

## Chương 3 : Các cấu trúc dữ liệu cơ bản

Trịnh Anh Phúc <sup>1</sup>

<sup>1</sup>Bộ môn Khoa Học Máy Tính, Viện CNTT & TT,  
Trường Đại Học Bách Khoa Hà Nội.

Ngày 20 tháng 3 năm 2015

# Giới thiệu

## 1 Các khái niệm

- Kiểu dữ liệu trừu tượng
- Cấu trúc dữ liệu
- Con trỏ

## 2 Mảng

## 3 Danh sách

- Định nghĩa
- Các cách cài đặt danh sách tuyến tính

## 4 Ngăn xếp

- Định nghĩa
- Các cách cài đặt ngăn xếp
- Ngăn xếp và đệ quy
- Ứng dụng

## 5 Hàng đợi

- Định nghĩa
- Các cách cài đặt hàng đợi
- Ứng dụng

## 6 Tổng kết

# Các khái niệm

## Kiểu dữ liệu

Các kiểu dữ liệu được đặc trưng bởi

- Tập các giá trị
- Cách biểu diễn dữ liệu được sử dụng chung cho tất cả các giá trị
- Tập các phép toán có thể thực hiện trên tất cả các giá trị này.

Ví dụ các kiểu dữ liệu trong C

Kiểu	Bits	Giá trị nhỏ nhất	Giá trị lớn nhất
char	8	-128	127
short	16	-32768	32767
unsigned int	16	0	65535
long	32	$-2^{31}$	$2^{31} - 1$
float	32	$-3.4 \times 10^{38}$	$3.4 \times 10^{38}$
double	64	$-1.7 \times 10^{308}$	$1.7 \times 10^{308}$

# Các khái niệm

## Kiểu dữ liệu trừu tượng

Kiểu dữ liệu trừu tượng bao gồm :

- Tập các giá trị
- Tập các phép toán có thể thực hiện trên tất cả các giá trị này.

Rõ ràng không có cách biểu diễn dữ liệu chung cho dữ liệu trừu tượng

Kiểu	Đối tượng	Phép toán
Mảng	các phần tử	khởi tạo (create), chèn (insert), ...
Danh sách	các phần tử	chèn (insert), xóa (delete), tìm (search), ...
Đồ thị	đỉnh, cạnh	duyet (traverse), tìm đường (search path), ...
Ngăn xếp	các phần tử	gấp (pop), ấn (push), kiểm tra rỗng, ...
Hàng đợi	các phần tử	vào hàng (enqueue), ra khỏi hàng (dequeue), ...
Cây	gốc, lá, cành	duyet (traverse), tìm kiếm (search), ...

## Cấu trúc dữ liệu

Định nghĩa : Cấu trúc dữ liệu là một họ các biến, có thể có kiểu dữ liệu khác nhau, được liên kết lại theo một cách thức nào đó.

- **Ô** (cell) là đơn vị cơ sở cấu thành cấu trúc dữ liệu. Có thể hình dung ô như cái hộp đựng giá trị phát sinh từ một kiểu dữ liệu cơ bản hay phức hợp.
- Cấu trúc dữ liệu đc tạo nhờ đặt tên cho một **nhóm** (group) các ô và đặt giá trị cho một số ô để mô tả sự *liên kết* giữa các ô.

# Các khái niệm

## Cấu trúc dữ liệu (tiếp)

Có ba phương pháp tạo nhóm

- Dùng mảng là một dãy các ô có cùng kiểu dữ liệu xác định  
e.g. **int name[100]**
- Dùng cấu trúc bản ghi là ô được tạo bởi một họ các ô (gọi là các trường) có thể có kiểu rất khác nhau.

```
struct record{  
    float data;  
    int next;} reclist[100];
```

- Dùng file là giống mảng tuy nhiên các phần tử của nó chỉ truy cập được một cách tuần tự.

## Con trỏ (pointer)

Định nghĩa : **Con trỏ** (pointer) là ô mà giá trị của nó chỉ sang một ô khác.



Khi vẽ cấu trúc dữ liệu sử dụng con trỏ, để thể hiện ô A trỏ sang ô B, ta sẽ sử dụng mũi tên hướng từ A đến B.

**Ví dụ** : Để khai báo con trỏ ptr trong C trỏ đến ô có kiểu dữ liệu cho trước tên là celltype

**celltype \*ptr**

## Phân loại các cấu trúc dữ liệu

Thông thường cách phân loại trong các sách dạy CTDL&GT

- **Cấu trúc dữ liệu cơ sở** (Base data structures) : int, char, float, double
- **Cấu trúc dữ liệu tuyến tính** (Linear data structures) : mảng, danh sách, hàng đợi, ngăn xếp...
- **Cấu trúc dữ liệu phi tuyến** (Non linear data structures) : cây, đồ thị, bảng băm ...



## 1 Các khái niệm

- Kiểu dữ liệu trừu tượng
- Cấu trúc dữ liệu
- Con trỏ

## 2 Mảng

## 3 Danh sách

- Định nghĩa
- Các cách cài đặt danh sách tuyến tính

## 4 Ngăn xếp

- Định nghĩa
- Các cách cài đặt ngăn xếp
- Ngăn xếp và đệ qui
- Ứng dụng

## 5 Hàng đợi

- Định nghĩa
- Các cách cài đặt hàng đợi
- Ứng dụng

## 6 Tổng kết

## Mảng

Mảng là dữ liệu trừu tượng

- Tập các giá trị : tập các cặp  $\langle index, value \rangle$  trong đó với mỗi giá trị của  $index$  có một giá trị từ tập các giá trị.
- Các thao tác cơ bản :
  - ▶ Khởi tạo
  - ▶ Chèn phần tử
  - ▶ Xóa bỏ một phần tử

## Phân bổ bộ nhớ cho mảng

Thông thường mảng được chiếm giữ một dãy liên tiếp các từ máy trong bộ nhớ, cần lưu ý khái niệm từ máy trong mỗi ngôn ngữ lập trình là khác nhau và kích thước khác nhau nên tính toán việc phân bổ này không thể đồng nhất cho mọi ngôn ngữ.

