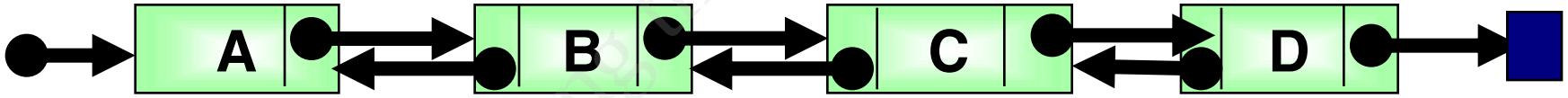
NỘI DUNG

DANH SÁCH LIÊN KẾT kép

Định Nghĩa

- Mỗi phần tử liên kết với phần tử đứng trước và sau nó trong danh sách
- Hình vẽ minh họa danh sách liên kết kép:



Cấu Trúc Dữ Liệu

- *Cấu trúc dữ liệu 1 nút*

```
typedef struct tagDnode
{   Data Info;
    struct tagDnode *pPre;
    struct tagDnode *pNext;
}DNode;
```

- *Cấu trúc List kép*

```
Typedef struct tagDList
{   DNode *pHead;
    DNode *pTail;
}DList;
```


Các Thao Tác Trên List Kép

- Khởi tạo danh sách liên kết kép rỗng
- Tạo 1 nút có thành phần dữ liệu = x
- Chèn 1 phần tử vào danh sách
  - Chèn vào đầu
  - Chèn sau phần tử Q
  - Chèn vào trước phần tử Q
  - Chèn vào cuối danh sách
- Huỷ 1 phần tử trong danh sách
  - Hủy phần tử đầu danh sách
  - Hủy phần tử cuối danh sách
  - Hủy 1 phần tử có khoá bằng x
- Tìm 1 phần tử trong danh sách
- Sắp xếp danh sách

Tạo 1 Danh Sách Rỗng

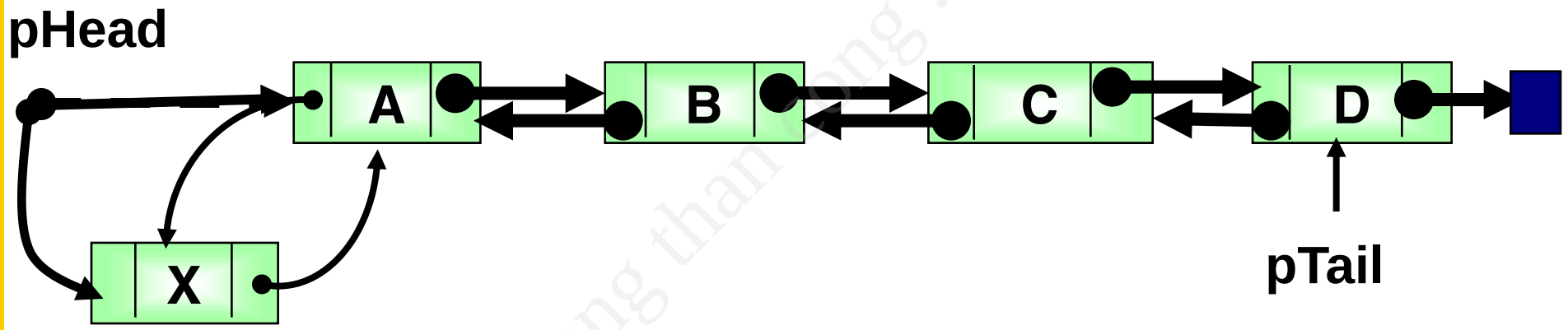
```
void CreateDList(DList &l)
{
        l.DHead=NULL;
        l.DTail=NULL;
}
```


Tạo 1 Nút Có Thành Phần Dữ Liệu = X

```
DNode *CreateDNode(int x)
{  DNode *tam;
   tam=new DNode;
   if(tam==NULL)
   {    printf("khong con du bo nho");
        exit(1);
   }
   else
   {    tam->Info=x;
        tam->pNext=NULL;
        tam->pPre=NULL;
        return tam;
   }
}
```

