Chương2

Cấu trúc dữ liệu động

Nội dung

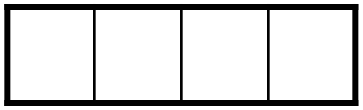
1. Biến tĩnh và biến động
2. Danh sách liên kết
3. Ngăn xếp - Stack
4. Hàng đợi - Queue

Biến Tĩnh & Biến Động

Biến tĩnh (Static Variant)

- Khai báo tường minh và được cấp phát vùng nhớ ngay khi khai báo, vùng nhớ được cấp cho biến tĩnh sẽ không thể thu hồi được nếu biến còn trong phạm vi hoạt động

Ví dụ: int X; → X  (2 bytes)

float Y; → Y  (4 bytes)

- Nhược điểm
  - Chúng có thể chiếm dụng bộ nhớ.
  - Một số thao tác tiến hành thiếu tự nhiên trên các đối tượng tĩnh: Chèn và xóa trong mảng.

Biến động (Dynamic Variant)

- Tính chất của biến động:
  - Thuộc một kiểu dữ liệu nào đó, không được khai báo tường minh → không có tên
  - Được cấp phát vùng nhớ và truy xuất thông qua một biến con trỏ (Biến tĩnh)
  - Có thể thay đổi kích thước hoặc thu hồi (hủy bỏ) vùng nhớ được cấp phát khi chương trình đang hoạt động
  - Việc tạo ra biến động (cấp phát vùng nhớ cho nó) và xóa bỏ nó được thực hiện bởi các thủ tục đã có sẵn

- Ví dụ:

```
int X=10, *P; // khai báo 2 biến tĩnh X, P (con trỏ)
P=&X; // Cho P trỏ đến X
printf("\nĐịa chỉ của biến X là %x",P);
printf("\nX=%d",*P); // hoặc printf("X=%d",X); in giá trị của X
P=(int*)malloc(sizeof(int)); // tạo biến động cho P trỏ đến
*P=X; //gán giá trị cho biến động bằng giá trị của X
printf("\nĐịa chỉ của biến động là %x",P);
printf("\nGiá trị của Biến động=%d",*P);
free(P); //hủy (thu hồi vùng nhớ) biến động do P trỏ đến
```

Tạo một biến động

- Dùng hàm có sẵn trong thư viện <ALLOC.H> hay <STDLIB.H>
  - **void *malloc (size);** Cấp phát vùng nhớ có kích thước **size** bytes và trả về địa chỉ của vùng nhớ đó.
  - **void *calloc (n, size);** Cấp phát vùng nhớ cho **n** phần tử, mỗi phần tử có kích thước **size** bytes và trả về địa chỉ của vùng nhớ đó.
  - **void * realloc (void *ptr, size_t nbyte):** Thay đổi kích thước vùng nhớ đã cấp phát trước đó cho biến con trỏ ptr là n byte, đồng thời chép dữ liệu vào vùng nhớ mới.

Tạo một biến động

- Dùng toán tử new (trong C++)

  <tên con trỏ> = new <tênkiểu>[(Số_phần_tử)];

  Công dụng như hàm malloc nhưng tự động thực hiện hàm sizeof(tênkiểu).

- **Ví dụ:**

```
int *p1, *p2, *p3; // khai báo 3 biến con trỏ
p1 = (int *) malloc( sizeof(int) ); //tạo biến động
p1 = (int*) realloc (p1, 4); //thay đổi kích thước biến
p2 = (int*) calloc(10, 2);//tạo 10 biến động
p2 = new int;
```

Hủy một biến động

- Dùng hàm

  free(Tên_con_trỏ);

- Dùng toán tử delete (trong C++)

  delete Tên_con_trỏ ;

Lưu ý: không thể dùng hàm **free** để hủy một biến được cấp phát bằng toán tử **new**

- Ví dụ:

```
int *p1, *p2; // khai báo 2 biến con trỏ
p1 = (int *) malloc( sizeof(int) ); //tạo biến động kiểu int
p2 = new float; // tạo biến động kiểu float
free(p1); //hủy biến động do p1 trỏ tới
delete p2; //hủy biến động do p2 trỏ tới
```

Truy xuất biến động

- **Tên_con_trỏ** ~ Địa chỉ của biến động
- ***Tên_con_trỏ** ~ Giá trị của biến động
- Ví dụ

```
int *P;
P=(int*) malloc(sizeof(int));// tạo biến động
*P=100; //gán giá trị cho biến động
print("\nĐịa chỉ của biến động là %x",P);
print("\nGiá trị của biến động là %d",*P);
```

Danh sách liên kết

A. Danh sách liên kết đơn
B. Danh sách liên kết đôi
C. Danh sách liên kết vòng

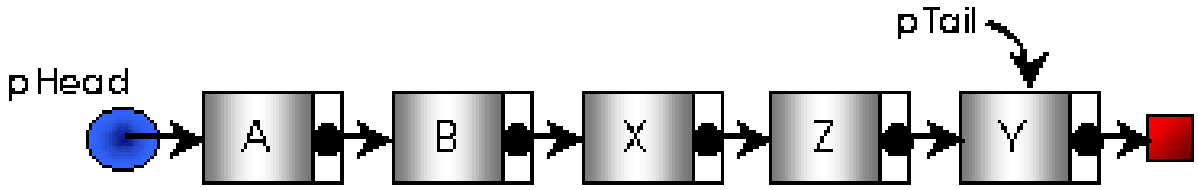
Danh sách liên kết

- Danh sách liên kết là 1 tập hợp các phần tử cùng kiểu, giữa 2 phần tử trong danh sách có một mối liên kết
- Cho trước kiểu dữ liệu T, Kiểu xâu liên kết

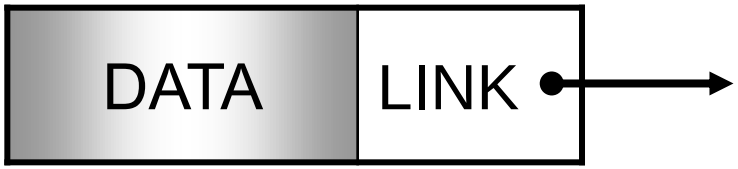
  Tx = < Vx, Ox> trong đó:
  - Vx = { Tập hợp có thứ tự gồm các biến động kiểu T }
  - Ox = {Tạo danh sách; Liệt kê các phần tử trong danh sách; Thêm; Hủy; Tìm; Sắp xếp }

Danh sách liên kết

Hình ảnh một danh sách liên kết



- Cấu trúc một node trong danh sách gồm
  - Thành phần DATA: chứa dữ liệu kiểu T nào đó
  - Thành phần LINK: là một con trỏ tới node kế tiếp



Danh sách liên kết đơn (xâu đơn) (Simple List)