## Ví dụ

```
# include <stdio.h>
# include <conio.h>
int main(){
int one[] = {0,1,2,3,4};
int *ptr; int rows = 5;
/* in địa chỉ của mảng một chiều dùng con trỏ */
int i; ptr = one;
for(i=0;i<rows;i++)
    printf("%8u %5d",ptr+i,* (ptr+i));
}
```

Tuy vậy, khi dùng hai trình dịch DEV C và TC lại cho kết quả khác nhau vì kích thước của dữ liệu **int** lần lượt là : 4, 2.

## Các thao tác với mảng

- Chèn một phần tử vào mảng : Do mảng có kích thước xác định nên nếu ta muốn chèn một phần tử mới vào thì ta phải dịch các phần tử phía sau ô được đánh dấu để đảm bảo trình tự của mảng
- Xóa bỏ một phần tử : Ngược lại khi xóa bỏ một phần tử thì ta dồn các phần tử còn lại sau chỗ đánh dấu lên.

## 1 Các khái niệm

- Kiểu dữ liệu trừu tượng
- Cấu trúc dữ liệu
- Con trỏ

## 2 Mảng

## 3 Danh sách

- Định nghĩa
- Các cách cài đặt danh sách tuyến tính

## 4 Ngăn xếp

- Định nghĩa
- Các cách cài đặt ngăn xếp
- Ngăn xếp và đệ qui
- Ứng dụng

## 5 Hàng đợi

- Định nghĩa
- Các cách cài đặt hàng đợi
- Ứng dụng

## 6 Tổng kết

## Danh sách tuyến tính

Danh sách tuyến tính là một dãy gồm không hoặc nhiều hơn các phần tử cùng kiểu cho trước :  $(a_1, a_2, \dots, a_n) \quad n \geq 0$

- $a_i$  là một phần tử của danh sách.
- $a_1$  là phần tử đầu tiên còn  $a_n$  là phần tử cuối cùng.
- $n$  là độ dài của danh sách.

Khi  $n = 0$  ta có danh sách rỗng, các phần tử được sắp xếp một cách tuyến tính hay  $a_i$  trước  $a_{i+1}$  và sau  $a_{i-1}$ .

Ví dụ :

- Danh sách các sinh viên được sắp theo tên
- Danh sách các cầu thủ có số áo tăng dần

## Danh sách tuyến tính (tiếp)

Các thao tác cơ bản :

- Khởi tạo
- Chèn phần tử
- Định vị trí của một phần tử
- Truy xuất một phần tử
- Xóa bỏ một phần tử
- Trả lại vị trí sau vị trí  $p$
- Trả lại vị trí trước vị trí  $p$
- Trả lại vị trí đầu tiên
- Tạo mảng rỗng
- In ra danh sách các phần tử



## Các cách cài đặt danh sách tuyến tính

Có ba cách cài đặt danh sách tuyến tính cơ bản sau

- Dùng mảng (Array list)
  - ▶ Cát giữ các phần tử của danh sách vào các ô liên tiếp của mảng.
- Danh sách liên kết (Linked list)
  - ▶ Các phần tử được cất giữ tại các chỗ khác nhau trong bộ nhớ.
  - ▶ Mỗi phần tử có một con trỏ (pointer) trỏ đến phần tử tiếp theo.
- Địa chỉ không trực tiếp (indirect addressing)
  - ▶ Các phần tử của danh sách có thể đc cất giữ các chỗ tùy ý trong bộ nhớ.
  - ▶ Tạo bảng các địa chỉ trong đó phần tử thứ  $i$  sẽ cho biết địa chỉ lưu trữ phần tử  $a_i$ .

## Cài đặt danh sách tuyến tính : dùng mảng

Ta cất giữ các phần tử của danh sách vào các ô liên tiếp của mảng. Vậy danh sách là cấu trúc gồm hai thành phần

- 1 là mảng gồm các phần tử.
- 2 last - vị trí của phần tử cuối cùng trong danh sách.

Vị trí có kiểu nguyên và chạy trong khoảng từ 0 đến độ dài mảng -1.

Phần tử đầu tiên	← first
Phần tử thứ hai	
⋮	
⋮	
Phần tử cuối cùng	← last
rỗng	
rỗng	

## Định nghĩa cấu trúc dữ liệu danh sách dùng mảng

Mã nguồn ngôn ngữ C cài đặt danh sách dùng mảng

```
#define MAXLEN 100  
typedef int element__type;  
typedef struct LIST{  
    element__type elements[MAXLEN];  
    int last;  
}list__type;
```

# Danh sách dùng mảng



## Thao tác chèn

Mã giả của thao tác chèn phần tử  $x$  vào danh sách  $L$  tại vị trí  $p$

**Procedure** Insert( $x$ ,  $p$ ,  $L$ )

- 1 **if** ( $p > L.last \geq MAXLEN$ ) **then** <Báo lỗi tràn danh sách> **else**
- 2     **if** ( $p > L.last + 1$ ) **or** ( $p < 1$ ) **then** <Báo lỗi vị trí  $p$  không tồn tại>
- 3     **else** /\* Đẩy các phần tử dưới  $p$  xuống 1 vị trí \*/
- 4         **for**  $q \leftarrow L.last$  **downto**  $p$  **do**
- 5              $L.elements[q+1] \leftarrow L.elements[q]$
- 6         **endfor**
- 7          $L.last \leftarrow L.last + 1$
- 8          $L.elements[p] \leftarrow x$
- 9     **endif**
- 10 **endif**