Thêm 1 Nút Vào Đầu Danh Sách

- Minh họa hình vẽ

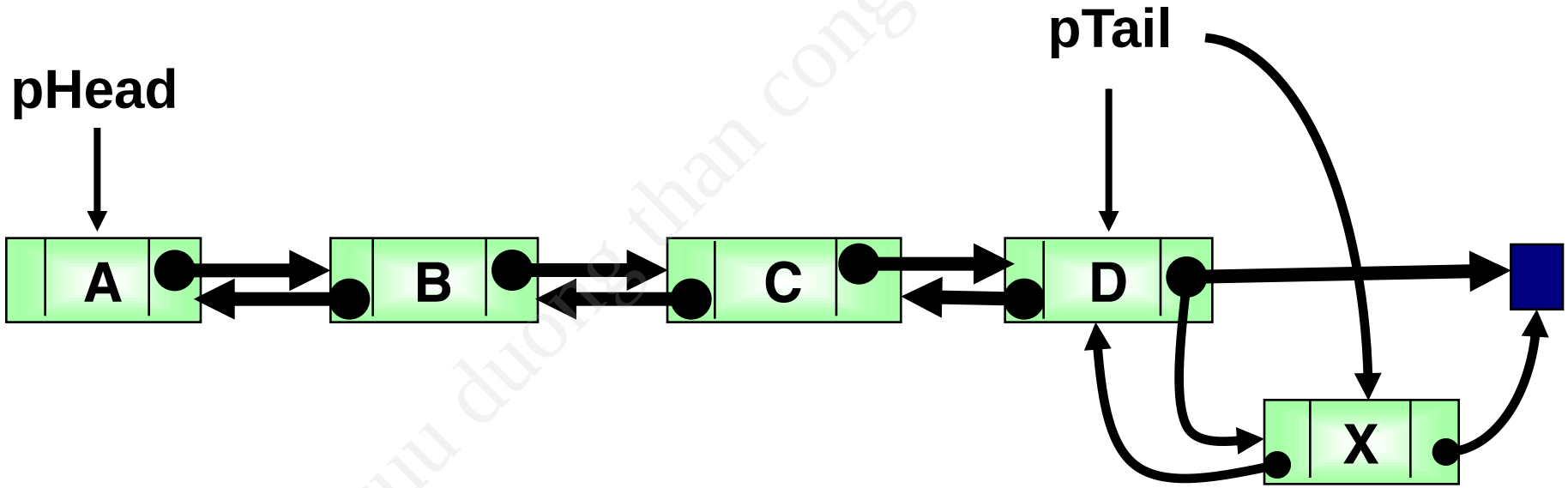


Cài Đặt Thêm 1 Nút Vào Đầu Danh Sách

```
void AddFirst(DList &l, DNode *tam)
{
   if(l.pHead==NULL)//xau rong
   {
        l.pHead=tam;
        l.pTail=l.pHead;
   }
   else
   {
        tam->pNext=l.pHead;
        l.pHead->pPre=tam;
        l.pHead=tam;
   }
}
```

