Cấu trúc dữ liệu cơ bản

Mảng động, danh sách liên kết đơn, danh sách liên kết đôi, ngăn xếp, hàng đợi

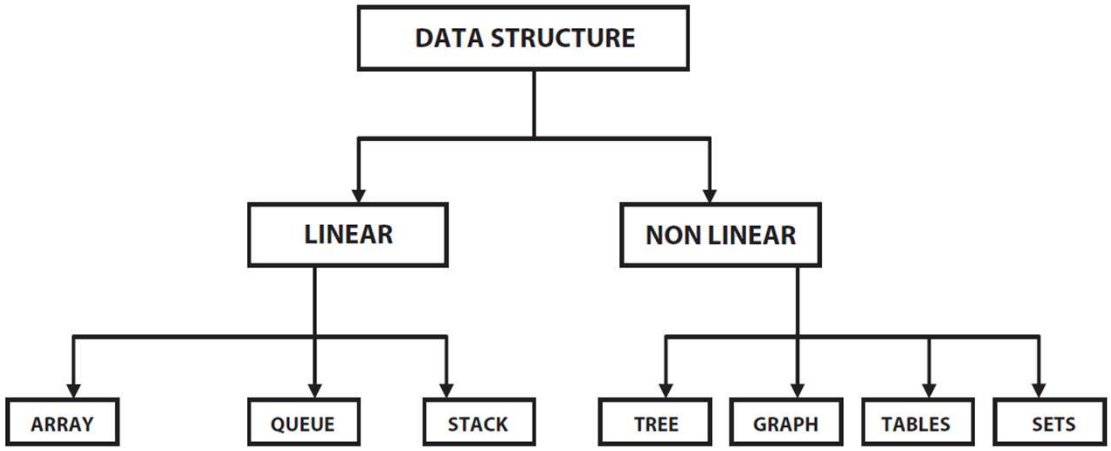
- Real-Life Examples of Data Structures In each of the following examples, please choose the best data structure(s). Options are: Array, Linked Lists, Stack, Queues, Trees, Graphs, Sets, Hash Tables. Note that there may not be one clear answer.
- 1. You have to store social network “feeds”. You do not know the size, and things may need to be dynamically added.
- 2. You need to store undo/redo operations in a word processor.
- 3. You need to evaluate an expression (i.e., parse).
- 4. You need to store the friendship information on a social networking site. I.e., who is friends with who.
- 5. You need to store an image (1000 by 1000 pixels) as a bitmap.
- 6. To implement printer spooler so that jobs can be printed in the order of their arrival.
- 7. To implement back functionality in the internet browser.
- 8. To store the possible moves in a chess game.
- 9. To store a set of fixed key words which are referenced very frequently.
- 10. To store the customer order information in a drive-in burger place. (Customers keep on coming and they have to get their correct food at the payment/food collection window.)
- 11. To store the genealogy information of biological species.

Nội dung

- Mảng động
- Danh sách liên kết đơn
- Danh sách liên kết đôi
- Danh sách tuyến tính
- Ngăn xếp – stack
- Hàng đợi – Queue

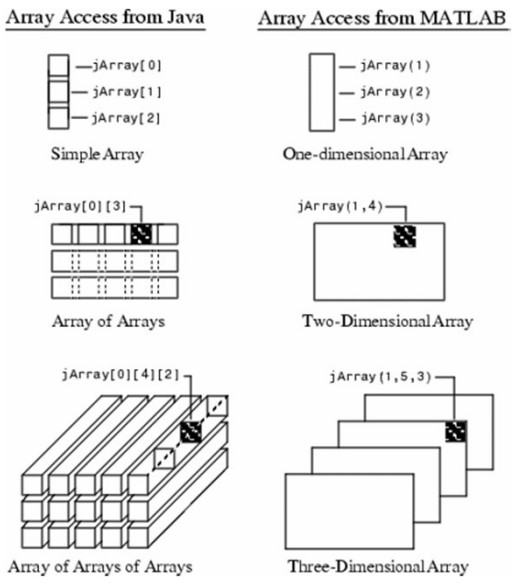
Cấu trúc dữ liệu

- Mô tả cách lưu trữ dữ liệu của bài toán vào trong máy tính
- Ảnh hưởng tới hiệu quả của thuật toán
- Các thao tác chính với một CTDL là
  - Duyệt
  - Tìm kiếm
  - Thêm phần tử
  - Xóa phần tử
  - Sắp xếp
  - Trộn
  - Copy - Clone
  - …



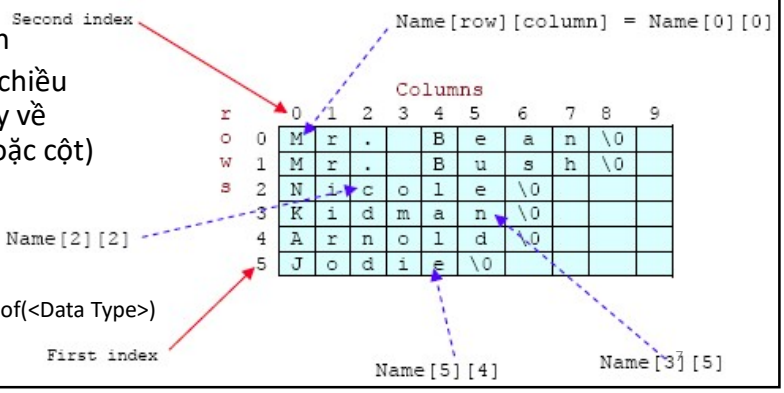
- Kiểu dữ liệu tuyến tính và Phi tuyến tính

Array – Mảng



Mảng

- **Mảng** - Array: là cấu trúc dữ liệu được cấp phát liên tục (liên tiếp) cơ bản
  - gồm các bản ghi có kiểu giống nhau, có kích thước cố định.
  - Mỗi phần tử được xác định bởi chỉ số (địa chỉ), là vị trí tương đối so với địa chỉ phần tử đầu mảng
  - Tên mảng = Hằng con trỏ trỏ tới địa chỉ phần tử đầu tiên
  - Trong máy tính chỉ có mảng 1 chiều mảng nhiều chiều sẽ được quy về mảng 1 chiều (ưu tiên hàng hoặc cột)

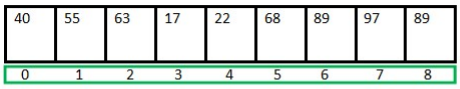


r/o/w/s	0	1	2	3	4	5	6	7	8	9
0	M	r	.		B	e	a	n	\0	
1	M	r	.		B	u	s	h	\0	
2	N	i	c	o	l	e	\0			
3	K	i	d	m	a	n	\0			
4	A	r	n	o	l	d	\0			
5	J	o	d	i	e	\0				

&Name[0][0] = Name
&Name[0][4] = Name + 4 * sizeof(<Data Type>)
&Name[i][j] = Name + (i*<column size> + j) * sizeof(<Data Type>)

Mảng

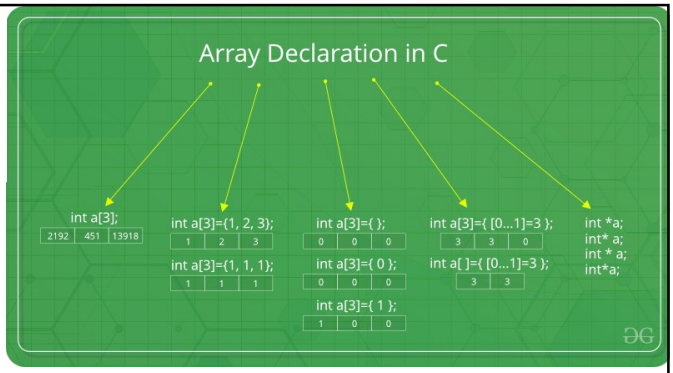
- **Ưu điểm** của mảng:
  - **Truy cập phần tử với thời gian hằng số $O(1)$**: vì thông qua chỉ số của phần tử ta có thể truy cập trực tiếp vào ô nhớ chứa phần tử.
  - **Sử dụng bộ nhớ hiệu quả**: chỉ dùng bộ nhớ để chứa dữ liệu nguyên bản, không lãng phí bộ nhớ để lưu thêm các thông tin khác.
  - **Tính cục bộ về bộ nhớ**: các phần tử nằm liên tục trong 1 vùng bộ nhớ, duyệt qua các phần tử trong mảng rất dễ dàng và nhanh chóng.
  - Các phần tử đặt dưới 1 tên chung nên dễ quản lý
- **Nhược điểm**:
  - không thể thay đổi kích thước của mảng khi chương trình đang thực hiện.
  - Các thao tác thêm/xóa phần tử mà dẫn đến phải dịch phần tử sẽ có chi phí lớn



Array Length = 9
First Index = 0
Last Index = 8

- Trong C/C++
  - Chỉ số bắt đầu từ 0
  - Có nhiều cách khai báo và khởi tạo mảng (chú ý phiên bản của C/C++)
  - KHÔNG check truy cập vượt ngoài phạm vi khai báo (các trình biên dịch có thể đưa ra cảnh báo với TH này)
  - KHÔNG check nếu khởi tạo quá số lượng

```
int arr[2] = { 10, 20, 30, 40, 50 };
```




```
int arr[2];

cout << arr[3] << " ";
cout << arr[-2] << " ";
```


https://www.geeksforgeeks.org/arrays-in-c-cpp/

Mảng trong C/C++

- Các phần tử sắp liên tiếp trong bộ nhớ

```
int arr[5], i;
cout << "Size of integer in this compiler is "
     << sizeof(int) << "\n";
for (i = 0; i < 5; i++)
    // The use of '&' before a variable name, yields
    // address of variable.
    cout << "Address arr[" << i << "] is " << &arr[i] << "\n";
```

```
Size of integer in this compiler is 4
Address arr[0] is 0x7ffe75c32210
Address arr[1] is 0x7ffe75c32214
Address arr[2] is 0x7ffe75c32218
Address arr[3] is 0x7ffe75c3221c
Address arr[4] is 0x7ffe75c32220
```

- Duyệt mảng

```
int arr[6]={11,12,13,14,15,16};
// Way -1
for(int i=0;i<6;i++)
    cout<<arr[i]<<" ";

cout<<endl;
  // Way 2
cout<<"By Other Method:"<<endl;
for(int i=0;i<6;i++)
    cout<<i[arr]<<" ";

cout<<endl;
```

```
11 12 13 14 15 16
By Other Method:
11 12 13 14 15 16
```

Mảng trong C/C++

- Mảng và con trỏ
  - Tên mảng = hằng con trỏ, trỏ tới ô nhớ đầu tiên cấp phát cho mảng (địa chỉ phần tử đầu tiên)
  - Không thể thay đổi địa chỉ mảng sau khi đã khai báo (KHÔNG thể gán 2 mảng trực tiếp)
  - Có thể dùng biến con trỏ để truy cập các phần tử trong mảng
  - Toán tử ++ và -- với con trỏ trỏ đến mảng để truy cập tới phần tử cách phần tử hiện tại 1 phần tử (về sau hoặc ở ngay trước)

```
int arr[] = { 10, 20, 30, 40, 50, 60 };
int* ptr = arr;
cout << "arr[2] = " << arr[2] << "\n";
cout << "*(arr + 2) = " << *(arr + 2) << "\n";
cout << "ptr[2] = " << ptr[2] << "\n";
cout << "*(ptr + 2) = " << *(ptr + 2) << "\n";
```

```
arr[2] = 30
*(arr + 2) = 30
ptr[2] = 30
*(ptr + 2) = 30
```

- Vector trong C++
  - Có trong STL của C++
  - Không cần chỉ ra trước số lượng phần tử tối đa (tự điều chỉnh theo nhu cầu)
  - Hỗ trợ sẵn một số hàm thêm, xóa và tìm kiếm
  - Thời gian thêm/xóa KHÔNG còn là hằng số như trong mảng thường (VD. thêm cuối)

Mảng trong C/C++

- Trong C luôn phải chỉ ra kích thước tối đa khi khai báo mảng, vậy có cách nào khắc phục khi
  - Không biết trước số lượng phần tử tối đa
  - Muốn tối ưu bộ nhớ, tránh lãng phí (các phần tử khai báo mà không dùng đến)
- Mảng cấp phát động nhiều lần(mảng với kích thước biến đổi)
  - Hàm cấp phát động trong C: malloc, calloc, relloc, và free
  - Ban đầu cấp phát 1 mảng nhỏ (VD. MAX_SIZE = 10 phần tử)
  - Tùy theo nhu cầu, nếu cần chứa phần tử > kích thước tối đa hiện tại → tạo mảng mới với kích thước gấp đôi mảng cũ (VD. MAX_SIZE = 2 * MAX_SIZE). Copy các phần tử mảng cũ vào nửa đầu mảng mới. VD. Java thường tăng thêm 50%, C++ tăng thêm 100%
  - Nếu số lượng phần tử thực sự trong mảng < ½ MAX_SIZE, tiến hành điều chỉnh co mảng với kích thước mảng mới MAX_SIZE = ½ MAX_SIZE để tránh lãng phí bộ nhớ (tùy vào NNLT)
  - Hệ số co giãn mảng – Load Factor thường chọn là 0.75, 1 (tùy NNLT có thể có hoặc không)

Mảng động với kích thước biến đổi

- Hệ số nạp $\lambda = \frac{n}{MAX\_SIZE}$
- Từ mảng 1 phần tử tới n phần tử, số lần phải thay đổi kích thước là $\log n$
- Số phần tử phải di chuyển

$$M = \sum_{i=1}^{\log n} i * \frac{n}{2^i} = n * \sum_{i=1}^{\log n} \frac{i}{2^i} < n * \sum_{i=1}^{\infty} \frac{i}{2^i} = 2n$$

Thời gian để duy trì mảng chỉ là $O(n)$

- **Nhược điểm**: một số thời gian thực hiện một số thao tác không còn đúng là hằng số nữa

Mảng trong Java

- Mảng trong Java
  - Luôn được cấp phát động
  - Là kiểu object nên có sẵn 1 số hàm hỗ trợ. VD. length
  - Kích thước mảng bị giới hạn bởi giá trị int hoặc short int (<4G)
  - Kiểu phần tử có thể là kiểu cơ sở hoặc object
  - Trong JAVA luôn có check truy cập ngoài phạm vi của mảng

```
int intArray[];    //declaring array
intArray = new int[20];  // allocating memory to array
```

OR

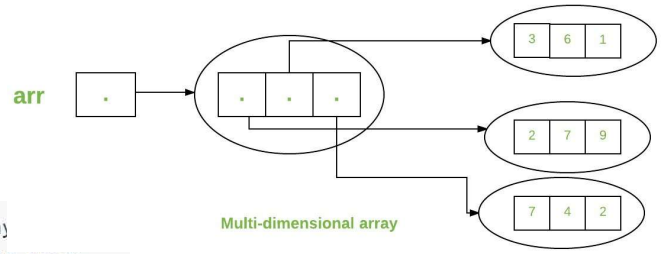
```
int[] intArray = new int[20]; // combining both statements in one
```

```
// both are valid declarations
int intArray[];
or int[] intArray;
```

```
public static void main (String[] args)
{
    int[] arr = new int[2];
    arr[0] = 10;
    arr[1] = 20;

    for (int i = 0; i <= arr.length; i++)
        System.out.println(arr[i]);
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
```

Mảng trong Java

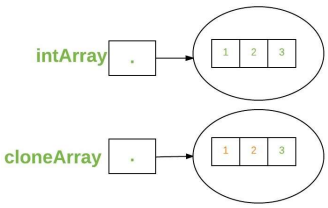


- Mảng nhiều chiều

```
int[][] intArray = new int[10][20]; //a 2D array
int[][][] intArray = new int[10][20][10]; //a 3D array
```

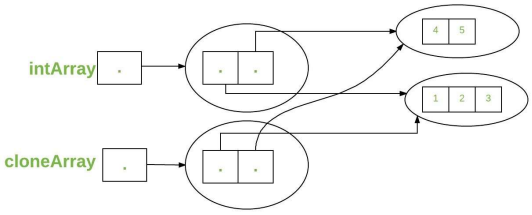
Clone Array: Deep copy VS Shallow Copy

```
int intArray[] = {1,2,3};

int cloneArray[] = intArray.clone();
```



Deep Copy is created for one-dimensional array by clone() method

```
int intArray[][] = {{1,2,3},{4,5}};

int cloneArray[][] = intArray.clone();
```

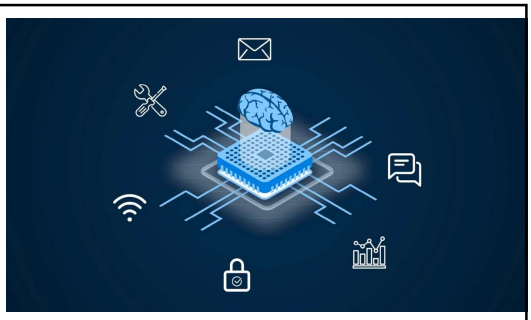


Shallow Copy is created for multi-dimensional array by clone() method

Mảng động dynamic array

- Lưu trữ dữ liệu có kích thước biến đổi
  - List/ArrayList
  - Dữ liệu trung gian trong thuật toán sắp xếp
  - Bộ nhớ đệm – buffer khi truyền tải dữ liệu
  - Xử lý xâu ký tự có độ dài không cố định
  - Cài đặt stack – queue
  - Bảng băm – HashTable
  - Lưu trữ ảnh, video với độ phân giải khác nhau

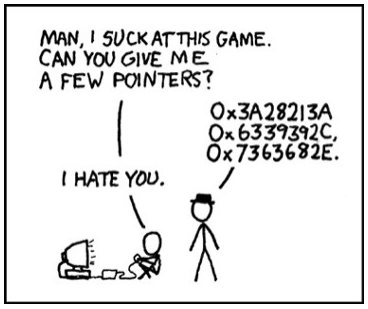
Mảng động dynamic array



- Quản lý bộ nhớ trong hệ thống nhúng, IoT
  - Kích thước bộ nhớ hạn chế, cấp phát tĩnh sẽ gây lãng phí
  - Kích thước dữ liệu không cố định (VD. Số lượng mẫu cảm biến tùy vào từng thời điểm)
  - Kích thước các gói tin truyền không cố định
  - Quản lý bộ đệm nhận dữ liệu
  - Phân vùng bộ nhớ động tại thời điểm chạy