**End**

## Xóa bỏ phần tử

Mã giả của thao tác loại phần tử tại vị trí  $p$  trong danh sách  $L$

**Procedure** Delete( $p, L$ )

- ❶ **if** ( $p > L.last + 1$ ) **or** ( $p < 1$ ) **then** <Báo lỗi vị trí  $p$  không tồn tại>
- ❷ **else**
- ❸     **for**  $q \leftarrow p$  **to**  $L.last$  **do** /\* Đẩy các phần tử dưới  $p$  lên 1 vị trí \*/
- ❹          $L.elements[q] \leftarrow L.elements[q+1]$
- ❺     **endfor**
- ❻      $L.last \leftarrow L.last - 1$
- ❼ **endif**

**End**

## Cài đặt danh sách tuyến tính : dùng danh sách móc nối (linked list)

Nhược điểm của việc sử dụng mảng là

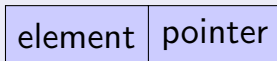
- Lưu trữ hay bổ sung một phần tử tốn kém thời gian.
- Lãng phí khoảng bộ nhớ rỗng không dùng đến.
- Phải duy trì một khoảng không gian lưu trữ liên tục trong bộ nhớ.

Để khắc phục nó ta có thể dùng danh sách móc nối sử dụng con trỏ (pointer), ta xét cách tổ chức móc nối sau

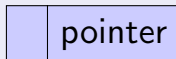
- Danh sách móc nối đơn (singly linked list)
- Danh sách móc nối kép (doubly linked list)

## Danh sách móc nối đơn

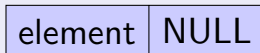
Danh sách gồm các ô (các nút), mỗi ô chứa một phần tử (element) của danh sách và con trỏ (pointer) trỏ đến ô kế tiếp của danh sách.



Có một ô đặc biệt gọi là ô header để trỏ ra ô chứa phần tử đầu tiên trong danh sách, ô này không chứa phần tử nào cả.



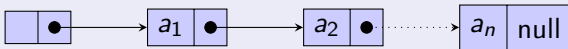
Ngược lại, ô cuối cùng trong danh sách lại có con trỏ trỏ vào giá trị NULL.



# Danh sách

## Danh sách móc nối đơn (tiếp)

Nếu danh sách gồm các phần tử  $a_1, a_2, \dots, a_n$  thì danh sách móc nối được tổ chức như trong hình vẽ



Mỗi nối chỉ ra địa chỉ bộ nhớ của nút tiếp theo trong danh sách

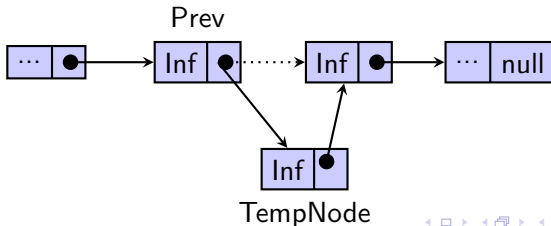
Cài đặt danh sách móc nối đơn trong C

```
typedef <kiểu dữ liệu> ElementType;  
struct _PointerType{  
    ElementType Inf;  
    struct _PointerType *Next;  
};  
typedef struct _PointerType PointerType;
```



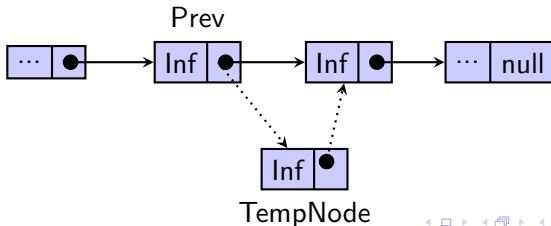
## Danh sách móc nối đơn : thao tác chèn

```
PointerType *InsertMiddle(PointerType *Prev, ElementType X){  
    PointerType *TempNode;  
    TempNode = (PointerType *)malloc(sizeof(PointerType));  
    TempNode->Inf = X;  
    TempNode->Next = Prev->Next;  
    Prev->Next = TempNode;  
    return TempNode;  
}
```



## Danh sách móc nối đơn : thao tác xóa

```
ElementType Delete(PointerType *Prev){  
    ElementType X;  
    PointerType *TempNode;  
    TempNode = Prev->Next; Prev->Next = Prev->Next->Next;  
    X = TempNode->Inf;  
    free(TempNode);  
    return X  
}
```

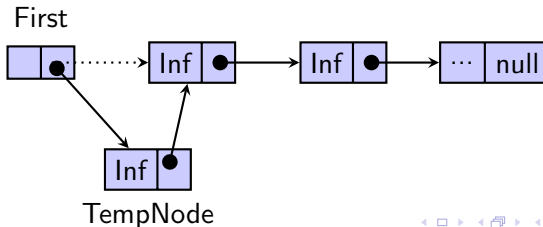


# Danh sách



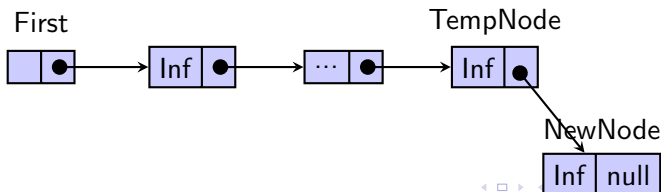
Danh sách móc nối đơn : chèn một nút vào đầu danh sách

```
PointerType *InsertToHead(PointerType *First, ElementType X){  
    PointerType *TempNode;  
    TempNode = (PointerType *) malloc(sizeof(PointerType));  
    TempNode->Inf = X;  
    TempNode->Next = First;  
    First = TempNode;  
    return First;  
}
```



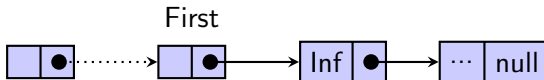
## Danh sách móc nối đơn : chèn một nút vào cuối danh sách

```
PointerType *InsertToLast(PointerType *First, ElementType X){  
    PointerType *NewNode; PointerType *TempNode;  
    NewNode = (PointerType *)malloc(sizeof(PointerType));  
    NewNode->Inf = X; TempNode = First;  
    while(TempNode->Next!=NULL)  
        TempNode = TempNode->Next;  
    TempNode->Next = NewNode;  
    return First;  
}
```



