Chương 2 : Các cấu trúc dữ liệu cơ bản

Trịnh Anh Phúc [1]

[1]Bộ môn Khoa Học Máy Tính, Viện CNTT & TT,
Trường Đại Học Bách Khoa Hà Nội.

Ngày 21 tháng 9 năm 2020

Giới thiệu

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	-3.4×10^{38}	3.4×10^{38}
double	64	-1.7×10^{308}	1.7×10^{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	duyệt (traverse), tìm đường (search path), ...
Ngăn xếp	các phần tử	gắp (pop), ấn (push), kiểm tra rỗng, ...
Hàng đợi	các phần tử	vào hàng (enqueue), ra khỏi hàng (dequeue),
Cây	gốc, lá, cành	duyệt (traverse), tìm kiếm (search), ...

Các khái niệm

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.

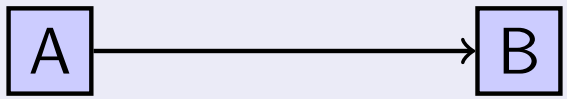
```
struct record{
    float data;
int next;} reclist[100];
```

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

Các khái niệm

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 kiêu dữ liệu cho trước tên là celltype

celltype *ptr

Danh sách

Danh sách tuyến tính

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

- 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

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 ví trí p
- Trả lại vị trí trước ví trí p
- Trả lại vị trí đầu tiên
- Tạo mảng rỗng
- In ra danh sách các phần tử

Danh sách

Các cách cài đặt danh sách tuyến tính

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

- Dùng mảng (Array list)
  - Cất 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.

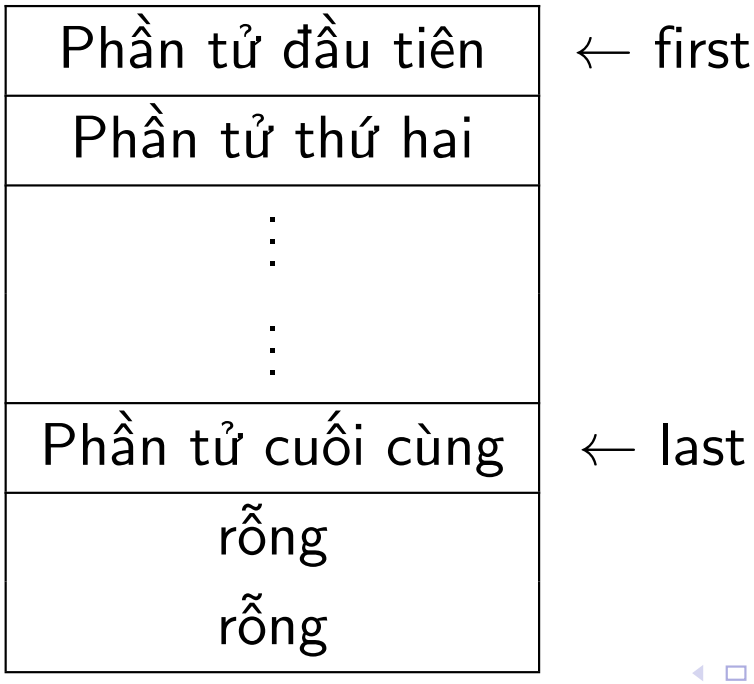
Danh sách

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.



Danh sách dùng mảng

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** (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

Danh sách dùng mảng

Xóa bỏ phần tử

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

Procedure Delete(p,L)

1. **if** (p>L.last + 1) **or** (p<1) **then** <Báo lỗi vị trí p không tồn tại>
2. **else**
3. L.last $\leftarrow$ L.last - 1
4. **for** q $\leftarrow$ p **to** L.last **do** /* Đẩy các phần tử dưới p lên 1 vị trí */
5. L.elements[q] $\leftarrow$ L.elements[q+1]
6. **endfor**
7. **endif**

End

Danh sách

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.

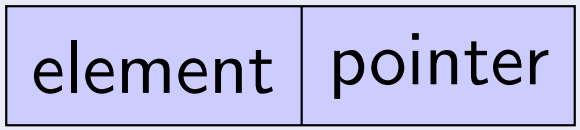
Để khắc phục nó ta có thể dùng danh sách móc nối sử dụng con trỏ (pointer), ta xét cách tổ chức móc nối sau

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

Danh sách

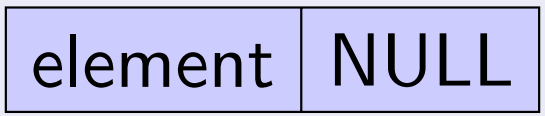
Danh sách móc nối đơn

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 đều tiên trong danh sách, ô này không chứa phần tử nào cả.

