

# 計算機程式設計

## C語言 Link List

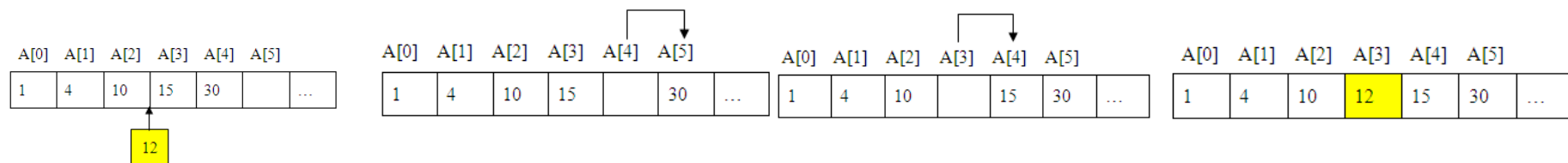
郭忠義

[jykuo@ntut.edu.tw](mailto:jykuo@ntut.edu.tw)

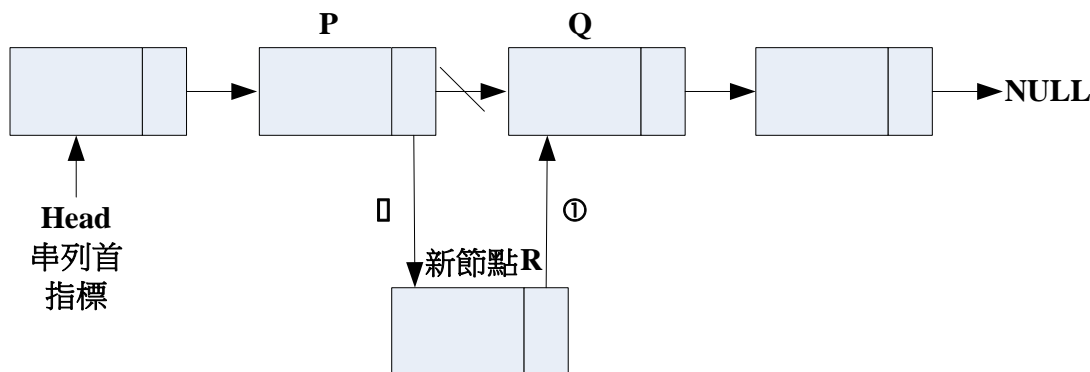
臺北科技大學資訊工程系

# 有序串列與無序串列

- 有序串列(ordered list)：如陣列(靜態資料結構)
  - 資料儲存在連續記憶空間，無法任意增/刪空間。
  - 陣列欲插入數字12到10與15之間，搬移過程。



- 無序串列(unordered list)-**鏈結串列(Link List)** 動態資料結構
  - 資料儲存在非連續性記憶空間，以指標彈性在串列增減元素。



# 陣列與鏈結串列

## □ 利用指標串接節點

靜態資料結構(如：陣列)	動態資料結構(如：鏈結串列)
固定需求空間下，較節省記憶體空間	固定需求空間下，較浪費記憶體空間，須多出一個指標
加入、刪除及合併須大量資料移動	記憶體配置較有彈性 加入、刪除及合併，只須改變指標即可
可以直接存取	不可以直接存取
可進行二分法搜尋	搜尋某元素可能較耗時 指標(point)斷裂時，資料會遺失(lost)

# 動態資料結構

```
struct node_s {  
    int data ;  
    struct node_s * next ;  
}  
struct node_s x1, x2;
```

- 每個節點儲存資料、指向下一個節點位置，最後節點指向 Null。

```
typedef struct node_s {    // struct + 結構名稱(自訂)
```

```
    int data ;
```

```
    struct node_s * next ;
```

```
} node_t;
```

```
typedef node_t * nodep_t
```

資料欄 指標欄



指向下一個節點

串列首



串列首



串列首



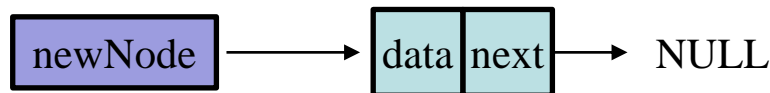
# 記憶體動態配置

```
#include <stdlib.h>
#include <stdio.h>
int main() {
    char *str; //str[80];
    str = (char *) malloc(80*sizeof(char));
    if (str == NULL) {
        printf("\1: unable to allocate memory for str.\n");
        exit(1);
    }
    printf("\1: Please input any sentence.\n");
    gets(str);
    printf("\2: The string that you input is.\n");
    puts(str);
}
```

# 動態資料結構-宣告、建節點

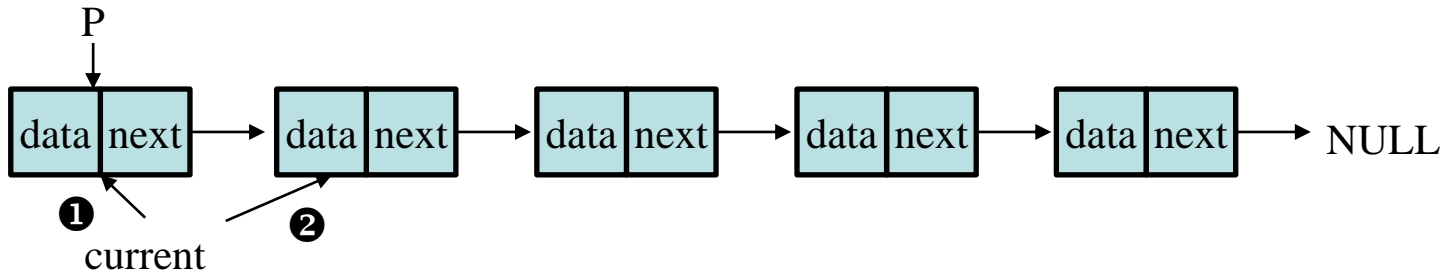
```
typedef struct node_s {                                //定義 node_t
    int data;                                           // struct node_s == node_t
    struct node_s *next;
} node_t;                                              // node_t * == nodep_t
typedef node_t * nodep_t;                             //定義 nodep_t

nodep_t create(int data) {                             //造出 newNode
    nodep_t newNode;                                  // newNode是指向 node_t 的指標變數
    newNode=(nodep_t)malloc(sizeof(node_t));          //配置記憶體空間
    newNode->data = data;
    newNode->next=NULL;
    return newNode;                                    //將新 node指標回傳
}
```



# 串列尋訪列印

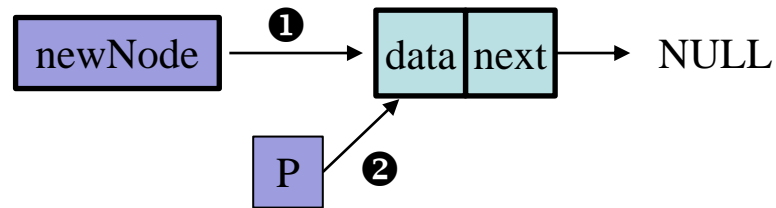
- 設定current指標指向串列首，再一一的往下一個node移動。



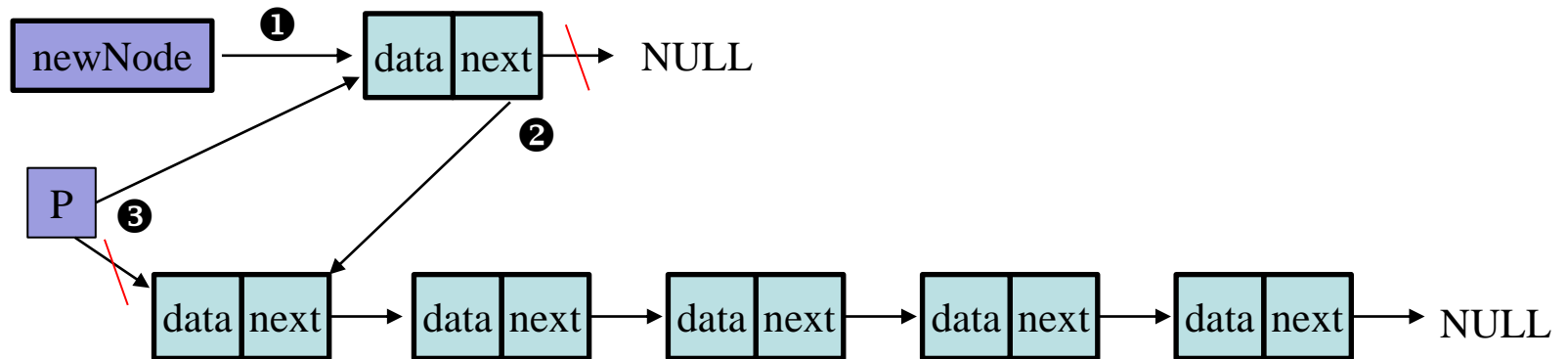
```
void printList(nodep_t p) {                                     //尋訪串列並列印
    nodep_t current = p;
    printf("\n The list is: ");
    while(current!=NULL) {
        printf("%d, ",current->data);
        current = current->next;
    }
}
```