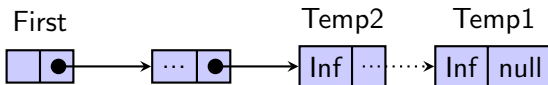
## Danh sách móc nối đơn : xóa nút đầu danh sách

```
PointerType *DeleteHead(PointerType *First){  
    PointerType *TempNode;  
    TempNode = First->Next;  
    free(First);  
    return TempNode;  
}
```



## Danh sách móc nối đơn : xóa nút cuối danh sách

```
PointerType *DeleteLast(PointerType *First){  
    PointerType *Temp1,*Temp2;  
    Temp1 = First; Temp2 = First;  
    while(Temp1->Next != NULL){  
        Temp2 = Temp1;  
        Temp1 = Temp1->Next;}  
    Temp2->Next = NULL;  
    free(Temp1);  
    return First;  
}
```



# Danh sách

## Danh sách móc nối đơn : kiểm tra danh sách rỗng

```
int IsEmpty(PointerType *First)
{
    return !First;
}
```

## Danh sách móc nối đơn : Xóa danh sách

```
PointerType *MakeNull(PointerType *First)
{
    while(!IsEmpty(First))
        First=DeleteHead(First);
    return First;
}
```

Danh sách móc nối đơn : in ra tất cả các phần tử trong danh sách

```
void Print(PointerType *First){
    PointerType *TempNode;
    TempNode = First; int count = 0;
    while(TempNode){
        printf("%6d",TempNode->Inf); count++;
        TempNode = TempNode->Next;
    }
    printf("\n");
}
```



## 1 Các khái niệm

- Kiểu dữ liệu trừu tượng
- Cấu trúc dữ liệu
- Con trỏ

## 2 Mảng

## 3 Danh sách

- Định nghĩa
- Các cách cài đặt danh sách tuyến tính

## 4 Ngăn xếp

- Định nghĩa
- Các cách cài đặt ngăn xếp
- Ngăn xếp và đệ qui
- Ứng dụng

## 5 Hàng đợi

- Định nghĩa
- Các cách cài đặt hàng đợi
- Ứng dụng

## 6 Tổng kết

## Kiểu dữ liệu trừu tượng ngăn xếp

**Định nghĩa** : là dạng đặc biệt của danh sách tuyến tính đã trình bày trong đó các phần tử được đẩy vào (push) và lấy ra (pop) chỉ từ một đầu, đc gọi là đỉnh (top), của danh sách đó.

**Nguyên tắc** : Vào sau ra trước, First-In Last-Out (FILO)

### Các thao tác với ngăn xếp

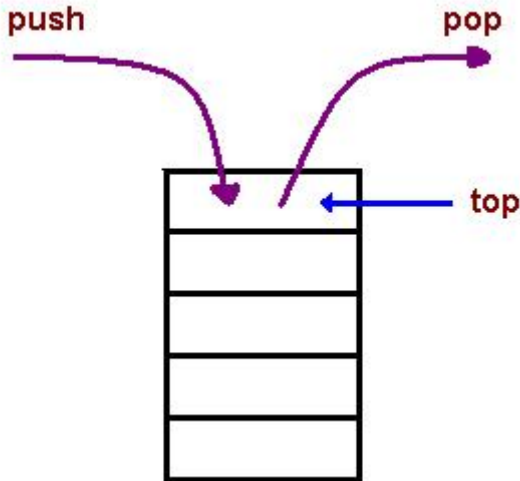
- Đẩy vào (push) bổ sung một phần tử.
- Lấy ra (pop) loại bỏ một phần tử.
- Trả lại phần tử nạp sau cùng (top) mà không loại bỏ.
- Trả lại số phần tử (size) trong ngăn xếp.
- Nhận biết nó có rỗng hay không (IsEmpty).

### Có hai cách cài đặt

- sử dụng mảng
- sử dụng con trỏ

# Ngăn xếp

Minh họa ngăn xếp với hai thao tác cơ bản : đẩy vào (push) và lấy ra (pop) đều thực hiện từ từ một đầu (top) của ngăn xếp.



## Cài đặt ngăn xếp sử dụng mảng trong ngôn ngữ C

```
typedef ... Item;  
static Item *s;  
static int N;  
  
void STACKinit(int maxN){  
s = (Item *)malloc(maxN*sizeof(Item));  
N=0;  
}  
  
int STACKempty(){ return N==0;}  
void STACKpush(Item item){ s[N++] = item;}  
item STACKpop(){return s[--N];}
```

## Cài đặt ngăn xếp sử dụng con trỏ

Cũng như cách mô tả dạng sách móc nối, chỉ khác là ngăn xếp được cài đặt sao cho thao tác bổ sung và loại bỏ chỉ làm việc với ô đầu tiên

```
struct StackNode{  
    float item;  
    StackNode *next;  
};  
  
struct Stack{  
    StackNode *top;  
}
```

## Cài đặt ngăn xếp sử dụng con trỏ (tiếp)

### Các phép toán cơ bản

- Khởi tạo
- Kiểm tra ngăn xếp rỗng
- Kiểm tra tràn ngăn xếp
- Đẩy phần tử vào ngăn xếp
- Lấy một phần tử ra
- In ra các phần tử của ngăn xếp

## Cài đặt ngăn xếp sử dụng con trỏ : Khởi tạo

```
Stack *StackInit(){
    Stack *s;
    s = (Stack *)malloc(sizeof(Stack));
    if (s==NULL){
        return NULL;
    }
    s->top = NULL;
    return s;
}
```

Cài đặt ngăn xếp sử dụng con trỏ : Hủy ngăn xếp

```
void StackDestroy(Stack *s){  
    while(!StackEmpty(s)){  
        StackPop(s);  
    }  
    free(s);  
}
```



Cài đặt ngăn xếp sử dụng con trỏ : các hàm hỗ trợ

```
int StackEmpty(const Stack *s){  
    return (s->top==NULL);  
}  
  
int StackFull(){  
    printf("\n Không còn bo nho");  
    getch();  
    return 1;  
}
```