Mảng - Array

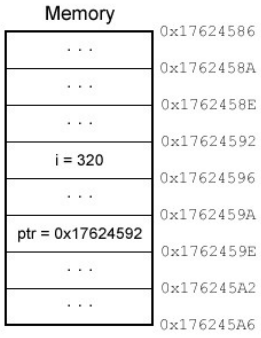
- VD 1. Viết chương trình xoay các phần tử của mảng đi k vị trí
- VD 2. Tìm phần tử trong mảng A có giá trị i*A[i] là lớn nhất
- VD 3. Viết chương trình sắp xếp lại mảng sao cho các phần tử âm ở đầu dãy và phần tử dương ở cuối dãy (không cần đúng thứ tự)
- VD 4. Cho 1 xâu chỉ chứa các ký tự là chữ số, hãy hoán đổi các ký tự trong xâu sao cho thu được biểu diễn của số có giá trị lớn nhất
- VD 5. Hãy viết chương trình xáo trộn mảng theo thứ tự ngẫu nhiên
- VD 6. Cho mảng A chứa n số nguyên, hãy tìm và in ra trung vị (median) của mảng
- VD 7. Cho mảng số thực A chứa n phần tử, tìm 2 số có tổng nhỏ nhất
- VD 8. Cho dãy chứa n phần tử, tìm dãy con độ dài k có giá trị trung bình nhỏ nhất
- VD 9. cho mảng n phần tử phân biệt và giá trị k, tìm xem có 2 phần tử trong mảng tổng bằng k



Cấu trúc liên kết

- Con trỏ và cấu trúc liên kết
- Danh sách liên kết đơn
- Các dạng khác của danh sách liên kết

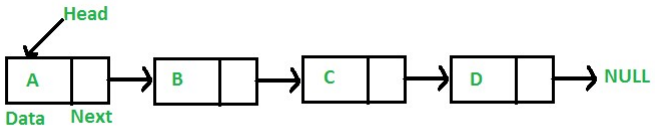
Con trỏ và cấu trúc liên kết



- **Con trỏ** lưu trữ địa chỉ của một vị trí trong bộ nhớ.
  VD. Visiting card có thể xem như con trỏ trỏ đến nơi làm việc của một người nào đó.
- Trong cấu trúc liên kết con trỏ được dùng để liên kết giữa các phần tử.
- Trong C/C++ :
  - ***ptr** chỉ **ptr** là một biến con trỏ
  - **&i** chỉ địa chỉ của biến **i** trong bộ nhớ (địa chỉ ô nhớ đầu tiên)
  - Con trỏ khi mới khởi tạo nhận giá trị **NULL** - con trỏ chưa được gán giá trị (không trỏ vào đâu cả)
  - Kiểu của con trỏ để xác định phạm vi bộ nhớ có thể truy cập
  - Các con trỏ có cùng kích thước trên 1 platform

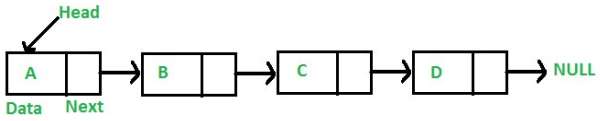
```
int i = 320;
int* ptr = &i;
```

Cấu trúc liên kết



- Cấu trúc liên kết
  - Các phần tử nằm rải rác trong bộ nhớ
  - Phần tử trước sẽ lưu lại địa chỉ phần tử tiếp theo (qua con trỏ)
  - Các phần tử chỉ có thể truy cập một cách tuần tự
  - Việc thêm/xóa phần tử đơn giản hơn so với cấu trúc liên tiếp (mảng)
  - Mỗi phần tử cần thêm ít nhất 1 con trỏ để duy trì liên kết trong cấu trúc (con trỏ được coi là bộ nhớ lãng phí dưới góc độ người dùng)
- Một số đại diện cấu trúc liên kết
  - Danh sách liên kết: danh sách liên kết đơn, danh sách liên kết đôi,…
  - Cây: cây nhị phân tổng quát, cây AVL, R-B tree, kD tree, prefix tree…
  - Đồ thị lưu bằng ma trận kề

Danh sách liên kết đơn



- Danh sách liên kết đơn
  - Là cấu trúc liên kết đơn giản nhất
  - Mỗi phần tử chỉ có thêm 1 con trỏ để lưu địa chỉ phần tử kế tiếp
- Ưu điểm so với mảng
  - Không cần khai báo trước số lượng tối đa
  - Dùng bao nhiêu, cấp phát đủ
  - Thêm/xóa các phần tử dễ dàng, không cần dịch (chỉ cần thay đổi giá trị con trỏ)
- Nhược điểm
  - Chỉ có thể truy cập phần tử một cách tuần tự
  - Mỗi phần tử tốn thêm 1 con trỏ

Cấu trúc liên kết – linked list

- Khai báo danh sách liên kết đơn (singly-linked list) :
  - Có 1 hay nhiều trường dữ liệu (item) chứa dữ liệu cần lưu trữ
  - Có ít nhất 1 con trỏ trỏ đến nút tiếp theo (next) → cần nhiều bộ nhớ hơn cấu trúc liên tục.
  - Cần 1 con trỏ lưu địa chỉ phần tử bắt đầu của cấu trúc.

```
struct Node {
    int data;
    struct Node* next;
};
```

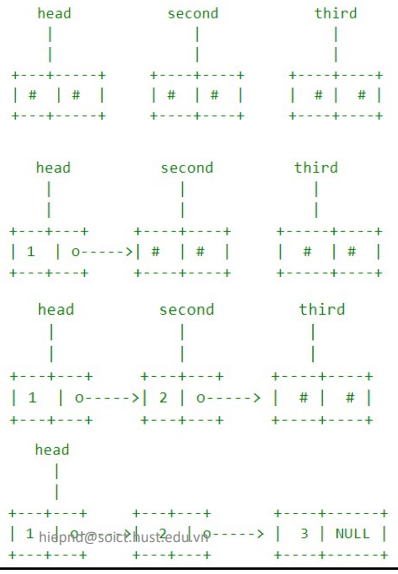
```
class Node {
public:
    int data;
    Node* next;
};
```

```
static class Node {
    int data;
    Node next;
    Node(int d)
    {
        data = d;
        next = null;
    } // Constructor
}
```

https://www.geeksforgeeks.org/linked-list-set-1-introduction/

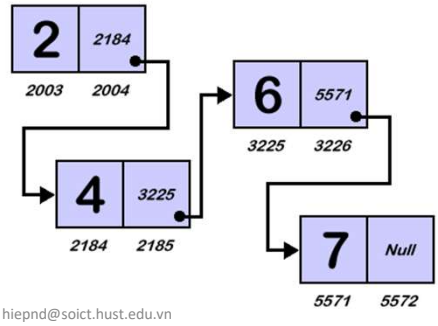
Danh sách liên kết đơn - Singly-linked list

```
struct Node* head = NULL;
struct Node* second = NULL;
struct Node* third = NULL;

// allocate 3 nodes in the heap
head = (struct Node*)malloc(sizeof(struct Node));
second = (struct Node*)malloc(sizeof(struct Node));
third = (struct Node*)malloc(sizeof(struct Node));

head->data = 1; // assign data in first node
head->next = second; // Link first node with

second->data = 2;
second->next = third;

third->data = 3; // assign data to third node
third->next = NULL;
```



Danh sách liên kết đơn

- Một số thao tác thông dụng trên danh sách liên kết đơn
  - Duyệt danh sách (in danh sách, đếm số phần tử)
  - Chèn một phần tử mới
  - Xóa một phần tử
  - Tìm kiếm một phần tử



Danh sách liên kết đơn

- Duyệt danh sách

```
void printList(struct Node* head)
{
    while (head != NULL) {
        printf(" %d ", head ->data);
        head = head ->next;
    }
}
```

```
int size(Node* head)
{
    int count=0;
    while (head != NULL) {
        count++;
        head = head->next;
    }
    return count;
}
```

```
class LinkedList {

    Node head; // head of list
    static class Node {
        int data;
        Node next;
        Node(int d)
        {
            this.data = d;
            next = null;
        } // Constructor
    }
    public void printList()
    {
        Node n = head;
        while (n != null) {
            System.out.print(n.data + " ");
            n = n.next;
        }
    }
    public static void main(String[] args)
    {
        /* Start with the empty list. */
        LinkedList llist = new LinkedList();


        llist.head = new Node(1);
        Node second = new Node(2);
        Node third = new Node(3);


        llist.head.next = second; // Link first node with the second node
        second.next = third; // Link second node with the third node


        llist.printList();
    }
}
```

Duyệt danh sách

- Tìm kiếm
  - Tìm xem khóa key có xuất hiện trong danh sách
  - Đếm số lượng phần tử có giá trị bằng giá trị cho trước

$$T(n) = O(n)$$

```
/* Checks whether the value x is present in linked list */
bool search(struct Node* head, int key)
{
    struct Node* current = head;  // Initialize current
    while (current != NULL)
    {
        if (current->key == key)
            return true;
        current = current->next;
    }
    return false;
}
```

Bài tập thêm

- Tìm phần tử ngay trước phần tử hiện tại?
- Tìm phần tử ở vị trí thứ k trong dãy
- Tìm phần tử ở giữa danh sách
- Đếm số lần xuất hiện của giá trị key

```
/* Checks whether the value x is present in linked list */
struct Node* search(struct Node* head, int key)
{
    struct Node* current = head;  // Initialize current
    while (current != NULL)
    {
        if (current->key == key)
            return current;
        current = current->next;
    }
    return NULL;
}
```


Danh sách liên kết đơn

- Thêm một phần tử mới
  - Thêm vào đầu tiên - push
  - Thêm vào giữa (sau 1 vị trí nào đó) – insertAfter
  - Thêm vào cuối - append

Dùng khai báo minh họa 1 nút của DSLK đơn chỉ với 1 trường kiểu int

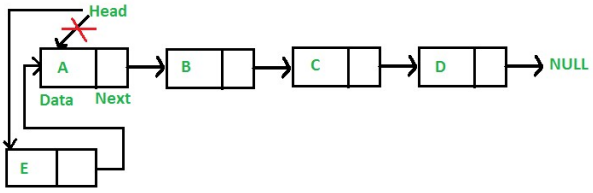
Trong thực tế KHÔNG nên khai báo DSLK đơn nếu dữ liệu lưu trữ chỉ là 1 trường int, TẠI SAO?

```
struct Node
{
  int data;
  struct Node *next;
};
```

Danh sách liên kết đơn

```c
// Hàm tạo một nút mới
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Lỗi cấp phát bộ nhớ!\n");
        return NULL;
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```


Thêm phần tử mới



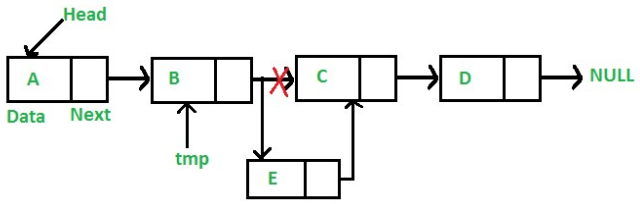
- **Thêm vào đầu tiên – push**
  - Danh sách ban đầu đang có A→B→C→D, chèn thêm phần tử E vào đầu để được danh sách mới E→A→B→C→D

Các bước cần làm gồm
- Cấp phát động lưu trữ phần tử mới
- Gán giá trị phần tử mới
- Cập nhật vị trí phần tử mới

Hàm push sẽ làm thay đổi giá trị con trỏ đầu danh sách , vì vậy trong hàm cần truyền vào là **head_ref

```c
void push(struct Node** head_ref, int new_data)
{
    /* 1. cấp phát động nút mới */
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));

    /* 2. Cập nhật dữ liệu nút mới  */
    new_node->data = new_data;

    /* 3. Biến nút mới thành đầu của dãy hiện tại*/
    new_node->next = (*head_ref);

    /* 4. Cập nhật lại giá trị của con trỏ đầu dãy */
    (*head_ref) = new_node;
}
```

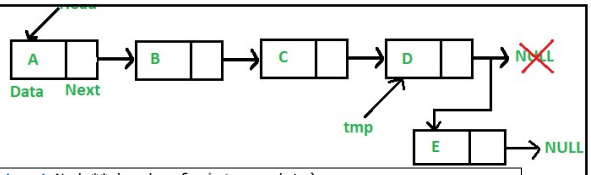
Thêm phần tử mới



- **Thêm vào sau 1 nút cho trước – insertAfter**
  - Danh sách ban đầu A→B→C→D, Chèn E vào sau phần tử B để được A→B→E→C→D
  - Cần thêm con trỏ trỏ vào vị trí trước chèn prev_node
  - Trong hàm KHÔNG làm thay đổi giá trị của con trỏ prev_node nên không cần **

```
void insertAfter(struct Node* prev_node, int new_data)
{
    /*1. kiểm tra vị trí chèn prev_node có phải NULL */
    if (prev_node == NULL) return;

    /* 2. Cấp phát động nút mới */
    struct Node* new_node =(struct Node*) malloc(sizeof(struct Node));


    /* 3. Gán giá trị */
    new_node->data = new_data;


    /* 4. Cập nhật phần tử kế tiếp của phần tử mới */
    new_node->next = prev_node->next;


    /* 5. Gắn phần tử mới vào sau prev_node */
    prev_node->next = new_node;
}
```


Thêm phần tử mới



- **Thêm vào cuối dãy – append**
  - Dãy đang là A→B→C→D, chèn thêm E vào cuối dãy để được dãy mới A→B→C→D→E
  - Nếu dãy rỗng → thêm vào đầu = cuối
  - Ngược lại, phải duyệt tuần tự tìm cuối dãy
  - Nếu dãy rỗng ta sẽ phải cập nhật lại con trỏ đầu dãy → cần truyền vào **

$$T(n) = ?$$

```
void append(struct Node** head_ref, int new_data)
{
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));


    struct Node *last = *head_ref;

    new_node->data  = new_data;


    /* Nút mới là cuối dãy nên next sẽ trỏ vào NULL*/
    new_node->next = NULL;


    /* Nếu danh sách đang rỗng → cập nhật lại đầu */
    if (*head_ref == NULL)
    {
       *head_ref = new_node;
       return;
    }

    /* Ngược lại, tìm cuối dãy*/
    while (last->next != NULL)
        last = last->next;

    /* Gắn nút mới là cuối dãy*/
    last->next = new_node;
    return;
}
```

Thêm phần tử mới

- Thử nghiệm các thao tác thêm
  - Printf và cout với C/C++

```
void printList(struct Node *node)
{
  while (node != NULL)
  {
     printf(" %d ", node->data);
     node = node->next;
  }
}
```

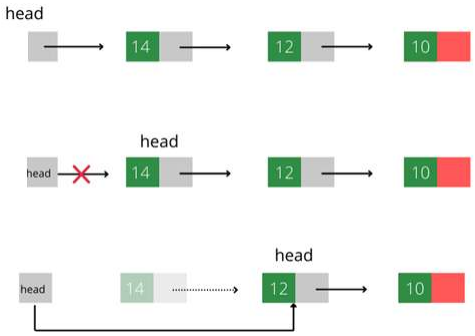
```
int main()
{
    /* Start with the empty list */
    Node* head = NULL;

    // Insert 6. So linked list becomes 6->NULL
    append(&head, 6);

    // Insert 7 at the beginning.
    // So linked list becomes 7->6->NULL
    push(&head, 7);

    // Insert 1 at the beginning.
    // So linked list becomes 1->7->6->NULL
    push(&head, 1);

    // Insert 4 at the end. So
    // linked list becomes 1->7->6->4->NULL
    append(&head, 4);

    // Insert 8, after 7. So linked
    // list becomes 1->7->8->6->4->NULL
    insertAfter(head->next, 8);

    cout<<"Created Linked list is: ";
    printList(head);

    return 0;
}
```


Danh sách liên kết đơn

- Xóa phần tử - delete
  - Xóa phần tử ở đầu
  - Xóa phần tử ở vị trí cho trước
  - Xóa phần tử ở cuối dãy
  - Xóa phần tử có khóa bằng key
  - Xóa toàn bộ dãy → giải phóng bộ nhớ

Xóa phần tử

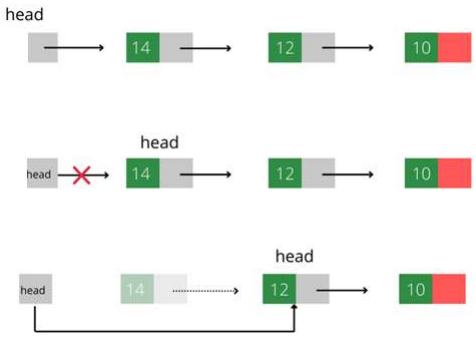
- Xóa phần tử ở đầu - removeFirst
  - Cập nhật lại con trỏ head
  - Gọi lệnh giải phóng bộ nhớ




```
void removeFirst(struct Node** head_ref)
{
    /* 1. Danh sách ban đầu rỗng thì không làm gì cả*/
    if(*head_ref==NULL) return;

    /* 2. Ghi nhận địa chỉ nút đầu cũ */
    struct Node* tmp = *head_ref;

    /* 3. Cập nhật lại đầu danh sách trỏ sang phần tử tiếp */
    *head_ref = (*head_ref)->next;

    /* 4. Giải phóng phần tử đầu cũ */
    free(tmp);
}
```


Xóa phần tử

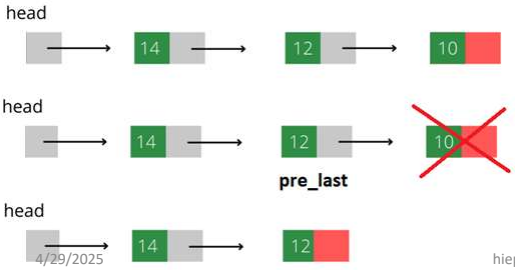
- Lấy phần tử ở đầu - pop
  - Cập nhật lại con trỏ head
  - Trả về phần tử




```
struct Node* pop(struct Node** head_ref)
{
    /* 1. Danh sách ban đầu rỗng thì không làm gì cả*/
    if(*head_ref==NULL) return NULL;

    /* 2. Ghi nhận địa chỉ nút đầu cũ */
    struct Node* tmp = *head_ref;

    /* 3. Cập nhật lại đầu danh sách trỏ sang phần tử tiếp */
    *head_ref = (*head_ref)->next;

    /* 4. Trả về*/
    tmp->next = NULL;
    return tmp;
}
```

Xóa phần tử

- **Xóa phần tử ở cuối**
  - Nếu danh sách rỗng → không làm gì cả
  - Danh sách có 1 phần tử ->Xóa đầu
  - Ngược lại → phải tìm phần tử gần cuối – pre_last
  - Xóa phần tử cuối từ pre-last, và cập nhật lại phần tử cuối

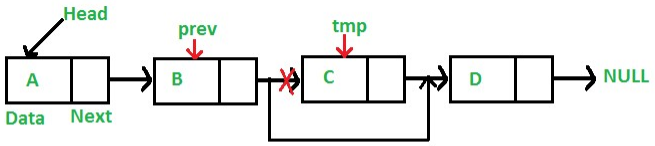



```
void removeLast(struct Node** head_ref)
{
   /* 1. Danh sách ban đầu rỗng thì không làm gì cả*/
   if(*head_ref==NULL) return;

   /* 2. Danh sách chỉ có 1 phần tử*/
   if((*head_ref)->next==NULL)
   {
      free(*head_ref);
      *head_ref = NULL;
      return;
   }
   /* 3. Tìm phần tử trước phần tử cuối cùng */
   struct Node* pre_last = *head_ref;
   while(pre_last->next->next !=NULL)
      pre_last = pre_last->next;

   /* 4. Xóa phần tử cuối cùng */
   free(pre_last->next);

   /* 5. Cập nhật lại con trỏ next của phần tử cuối dãy mới */
   pre_last->next = NULL;
}
```

Tại sao cần ** head_ref

$T(n) = ?$

Xóa phần tử

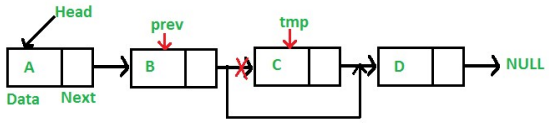


- Xóa phần tử ở vị trí cho trước
  - Phần tử trước vị trí cần xóa trỏ bởi con trỏ prev
  - Không rơi vào trường hợp đầu hoặc cuối
  - Phần tử cần xóa trỏ bởi tmp
- Nếu phần tử cần xóa là
  - Đầu/Cuối thì xử lý thế nào?
  - Xóa phần tử có giá trị bằng khóa K
  - Xóa toàn bộ dãy

```
void deleteNode(struct Node* prev)
{
   struct Node * temp = prev->next;

   // Unlink the node from linked list
   prev->next = temp->next;

   free(temp); // Free memory
}
```

Xóa phần tử

- Xóa phần tử có giá trị bằng khóa key
  - Key có thể là đầu
  - Có thể là giữa/ cuối
  - Luôn phải free bộ nhớ



Tại sao cần **head_ref ???