# 串列從前面加入節點

- 空串列，把根指標p指向新造出的節點newNode。



- 非空串列，把新節點指標指向串列首，再把串列首指到新節點上。





# 串列從前面加入節點

- ❑ 空串列，把根指標p指向新的節點。
- ❑ 非空串列，把新節點指標指向串列首，再把串列首移到新節點上。

```
void insertFromFront(nodep_t *p, int data) {  
    nodep_t newNode = create(data);  
    if ((*p)==NULL) {  
        (*p) = newNode;  
    }  
    else {  
        newNode->next = (*p);  
        (*p) = newNode;  
    }  
}
```

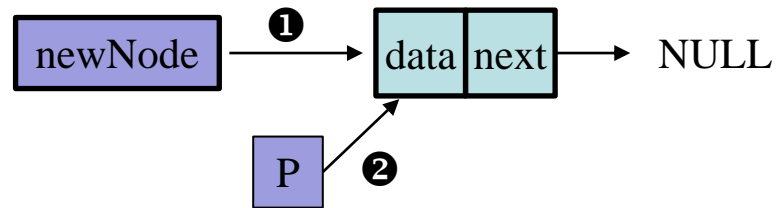
```
nodep_t p;  
insertFromFront(&p, 15);
```

```
nodep_t insertFromFront(nodep_t p, int data) {  
    nodep_t newNode = create(data);  
    if (p==NULL) {  
        p = newNode;  
    }  
    else {  
        newNode->next = p;  
        p = newNode;  
    }  
    return p;  
}
```

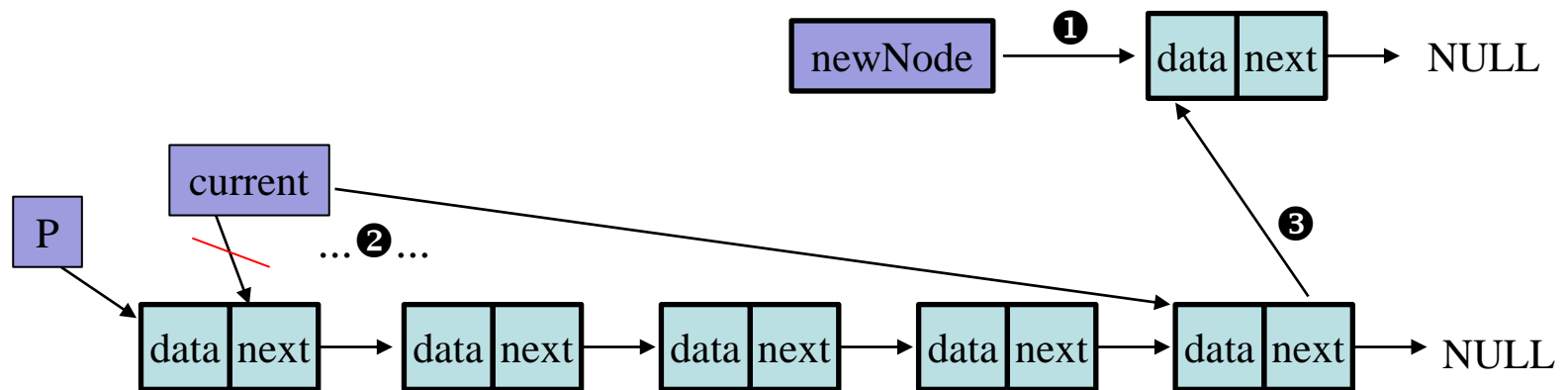
```
nodep_t p;  
p = insertFromFront(p, 15);
```

# 串列從後面加入節點

- 空串列，把根指標p指向新造出的節點newNode。



- 非空串列，設定current指標變數從頭開始尋訪到尾，把current指標的next指向新節點上。



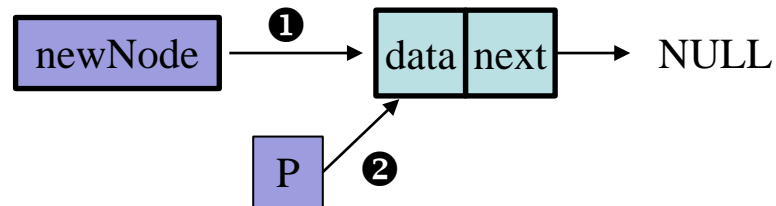
# 串列從後面加入節點

- ❑ 空串列，把根指標p指向新造出的節點newNode。
- ❑ 非空串列，設定current指標變數從頭開始尋訪到尾，把current指標的next指向新節點上。

```
void insertFromBack(nodep_t *p, int data) {  
    nodep_t current;  
    nodep_t newNode = create(data);  
    if ((*p)==NULL) {  
        (*p) = newNode;  
    }  
    else {  
        current = (*p);  
        while (current->next!=NULL)  
            current = current->next;  
        current->next = newNode;  
    }  
}
```

# 串列加入節點排序(疊代)

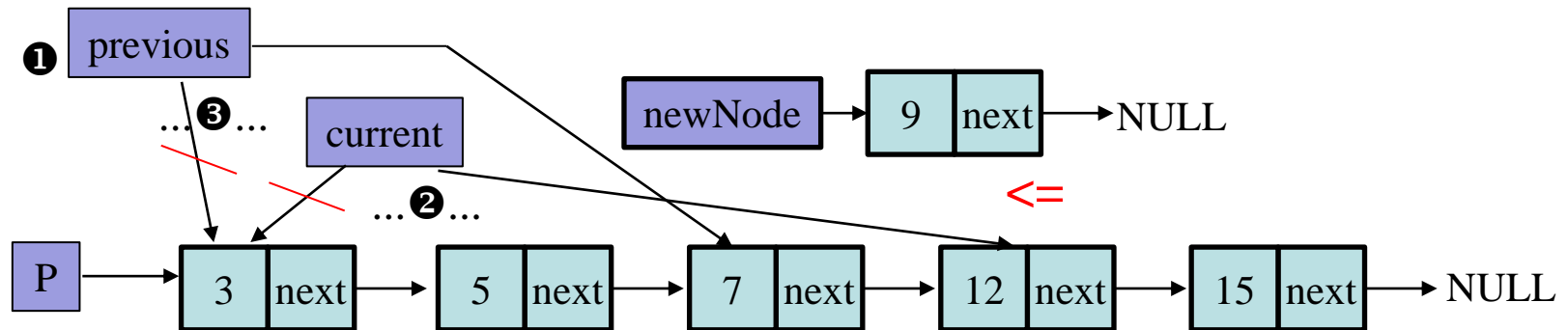
- ❑ 空串列，把根指標p指向新造出的節點newNode。



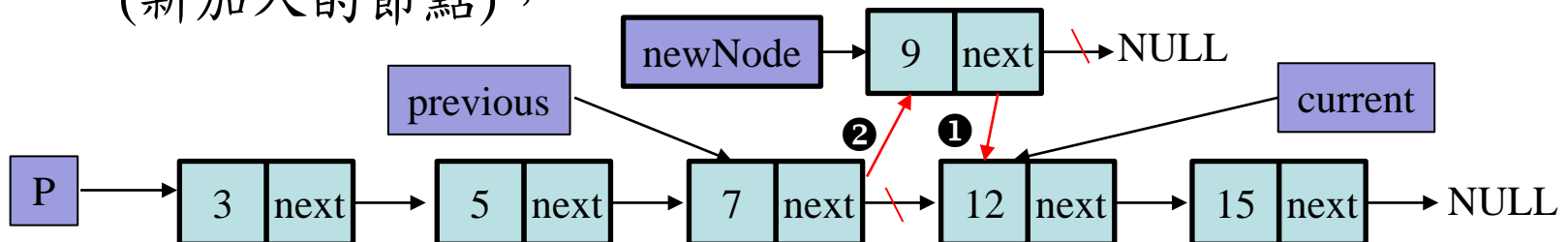
```
nodep_t newNode, current, previous;
newNode = create(data);
if (p == NULL) { return newNode; }
else if ((p->data) >= data) {
    newNode->next = p;
    return newNode;
}
```