Cài đặt ngăn xếp sử dụng con trỏ : đẩy phần tử vào ngăn xếp

```
int StackPush(Stack *s, float *item){
    StackNode *node;
    node = (StackNode *)malloc(sizeof(StackNode));
    if(node==NULL){
        StackFull();
        return 1;
    }
    node->item = item;
    node->next = s->top;
    s->top = node;
    return 0;
}
```

Cài đặt ngăn xếp sử dụng con trỏ : lấy phần tử từ ngăn xếp

```
float StackPop(Stack *s){  
    float item;  
    StackNode *node;  
    if(StackEmpty(s)){  
        printf("\n Ngan xep rong");  
        return NULL;  
    }  
    node = s->stop;  
    item = node->item;  
    s->top = node->next;  
    free(node);  
    return item;  
}
```

Cài đặt ngăn xếp sử dụng con trỏ : in các phần tử ngăn xếp

```
int disp(Stack *s){
    StackNode *node;
    float m;
    printf("\n DANH SACH CAC PHAN TU CUA NGAN XEP \n");
    if(StackEmpty(s)){
        printf("\n ngan xep rong \n");
        getch();
    }else{
        node = s->top;
        do{
            m = node->item; printf("% 8.3f",m);
            node = node->next;
        }while(!(node==NULL));
    }
}
```

## Ngăn xếp và đệ qui

Để phân tích giải thuật đệ qui, ta cần hiểu được cách hoạt động của máy tính khi gọi hàm hay thủ tục trong chương trình

- Mỗi khi gọi một hàm hay thủ tục, chương trình sẽ cất địa chỉ và các tham số của nó vào một không gian nhớ đệm (buffer) *tổ chức dạng ngăn xếp*
- Khi hàm hay thủ tục kết thúc, điều đó đồng nghĩa việc địa chỉ và các tham số của nó trong không gian nhớ này được giải phóng.
- Số các địa chỉ và tham số được cất giữ, hay kích thước không gian nhớ đệm, thường được xác định khi bắt đầu chương trình.

## Ngăn xếp và đệ qui (tiếp)

Ta lấy ví dụ trong sách, cho hàm ngôn ngữ lập trình C như sau

```
double power(double x,int n){  
    double tmp = 1;  
    if(n>0)  
    {  
        tmp = power(x, n/2);  
        if(n % 2 == 0) tmp = tmp*tmp;  
        else tmp = tmp*tmp*x;  
    }  
    return tmp;  
}
```

Gọi hàm **power(2,5)** thì ta có trạng thái bộ nhớ đệm như sau

```
double power(double x,int n){  
    double tmp = 1;  
    if(n>0)  
    {  
        tmp = power(x, n/2);  
        if(n % 2 == 0)  
            tmp = tmp*tmp;  
        else  
            tmp = tmp*tmp*x;  
    }  
    return tmp;  
}
```

power(2,5) x=2,n=5,tmp=1



power(2,2) x=2,n=2,tmp=1
power(2,5) x=2,n=5,tmp=1



```
double power(double x,int n){  
    double tmp = 1;  
    if(n>0)  
    {  
        tmp = power(x, n/2);  
        if(n % 2 == 0)  
            tmp = tmp*tmp;  
        else  
            tmp = tmp*tmp*x;  
    }  
    return tmp;  
}
```

⇓

power(2,1)	x=2,n=1,tmp=1
power(2,2)	x=2,n=2,tmp=1
power(2,5)	x=2,n=5,tmp=1

⇓

power(2,0)	x=2,n=0,tmp=1
power(2,1)	x=2,n=1,tmp=1
power(2,2)	x=2,n=2,tmp=1
power(2,5)	x=2,n=5,tmp=1

⇓



```
double power(double x,int n){  
    double tmp = 1;  
    if(n>0)  
    {  
        tmp = power(x, n/2);  
        if(n % 2 == 0)  
            tmp = tmp*tmp;  
        else  
            tmp = tmp*tmp*x;  
    }  
    return tmp;  
}
```

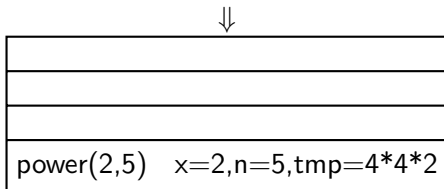
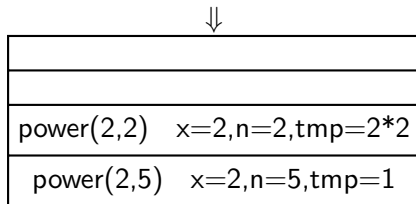


power(2,1)	x=2,n=1,tmp=1
power(2,2)	x=2,n=2,tmp=1
power(2,5)	x=2,n=5,tmp=1



power(2,1)	x=2,n=1,tmp=1*1*2=2
power(2,2)	x=2,n=2,tmp=1
power(2,5)	x=2,n=5,tmp=1

```
double power(double x,int n){
    double tmp = 1;
    if(n>0)
    {
        tmp = power(x, n/2);
        if(n % 2 == 0)
            tmp = tmp*tmp;
        else
            tmp = tmp*tmp*x;
    }
    return tmp;
}
```



Kết thúc gọi đệ qui hàm **power(2,5)** giá trị trả lại  $2^5 = 32$

## Ứng dụng 1 : Bài toán đổi cơ số

Cho một số trong số thập phân, ví dụ  $n = 2013$  chuyển nó sang số có giá trị tương đương trong hệ cơ số

- $b = 2$  thì ta có số  $n_{(b)} = 11111011101$
- $b = 8$  thì ta có số  $n_{(b)} = 3735$
- $b = 16$  thì ta có số  $n_{(b)} = 7DD$

Việc đổi cơ số có được do sử dụng ngăn xếp để tạo nên giá trị tương đương của  $n$  trong hệ cơ số mới  $b$  được trình bày bởi giải thuật sau...