```
void deleteNode(struct Node** head_ref, int key)
{
   // Luu lai dia chi phan tu dau
   struct Node * temp = *head_ref, * prev;

   // phan tu can xoa chinh la dau day
   if (temp != NULL && temp->data == key) {
      * head_ref = temp->next; // Changed head
      free(temp); // free old head
      return;
   }

   // Tìm phan tu khoa key, luu lai phan tu truoc
   // phan tu can xoa la tmp, se la 'prev->next'
   while (temp != NULL && temp->data != key) {
      prev = temp;
      temp = temp->next;
   }

   // Neu danh sach không co phan tu bang key
   if (temp == NULL)
      return;

   // Xoa phan tu temp
   prev->next = temp->next;

   free(temp); // Free memory
}
```


Xóa phần tử

- Test code

```
void printList(struct Node *node)
{
  while (node != NULL)
  {
     printf(" %d ", node->data);
     node = node->next;
  }
}
```

```
int main()
{
   node* head = NULL;
   // tao day 15→14→12→10
   push(head, 10);
   push(head, 12);
   push(head, 14);
   push(head, 15);

   // original list, in ra 15,14,12,10
   print(head);

   pop(&head); // xoa phan tu dau tien
   print(head); // in ra 14,12,10

   deleteNode(head, 10);
   print(head); // in ra 14,12

   removeLast(&head);
   print(head); //in ra 14

   return 0;
}
```

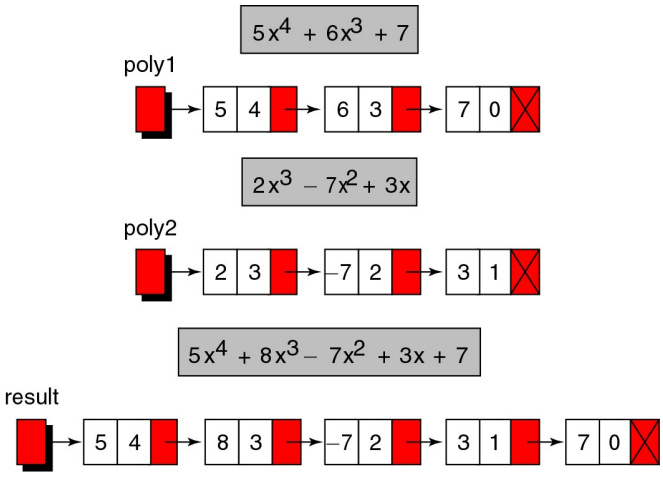
Danh sách liên kết đơn

- Bài tập thêm
  - Xóa toàn bộ dãy (cần giải phóng toàn bộ phần tử)
  - Xóa toàn bộ phần tử có khóa bằng key trong dãy
  - Xóa phần tử tại vị trí thứ i trong dãy (phần tử đầu tiên là vị trí 0)
  - Phát hiện vòng trong danh sách liên kết đơn
  - Loại bỏ các phần tử bị lặp trong danh sách liên kết đơn
  - Tìm giao của 2 danh sách liên kết đơn
  - Hoán đổi 2 phần tử ở vị trí i và j trong danh sách
  - Đảo ngược danh sách liên kết
  - Kiểm tra danh sách có phải đối xứng – palindrome
  - Copy danh sách liên kết đơn
  - Xóa phần tử hiện tại mà KHÔNG có địa chỉ phần tử đầu danh sách

https://www.geeksforgeeks.org/data-structures/linked-list

Danh sách liên kết đơn

- Bài toán biểu diễn đa thức



Danh sách liên kết đơn

- **typedef struct poly{**
  float heSo;
  float soMu;
  struct poly *nextNode;
  } POLY;

- **Các thao tác:**
  - Nhập đa thức
  - Hiển thị
  - Cộng
  - Trừ
  - Nhân
  - Tính giá trị đa thức
  - Chia
  - ….

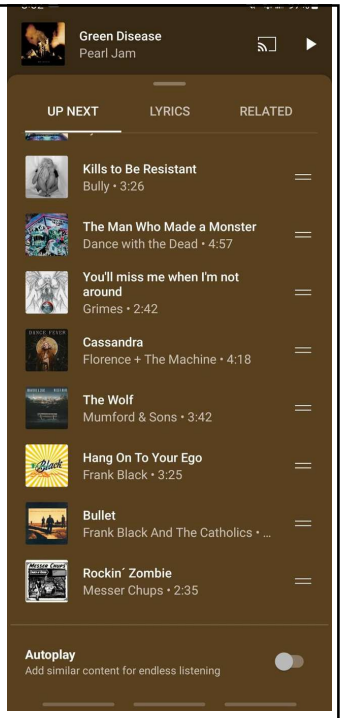
Danh sách liên kết đơn

- Biểu diễn playlist nhạc dùng danh sách liên kết đơn

```
struct Song {
char title[100];
char artist[100];
int length; // in seconds
char file_path[150];
struct Song* next;
};
```

Thao tác:
- Thêm bài hát mới
- Xóa 1 bài hát
- Play all playlist
- Play bài hát kế tiếp

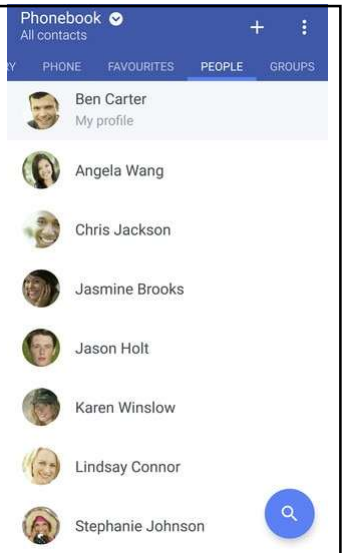
Play random hoặc shuffle?



Danh sách liên kết đơn

- Quản lý danh bạ liên lạc

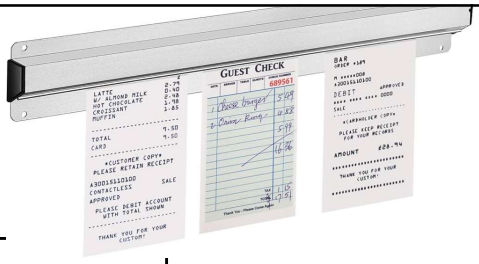
```
struct Contact {
char name[100];          // Tên liên lạc
char phone[15];          // Số điện thoại
char email[100];         // Địa chỉ email
struct Contact* next;
};
```



Thao tác:
- Thêm contact mới
- Xóa contact
- Tìm kiếm contact theo tên
- In ra toàn bộ danh sách contact

Vấn đề?
- Tìm kiếm tuần tự?

Danh sách liên kết đơn



- Quản lý danh sách đơn hàng

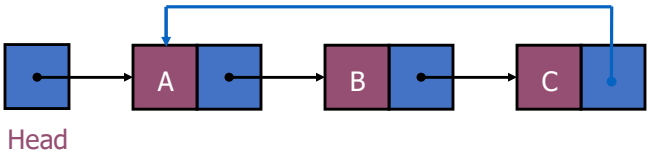
```
// Cấu trúc đơn hàng
struct Order {
    char orderID[20];
    char customerName[100]; //tên khách hàng hoặc mã
    char productList[200]; // danh sách sản phẩm
    int quantity; // tổng số lượng sản phẩm
    double totalAmount; // tổng tiền
    char status[20]; // trạng thái: pending, processing,
    finished
    struct Order* next;
};
```

Thao tác:
- Thêm 1 order
- Xóa order
- Cập nhật trạng thái order
- Tìm kiếm order

Danh sách liên kết

Một số dạng mở rộng khác của danh sách liên kết

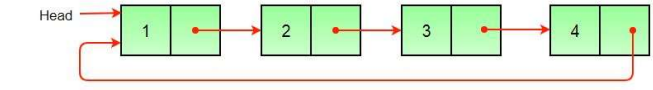
- **Danh sách liên kết đơn nối vòng**: Con trỏ của phần tử cuối trỏ về đầu danh sách

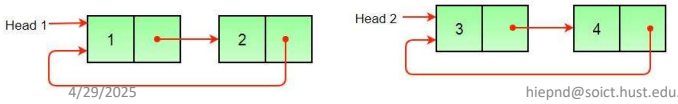


- Tác dụng:
  - có thể quay lại từ đầu khi đã ở cuối dãy
  - Kiểm tra ở cuối dãy : currentNode->next == head ?
  - Kiểm tra đang ở đầu dãy: currentNode == head

Danh sách liên kết đơn nối vòng

```
void printList(Node* head)
{
    Node* temp = head;

    // If linked list is not empty
    if (head != NULL) {

        // Print nodes till we reach first node again
        do {
            cout << temp->data << " ";
            temp = temp->next;
        } while (temp != head);
    }
}
```






```
void splitList(Node *head, Node **head1_ref,
                           Node **head2_ref)
{
    Node *slow_ptr = head;
    Node *fast_ptr = head;

    if(head == NULL)
        return;

    /* If there are odd nodes in the circular list then
       fast_ptr->next becomes head and for even nodes
       fast_ptr->next->next becomes head */
    while(fast_ptr->next != head &&
          fast_ptr->next->next != head)
    {
        fast_ptr = fast_ptr->next->next;
        slow_ptr = slow_ptr->next;
    }

    /* If there are even elements in list
       then move fast_ptr */
    if(fast_ptr->next->next == head)
        fast_ptr = fast_ptr->next;

    /* Set the head pointer of first half */
    *head1_ref = head;

    /* Set the head pointer of second half */
    if(head->next != head)
        *head2_ref = slow_ptr->next;

    /* Make second half circular */
    fast_ptr->next = slow_ptr->next;

    /* Make first half circular */
    slow_ptr->next = head;
}
```

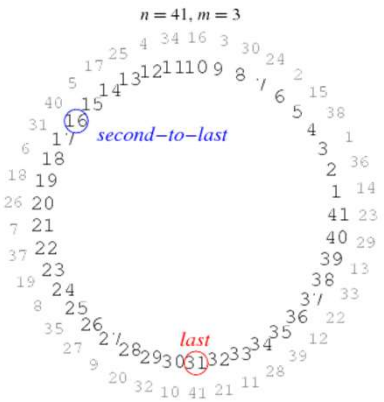
Danh sách liên kết đơn nối vòng

- Một số bài toán
  - Thêm phần tử vào danh sách liên kết đơn nối vòng
  - Xóa phần tử
  - Tìm phần tử giữa danh sách
  - Đếm số lượng phần tử trong danh sách
  - Bài toán vòng tròn Josephus
  - Chuyển danh sách liên kết đơn thường sang dạng nối vòng

Ứng dụng

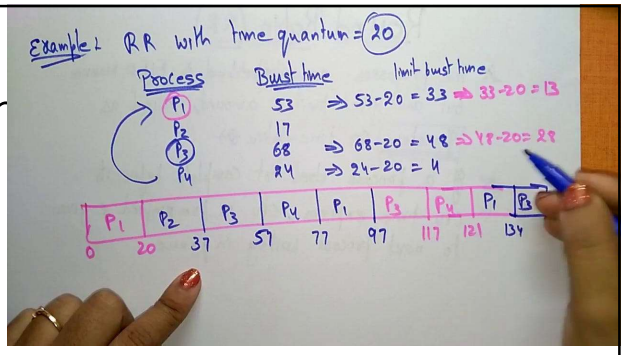
Ví dụ. Bài toán Josephus

- Có một nhóm gồm *n* người được xếp theo một vòng tròn. Từ một vị trí bất kỳ đếm theo chiều ngược chiều kim đồng hồ và loại ra người thứ *m* trong vòng. Sau mỗi lần loại lại bắt đầu đếm lại vào loại tiếp cho đến khi chỉ còn lại 1 người duy nhất.



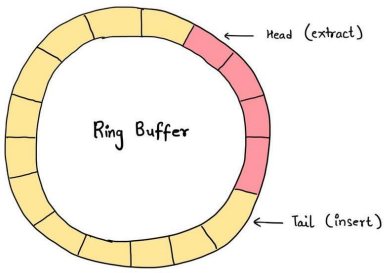
- **Cài đặt** : sử dụng danh sách liên kết đơn nối vòng

Danh sách liên kết đơn r



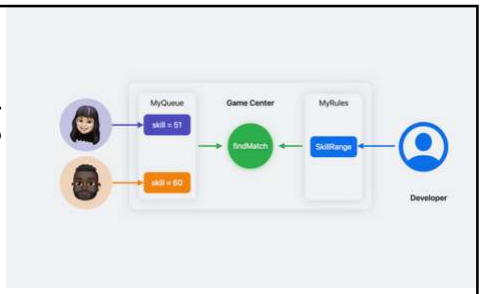
- Biểu diễn playlist
  - Dùng danh sách liên kết đơn nối vòng thay vì danh sách liên kết đơn thông thường
  - Repeat playlist dễ hơn
- Điều phối tiến trình trong HDH theo cơ chế Round-robin Scheduling
  - Thông tin mỗi tiến trình sẽ được lưu trong 1 node (ID tiến trình, trạng thái, thời gian thực hiện còn lại, v.v)
  - Mỗi tiến trình sẽ được cấp phát 1 chu kỳ *time slice* dùng CPU. Sau mỗi khoảng *time slice*, hệ điều hành sẽ thực hiện việc chuyển đổi (context switch) từ tiến trình hiện tại sang tiến trình tiếp theo trong danh sách.
  - Việc thêm/xóa tiến trình và xoay vòng dễ dàng hơn với danh sách liên kết đơn nối vòng

Danh sách liên kết đơn nối vòng



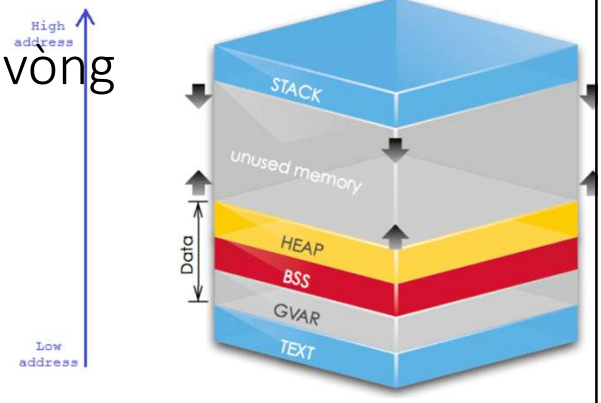
- cài đặt **Circular Buffer** (bộ đệm vòng)
  - Bộ nhớ đệm có kích thước cố định, khi đầy dữ liệu mới sẽ được ghi đè lên dữ liệu cũ nhất.
  - Được sử dụng trong các hệ thống có yêu cầu xử lý dữ liệu liên tục với tốc độ nhanh, ví dụ thiết bị đầu vào/ra, các luồng dữ liệu truyền thông,..
  - Dùng 2 con trỏ:
    - Tail: trỏ tới vị trí mà dữ liệu mới sẽ được thêm vào.
    - Head : trỏ tới vị trí của phần tử cũ nhất trong bộ đệm.
  - Khi thêm: Tail sẽ di chuyển tuần hoàn, nếu hết bộ nhớ sẽ ghi đè lên Head (sau đó Head cũng sẽ được di chuyển)
  - Khi đọc: Head được di chuyển sang node tiếp theo. Head di chuyển tới Tail → bộ đệm rỗng

Danh sách liên kết đơn nối vòng



- Cấu trúc hàng đợi trò chơi trực tuyến (Queue in Multiplayer Games)
  - Mỗi node là người chơi đợi ghép trận
  - Khi được ghép trận, người chơi sẽ được xóa khỏi hàng đợi
  - Ghép trận
    - PvP Dota 2, League of Legends, hay Fortnite, hàng đợi ghép trận cho người chơi đang chờ trận đấu, khi hệ thống tìm thấy đủ người chơi có cấp độ kỹ năng tương đồng, họ sẽ được đưa vào trận đấu.
    - PvE World of Warcraft: hàng đợi để quản lý những người chơi muốn tham gia các nhiệm vụ đặc biệt, chẳng hạn như raid hoặc dungeon.

Danh sách liên kết đơn nối vòng

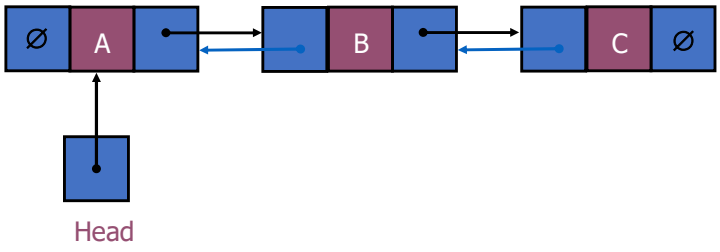


- Quản lý bộ nhớ cấp phát liên tục
  - Bộ nhớ động được cấp phát tự động
  - Mỗi node sẽ quản lý 1 vùng nhớ tự do
    - Chứa thông tin địa chỉ bắt đầu, kết thúc
  - Con trỏ cuối trỏ về đầu → vòng tuần hoàn dễ dàng tìm kiếm vùng nhớ trống phù hợp kích thước yêu cầu
  - Khi giải phóng 1 vùng nhớ → vùng nhớ sẽ được thêm vào danh sách
  - Giúp giám sát các vùng nhớ động và, dễ dàng mở rộng hoặc thu hẹp vùng nhớ cho phép

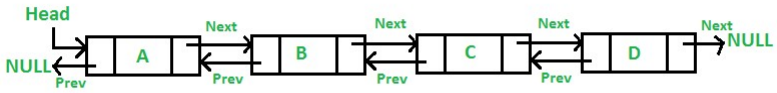
Danh sách liên kết đôi

Danh sách liên kết mở rộng

- **Danh sách liên kết đôi:**
  - Mỗi nút có 2 con trỏ: con trỏ phải trỏ đến phần tử tiếp sau, con trỏ trái trỏ đến phần tử ngay trước.

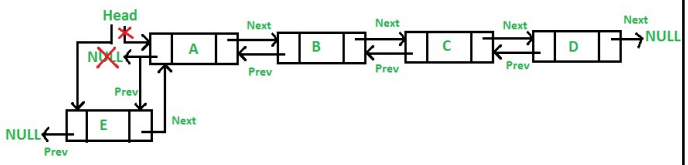


  - **Ưu điểm**: có thể duyệt danh sách theo cả hai chiều, thêm/xóa đơn giản hơn so với DS liên kết đơn
  - Kiểm tra cuối danh sách: con trỏ phải là NULL
  - Đầu danh sách: con trỏ trái là NULL
  - Tốn thêm 1 con trỏ để duy trì danh sách, thêm thao tác xử lý với con trỏ thứ 2



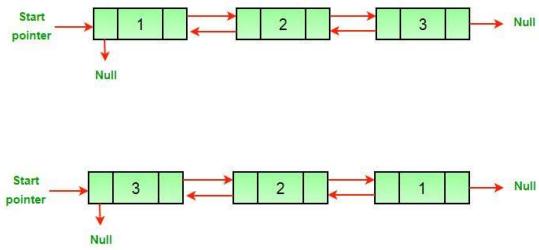