Khai báo kiểu 1 nút trong xâu liên kết đơn:

```
typedef  struct Node
{       KiểuT Data;
        struct Node *Next;
} NodeType;
```

```
typedef struct Node NodeType;
struct Node
{       KiểuT Data;
        NodeType *Next;
} ;
```

Khai báo con trỏ đầu xâu: NodeType *Head, * Tail;

Có thể khai báo kiểu con trỏ đến kiểu nút :

*typedef NodeType *NodePtr;*
NodePtr Head, Tail;

Ví dụ

- Khai báo xâu liên kết lưu trữ các hệ số của một đa thức

```
typedef struct
{   float GiaTri;
    int Bac;
} HeSo;
typedef struct Node
{
    HeSo  Data;
    NodeType *Next ; /* Con trỏ liên kết */
} NodeType;
typedef NodeType *NodePtr;
NodePtr  Head, Tail; /* Con trỏ đầu xâu */
```

Danh sách liên kết

Các thao tác trên xâu đơn

- Khởi tạo xâu rỗng
- Kiểm tra xâu rỗng
- Tạo nút mới
- Chèn nút vào xâu
- Tạo xâu
- Hủy nút trên xâu
- Tìm kiếm giá trị
- Duyệt xâu
- Sắp xếp dữ liệu

Các thao tác trên xâu đơn

- **Khởi tạo 1 xâu mới rỗng:** Head = Tail =NULL;
- **Kiểm tra xâu rỗng:** if (Head == NULL)...
- **Tạo Nút chứa giá trị kiểu T:**

  Thuật toán: Trả về địa chỉ biến động chứa giá trị X

  b1: Tạo biến động kiểu T và lưu địa chỉ vào biến con trỏ P

  b2: Nếu không tạo được thì báo lỗi và kết thúc ngược lại chuyển sang b3

  b3: Lưu giá trị X vào phần dữ liệu của nút

  b4: Gán phần Liên kết của Nút giá trị NULL

  b5: return P;

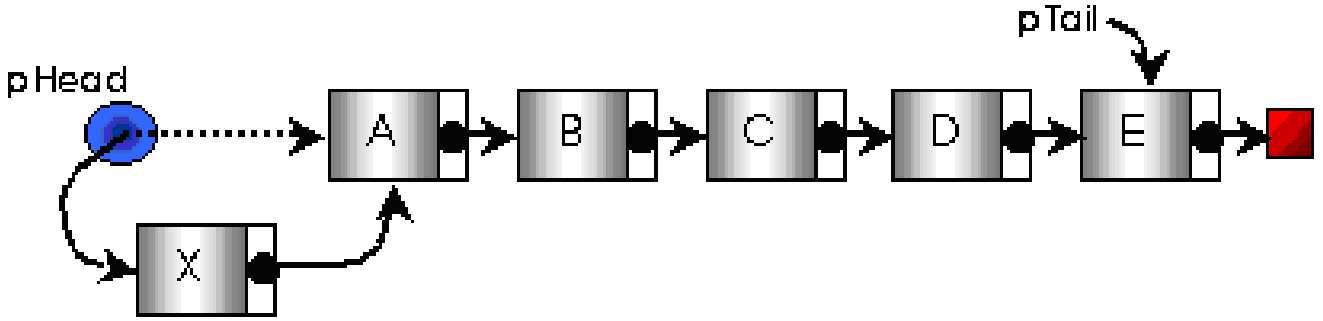
Các thao tác trên xâu đơn _ Tạo nút chứa giá trị X

- **Cài đặt**

```
NodePtr CreateNode( T  x)
{   NodePtr        P;
    P = (NodePtr) malloc(sizeof(NodeType));
    if (p == NULL) {printf("Khong du bo nho"); exit(1);}
    P->Data = x;
    P->Next = NULL;
    return P;
}
```

Các thao tác trên xâu đơn _ Chèn nút vào xâu

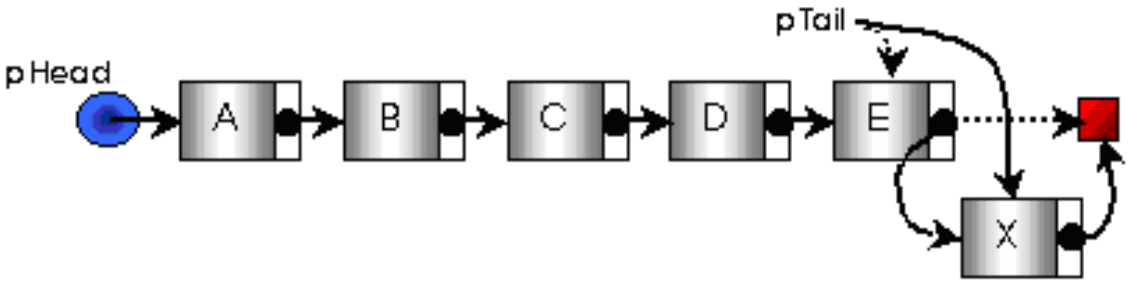
- **Chèn nút mới vào đầu xâu:**




```
void  InsertFirst(NodePtr P, NodePtr &Head, NodePtr &Tail)
{    P->Next = Head;
     if (Head == NULL) Tail = P;
     Head =  P;
}
```

Các thao tác trên xâu đơn _ Chèn nút vào xâu

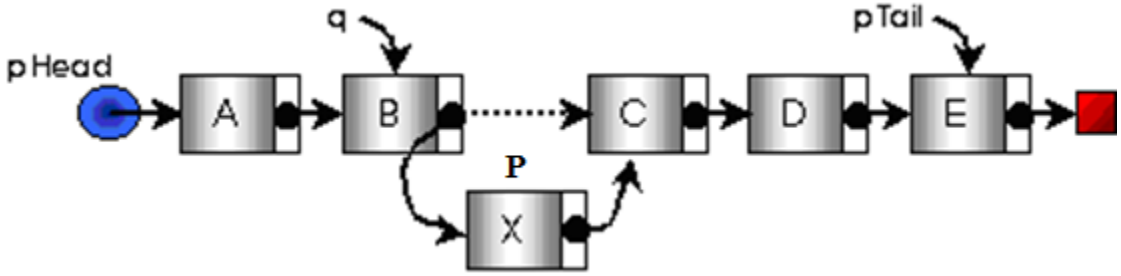
- **Chèn nút mới vào cuối xâu:**




```
void  InsertLast(NodePtr P, NodePtr &Head)
{     If (Head == NULL)
            {          Head = P; Tail = Head;          }
            else
            {          Tail->Next = P; Tail = P;       }
}
```

Các thao tác trên xâu đơn _ Chèn nút vào xâu

- **Chèn nút mới vào sau nút trỏ bởi Q:**




```
void  InsertAfter(NodePtr P, NodePtr Q, NodePtr &Tail)
{   If (Q != NULL)
    {       P->Next = Q->Next;
            Q->Next = P;
            if (Q==Tail) Tail = P;
    }
}
```

