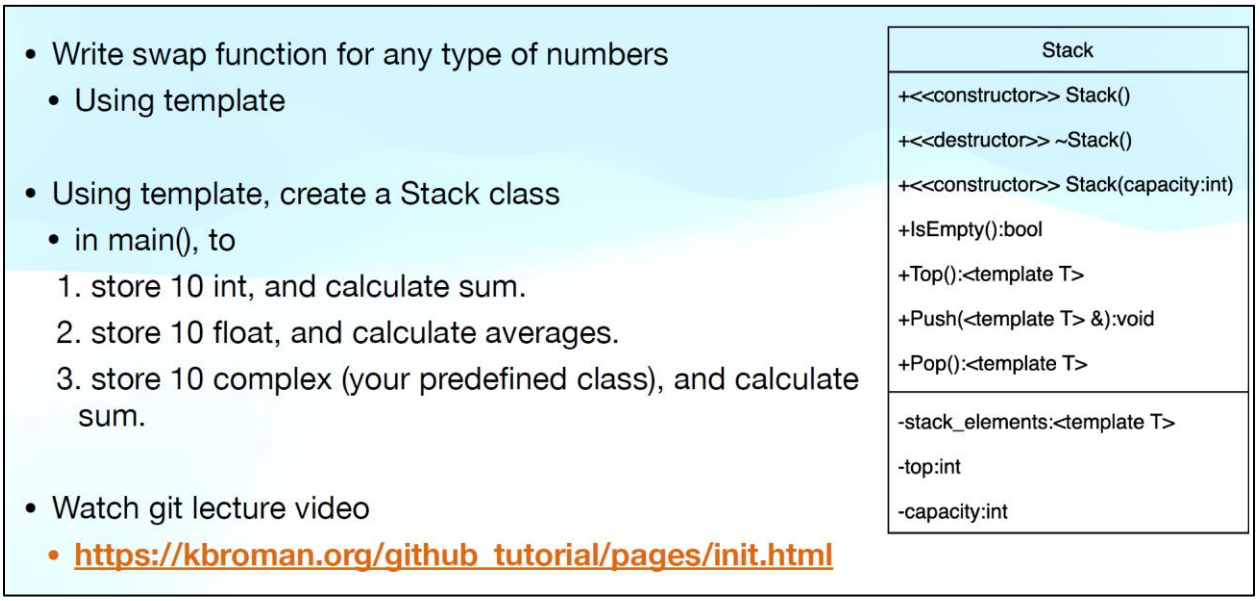
班級：醫工二甲
學號：11125107
姓名：李俞憲

作業一、



執行流程：

```
作業1
置換前(int): a=3, b=5
置換前(float): i=2.42, j=1.76
置換前(double): x=2.3, y=4.6
置換後(int): a=5, b=3
置換後(float): i=1.76, j=2.42
置換後(double): x=4.6, y=2.3
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
作業2:
堆疊(int)為空
堆疊(float)為空
堆疊(complex)為空

開使添加數值...
頂部元素(int): 0
頂部元素(float)0.1
頂部元素(complex)1+9i

彈出的元素(int)是: 0,9,8,7,6,5,4,3,2,1
int的sum= 45
彈出的元素(float)是: 0.1,9.9,8.8,7.7,6.6,5.5,4.4,3.3,2.2,1.1
float的ave= 4.96
彈出的元素(complex)是:
1+9i
2+8i
3+7i
4+6i
5+5i
6+4i
7+3i
8+2i
9+1i
10+0i
complex的sum為: 55+45i

堆疊(int)為空
堆疊(float)為空
堆疊(complex)為空
PS D:\vscode>
```

說明：

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
cout<< 作業1\n";
int a = 3;
int b = 5;

float i = 2.42;
float j = 1.76;

double x = 2.3;
double y = 4.6;

cout << "置換前(int): a=" << a << ", b=" << b << endl;
cout << "置換前(float): i=" << i << ", j=" << j << endl;
cout << "置換前(double): x=" << x << ", y=" << y << endl;

mySwap(a, b);
mySwap(i, j);
mySwap(x, y);

cout << "置換後(int): a=" << a << ", b=" << b << endl;
cout << "置換後(float): i=" << i << ", j=" << j << endl;
cout << "置換後(double): x=" << x << ", y=" << y << endl;
```

