

週一作業討論



HW1

使用二維陣列完成以下程式，使輸出結果如右



```
Struct symmetricArray{  
    int** data;  
  
    int n;  
    建構函式  
    set函式  
    plus函式  
    show函式 }
```

```
int main(){  
    //宣告3個對稱陣列  
    //用for迴圈設定元素值  
    //陣列C=A+B  
}
```

```
0  
1 2  
3 4 5  
//A陣列  
0  
2 4  
6 8 10  
//B陣列  
0  
3 6  
9 12 15  
//C陣列
```

週一作業討論

```
#include <iostream>
using namespace std;

struct symmetricArray {
    int** data;
    int n;

    // 1. 建構函式 (Constructor)
    // 初始化陣列大小，並配置記憶體
    symmetricArray(int size) {
        n = size;
        data = new int* [n]; // 配置列 (Rows)
        for (int i = 0; i < n; i++) {
            // 因為是對稱陣列(顯示為下三角)，第 i 列只需要 i+1 個空間
            data[i] = new int[i + 1];
        }
    }

    // 2. set函式
    // 設定特定位置的數值
    void set(int row, int col, int value) {
        if (row >= 0 && row < n && col >= 0 && col <= row) {
            data[row][col] = value;
        }
    }
}
```

週一作業討論

```
// 3. plus函式
// 將當前陣列與另一個陣列 B 相加，並回傳新的陣列 C
symmetricArray plus(symmetricArray& other) {
    symmetricArray temp(n); // 建立一個新的陣列用來存放結果
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            // 對應位置相加
            temp.data[i][j] = this->data[i][j] + other.data[i][j];
        }
    }
    return temp;
}

symmetricArray operator+(symmetricArray& other) {
    symmetricArray result(n);

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j <= i; ++j) {
            result.data[i][j] = data[i][j] + other.data[i][j];
        }
    }

    return result;
}
```

週一作業討論

```
// 4. show函式
// 顯示陣列內容
void show() {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            cout << data[i][j] << " ";
        }
        cout << endl; // 換行
    }
};

int main() {
    int size = 3;

    // 宣告3個對稱陣列 (A 和 B 需要初始化, C 用來接結果)
    symmetricArray A(size);
    symmetricArray B(size);

    // 用for迴圈設定元素值
    // 觀察圖片規律: A是 0, 1 2, 3 4 5 (連續數字)
    // B是 0, 2 4, 6 8 10 (A的兩倍)
    int counter = 0;
    for (int i = 0; i < size; i++) {
        for (int j = 0; j <= i; j++) {
            A.set(i, j, counter);
            B.set(i, j, counter * 2);
            counter++;
        }
    }
}
```

```
// 陣列 C = A + B (使用 plus 函式)
symmetricArray C = A.plus(B);
symmetricArray D = A + B;

// 輸出結果
A.show();
cout << "//A陣列" << endl;

B.show();
cout << "//B陣列" << endl;

C.show();
cout << "//C陣列" << endl;

D.show();
cout << "//D陣列" << endl;

return 0;
}
```

Linked List

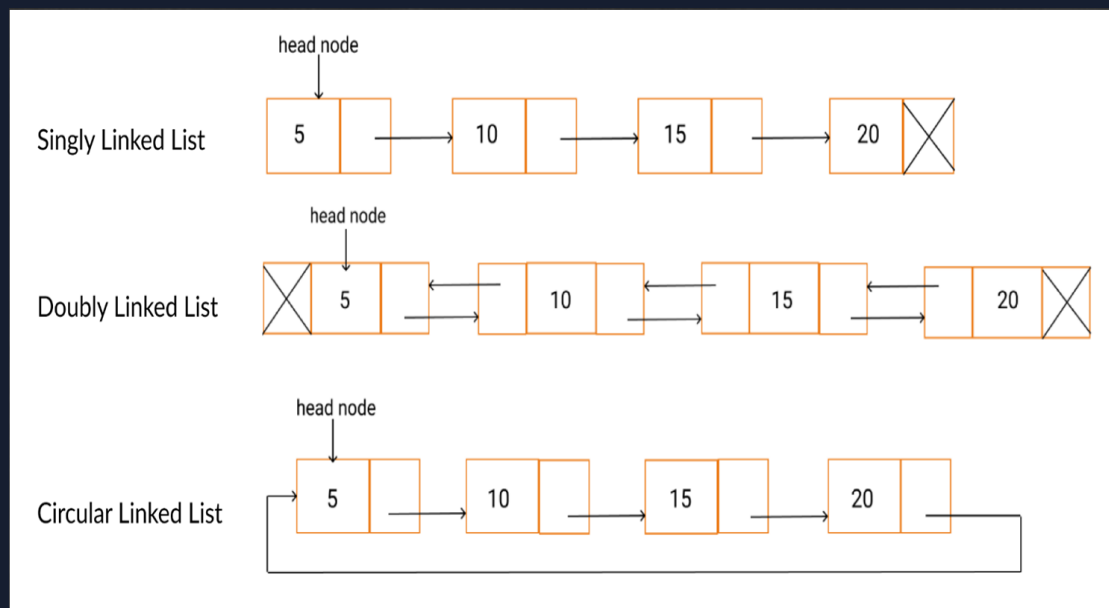
解放連續記憶體

什麼是 List (鏈結串列)？

為了解決 Array 「必須連續」的問題，發明了 Linked List。

- **記憶體可分散**：資料可以散落在記憶體的任何角落，不需要預先要一大塊空間。
- **節點 (Node)**：每個資料被裝在一個特製的「盒子」裡。
- 盒子裡包含兩樣東西：**1. 你的資料 (Data)**、**2. 指標 (Next Pointer)**。
- 指標就像一張紙條，上面寫著「下一個盒子在哪裡」。