Các thao tác trên xâu đơn _ Chèn nút vào xâu

- **Chèn nút vào xâu theo thứ tự tăng của node**
- **Thuật toán:**

Bước 1: Tìm vị trí cần chèn (Ghi nhận nút đứng trước vị trí cần chèn)

Bước 2: Nếu vị trí cần chèn ở đầu xâu thì chèn vào đầu danh sách

Bước 3: Ngược lại thì chèn vào sau nút tìm được

Các thao tác trên xâu đơn _ Chèn nút vào xâu

```
void  InsertListOrder(NodePtr G,NodePtr &Head,NodePtr &Tail)
{       NodePtr  P, Q;
        P = Head;
        while (P != NULL)
        {    if (P->Data >= G->Data) break;
             Q = P;   P = Q->Next;
        }
        if (P == Head)         /*InsertFirst(G, Head, Tail)*/
        {    G->Next = Head;
             Head = G;
        }
        else            /*InsertAfter(G,Q, Tail);*/
        {    G->Next = Q->Next;
             Q->Next = G;
        }
}
```

Các thao tác trên xâu đơn _ Tạo xâu

- **Tạo một Danh sách liên kết:**

```
void CreateList (NodePtr &Head, NodePtr &Tail)
{   Head = NULL;
    do {
            Nhập gía trị mới X
            Nếu (không nhập X) thì break;
            Tạo Nút chứa X
            Chèn Nút vào xâu
    } while (1);
}
```

Các thao tác trên xâu đơn _ Tìm phần tử trong xâu

- **Thuật toán:** Áp dụng thuật toán tìm kiếm tuyến tính. Sử dụng 1 con trỏ phụ P để lần lượt trỏ đến các phần tử trong xâu.

  b1: Cho P trỏ phần tử đầu xâu: P = Head;

  b2: Trong khi chưa hết danh sách (P != NULL)

  Nếu P->Data == X thì báo tìm thấy và kết thúc ngược lại thì chuyển sang phần tử kế tiếp (P = P->Next)

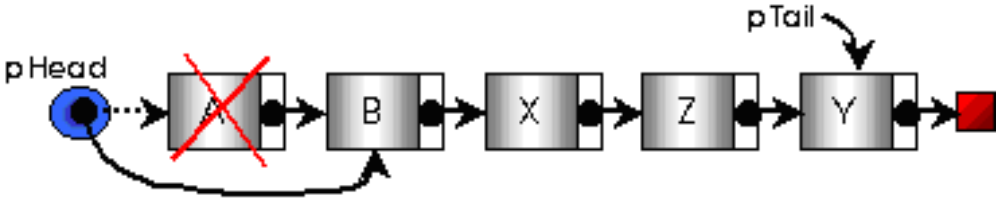
  b3: Báo không tìm thấy

Các thao tác trên xâu đơn _ Tìm phần tử trong xâu

```
NodePtr Search(T X,  NodePtr Head)
{   NodePtr P;
    P = Head;
    while (P != NULL)
            if (P->Data == X) return P;
            else P = P->Next ;
    return NULL;
}
```

Các thao tác trên xâu đơn _ Hủy nút trong xâu

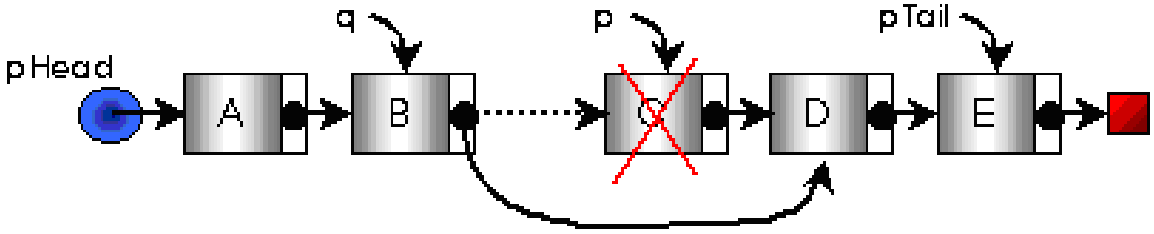
- **Hủy nút đầu xâu**




```
void DeleteFirst(NodePtr &Head, NodePtr &Tail)
{   NodePtr P ;
    if (Head != NULL)
    {       P = Head;     Head = P->Next;
            if (Head == NULL) Tail = NULL;
            free(P);
    }
}
```

Các thao tác trên xâu đơn _ Hủy nút trong xâu

- **Hủy nút sau nút trỏ bởi Q**




```
void DeleteAfter( NodePtr Q, NodePtr &Tail)
{    NodePtr  P;
     if ( Q != NULL)
     {          P = Q->Next;
                if (P != NULL)
                {          Q->Next = P->Next;       if (P == Tail) Tail = Q;
                           free(P); //delete P;
                }
     }
}
```

Các thao tác trên xâu đơn _ Hủy xâu

- **Thuật toán :**

Bước 1: Trong khi (Danh sách chưa hết) thực hiện

B11:

```
p = Head;
Head:=Head->pNext; // p trỏ tới phần tử kế
```

B12:

```
Hủy p;
```

Bước 2:

```
Tail = NULL; //Bảo đảm tính nhất quán khi xâu rỗng
```

Các thao tác trên xâu đơn _ Tìm phần tử trong xâu

```
void RemoveList(NodePtr &Head, NodePtr &Tail)
{   NodePtr p;
    while (Head!= NULL)
    {
            p = Head;
            Head = p->Next;
            delete p;
    }
    Tail = NULL;
}
```

Các thao tác trên xâu đơn _ Duyệt xâu

```
void TraverseList(NodePtr Head)
{   NodePtr P, Q;
    P = Head;
    while (P != NULL)
    {       Q = P;
            P = Q->Next;
            Xu_Ly_Nut(Q);
    }
}
```

Các thao tác trên xâu đơn _ Sắp xếp xâu

- **Ý tưởng:** Tạo xâu mới có thứ tự từ xâu cũ (đồng thời hủy xâu cũ)
- **Thuật toán:**
  - **B1**: Khởi tạo xâu mới Result rỗng;
  - **B2**: Tách phần tử đầu xâu cũ ra khỏi danh sách
  - **B3**: Chèn phần tử đó vào xâu Result theo đúng thứ tự sắp xếp.
  - **B5**: Lặp lại bước 2 trong khi xâu cũ chưa rỗng.