# 串列加入節點排序(疊代)

- ❑ 非空串列，設定current指標從頭開始尋訪，直到newNode的data(新加入 9 )小於current指標指向node的data (12)。
  - 設定previous指標，在current前面



- 將newNode指向節點 9 的next，指向current指標所指節點 12，
- 將previous指向的節點(7)的next，指向newNode指標所指位址 (新加入的節點)，

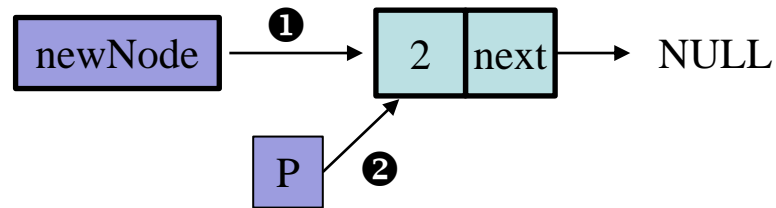


# 串列加入節點排序(疊代)

```
nodep_t insertInOrderI(nodep_t p, int data) {
    nodep_t newNode, current, previous;
    newNode = create(data);
    if (p == NULL) { return newNode; }
    else if ((p->data) >= data) {
        newNode->next = p;
        return newNode;
    }
    else {
        current = previous = p;
        while (current != NULL) {
            if ((current->data) < data) {
                previous = current;
                current = current->next;
            }
            else break;
        }
        previous->next = newNode;
        newNode->next = current;
        return p;
    }
    return (p);
}
```

# 串列加入節點排序(遞迴)

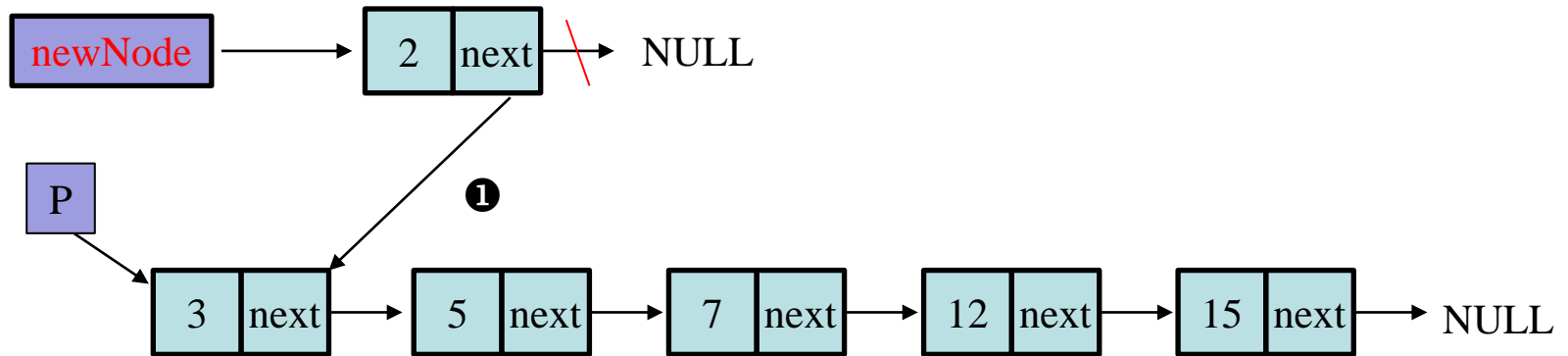
- 空串列，把根指標p指向新造出的節點newNode。



```
nodep_t newNode, current, previous;  
if (p == NULL) { return create(data); }
```

# 串列加入節點排序(遞迴)

- ❑ 非空字串，新節點資料小於串列頭的資料。
  - 從前面加入新節點

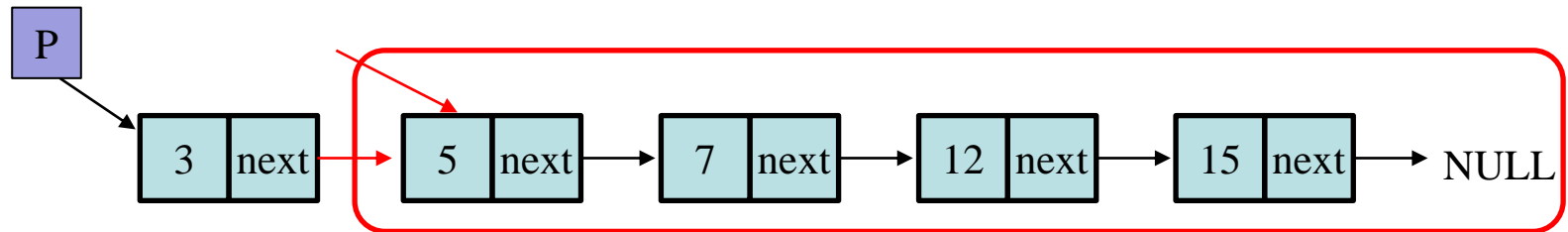


```
else if ((p->data) >= data) {  
    newNode = create(data);  
    newNode->next = p;  
    return newNode;  
}
```



# 串列加入節點排序(遞迴)

- ❑ 非空字串，新節點資料不小於串列頭的資料。
  - 將問題變小－指標往下 next 移動，
  - 呼叫本身函式處理，完成回傳指標接到 p->next



```
else {  
    p->next = insertInOrderR(p->next, data);  
    return p;  
}
```

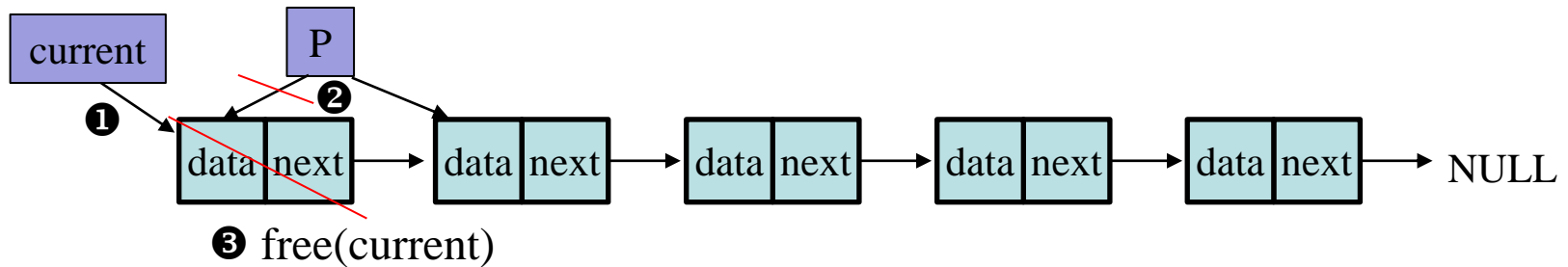
# 串列加入節點排序(遞迴)

- ❑ 加入新節點，把新節點指標指向串列首，再把串列首移到新節點上。

```
nodep_t insertInOrderR(nodep_t p, int data) {  
    nodep_t newp, current, prev;  
    if (p == NULL) { return create(data); }  
    else if ((p->data) >= data) {  
        newp = create(data);  
        newp->next = p;  
        return newp;  
    }  
    else {  
        p->next = insertInOrderR(p->next, data);  
        return p;  
    }  
}
```