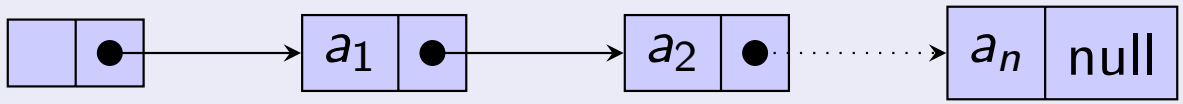
Ngược lại, ô cuối cùng trong danh sách lại có con trỏ trỏ vào giá trị NULL.



Danh sách

Danh sách móc nối đơn (tiếp)

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



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

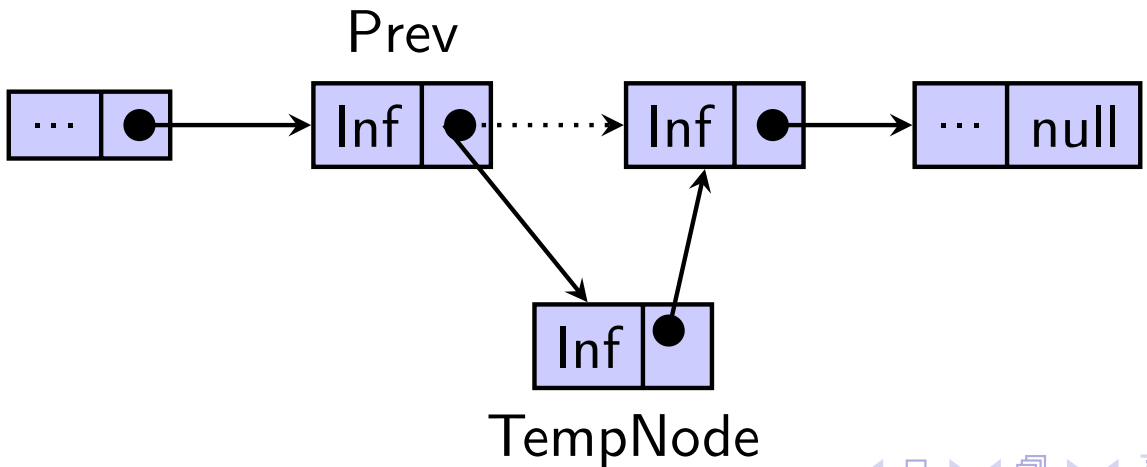
Cài đặt danh sách móc nối đơn trong C

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

Danh sách

Danh sách móc nối đơn : thao tác chèn

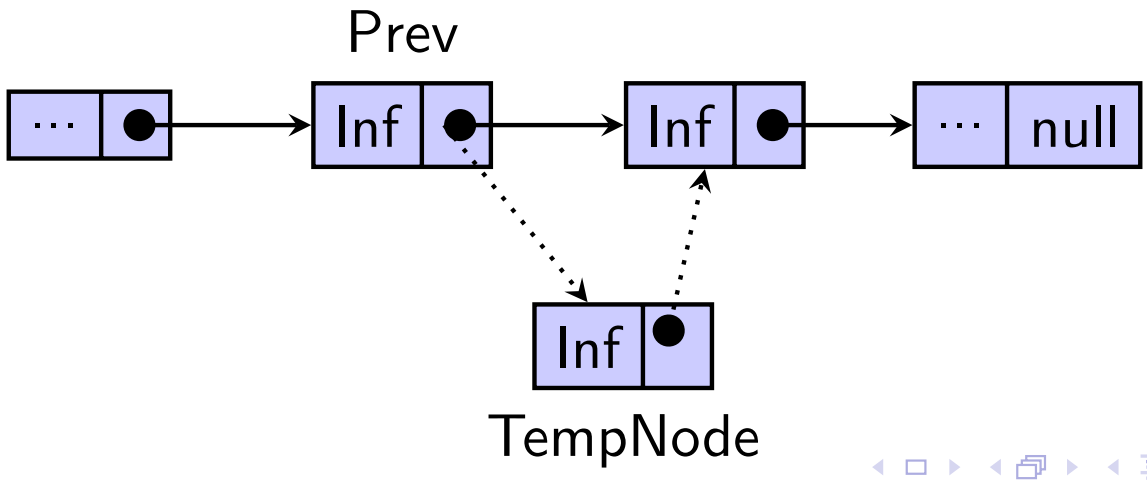
```
PointerType *InsertMiddle(PointerType *Prev, ElementType X){
    PointerType *TempNode;
    TempNode = (PointerType *)malloc(sizeof(PointerType));
    TempNode->Inf = X;
    TempNode->Next = Prev->Next;
    Prev->Next = TempNode;
    return TempNode;
}
```



