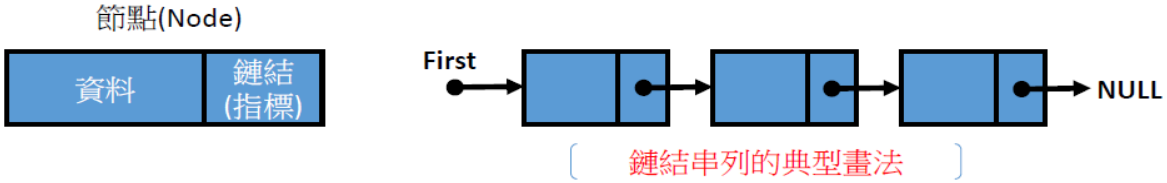
伍、鏈結串列

一、鏈結串列(Linked List)簡介

1. 何謂鏈結串列(Linked List)：是由一個或一個以上動態記憶體分配的節點(node)所組成，每個節點至少會有兩個或兩個以上的欄位，分別存放資料和指標，此指標稱為鏈結(Link)

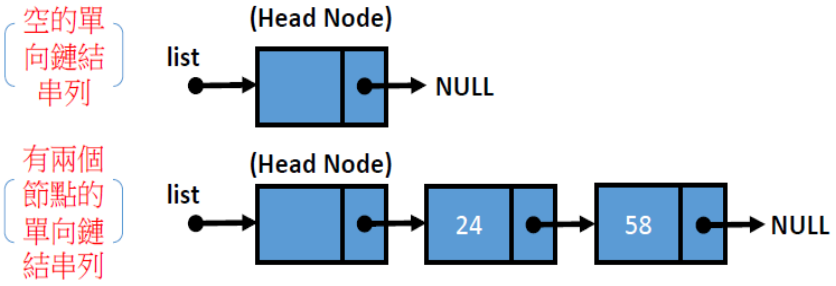


2. 以 C 的 struct 表示節點(node){假設資料為整數}

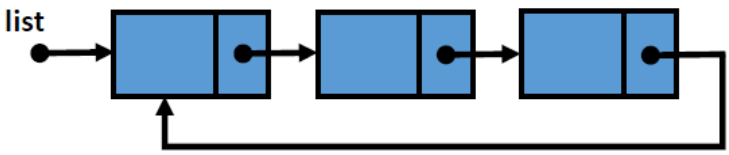
```
//NODE 定義
typedef struct node
{int data ;
struct node *link ;
} NODE ;
//產生可指向實體 NODE 的指標變數
NODE *list;
//建立標頭節點
list = (NODE*)malloc(sizeof(NODE));//配置空間產生一個實體 NODE
```

3. 鏈結串列種類

①單向鏈結串列：除了標頭節點(head node)外，由零個或零個以上的節點鏈結形成的串列，其每個節點只有一個鏈結欄位，其指向次一節點，且最後一個節點的鏈結指向 NULL

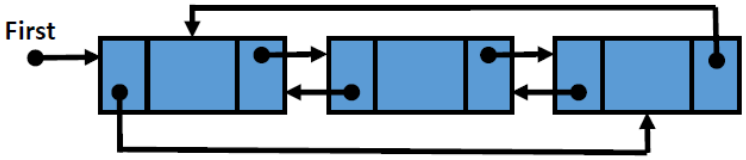


②環狀鏈結串列(Circular Linked List)：鏈結串列的最後一個節點，指向鏈結串列的第一個節點，稱之環狀鏈結串列



③雙向鏈結串列(Double Linked List)：由零個或零個以上的節點鏈結形成的串列，其每節點都有兩個鏈結欄位，一個指向次一節點，另一個指向前一個節點

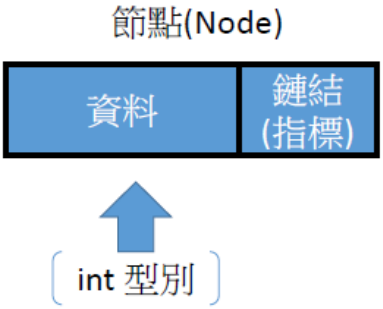
④環狀雙向鏈結串列(Circular Double Linked List)：由零個或零個以上的節點鏈結形成的串列，其每個節點都有兩個鏈結欄位，一個指向次一節點，另一個指向前一個節點，頭指向尾，尾指向頭。