# 串列從前面刪除節點

- 設定current指標指向 p 所指位址，串列頭，把p指標指向第二個節點，釋放current指標指向的空間，原串列頭。



# 串列從前面刪除節點

- ❑ 設定current指標指向 p 所指位址，串列頭，把p指標指向第二個節點，釋放current指標指向的空間，原串列頭。

```
int delFromFront(nodep_t *p) {  
    int data = -1;  
    nodep_t current;  
    if ((*p) != NULL) {  
        current = (*p);  
        (*p) = current->next;  
        data = current->data;  
        free(current);  
        return data;  
    }  
    return data;  
}
```

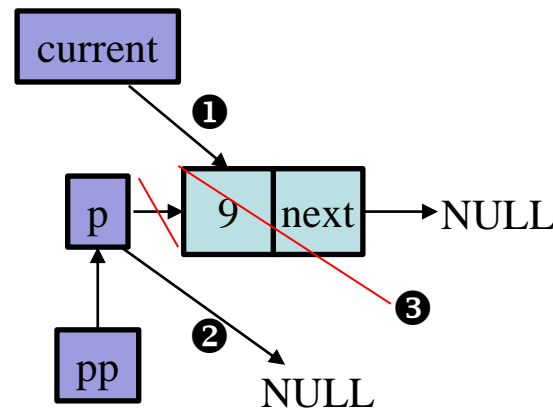
# 串列從後面刪除節點

- ❑ 空串列，回傳-1。

```
nodep_t previous, current;           // (1)
int data = -1;
previous = current = (*pp);
if (current==NULL) return data;
```

- ❑ 串列只有一個，刪除後free記憶體空間，p (\*pp) 指向NULL。

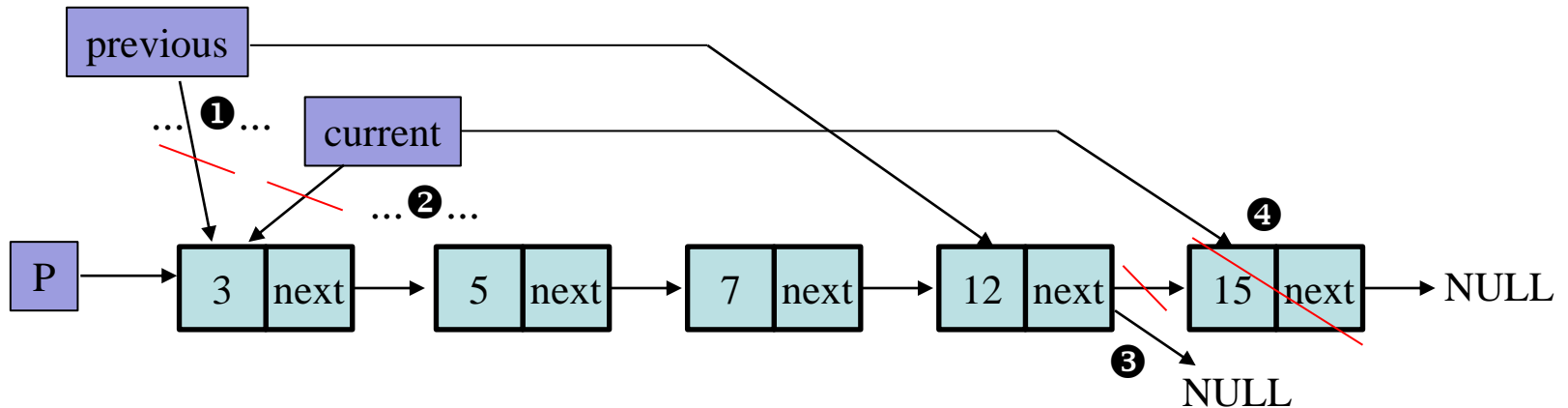
```
else if (current->next==NULL) {
    (*pp) = NULL;           // (2)
    data = current->data;
    free(current);          // (3)
    return data;
}
```



# 串列從後面刪除節點

❑ 串列至少二個，

○ 刪除後free記憶體空間，p (\*pp) 指向NULL。



```
while (current->next!=NULL) {  
    previous = current;      //(1)  
    current=current->next;   //(2)  
}  
data = current->data;  
previous->next = NULL;      //(3)  
free(current);              //(4)
```

# 串列從後面刪除節點

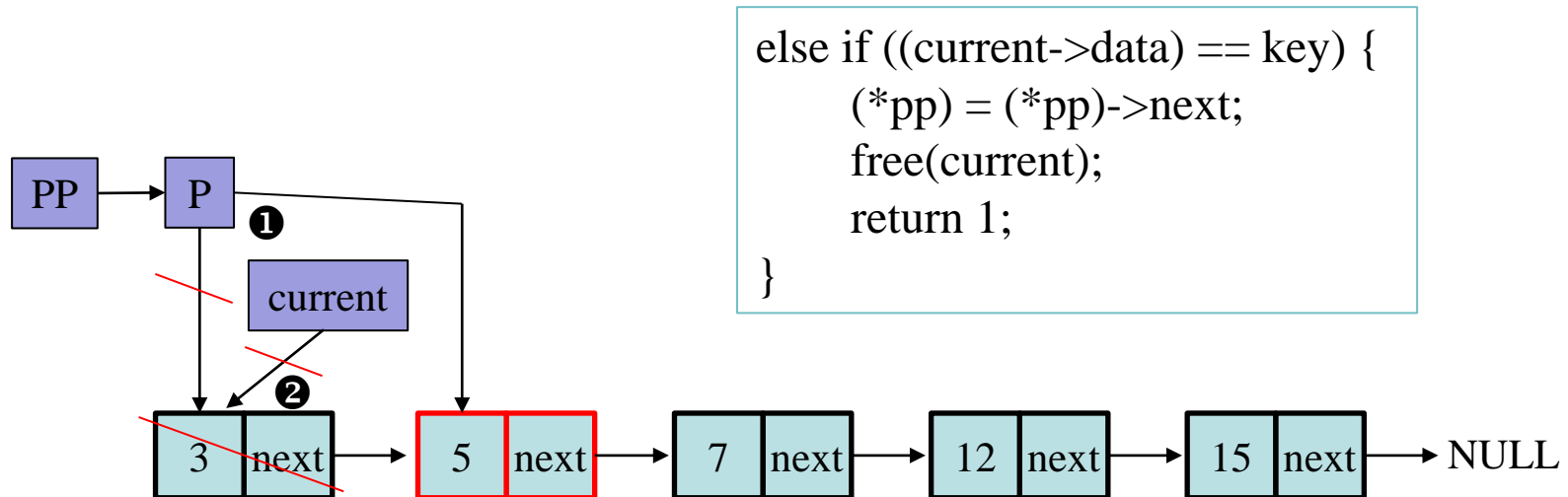
```
int delFromBack(nodep_t *pp) {
    nodep_t previous, current;
    int data = -1;
    previous = current = (*pp);
    if (current==NULL) return data;
    else if (current->next==NULL) {
        (*pp) = NULL;
        data = current->data;
        free(current);
        return data;
    }
    while (current->next!=NULL) {
        previous = current;
        current=current->next;
    }
    data = current->data;
    previous->next = NULL;
    free(current);
    return data;
}
```

# 串列根據key刪除節點

- 空串列，回傳-1。

```
nodep_t previous, current;           // (1)
int data = -1;
previous = current = (*pp);
if (current==NULL) return data;
```

