班級：醫工二甲
學號：11125114
姓名：許仁駿

作業一、 【創建一個 Student class，再用 template 創一個 Queue class，並將 Student 用進 Queue】

執行流程：

【截圖 1】

```
#pragma once
#include <iostream>
#include <cstring>
#pragma warning(disable : 4996)
using namespace std;

//Student class 實作
class Student
{
public:
    Student();
    Student(int stud_id, const char* stud_name);
    ~Student() {};

    int SetID(int stud_id);
    int SetName(const char* stud_name);
    int GetID();
    char* GetName();
    void Print();

private:
    int ID;
    char* name;
};
```

說明： 先在 Header 檔創建所需的類別 Student。

【截圖 2】

班級：醫工二甲
學號：11125114
姓名：許仁駿

```
#include "1019.h"
#include <iostream>
using namespace std;

Student::Student() //建構子
{
    ID = 0;
    name = NULL;
}

Student::Student(int stud_id, const char* stud_name) //建構子
{
    ID = stud_id;
    name = new char[10];
    std::strcpy(name, stud_name);
}

int Student::SetID(int stud_id)
{
    ID = stud_id;
    return 1;
}
```

班級：醫工二甲
學號：11125114
姓名：許仁駿

```
int Student::SetName(const char* stud_name)
{
    name = new char[10];
    std::strcpy(name, stud_name);
    return 1;
}

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

char* Student::GetName()
{
    return name;
}


void Student::Print()
{
    cout << "student's id is:" << ID << endl;
    cout << "student's name is:" << name << endl;
}
```

說明：Source 檔定義函式。

【截圖 3】

班級：醫工二甲
學號：11125114
姓名：許仁駿

```
//Queue 實作
template<class T>
class Queue
{
public:
    Queue();
    ~Queue();
    Queue(int MaxQueueSize);

    bool IsEmpty();

    T& Front();
    T Rear();
    void Push(T& element);
    void Pop();

private:
    T* queue_elements;
    int front;
    int rear;
    int capacity;
};
```

說明：在 Header 檔創建所需的類別 Queue (用 template 創建)

【截圖 4】

班級：醫工二甲
學號：11125114
姓名：許仁駿

```
template <class T>
inline Queue<T>::Queue()
{
    this->capacity = 3;
    queue_elements = new T[capacity];
    front = 0; //計算Pop次數，以得到front的座標
    rear = -1; //初始化rear指標為-1，表示佇列是空的
}

template <class T>
Queue<T>::Queue(int capacity)
{
    this->capacity = capacity;
    queue_elements = new T[capacity];
    front = 0; //計算Pop次數，以得到front的座標
    rear = -1; //初始化rear指標為-1，表示佇列是空的
}

template <class T>
Queue<T>::~Queue()
{
    delete[] queue_elements; //delete堆疊內存
}
```

```
template <class T>
bool Queue<T>::IsEmpty()
{
    if (rear - front == -1) //若成立，代表剛Pop出rear那一格的東西，即empty。
    {
        front = 0;
        rear = -1;
        return true;
    }
    else
    {
        return false;
    }
}
```

班級：醫工二甲
學號：11125114
姓名：許仁駿

```
template <class T>
void Queue<T>::Push(T& element)
{
    if (rear == capacity - 1 && front==0) //若成立，表示rear在最後一格，且front在第一格，佇列已滿
    {
        throw std::runtime_error("佇列已滿，無法推進元素");
    }
    else if (rear == capacity - 1) //若成立，表示rear到底，但front前面還有空。
    {
        for (int i = front;i <= capacity-1;i++)
        {
            queue_elements[i - front] = queue_elements[i]; //所以要把整個佇列往前移front格
        }
    }
    else
    {
        queue_elements[++rear] = element; //推進元素並更新rear指標
    }
}
```

```
template <class T>
void Queue<T>::Pop()
{
    if (IsEmpty()) //若為empty，則在IsEmpty()把front跟rear重製
    {
        throw std::runtime_error("堆疊是空的，無法彈出元素");
    }

    front += 1; //計算Pop次數，同時表示front座標
    cout << "已 Pop 出 Front 之成員!" << endl;
}

template <class T>
T& Queue<T>::Front()
{
    return queue_elements[front];
}

template <class T>
T Queue<T>::Rear()
{
    return queue_elements[rear];
}
```

班級：醫工二甲
學號：11125114
姓名：許仁駿

說明：在 Header 檔 定義函式。

【截圖 5】

```
#include <iostream>
using namespace std;
#include"1019.h"

int main()
{
    //Student class 實作
    Student Eric;
    const char* name = "Eric";
    Eric.SetID(11125114);
    Eric.SetName(name);
    int a = Eric.GetID();
    char* b = Eric.GetName();
    cout << a << endl;
    cout << b << endl;
    Eric.Print();

    //create 3 Student class by DMA
    Student* Eric_ck = new Student(11125114, "Eric");
    Student* Liang = new Student(11141346, "Liang");
    Student* Peng = new Student(11225228, "Peng");
```