二、單向鏈結串列的實作與相關運算

1.實作單向鏈結串列的節點{以 C 的 struct 實作，假設資料為 int 型別}

```
//定義節點型別
typedef struct node
{int data ;//節點內資料
struct node *link ; //指向下一個節點
} NODE ;
//建立指向串列的指標變數
NODE *list ;
```



2.單向鏈結串列的建立

```
NODE*newNode( void ) {
NODE *pt;
pt= (NODE*) malloc( sizeof(NODE) ) ; //動態配置空間給新節點
if( pt== NULL ) {
printf( "記憶體空間不足\n" );
system( "pause" ) ;
return (NODE*)0 ;
}pt->link = (NODE*)NULL ;
return pt;}
```

3.單向鏈結串列的建立(續)

```
NODE *list = newNode() ; //建立新串列的標頭節點
//將一個含有資料為整數 7 的節點，新增到串列最後端：
```

```
int appendListNode( NODE *list, intdata ) {
NODE *pt, *current;
pt= newNode( ) ;
pt->data = data ;
current = list ; //招出串列節點
while( current->list != NULL ){ current = current->list ; }
current->list = pt; //將新節點附加到串列的最末端
return 0 ;} //完成(return=1 回傳錯誤)
```

4.新節點插入到串列第一個資料節點前

```
Int insertFirstNode( NODE* list, NODE* newN){
newN->list = list->list ;
list->list = newN;
return 0; }
```

5.將新節點插入指定節點之前

```
//Find Node(插入在節點前面)
NODE *pos, *newN;
newN= newNode();
newN->data = 4 ;  //New Node
pos= findNode1( list, 3 ) ;
if( pos!= NULL ){
insertBefore( list, pos, newN) ;
} else {
printf("INSERT FAILURE!\n");}
//find *posNode(找到目標的前一個)
NODE *findNode1( NODE *list, int nodeData) {
NODE *pos;
pos= list->list ;
while( pos!= NULL && pos->list->data != nodeData)
{pos= pos->list ;}
if( pos!= NULL )return pos; //Successful
else return (NODE*)NULL ; } //Failure(回傳 0)
//Insert new node before pos->list node
intinsertBefore( NODE *list, NODE *pos, NODE* newN) {
newN->list = pos->list ; //位置複製
pos->list = newN;  //指向新節點
```

```
return 0 ;} //Succesdful
```

6.新節點插入到串列第一個資料節點後

```
//Insert new node before pos->list node
intinsertAfter( NODE *list, NODE *pos, NODE* newN) {
newN->list = pos->list ;
pos->list = newN;
return 0 ;} //Successful
//插入邏輯
NODE *pos, *newN;
newN= newNode();
newN->data = 4 ; //New Node
pos= findNode2( list, 3 ) ;  //找尋位置
if( pos!= NULL ){   //Check
insertAfter( list, pos, newN) ;
} else {
printf("INSERT FAILURE!\n");}
//find *posNode(找到要找的)
NODE *findNode2( NODE *list, int nodeData) {
NODE *pos;
pos= list->list ;
while( pos!= NULL && pos->data != nodeData) {
pos= pos->list ;}
if( pos!= NULL )return pos; //Successful
else return (NODE*)NULL ;} //Failure(回傳 0)
```

7.刪除指定節點

```
int deleteNode( NODE *list, NODE *delN) {
NODE *preNode;  //Front
preNode= list ;
while(preNode->list != NULL && preNode->list != delN) {
preNode= preNode->list ;}
if( preNode->list != NULL ) {
preNode->list = delN->list ;
delN->list = (NODE*)NULL ;
return 0 ; //Successful
} else {
```

```
return 1;}} //Failure
```

三、環狀鏈結串列的相關運算

1.以結構描述節點{此部分雷同於單向鏈結串列的節點描述}

```
typedef struct node
{
int data ; //節點內資料
struct node *link ; //指向下一個節點
} NODE ;
//建立指向串列的指標變數
NODE *list ;
```

2.環狀鏈結串列-建立空節點

```
NODE *newNode( void ) {
NODE *pt;
pt= (NODE*) malloc( sizeof(NODE) ) ; //動態配置空間給新節點
if( pt== NULL ) {
printf("記憶體空間不足\n") ;
system( "pause" ) ;
return (NODE*)0 ;}
pt->link = (NODE*)NULL ; //讓 link 不要指向任何 data
return pt;}
```