## Ứng dụng 1 : Bài toán đổi cơ số (tiếp)

Chuyển số bất kỳ trong hệ thập phân  $n$  thành số giá trị tương đương trong hệ cơ số  $b$ , nghĩa là  $n_{(b)}$ .

### **Giải thuật dùng ngăn xếp**

Đầu vào : số trong hệ đếm thập phân  $n$ .

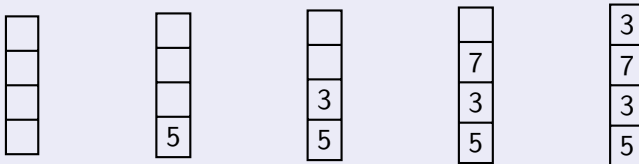
Đầu ra : số trong hệ đếm cơ số  $b$  tương ứng

- 1 Chia lấy phần dư  $n \% b$  được bao nhiêu đẩy vào ngăn xếp.
- 2 Gán lại  $n$  bằng  $n / b$ .
- 3 Lặp lại hai bước 1-2 cho đến khi  $n = 0$ .
- 4 Lấy các giá trị ra khỏi ngăn xếp

## Ứng dụng 1 : Bài toán đổi cơ số (tiếp)

Minh họa ngăn xếp trong khi chuyển đổi  $n = 2013$  sang số có giá trị tương đương trong hệ cơ số 8 (hình minh họa trái sang phải)

- 1  $n = 2013$
- 2  $n \% 8 = 5$  và  $n / 8 = 251$  gán  $n = 251$
- 3  $n \% 8 = 3$  và  $n / 8 = 31$  gán  $n = 31$
- 4  $n \% 8 = 7$  và  $n / 8 = 3$  gán  $n = 3$
- 5  $n \% 8 = 3$  và  $n / 8 = 0$  gán  $n = 0$  (kết thúc)



## Ứng dụng 2 : Bài toán tính giá trị biểu thức số học

Thông thường các công thức toán học được biểu diễn dạng trung tố (infix notation), trình tự thực hiện các phép toán trong đó được ưu tiên bởi dấu ngoặc hay các phép toán - *nhân chia trước cộng trừ sau*.

Ví dụ, công thức số học trung tố

$$(25 - 14) * 2 + 65 = 87$$

Tuy nhiên, nhà toán học Jan Lukasiewicz (1878-1956) đã chỉ ra là ta có thể loại bỏ ngoặc và tính được công thức toán học trên dưới dạng hậu tố (postfix notation) tương đương như sau

$$25\ 14 -\ 2 * 65 +$$

Ta cũng có thể dùng ngăn xếp để tính giá trị biểu thức hậu tố này

## Ứng dụng 2 : Bài toán tính giá trị biểu thức số học (tiếp)

Giải thuật tính giá trị biểu thức hậu tố sử dụng ngăn xếp như sau - giả thuyết ta đã có biểu thức hậu tố cho trước

- 1 Duyệt biểu thức dạng hậu tố từ trái qua phải
- 2 Nếu gặp số hạng thì đẩy nó vào ngăn xếp
- 3 Nếu gặp phép toán  $(+,-,*,/)$  thì thực hiện phép toán với hai số hạng được lấy ra đầu ngăn xếp rồi đẩy kết quả lại vào ngăn xếp
- 4 Tiếp tục duyệt hết biểu thức cho đến khi ngăn xếp chỉ còn giá trị duy nhất, đây chính là kết quả của biểu thức.

## Ứng dụng 2 : Bài toán tính giá trị biểu thức số học (tiếp)

Minh họa ngăn xếp khi tính biểu thức dạng hậu tố  $25\ 14 - 2 * 65 +$  khi duyệt từ trái sang phải

- Số hạng 25 được đẩy vào ngăn xếp
- Số hạng 14 được đẩy vào ngăn xếp
- Với toán hạng - thì lấy hai số hạng  $25 - 14 = 11$  sau đó đẩy lại vào ngăn xếp

25

14
25

11



## Ứng dụng 2 : Bài toán tính giá trị biểu thức số học (tiếp tục)

Minh họa ngăn xếp khi tính biểu thức dạng hậu tố  $25\ 14 - 2 * 65 +$  khi duyệt từ trái sang phải

- Số hạng 2 được đẩy vào ngăn xếp
- Với toán hạng  $*$  thì lấy hai số hạng  $11 * 2 = 22$  sau đó lại đẩy vào ngăn xếp
- Số hạng 65 được đẩy vào ngăn xếp
- Với toán hạng  $+$  thì ta lấy hai số hạng  $22 + 65 = 87$  sau đó lại đẩy vào ngăn xếp (kết thúc duyệt biểu thức dạng hậu tố)

2
11

22

65
22

87

## Ứng dụng 3 : Chuyển biểu thức dạng trung tố về dạng hậu tố

Bây giờ, ta xét đến việc chuyển biểu thức dạng trung tố về hậu tố gồm

- Các toán hạng cộng (+), trừ (-), nhân (\*), chia (/) và dấu ngoặc
- Các số hạng

Trước hết cũng cần nhắc lại qui tắc tính giá trị biểu thức dạng trung tố

- Thứ tự ưu tiên (precedence) : Mũ; Nhân/Chia ; Cộng/Trừ
- Qui tắc kết hợp (associativity) : Khi phép toán có cùng mức ưu tiên
  - ▶ Mũ : Phải qua Trái (PT)
  - ▶ Nhân/Chia : Trái qua phải (TP)
  - ▶ Cộng/Trừ : Trái qua phải (TP)
- Dấu ngoặc được thực hiện trước cả thứ tự ưu tiên và qui tắc kết hợp.

