C++程式設計

單元：列舉、結構及類別

列舉型態指令—enum

- 列舉是由使用者自定的資料型態，內容是由一組常數的名稱及常數值所集合而成，並對這些常數值給予不同的命名，以提高程式之可讀性。
- 列舉的結構可用於類似菜單目錄的場合，利用列舉可以同一變數名稱來代替不同的常數，如此，可簡化程式。

列舉型態的定義與宣告(一)

- 宣告語法格式一：宣告時直接指定變數

```
enum   列舉型態名稱{
           列舉成員1[=值1],
           列舉成員2[=值2],
                  .
           列舉成員N[=值3]
} [列舉型態變數名稱1,列舉型態變數名稱2…];
```

註：上述中中括號表示可省略值的設定，若省略則會依順序由０～Ｎ給予各成員所表示的值；或指定第一筆之數值後其它的會依序遞增。

列舉型態的定義與宣告(二)

- 宣告語法格式一：先列舉，之後再宣告變數名

```
enum   列舉型態名稱{
          列舉成員1[=值1],
          列舉成員2[=值2],
                  .
          列舉成員N[=值3]
};

enum 列舉型態名稱  變數名稱1,變數名稱2……;
```

列舉宣告範例

以下為各飲品的名稱及定價所組成的列舉：

```
enum   teacost{
          redtea=30,
          greentea=20,
          milktea=40

} drink;
```

列舉的應用一(一)

- **選擇飲料及數量，計算應付金額**

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

enum tea{       //定義一名為 tea 的列舉，並設定列舉中每一個成員之值
        redtea=30,
        greentea=20,
        milktea=40
} drink;    //以drink 之變數名引用此列舉


main(){
        int kind,cost,c;
        cout<<"請選擇你要買的飲料：(1)紅茶  (2)綠茶  (3)奶茶：";cin>>kind;
        cout<<"請選輸入您要買的數量：";cin>>c;
```

列舉的應用一(二)

```
    switch(kind){      //依所選擇之飲料種類讀取對應的列舉成員之值
        case 1:
            cost=redtea;     //等於將cost設定為redtea所代表之數值
            break;
        case 2:
            cost=greentea; //等於將cost設定為greentea所代表之數值
            break;
        case 3:
            cost=milktea; //等於將cost設定為milktea所代表之數值
    }
    cout<<"你的消費總金額為："<<cost*c<<endl;   //以讀取的金額計算
    system("pause");

    return 0;
}
```

列舉的應用二(一)

- 輸入月份判斷是季節中的第幾季

```
#include <iostream>
using namespace std;
//定義一成員以季節為名稱之列舉，並以1為啟始設定其值，然後以
season為名讀取
enum F_season{spring=1,summer,autumn,winnter} season;

main(){
        int month;
        cout<<“四季中第”<<spring<<“季是春天”<<endl;  //引用各成員之值
        cout<<"四季中第"<<summer<<"季是夏天"<<endl;
        cout<<"四季中第"<<autumn<<"季是秋天"<<endl;
        cout<<"四季中第"<<winnter<<"季是冬天"<<endl;
        cout<<"請輸入月份(1~12)：";cin>>month;
```

列舉的應用二(二)

```
    switch(month){    //依輸入的月份去讀取列舉值
        case 2 ... 4:
            season=spring;
            break;
        case 5 ... 7:
            season=summer;
            break;
        case 8 ... 10:
            season=autumn;
            break;
        case 11 ... 12:case 1:
            season=winnter;
    }
    cout<<month<<"月是第 "<< season <<" 季\n";
    system("pause");
    return 0;
}
```

結構（struct）

- 一個變數可儲存一個指定的資料型態值；而陣列則是可以儲存多個相同資料型態的資料；但若一組資料中同時包含多種不同的資料型態時，就無法用單一陣列來儲存；如學生成績資料，其中可含姓名（字串）、各科成績（數值）。
- 結構則可解決上述問題，可讓設計師在同一筆資料中可同時包含多種資料型態的值；因此，結構是屬於依需求自行定義的資料型態。

結構定義與宣告(一)

- 宣告語法格式一：宣告時直接指定變數

```
struct   結構型態名稱{
         資料型態  欄位成員名稱1;
         資料型態  欄位成員名稱2;
                  .
         資料型態  欄位成員名稱3;
} 變數名稱1，變數名稱2…;
```

和列舉不同的是，列舉是常數集合；而結構是變數宣告的組合

結構定義與宣告(二)

- 宣告語法格式二：宣告後再以此命名變數