- 第一個找到，刪除後free記憶體空間，p(\*pp)指向下一個。

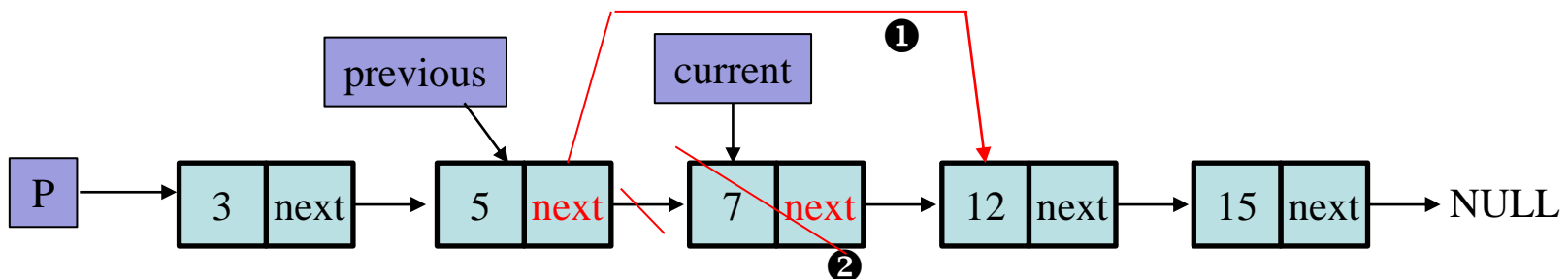




# 串列根據key刪除節點

- 一直尋訪直到找到(假設 7)，刪除後free記憶體空間，p (\*pp) 指向下一個。

```
while (current!=NULL) {  
    if ((current->data) == key) {  
        // 找到  
        previous->next = current->next; // (1)  
        free(current); // (2)  
        return 1;  
    }  
    else {  
        // 沒找到  
        // 一直往後找  
        previous = current;  
        current = current->next;  
    }  
}
```



# 串列根據key刪除節點

```
int deleteByKey(nodep_t *pp, int key) {
    nodep_t previous, current;
    previous = current = (*pp);
    if (current==NULL) return -1;
    else if ((current->data) == key) {
        (*pp) = (*pp)->next;
        free(current);
        return 1;
    }
    while (current!=NULL) {
        if ((current->data) == key) {
            previous->next = current->next;
            free(current);
            return 1;
        }
        else {
            previous = current;
            current = current->next;
        }
    }
    return -1;
}
```

# 抽象資料型別

- ❑ 抽象資料型別(Abstract Data Type, ADT)特性與優點
  - 介面操作規範 (The specification of the operation)
    - 函式名稱 (Function Name)
    - 參數型別 (The type of its argument)
    - 結果型別 (The type of its result)
    - 函式功能的描述 (不必說明內部表示法或實作細節)
  - 介面是"封裝隔離"機制，
    - 客戶端程式呼叫存取，僅需操作其所提供的介面函式。
    - 不用也不需要知道內部如何實作。
    - 系統具高維護性與彈性，不論擴充或重構，客戶端程式僅受最小幅度的影響。

# 抽象資料型別

- ❑ 抽象資料型別實作獨立(Implementation-Independent)。
- ❑ 堆疊STACK實作方法
  - 陣列，會受到大小須事先宣告的限制。
  - 鏈結(nodep\_ted List)，以動態記憶體配置方式新增每個元素。
- ❑ 靜態記憶體配置，編譯階段要求配置所需記憶體空間。
- ❑ 動態記憶體配置，執行階段要求配置所需記憶體空間。
  - 指標變數 = (資料型別\*) malloc(sizeof(資料型別));
  - int \*pt;
  - pt = (int \*) malloc(sizeof(int));
- ❑ 釋放記憶體運算子
  - free(指標變數); free(pt);

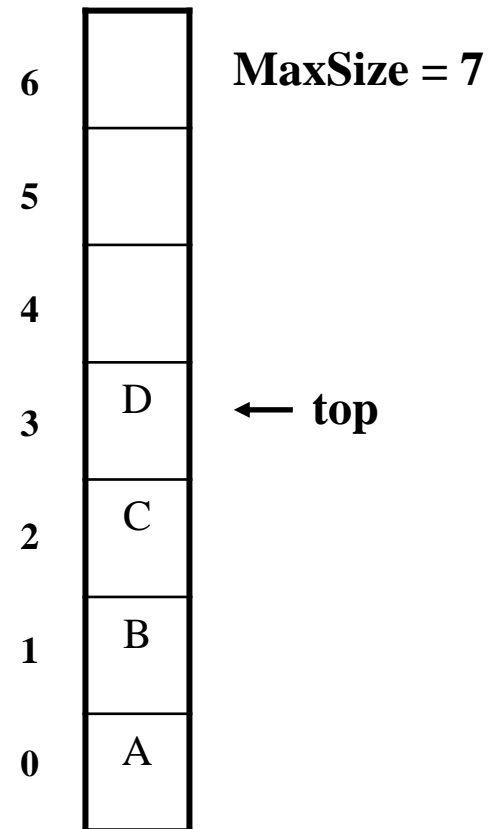
# 堆疊(Stack)

## ❑ 後進先出(Last In First Out, LIFO)

- 將資料依序從下面(bottom)儲存，從上面(top)將資料取出(Stack)。

## ❑ 動作：

- create：建立一個空堆疊。
- push：將資料存入堆疊。
- pop：將資料從堆疊中取出。
- isEmpty：判斷堆疊是否為空堆疊。
- isFull：判斷堆疊是否已滿。



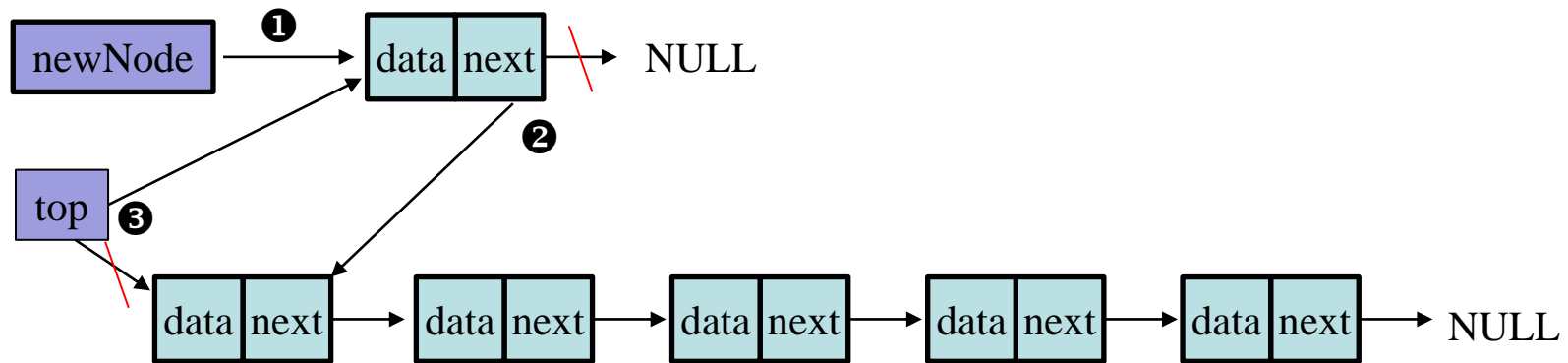
# 堆疊 – 使用陣列

```
#include <stdio.h>
#define MaxSize 7
int isEmpty(int top) {
    if (top==-1) { return 1;}
    else { return 0; }
}
int isFull(int top){
    if (top==MaxSize-1) { return 1; }
    else { return 0; }
}
// return 1 is success
int push(int stack[MaxSize], int *p_top, int n){
    if (!isFull(*p_top)) {
        stack[++(*p_top)]=n;
        return 1;
    }
    else { return 0;}
}
```

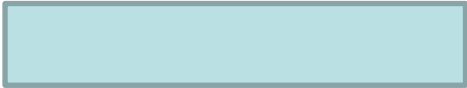
```
// return 1 is success
int pop(int stack[MaxSize], int *p_top, int *data) {
    int k;
    if (!isEmpty(*p_top)) {
        (*data) = stack[(*p_top)--];
        return 1;
    }
    else { return 0; }
}
int main() {
    int stack[MaxSize];
    int top = -1, flag, data;
    flag = push(stack, &top, 5);
    flag = push(stack, &top, 8);
    flag = pop(stack, &top, &data);
    printf("success=%d, data=%d\n", flag, data);
    flag = pop(stack, &top, &data);
    printf("success=%d, data=%d\n", flag, data);
    flag = pop(stack, &top, &data);
    printf("success=%d, data=%d\n", flag, data);
    return 0;
}
```