```
void push(Node** head_ref, int new_data)
{
    /* 1. allocate node */
    Node* new_node = new Node();

    /* 2. put in the data */
    new_node->data = new_data;

    /* 3. Make next of new node as head
    and previous as NULL */
    new_node->next = (*head_ref);
    new_node->prev = NULL;

    /* 4. change prev of head node to new node */
    if ((*head_ref) != NULL)
        (*head_ref)->prev = new_node;

    /* 5. move the head to point to the new node */
    (*head_ref) = new_node;
}
```

```
/* Node of a doubly linked list */
struct Node {
    int data;
    struct Node* next; // Pointer to next node in DLL
    struct Node* prev; // Pointer to previous node in DLL
};
```



https://www.geeksforgeeks.org/doubly-linked-list/

Đảo ngược danh sách

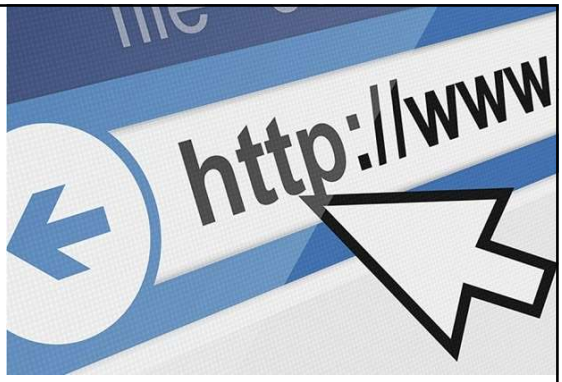



```
void reverse(Node **head_ref)
{
    Node *temp = NULL;
    Node *current = *head_ref;

    /* swap next and prev for all nodes of
    doubly linked list */
    while (current != NULL)
    {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;
        current = current->prev;
    }

    /* Before changing the head, check for the cases like empty
        list and list with only one node */
    if(temp != NULL )
        *head_ref = temp->prev;
}
```

Danh sách liên kết đôi

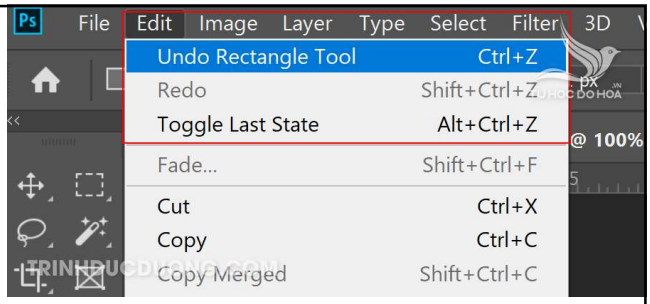
- Một số bài toán
  - Thêm phần tử vào vị trí bất kỳ
  - Xóa phần tử giá trị Key trong danh sách
  - Loại bỏ phần tử trùng lặp
  - Xoay danh sách liên kết đôi
  - Biểu diễn số nguyên lớn với các toán tử +, -, *, / (cỡ hàng nghìn chữ số)
  - Tìm 2 số có tổng bằng giá trị k cho trước

Danh sách liên kết đôi



- Một số ứng dụng
  - Quản lý các URL trong Browser
    - Số lượng URL là không biết trước
    - Thực hiện Back và Forward để di chuyển đến các URL đã thăm và di chuyển tiếp
    - Khi chuyển sang URL mới, sẽ xóa các URL đã thăm ở trước (trong danh sách Forward và Back)
    - Thời gian xử lý O(1)

Danh sách liên kết đôi



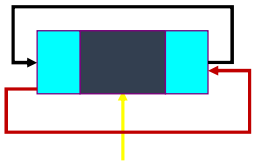
- Quản lý các thao tác trong các phần mềm đồ họa hoặc soạn thảo văn bản
  - Lưu trữ thao tác/ chuỗi thao tác
  - Thực hiện Undo và Redo
  - Xóa 1 thao tác/chuỗi thao tác
  - Lưu trữ số lượng thao tác không cố định dễ dàng
  - Thời gian xử lý O(1)

Danh sách liên kết đôi

- Các ứng dụng khác
  - Quản lý bộ nhớ động (Garbage Collection): duyệt qua các khối dễ dàng hơn so với danh sách liên kết đơn
  - Quản lý các tác vụ trong hệ thống đa nhiệm (multitasking systems):
    - quản lý các tác vụ chờ xử lý
    - Di chuyển qua lại giữa các tác vụ
  - Tổ chức cấu trúc hàng đợi 2 đầu Deque (Double-Ended Queue)
  - Cài đặt LRU Cache (Least Recently Used Cache)
  - Quản lý thư viện nhạc hoặc video (Playlist Management): Di chuyển 2 chiều dễ hơn danh sách liên kết đơn (nhưng tốn thêm bộ nhớ phụ)
  - Quản lý các node trung gian trong hệ thống tàu điện ngầm hoặc bản đồ (Subway System or Map Navigation)

Danh sách liên kết đôi nối vòng

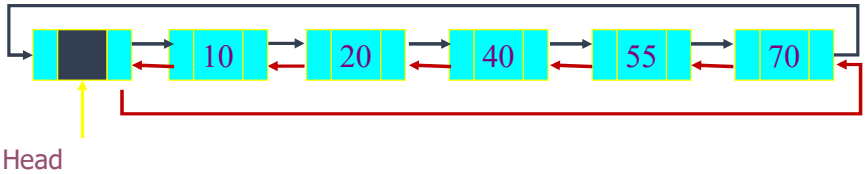
- **Ưu điểm**: có thể di chuyển theo hai chiều, và từ phần tử cuối (đầu) có thể nhảy ngay đến phần tử đầu (cuối) dãy.
- Kiểm tra danh sách rỗng: pNext, pPrev đều trỏ vào 1 phần tử Head.
- Kiểm tra phần tử cuối dãy: pNext trỏ tới Head
- Không dùng nút giả? Kiểm tra cuối và đầu phức tạp hơn 1 chút



Danh sách rỗng

Danh sách liên kết đôi

- **Danh sách liên kết đôi nối vòng**
  - Các con trỏ thừa ở cuối trỏ về đầu và con trỏ thừa ở đầu sẽ trỏ xuống cuối
  - Có thể dùng thêm nút giả để phân biệt đầu và cuối
  - Con trỏ pNext của nút cuối trỏ vào nút đầu và con trỏ pPrev của nút đầu trỏ vào nút cuối.
  - Nút đầu danh sách là nút giả



Danh sách liên kết đôi nối vòng

- Ứng dụng: có thể dùng thay thế cho danh sách liên kết đơn nối vòng
  - Thuật toán điều độ tiến trình HDH - Round-Robin Scheduling
  - Hệ thống quản lý bộ đệm vòng (Circular Buffers)
  - Playlist nhạc
  - Quản lý giỏ hàng trong trang thương mại điện tử
  - Hệ thống Menu xoay vòng (Circular Menus)
  - Mô phỏng các trò chơi theo lượt (Turn-Based Games)
  - Hệ thống quản lý các buffer đa nhiệm (Multi-task Buffer Management)

Danh sách liên kết

- Danh sách liên kết trong C++ STL

  http://www.cplusplus.com/reference/list/list/

```
#include <iostream>
#include <list>
#include <iterator>
using namespace std;

//function for printing the elements in a list
void showlist(list <int> g)
{
    list <int> :: iterator it;
    for(it = g.begin(); it != g.end(); ++it)
        cout << '\t' << *it;
    cout << '\n';
}
```

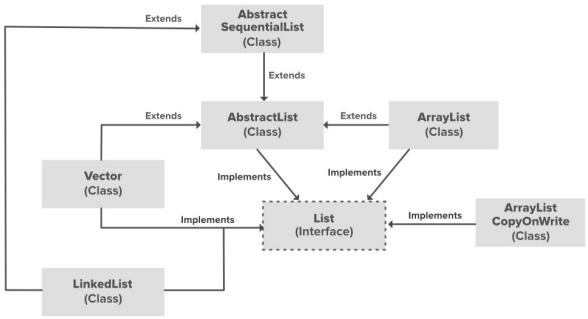
```
int main()
{

    list <int> gqlist1, gqlist2;


    for (int i = 0; i < 10; ++i)
    {
        gqlist1.push_back(i * 2);
        gqlist2.push_front(i * 3);
    }
    cout << "\nList 1 (gqlist1) is : ";
    showlist(gqlist1);

    cout << "\nList 2 (gqlist2) is : ";
    showlist(gqlist2);

    cout << "\ngqlist1.front() : " << gqlist1.front();
    cout << "\ngqlist1.back() : " << gqlist1.back();

    cout << "\ngqlist1.pop_front() : ";
    gqlist1.pop_front();
    showlist(gqlist1);

    cout << "\ngqlist2.pop_back() : ";
    gqlist2.pop_back();
    showlist(gqlist2);

    cout << "\ngqlist1.reverse() : ";
    gqlist1.reverse();
    showlist(gqlist1);

    cout << "\ngqlist2.sort(): ";
    gqlist2.sort();
    showlist(gqlist2);

    return 0;

}
```

Danh sách liên kết

- Một số thao tác cơ bản của list trong STL C++
  - front() – Returns the value of the first element in the list.
  - back() – Returns the value of the last element in the list .
  - push_front(g) – Adds a new element 'g' at the beginning of the list .
  - push_back(g) – Adds a new element 'g' at the end of the list.
  - pop_front() – Removes the first element of the list, and reduces size of the list by 1.
  - pop_back() – Removes the last element of the list, and reduces size of the list by 1
  - list::begin() and list::end() in C++ STL– **begin()** function returns an iterator pointing to the first element of the list
  - end()– **end()** function returns an iterator pointing to the theoretical last element which follows the last element.
  - empty() – Returns whether the list is empty(1) or not(0).
  - insert() – Inserts new elements in the list before the element at a specified position.
  - erase() – Removes a single element or a range of elements from the list.
  - assign() – Assigns new elements to list by replacing current elements and resizes the list.
  - remove() – Removes all the elements from the list, which are equal to given element.
  - list::remove_if() in C++ STL– Used to remove all the values from the list that correspond true to the predicate or condition given as parameter to the function.
  - reverse() – Reverses the list.
  - size() – Returns the number of elements in the list.
  - list resize()function in C++ STL– Used to resize a list container.
  - sort() – Sorts the list in increasing order.

Danh sách liên kết

```java
import java.util.*;

public class Test {

    public static void main(String args[])
    {
        // Creating object of the
        // class linked list
        LinkedList<String> ll
            = new LinkedList<String>();

        // Adding elements to the linked list
        ll.add("A");
        ll.add("B");
        ll.addLast("C");
        ll.addFirst("D");
        ll.add(2, "E");

        System.out.println(ll);

        ll.remove("B");
        ll.remove(3);
        ll.removeFirst();
        ll.removeLast();

        System.out.println(ll);
    }
}
```

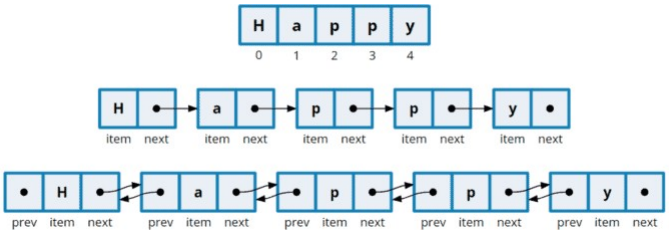


- Danh sách liên kết trong Java

https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

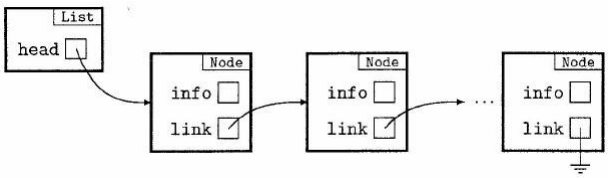
Method	Description
add(int index, E element)	This method Inserts the specified element at the specified position in this list.
add(E e)	This method Appends the specified element to the end of this list.
addAll(int index, Collection<E> c)	This method Inserts all of the elements in the specified collection into this list, starting at the specified position.
addAll(Collection<E> c)	This method Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
addFirst(E e)	This method Inserts the specified element at the beginning of this list.
addLast(E e)	This method Appends the specified element to the end of this list.
clear()	This method removes all of the elements from this list.
clone()	This method returns a shallow copy of this LinkedList.
contains(Object o)	This method returns true if this list contains the specified element.
descendingIterator()	This method returns an iterator over the elements in this deque in reverse sequential order.
element()	This method retrieves, but does not remove, the head (first element) of this list.
get(int index)	This method returns the element at the specified position in this list.
getFirst()	This method returns the first element in this list.
getLast()	This method returns the last element in this list.
indexOf(Object o)	This method returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
lastIndexOf(Object o)	This method returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
listIterator(int index)	This method returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list.
offer(E e)	This method Adds the specified element as the tail (last element) of this list.
offerFirst(E e)	This method Inserts the specified element at the front of this list.
offerLast(E e)	This method Inserts the specified element at the end of this list.
peek()	This method retrieves, but does not remove, the head (first element) of this list.

peekFirst()	This method retrieves, but does not remove, the first element of this list, or returns null if this list is empty.
peekLast()	This method retrieves, but does not remove, the last element of this list, or returns null if this list is empty.
poll()	This method retrieves and removes the head (first element) of this list.
pollFirst()	This method retrieves and removes the first element of this list, or returns null if this list is empty.
pollLast()	This method retrieves and removes the last element of this list, or returns null if this list is empty.
pop()	This method Pops an element from the stack represented by this list.
push(E e)	This method Pushes an element onto the stack represented by this list.
remove()	This method retrieves and removes the head (first element) of this list.
remove(int index)	This method removes the element at the specified position in this list.
remove(Object o)	This method removes the first occurrence of the specified element from this list, if it is present.
removeFirst()	This method removes and returns the first element from this list.
removeFirstOccurrence(Object o)	This method removes the first occurrence of the specified element in this list (when traversing the list from head to tail).
removeLast()	This method removes and returns the last element from this list.
removeLastOccurrence(Object o)	This method removes the last occurrence of the specified element in this list (when traversing the list from head to tail).
set(int index, E element)	This method replaces the element at the specified position in this list with the specified element.
size()	This method returns the number of elements in this list.
spliterator()	This method Creates a late-binding and fail-fast Spliterator over the elements in this list.
toArray()	This method returns an array containing all of the elements in this list in proper sequence (from first to last element).
toArray(T[] a)	This method returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.
toString()	This method returns a String containing all of the elements in this list in proper sequence (from first to last element), each element is separated by commas and the String is enclosed in square brackets.



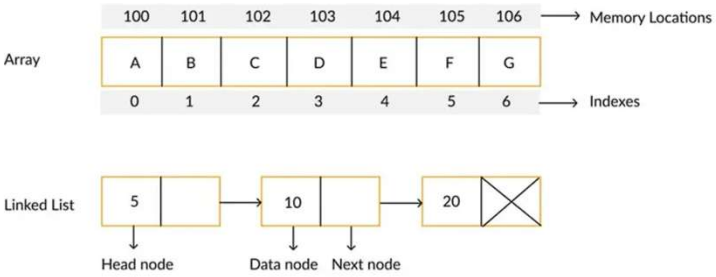
Danh sách tuyến tính – linear list

Cấu trúc dữ liệu – danh sách tuyến tính



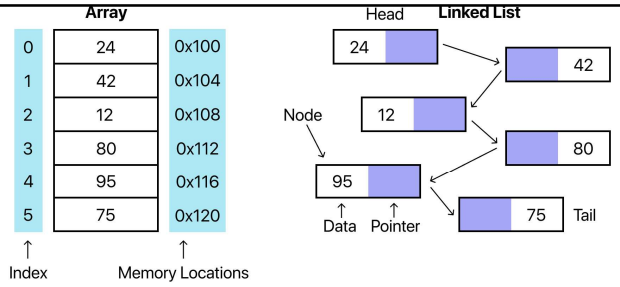
- Danh sách tuyến tính – linear list
  - Các phần tử có cùng kiểu, và
  - Tuân theo thứ tự tuyến tính
  - Thao tác thêm/xóa cần giữ thứ tự của các phần tử đúng
- VD. Danh sách ban đầu A, B, C, D
  - Thêm E và vị trí thứ 2: A, E, B, C, D
  - Xóa A: E, B, C, D
  - Thêm G vào vị trí đầu tiên: G, E, B, C, D

Danh sách tuyến tính



- Các thao tác cơ bản
  - Thêm phần tử:
    - Thêm vào vị trí bất kỳ - insertAt
    - Thêm vào đầu: push
    - Thêm vào cuối: append
  - Lấy phần tử:
    - Lấy khỏi đầu: pop
    - Lấy phần tử ở vị trí bất kỳ: removeAt
  - Xóa phần tử:
    - Xóa 1 phần tử ở vị trí cho trước
    - Xóa toàn bộ danh sách
  - Tìm kiếm:
    - Tìm kiếm theo giá trị phần tử
  - Số lượng phần tử hiện tại
  - Kiểm tra danh sách rỗng
  - Cập nhật giá trị phần tử
  - ….

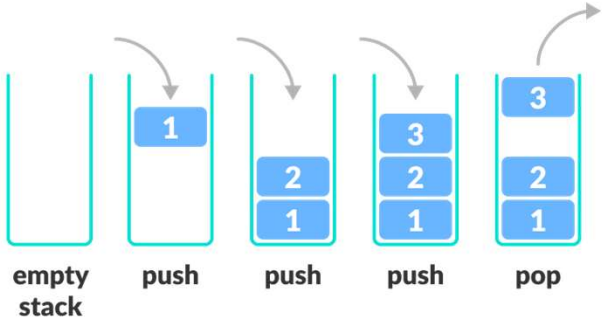
Danh sách tuyến tính



- Cài đặt dùng mảng
  - Truy cập phần tử nhanh O(1)
  - Cần biết trước số lượng tối đa (hoặc dùng mảng động kích thước biến đổi)
  - Thêm/xóa phải dịch phần tử
- Cài đặt dùng danh sách liên kết đơn
  - Truy cập phần tử O(n)
  - Không cần biết trước số lượng tối đa
  - Không phải dịch phần tử khi thêm/xóa
  - Mỗi phần tử tốn thêm bộ nhớ phụ lưu trữ con trỏ

Danh sách tuyến tính

- Một số ứng dụng:
  - Cài đặt stack và queue
  - Quản lý các danh sách: VD danh sách sinh viên với thứ tự cố định theo MSSV
  - Danh sách các nhiệm vụ trong to-do list: các nhiệm vụ đã dc xếp theo thứ tự cho trước
  - Quản lý lịch sử duyệt web hoặc lịch sử thao tác trên ứng dụng
  - Quản lý agenda trong các ứng dụng lịch như google calendar
  - Quản lý danh bạ: các contact đã theo thứ tự sdt hoặc họ tên
  - Quản lý sản phẩm trong kho: các sản phẩm theo thứ tự mã sản phẩm hoặc thời gian thêm



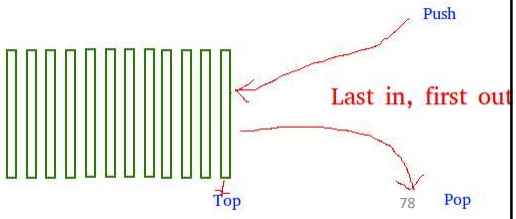
Ngăn xếp – Stack

Các cấu trúc dữ liệu cơ bản

- **Container** : là các cấu trúc dữ liệu cho phép lưu trữ và lấy dữ liệu độc lập với nội dung của dữ liệu.
- Các container phân biệt theo thứ tự lấy mà chúng hỗ trợ.
- Một số kiểu container quan trọng, thứ tự lấy của chúng phụ thuộc vào thứ tự chèn:
  - **Stack**: hỗ trợ lấy theo thứ tự vào sau ra trước – Last In First Out
  - **Queue**: hỗ trợ lấy theo thứ tự vào trước ra trước – First In First Out

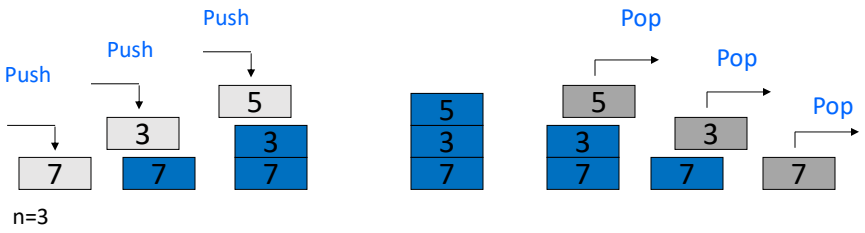
Ngăn xếp – Stack

- Ngăn xếp – Stack
  - Thêm vào và lấy ra chỉ diễn ra tại 1 đầu – top
  - Phần tử thêm vào sau cùng sẽ được lấy ra trước tiên – LIFO
  - Đầu còn lại của ngăn xếp là bottom – đáy
  - Các thao tác cơ bản
    - Push(x, S): chèn 1 phần tử x vào ngăn xếp
    - Pop(S): lấy ra 1 phần tử khỏi ngăn xếp
    - Top(S): truy cập phần tử ở đỉnh của ngăn xếp
    - Empty(S): trả về true nếu ngăn xếp rỗng



Ngăn xếp – Stack

- Ví dụ: đọc vào một danh sách gồm n số nguyên (n<100), và in ra các số đã đọc theo thứ tự ngược



Nhập vào: 7, 3, 5

In ra: 5, 3, 7

Ngăn xếp – Stack

- Cài đặt ngăn xếp
  - Dùng mảng
    - Đỉnh stack = cuối mảng
    - Thêm và lấy ra tại cuối mảng → thời gian O(1)
    - Cần biết trước số lượng tối đa hoặc dùng mảng kích thước biến đổi (thời gian có thể tới O(n))
  - Dùng danh sách liên kết đơn
    - Đỉnh stack = đầu danh sách
    - Thêm và lấy ra tại đầu → thời gian O(1)
    - Mỗi phần tử sẽ tốn thêm 1 con trỏ

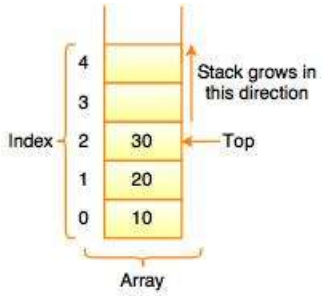
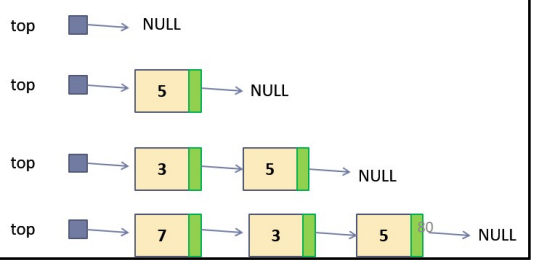


Fig. Implementation Stack using Array