```
struct   結構型態名稱{
         資料型態  欄位成員名稱1;
         資料型態  欄位成員名稱2;
                  .
         資料型態  欄位成員名稱3;
};

struct 已定義之結構型態名稱  變數名稱1，變數名稱2;
```

結構定義與宣告範例(一)

以下為一筆紀錄學生身高及體重結構變數

```
struct  profile{
    string name;
    float height;
    float weight;
} student1,student2;
```

結構定義與宣告範例(二)

宣告時不指定變數，宣告後再指定

```
struct  profile{
    string name;
    float height;
    float weight;
};

struct profile student1,student2;
```

結構定義與宣告範例(三)

宣告後指定初值

```
struct  profile{
    string name;
    float height;
    float weight;
} student={"Wen",180,90};
```

結構定義與宣告範例(四)

以自訂結構宣告陣列，並指定初值

```
struct   profile{
    string name;
    float height;
    float weight;
};
struct profile student[3]={"Wen",180,90,
                           "Lee",175,70,
                           "Kao",175,70};
```

結構型態資料的存取

- 設定結構變數欄位成員內容

  結構變數名稱.欄位成員名稱 = 指定值

- 讀取結構變數欄位成員內容

  變數 = 結構變數名稱.欄位成員名稱

- 同結構之結構變數值互相指定

  結構變數名稱1 = 結構變數名稱2

結構型態資料的存取範例

- 設定範例

```
student.height = 150;
student[1].height = 150;
```

- 讀取範例

```
w = student.height;
w = student[1].height;
```

- 互相指定

```
student1= student2;
```

結構存取範例一(一)

定義一內含姓名、身高、體重之結構型態，然後利用此一結構宣告變數s1、s2。

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

struct profile{        //定義一名為 profile 之結構
        string name;
        float height;
        float weight;
};

struct profile s1,s2;   // 宣告s1、s2為 profile 結構型態
```

結構存取範例一(二)

```
main(){
	cout<<"請輸入學生姓名：";cin>>s1.name;
	cout<<"請輸入學生身高：";cin>>s1.height;
	cout<<"請輸入學生體重：";cin>>s1.weight;

	s2=s1;  //將 s1 的內容指定給 s2

	cout<<"變數名稱  姓名欄位  身高欄位  體重欄位\n";
	cout<<"=====================================\n";
	cout<<setw(8)<<"s1"<<setw(10)<<s1.name<<setw(10)<<s1.height<<setw(
10)<<s1.weight<<endl;
	cout<<setw(8)<<"s2"<<setw(10)<<s2.name<<setw(10)<<s2.height<<setw(
10)<<s2.weight<<endl;
	system("pause");
	return 0;
}
```

結構存取範例二(一)

定義一內含姓名、身高、體重之結構型態，然後利用此一結構宣告陣列變數s並指定初值後印出。

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

struct profile{
        string name;
        float height;
        float weight;
};

struct profile s[3]={"Wen",180,90,
                     "Lee",175,70,
                     "Kao",175,70};
```

結構存取範例一(二)

```
main(){
	int i;

	cout<<"陣列元素  姓名欄位  身高欄位  體重欄位\n";
	cout<<"=====================================\n";

	for(i=0;i<3;i++){
		cout<<setw(6)<<"s["<<i<<"]";
		cout<<setw(10)<<s[i].name;
		cout<<setw(10)<<s[i].height;
		cout<<setw(10)<<s[i].weight<<endl;
	}

	system("pause");
	return 0;
}
```

結構指標的宣告與存取

指標結構之宣告：

```
struct 已定義之結構型態名稱 *p;
```

指標結構之存取：

```
p=&已宣告之結構變數
變數 = p->欄位成員名稱;
p->欄位成員名稱 = 值;
```

結構指標存取範例（一）

定義一內含姓名、身高、體重之結構型態，然後利用此一結構宣告變數s及*p。

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

struct profile{          //定義一名為 profile 之結構
        string name;
        float height;
        float weight;
} s;

struct profile *p=&s;
```

結構指標存取範例（二）

```
main(){
	cout<<"請輸入學生姓名：";cin>>p->name;
	cout<<"請輸入學生身高：";cin>>p->height;
	cout<<"請輸入學生體重：";cin>>p->weight;


	cout<<"姓名欄位  身高欄位  體重欄位\n";
	cout<<"============================\n";
	cout<<setw(8)<<p->name<<setw(10)<<p->height<<setw(10)<<p->weight<<endl;
	system("pause");
	return 0;
}
```