Các thao tác trên xâu đơn _ Sắp xếp xâu

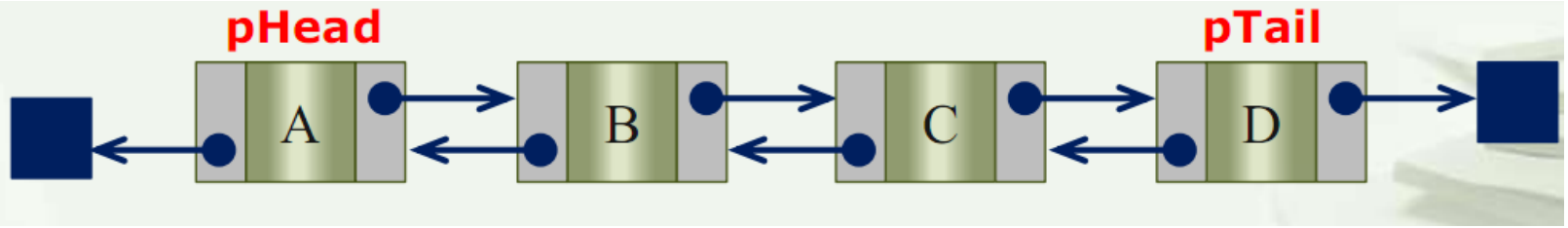
```
Void SapXep(NodePtr &Head, NodePtr & Tail)
{   NodePtr H,T, P;
    H = T = NULL;
    while (Head != NULL)
    {       P = Head; Head = P->Next; P->Next = NULL;
            InsertListOrder(P,H,T);
    }
    Head = H; Tail = T;
}
```

Danh sách liên kết đôi

- Là danh sách liên kết mà mỗi phần tử có 2 liên kết đến phần tử liền trước và liền sau nó.
- Cấu trúc một node của danh sách liên kết đôi

Danh sách liên kết đôi

- Cấu trúc danh sách liên kết đôi



Danh sách liên kết đôi

- Tổ chức biểu diễn một node của liên kết đôi

```
typedef  struct Node
{    KiểuT Data;
     struct Node *Next;
     struct Node *Prev;
} NodeType
```

- **Khai báo con trỏ đầu xâu:** NodeType *Head, * Tail;
- **khai báo kiểu con trỏ đến kiểu nút :**

*typedef NodeType *NodePtr;*

NodePtr Head, Tail;

Danh sách liên kết đôi

Các thao tác trên xâu đôi

- Khởi tạo xâu rỗng
- Kiểm tra xâu rỗng
- Tạo nút mới
- Chèn nút vào xâu
- Tạo xâu
- Hủy nút trên xâu
- Tìm kiếm giá trị
- Duyệt xâu
- Sắp xếp dữ liệu

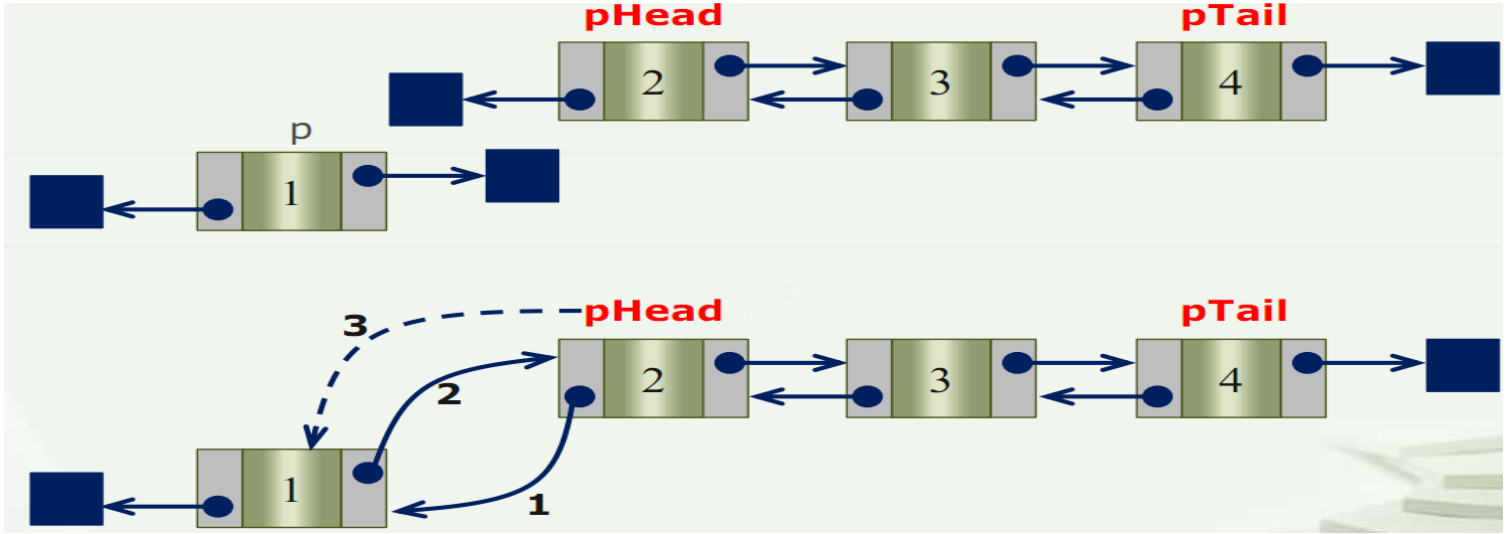
Các thao tác trên xâu kép_ Tạo nút chứa giá trị X

- **Cài đặt**

```
NodePtr CreateNode( T  x)
{   NodePtr        P;
    P = (NodePtr) malloc(sizeof(NodeType));
    if (p == NULL) {printf("Khong du bo nho"); exit(1);}
    P->Data = x;
    P->Next = NULL;
    P->Prev=NULL
    return P;
}
```