3.環狀鏈結串列-建立空串列

```
NODE *list = newNode() ; //建立新串列的標頭節點
list->link = list ; //list 放進 data 裡面
```

4.查詢串列是否為空串列(亦可 int)

```
#include <stdbool.h>
bool isEmpty( NODE* list ){
if ( list->link == list ) return true; //link=自己為空
else return false;}
```

5.將新節點插入串列的第一個資料節點之前

```
int insertFirst( NODE *list, NODE *newN) {
newN->link = list->link ; //新節點指向標頭節點的下個節點
list->link = newN; //標頭節點指向新節點
return 0;}
```

6.將新節點插入串列的第一個資料節點之前(續)

```
int insertFirst( NODE *list, NODE *newN) {
```

```
newN->link = list->link ;
list->link = newN;
return 0;}
```

7.尋找最後一個資料節點

```
NODE *findRear( NODE *list ) {
NODE *rearN;
rearN= list ;
while( rearN->link != list ) { rearN= rearN->link; }
return rearN;}
```

8.將新節點插入串列的最後一個資料節點之後

```
int insertRear( NODE *rearN, NODE *newN) {
newN->link = rearN->link ;
rearN->link = newN;
return 0 ;}
```

9.尋找串列內特定節點(data 有指定值者)

```
NODE*findNode( NODE *list, intdata ) {
NODE *pN;
if( isEmpty( list ) ) return (NODE*)NULL;
pN= list->link ; //指向第一個資料節點
do{
if( pN->data != data ){
pN= pN->link ; //移往下一個節點
} else {
return pN; //找到→回傳位址
}
}while(pN!= list) ; //若還沒看到最後一個節點，繼續
return (NODE*)NULL;} //沒找到
```

10.尋找串列內指向同串列內特定節點的節點

```
NODE*findPreNode( NODE *list, intdata ) {
NODE *pN, *nextN;
if( isEmpty( list ) ) return (NODE*)NULL; //此串列是空串列，找尋失敗。
pN= list; //指向第一個資料節點的前一個節點(標頭節點)
do{
nextN= pN->link; //取得下一個節點的位址(第一次為第一個節點)
```

```
if( nextN->data == data ){ //下一個節點是否為指定節點
return pN; //找到，回傳現在這個節點位址
} else {
pN= pN->link ; //不是，移動到下一個節點
}
}while(pN->link != list) ; //非最後一個
return (NODE*)NULL ;} //沒找到指定節點
```

11.將新節點插入串列內指定節點之前

```
int insertBefor( NODE *list, NODE *newN, intdata ) {
NODE *preN;
preN= findPreNode( list, data ) ;
if( preN== NULL ) return 1 ; //插入失敗(檢查)
newN->link = preN->link ; //新節點接續
preN->link = newN; //前一個節點接現在節點
return 0 ;} //成功插入
```

12.將新節點插入串列內指定節點之後

```
int insertAfter( NODE *list, NODE *newN, intdata ) {
NODE *dataN;
dataN= findNode( list, data ) ;
if( dataN== NULL ) return 1 ; //插入失敗
newN->link = dataN->link ;
dataN->link = newN;
return 0 ;} //成功插入
```

13.刪除特定節點

```
NODE*deletNode( NODE *list, intdata ) { //串列拿出給呼叫者(並未刪除)
NODE *preN, *deletN;
if( isEmpty( list ) )return (NODE*)0 ; //串列是空串列，無節點可刪除，刪除失敗。
preN= findPreNode( list, data ); //找出要刪除節點的前一個節點
if( preN== NULL )return (NODE*)0; //找不到要刪除的節點，刪除失敗
deletN= preN->link ; //保留要刪除節點的位址
preN->link = deletN->link ; //將要刪除節點的下一節點位址設定給要刪除節點的前一個節點(將要刪除節點移出串列)
```

```
deletN->link = (NODE*)0;
return deletN ;} //回傳被刪除的節點
```

四、雙向鏈結串列實作與相關運算

1.雙向鏈結串列的節點實作{使用結構來定義其骨架}

```
typedef struct{
int data;
DLLNODE *pLLink;
DLLNODE *pRLink;
} DLLNODE;
//產生實體
DLLNODE *ddlisk= (DLLNODE*)malloc( sizeof(DLLNODE) ) ;
```