# 堆疊 – 使用鏈結串列push資料

- 產生新資料項的新節點newNode，加入資料到鏈結堆疊中



# Exercise:堆疊使用鏈結串列加入資料

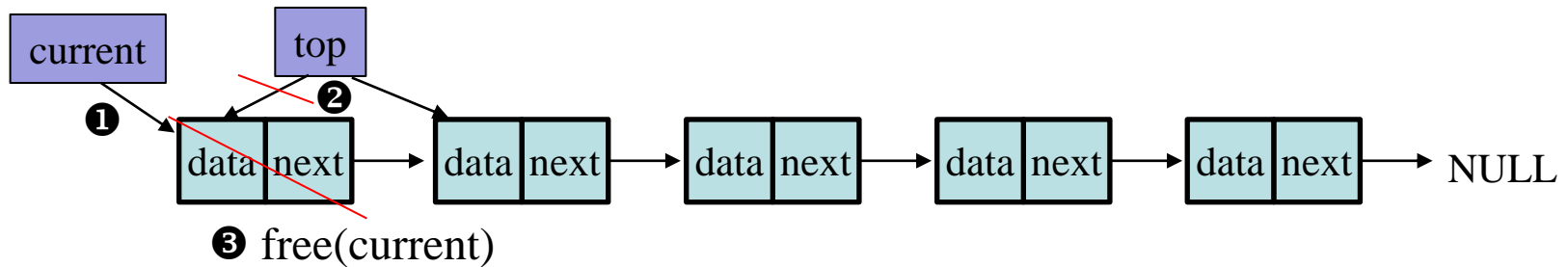
```
#include <stdio.h>
#include <stdlib.h>
//C99 _Bool type macro - bool/true/false
#include <stdbool.h>
typedef struct node_s {
    int data;
    struct node_s * next;
} node_t;
typedef node_t * nodep_t;
nodep_t create(int data) {
    nodep_t newNode;
    newNode=(nodep_t)malloc(sizeof(node_t));
    newNode->data = data;
    newNode->next=NULL;
    return newNode;
}
void printData(bool flag, int data) {
    if (flag) printf("pop success=%d, data=%d\n", flag, data);
    else printf("pop fail");
}
bool isEmpty(nodep_t top) { return (top==NULL); }
bool push(nodep_t *p_top, int x) {
    nodep_t newNode = create(x);
    if (newNode == NULL) return false;
    
    return true;
}
```

```
void printStack(nodep_t top) {
    nodep_t current = top;
    printf("The stack is: ");
    while(current!=NULL) {
        printf("%d, ",current->data);
        current = current->next;
    }
    printf("\n");
}
int main() {
    int data;
    bool flag;
    nodep_t top = NULL;
    printf("push sucess=%d\n", push(&top, 5));
    printf("push sucess=%d\n", push(&top, 8));
    printStack(top); // 8 5
    flag = pop(&top, &data);
    printData(flag, data); // 5
    printStack(top); // 8
    flag = pop(&top, &data);
    printData(flag, data); // 8
    flag = pop(&top, &data);
    printData(flag, data); // fail
    return 0;
}
```



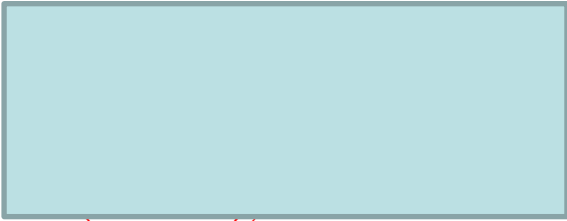
# 堆疊 – 使用鏈結串列刪除資料

- 刪除頂端資料時，須利用一個指標指向頂端節點，以便在頂端指標top改為指向第二個節點後，還能參照到原來頂端節點，且將其空間歸還系統。



# Exercise

## ❑ 堆疊 – 使用鏈結串列刪除資料

```
bool pop(nodep_t *p_top, int *data) {  
    if (isEmpty(*p_top)) return false;  
      
    return true;  
}
```

# 佇列的操作

## □ 佇列(Queue)，First In First Out, FIFO

- 將資料依序從尾端(tail) 儲存，從開頭(head) 將資料取出。

## □ 動作

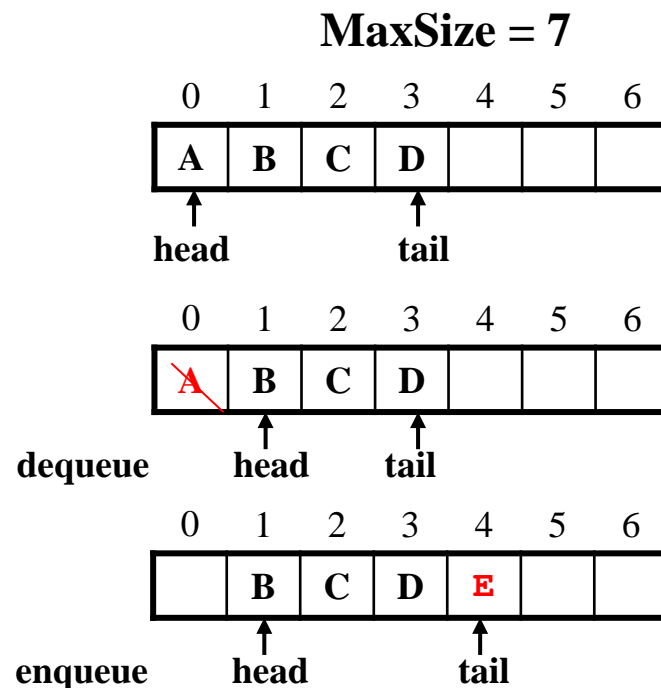
create : 建立一個空佇列。

enqueue : 將資料存入佇列中。

dequeue : 將資料從佇列中取出。

isEmpty : 判斷佇列是否為空佇列。

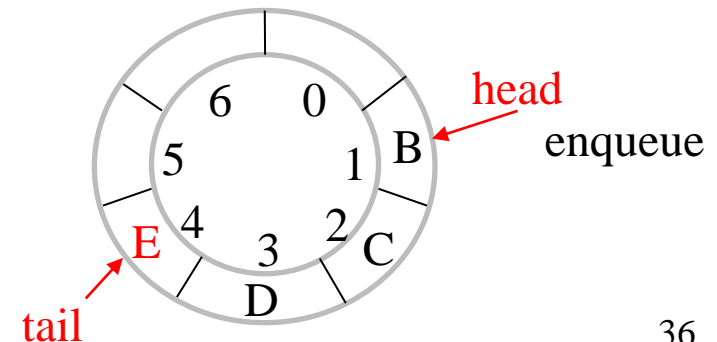
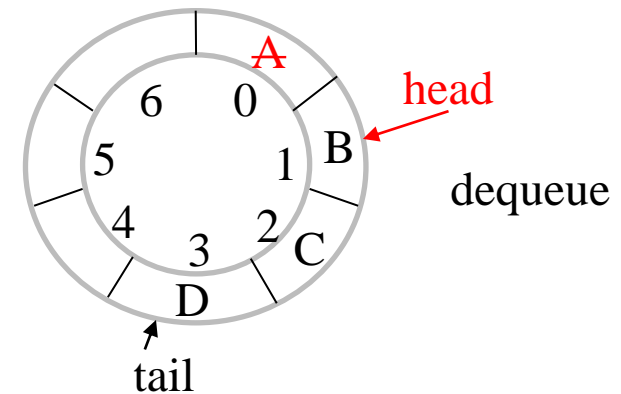
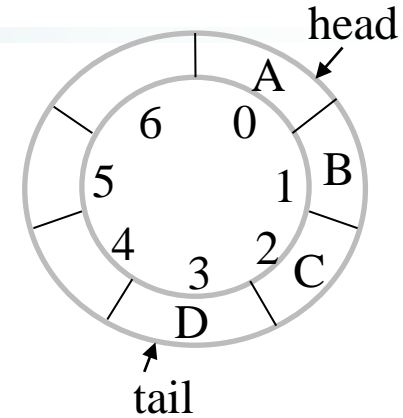
front : 傳回佇列前端 head 的值。