Các thao tác trên xâu đôi_ Chèn nút vào xâu

- **Chèn nút mới vào đầu xâu:**

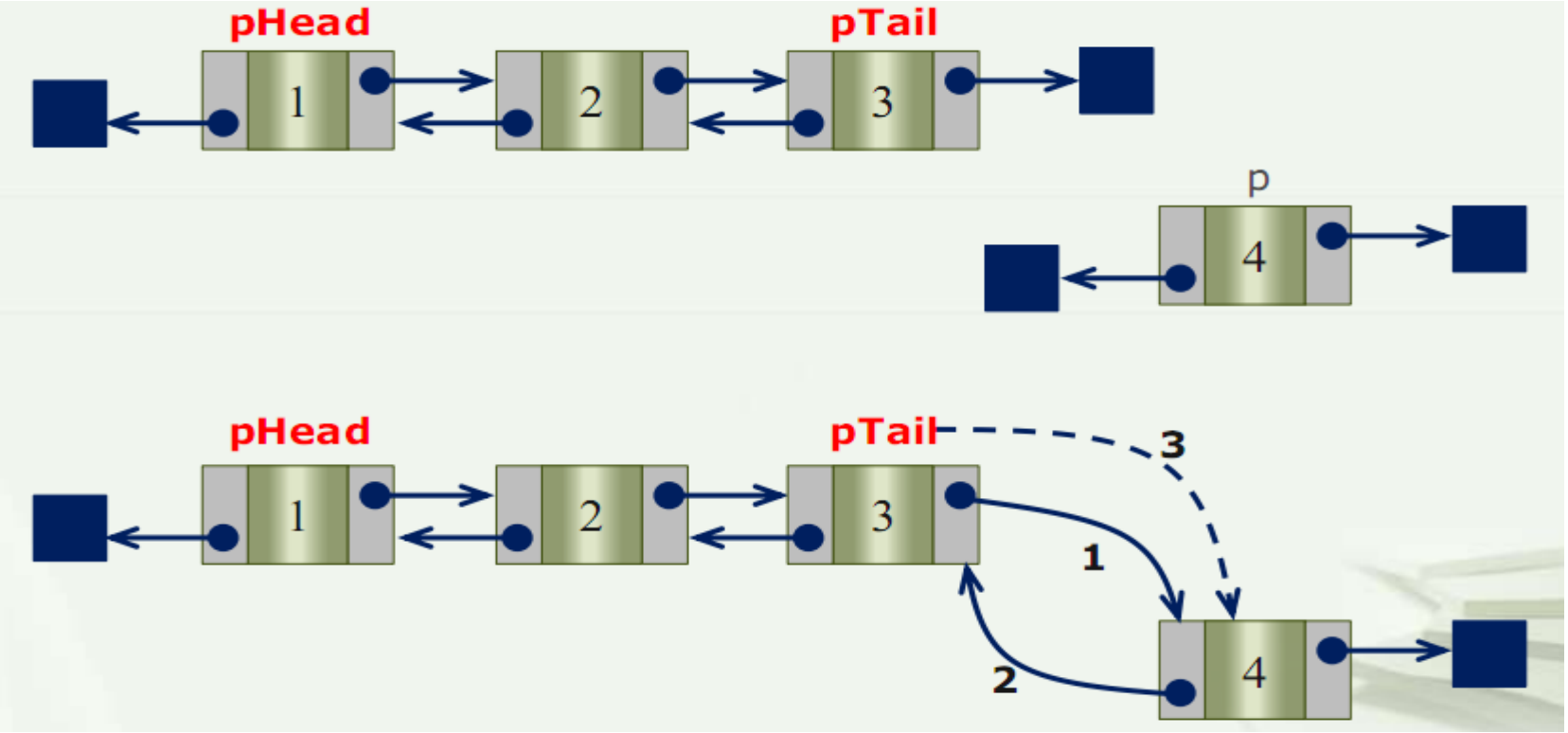


Các thao tác trên xâu đôi_ Chèn nút vào xâu

```
void  InsertFirst(NodePtr P, NodePtr &Head, NodePtr &Tail)
{     if(Head==NULL)
        {       Head=p;  Tail=p;  }
      else
      {
              p->Next=Head;
              Head->Prev=p;
              Head=p;
       }
}
```

Các thao tác trên xâu đôi_ Chèn nút vào xâu

- **Chèn nút mới vào cuối xâu:**



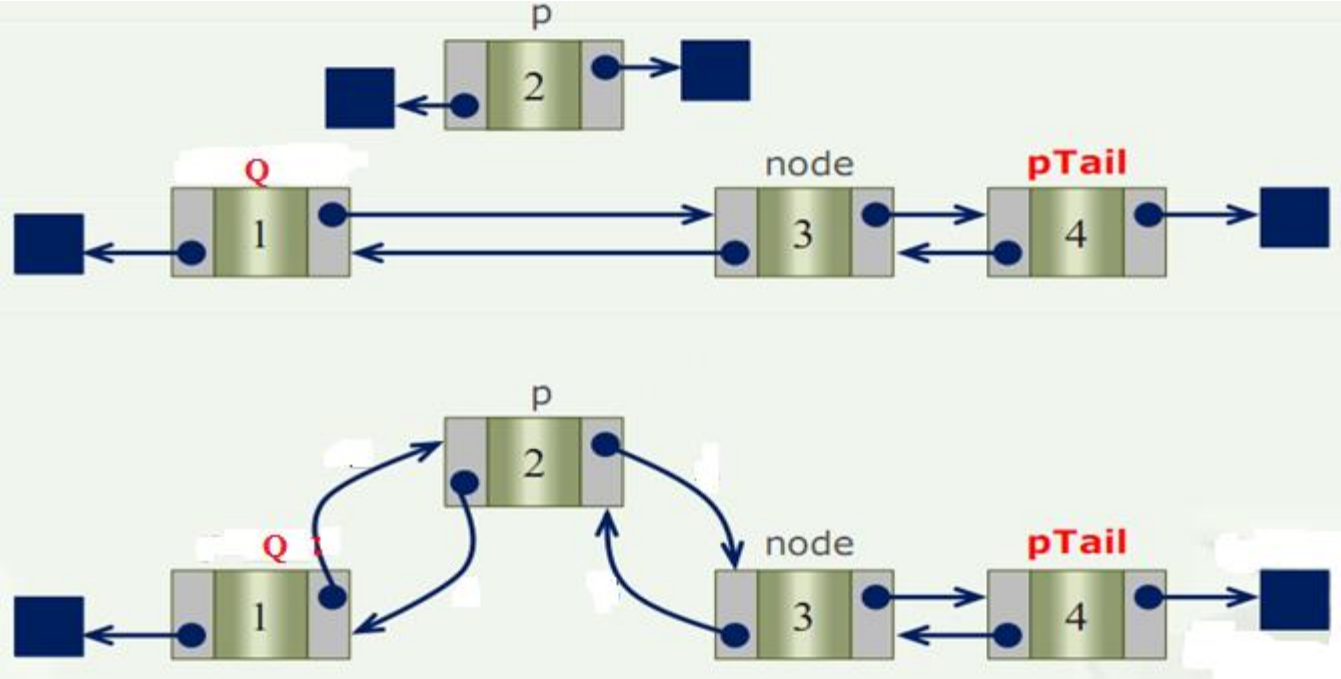
Các thao tác trên xâu đôi_ Chèn nút vào xâu

- **Chèn nút mới vào cuối xâu:**

```
void  InsertLast(NodePtr P, NodePtr &Head, Nodeptr &Tail)
    {       NodePtr Last ;
            If (Head == NULL)
            {       Head = P; Tail = Head;          }
            else
            {       Tail->Next = P;
                    p->Prev=Tail;
                    Tail = P;
            }
    }
```