Danh sách

Danh sách móc nối đơn : thao tác xóa

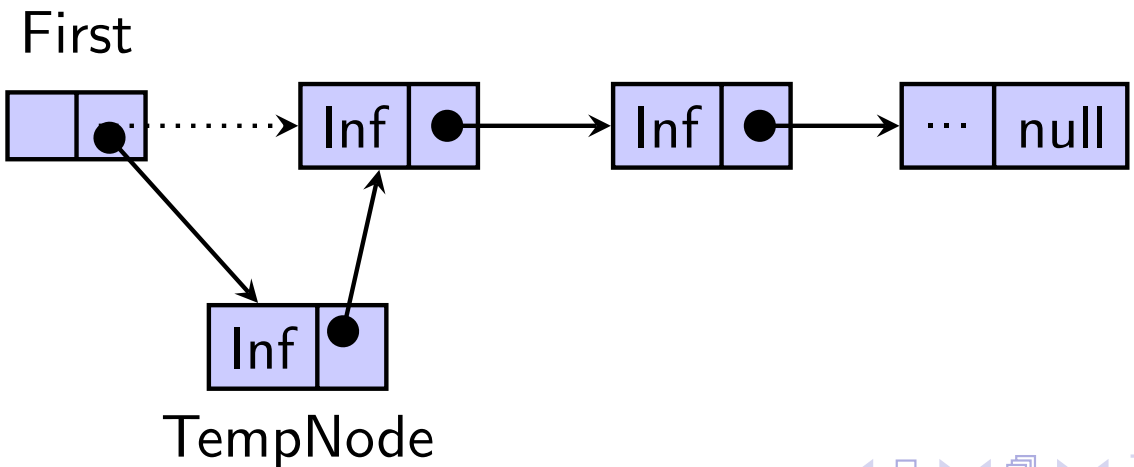
```
ElementType Delete(PointerType *Prev){
    ElementType X;
    PointerType *TempNode;
    TempNode = Prev->Next; Prev->Next = Prev->Next->Next;
    X = TempNode->Inf;
    free(TempNode);
    return X
}
```



Danh sách

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

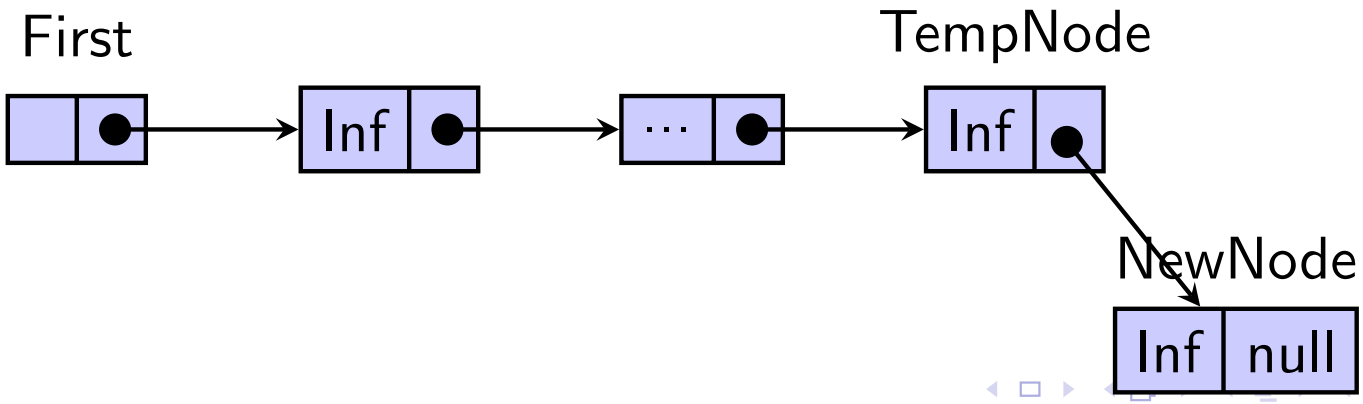
```
PointerType *InsertToHead(PointerType *First, ElementType X){
    PointerType *TempNode;
    TempNode = (PointerType *) malloc(sizeof(PointerType));
    TempNode->Inf = X;
    TempNode->Next = First;
    First = TempNode;
    return First;
}
```