```c
#include <stdio.h>
#define MAX 100

typedef struct {
    int arr[MAX];
    int top;
} Stack;

// Khởi tạo stack
void init(Stack* s) {
    s->top = -1;
}

// Kiểm tra stack rỗng
int isEmpty(Stack* s) {
    return s->top == -1;
}
```

Cài đặt dùng mảng


```c
// Kiểm tra stack đầy
int isFull(Stack* s) {
    return s->top == MAX - 1;
}

// Thêm phần tử vào stack (push)
int push(Stack* s, int value) {
    if (isFull(s)) {
        printf("Stack day! Khong the them phan tu.\n");
        return 0;  // Push thất bại
    }
    s->arr[++(s->top)] = value;
    return 1;  // Push thành công
}

// Lấy phần tử ra khỏi stack (pop)
int pop(Stack* s, int* value) {
    if (isEmpty(s)) {
        printf("Stack rong! Khong the lay phan tu.\n");
        return 0;  // Pop thất bại
    }
    *value = s->arr[(s->top)--];
    return 1;  // Pop thành công
}
```

```c
// Lấy phần tử đầu stack mà không xóa (peek)
int peek(Stack* s, int* value) {
    if (isEmpty(s)) {
        printf("Stack rong!\n");
        return 0;
    }
    *value = s->arr[s->top];
    return 1;
}
```

```c
int main() {
    Stack s;
    init(&s);

    push(&s, 10);
    push(&s, 20);
    push(&s, 30);

    int x;
    peek(&s, &x);
    printf("Phan tu tren cung: %d\n", x);

    while (!isEmpty(&s)) {
        pop(&s, &x);
        printf("Lay ra: %d\n", x);
    }

    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

// Định nghĩa nút của stack
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Khởi tạo
typedef struct {
    Node* top;
} Stack;

// Khởi tạo stack rỗng
void init(Stack* s) {
    s->top = NULL;
}
```

```c
// Kiểm tra stack rỗng
int isEmpty(Stack* s) {
    return s->top == NULL;
}

// Push: thêm phần tử vào đầu stack
void push(Stack* s, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Khong du bo nho!\n");
        return;
    }
    newNode->data = value;
    newNode->next = s->top;
    s->top = newNode;
}
```

Cài đặt dùng danh sách liên kết đơn


```c
// Pop: lấy phần tử đầu stack
int pop(Stack* s, int* value) {
    if (isEmpty(s)) {
        printf("Stack rong!\n");
        return 0;
    }
    Node* temp = s->top;
    *value = temp->data;
    s->top = temp->next;
    free(temp);
    return 1;
}

// Peek: xem phần tử trên cùng mà không xóa
int peek(Stack* s, int* value) {
    if (isEmpty(s)) {
        printf("Stack rong!\n");
        return 0;
    }
    *value = s->top->data;
    return 1;
}
```

```c
int main() {
    Stack s;
    init(&s);

    push(&s, 10);
    push(&s, 20);
    push(&s, 30);

    int x;
    peek(&s, &x);
    printf("Phan tu tren cung: %d\n", x);

    while (!isEmpty(&s)) {
        pop(&s, &x);
        printf("Lay ra: %d\n", x);
    }

    return 0;
}
```

Ngăn xếp – Stack

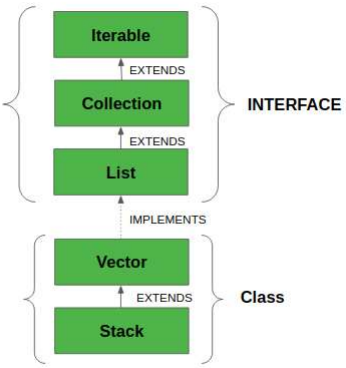
- Stack trong STL của C++: cài dặt dùng các container như vector, deque và list
- Các thao tác cơ bản
  - empty() – Returns whether the stack is empty – Time Complexity : O(1)
    size() – Returns the size of the stack – Time Complexity : O(1)
    top() – Returns a reference to the top most element of the stack – Time Complexity : O(1)
    push(g) – Adds the element 'g' at the top of the stack – Time Complexity : O(1)
    pop() – Deletes the top most element of the stack – Time Complexity : O(1)

```cpp
#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<int> stack;
    stack.push(21);
    stack.push(22);
    stack.push(24);
    stack.push(25);

    stack.pop();
    stack.pop();


    while (!stack.empty()) {
        cout << ' ' << stack.top();
        stack.pop();
    }
}
```

https://www.cplusplus.com/reference/stack/stack/

Ngăn xếp – Stack

- Stack trong Java




```java
import java.util.*;
import java.io.*;

public class StackDemo {


    // Main Method
    public static void main(String args[])
    {
        // Creating an empty Stack
        Stack<String> stack = new Stack<String>();


        // Use push() to add elements into the Stack
        stack.push("Welcome");
        stack.push("To");
        stack.push("Geeks");
        stack.push("For");
        stack.push("Geeks");


        // Displaying the Stack
        System.out.println("Initial Stack: " + stack);


        // Fetching the element at the head of the Stack
        System.out.println("The element at the top of the"
                           + " stack is: " + stack.peek());


        // Displaying the Stack after the Operation
        System.out.println("Final Stack: " + stack);
    }
}
```

Stack trong Java

METHOD	DESCRIPTION
empty()	It returns true if nothing is on the top of the stack. Else, returns false.
peek()	Returns the element on the top of the stack, but does not remove it.
pop()	Removes and returns the top element of the stack. An 'EmptyStackException' An exception is thrown if we call pop() when the invoking stack is empty.
push(Object element)	Pushes an element on the top of the stack.
search(Object element)	It determines whether an object exists in the stack. If the element is found, It returns the position of the element from the top of the stack. Else, it returns -1.

Ứng dụng của stack

Ứng dụng của stack

- Quản lý các lời gọi hàm trong ngôn ngữ lập trình
- Stack trong máy tính
  - Xử lý ngắt
  - Quản lý bộ nhớ tạm, truy vết (backtrack)
- Trong các chương trình biên dịch (compiler)
  - Sử dụng để tính giá trị biểu thức
  - Chuyển đổi biểu thức dạng trung tố sang hậu tố
  - Kiểm tra dấu ngoặc hợp lệ trong biểu thức
- Trong trình duyệt web, trình soạn thảo văn bản, phần mềm đồ họa
  - Thao tác Undo/ Redo, Back/Forward
- Quản lý chức năng quay lại trong ứng dụng (App Navigation)
- Đảo ngược chuỗi ký tự hoặc danh sách liên kết đơn
- Duyệt đồ thị, cây hoặc trong thuật toán quay lui – back tracking, khử đệ quy

Ứng dụng của stack

- Định giá biểu thức :

  A = b + c * d / e – f

- **Biểu thức trung tố** : toán tử hai ngôi đứng giữa hai toán hạng mà nó tác động, toán tử một ngôi đứng trước/sau toán hạng.
- **Biểu thức hậu tố** : Toán tử đứng sau toán hạng
- Ví dụ:

Trung tố	$a * (b - c)/d$
Hậu tố	$abc - * d/$
Tiền tố	$/ * a - bcd$

Định giá biểu thức

- **Tính giá trị biểu thức hậu tố** :
  Trung tố : `(4 * 2) + 5 + (6 * 5)`
  Hậu tố : `4 2 * 5 + 6 5 * +`

- **Thuật toán**: Duyệt biểu thức hậu tố
  - **Gặp toán hạng**: đẩy vào stack
  - **Gặp toán tử 1 ngôi**: lấy ra 1 toán hạng trong stack, áp dụng toán tử lên toán hạng và đẩy kết quả trở lại stack
  - **Gặp toán tử 2 ngôi**: lấy 2 toán hạng ở đỉnh stack theo thứ tự toán hạng 2 lấy trước, áp dụng toán tử lên 2 toán hạng đó, kết quả lại đẩy vào stack.

  Kết thúc, đưa ra kết quả là giá trị ở đỉnh stack.

Định giá biểu thức

- Tính giá trị biểu thức hậu tố sau: `4 2 * 4 + 6 5 * +`

Thứ tự	STACK	Diễn giải
4	4	Gặp toán hạng, đẩy vào STACK
2	4,2	Đỉnh stack ở sau
*	8	Lấy 2 toán hạng, áp dụng 4*2 = 8, đẩy KQ lại STACK
4	8,4	
+	12	Lấy 2 toán hạng, TH2 là 4 lấy trước, áp dụng 8+4=12, đẩy KQ vào Stack
6	12,6	
5	12,6,5	
*	12,30	Gặp toán tử *, lấy 2 toán hạng, TH2 là 5, áp dụng 6*5=30, đẩy KQ và STACK
+	42	Gặp toán tử +, lấy 2 TH, 12+30 = 42
END	42	Kết quả biểu thức trên là 42

Định giá biểu thức hậu tố

$$2\ 4\ 9\ \sqrt{}\ -\ +\ 4\ 2\ \hat{}\ *$$

Thứ tự	Mô tả	Trạng thái stack	Ghi chú
1	Gặp 2 (toán hạng)	2	
2	4	$2, 4$	
3	9	$2, 4, 9$	
4	$\sqrt{}$ (toán tử 1 ngôi)	$2, 4, 3$	Thực hiện $\sqrt{9}$
5	$-$ (toán tử 2 ngôi)	$2, 1$	Thực hiện $4 - 3$
6	$+$ (toán tử 2 ngôi)	3	Thực hiện $2 + 1$
7	4	$3, 4$	
8	2	$3, 4, 2$	
9	^ (toán tử 2 ngôi)	$3, 16$	Thực hiện 4^2
10	$*$ (toán tử 2 ngôi)	48	Thực hiện $3 * 16$
END		48	Kết quả là 48

Định giá biểu thức

- Xây dựng chương trình tính giá trị của một một biểu thức hậu tố được lưu trong một xâu ký tự.
- **Toán hạng**:
  các số nguyên không âm
- **Toán tử** : (xét các toán hạng 2 ngôi đơn giản)
  +, -, *, /, %, ^ (lũy thừa)

```
int compute(int left, int right, char op);
        /* Thực hiện tính: "left op right" */

int isOperator(char op);
        /* Kiểm tra op có phải là toán tử không?
           op phải là một trong số '+','-','*','/','%','^'
           nếu đúng thì trả về 1, sai thì trả về 0 */
```

Định giá biểu thức

```
int isOperator(char op)
{
if ((op == '+') || (op == '-') ||
   (op == '*' )|| (op == '%' )||
   (op == '/' )|| (op == '^' ))
     return 1;
   else
     return 0;
}
```

```
int compute(int left, int right, char op)
{
  int value;
  switch(op){
    case '+':value = left + right;
       break;
    case '-':value = left - right;
       break;
    case '*':value = left * right;
       break;
    case '%':value = left % right;
       break;
    case '/':value = left / right;
       break;
    case '^':value = pow(left, right);
       break;
  }
  return value;
}
```



```
int main()
{
    char eq[]="21*56*3/+7-";
int left, right, expValue;
char ch;
NODE *top=NULL;

for(int i=0;i<strlen(eq);i++)
{
         if(isOperator(eq[i])==0)
         {
              //printf("%d\n",eq[i]-'0');
              push(top,eq[i]-'0');
         }
```