結構變數與函數

型式宣告：

```
void 函數名(struct 已定義結構名稱);      //傳值
void 函數名(struct 已定義結構名稱 *);    //傳址
```

實體宣告：

```
void 函數名(struct 已定義結構名稱 參數名) {
                                  } 傳值
void 函數名(struct 已定義結構名稱 *參數名){
                                  } 傳址
```

結構變數傳值呼叫範例（一）

宣告結構變數內含身高、體重，然後將此一變數傳給BMI計算函數，算出BMI後將其印出。

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

struct BMI_data{        //定義一名為 profile 之結構
        float height;
        float weight;
} info;

float BMI(struct BMI_data);
```

結構變數傳值呼叫範例(二)

```
main(){
	cout<<"請輸入身高(cm)：";cin>>info.height;
	cout<<"請輸入體重(Kg)：";cin>>info.weight;
	cout<<"\n身高為 "<<info.height;
	cout<<"、體重為 "<<info.weight;
	cout<<"、BMI="<<BMI(info)<<endl;
	system("pause");
	return 0;
}

float BMI(struct BMI_data k){
	float result;
	k.height=k.height/100;
	result=k.weight/(k.height*k.height);
	return result;
}
```

結構變數傳址呼叫範例（一）

宣告結構變數內含身高、體重，然後將此一變數傳給BMI計算函數，然後由函數將身高改為公尺為單位，並計算BMI

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

struct BMI_data{        //定義一名為 profile 之結構
        float height;
        float weight;
        float bmi;
} info;

float BMI(struct BMI_data *);
```

結構變數傳址呼叫範例(二)

```
main(){
	cout<<"請輸入身高(cm)：";cin>>info.height;
	cout<<"請輸入體重(Kg)：";cin>>info.weight;
	BMI(&info);
	cout<<"\n身高(M)為 "<<info.height;
	cout<<"、體重(Kg)為 "<<info.weight;
	cout<<"、BMI="<<info.bmi<<endl;
	system("pause");
	return 0;
}

float BMI(struct BMI_data *p){

	p->height=p->height/100;
	p->bmi=p->weight/(p->height*p->height);
}
```

物件導向之路—類別（class）

- 類別是結構型態的衍生，二者的差別在於結構只能包含變數資料；而類別除了變數資料外，亦可包含處理資料的函數。

- 在物件導向程式設計（ＯＯＰ）時，所有的物件是建立在類別上，因此，類別是物件的抽象定義，而物件是類別實體化之後的產生物。例如：客機、運輸機是不同的機種，但都是屬於飛機類，因此，雖然機種不同，但會擁有一些共同的特性。

物件導向之路—類別（class）

- 在類別中所定義的各個變數稱為成員變數(即物件屬性)；相同的，在類別中定義函數時，此函數則稱為成員函數(即物件方法)。

- 類別封裝中，依資料或函數是否開放給外部程式讀取或操作，可分為私有成員(private)、公有成員(public)、保護成員(protected)三種。

類別原型宣告與物件建立

```
class  類別名稱{
        private:
            私有成員(可含變數及函數)
        protected:
            保護成員(可含變數及函數)
        public:
            公用成員(可含變數及函數)
};
```

類別原型宣告說明

- private—私有成員：

  在這個區域的成員，僅能在類別內部使用，外部無法操作它；若在類別中沒有指定是何種成員時，皆會被視為私有成員。

- public—公有成員：

  在這個區域中所宣告的變數或函數，在使用上不受限制（即內外皆可操作），因此，此區塊內的成員是提供給使用者存取的介面。

- protected—保護成員：

  此區域的成員亦無法讓外界使用，而此區域主要是為物件用於繼承時成員存取的一種特殊模式。

物件建立及操作

- 依已定義的類別建立物件：

```
已定義的類別　物件名稱；
```

- 物件中資料成員的存取

```
變數 = 物件名稱.資料成員；
物件名稱.資料成員 = 值；
```

- 物件中函數成員的存取

```
物件名稱.函數成員(參數)；
```

範例

為紀錄學生BMI，定義包含姓名、身高、體重、BMI等資料成員，及可輸入資料、計算BMI及顯示BMI的成員函數的類別。

```
class profile{
   private:
      string name;
      float h,w,bmi;

   public:
       void keyinData(){
          cout<<"輸入姓名：";cin>>name;
          cout<<"輸入身高(cm)：";cin>>h;
          cout<<"輸入體重(Kg)：";cin>>w;
          h=h/100;
          bmi=w/(h*h);
       }
       void showBMI(){
          cout<<name<<"的BMI為：";
          cout<<bmi<<endl;
       }
};
```