Các thao tác trên xâu đôi_ Chèn nút vào xâu

- **Thêm node p vào trước node cho trước**

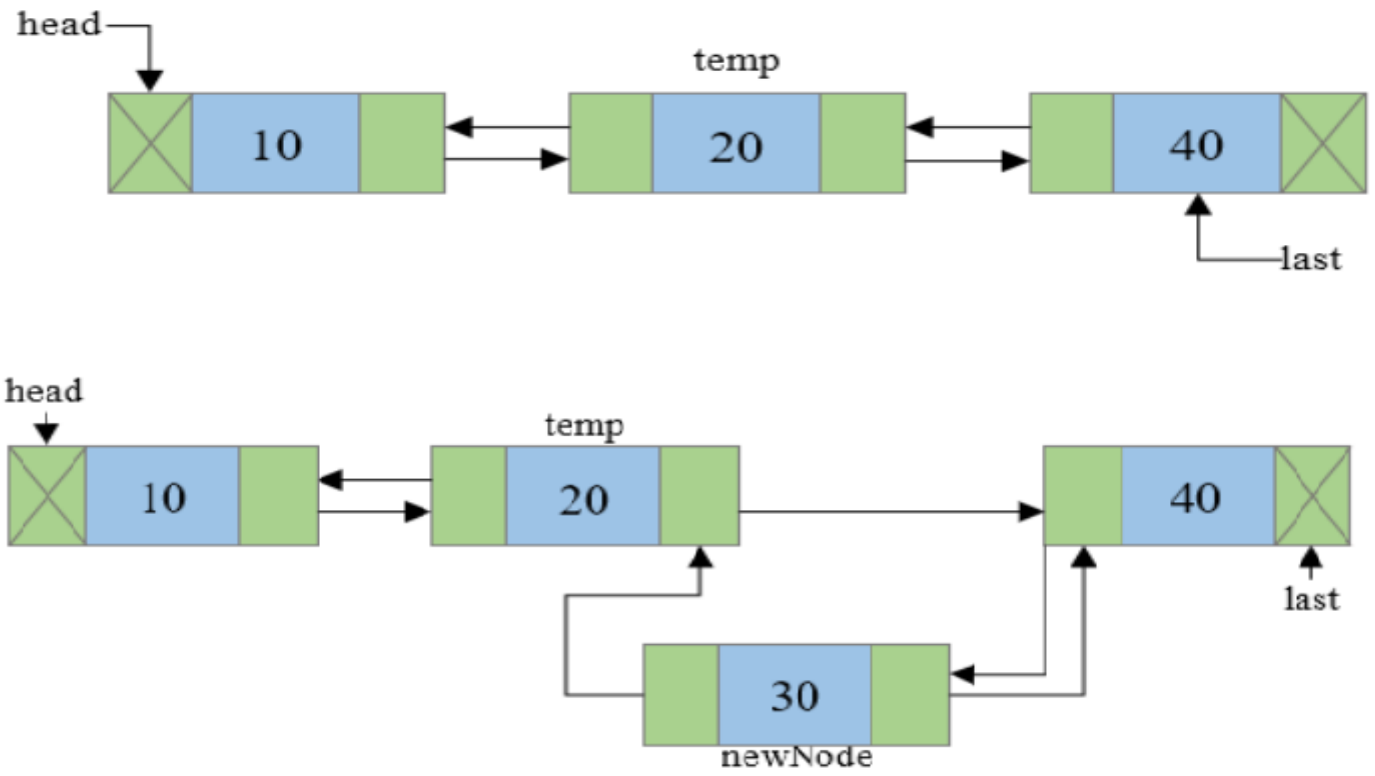


Các thao tác trên xâu đôi_ Chèn nút vào xâu

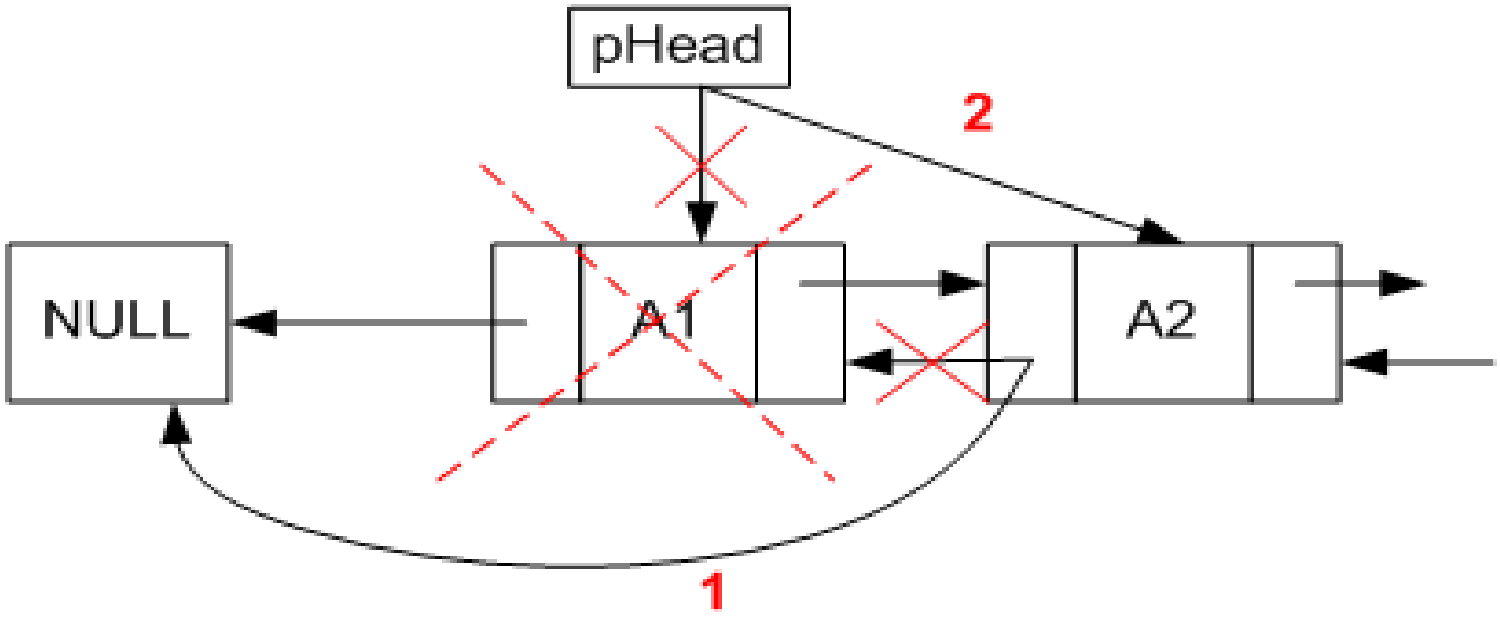
- **Thêm node p vào trước node cho trước**

```
void  InsertBefore(NodePtr P, Nodeptr Node, Nodeptr
&Head,NodePtr &Tail)
{   If (Node != NULL)
    {       if(Node==Head) InsertFirst(p,Node,Tail);
            P->Next = Q->Next;
            Node->Prev=P;
            Q->Next = P;
            P->Prev=Q;
    }
}
```