# 環狀佇列 – 使用陣列

## ❑ 重複利用空間

```
#include <stdio.h>
#define SIZE 3
// #define TRUE 1
// #define FALSE 0
typedef enum {FALSE, TRUE} bool;
bool isEmpty(int front, int back) {
    return (front==back);
}
bool isFull(int front, int back) {
    return (((back+1)%SIZE)==front);
}
bool enqueue(int data[], int index[], int key) {
    // front = index[0];    back = index[1];
    if (isFull(index[0], index[1])) return FALSE;
    index[1] = (index[1]+1)%SIZE;
    data[index[1]] = key;
    return TRUE;
}
```



# 環狀佇列 – 使用陣列

## ❑ 重複利用空間

```
int dequeue(int data[], int index[]) {  
    //front = index[0]; back = index[1]; dequeue data = index[2]  
    if (isEmpty(index[0], index[1])) return FALSE;  
    index[0] = (index[0]+1)%SIZE;  
    index[2] = data[index[0]];   
    return TRUE;  
}  
  
int main() {  
    int k=0, i=0;  
    //front = index[0]; back = index[1]; dequeue data = index[2]  
    int index[3]={0, 0, 0};  
    int data[SIZE];  
    bool result;  
    for (i=0; i<5; i++) {  
        result=enqueue(data, index, k++);  
        if (!result) printf("Queue Full!\n");  
        else printf("enqueue %d\n", k-1);  
    }  
}
```

# 環狀佇列 – 使用陣列

## ❑ 重複利用空間

```
result = dequeue(data, index);  
if (!result) printf("Queue Empty!\n");  
else printf("%d\n", index[2]);
```

```
result = dequeue(data, index);  
if (!result) printf("Queue Empty!\n");  
else printf("%d\n", index[2]);
```

```
result = dequeue(data, index);  
if (!result) printf("Queue Empty!\n");  
else printf("%d\n", index[2]);
```

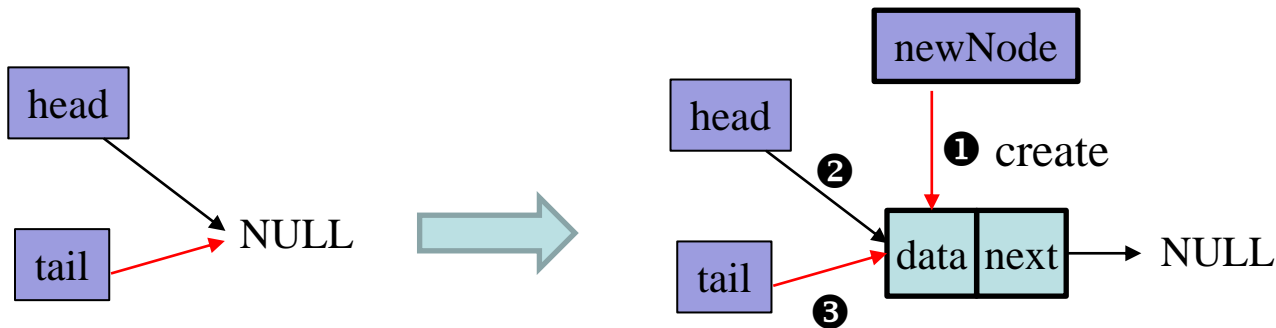
```
return 0;
```

```
}
```

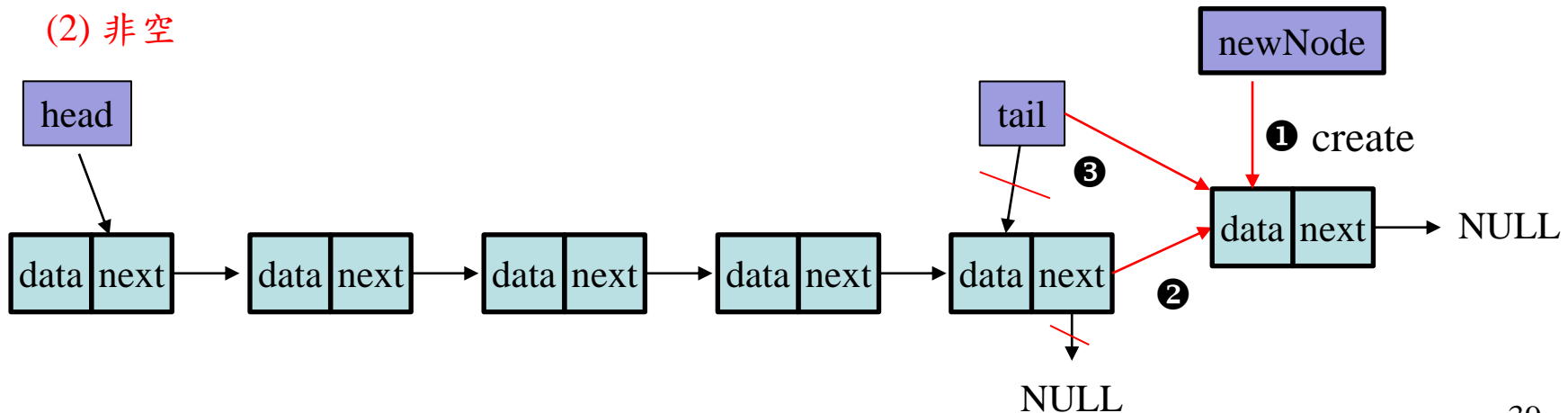
# 鏈結佇列的資料加入

- 產生新資料項新節點newNode，加到鏈結佇列的尾端

(1) 空的情況



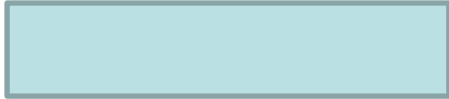
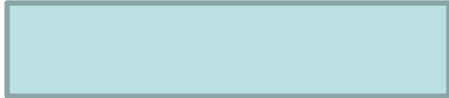
(2) 非空



# Exercise

## ❑ 鏈結佇列的資料加入

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
typedef struct node_s {
    int data;
    struct node_s * next;
} node_t;
typedef node_t * nodep_t;
nodep_t create(int data) {
    nodep_t newNode;
    newNode=(nodep_t)malloc(sizeof(node_t));
    newNode->data = data;
    newNode->next=NULL;
    return newNode;
}
void printData(bool flag, int data) {
    if (flag)
        printf("dequeue success=%d, data=%d\n", flag, data);
    else printf("pop fail\n");
}
```

```
bool enqueue(nodep_t *p_head, nodep_t *p_tail, int x) {
    nodep_t newNode = create(x);
    if (newNode == NULL) return false;
    if ((*p_head)==NULL) {

    }
    else {

    }
    return true;
}
```



# Exercise

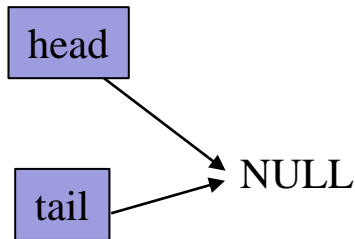
## ❑ 鏈結佇列的資料加入

