Queue

📝 Queue ADT 定義與操作說明

Queue 特性

- FIFO（先進先出）
- 主要操作：
  - `enqueue(item)`：加入元素到隊尾
  - `dequeue()`：移除並回傳隊首元素
  - `front()`：查看隊首元素（不移除）
  - `isEmpty()`：判斷是否為空
  - `isFull()`：判斷是否已滿

陣列版 Queue 特點

- 使用固定大小的一維陣列存放元素
- 需要兩個索引：`front` 和 `rear` 追蹤隊首與隊尾位置
- 非 Circular 版本中，`rear` 只能往右移動，陣列空間用完後不能重複使用已釋放空間（沒有環繞）

🛠 C++ 簡易實作範例

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

class Queue {
private:
    int* arr;     // 陣列指標
    int capacity;  // 最大容量
    int frontIdx;  // 隊首索引
    int rearIdx;   // 隊尾索引（下一個插入位置）
public:
    Queue(int size) {
        capacity = size;
```

```
    arr = new int[capacity];
    frontIdx = 0;
    rearIdx = 0;
}

~Queue() {
    delete[] arr;
}

bool isEmpty() {
    return frontIdx == rearIdx;
}

bool isFull() {
    return rearIdx == capacity;
}

bool enqueue(int item) {
    if (isFull()) {
        cout << "Queue is full, cannot enqueue.\n";
        return false;
    }
    arr[rearIdx++] = item;  // 尾巴插入，rearIdx 往右移
    return true;
}

bool dequeue(int &item) {
    if (isEmpty()) {
        cout << "Queue is empty, cannot dequeue.\n";
        return false;
    }
    item = arr[frontIdx++]; // 取出隊首，frontIdx 往右移
    return true;
}

bool front(int &item) {
    if (isEmpty()) {
        cout << "Queue is empty.\n";
```

```
            return false;
        }
        item = arr[frontIdx];
        return true;
    }

    int size() {
        return rearIdx - frontIdx;
    }
};
```

🔍 使用範例

```
int main() {
    Queue q(5);

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    int val;
    q.front(val);
    cout << "Front element: " << val << endl; // 10

    q.dequeue(val);
    cout << "Dequeued element: " << val << endl; // 10

    q.front(val);
    cout << "Front element: " << val << endl; // 20

    return 0;
}
```

Circular Queue ADT 概念說明

- Circular Queue 利用陣列首尾相接的特性，避免非 Circular Queue「空間浪費」問題。
- 有兩個指標 `front` 和 `rear`：
  - `front` 指向隊首元素（若不空）
  - `rear` 指向下一個可放入元素的位置
- 判斷是否滿或空，採用「預留一格空間」技巧：
  - `isEmpty`：`front == rear`
  - `isFull`：`(rear + 1) % capacity == front`（下一格是 front，表示滿）

🔧 C++ Circular Queue 實作

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

class CircularQueue {
private:
    int* arr;
    int capacity;
    int front;
    int rear;

public:
    CircularQueue(int size) {
        capacity = size + 1;  // 預留一格空間區分滿與空
        arr = new int[capacity];
        front = 0;
        rear = 0;
    }

    ~CircularQueue() {
        delete[] arr;
    }

    bool isEmpty() {
        return front == rear;
    }
```

```
    bool isFull() {
        return (rear + 1) % capacity == front;
    }

    bool enqueue(int item) {
        if (isFull()) {
            cout << "Queue is full, cannot enqueue.\n";
            return false;
        }
        arr[rear] = item;
        rear = (rear + 1) % capacity;
        return true;
    }

    bool dequeue(int &item) {
        if (isEmpty()) {
            cout << "Queue is empty, cannot dequeue.\n";
            return false;
        }
        item = arr[front];
        front = (front + 1) % capacity;
        return true;
    }

    bool getFront(int &item) {
        if (isEmpty()) {
            cout << "Queue is empty.\n";
            return false;
        }
        item = arr[front];
        return true;
    }

    int size() {
        return (rear + capacity - front) % capacity;
    }
};
```

使用範例

```
int main() {
    CircularQueue q(5);  // 只能放 5 個元素，因為多預留一格

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.enqueue(40);
    q.enqueue(50); // 這時候會滿，下一個 enqueue 會失敗

    int val;
    while (!q.isEmpty()) {
        q.dequeue(val);
        cout << val << " ";
    }
    cout << endl;

    return 0;
}
```

小結

- Circular queue 利用 `% capacity` 運算達成循環索引
- `front == rear` 代表空， `(rear + 1) % capacity == front` 代表滿
- 容量要多設一格以區分滿和空的狀態

Deque (Double Ended Queue) 的 ADT

Deque 是一種可以同時從「前端」和「後端」插入或刪除元素的佇列。

主要操作（API）

函式名稱	功能說明
isEmpty()	判斷佇列是否為空
isFull()	判斷佇列是否已滿
insertFront(x)	從前端插入元素 x
insertRear(x)	從後端插入元素 x
deleteFront()	從前端刪除元素，並回傳被刪除元素
deleteRear()	從後端刪除元素，並回傳被刪除元素
getFront()	取得前端元素
getRear()	取得後端元素

---

2. 用一維陣列實作非環狀 Deque (簡單版)

這裡用兩個指標 `front` 和 `rear` 來記錄頭尾，且陣列大小是固定 `capacity`。

- `front` 從左邊開始（0），`rear` 從右邊開始（capacity-1）
- 插入前端時，`front` 往右推
- 插入後端時，`rear` 往左推

---

3. C++ 代碼範例

```
#include <iostream>
#include <stdexcept>

template<typename T>
class Deque {
private:
    int front, rear;     // front 往右移, rear 往左移
    int capacity;
    T* arr;

public:
    Deque(int size) : capacity(size) {
```

```
    arr = new T[capacity];
    front = -1;     // front 空時設為 -1
    rear = capacity;  // rear 空時設為 capacity (超出右界)
}
~Deque() {
    delete[] arr;
}

bool isEmpty() {
    return (front == -1 && rear == capacity);
}

bool isFull() {
    return (front + 1 == rear);
}

void insertFront(T x) {
    if (isFull()) throw std::overflow_error("Deque is full");
    front++;
    arr[front] = x;
}

void insertRear(T x) {
    if (isFull()) throw std::overflow_error("Deque is full");
    rear--;
    arr[rear] = x;
}

T deleteFront() {
    if (isEmpty()) throw std::underflow_error("Deque is empty");
    T val = arr[front];
    front--;
    // 當 front 回到 -1 且 rear == capacity，表示空了
    if (front == -1 && rear == capacity) {
        // 什麼都不用做，isEmpty() 會回 true
    }
    return val;
}
```

```
    T deleteRear() {
        if (isEmpty()) throw std::underflow_error("Deque is empty");
        T val = arr[rear];
        rear++;
        return val;
    }

    T getFront() {
        if (isEmpty()) throw std::underflow_error("Deque is empty");
        return arr[front];
    }

    T getRear() {
        if (isEmpty()) throw std::underflow_error("Deque is empty");
        return arr[rear];
    }
};
```

---

4. 這個版本的特點與限制

- `front` 指標從左往右移，`rear` 指標從右往左移
- 當 `front + 1 == rear` ，表示陣列塞滿了
- 不用環狀方式，所以刪除前面元素後陣列頭不會自動回收空間
- 這個版本比較直覺，但效能不佳，因為沒有利用環狀陣列