Các thao tác trên xâu đôi_ Chèn nút theo thứ tự



Các thao tác trên xâu đôi_ Hủy nút trong xâu

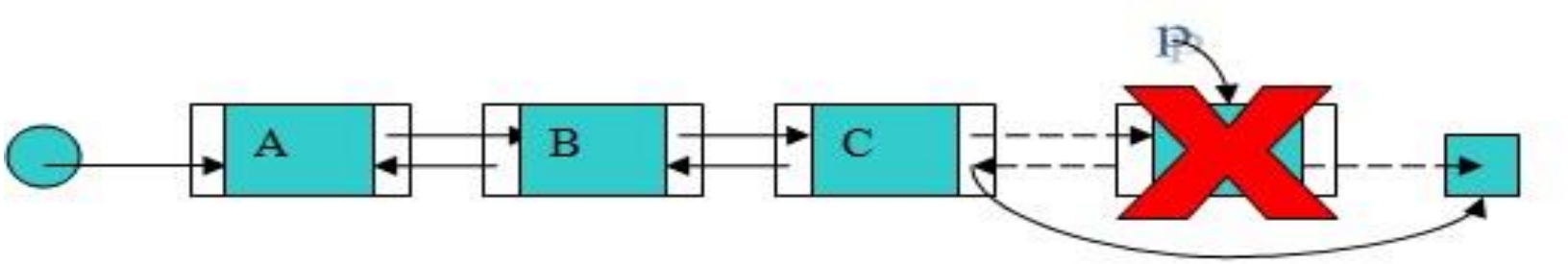


Các thao tác trên xâu đôi_ Hủy nút trong xâu

- **Hủy nút đầu xâu**

```
void DelFirst(Nodeptr &Head, Nodeptr &Tail){
        Nodeptr p;
        if(Head==NULL)   // danh sach rong
                cout<<"\n Linked List Empty";
        else
        if(Head->Next==NULL) {   // Danh sach co 1 nut
                p=Head;
                Head=p->Next;
                p->Next=NULL;
                Head=Tail=NULL;
                free(p);   }
        else{                                   // nhieu hon 1 nut
                p=Head;
                Head=p->Next;
                p->Next=NULL;
                Head->Prev=NULL;
                free(p);   }}
```

Các thao tác trên xâu đôi_ Hủy nút cuối xâu

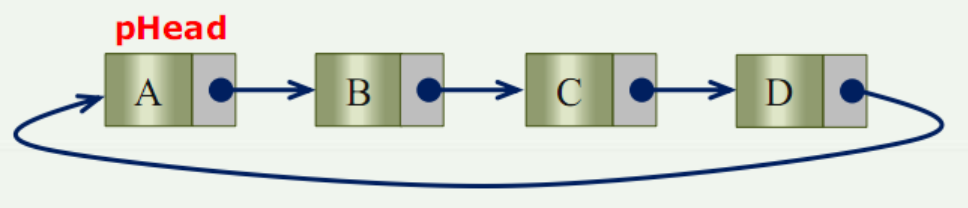


Các thao tác trên xâu đôi_ Hủy nút cuối xâu

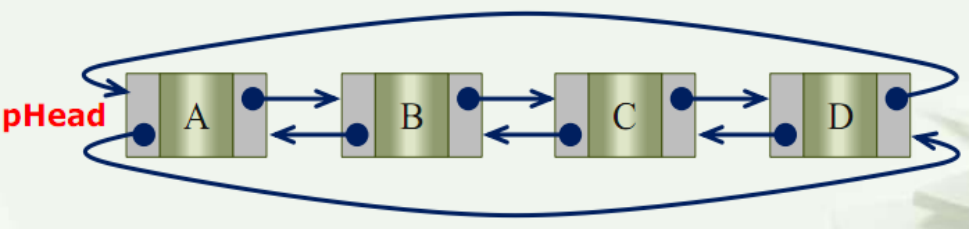
```
void DelLast(Nodeptr &Head, Nodeptr &Tail){

        Nodeptr p;
        if(Head==NULL)                          // danh sach rong
                cout<<"\n Linked list empty";
        else{
                p=Tail;
                if(p==Head)                     // danh sach co 1 nut
                {
                        Head=Tail=NULL;
                        free(p);}
                else                            // danh sach nhieu hon 1 nut
                {
                        Tail=Tail->Prev;
                        Tail->Next=NULL;
                        p->Prev=NULL;
                        free(p);
                }}}
```

Danh sách liên kết vòng

- Cấu trúc danh sách liên kết vòng
  - Biểu diễn bằng danh sách liên kết đơn



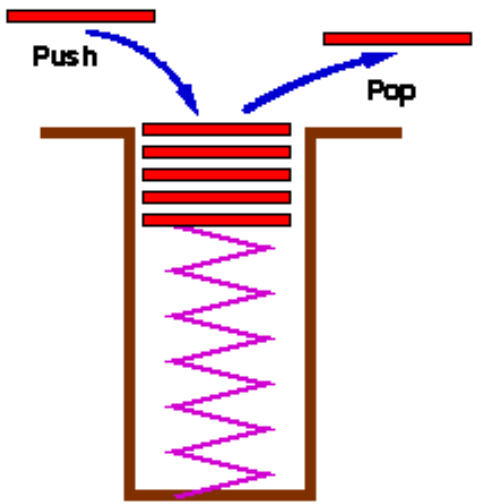
  - Biểu diễn bằng danh sách liên kết đôi



Ngăn xếp _ Stack

- Ngăn xếp thường được sử dụng để lưu trữ dữ liệu tạm thời trong quá trình chờ xử lý theo nguyên tắc: **vào sau ra trước (Last In First Out - LIFO)**
- **Khai báo Cấu trúc dữ liệu (dùng xâu đơn)**

```
typedef <Định nghĩa kiểu T>;
typedef struct Node
{       T  Data;
        struct Node *Next; //Con trỏ tới nút kế
} NodeType;
typedef NodeType *StackPtr;
```



- **Khai báo con trỏ đầu Stack**

```
StackPtr  Stack;
```

Các thao tác trên Stack (dùng xâu đơn)

- **Tạo Stack Rỗng:** Stack = NULL;
- **Kiểm tra Ngăn xếp Rỗng:** if (Stack == NULL)..
- **Thêm 1 phần tử X vào đầu Stack:**

```
void Push(DataType X , StackPtr &Stack )
{
    NodePtr  P;
    P = CreateNode(x);
    P->Next = Stack; /*InsertFirst(P, Stack);*/
    Stack = P;
}
```

Các thao tác trên Stack (dùng xâu đơn)

- **Lấy phần tử ở đỉnh Stack**

```
KieuT  Pop(StackPtr &Stack)
{   DataType  x;
    if (Stack!=NULL)
    {       x = (Stack)->Data; /*Xoa nut dau*/
            P = Stack;
            Stack = P->Next;
            free(P);
            return x;
    }
}
```

Ngăn xếp _ Stack

- **Khai báo Cấu trúc dữ liệu (dùng mảng)**
  - #define MaxSize 100 /*Kích thước Stack*/
  - typedef <Định nghiã kiểu T >
- **Khai báo kiểu mảng:**
  - typedef KiểuT StackArray[MaxSize];
- **Khai báo một Stack:**
  - StackArray Stack; int top; //chỉ mục phần tử đầu Stack