```
int main() {  
    int flag, x=0;  
    nodep_t tail = NULL, head=NULL;  
    enqueue(&head, &tail, 5);  
    enqueue(&head, &tail, 8);  
    enqueue(&head, &tail, 1);  
    flag = dequeue(&head, &tail, &x);  
    printData(flag, x);  
    flag = dequeue(&head, &tail, &x);  
    printData(flag, x);  
    flag = dequeue(&head, &tail, &x);  
    printData(flag, x);  
    flag = dequeue(&head, &tail, &x);  
    printData(flag, x);  
    enqueue(&head, &tail, 9);  
    flag = dequeue(&head, &tail, &x);  
    printData(flag, x);  
    return 0;  
}
```

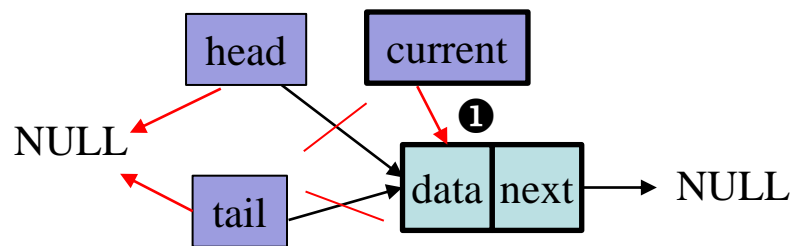
# 鏈結佇列的資料刪除

- 判斷鏈結堆疊「是否為空」，若不為空，則改變前端指標head至下一個節點，歸還原來前端節點空間給系統

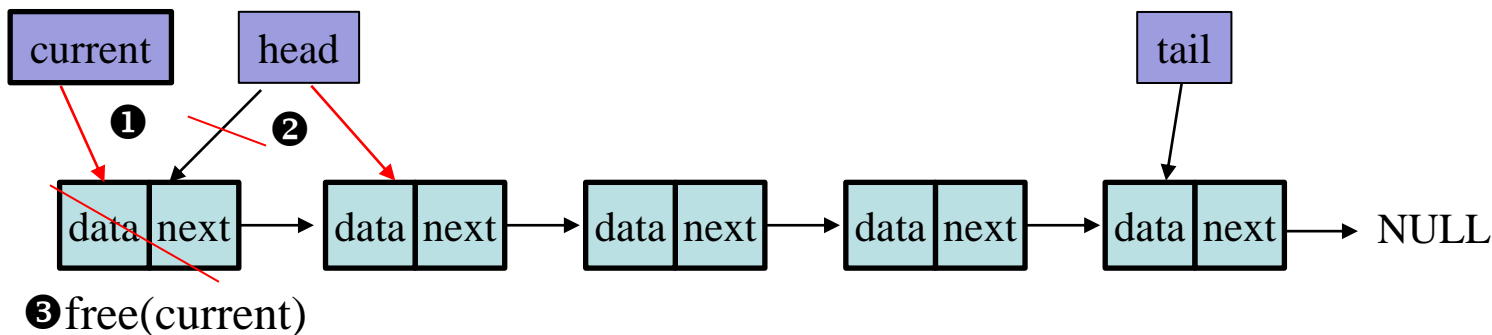
(1) 空的情況



(2) 只剩一個

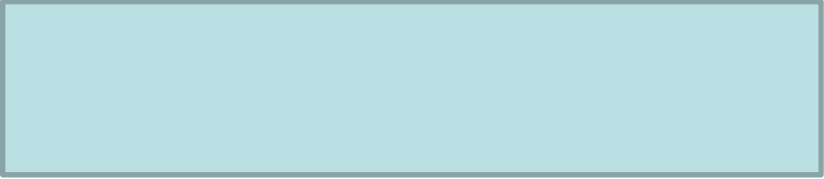


(3) 超過一個



# Exercise

## ❑ 鏈結佇列的資料刪除

```
bool dequeue(nodep_t *p_head, nodep_t *p_tail, int *data)
{
    if ((*p_head)==NULL) return false;
    (*data) = (*p_head)->data;
    
    return true;
}
```

# 反向列印串列

```
int main() {
    nodep_t ptr, head, tail;
    int num,i;
    tail = (nodep_t) malloc(sizeof(node_t));
    tail->next = NULL;
    ptr = tail;
    printf("\1: Please input 5 different data\n");
    for ( i = 0; i <= 4; i++ ) {
        scanf("%d",&num);
        ptr->data = num;
        head = (nodep_t) malloc(sizeof(node_t));
        head->next = ptr;
        ptr = head;
    }
    ptr = ptr->next; /* because final head does not have data */
    printf("\2: Reverse print the list\n");
    while ( ptr != NULL ) {
        printf("\2: The value is ==> %d\n",ptr->data);
        ptr = ptr->next;
    }
}
```

# Homework I

- 使用 Link List，輸入兩個多項式，輸出相加、相減、相乘的結果。例如：
  - $2x^4 + 3x^3 + x - 1$
  - $x^5 - x^3 + 4x^2 - 3x + 2$
  - 結果：
    - $2x^9 + 3x^8 - 2x^7 + 6x^6 + 5x^5 - 6x^4 + 11x^3 - 7x^2 + 5x - 2$
  - 輸入說明
    - 輸入兩筆資料，分別代表兩個多項數。
    - 每一筆輸入  $n$  個整數，第一個代表  $n-1$  次方的係數，第  $n$  個代表 0 次方係數。

# Homework I

## ○ 輸出說明

- 兩個多項式相乘後從最高次方到0次方的係數，相乘結果為0的係數也需印出。
- Sample Input
  - 2 3 0 1 -1
  - 1 0 -1 4 -3 2
- Sample Output
  - 2 3 -2 6 5 -6 11 -7 5 -2

# Homework II

## □ 使用 Link List 實作stack

- 在一端進行後進先出（LIFO, Last In First Out）的原理運作。
- 兩種基本操作：push 和 pop
  - push：將數據放入堆疊的頂端（串列形式），堆疊頂端top指標加一。
  - pop：將頂端數據資料輸出（回傳），堆疊頂端top指標減一。
- 每一次push和pop的一筆資料都包含姓名、年齡、生日(年、月、日)
- 輸入說明：
  - 1 代表 push，再依序輸入姓名、年齡、生日(年、月、日)，參數之間以空白相隔
  - 2 代表 pop，再輸入一個數字，進行不同的操作，操作數字如下：

# Homework II

- 1:印出該次pop的資料中的姓名
- 2:印出該次pop的資料中的年齡
- 3:印出該次pop的資料中的生日(年、月、日之間以底線連結)
- 若stack中為空則印出 The Stack is empty\n
- 3 結束程式。

## Sample Input

```
1 "Marry Hu" 19 1989 7 16
1 "Tom Chen" 22 1996 10 19
2 1
1 "Billy Wu" 15 2005 3 18
2 3
2 2
1 "Lucas Su" 24 1993 5 21
2 3
2 1
3
```

## Sample Output

```
Tom Chen
2005_3_18
19
1993_5_21
The Stack is empty
```



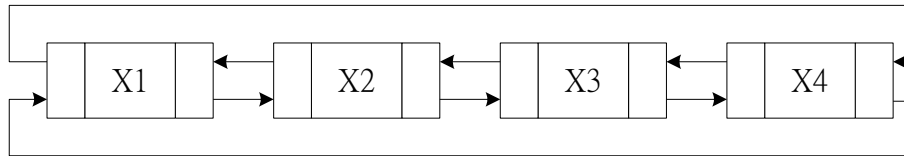
# Homework III

- 使用 Link List 實作queue

- 在一端進行先進先出（FIFO, First In First Out）的原理運作。

# 雙向鏈結串列(Double nodep\_ted List)

- ❑ 資料欄，儲存節點資料，
- ❑ 兩個指標，一個指向前一個節點，另一個指向下一個節點。
- ❑ 單向鏈結串列，在搜尋串列時，只能沿一個方向搜尋，無法往回搜尋。雙向鏈結串列可克服此問題。



```
typedef struct dnode_s {  
    int data;  
    struct dnode_s * front;  
    struct dnode_s * back;  
} node_t;
```

```
typedef node_t * nodep_t;
```

# 雙向鏈結串列

```
#include <stdlib.h>
#include <stdio.h>
typedef struct  dnode_s  {
    int  data;
    struct dnode_s * front;
    struct dnode_s * back;
} node_t;
typedef node_t * nodep_t;

void reversePrint(nodep_t ptr) {
    while (ptr) {
        printf("=>%d\n",ptr->data);
        ptr = ptr->back;
    }
}
```

# 雙向鏈結串列

```
void insert(nodep_t* tailp, nodep_t* headp, int num) {
    nodep_t ptr;
    ptr = (nodep_t) malloc(sizeof(node_t));
    ptr->data = num;
    ptr->front = ptr->back=NULL;
    if ((*headp)==NULL) {
        (*tailp) = (*headp) = ptr;
        return;
    }
    (*headp)->front = ptr;
    ptr->back=(*headp);
    (*headp)=(*headp)->front;
}

void print(nodep_t ptr) {
    while (ptr) {
        printf("=> %d\n",ptr->data);
        ptr = ptr->front;
    }
}
```

# 雙向鏈結串列

```
int main() {
    nodep_t ptr=NULL, tail=NULL, head=NULL;
    int num;
    char c;
    while (1) {
        scanf("%c",&c);
        if (c=='e') break;
        else if (c=='i') {
            scanf("%d",&num);
            insert(&tail, &head, num);
        }
        else if (c=='p') { print(tail); }
        else if (c=='r') { reversePrint(head); }
    }
    return 0;
}
```