```
         else
         {
              pop(top,right);
              pop(top,left);
              expValue=compute(left,right,eq[i]);
              printf("%d %c %d =
                  %d\n",left,eq[i],right,expValue);
              push(top,expValue);
         }
    }
    pop(top,expValue);
    printf("%d",expValue);
    getch();
    return 0;
}
```

Chuyển biểu thức dạng trung tố sang hậu tố

Biểu thức dạng trung tố	Biểu thức dạng hậu tố
3+5	3 5 +
3+5*2	3 5 2 * +
(3+5)*2	3 5 + 2*
a*b*c*d*e*f	a b*c*d*e*f*
1 + (-5) / (6 * (7+8))	1 5 - 6 7 8 + * / +

Chuyển biểu thức sang trung tố sang hậu tố

Toán tử	Mức ưu tiên
!, - (toán tử 1 ngôi giai thừa và đảo dấu)	6
abs, sin, cos, tan, exp, ln, lg, round, trunc, sqr, sqrt, arctan	6
^(toán tử 2 ngôi)	6 (5.5)
*, /, % (toán tử 2 ngôi)	5
+, - (toán tử 2 ngôi)	4
==, !=, <, >, ≤, ≥ (toán tử quan hệ)	3
Not (toán tử logic)	2
&&, || (toán tử logic)	1
= (toán tử gán)	0

Chuyển biểu thức dạng trung tố sang hậu tố

Duyệt lần lượt biểu thức trung tố từ trái qua phải:

1. **Gặp toán hạng**: viết sang biểu thức kết quả (là biểu thức hậu tố cần tìm)
2. **Gặp toán tử** (có độ ưu tiên nhỏ hơn 6):
   - nếu **stack rỗng**, hoặc đỉnh stack là toán tử có độ ưu tiên nhỏ hơn, hoặc là **'('** : đẩy toán tử đang xét vào stack
   - ***Ngược lại***: lấy các toán tử ở đỉnh stack có độ ưu tiên lớn hơn hoặc bằng toán tử đang xét lần lượt đưa vào biểu thức kết quả và đẩy toán tử đang xét vào stack
3. **Gặp toán tử** có độ ưu tiên 6, hoặc **'('**: đẩy vào stack
4. Gặp **')'**: lấy tất cả các toán tử trong stack cho đến khi gặp '(' đầu tiên, đưa sang biểu thức kết quả theo đúng thứ tự (không đưa '(,)' vào biểu thức kết quả), và đẩy 1 ký hiệu '(' ra khỏi stack.
5. Nếu duyệt hết biểu thức trung tố, lấy nốt những toán tử trong stack đưa sang biểu thức kết quả (theo đúng thứ tự).

Chuyển biểu thức sạng trung tố sang hậu tố

- Ví dụ: Chuyển biểu thức sau sang dạng hậu tố

$$-3 + \frac{4^{5^a}}{2} - 7$$

- Biểu thức trên được viết lại là

$$-3 + 4\text{^}(5\text{^}a\ )/2 - 7$$

hoặc $-3 + 4\text{^}5\text{^}a/2 - 7$

- Chú ý: dấu $-$ trong -3 là toán tử 1 ngôi có độ ưu tiên 6

$-3+4\hat{}5\hat{}a/2-7$

Thứ tự	Mô tả	Trạng thái stack	Kết quả
0	Stack rỗng	$\emptyset$	$\emptyset$
1	Gặp $-$ (toán tử 1 ngôi)	$-$	$\emptyset$
2	Gặp 3 (toán hạng), đưa sang biểu thức kết quả)	$-$	3
3	Gặp $+$ (toán tử 2 ngôi), lấy $-$ khỏi stack và đẩy $+$ vào	$+$	$3\ -$
4	Gặp 4 (toán hạng)	$+$	$3-4$
5	Gặp ^ (toán tử 2 ngôi bậc 6)	$+, \hat{}$	$3-4$
6	Gặp 5	$+, \hat{}$	$3-4\ 5$
7	Gặp ^	$+, \hat{}, \hat{}$	$3-4\ 5$
8	Gặp a (toán hạng)	$+, \hat{}, \hat{}$	$3-4\ 5\ a$

$-3+4\hat{}5\hat{}a/2-7$

Bước	Mô tả	Trạng thái stack	Kết quả
9	Gặp / (toán tử 2 ngôi), lấy các toán tử có độ ưu tiên lớn hơn hoặc bằng ra khỏi stack, sau đó đẩy / vào	$+,/$	$3-4\ 5\ a\ \hat{}\ \hat{}$
10	Gặp 2	$+,/$	$3-4\ 5\ a\ \hat{}\ \hat{}\ 2$
11	Gặp $-$ (toán tử 2 ngôi), lấy / và $+$ khỏi stack, rồi đẩy $-$ vào	$-$	$3-4\ 5\ a\ \hat{}\ \hat{}\ 2\ /\ +$
12	Gặp 7 (toán hạng)	$-$	$3-4\ 5\ a\ \hat{}\ \hat{}2\ /\ +7$

Biểu thức kết quả: $3-4\ 5\ a\ \hat{}\ \hat{}2\ /\ +7\ -$

Chuyển biểu thức sạng trung tố sang hậu tố

- Viết chương trình minh họa chuyển biểu thức dạng trung tố sang dạng hậu tố
- Biểu thức chỉ gồm:
  - Toán tử: $+, -, *, /, \%, \hat{}$
  - Toán hạng: 1 ký tự
    vd. a, b, c, 3, 4, 6,…
  - Dấu ngoặc: (, và)
- Ví dụ:

$$3 + \frac{5}{3} + (1 - a/b\hat{}2) \text{ hay } 3 + 5/3 + (1 - a/b\hat{}2)$$


```c
int isOperator(char op)
{
if ((op == '+') || (op == '-') ||
   (op == '*' )|| (op == '%' )||
   (op == '/' )|| (op == '^' ))
     return 1;
   else
     return 0;
}
```

```c
int priority(char op)
{
    if((op == '+') || (op == '-')) return 4;
    if((op == '*' )|| (op == '%' )||
         (op == '/' )) return 5;
    if(op == '^')return 6;

    return 0;
}
```

```c
typedef struct node
{
   char data;
   struct node *pNext;
} NODE;
```

```
int main()
{
    char eq[]="(3+5*(7-6*5)+4)^5^6";
    //char eq[]="1+3*4/5";
    char ch;
    char *retVal=(char*)calloc(sizeof(char),strlen(eq)+1);

    NODE *top=NULL;
    int j=0;
    for(int i=0;i<strlen(eq); i++)
    {
        //printf("%c  ",eq[i]);
        if((eq[i]!='(')&&(eq[i]!=')'))
            if(isOperator(eq[i])==0)//toan hang
            {
                retVal[j]=eq[i];
                j++;
            }
```



```
else //toan tu
{
    if(top==NULL || top->data=='(' || priority(eq[i])==6 ||
       priority(eq[i])>priority(top->data)) push(top,eq[i]);
    else
        {
        ch = top->data;
        while(top !=NULL && priority(eq[i]) <= priority(ch)) {
            retVal[j]=pop(top);
            j++;
            if(top!=NULL) ch = top->data;
        }
        push(top,eq[i]);
        }
}
```

```
else //dau ( hoac )
    {
        if(eq[i]=='(') push(top,eq[i]);
        else
        {
            ch=pop(top) ;
            while(ch!='(')
            {
                retVal[j]=ch;
                j++;
                ch=pop(top) ;
            }
        }
    }
}
```

```
while(top!=NULL) //lay not cac toan tu con lai
    {
        ch=pop(top) ;
        retVal[j]=ch;
        j++;
    }
    retVal[j]='\0';

    printf("\n\n%s",retVal);
    getch();
    return 0;
}
```


Một số bài tập

- Bài 1. Viết chương trình kiểm tra cặp ngoặc hợp lệ Có 3 loại ngoặc (), {}, và []
  [({})](): hợp lệ
  ([} {}): không hợp lệ
- Bài 2. Viết chương trình kiểm tra thẻ xml/html hợp lệ
- Bài 3. Viết chương trình minh họa thao tác undo và redo trong soạn thảo văn bản
- Bài 4. Cài đặt Stack mở rộng thêm theo tác findmin() - tìm và trả về phần tử có giá trị nhỏ nhất trong stack với chi phí thời gian của tất cả các thao tác chỉ là O(1)
- Bài 5. Cài đặt 2 stack chỉ dùng 1 mảng duy nhất
- Bài 6. Tìm phần tử phổ biến nhất dãy (có tần số >n/2)
  VD. Dãy 1,2,2,3,4,3,2,2 thì 2 là phần tử phổ biến

Kiểm tra cặp ngoặc hợp lệ

Ý tưởng dùng Stack:

- Duyệt từng ký tự trong chuỗi:
  - Nếu gặp dấu **mở** ((, {, [) → **push** vào Stack.
  - Nếu gặp dấu **đóng** (), },]) → kiểm tra:
    - Nếu Stack **rỗng** → Sai ngay.
    - Lấy dấu mở trên cùng Stack ra (pop).
    - Kiểm tra dấu mở và dấu đóng có **khớp** không.
- Kết thúc:
  - Nếu Stack **rỗng** hoàn toàn → Chuỗi **hợp lệ**.
  - Nếu còn dư dấu mở → **không hợp lệ**.

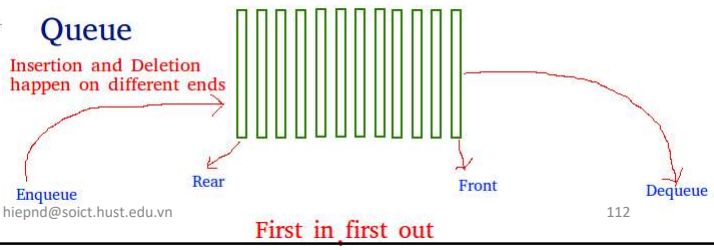

```c
int isMatchingPair(char open, char close) {
    return (open == '(' && close == ')') ||
        (open == '{' && close == '}') ||
        (open == '[' && close == ']');
}
```

```c
int isValid(char* str) {
    Stack s;
    init(&s);

    for (int i = 0; i < strlen(str); i++) {
        char c = str[i];
        if (c == '(' || c == '{' || c == '[') {
            push(&s, c);
        }
        else if (c == ')' || c == '}' || c == ']') {
            if (isEmpty(&s)) return 0; // Không có dấu mở
            char open = pop(&s);
            if (!isMatchingPair(open, c)) return 0; // Không khớp
        }
    }

    return isEmpty(&s); // Nếu stack rỗng thì OK
}
```

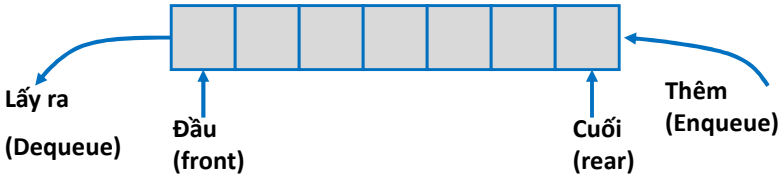
Hàng đợi - queue

Hàng đợi - Queue

- **Hàng đợi – Queue**
  - Việc thêm vào và lấy ra được diễn ra tại 2 đầu khác nhau
  - Phần tử thêm vào trước tiên sẽ được lấy ra trước tiên: lưu trữ và lấy dữ liệu theo thứ tự vào trước ra trước (FIFO)



Cài đặt queue



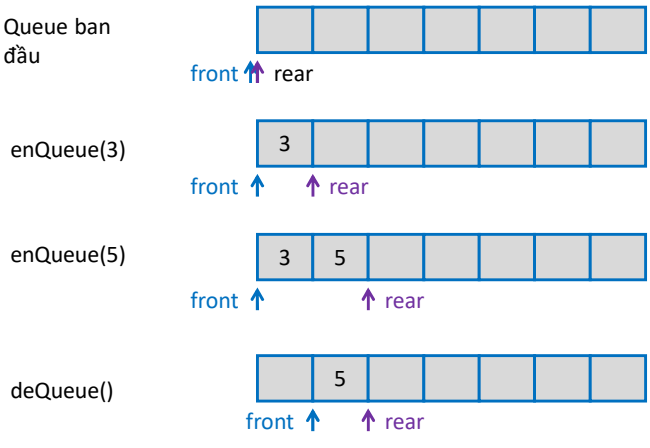
- Các phương thức cơ bản:
  - enQueue() : thêm một phần tử mới vào queue
  - deQueue() : lấy một phần tử khỏi queue
  - empty() : kiểm tra xem queue có rỗng hay không
  - front() : trả về giá trị phần tử ở đầu queue, mà không hủy nó
- Lưu trữ queue
  - Lưu trữ kế tiếp dùng mảng
  - Lưu trữ sử dụng danh sách móc nối

Lưu trữ bằng mảng

Cần có 2 chỉ số
- Chỉ số phần tử đầu Queue hiện tại
- Chỉ số phần tử cuối Queue hiện tại

Khởi tạo: chỉ số là -1
Queue rỗng → front = rear

```
int isEmpty(int queue[], int front, int rear)
{
    if(rear==front) return 1;
    else return 0;
}
```



Lưu trữ bằng mảng

- Thêm một phần tử mới vào queue: thêm vào cuối mảng

```
int enQueue(int queue[], int &rear, int value)
{
   if(rear < MAX-1)
   {
      rear= rear +1;
      queue[rear] = value;
      return 0;
   }
   else
   {
      printf("queue da day !\n");
      return -1;
   }
}
```

- Lấy một phần tử khỏi queue: lấy ở đầu mảng

```
int deQueue(int queue[], int &front, int rear,
  int &value)
{
   if(front == rear)
   {
      printf("Queue rong !\n");
      return -1;
   }

   front = front + 1;
   value = queue[front];
}
```

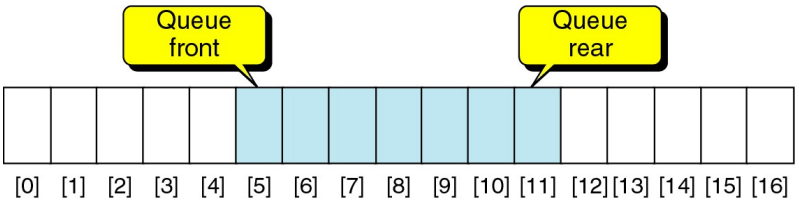

Lưu trữ bằng mảng

- Hàm Front: Trả về giá trị phần tử đang ở đầu queue

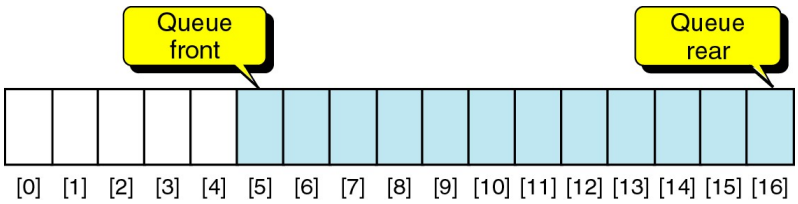
```
int Front(int queue[], int front, int rear, int &value)
{
   if(front == rear)
   {
     printf("Queue rong !\n");
     return -1;
   }
   value = queue[front+1];
}
```

```
#define MAX 10
int main()
{
   int queue[MAX];
   int front,rear;
   int n,value;
   front=rear=(-1);
   enQueue(queue,rear,10);
   if(empty(queue,front,rear)==1)
        printf("Queue dang rong!\n");
   else {
        Front(queue,front,rear,value);
        printf("Dau queue : %d\n",value);
        }

   getch();
}
```

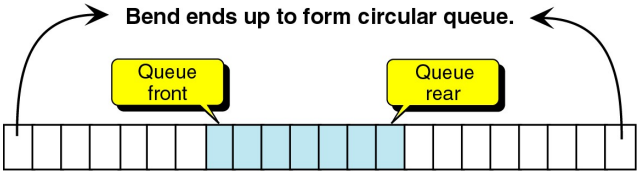
Lưu trữ bằng mảng



- Queue tăng hết mảng

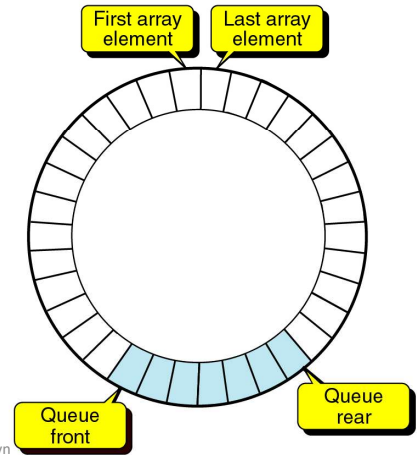


Cần sử dụng mảng kích thước rất lớn ?



Sử dụng mảng dạng vòng (về mặt logic)

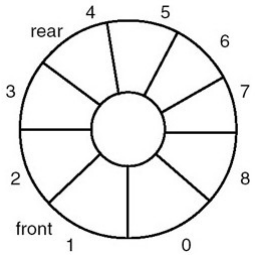
- Trong máy tính lưu trữ vật lý chỉ có duy nhất mảng 1 chiều!



Queue dạng vòng

- Ý tưởng: Khởi tạo ban đầu

```
front=rear=(0);
```




```
int deQueue(int queue[], int &front, int
  rear, int &value)
{
   if(front == rear)
   {
      printf("Queue rong !\n");
      return -1;
   }

   front = (front + 1)%MAX;
   value = queue[front];
}
```

```
int enQueue(int queue[], int front, int &rear, int
  value)
{
    if(((rear +1)%MAX) == front)
    {
       printf("queue da day, khong the them
  %d!\n",value);
       return -1;
    }
    rear= (rear +1)%MAX;
    queue[rear] = value;
    return 0;
}
```


Queue dạng vòng

- Làm thế nào để phân biệt được hàng đợi đầy và hàng đợi rỗng?
  - Chỉ số front = rear trong cả 2 trường hợp
- Dùng 1 phần tử làm chốt
  - Mảng MAX phần tử, chỉ dùng đến MAX-1
  - 1 phần tử làm chốt để tránh rơi vào trường hợp front=rear khi đầy
  - Tiện dùng, tuy nhiên lãng phí bộ nhớ cho 1 phần tử
- Cải tiến đơn giản hơn?
  - Dùng 1 biến đếm số lượng phần tử hiện có trong Hàng đợi
  - Chỉ tốn 4 Byte cho biến int

```c
#include <stdio.h>
#define MAX 5

typedef struct {
    int data[MAX];
    int front, rear;
} CircularQueue;

// Khởi tạo hàng đợi
void init(CircularQueue* q) {
    q->front = q->rear = -1;
}

// Kiểm tra rỗng
int isEmpty(CircularQueue* q) {
    return q->front == -1;
}

// Kiểm tra đầy
int isFull(CircularQueue* q) {
    return (q->rear + 1) % MAX == q->front;
}
```

```c
// Thêm phần tử
void enqueue(CircularQueue* q, int value) {
    if (isFull(q)) {
        printf("Queue day, khong the them!\n");
        return;
    }
    if (isEmpty(q)) {
        q->front = q->rear = 0;
    }
    else {
        q->rear = (q->rear + 1) % MAX;
    }
    q->data[q->rear] = value;
}
```

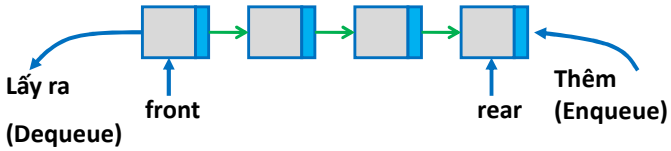


```c
// Lấy phần tử
int dequeue(CircularQueue* q) {
    if (isEmpty(q)) {
        printf("Queue rong, khong the lay phan tu!\n");
        return -1;
    }
    int value = q->data[q->front];
    if (q->front == q->rear) {
        // Sau khi lấy, queue trống hoàn toàn
        q->front = q->rear = -1;
    }
    else {
        q->front = (q->front + 1) % MAX;
    }
    return value;
}
```