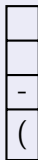
## Ứng dụng 3 : Giải thuật Shunting-yard (E.Dijkstra)

- ❶ Duyệt biểu thức từ trái qua phải
- ❷ Nếu gặp số đưa ra lập tức
- ❸ Nếu gặp phép toán  $o_1$  thì
  - ▶ khi phép toán  $o_2$  ở đầu ngăn xếp còn thuộc một trong hai tình huống
    - ★  $o_1$  có quy tắc kết hợp TP, thứ tự ưu tiên nhỏ hơn bằng  $o_2$
    - ★  $o_1$  có quy tắc kết hợp PT, thứ tự ưu tiên nhỏ hơn  $o_2$
  - thì đưa  $o_2$  ra
  - ▶ nạp  $o_1$  vào
- ❹ Nếu gặp dấu mở ngoặc ( thì nạp nó vào ngăn xếp.
- ❺ Nếu gặp dấu đóng ngoặc ) thì lấy các phần tử ra khỏi ngăn xếp đến khi gặp dấu mở ngoặc đầu tiên. Lấy nốt dấu ) nhưng không đưa ra.
- ❻ Khi đã duyệt hết biểu thức, đưa tất cả các phép toán còn lại ra khỏi ngăn xếp.

## Ứng dụng 3 : Chuyển biểu thức dạng trung tố về dạng hậu tố (tiếp)

Minh họa ngăn xếp khi chuyển biểu thức dạng trung tố  $(25 - 14) * 2 + 65$  sang dạng hậu tố, duyệt từ trái qua phải

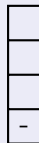
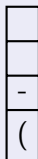
- Gặp dấu mở ngoặc (, nạp nó vào ngăn xếp
- Gặp số hạng 25, đưa ra tức thì  $\Rightarrow 25$
- Gặp dấu - đưa vào ngăn xếp
- Gặp số hạng 14, đưa ra tức thì  $\Rightarrow 25 \ 14$



## Ứng dụng 3 : Chuyển biểu thức dạng trung tố về dạng hậu tố (tiếp tục)

Minh họa ngăn xếp khi chuyển biểu thức dạng trung tố về dạng hậu tố  $(25 - 14) * 2 + 65$

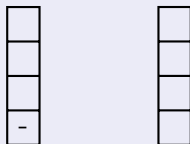
- Gặp dấu đóng ngoặc  $)$ , đẩy - ra khỏi ngăn xếp cho đến khi gặp dấu mở ngoặc  $( \Rightarrow 25 \ 14 \ -$
- Gặp dấu  $*$  thì đẩy vào ngăn xếp
- Gặp số 2 thì đưa ra tức thì  $\Rightarrow 25 \ 14 \ -2$
- Gặp dấu  $+$  thì đẩy vào ngăn xếp, đưa  $*$  ra do  $*$  có thứ tự ưu tiên cao hơn  $\Rightarrow 25 \ 14 \ -2*$



## Ứng dụng 3 : Chuyển biểu thức dạng trung tố về dạng hậu tố (tiếp tục)

Minh họa ngăn xếp khi chuyển biểu thức dạng trung tố  $(25 - 14) * 2 + 65$

- Gặp số hạng 65 thì đưa ra tức thì  $\Rightarrow 25 \ 14 \ - \ 2 * 65$
- Đã duyệt hết và lấy hết phép toán trong ngăn xếp đưa ra  $\Rightarrow 25 \ 14 \ - \ 2 * 65 +$



## 1 Các khái niệm

- Kiểu dữ liệu trừu tượng
- Cấu trúc dữ liệu
- Con trỏ

## 2 Mảng

## 3 Danh sách

- Định nghĩa
- Các cách cài đặt danh sách tuyến tính

## 4 Ngăn xếp

- Định nghĩa
- Các cách cài đặt ngăn xếp
- Ngăn xếp và đệ qui
- Ứng dụng

## 5 Hàng đợi

- Định nghĩa
- Các cách cài đặt hàng đợi
- Ứng dụng

## 6 Tổng kết

# Hàng đợi

## Kiểu dữ liệu trừu tượng ngăn xếp (Queue)

**Định nghĩa** : là danh sách tuyến tính mà phép toán chen luôn được thực hiện chỉ được thực hiện ở một phía, gọi là phía sau hay phía cuối (back or rear), trong khi đó phép toán xóa chỉ được thực hiện ở phía còn lại, gọi là phía trước hay đầu (front or head).

**Nguyên tắc** : Vào trước Ra trước, First-In First-Out (FIFO)

### Các phép toán cơ bản

- Khởi tạo
- Kiểm tra rỗng isEmpty()
- Xác định có tràn hay không
- Trả lại phần tử đầu hàng front()
- Chen phần tử vào cuối hàng enqueue()
- Xóa và lấy ra phần tử đầu hàng dequeue()
- In ra hàng đợi print()



# Hàng đợi

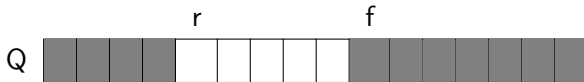
## Cài đặt hàng đợi bằng mảng

Sử dụng mảng Q có kích thước N theo thứ tự vòng tròn. Có hai biến để lưu vị trí đầu và cuối (front và rear) :

- f chỉ số của phần tử đầu hàng đợi
- r chỉ số của vị trí ở ngay sau vị trí của phần tử cuối cùng của hàng đợi. Vị trí r được giữ là rỗng.