Thêm Vào Cuối Danh Sách

- Minh họa thêm 1 phần tử vào sau danh sách

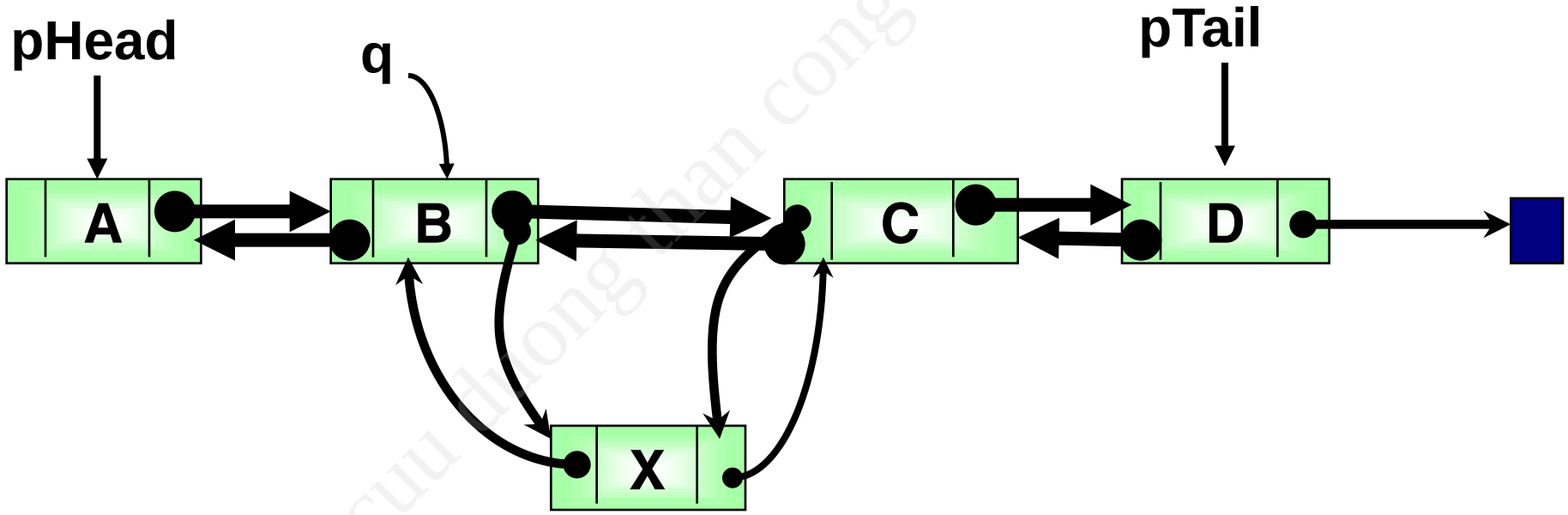


Cài Đặt Thêm 1 Nút Vào Cuối Danh Sách

```
void AddEnd(DList &l,DNode *tam)
{
   if(l.pHead==NULL)
   {
        l.pHead=tam;
        l.pTail=l.pHead;
   }
   else
   {
        tam->pPre=l.pTail;
        l.pTail->pNext=tam;
        tam=l.pTail;
   }
}
```