```
template <class T>
void mySwap(T &x, T &y)
{
    T temp = x;
    x = y;
    y = temp;
}
```

作業 1 的部分其實只是熟悉 template 的性質而已，其中的 function 採用傳址的方式，使得空間更加有效率的進行運用。

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
//作業2
//使用陣列的方式與for迴圈搭配給值，增加效率和避免關節炎
int int_list[10]={1,2,3,4,5,6,7,8,9,0};
float float_list[10]={1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9,0.1};
const int numInstances = 10;
complex cpx[numInstances];
for(int i=0;i<numInstances;i++)
{
    cpx[i] = complex(10-i,i);
}

Stack<int> intStack(10); // 初始化堆疊(int)容量為10
Stack<float> floatStack(10); // 初始化堆疊(int)容量為10
Stack <complex> complexStack(10);
```

再來是作業 2 的部分，這邊先將要輸入的資料先創建出來，我使用陣列的方式已增加效率。

右圖是檢驗我的值是否為空，因此時尚未輸入，因此堆疊為空才是正確的。

最後將陣列內容逐一輸出，為了美觀，我在內部使用 if 判斷式，將每一列的頭尾加上中括號，並分段。

```
// 檢查堆疊是否為空
if (intStack.IsEmpty())
{
    cout << "堆疊(int)為空" << endl;
}
else
{
    cout << "堆疊(int)不為空" << endl;
}

if (floatStack.IsEmpty())
{
    cout << "堆疊(float)為空" << endl;
}
else
{
    cout << "堆疊(float)不為空" << endl;
}
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
// 取得頂部元素
int int_top = intStack.Top();
cout << "頂部元素(int): " << int_top << endl;
float float_top = floatStack.Top();
cout<< "頂部元素(float)" << float_top << endl;
complex temp;
temp = complexStack.Top();
cout << "頂部元素(complex)";
temp.print();
cout<<endl;
```

這邊為了取得頂部的元素，先創建一組同類型的資料作為 temp 儲存，再輸出，尤其是 complex 是 class 型態可以配合自身的 print()使用

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
cout << "弹出的元素(int)是: ";
int sum = 0;
for(int i=0;i<10;i++)
{
    int int_popped = intStack.Pop();
    if (i<9)
    {
        cout << int_popped << ",";
    }
    else
    {
        cout << int_popped << endl;
    }
    sum += int_popped;
}
cout << "int的sum= " << sum << endl;
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
cout << "彈出的元素(float)是: ";
float ave = 0.0;
for(int i=0;i<10;i++)
{
    float float_popped = floatStack.Pop();
    if (i<9)
    {
        cout << float_popped << ",";
    }
    else
    {
        cout << float_popped << endl;
    }
    ave += float_popped;
}
cout << "float的ave= " << (ave/10) << endl;
```

```
cout << "彈出的元素(complex)是:\n";
complex cpx_add(0,0);
for(int i=0;i<10;i++)
{
    complex out_temp;
    out_temp = complexStack.Pop();
    out_temp.print();
    cpx_add = (cpx_add + out_temp);
}
cout << "complex的sum為: ";
cpx_add.print();
cout<<endl;
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

接下來是將內部的資料逐一 pop 出來，int、float 型態的處理方式很簡單，也是用類似上述的方法進行輸出，complex 的部份也是如此，不過再計算 sum 時須配合 operator 的 function 使用。

