biểu thức và bStack rỗng → return **True**, ngược lại, return **False**.

Ứng dụng Queue

Palindromes

- Khái niệm: Một chuỗi được gọi là Palindrome nếu như đọc xuôi giống đọc ngược.
- Bài toán: Cho trước một chuỗi, kiểm tra xem chuỗi đó có phải là chuỗi palindrome hay không?
- Ví dụ về chuỗi palindrome: **Able was I ere I saw Elba**

Giải pháp:

- Để tránh ảnh hưởng tới chuỗi ban đầu, đọc chuỗi nói trên vào stack và queue.
- So sánh từng phần tử của stack và queue, nếu giống nhau từng cặp thì đó là chuỗi Palindrome, ngược lại thì chuỗi trên không phải là chuỗi Palindrome.

Demerging

- Tổ chức dữ liệu hợp lý **- Demerging**

Bài toán: Xem xét bài toán sau:

- Giả sử, với một hệ thống quản lý nhân sự. Các bản ghi được lưu trên file.
- Mỗi bản ghi gồm các trường: Họ tên, giới tính, ngày tháng năm sinh, ...
- Dữ liệu trên đã được sắp theo ngày tháng năm sinh.
- Cần tổ chức lại dữ liệu sao cho nữ được liệt kê trước nam nhưng vẫn giữ được tính đ. sắp theo ngày tháng năm sinh.

Ứng dụng Queue

Demerging

1. Tạo 2 queue rỗng, có tên lần lượt là NU và NAM.
2. Với mỗi bản ghi p, xem xét:
   1. Nếu p có giới tính nữ, đưa vào queue NU.
   2. Nếu p có giới tính nam, đưa vào queue NAM.
3. Xét queue NU, khi queue chưa rỗng:
   1. Lấy từng phần tử trong queue này.
   2. Ghi vào file output nào đó.
4. Xét queue NAM, khi queue chưa rỗng:
   1. Lấy từng phần tử trong queue này.
   2. Ghi tiếp vào file output trên.
5. Kết thúc giải thuật.

Chúc các em học tốt!