Thêm Vào Sau Nút Q

- Minh họa thêm nút X vào sau nút q

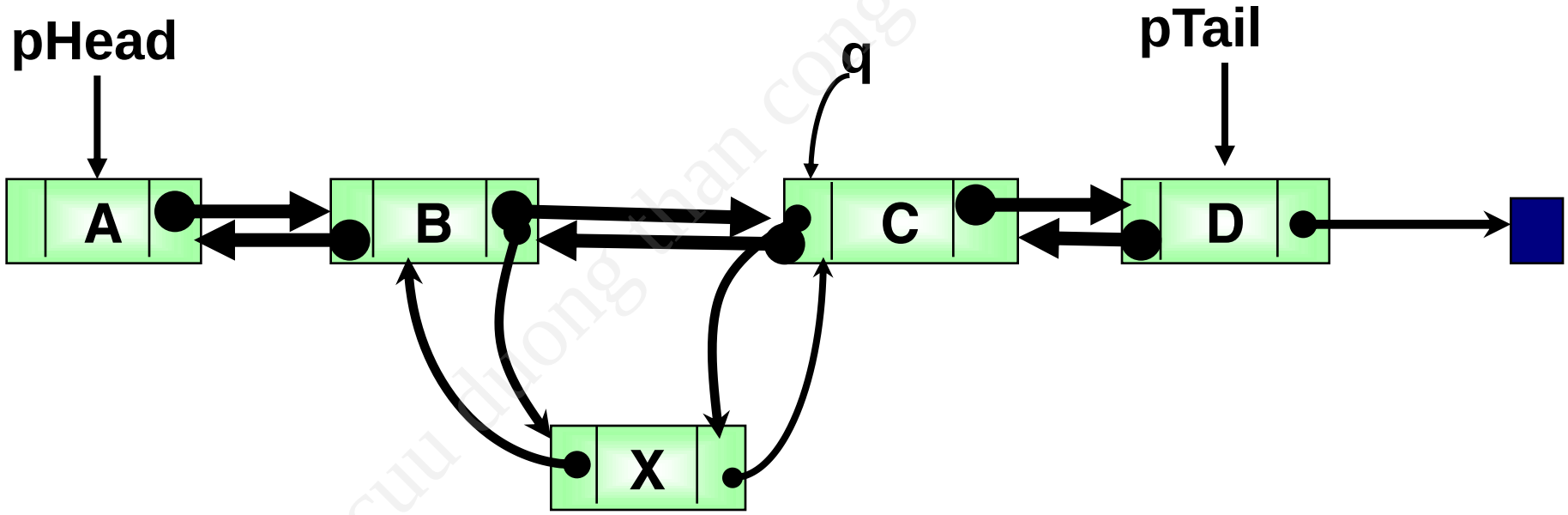


Cài Đặt Thêm 1 Nút Vào Sau Nút Q

```
void AddLastQ(DList &l,DNode *tam, DNode *q)
{
    DNode *p;
    p=q->pNext;
    if(q!=NULL)//them vao duoc
    {
        tam->pNext=p;
        tam->pPre=q;
        q->pNext=tam;
        if(p!=NULL)
            p->pPre=tam;
        if(q==l.pTail) //them vao sau danh sach lien ket.
            l.pTail=tam;
    }
    else
        AddFirst(l,tam);
}
```

Thêm 1 Nút Vào Trước Nút Q

- Minh hoạ thêm 1 nút vào trước nút q



Cài Đặt Thêm 1 Nút Vào Trước Nút Q

```
void AddBeforeQ(DList &l,DNode *tam,DNode *q)
{   DNode *p;
    p=q->pPre;
    if(q!=NULL)
    {
        tam->pNext=q;
        q->pPre=tam;
        tam->pPre=p;
        if(p!=NULL)
            p->pNext=tam;
        if(q==l.pHead)
            l.pHead = tam;
    }
    else
        AddEnd(l,tam);
}
```

Xoá Phần Tử Đầu Danh Sách

```
void DeleteFirst(DList &l)
{
    DNode *p;
    if(l.pHead!=NULL)
    {
        p=l.pHead;
        l.pHead=l.pHead->pNext;
        l.pHead->pPre=NULL;
        delete p;
        if(l.pHead==NULL)
            l.pTail=NULL;
    }
}
```


Xoá 1 Phần Tử Cuối Danh Sách

```
void DeleteEnd(DList &l )
{
    DNode *p;
    if(l.pHead!=NULL) //tuc xau co hon mot phan tu
    {
        p=l.pTail;
        l.pTail=l.pTail->Pre;
        l.pTail->pNext=NULL;
        delete p;
        if(l.pTail==NULL)
            l.pHead=NULL;
    }
}
```

Hủy 1 Nút Sau Nút Q

```
void DeleteLastQ(DList &l,DNode *q)
{
    DNode *p;//luu node dung sau node q
    if(q!=NULL)
    {
        p=q->pNext;
        if(p!=NULL)
        {
            q->pNext=p->pNext;
            if(p==l.pTail)//xoa dung nu't cuoi
                l.pTail=q;
            else     //Nut xoa khong phai nut cuoi
                p->pNext->pPre=q;
            delete p;
        }
    }
    else
        DeleteFirst(l);
}
```

Huỷ 1 Nút Đứng Trước Nút Q

```
void DeleteBeforeQ(DList &l,DNode *q)
{
    DNode *p;
    if(q!=NULL) //tuc ton tai node q
    {
        p=q->pPre;
        if(p!=NULL)
        {
            q->pPre=p->pPre;
            if(p==l.pHead)//p la Node dau cua danh sach
                l.pHead=q;
            else //p khong phai la node dau
                p->pPre->pNext=q;
            delete p;
        }
    }
    else
        DeleteEnd(l);
}
```


Xoá 1 Phần Tử Có Khoá = X

```
int DeleteX(DList &l,int x)
{
    DNode *p;
    DNode *q;
    q=NULL;
    p=l.pHead;
    while(p!=NULL)
    {
        if(p->Info==x)
            break;
        q=p;//q la Node co truong Info = x
        p=p->pNext;
    }
    if(q==NULL) return 0;//khong tim thay Node nao co truong Info =x
    if(q!=NULL)
        DeleteLastQ(l,q);
    else
        DeleteFirst(l);
    return 1;
}
```

Sắp Xếp

```
void DoiChoTrucTiep(DList &l)
{  DNode *p,*q;
   p=l.pHead;
   while(p!=l.pTail)
   {
        q=p->pNext;
        while(q!=NULL)
        {
             if(p->Info>q->Info)
                  HV(p,q);
             q=q->pNext;
        }
        p=p->pNext;
   }}
```

Hàng đợi 2 đầu

- Hàng đợi hai đầu (gọi tắt là deque) là một vật chứa các đối tượng mà việc thêm hoặc hủy một đối tượng được thực hiện ở cả 2 đầu của nó.
- Deque là một CTDL trừu tượng (ADT) hỗ trợ các thao tác chính sau:

- InsertFirst(e):Thêm đối tượng e vào đầu deque
- InsertLast(e):Thêm đối tượng e vào cuối deque
- RemoveFirst():Lấy đối tượng ở đầu deque ra khỏi deque và trả về giá trị của nó.
- RemoveLast():Lấy đối tượng ở cuối deque ra khỏi deque và trả về giá trị của nó.

