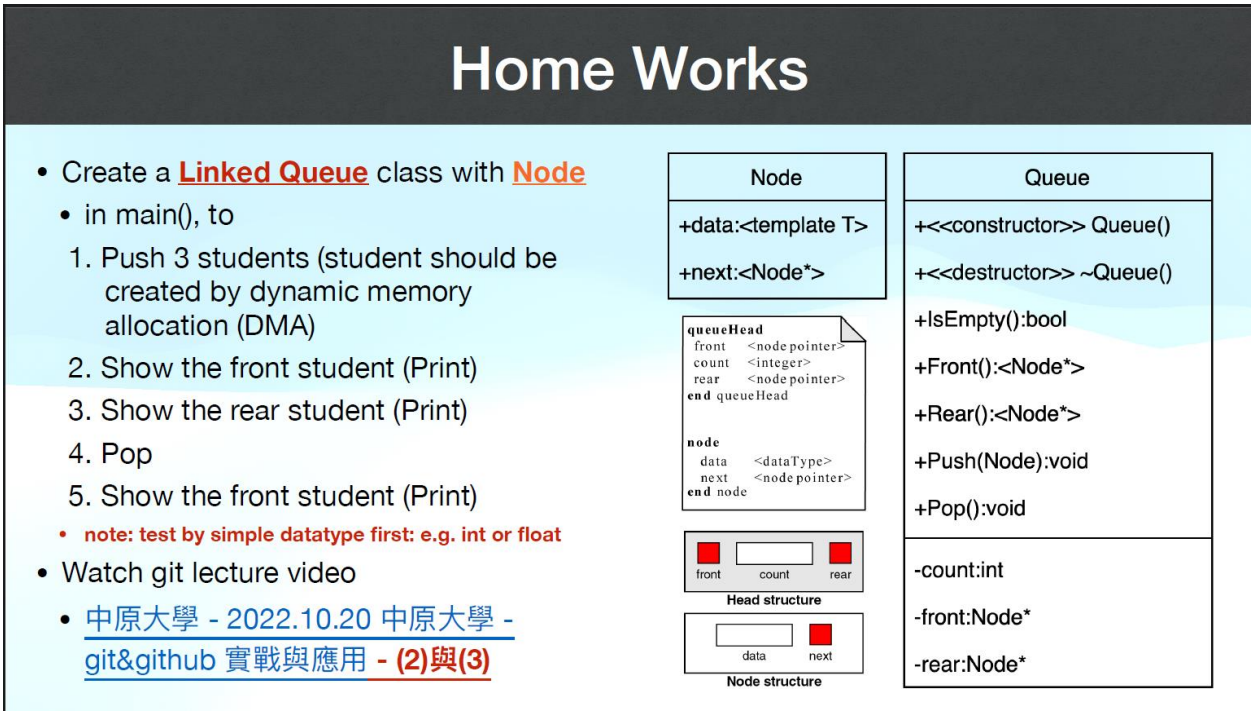
班級：醫工二甲
學號：11125107
姓名：李俞憲

作業一、



執行流程：

```
The front students data:
ID: 1
Name:Alice
The rear students data:
ID: 3
Name:Jack
After popped student's data:
ID: 2
Name:Bob
PS D:\vscode>
```

說明：

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
Queue<Student> StudentQueue;//class:Stack-type:Student(class)
Student *studentArray = new Student[3];
// 使用動態分配的陣列
studentArray[0] = Student(1, "Alice");
studentArray[1] = Student(2, "Bob");
studentArray[2] = Student(3,"Jack");
for(int i=0;i<3;i++)
{
    StudentQueue.Push(studentArray[i]);
}
```

```
cout<<"The front students data:\n";
StudentQueue.Front() -> data.Print();
cout<<"The rear students data:\n";
StudentQueue.Rear() -> data.Print();