程式碼：

//main

```
#include "1012_11125107.h"
#include "1012_11125107f.cpp"

using namespace std;

int main()
{
    //交換數值
    cout<<"作業 1\n";
    int a = 3;
    int b = 5;

    float i = 2.42;
    float j = 1.76;

    double x = 2.3;
    double y = 4.6;

    cout << "置換前(int): a=" << a << ", b=" << b << endl;
    cout << "置換前(float): i=" << i << ", j=" << j << endl;
    cout << "置換前(double): x=" << x << ", y=" << y << endl;

    mySwap(a, b);
    mySwap(i, j);
    mySwap(x, y);

    cout << "置換後(int): a=" << a << ", b=" << b << endl;
    cout << "置換後(float): i=" << i << ", j=" << j << endl;
    cout << "置換後(double): x=" << x << ", y=" << y << endl;

    cout<<"\n 作業 2:\n";
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
//作業 2
//使用陣列的方式與 for 迴圈搭配給值，增加效率和避免關節炎
int int_list[10]={1,2,3,4,5,6,7,8,9,0};
float float_list[10]={1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9,0.1};
const int numInstances = 10;
complex cpx[numInstances];
for(int i=0;i<numInstances;i++)
{
    cpx[i] = complex(10-i,i);
}

Stack<int> intStack(10); // 初始化堆疊(int)容量為 10
Stack<float> floatStack(10); // 初始化堆疊(int)容量為 10
Stack <complex> complexStack(10);


// 檢查堆疊是否為空
if (intStack.IsEmpty())
{
    cout << "堆疊(int)為空" << endl;
}
else
{
    cout << "堆疊(int)不為空" << endl;
}

if (floatStack.IsEmpty())
{
    cout << "堆疊(float)為空" << endl;
}
else
{
    cout << "堆疊(float)不為空" << endl;
}

if (complexStack.IsEmpty())
{
    cout << "堆疊(complex)為空" << endl;
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
    }
    else
    {
        cout << "堆疊(complex)不為空" << endl;
    }

    cout<<"\n 開使添加數值..."<<endl;

    for(int i=0;i<10;i++)
    {
        intStack.Push(int_list[i]);
    }

    for(int i=0;i<10;i++)
    {
        floatStack.Push(float_list[i]);
    }

    for(int i=0;i<10;i++)
    {
        complexStack.Push(cpx[i]);
    }

    // 取得頂部元素
    int int_top = intStack.Top();
    cout << "頂部元素(int): " << int_top << endl;
    float float_top = floatStack.Top();
    cout<< "頂部元素(float)" << float_top << endl;
    complex temp;
    temp = complexStack.Top();
    cout << "頂部元素(complex)";
    temp.print();
    cout<<endl;
    // 逐一彈出頂部元素並進行計算
    cout << "彈出的元素(int)是: ";
    int sum = 0;
    for(int i=0;i<10;i++)
    {
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
        int int_popped = intStack.Pop();
        if (i<9)
        {
            cout << int_popped << ",";
        }
        else
        {
            cout << int_popped << endl;
        }
        sum += int_popped;
    }
    cout << "int 的 sum= " << sum << endl;

    cout << "彈出的元素(float)是: ";
    float ave = 0.0;
    for(int i=0;i<10;i++)
    {
        float float_popped = floatStack.Pop();
        if (i<9)
        {
            cout << float_popped << ",";
        }
        else
        {
            cout << float_popped << endl;
        }
        ave += float_popped;
    }
    cout << "float 的 ave= " << (ave/10) << endl;

    cout << "彈出的元素(complex)是:\n";
    complex cpx_add(0,0);
    for(int i=0;i<10;i++)
    {
        complex out_temp;
        out_temp = complexStack.Pop();
        out_temp.print();
        cpx_add = (cpx_add + out_temp);
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
    }
    cout << "complex 的 sum 為: ";
    cpx_add.print();
    cout<<endl;

    // 檢查堆疊是否為空
    if (intStack.IsEmpty())
    {
        cout << "堆疊(int)為空" << endl;
    }
    else
    {
        cout << "堆疊(int)不為空" << endl;
    }

    if (floatStack.IsEmpty())
    {
        cout << "堆疊(float)為空" << endl;
    }
    else
    {
        cout << "堆疊(float)不為空" << endl;
    }

    if (complexStack.IsEmpty())
    {
        cout << "堆疊(complex)為空" << endl;
    }
    else
    {
        cout << "堆疊(complex)不為空" << endl;
    }
    return 0;
}
```