2.產生新節點的函式

```
DLLNODE* newDLLNode( void ) {
DLLNODE *pNewNode= (DLLNODE*)0;
pNewNode= (DLLNODE*)malloc( sizeof(DLLNODE) ) ;
return pNewNode;}
```

3.建立具有資料的新節點

```
DLLNODE* newDataNode( intdata ){
DLLNODE* tempN= newDLLNode( ) ;
if( tempN== (DLLNODE*)0 ) return tempN; //建立節點失敗
else {
tempN->data = data ;
tempN->RLink= (DLLNODE*)0;
tempN->LLink= (DLLNODE*)0;
return tempN ;} //建立資料節點成功
```

4.建立雙向鏈結串列的空串列

```
DLLNODE* newEmptyDLL( void ) {
DLLNODE*pNewDLList= (DLLNODE*)0; //NULL
pNewDLList= newDataNode(0); //建立標頭節點
return pNewDLList;}
//產生實體
DLLNODE*pDDList= newEmptyDLL();
if( pDDList== (DLLNODE*)0 ) return -1; //建立空串列失敗
```

5.判斷雙向鏈結串列是否為空串列

```
bool isEmpty( DLLNODE* pDLList){
```

```
if((pDLList->LLink==(DLLNODE*)0)&&
(pDLLisst->Rlink== (DLLNODE*)0)
)return true;
else return false ;}
```

6.於插入點之後插入新資料節點

```
int insertAfterNode( DLLNODE *pDLList,DLLNODE *insertPoint,
DLLNODE *newNode)
{newNode->RLink= insertPoint->RLink;
newNode->LLink= insertPoint->RLink->LLink;
insertPoint->RLink->LLink= newNode;
insertPoint->RLink= newNode;
return 0;} //Successful
```

7.於插入點之前插入新資料節點

```
int insertBeforeNode( DLLNODE *pDLList,
DLLNODE *insertPoint,
DLLNODE *newNode)
{newNode->RLink= insertPoint;
newNode->LLink= insertPoint->LLink;
insertPoint->LLink->RLink= newNode;
insertPoint->LLink= newNode;
return 0;} //Successful
```

8.插入新資料節點在第一個資料節點之前

```
int insertAfterHead( DLLNODE *pDLList, DLLNODE *
pNewNode){
if(insertAfterNode(pDLList,pDLList,pNewNode)==
(DLLNODE*)0 )
return -1; //Insert Failure
else
return 0;} //Successful
```

9.插入新資料節點在最後一個資料節點之後

```
int insertAfterRear( DLLNODE *pDLList, DLLNODE *
pNewNode){
DLLNODE *pRear= (DLLNODE*)0;
pRear= findLastNode( pDLList) ;
if( insertAfterNode( pDLList, pRear, pNewNode) ==
```

```
(DLLNODE*)0 )
return -1; //Insert Failure
else
return 0;} //Successful
```

10.找出最後一個資料節點

```
DLLNode* findLastNode( DLLNode* pDLList){
DLLNode* pTempN= (DLLNode*)0 ;
pTempN= pDLList;
while( pTempN->RLink!= (DLLNODE*)0 ){
pTempN= pTempN->RLink;
}
return pTempN;}
```

11.找出特定資料節點

```
DLLNODE* findData( DLLNODE *pDLList, constintdata ){
DLLNODE* pN= (DLLNODE*)0;
if( isEmpty( pDLList) )return -1; //尋找失敗
pN=pDLList->RLink;
do{
if( pN->data == data )return pN;
}{else pN= pN->RLink;
}while(pN!=(DLLNODE*)0);
return -1;} //尋找失敗
```

12.移除特定資料節點

```
int removeNode( DLLNODE * pDLList, DLLNODE * removedN){
if( isEmpty( pDLList) ) return -1 ; //移除失敗
if( removedN->LLink== (DLLNODE*)0 ) return -2; //removedN
非串列內資料節點
removedN->LLink->RLink= removedN->RLink;
if( removedN->RLink!= (DLLNODE*)0 ) //被刪節點非最後一個
removedN->RLink->LLink= removedN->LLink;
removedN->RLink= (DLLNODE*)0;
removedN->LLink= (DLLNODE*)0;
return 0;} //移除成功
```