說明：在 main 檔 ，先做 Student 的測試，再用動態記憶體創建三個。

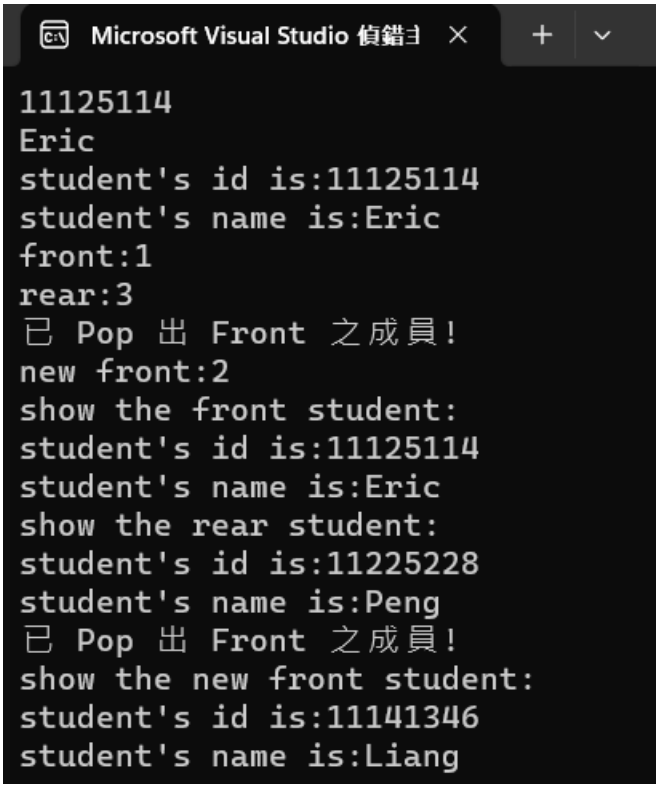
【截圖 6】

班級：醫工二甲
學號：11125114
姓名：許仁駿

```
//Queue class 實作 (T = int)
Queue<int> Number;
int aa = 1, bb = 2, cc = 3;
Number.Push(aa);
Number.Push(bb);
Number.Push(cc);
int front = Number.Front();
int rear = Number.Rear();
cout << "front:" << front << endl;
cout << "rear:" << rear << endl;
Number.Pop();
int new_front= Number.Front();
cout << "new front:" << new_front << endl;

//Queue class 實作 (T = Student, Student 為 class)
Queue<Student> Student_Queue;
Student_Queue.Push(*Eric_ck); //Push 3 students
Student_Queue.Push(*Liang);
Student_Queue.Push(*Peng);
cout << "show the front student:" << endl;
Student_Queue.Front().Print(); //Show the front student
cout << "show the rear student:" << endl;
Student_Queue.Rear().Print(); //Show the rear student
Student_Queue.Pop();//Pop
cout << "show the new front student:" << endl;
Student_Queue.Front().Print(); //Show the front student
```

班級：醫工二甲
學號：11125114
姓名：許仁駿



說明：之後在 main 檔測試 Queue， 再把剛剛的 3 個 Student 丟進去。

程式碼：

Main 檔:

```
#include <iostream>
using namespace std;
#include"1019.h"

int main()
{
    //Student class 實作
    Student Eric;
    const char* name = "Eric";
    Eric.SetID(11125114);
    Eric.SetName(name);
    int a = Eric.GetID();
    char* b = Eric.GetName();
```

班級：醫工二甲
學號：11125114
姓名：許仁駿

```
    cout << a << endl;
    cout << b << endl;
    Eric.Print();

    //create 3 Student class by DMA
    Student* Eric_ck = new Student(11125114, "Eric");
    Student* Liang = new Student(11141346, "Liang");
    Student* Peng = new Student(11225228, "Peng");

    //Queue class 實作 (T = int)
    Queue<int> Number;
    int aa = 1, bb = 2, cc = 3;
    Number.Push(aa);
    Number.Push(bb);
    Number.Push(cc);
    int front = Number.Front();
    int rear = Number.Rear();
    cout << "front:" << front << endl;
    cout << "rear:" << rear << endl;
    Number.Pop();
    int new_front= Number.Front();
    cout << "new front:" << new_front << endl;

    //Queue class 實作 (T = Student, Student 為 class)
    Queue<Student> Student_Queue;
    Student_Queue.Push(*Eric_ck); //Push 3 students
    Student_Queue.Push(*Liang);
    Student_Queue.Push(*Peng);
    cout << "show the front student:" << endl;
    Student_Queue.Front().Print(); //Show the front student
    cout << "show the rear student:" << endl;
    Student_Queue.Rear().Print(); //Show the rear student
    Student_Queue.Pop();//Pop
    cout << "show the new front student:" << endl;
    Student_Queue.Front().Print(); //Show the front student

    return 0;
}
```