Cấu hình bình thường



Cấu hình xoay vòng tròn

## Cài đặt hàng đợi bằng mảng

Cài đặt các phép toán cơ bản viết bằng mã giả

- Tính kích thước hàng đợi

```
Function size()
```

```
    return (N-f+r) mod N
```

```
End
```

- Kiểm tra hàng đợi có rỗng không

```
Function isEmpty()
```

```
    return (f=r)
```

```
End
```

trong đó **mod** là phép chia lấy phần dư

## Cài đặt hàng đợi bằng mảng (tiếp)

Cài đặt các phép toán cơ bản viết bằng mã giả

- Chèn phần tử vào cuối hàng đợi

**Procedure** enqueue(o)

**if** (size=N-1) **then**

        Hiện ra lỗi tràn hàng đợi

**else**

$Q[r] \leftarrow o$

$r \leftarrow (r+1) \bmod N$

**endif**

**End**

# Hàng đợi

## Cài đặt hàng đợi bằng mảng (tiếp)

Cài đặt các phép toán cơ bản viết bằng mã giả

- Lấy phần tử tại đầu hàng đợi

**Function** dequeue()

o  $\leftarrow$  NUL

**if** isEmpty() **then**

    Hiện ra hàng đợi đã rỗng

**else**

    o  $\leftarrow$  Q[f]

    f  $\leftarrow$  (f+1) mod N

**endif**

**return** o

**End**

## Cài đặt hàng đợi bằng danh sách móc nối

Khi cài đặt hàng đợi bằng danh sách móc nối đơn.

```
struct qnode{  
    int element;  
    struct qnode *next;  
} node;  
struct queue {node *front; node *back;};
```

DataType là kiểu dữ liệu cần lưu trữ, được khai báo trước.

## Cài đặt hàng đợi bằng danh sách móc nối (tiếp)

Các toán tử được khai báo trong ngôn ngữ C như sau :

**// Các hàm thực hiện các toán tử**

**queue \*create();**

**int isEmpty(queue \*q);**

**int size(queue \*q);**

**void enqueue(queue \*q, node \*newNode);**

**node \*dequeue(queue \*q);**

**// In ra các phần tử trong hàng đợi**

**void print(queue \*q)**

## Cài đặt hàng đợi bằng danh sách móc nối (tiếp)

Các toán tử với mã nguồn C tương ứng

```
queue *create(){  
    queue *q;  
    q = (queue *)malloc(sizeof(queue));  
    if(q==NULL) return NULL; // Không còn bộ nhớ  
    q->front = NULL; q->rear = NULL;  
    return q;  
}
```

## Cài đặt hàng đợi bằng danh sách móc nối (tiếp)

Các toán tử với mã nguồn C tương ứng

```
int isEmpty(queue *q){  
    return ((q->front==NULL)&&(q->rear==NULL));  
}  
  
int size(queue *q){  
    queue *ptr=q->front; int count=0;  
    while(ptr!=NULL){  
        ptr = ptr->next; count++;  
    }  
    return count;  
}
```



## Cài đặt hàng đợi bằng danh sách móc nối (tiếp)

Các toán tử với mã nguồn C tương ứng

```
void enqueue(queue *q, node *newNode){  
/* trong trường hợp ta đã dùng hàm malloc để tạo newNode */  
if(!isEmpty()){/* nối vào đuôi hàng đợi */  
    q->rear->next = *newNode;  
    q->rear = *newNode;  
}  
else  
{/* nút dữ liệu đầu tiên của hàng đợi*/  
    q->rear = *newNode;  
    q->front = *newNode;  
}  
}
```

# Hàng đợi

## Cài đặt hàng đợi bằng danh sách móc nối (tiếp)

Các toán tử với mã nguồn C tương ứng

```
node *dequeue(queue *q){  
    node *ptr = q->front;  
    if(ptr!=NULL){/* có nút dữ liệu trong hàng đợi */  
        q->front = q->front->next;  
        if(q->front==NULL) /* là nút dữ liệu cuối cùng */  
            q->rear = NULL;  
        ptr->next = NULL; /* ko trở về tiếp vào front nữa */  
    }  
    return ptr; /* trả lại con trỏ chưa free bộ nhớ */  
}
```

## Ứng dụng 1 : Chuyển đổi xâu ký tự số thành số thập phân n

Ý tưởng :

- Đưa lần lượt các ký tự số trong xâu ký tự vào hàng đợi Q
- Khởi tạo giá trị

$$n \leftarrow 0$$

- Lấy từng ký tự số ra khỏi hàng đợi Q và cập nhật số thập phân n theo công thức

$$n \leftarrow n \times 10 + \text{giá trị ký tự số}$$

## Ứng dụng 1 : Chuyển đổi xâu ký tự số thành số thập phân n (tiếp)

Mã giả của thuật toán như sau :

```
// Nạp các ký tự số ch vào hàng đợi
```

```
do
```

```
    enqueue(Q,ch)
```

```
while(ch = digit)
```

```
// khởi tạo giá trị số n
```

```
n  $\leftarrow$  0
```

```
done  $\leftarrow$  false
```

```
// Vòng lặp cập nhật giá trị số thập phân n
```

```
do
```

```
    n  $\leftarrow$  n  $\times$  10 + giá trị ký tự số
```

```
    if(not isEmpty(Q)) dequeue(Q,ch) else done  $\leftarrow$  true endif
```

```
while( done or (ch not digit))
```

## Ứng dụng 2 : Nhận biết chuỗi ký tự palindromes

Chuỗi ký tự palindromes là chuỗi ký tự mà đọc từ trái qua phải cũng giống như đọc từ phải qua trái. Ví dụ các từ sau đây

- "NOON", "DEED", "RADAR", "MADAM", "POP"
- "ABLE WAS I ERE I SAW ELBA"
- "các", "cục", "tịt", "tít", "ôô"...

Ý tưởng giải thuật nhận biết chuỗi ký tự palindromes :

- Cho chuỗi ký tự vào một hàng đợi và một ngăn xếp
- Lấy từng ký tự một từ hàng đợi và một từ ngăn xếp
- nếu chúng giống nhau tất cả thì là chuỗi ký tự palindromes, ngược lại khi không giống một lần thì không phải là chuỗi ký tự palindromes.

- Phân biệt được dữ liệu và cấu trúc dữ liệu (ô dữ liệu + liên kết)
- Hiểu được ý nghĩa và các phép toán của các cấu trúc dữ liệu : danh sách, ngăn xếp, hàng đợi
- Hiểu được mối liên quan giữa không gian đệm dạng ngăn xếp khi gọi hàm và thủ tục
- Các ứng dụng của danh sách, ngăn xếp, hàng đợi