Các thao tác trên Stack (dùng mảng)

- **Khởi tạo 1 Stack rỗng:** top = -1
- **Kiểm tra ngăn xếp rỗng:** if (top == -1)...
- **Kiểm tra ngăn xếp đầy:** if(top == MaxSize-1)...
- **Thêm 1 phần tử có nội dung x vào đầu Stack:**

```
void Push(KieuT x, StackArray Stack, int &top)
{
    if (top < MaxSize-1)
    {       top++;  Stack[top]= x;
    }
}
```

Các thao tác trên Stack (dùng mảng)

- **Lấy phần tử ở đỉnh Stack**

```
KieuT Pop(StackArray Stack, int top)
{
    KieuT  Item;
    if (top != -1)
    {       Item = Stack[top];  top--;
            return Item;
    }
}
```

Ứng dụng của Stack

- Chuyển đổi các hệ thống số
  - Thập phân → nhị phân
  - Nhị phân → thập phân
  - ....
- Xử lý biểu thức hậu tố
  - Chuyển đổi biểu thức ngoặc toàn phần sang biểu thức tiền tố, trung tố, hậu tố
  - Ước lượng giá trị các biểu thức
  - ...
- ...

Hàng đợi _ Queue

- Loại danh sách này có hành vi giống như việc xếp hàng chờ mua vé, với qui tắc **Đến trước - Mua trước. (First in First Out - FIFO)**

  Ví dụ: Bộ đệm bàn phím, tổ chức công việc chờ in trong Print Manager của Windows
- Hàng đợi là một kiểu danh sách đặc biệt có
  - Các thao tác chèn thêm dữ liệu đều thực hiện ở cuối danh sách
  - Các thao tác lấy dữ liệu được thực hiện ở đầu danh sách.

Hàng đợi _ Queue

Các thao tác trên hàng đợi

- Khai báo hang đợi
- Khởi tạo hàng đợi rỗng
- Kiểm tra hàng đợi rỗng
- Chèn dữ liệu X vào cuối hàng đợi
- Lấy dữ liệu từ đầu hàng đợi

Hàng đợi _ Queue

- **Khai báo Cấu trúc dữ liệu (dùng xâu đơn)**

```
typedef        <Định nghĩa kiểu T>;
typedef struct Node
{        T  Data;
         struct Node *Next; //Con trỏ tới nút kế
} NodeType;
typedef NodeType *QueuePtr;
```

- **Khai báo con trỏ**

```
QueuePtr  Head, Tail;
```

Các thao tác trên Queue (dùng xâu đơn)

- **Khởi tạo hàng đợi rỗng:** Head = NULL; Tail = NULL;
- **Kiểm tra hàng đợi rỗng:** if (Head == NULL)...
- **Chèn dữ liệu X vào cuối hàng đợi:**

```
void Push( KieuT x, QueuePtr &Head, QueuePtr &Tail )
{   QueuePtr P;
    P = CreateNode(x);
    if (Head == NULL){  Head = P; Tail = Head;      }
    else   {          Tail->Next = P; Tail = P;  }
}
```

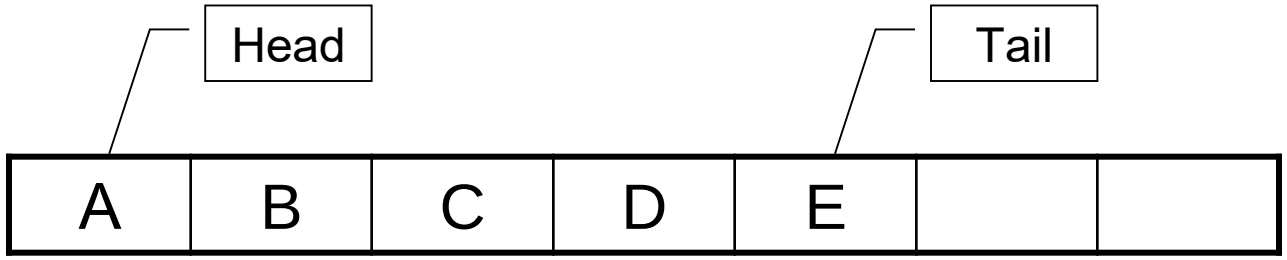
Các thao tác trên Queue (dùng xâu đơn)

- **Lấy dữ liệu từ đầu hàng đợi**

```
KieuT Pop( QueuePtr  &Head, QueuePtr &Tail)
{   QueuePtr P; KieuT x;
    if (Head != NULL)
    {       x = Head->Data;
            P = Head;                /* DeleteFirst(Head);*/
            Head = P->Next;
            free(P);
            If (Head == NULL) Tail = NULL;
    }
    return x;
}
```

Cài đặt Queue dùng mảng

- Sử dụng kỹ thuật xác định chỉ số vòng tròn để định vị trí đầu và cuối hàng đợi.



- Head là vị trí phần tử đầu hàng đợi. Tail là vị trí phần tử cuối hàng đợi
  - Hàng đợi rỗng: Head = Tail
  - Vị trí đầu mới = (Head + 1) mod Maxsize
  - Vị trí cuối mới = (Tail + 1) mod Maxsize
  - Hàng đợi đầy: Vị trí cuối mới = Head

Cài đặt Queue dùng mảng

- **Khai báo kích thước Queue**
  - **#define** MaxSize 100
  - typedef /* khai báo kiểu T*/
- **Khai báo cấu trúc Queue**

```
typedef struct
{           int Head, Tail;
            KiểuT  Node[MaxSize] ;
} QueueType;
QueueType  Queue ;
```

Các thao tác trên Queue (dùng mảng)

- **Khởi Tạo Queue rỗng:**

```
void CreateQ(QueueType &queue)
{    queue.Head = 0;
     queue.Tail = 0;
}
```

- **Kiểm tra hàng đợi rỗng: Head == Tail**

```
int EmptyQ(QueueType  queue )
{
     return (queue.Head == queue.Tail ? 1 : 0));
}
```

Các thao tác trên Queue (dùng mảng)

- **Thêm phần tử vào cuối hàng đợi:**

```
void AddQ(KieuT item, Queuetype &q)
{   int Vitri;
    Vitri = (q.Tail + 1)% maxsize;
    if (Vitri == q.Head)
            printf("\nHang đoi da day");  /*Day hang doi*/
    else
    {       q.Node[Tail]=Item;
            q.Tail = Vitri;
    }
}
```

Các thao tác trên Queue (dùng mảng)

- **Lấy ra 1 phần tử ở đầu hàng đợi:**

```
KieuT GetQ(QueueType  &q)
{   KieuT   Item;
    int Vitri;
    if ( ! EmptyQ(q))
    {       Vitri = (q.Head + 1) % MaxSize;
            Item  = q.Node[Vitri];
            q.Head = Vitri;
            return Item;
    }
}
```