Source 檔:

```
#include "1019.h"
#include <iostream>
using namespace std;
```

班級：醫工二甲
學號：11125114
姓名：許仁駿

```
Student::Student() //建構子
{
        ID = 0;
        name = NULL;
}

Student::Student(int stud_id, const char* stud_name) //建構子
{
        ID = stud_id;
        name = new char[10];
        std::strcpy(name, stud_name);
}

int Student::SetID(int stud_id)
{
        ID = stud_id;
        return 1;
}

int Student::SetName(const char* stud_name)
{
        name = new char[10];
        std::strcpy(name, stud_name);
        return 1;
}

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

char* Student::GetName()
{
        return name;
}


void Student::Print()
{
        cout << "student's id is:" << ID << endl;
        cout << "student's name is:" << name << endl;
        }
```

班級：醫工二甲
學號：11125114
姓名：許仁駿

Header 檔:

```
#pragma once
#include <iostream>
#include <cstring>
#pragma warning(disable : 4996)
using namespace std;

//Student class 實作
class Student
{
public:
    Student();
    Student(int stud_id, const char* stud_name);
    ~Student() {};

    int SetID(int stud_id);
    int SetName(const char* stud_name);
    int GetID();
    char* GetName();
    void Print();

private:
    int ID;
    char* name;
};


//Queue 實作
template<class T>
class Queue
{
public:
    Queue();
    ~Queue();
    Queue(int MaxQueueSize);

    bool IsEmpty();

    T& Front();
    T Rear();
    void Push(T& element);
    void Pop();
```

班級：醫工二甲
學號：11125114
姓名：許仁駿

```
private:
    T* queue_elements;
    int front;
    int rear;
    int capacity;
};

template <class T>
inline Queue<T>::Queue()
{
    this->capacity = 3;
    queue_elements = new T[capacity];
    front = 0; //計算Pop次數，以得到front的座標
    rear = -1; //初始化rear指標為-1，表示佇列是空的
}

template <class T>
Queue<T>::Queue(int capacity)
{
    this->capacity = capacity;
    queue_elements = new T[capacity];
    front = 0; //計算Pop次數，以得到front的座標
    rear = -1; //初始化rear指標為-1，表示佇列是空的
}

template <class T>
Queue<T>::~Queue()
{
    delete[] queue_elements; //delete堆疊內存
}

template <class T>
bool Queue<T>::IsEmpty()
{
    if (rear - front == -1) //若成立，代表剛Pop出rear那一格的東西，即empty。
    {
        front = 0;
        rear = -1;
        return true;
    }
    else
    {
        return false;
```

班級：醫工二甲
學號：11125114
姓名：許仁駿

```
    }
}

template <class T>
void Queue<T>::Push(T& element)
{
    if (rear == capacity - 1 && front==0) //若成立，表示rear在最後一格，且front在第一格，佇列已滿
    {
        throw std::runtime_error("佇列已滿，無法推進元素");
    }
    else if (rear == capacity - 1) //若成立，表示rear到底，但front前面還有空。
    {
        for (int i = front;i <= capacity-1;i++)
        {
            queue_elements[i - front] = queue_elements[i]; //所以要把整個佇列往前移front格
        }
    }
    else
    {
        queue_elements[++rear] = element; //推進元素並更新rear指標
    }
}

template <class T>
void Queue<T>::Pop()
{
    if (IsEmpty()) //若為empty，則在IsEmpty()把front跟rear重製
    {
        throw std::runtime_error("堆疊是空的，無法彈出元素");
    }

    front += 1; //計算Pop次數，同時表示front座標
    cout << "已 Pop 出 Front 之成員!" << endl;
}

template <class T>
T& Queue<T>::Front()
{
    return queue_elements[front];
}

template <class T>
T Queue<T>::Rear()
```

班級：醫工二甲
學號：11125114
姓名：許仁駿

```
{
    return queue_elements[rear];
}
```

補充說明（遇到的困難或心得，選填）：

在上一次作業中了解 template 以及 Stack 後，這次的 Queue 概念其實相似，改變的大概是把 top 去掉，多了 front 以及 rear 的概念，並且 Pop 是把最前先 Push 的移出，就好比 Stack 是堆疊，Queue 是排隊一樣，了解完後，其實我在上課就順利完成作業了!