StudentQueue.Pop();
cout<<"After popped student's data:\n";
StudentQueue.Front() -> data.Print();
return 0;
```

先創建 Student 的 array，以及 Queue，隨後進行添加
隨後根據題意進行相應操作，

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
template<class T>
int Queue<T>::Push(T &item) {
    Node<T>* new_node = new Node<T>(); //利用動態記憶體，直到用盡記憶體空間
    new_node->data = item; //存入對應資料
    new_node->next = NULL; //新資料應存入NULL連結(GND)

    if (IsEmpty())
    {
        front = new_node; //若內部無元素，front指向新元素
    }
    else
    {
        rear->next = new_node; //否則舊rear指向新元素的位置
    }

    rear = new_node; //rear指向新元素
    count++;

    return 0;
}
```

Queue 的 Push 在這邊使用了 2 個指標，以及新的資料需要接地，再來有 2 種情形需要討論，分別是內部第一筆資料傳入時，front 需要指向新資料，另一種是當內部有一筆資料之後需要將舊資料指向新資料以及 rear 指向新資料

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
template<class T>
int Queue<T>::Pop() {
    if (IsEmpty()) {
        throw std::runtime_error("Queue is empty.");//當內部無值時報錯
    }

   Node<T>* temp = front;
    front = front->next;
    delete temp;
    count--;

    if (count == 0) //如果沒有元素時
    {
        rear = NULL;
        front = NULL;
    }
    else if (count == 1) // 如果僅有一個元素時
    {
        rear = front;
    }

    return 0;
}
```

Pop 的部份是將 fornt 的資料刪除，但是當 Queue 沒有元素時，rear、front 需要初始化，皆指向 NULL，或是當內部僅有一個元素時 rear 和 front 需要指向該元素

程式碼：

//main

```
#include "1026_11125107.hpp"
#include "1026_11125107f.cpp"
#include "1026_11125107_Node.hpp"
#include "1026_11125107_Student.hpp"

using namespace std;
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
int main()
{
    Queue<Student> StudentQueue;//class:Stack-type:Student(class)
    Student *studentArray = new Student[3];
    // 使用動態分配的陣列
    studentArray[0] = Student(1, "Alice");
    studentArray[1] = Student(2, "Bob");
    studentArray[2] = Student(3,"Jack");
    for(int i=0;i<3;i++)
    {
        StudentQueue.Push(studentArray[i]);
    }
    cout<<"The front students data:\n";
    StudentQueue.Front() -> data.Print();
    cout<<"The rear students data:\n";
    StudentQueue.Rear() -> data.Print();

    StudentQueue.Pop();
    cout<<"After popped student's data:\n";
    StudentQueue.Front() -> data.Print();
    return 0;
}
```

//.hpp

```
#include <iostream>
#include<cstdlib>
#include<cstring>
#include <typeinfo>
#include "1026_11125107_Node.hpp"
#include "1026_11125107_Student.hpp"
using namespace std;
#pragma once



template<class T>
class Queue {
private:
    int count;
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
    Node<T>* front; // 序列前端指標
    Node<T>* rear; // 序列後端指標

public:
    Queue();
    ~Queue();
    bool IsEmpty();//檢查是否為空，T:為空；F:不為空
    Node<T>* Front();//前端元素
    Node<T>* Rear();//後端元素
    int Push(T &item);//放入元素
    int Pop();//刪除元素
};

template<class T>
Queue<T>::Queue() : front(NULL) , rear(NULL) , count(0) {}//預設值

template<class T>
Queue<T>::~Queue()
{
    while (front) //front 會依次將所有元素進行刪除，直到其指向 NULL
    {
        Node<T>* temp = front;
        front = front->next;
        delete temp;//設定暫存元素進行刪除(比較保險)
    }
}

template<class T>
bool Queue<T>::IsEmpty() {
    if (front == NULL)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
template<class T>
Node<T>* Queue<T>::Front() {
    if (IsEmpty())
    {
        throw std::runtime_error("Queue is empty.");//出現空時報錯
    }
    else
    {
        return front;
    }
}