```c
// In queue
void printQueue(CircularQueue* q) {
    if (isEmpty(q)) {
        printf("Queue rong!\n");
        return;
    }
    printf("Queue hien tai: ");
    int i = q->front;
    while (1) {
        printf("%d ", q->data[i]);
        if (i == q->rear) break;
        i = (i + 1) % MAX;
    }
    printf("\n");
}
```

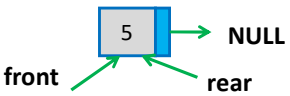
```c
int main() {
    CircularQueue q;
    init(&q);

    enqueue(&q, 1);
    enqueue(&q, 2);
    enqueue(&q, 3);
    enqueue(&q, 4);
    printQueue(&q);

    printf("Lay ra: %d\n", dequeue(&q));
    printf("Lay ra: %d\n", dequeue(&q));
    printQueue(&q);

    enqueue(&q, 5);
    enqueue(&q, 6);
    printQueue(&q);

    enqueue(&q, 7); // báo đầy

    return 0;
}
```


Lưu trữ bằng danh sách móc nối

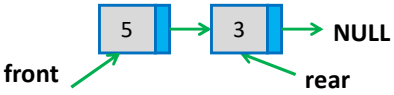


Khởi đầu

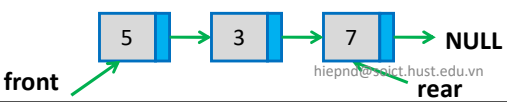
Thêm phần tử thứ nhất



Thêm phần tử thứ hai



Thêm phần tử thứ ba



- **enQueue()** : thêm phần tử vào cuối danh sách móc nối
- **deQueue():** xóa phần tử ở đầu danh sách móc nối

```c
int empty(NODE * front, NODE *rear)
{
   if(front==NULL) return 1;
   else return 0;
}
```

```
int enQueue(NODE *&front, NODE *&rear, int value)
{
    NODE * ptr=(NODE*)malloc(sizeof(NODE));
    if(ptr==NULL)
    {
         printf("Khong con du bo nho !\n");
         return -1;
    }
    ptr->data=value;
    ptr->pNext=NULL;
    if(front==NULL)
    {
        front=ptr;
        rear=front;
    }
    else
    {
        rear->pNext=ptr;
        rear=ptr;
    }
    return 0;
}
```

1. `int deQueue(NODE *&front, NODE *&rear, int &value)`
2. `int front(NODE * front, NODE *rear, int &value)`


```c
#include <stdio.h>
#include <stdlib.h>

// Node trong danh sách liên kết
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Queue quản lý bằng 2 con trỏ: front và rear
typedef struct {
    Node* front;
    Node* rear;
} Queue;

// Khởi tạo queue rỗng
void initQueue(Queue* q) {
    q->front = q->rear = NULL;
}
```

```c
// Kiểm tra queue rỗng
int isEmpty(Queue* q) {
    return (q->front == NULL);
}
```

```c
// In queue
void printQueue(Queue* q) {
    Node* current = q->front;
    printf("Queue hien tai: ");
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

```c
// Thêm phần tử vào queue
void enqueue(Queue* q, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Cap phat bo nho that bai!\n");
        return;
    }
    newNode->data = value;
    newNode->next = NULL;

    if (q->rear == NULL) {
        // Queue đang rỗng
        q->front = q->rear = newNode;
    }
    else {
        q->rear->next = newNode;
        q->rear = newNode;
    }
}
```



```c
// Lấy phần tử ra khỏi queue
int dequeue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue rong!\n");
        return -1;
    }
    Node* temp = q->front;
    int value = temp->data;
    q->front = q->front->next;

    if (q->front == NULL) {
        // Nếu sau khi lấy ra mà queue
        // rỗng thì rear cũng phải NULL
        q->rear = NULL;
    }

    free(temp);
    return value;
}
```

```c
int main() {
    Queue q;
    initQueue(&q);

    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);
    printQueue(&q);

    printf("Lay ra: %d\n", dequeue(&q));
    printQueue(&q);

    enqueue(&q, 40);
    printQueue(&q);

    return 0;
}
```

Queue trong STL C++

- Được cài đặt dùng deque hoặc list

1.empty() – Returns whether the queue is empty.
2.size() – Returns the size of the queue.
3.queue::swap() in C++ STL: Exchange the contents of two queues but the queues must be of same type, although sizes may differ.
4.queue::emplace() in C++ STL: Insert a new element into the queue container, the new element is added to the end of the queue.
5.queue::front() and queue::back() in C++ STL– **front()** function returns a reference to the first element of the queue. **back()** function returns a reference to the last element of the queue.
6.push(g) and pop() – **push()** function adds the element 'g' at the end of the queue. **pop()** function deletes the first element of the queue.

Queue trong STL C++

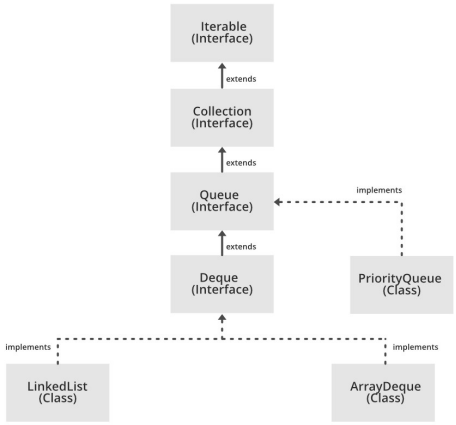
```
// Queue in Standard Template
Library (STL)
#include <iostream>
#include <queue>

using namespace std;

// Print the queue
void showq(queue<int> gq)
{
    queue<int> g = gq;
    while (!g.empty()) {
        cout << '\t' << g.front();
        g.pop();
    }
    cout << '\n';
}
```

```
int main()
{
    queue<int> gquiz;
    gquiz.push(10);
    gquiz.push(20);
    gquiz.push(30);

    cout << "The queue gquiz is : ";
    showq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.front() : " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();

    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showq(gquiz);

    return 0;
}
```

https://www.geeksforgeeks.org/queue-cpp-stl/

Queue trong Java

Modifier and Type	Method and Description
boolean	**add**(E e)Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an IllegalStateException if no space is currently available.
E	**element**()Retrieves, but does not remove, the head of this queue.
boolean	**offer**(E e)Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.
E	**peek**()Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
E	**poll**()Retrieves and removes the head of this queue, or returns null if this queue is empty.
E	**remove**()Retrieves and removes the head of this queue.



https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html

Queue trong Java

```
import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {

    public static void main(String[] args)
    {
        Queue<Integer> q
            = new LinkedList<>();

        // Adds elements {0, 1, 2, 3, 4} to the queue
        for (int i = 0; i < 5; i++)
            q.add(i);

        // Display contents of the queue.
        System.out.println("Elements of queue " + q);

        // To remove the head of queue.
        int removedele = q.remove();
        System.out.println("removed element-" + removedele);

        System.out.println(q);

        // To view the head of queue
        int head = q.peek();
        System.out.println("head of queue-" + head);

        // Rest all methods of collection interface like size and contains
        // can be used with this implementation.
        int size = q.size();
        System.out.println("Size of queue-" + size);
    }
}
```

Ứng dụng

- Hàng đợi công việc trong hệ điều hành: quản lý các tiến trình process
- Quản lý in ấn (Print Queue): quản lý các job in, nhất là máy in trong LAN
- Tính toán lập lịch vòng (Round-robin Scheduling): trong bài toán phân chia thời gian CPU cho các tiến trình
- Bộ đệm dữ liệu (Data Buffer): network buffer, I/O buffer, Video buffer
- Hệ thống hàng đợi trong mạng máy tính: quản lý gửi và nhận các packet
- Xử lý tác vụ đa luồng (Multithreading): thread pool, các job chờ dc xử lý
- Hệ thống xử lý yêu cầu của máy chủ (Server Request Handling): request được xử lý lần lượt, và xếp trong queue để tránh bị drop
- Mô phỏng hàng đợi thực tế (Queue Simulation) : VD mô phỏng mạng giao thông, mô hình mạng,..

Ứng dụng

- Hệ thống viễn thông (Telecommunications): gửi và nhận các gói tin, dữ liệu trên mạng di động
- Xử lý luồng công việc (Workflow Processing)
- Hệ thống phát nhạc trực tuyến (Music Streaming): các bài hát được thêm vào hàng đợi chờ play
- Hệ thống phân phối dữ liệu (Data Streaming)
- Trình xử lý tác vụ nền (Background Task Processing): các tác vụ được thêm vào hàng đợi và sẽ được xử lý khi có CPU rảnh
- Xử lý sự kiện trong lập trình GUI (Event Queue in GUI): người dùng tạo event khi nhấn phím, click chuột,…
- Cấu trúc dữ liệu để tính toán song song (Parallel Processing)

Hàng đợi quản lý công việc in trong LAN

```c
// Cấu trúc công việc in
typedef struct PrintJob {
    char sender[50];
    char documentName[100];
    int pageCount;
    char timestamp[20];
    struct PrintJob* next;
} PrintJob;

// Queue máy in
typedef struct {
    PrintJob* front;
    PrintJob* rear;
} PrintQueue;
```

```c
// Khởi tạo queue
void initQueue(PrintQueue* q) {
    q->front = q->rear = NULL;
}

// Kiểm tra queue rỗng
int isEmpty(PrintQueue* q) {
    return (q->front == NULL);
}
```



```c
// Thêm công việc in vào queue
void enqueue(PrintQueue* q, const char* sender, const char* docName, int pages,
const char* time) {
    PrintJob* newJob = (PrintJob*)malloc(sizeof(PrintJob));
    if (!newJob) {
        printf("Khong du bo nho!\n");
        return;
    }
    strcpy(newJob->sender, sender);
    strcpy(newJob->documentName, docName);
    newJob->pageCount = pages;
    strcpy(newJob->timestamp, time);
    newJob->next = NULL;

    if (q->rear == NULL) {
        q->front = q->rear = newJob;
    }
    else {
        q->rear->next = newJob;
        q->rear = newJob;
    }
}
```

```c
// Xử lý (in) công việc đầu tiên trong queue
void dequeue(PrintQueue* q) {
    if (isEmpty(q)) {
        printf("Khong co cong viec nao de in!\n");
        return;
    }
    PrintJob* temp = q->front;
    printf("Dang in: %s tu %s, %d trang, gui luc %s\n",
        temp->documentName, temp->sender, temp->pageCount, temp->timestamp);

    q->front = q->front->next;
    if (q->front == NULL)
        q->rear = NULL;

    free(temp);
}
```


Queue – Hàng đợi

- Ví dụ 1. Cài đặt các thao tác của hàng đợi dùng 2 stack
- Ví dụ 2. Cài đặt các thao tác của Stack dùng 2 hàng đợi
- Ví dụ 3. Cài đặt k hàng đợi chỉ dùng 1 mảng
- Ví dụ 4. Các bài toán duyệt trên cây, đồ thị dùng với queue
- Ví dụ 5. Đảo ngược hàng đợi
  Gợi ý: Dùng thêm Stack
- Ví dụ 6. Cho dãy n phần tử, hãy tìm đoạn con (các phần tử liên tiếp) kích thước k có tổng lớn nhất
- Ví dụ 7. Tìm quãng đường từ ô hiện tại tới ô giá trị 1 gần nhất theo hàng hoặc cột trong ma trận nhị phân

```
Input : N = 3, M = 4
        mat[][] = { 0, 0, 0, 1,
                    0, 0, 1, 1,
                    0, 1, 1, 0 }
Output : 3 2 1 0
         2 1 0 0
         1 0 0 1
```

Hàng đợi (Queue)

- Bài toán rót nước
  - Có 1 bể chứa nước (vô hạn)
  - Có 2 cốc với dung tích là a, b (nguyên dương) lít
  - Tìm cách đong đúng c (nguyên dương) lít nước

- Bài toán rót nước: a = 6, b = 8, c = 4

Bước	Thực hiện	Trạng thái
1	Đổ đầy nước vào cốc 1	(6,0)
2	Đổ hết nước từ cốc 1 sang cốc 2	(0,6)
3	Đổ đầy nước vào cốc 1	(6,6)
4	Đổ nước từ cốc 1 vào đầy cốc 2	(4,8)

Hàng đợi (Queue)

- Bài toán rót nước: a = 4, b = 19, c = 21

Bước	Thực hiện	Trạng thái
1	Đổ đầy nước vào cốc 1	(4,0)
2	Đổ hết nước từ cốc 1 sang cốc 2	(0,4)
3	Đổ đầy nước vào cốc 1	(4,4)
4	Đổ hết nước từ cốc 1 sang cốc 2	(0,8)
5	Đổ đầy nước vào cốc 1	(4,8)
6	Đổ hết nước từ cốc 1 sang cốc 2	(0,12)
7	Đổ đầy nước vào cốc 1	(4,12)
8	Đổ hết nước từ cốc 1 sang cốc 2	(0,16)
9	Đổ đầy nước vào cốc 1	(4,16)
10	Đổ nước từ cốc 1 vào đầy cốc 2	(1,19)
11	Đổ hết nước ở cốc 2 ra ngoài	(1,0)

Hàng đợi (Queue)

- Bài toán rót nước: a = 4, b = 19, c = 21 (tiếp)

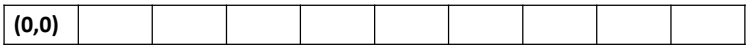
Bước	Thực hiện	Trạng thái
12	Đổ hết nước từ cốc 1 sang cốc 2	(0,1)
13	Đổ đầy nước vào cốc 1	(4,1)
14	Đổ hết nước từ cốc 1 sang cốc 2	(0,5)
15	Đổ đầy nước vào cốc 1	(4,5)
16	Đổ hết nước từ cốc 1 sang cốc 2	(0,9)
17	Đổ đầy nước vào cốc 1	(4,9)
18	Đổ hết nước từ cốc 1 sang cốc 2	(0,13)
19	Đổ đầy nước vào cốc 1	(4,13)
20	Đổ hết nước từ cốc 1 sang cốc 2	(0,17)
21	Đổ đầy nước vào cốc 1	(4,17)

Hàng đợi (Queue)

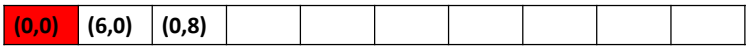
- Thiết kế thuật toán và cấu trúc dữ liệu
  - Trạng thái là bộ (x, y): lượng nước có trong cốc 1 và 2
  - Trạng thái ban đầu (0, 0)
  - Trạng thái kết thúc: $x = c$ hoặc $y = c$ hoặc $x + y = c$
  - Chuyển trạng thái
    - (1) Đổ đầy nước từ bể vào cốc 1: (a, y)
    - (2) Đổ đầy nước từ bể vào cốc 2: (x, b)
    - (3) Đổ hết nước từ cốc 1 ra ngoài: (0, y)
    - (4) Đổ hết nước từ cốc 2 ra ngoài: (x, 0)
    - (5) Đổ nước từ cốc 1 vào đầy cốc 2: ($x + y - b$, b), nếu $x + y \geq b$
    - (6) Đổ hết nước từ cốc 1 sang cốc 2: (0, $x + y$), nếu $x + y \leq b$
    - (7) Đổ nước từ cốc 2 vào đầy cốc 1: (a, $x + y - a$), nếu $x + y \geq a$
    - (8) Đổ hết nước từ cốc 2 sang cốc 1: ($x + y$, 0), nếu $x + y \leq a$

Hàng đợi (Queue)

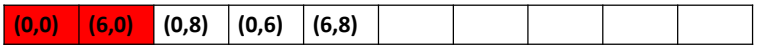
- Đưa (0,0) vào hàng đợi



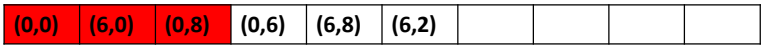
- Lấy (0,0) ra và đưa (6,0), (0,8) vào hàng đợi



- Lấy (6,0) ra và đưa (0,6) và (6,8) vào hàng đợi

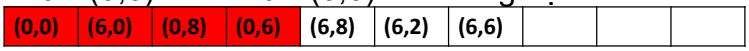


- Lấy (0,8) ra và đưa (6,2) vào hàng đợi

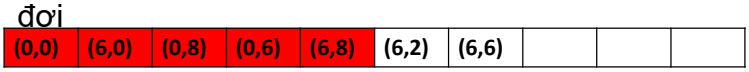


Hàng đợi (Queue)

- Đưa (0,6) ra và đưa (6,6) vào hàng đợi

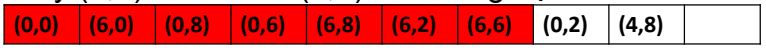


- Lấy (6,8) ra và không đưa trạng thái mới nào vào hàng đợi



- Lấy (6,2) ra và đưa (0,2) vào hàng đợi

- Lấy (6,6) ra và đưa (4,8) vào hàng đợi



Hàng đợi (Queue)

- Thiết kế thuật toán và cấu trúc dữ liệu
  - Hàng đợi Q để ghi nhận các trạng thái được sinh ra
  - Mảng 2 chiều để đánh dấu trạng thái đã được xét đến
    - visited[*x*][*y*] = true, nếu trạng thái (*x*, *y*) đã được sinh ra
  - Ngăn xếp để in ra chuỗi các hành động để đạt được kết quả
  - Danh sách L để lưu các con trỏ trỏ đến các vùng nhớ được cấp phát động (phục vụ cho việc thu hồi bộ nhớ khi kết thúc chương trình)

```
#include <stdio.h>
#include <stdlib.h>
#include <queue>
#include <stack>
#include <list>
using namespace std;
struct State{
  int x;
  int y;
  char* msg;// action to generate current state
  State* p;// pointer to the state generating current state
};
bool visited[10000][10000];
queue<State*> Q;
list<State*> L;
State* target;
int a,b,c;
```


Hàng đợi (Queue)

- Các hàm khởi tạo mảng đánh dấu, đánh dấu trạng thái, kiểm tra trạng thái đích, giải phóng bộ nhớ

```
void initVisited(){
  for(int x = 0; x < 10000; x++)
    for(int y = 0; y < 10000; y++)
      visited[x][y] = false;
}
bool reachTarget(State* S){
    return S->x == c || S->y == c ||
       S->x + S->y == c;
}
void markVisit(State* S){
    visited[S->x][S->y] = true;
}
void freeMemory(){
    list<State*>::iterator it;
    for(it = L.begin(); it != L.end(); it++){
        delete *it;
    }
}
```

Hàng đợi (Queue)

- Hàm sinh trạng thái bởi hành động (3): đổ hết nước từ cốc 1 ra ngoài

```
bool genMove1Out(State* S){
    if(visited[0][S->y]) return false;
    State* newS = new State;
    newS->x = 0;
    newS->y = S->y;
    newS->msg = "Do het nuoc o coc 1 ra ngoai";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```


Hàng đợi (Queue)

- Hàm sinh trạng thái bởi hành động (4): đổ hết nước từ cốc 2 ra ngoài

```
bool genMove2Out(State* S){
    if(visited[S->x][0]) return false;
    State* newS = new State;
    newS->x = S->x;
    newS->y = 0;
    newS->msg = "Do het nuoc o coc 2 ra ngoai";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```

Hàng đợi (Queue)