Hàng đợi 2 đầu

- Ngoài ra, deque cũng hỗ trợ các thao tác sau:
- IsEmpty():Kiểm tra xem deque có rỗng không.
- First():Trả về giá trị của phần tử nằm ở đầu deque mà không hủy nó.
- Last():Trả về giá trị của phần tử nằm ở cuối deque mà không hủy nó.

Hàng đợi 2 đầu

- Do đặc tính truy xuất hai đầu của deque, việc xây dựng CTDL biểu diễn nó phải phù hợp.
- Ta có thể cài đặt CTDL hàng đợi hai đầu bằng danh sách liên kết đơn. Tuy nhiên, khi đó thao tác RemoveLast hủy phần tử ở cuối deque sẽ tốn chi phí O(n). Điều này làm giảm hiệu quả của CTDL.
- Thích hợp nhất để cài đặt deque là dùng danh sách liên kết kép. Tất cả các thao tác trên deque khi đó sẽ chỉ tốn chi phí O(1).

Danh sách liên kết có thứ tự (Ordered List)

- Danh sách liên kết có thứ tự (gọi tắt là OList) là một vật chứa các đối tượng theo một trình tự nhất định. Trình tự này thường là một khóa sắp xếp nào đó. Việc thêm một đối tượng vào OList phải bảo đảm tôn trọng thứ tự này.
- Ta có thể cài đặt OList bằng DSLK đơn hoặc DSLK đôi với việc định nghĩa lại duy nhất một phép thêm phần tử: thêm bảo toàn thứ tự. Nghĩa là trên OList chỉ cho phép một thao tác thêm phần tử sao cho thứ tự định nghĩa trên OList phải bảo toàn.

Danh sách liên kết có thứ tự (Ordered List)

- Cài đặt OList bằng DSLK đơn hoặc DSLK đôi với việc định nghĩa lại duy nhất một phép thêm phần tử: thêm bảo toàn thứ tự. Nghĩa là trên OList chỉ cho phép một thao tác thêm phần tử sao cho thứ tự định nghĩa trên OList phải bảo toàn.

- Có thể dùng OList để cài đặt CTDL hàng đợi có độ ưu tiên.

Câu hỏi và Bài tập

1. Hãy cho biết nội dung của stack sau mỗi thao tác trong dãy: EAS*Y**QUE***ST***I*ON

Với một chữ cái tượng trưng cho thao tác thêm chữ cái tương ứng vào stack, dấu * tượng trưng cho thao tác lấy nội dung một phần tử trong stack in lên màn hình.

Hãy cho biết sau khi hoàn tất chuỗi thao tác, những gì xuất hiện trên màn hình?

2. Áp dụng bài tập 1 với hàng đợi.

Câu hỏi và Bài tập

3. Trình bày thuật toán và cài đặt các thao tác trên danh sách liên kết đôi trong trường hợp chỉ quản lý bằng con trỏ đầu trong danh sách.

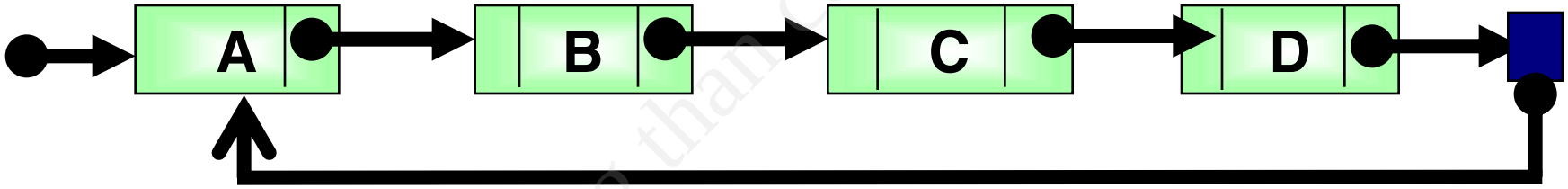
4. Cài đặt thuật toán Merge trên danh sách liên kết đôi.