template<class T>
Node<T>* Queue<T>::Rear() {
    if (IsEmpty())
    {
        throw std::runtime_error("Queue is empty.");//出現空時報錯
    }
    else
    {
        return rear;
    }
}


template<class T>
int Queue<T>::Push(T &item) {
    Node<T>* new_node = new Node<T>(); //利用動態記憶體，直到用盡記憶體空間
    new_node->data = item; //存入對應資料
    new_node->next = NULL; //新資料應存入 NULL 連結(GND)

    if (IsEmpty())
    {
        front = new_node; //若內部無元素，front 指向新元素
    }
    else
    {
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
        rear->next = new_node; //否則舊 rear 指向新元素的位置
    }

    rear = new_node; //rear 指向新元素
    count++;

    return 0;
}


template<class T>
int Queue<T>::Pop() {
    if (IsEmpty()) {
        throw std::runtime_error("Queue is empty.");//當內部無值時報錯
    }

   Node<T>* temp = front;
    front = front->next;
    delete temp;
    count--;

    if (count == 0) //如果沒有元素時
    {
        rear = NULL;
        front = NULL;
    }
    else if (count == 1) // 如果僅有一個元素時
    {
        rear = front;
    }

    return 0;
}
```

//Node.hpp

```
#include <iostream>
#include<cstdlib>
#include<cstring>
#include <typeinfo>
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
using namespace std;
#pragma once

//Node
template <class T>
class Node {
public:
    T data; // 資料
    Node<T>* next; // 傳送門

    Node();//建構子
    ~Node();//解構子
};

template <class T>
Node<T>::Node() : next(NULL), data() {}//預設值

template<class T>
Node<T>::~Node()
{

}
```

//Student.hpp

```
#include <iostream>
#include<cstdlib>
#include<cstring>
#include <typeinfo>

using namespace std;
#pragma once

class Student {
private:
    int ID;
    char* name;

public:
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
    Student();
    ~Student();
    Student(int stud_id, const char* stud_name);
    int SetID(int stud_id);
    int SetName(char* stud_name);
    int GetID();
    char* GetName();
    void Print();
    void operator=(Student src);//等號(class=class)
};
```

//.cpp

```
#include "1026_11125107.hpp"

Student::Student() : ID(0), name(nullptr) {}


Student :: ~Student()
{
    // if (name != NULL)
    // {
    //     delete[] name;
    // }
    // name = NULL;
}

Student::Student(int stud_id, const char* stud_name) {
    ID = stud_id;
    name = new char[strlen(stud_name) + 1];
    strcpy(name, stud_name);
}

int Student :: SetID(int stud_id)
{
    ID = stud_id;
    return 0;
}
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
int Student :: SetName(char* stud_name)
{
    if (name != stud_name) {
        if (name != nullptr) {
            delete[] name;
        }
        name = new char[strlen(stud_name) + 1];
        strcpy(name, stud_name);
    }
    return 0;
}

int Student :: GetID()
{
    int id_temp = ID;
    return id_temp;
}

char* Student :: GetName()
{
    char* data2pub = new char[strlen(name)+1];
    strcpy(data2pub,name);
    return data2pub;
}

void Student :: Print()
{
    cout << "ID: " << GetID()<<endl<<"Name:";
    for (int i=0;i<strlen(name);i++)
    {
        char* ptr;
        ptr = GetName();
        cout<<ptr[i];
    }
    cout<<endl;
}

void Student::operator=(Student src)//A=B
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
{
    if (name != NULL)
    {
        delete[] name;//刪除舊資料
        name = NULL;
    }

    ID = src.ID;
    name = new char[strlen(src.name)];
    strcpy(name,src.name);
    return;
}
```

補充說明（遇到的困難或心得，選填）：

整體來說，改變不大，只是多了 Node 的指標需要多花點時間去理解，而且在進行這次作業後發現這種方式的確比使用陣列來存數據還要來得方便許多，直接免去了排序問題，但是有一個風險是每筆資料環環相扣，如果 Queue 的指標未正確使用，很可能導致數據丟失的問題。