Danh sách

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

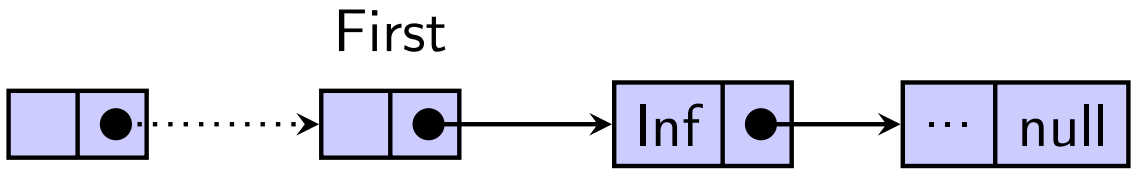
```
PointerType *InsertToLast(PointerType *First, ElementType X){
    PointerType *NewNode; PointerType *TempNode;
    NewNode = (PointerType *)malloc(sizeof(PointerType));
    NewNode->Inf = X; TempNode = First;
    while(TempNode->Next!=NULL)
        TempNode = TempNode->Next;
    TempNode->Next =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



Danh sách

Danh sách móc nối đơn : xóa nút đầu danh sách

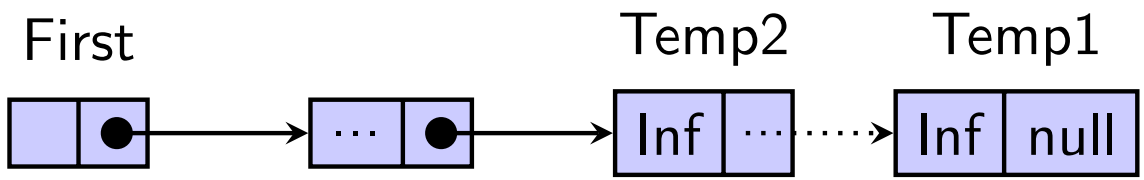
```
PointerType *DeleteHead(PointerType *First){
    PointerType *TempNode;
    TempNode = First->Next;
    free(First);
    return TempNode;
}
```



Danh sách

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

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");
}
```

Ngăn xếp

Kiểu dữ liệu trừu tượng ngăn xếp

Định nghĩa : là dạng đặc biệt của danh sách tuyến tính đã trình bầy 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

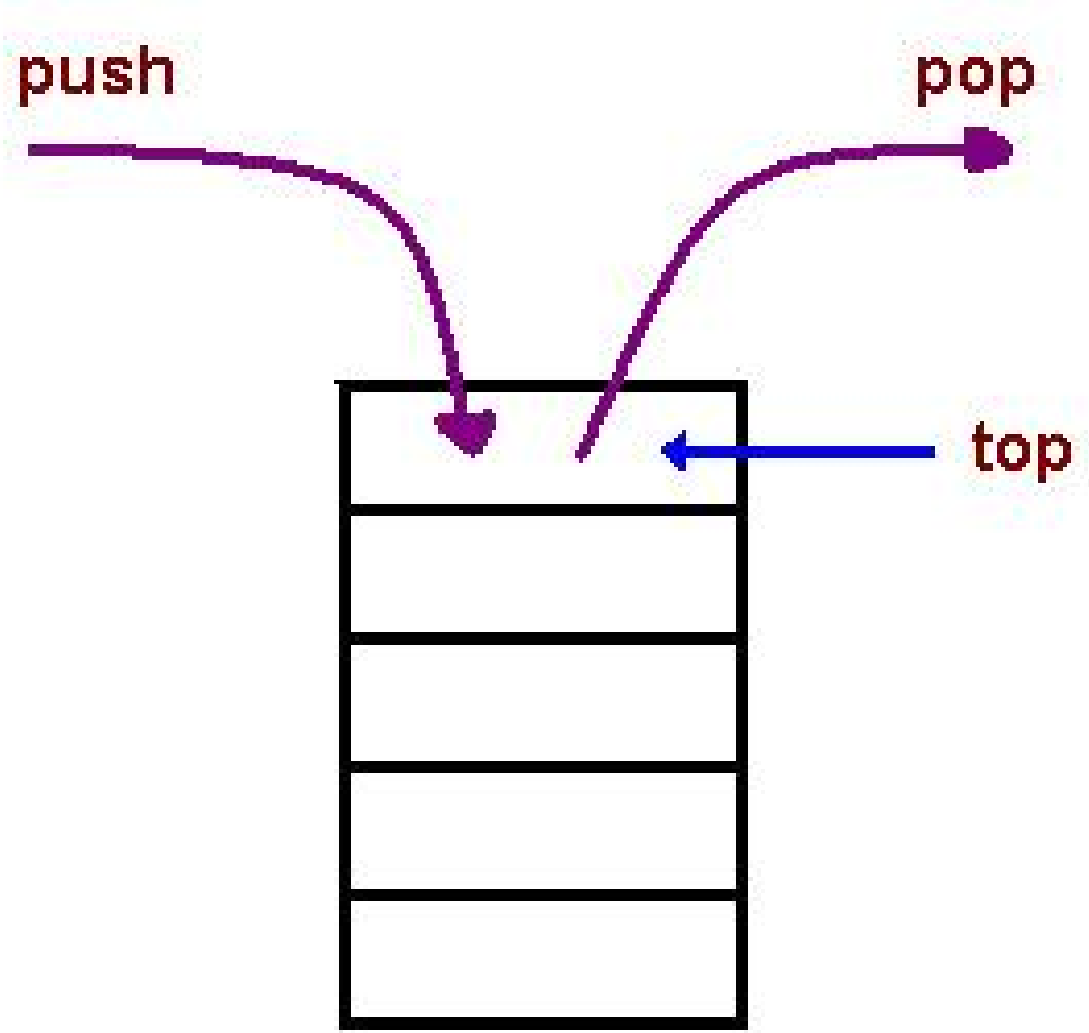
- Đẩy vào (push) bổ sung một phần tử.
- Lấy ra (pop) loại bỏ một phần tử.
- Trả lại phần tử nạp sau cùng (top) mà không loại bỏ.
- Trả lại số phần tử (size) trong ngăn xếp.
- Nhận biết nó có rồng hay không (IsEmpty).