- Hàm sinh trạng thái bởi hành động (5): đổ nước từ cốc 1 vào đầy cốc 2

```
bool genMove1Full2(State* S){
    if(S->x+S->y < b) return false;
    if(visited[S->x + S->y - b][b]) return false;
    State* newS = new State;
    newS->x = S->x + S->y - b;
    newS->y = b;
    newS->msg = "Do nuoc tu coc 1 vao day coc 2";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```


Hàng đợi (Queue)

- Hàm sinh trạng thái bởi hành động (7): đổ nước từ cốc 2 vào đầy cốc 1

```
bool genMove2Full1(State* S){
    if(S->x+S->y < a) return false;
    if(visited[a][S->x + S->y - a]) return false;
    State* newS = new State;
    newS->x = a;
    newS->y = S->x + S->y - a;
    newS->msg = "Do nuoc tu coc 2 vao day coc 1";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```

Hàng đợi (Queue)

- Hàm sinh trạng thái bởi hành động (6): đổ hết nước từ cốc 1 sang cốc 2

```
bool genMoveAll12(State* S){
    if(S->x + S->y > b) return false;
    if(visited[0][S->x + S->y]) return false;
    State* newS = new State;
    newS->x = 0;
    newS->y = S->x + S->y;
    newS->msg = "Do het nuoc tu coc 1 sang coc 2";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```


Hàng đợi (Queue)

- Hàm sinh trạng thái bởi hành động (8): đổ hết nước từ cốc 2 sang cốc 1

```
bool genMoveAll21(State* S){
    if(S->x + S->y > a) return false;
    if(visited[S->x + S->y][0]) return false;
    State* newS = new State;
    newS->x = S->x + S->y;
    newS->y = 0;
    newS->msg = "Do het nuoc tu coc 2 sang coc 1";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```

Hàng đợi (Queue)

- Hàm sinh trạng thái bởi hành động (1): đổ đầy nước từ bể vào cốc 1

```
bool genMoveFill1(State* S){
    if(visited[a][S->y]) return false;
    State* newS = new State;
    newS->x = a;
    newS->y = S->y;
    newS->msg = "Do day nuoc vao coc 1";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;
}
```


Hàng đợi (Queue)

- Hàm sinh trạng thái bởi hành động (2): đổ đầy nước từ bể vào cốc 2

```
bool genMoveFill2(State* S){
    if(visited[S->x][b]) return false;
    State* newS = new State;
    newS->x = S->x;
    newS->y = b;
    newS->msg = "Do day nuoc vao coc 2";
    newS->p = S;
    Q.push(newS); markVisit(newS);
    L.push_back(newS);
    if(reachTarget(newS)){
        target = newS;
        return true;
    }
    return false;

}
```

Hàng đợi (Queue)

- Hàm in chuỗi hành động để thu được kết quả

```
void print(State* target){
    printf("-----------RESULT------------\n");
    if(target == NULL)
       printf("Khong co loi giai!!!!!!!");
    State* currentS = target;
    stack<State*> actions;
    while(currentS != NULL){
        actions.push(currentS);
        currentS = currentS->p;
    }
    while(actions.size() > 0){
        currentS = actions.top();
        actions.pop();
        printf("%s, (%d,%d)\n",
             currentS->msg,currentS->x,
                 currentS->y);
    }
}
```


Hàng đợi (Queue)

- Khởi tạo, sinh trạng thái ban đầu và đưa vào hàng đợi
- Tại mỗi bước, lấy 1 trạng thái ra khỏi hàng đợi, sinh trạng thái mới và đưa vào hàng đợi

```
void solve(){
  initVisited();
  // sinh ra trang thai ban dau (0,0) va dua vao Q
  State* S = new State;
  S->x = 0; S->y = 0; S->p = NULL;
  Q.push(S); markVisit(S);
  while(!Q.empty()){
    State* S = Q.front(); Q.pop();
    if(genMove1Out(S)) break;
    if(genMove2Out(S)) break;
    if(genMove1Full2(S)) break;
    if(genMoveAll12(S)) break;
    if(genMove2Full1(S)) break;
    if(genMoveAll21(S)) break;
    if(genMoveFill1(S)) break;
    if(genMoveFill2(S)) break;
  }
}
```

Hàng đợi (Queue)

- Hàm main
  - Thử nghiệm với bộ dữ liệu: a = 4, b = 7, c = 9

```
int main(){
    a = 4;
    b = 7;
    c = 9;
    target = NULL;
    solve();
    print(target);
    freeMemory();
}
```


Priority queue – hàng đợi ưu tiên

Hàng đợi mở rộng

- Hàng đợi ưu tiên – Priority Queue
  - Mỗi phần tử có thêm độ ưu tiên
  - Ưu tiên xử lý phần tử có độ ưu tiên cao nhất trước (thao tác deQueue)
  - Nếu các phần tử có cùng độ ưu tiên → phần tử vào trước sẽ được xử lý trước
- Ví dụ. Các phần tử có độ ưu tiên là số nguyên
  enQueue(A,5)
  enQueue(B,7)
  enQueue(C,4)
  enQueue(D,5)
  Khi lấy ra sẽ lần lượt nhận là B, A , D và C

Hàng đợi ưu tiên – Priority Queue

- Cài đặt hàng đợi ưu tiên
  - Dùng mảng hoặc Danh sách liên kết?
    - Thời gian thực hiện thao tác thêm/lấy ra có thể lên tới O(n) (tùy theo cách cài đặt)
    - VD. Nếu mặc định đầu hàng là phần tử có độ ưu tiên cao nhất → lấy ra O(1), thêm vào O(n)
  - Dùng cấu trúc heap – sẽ gặp trong phần sắp xếp
    - Thời gian thực hiện các thao tác chỉ cỡ O(logn)
    - Có nhiều biến thể của heap

Operation	find-min	delete-min	insert	decrease-key	meld
Binary[5]	Θ(1)	Θ(log *n*)	*O*(log *n*)	*O*(log *n*)	Θ(*n*)
Leftist	Θ(1)	Θ(log *n*)	Θ(log *n*)	*O*(log *n*)	Θ(log *n*)
Binomial[5][6]	Θ(1)	Θ(log *n*)	Θ(1)[a]	Θ(log *n*)	*O*(log *n*)[b]
Fibonacci[5][7]	Θ(1)	*O*(log *n*)[a]	Θ(1)	Θ(1)[a]	Θ(1)
Pairing[8]	Θ(1)	*O*(log *n*)[a]	Θ(1)	*o*(log *n*)[a][c]	Θ(1)
Brodal[11][d]	Θ(1)	*O*(log *n*)	Θ(1)	Θ(1)	Θ(1)
Rank-pairing[13]	Θ(1)	*O*(log *n*)[a]	Θ(1)	Θ(1)[a]	Θ(1)
Strict Fibonacci[14]	Θ(1)	*O*(log *n*)	Θ(1)	Θ(1)	Θ(1)
2–3 heap[15]	*O*(log *n*)	*O*(log *n*)[a]	*O*(log *n*)[a]	Θ(1)	?

Thời gian thực hiện với min-heap

Hàng đợi cấp cứu – hàng đợi ưu tiên

```c
// Cấu trúc bệnh nhân
typedef struct Patient {
    char name[50];
    int priority; // Mức độ nghiêm trọng: số nhỏ hơn thì ưu tiên cao hơn
    struct Patient* next;
} Patient;

// Hàng đợi ưu tiên
typedef struct {
    Patient* front;
} PriorityQueue;

// Khởi tạo queue
void initQueue(PriorityQueue* q) {
    q->front = NULL;
}
```



```c
// Thêm bệnh nhân vào queue theo đúng thứ tự ưu tiên
void enqueue(PriorityQueue* q, const char* name, int priority) {
    Patient* newPatient = (Patient*)malloc(sizeof(Patient));
    if (!newPatient) {
        printf("Khong du bo nho!\n");
        return;
    }
    strcpy(newPatient->name, name);
    newPatient->priority = priority;
    newPatient->next = NULL;

    // Nếu queue rỗng hoặc bệnh nhân mới có độ ưu tiên cao hơn front
    if (q->front == NULL || priority < q->front->priority) {
        newPatient->next = q->front;
        q->front = newPatient;
    }
    else {
        // Tìm đúng vị trí để chèn
        Patient* current = q->front;
        while (current->next != NULL && current->next->priority <= priority) {
            current = current->next;
        }
        newPatient->next = current->next;
        current->next = newPatient;
    }
}
```

```c
void dequeue(PriorityQueue* q) {
    if (q->front == NULL) {
        printf("Khong co benh nhan trong danh sach!\n");
        return;
    }
    Patient* temp = q->front;
    printf("Dang kham cho: %s (do uu tien: %d)\n", temp->name, temp->priority);
    q->front = q->front->next;
    free(temp);
}

// In danh sách bệnh nhân
void printQueue(PriorityQueue* q) {
    Patient* current = q->front;
    printf("Danh sach benh nhan dang cho:\n");
    while (current != NULL) {
        printf("- %s (do uu tien: %d)\n", current->name, current->priority);
        current = current->next;
    }
}
```


Hàng đợi cấp cứu – hàng đợi ưu tiên

```c
int main() {
    PriorityQueue q;
    initQueue(&q);

    enqueue(&q, "Alice", 3);    // mức độ nhẹ
    enqueue(&q, "Bob", 1);      // nặng (ưu tiên cao)
    enqueue(&q, "Charlie", 2);  // trung bình
    enqueue(&q, "Trump", 1);  // trung bình

    printQueue(&q);

    dequeue(&q); // Khám Bob
    dequeue(&q); // Khám Trump

    printQueue(&q);

    return 0;
}
```

Priority Queue trong STL C++

Methods of priority queue are:

•priority_queue::empty() in C++ STL– **empty()** function returns whether the queue is empty.
•priority_queue::size() in C++ STL– **size()** function returns the size of the queue.
•priority_queue::top() in C++ STL– Returns a reference to the top most element of the queue
•priority_queue::push() in C++ STL– **push(g)** function adds the element 'g' at the end of the queue.
•priority_queue::pop() in C++ STL– **pop()** function deletes the first element of the queue.
•priority_queue::swap() in C++ STL– This function is used to swap the contents of one priority queue with another priority queue of same type and size.
•priority_queue::emplace() in C++ STL – This function is used to insert a new element into the priority queue container.
•priority_queue value_type in C++ STL– Represents the type of object stored as an element in a priority_queue. It acts as a synonym for the template parameter.

Priority Queue trong STL C++

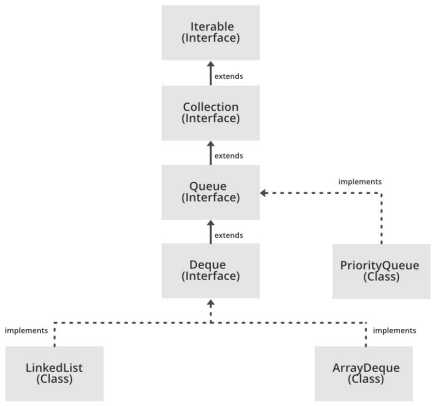
```
#include <iostream>
#include <queue>

using namespace std;

void showpq(priority_queue<int> gq)
{
    priority_queue<int> g = gq;
    while (!g.empty()) {
        cout << '\t' << g.top();
        g.pop();
    }
    cout << '\n';
}
```

```
int main()
{
    priority_queue<int> gquiz;
    gquiz.push(10);
    gquiz.push(30);
    gquiz.push(20);
    gquiz.push(5);
    gquiz.push(1);

    cout << "The priority queue gquiz is : ";
    showpq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.top() : " << gquiz.top();

    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showpq(gquiz);

    return 0;
}
```

```
The priority queue gquiz is :     30    20    10    5    1

gquiz.size() : 5
gquiz.top() : 30
gquiz.pop() :     20    10    5    1
```

Priority Queue trong Java

```
import java.util.*;

class PriorityQueueDemo {

      // Main Method
    public static void main(String args[])
    {
        // Creating empty priority queue
        PriorityQueue<Integer> pQueue = new PriorityQueue<Integer>();

        // Adding items to the pQueue using add()
        pQueue.add(10);
        pQueue.add(20);
        pQueue.add(15);

        // Printing the top element of PriorityQueue
        System.out.println(pQueue.peek());

        // Printing the top element and removing it
        // from the PriorityQueue container
        System.out.println(pQueue.poll());

        // Printing the top element again
        System.out.println(pQueue.peek());
    }
}
```



Hàng đợi ưu tiên – Priority Queue

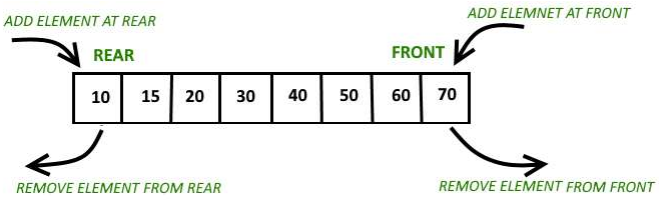
- Ứng dụng của hàng đợi ưu tiên
  - Trong quản lý message, CPU Scheduling, resources…
  - Quản lý băng thông mạng
  - Mô phỏng các sự kiện rời rạc
  - Dùng trong cài đặt thuật toán Dijkstra, Huffman coding, Prim,…
  - Thuật toán A* Search
  - Hệ thống hàng đợi sự kiện (Event-Driven Systems)
  - Hệ thống cứu hộ và dịch vụ khẩn cấp
  - Hệ thống ngân hàng và tài chính
  - Hệ thống quản lý công việc
  - Quản lý người chơi, tài nguyên trong các hệ thống MMORPG

Priority Queue – Hàng đợi ưu tiên

- Ví dụ 1. Tìm phần tử lớn thứ k trong dãy n phần tử
- Ví dụ 2. Cho 1 stream các số nguyên xuất hiện liên tục, hãy đưa ra các số hay xuất hiện gần đây nhất
- Ví dụ 3. có k dãy số nguyên đã có thứ tự, hãy trộn các dãy đó để thu được 1 dãy cũng có thứ tự

Deque hàng đợi 2 đầu

Hàng đợi 2 đầu- deque



- Hàng đợi 2 đầu – deque (*double-ended queue)*
  - Có thể thêm vào và lấy ra ở cả 2 đầu
  - Các thao tác tương tự với queue
- Các thao tác căn bản của Deque:
  - insertFront(): Chèn thêm một phần tử mới vào đầu Deque
  - insertLast(): Chèn thêm một phần tử mới vào cuối Deque
  - deleteFront(): Xóa đi một phần tử nằm ở đầu Deque
  - deleteLast(): Xóa đi một phần tử nằm ở cuối Deque
- Ngoài ra còn có:
  - getFront(): Lấy ra phần tử nằm ở đầu Deque
  - getRear(): Lấy ra phần tử nằm ở cuối Deque
  - isEmpty(): Kiểm tra xem liệu rằng Deque có trống hay không
  - isFull() Kiểm tra xem liệu rằng Deque đã đầy chưa.

Hàng đợi 2 đầu- deque

- Ứng dụng của Deque:
  - Dùng để cài đặt cho cả Stack và Queue
  - Có thể truy cập các phần tử theo thứ tự xuôi hoặc ngược đều được, T(n)=O(1)
- Cài đặt deque
  - Dùng danh sách liên kết đôi
  - Dùng mảng “vòng”
  - Thời gian thực hiện các thao tác O(1)

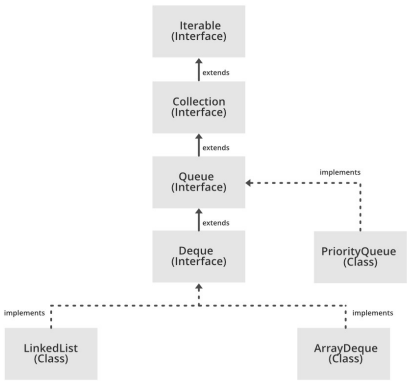
Hàng đợi 2 đầu- deque

- Deque trong STL

```cpp
#include <iostream>
#include <deque>
using namespace std;
void showdq(deque <int> g)
{
    deque <int> :: iterator it;
    for (it = g.begin(); it != g.end(); ++it)
        cout << '\t' << *it;
    cout << '\n';
}
int main()
{
    deque <int> gquiz;
    gquiz.push_back(10);
    gquiz.push_front(20);
    gquiz.push_back(30);
    gquiz.push_front(15);
    cout << "The deque gquiz is : ";
    showdq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.max_size() : " << gquiz.max_size();

    cout << "\ngquiz.at(2) : " << gquiz.at(2);
    cout << "\ngquiz.front() : " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();

    cout << "\ngquiz.pop_front() : ";
    gquiz.pop_front();
    showdq(gquiz);

    cout << "\ngquiz.pop_back() : ";
    gquiz.pop_back();
    showdq(gquiz);

    return 0;
}
```


Hàng đợi 2 đầu- deque

- Deque trong java




```java
import java.util.*;

public class DequeExample {
    public static void main(String[] args)
    {
        Deque<String> deque
            = new LinkedList<String>();

        // We can add elements to the queue
        // in various ways

        // Add at the last
        deque.add("Element 1 (Tail)");

        // Add at the first
        deque.addFirst("Element 2 (Head)");

        // Add at the last
        deque.addLast("Element 3 (Tail)");

        // Add at the first
        deque.push("Element 4 (Head)");

        // Add at the last
        deque.offer("Element 5 (Tail)");

        // Add at the first
        deque.offerFirst("Element 6 (Head)");

        System.out.println(deque + "\n");

        // We can remove the first element
        // or the last element.
        deque.removeFirst();
        deque.removeLast();
        System.out.println("Deque after removing "
                           + "first and last: " + deque);
    }
}
```

Hàng đợi 2 đầu- deque

- ArrayDeque sử dụng mảng động:
  - Nhanh hơn khi không cần resize nhiều lần và không hỗ trợ null values.
  - Hiệu quả về mặt hiệu suất (trong hầu hết các trường hợp).
- LinkedList sử dụng danh sách liên kết đôi
  - Phù hợp nếu bạn cần thêm/xóa liên tục các phần tử từ giữa deque.
  - linh hoạt hơn trong việc quản lý phần tử, đặc biệt với việc xử lý danh sách liên kết.

Hàng đợi 2 đầu- deque

- Sliding Window (Cửa sổ trượt): tìm đoạn con k phần tử theo thứ tự tăng, tổng lớn nhất,…
  - Phân tích giá cổ phiếu, chỉ số kinh tế,…
- Undo/Redo (Hoàn tác và làm lại)
  - Undo: Khi người dùng muốn hoàn tác (undo), các thao tác gần nhất có thể được truy cập từ đầu deque.
  - Redo: Các thao tác làm lại (redo) có thể được truy cập từ cuối deque.
- Hệ thống quản lý tác vụ
  - Các tiến trình quan trọng có thể được thêm vào đầu deque (để được xử lý trước), trong khi các tiến trình ít quan trọng hơn có thể được thêm vào cuối.