//.h

```
#include <iostream>
#include<cstdlib>
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
#include<cstring>
#include <typeinfo>

using namespace std;
#pragma once

template<class T>
class Stack {
private:
    T* stack_elements; // 堆疊元素量
    int top; // 堆疊頂部指標
    int capacity; // 堆疊容量

public:
    Stack();//動態記憶體自動配置
    Stack(int cap);//手動輸入空間值
    ~Stack();//解構子 delete[]
    bool IsEmpty();//檢查是否為空，T:為空；F:不為空
    T& Top();//頂部元素
    void Push(T &item);//放入元素
    T Pop();//取出元素
};

template<class T>
Stack<T>::Stack() : stack_elements(NULL), top(-1), capacity(0) {}//預設值

template<class T>
Stack<T>::Stack(int cap) : top(-1), capacity(cap) {
    stack_elements = new T[capacity];
}

template<class T>
Stack<T>::~Stack() {
    delete[] stack_elements;
}

template<class T>
bool Stack<T>::IsEmpty() {
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
    if (top == -1)
        return true;
    else
        return false;
}

template<class T>
T& Stack<T>::Top() {
    if (top == -1) {
        throw std::runtime_error("Stack is empty.");//出現空時報錯
    }
    return stack_elements[top];//輸出 top 指標所指的位置
}

template<class T>
void Stack<T>::Push(T &item) {
    if (top == capacity - 1) {
        throw std::runtime_error("Stack is full.");//當空間滿時報錯
    }
    stack_elements[++top] = item;//新物件加入時，top 值++
}

template<class T>
T Stack<T>::Pop() {
    if (top == -1) {
        throw std::runtime_error("Stack is empty.");//當內部無值時報錯
    }
    return stack_elements[top--];//物件取出時 top 值--
}

template <class T>
void mySwap(T &x, T &y)
{
    T temp = x;
    x = y;
    y = temp;
}
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
//complex
class complex
{
private:
    float real;
    float image;
public:
    complex();
    ~complex(){};
    complex(float,float);
    int set(float,float);
    void print();
    float GetReal();
    float GetImage();
    complex operator=(complex src);//等號(class+class)
    complex operator+(complex src);
};
```

//.cpp

```
#include "1012_11125107.h"

complex::complex()
{
    real=0;
    image=0;
}

complex::complex(float real_in,float image_in)
{
    real = real_in;
    image = image_in;
}
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
int complex::set(float real_set,float image_set)
{
    real = real_set;
    image = image_set;
    return 0;
}

float complex::GetReal()
{
    float temp_real=real;
    return temp_real;
}

float complex::GetImage()
{
    float temp_Image=image;
    return temp_Image;
}

void complex::print()
{
    cout<<GetReal()<<"+"<<GetImage()<<"i";
    cout<<endl;

}

complex complex:: operator=(complex src)//this 指標是(當下呼叫而已)
{
    complex temp_equal;
    this->real= src.real;
    this->image= src.image;
    temp_equal.real = this ->real;
    temp_equal.image = this ->image;
    return temp_equal;
}

complex complex:: operator+(complex src)
{
```

班級：醫工二甲
學號：11125107
姓名：李俞憲

```
    complex temp;
    temp.real=this->real+src.real;
    temp.image=this->image+src.image;
    return temp;
}
```

補充說明（遇到的困難或心得，選填）：

這次的內容有點像是一種衍伸版本，雖然運用到的技巧和之前教的內容差異不大，但是多了一個 template，雖然在城市的表達方式有些不同，但是熟悉後其時不會太困難。我覺得自己在程式方面的功力也有提升了，一開始在創建 10 組資料(int、float、complex)時想要用最簡單的方式，一個一個創，但是這樣子反而花費更多時間以及維護性更低，後來想到可以用陣列的方式會更加有效率，其中 complex 的陣列技巧就是上次上課有提到的內容。