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.



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];}
```

Ngăn xếp

Cài đặt ngăn xếp sử dụng con trỏ

Cũng như cách mô tả dang 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 tư 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;
}
```

Ngăn xếp

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 bổ trợ

```
int StackEmpty(const Stack *s){
    return (s->top==NULL);
}

int StackFull(){
    printf("\n Khong con bo nho");
    getch();
    return 1;
}
```

Ngăn xếp

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;
}
```

Ngăn xếp

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;
}
```

Ngăn xếp

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

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 bầy 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

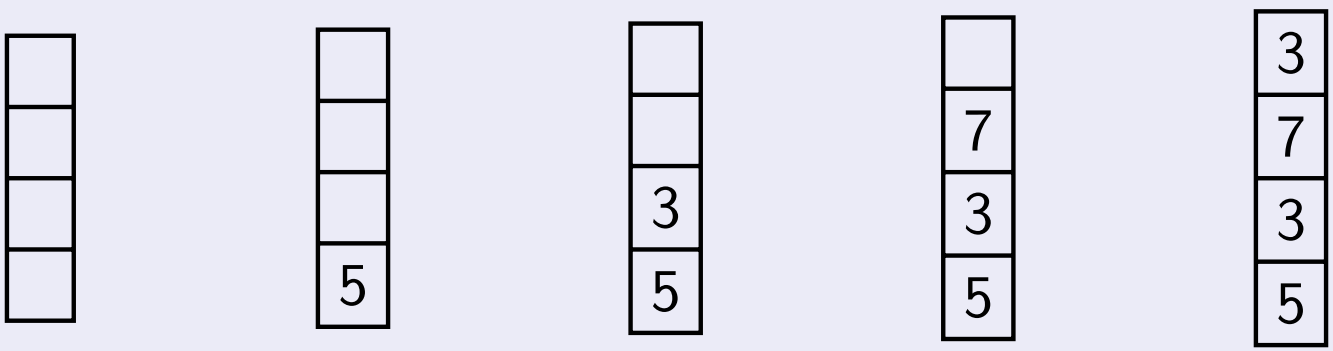
1. Chia lấy phần dư $n\%b$ được bao nhiêu đẩy vào ngăn xếp.
2. Gán lại n bằng n/b.
3. Lặp lại hai bước 1-2 cho đến khi $n = 0$.
4. Lấy các giá trị ra khỏi ngăn xếp