5. Cài đặt chương trình cho phép thực hiện các phép tính +,-,*,/ trên các số có tối đa 30 chữ số, có chức năng nhớ (M+, M-, MC, MR).

6. Viết chương trình thực hiện các thao tác trên đa thức.

Danh sách liên kết vòng

Là một danh sách đơn (hoặc kép) mà phần tử cuối danh sách thay vì mang giá trị NULL, trỏ tới phần tử đầu danh sách.



Tìm 1 phần tử trong DSLK vòng

```
Node *Search(LIST &l, Data  x)
{
        Node            *p;
        p = l.pHead;
        do{
                if (p->Info == x)
                        return p;
                p = p->pNext;
        }while(p!=l.pHead);
    return p;
}
```

Thêm phần tử đầu trong DSLK vòng

```
void AddHead(LIST &l, Node* new_ele)
{
        if (l.pHead==NULL) {
                l.pHead = l.pTail = new_ele;
                l.pTail->pNext = l.pHead;
        }
        else{
                new_ele->pNext = l.pHead;
                l.pTail->pNext = new_ele;
                l.pHead = new_ele;
        }
}
```

Thêm phần tử cuối trong DSLK vòng

```
void AddTail(LIST &l, Node *new_ele)
  {
      if (l.pHead==NULL)  {
            l.pHead = l.pTail = new_ele;
            l.pTail->pNext = l.pHead;
      }
      else{
            new_ele->pNext = l.pHead;
            l.pTail->pNext = new_ele;
            l.pTail = new_ele;
      }
  }
```

Thêm phần tử sau nút q trong DSLK vòng

```
void AddAfterQ(List &l, Node *q, Node *new_ele)
 {
    if(l.pHead==NULL)
    {
       l.pHead= l.pTail=new_ele;
       l.pTail->pNext=l.pHead;
    }
    else
        new_ele ->pNext=q->Next;
       q->pNext= new_ele;
       if(q==l.pTail)
              l.Tail= new_ele;
}
```

Hủy phần tử đầu trong DSLK vòng

```
void RemoveHead(List &l)
   {
        Node *p = l.pHead;
        if (p==NULL)        return;
        if(l.pHead==l.pTail)
                l.pHead=l.pTail=NULL;
        else{
                l.pHead=p->pNext;
                if(p==l.pTail)
                        l.pTail->pNext=l.pHead;
        }
        delete p;
   }
```


Hủy phần tử sau nút q trong DSLK vòng

```
void RemoveAfter(List &l, Node *q)
   {
        Node *p;
        if (q!=NULL){
                p=q->pNext;
                if(p==q)
                        l.pHead=l.pTail=NULL;
                else{
                        q->pNext=p->pNext;
                        if(p==l.pTail)
                                l.pTail=q;
                }
        delete p;
        }
   }
```

Danh sách có nhiều mối liên kết

- Là danh sách mà mỗi phần tử có nhiều khóa và chúng được liên kết với nhau theo từng loại khóa.
- Thường được sử dụng trong các ứng dụng quản lý một CSDL lớn với những nhu cầu tìm kiếm dữ liệu theo những khóa khác nhau.
- Các thao tác trên một danh sách nhiều mối liên kết được tiến hành tương tự như trên danh sách đơn nhưng được thực hiện làm nhiều lần và mỗi lần cho một liên kết.

Danh sách tổng quát

- Danh sách tổng quát là 1 danh sách mà mỗi phần tử của nó có thể lại là một danh sách khác. Các ví dụ sau minh họa các cấu trúc danh sách tổng quát. Các thao tác trên 1 danh sách được xây dựng dựa trên cơ sở các thao tác trên danh sách liên kết đã khảo sát.
- Ví dụ 1:

```
typedef    struct tagNode
{
   Data info;
   struct tagNode* pNext;
   void *subList;
}Node;
typedef Node *pNode;
```


Danh sách tổng quát

- Ví dụ 2:

```
typedef    struct tagNguoi
{
   char ten[35];
   char gioitinh;
   int namsinh;
   struct tagNguoi *cha, *me;
   struct tagNguoi *anh, *chi, *em;
}Nguoi;
typedef Nguoi *pNguoi;
```

- Ví dụ 3: Đồ thị (Graph)