Head (頭指標) 負責記住起點



最後一個節點的紙條寫著 Null (空)，代表尋寶結束。

Singly Link List



🎵 單向鏈結串列

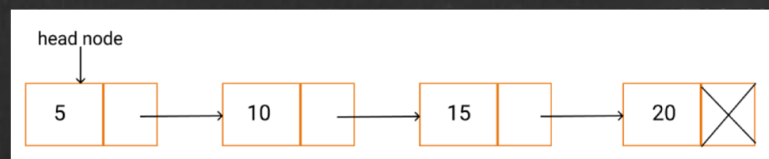
Q:完成以下程式，使輸出結果如右



```
struct SLLNode {  
    int data;  
    SLLNode* next;  
}  
  
struct SLL {  
    SLLNode* head;  
};
```

```
Int main(){  
    //宣告3個SLL  
    //其中2個分別用pushback和pushfront放入  
    11、22、33、44、55  
    //印出3個SLL  
}
```

```
11 22 33 44 55  
55 44 33 22 11  
empty
```





單向鏈結串列

Q:完成以下程式，使輸出結果如右



```
Int main(){
```

```
//在A的33之後插入1010，在55之後插入2020
```

```
//在B的33之前插入1010，在55之前插入2020
```

```
}
```

```
11 22 33 1010 44 55 2020
2020 55 44 1010 33 22 11
```

進階變化：雙向與環狀串列

Doubly Linked List (雙向鏈結串列)

每個節點有兩張紙條！一張寫下一個人 (Next)，一張寫前一個人 (Prev)。

優點：可以往回走，刪除時不需要特地找前一個節點。

缺點：多用了一點記憶體存指標。



Circular Linked List (環狀鏈結串列)

沒有 Null！最後一個節點的 Next 指標，會偷偷指回第一個節點 (Head)。

優點：適合做輪播圖、作業系統的資源輪詢排程 (Round-Robin)。

Circular Linked List





環狀鏈結串列

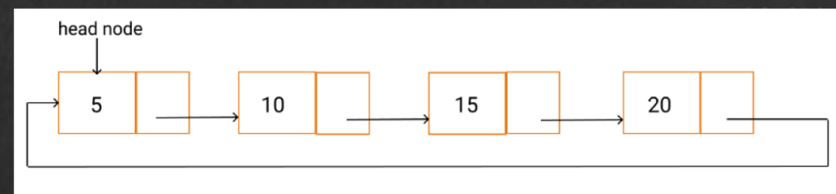
Q:完成以下程式，使輸出結果如右



```
struct CLLNode {  
    int data;  
    CLLNode* next;  
};  
  
struct CLL {  
    CLLNode* head;  
    CLLNode* tail;  
}
```

```
Int main(){  
    //宣告3個CLL  
    //其中2個分別用pushback和pushfront  
    放入11、22、33、44、55  
    //印出3個CLL  
}
```

```
11 22 33 44 55  
55 44 33 22 11  
empty
```





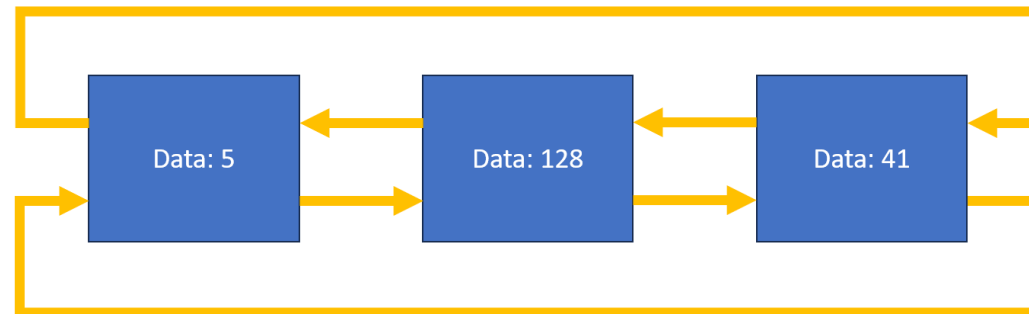
Doubly Linked List



雙向(環狀)鏈結串列

- PushFront

```
a->next = head;  
// 先將新的後指標指到頭  
a->prev = head->prev;  
// 新的前指標指到最後(頭的前一項)  
head->prev->next = a;  
// 將原本最後一項的後指標指到新的節點  
head->prev = a;  
// 將頭的前指標指到新的節點  
head = a;  
// 更新頭指標到新的節點
```



- PushBack

```
a->prev = head->prev;  
// 先將新的前指標只到最後(頭的前一項)  
a->next = head;  
// 將新的後指標指到頭  
head->prev->next = a;  
// 將原本最後一項的後指標指到新的節點  
head->prev = a;  
// 將頭的前指標指到新的節點
```



HW

雙向(環狀)鏈結串列

Q:完成以下程式，使輸出結果如右



```
struct DLLNode {  
    int data;  
    DLLNode* prev;  
    DLLNode* next;  
};  
  
struct DLL {  
    DLLNode* head;  
}
```

```
Int main(){  
    //宣告3個DLL  
    //其中2個分別用pushback和pushfront  
    放入11、22、33、44、55  
    //印出3個DLL  
}
```

```
11 22 33 44 55  
55 44 33 22 11  
empty
```

