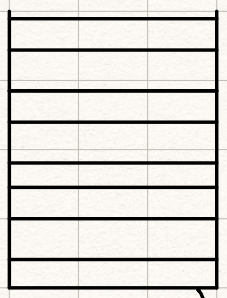
堆疊 stack



函數、程式、視窗、執行檔
皆是如此(最後呼叫先執行)

先進後出(疊盤子)，大小事先定義
(固定大小)

0(沒有東西的話是-1)

```
※ define  size  10
class  stack
  {
    public:
      int   Data[size];   // 資料
      int   Top;          // 指向資料項端

      void  push(int);    // 放資料進stack
      void  pop( );       // 拿資料出來
      void  list( );

    stack( ) { Top = -1; } // 構造函數
  };
```

屬性

方法

※以上為宣告，若太大，會開另個.h檔儲存

```
void stack::push (int i)
{
    if (Top ≥ (size -1))
    {
        cout << "滿了" << endl;
        return ;
    }
    Top ++ ;
    Data [Top] = i ;
}
```

不符合就直接return
減少{}了數量

取出後，原位不用歸零

```
void stack::pop ( )
{
    if( Top < 0 )
    {
        cout << "空了" ;
        return ;
    }
    cout << Data [Top] ;
    Top -- ;
}
```

實作

```
void main ( )
{
    stack st1;
    st1.push(1);
    st1.push(2);
    st1.list( );
    st1.pop( );
    st1.pop( );
    st1.pop( );
}
```

用 for 迴圈 push 11 次，pop 11 次

```
void main ( )
{
    int i; stack st1;
    for( i = 1 ; i < 12 ; i++ )
        st1.push(i);
    while( --i )
        st1.pop( );
    return ;
}
```

```
Template <class T>
class class_name
{
    T Top;
    Void Push(T);

    Stack(T ini)
    {
        Top = ini;
    }
}
```

Template 為一種
先不宣告 資料型態
待進入 main function 後
再宣告 資料型態的方法

```
Template <class T>
void stack<T> :: Push(T i)
{
    ;
    ;
}
```

class 的 function 寫法

```
int main( )
{
  stack <int> st1(-1);
  stack <char> st2(-1);
    ;
    ;
}
```

在 main 中的宣告物件方法