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

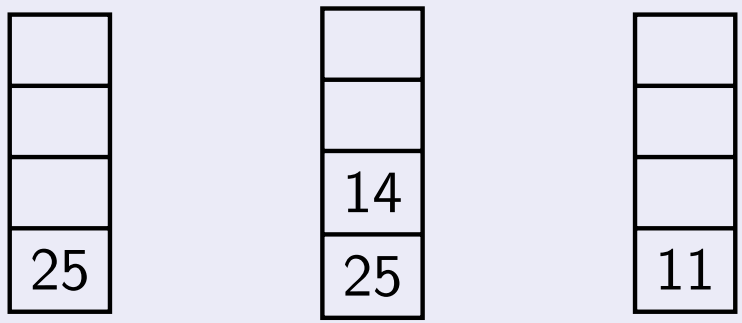
1. Duyệt biểu thức dạng hậu tố từ trái qua phải
2. Nếu gặp số hạng thì đẩy nó vào ngăn xếp
3. Nếu gặp phép toán (+,-,*,/) thì thực hiện phép toán với hai số hạng được lấy ra đầu ngăn xếp rồi đẩy kết quả lại vào ngăn xếp
4. Tiếp tục duyệt hết biểu thức cho đến khi ngăn xếp chỉ còn giá trị duy nhất, đây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

Ngăn xếp

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

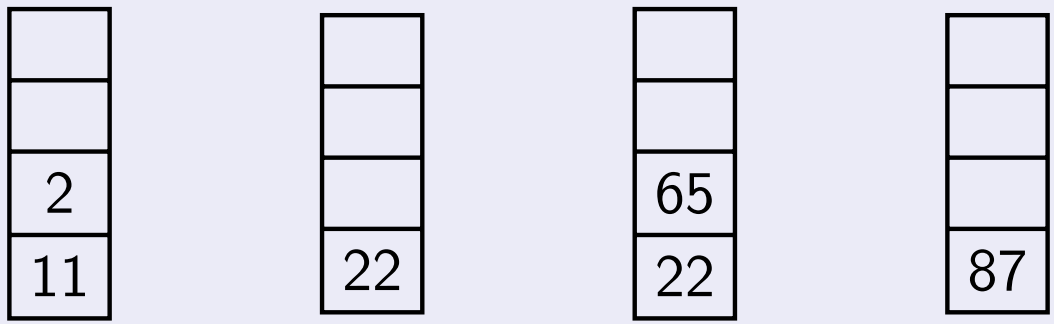


Ngăn xếp

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)



Hàng đợi

Kiểu dữ liệu trừu tượng ngăn xếp (Queue)

Định nghĩa : là danh sách tuyến tính mà phép toán chèn 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 ơ 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()
- Chèn phần tử vào cuối hàng enqueue()
- Xóa và lấy ra phần tử đầu hàng dequeue()
- In ra hàng đợi print()

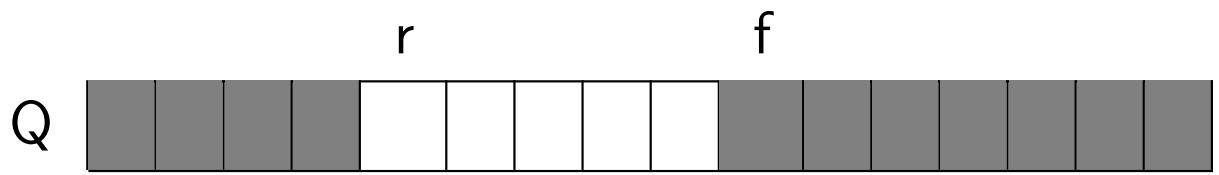
Hàng đợi

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

Hàng đợi

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

Hàng đợi

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] ← o
        r ← (r+1) mod 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 ← NUL
    if isEmpty() then
        Hiện ra hàng đợi đã rỗng
    else
        o ← Q[f]
        f ← (f+1) mod N
    endif
return o
End
```

Hàng đợi

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;
}
```

Hàng đợi

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;
    }
}
```

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 trỏ kế tiếp vào front nữa */
    }
    return ptr;/* trả lại con trỏ chưa free bộ nhớ */
}
```

Hàng đợi

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

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

Hàng đợi

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))

Hàng đợi

Ứng dụng 2 : Nhận biết xâu ký tự palidromes

Xâu ký tự palidromes là xâu 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","ôtô"...

Ý tưởng giải thuật nhận biết xâu ký tự palidromes :

- Cho xâu 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 xâu ký tự palidromes, ngược lại khi không giống một lần thì không phải là xâu ký tự palidromes.

Tổng kết

- 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