此類別中，可供外界存取的有：

keyinData()

showBMI()

兩個成員函數。

實際使用

利用上述之類別產生一名為 student 的元件，並進行操作

```
#include<iostream>
using namespace std;
class Cprofile{
   private:
      string name;
      float h,w,bmi;

   public:
      void keyinData(){
            cout<<"輸入姓名：";cin>>name;
            cout<<"輸入身高(cm)：";cin>>h;
            cout<<"輸入體重(Kg)：";cin>>w;
            h=h/100;
            bmi=w/(h*h);
      }
      void showBMI(){
         cout<<name<<"的BMI為：";
         cout<<bmi<<endl;
      }
};

main(){
      Cprofile student;

      student.keyinData();
      student.showBMI();
      system("pause");
      return 0;
 }
```

將成員函數定義在類別外

類別中的成員函數，除了定義在類別內之外，亦可定義在類別外；但在類別中需先做型式的宣告，方法如下：

```
class  類別名稱{
        private:
              私有成員(可含變數及函數)
         public:
              傳回資料型態   函數名稱（參數）;
};

傳回資料型態   類別名稱::函數名稱（參數）{
            函數內程式
};
```

要一致

修改前述BMI類別範例

```
class Cprofile{
   private:
      string name;
      float h,w,bmi;

   public:
       void keyinData();
       void showBMI();
};
void Cprofile::keyinData(){
          cout<<"輸入姓名：";cin>>name;
          cout<<"輸入身高(cm)：";cin>>h;
          cout<<"輸入體重(Kg)：";cin>>w;
          h=h/100;
          bmi=w/(h*h);
}
```

```
void Cprofile::showBMI(){
          cout<<name<<"的BMI為：";
          cout<<bmi<<endl;
}
```

實際使用

```
#include<iostream>
using namespace std;
class Cprofile{
   private:
      string name;
      float h,w,bmi;
    public:
      void keyinData();
      void showBMI();
};
void Cprofile::keyinData(){
         cout<<"輸入姓名：";cin>>name;
         cout<<"輸入身高(cm)：";cin>>h;
         cout<<"輸入體重(Kg)：";cin>>w;
         h=h/100;
         bmi=w/(h*h);
}
```

```
void Cprofile::showBMI(){
      cout<<name<<"的BMI為：";
      cout<<bmi<<endl;
}

main(){
      Cprofile student;

      student.keyinData();
      student.showBMI();
      system("pause");
      return 0;
 }
```

建構函數—類別成員之初始化

建構函數是指在物件生成時，會自動執行的函數；所以我們可以將所有對該類別物件的初始化設定置於其中，C++ 中規定，初始化函數的名稱必需要和類別名稱相同，且不需要傳回值，如:

```
class   類別名稱{
                .
  public:
    類別名稱([參數資料型態]);
                .
};
```

```
類別名稱::類別名稱([參數]){
        初始化程式內容
}
```

建構函數範例—無參數型

```
#include<iostream>
using namespace std;
class Cprofile{
   private:
      string name;
      float h,w,bmi;
    public:
      Cprofile();
      void showBMI();
};
Cprofile::Cprofile(){
      name= "王大牛";
      h=200;
      w=90;
      bmi=w/(h*h/10000);
}
```

```
void Cprofile::showBMI(){
      cout<<endl;
      cout<<name<<" 的身高為：";
      cout<<h<<endl;
      cout<<name<<" 的體重為：";
      cout<<w<<endl;
      cout<<name<<" 的BMI為：";
      cout<<bmi<<endl;
}
main(){
      Cprofile student;

      student.showBMI();
      system("pause");
      return 0;
 }
```

建構函數範例—有參數型

```
#include<iostream>
using namespace std;
class Cprofile{
   private:
      string name;
      float h,w,bmi;
    public:
      Cprofile(string,float,float);
      void showBMI();
};
Cprofile::Cprofile(string n, float h1, float w1){
      name= n;
      h=h1;
      w=w1;
      bmi=w/(h*h/10000);
}
```

```
void Cprofile::showBMI(){
      cout<<endl;
      cout<<name<<" 的身高為：";
      cout<<h<<endl;
      cout<<name<<" 的體重為：";
      cout<<w<<endl;
      cout<<name<<" 的BMI為：";
      cout<<bmi<<endl;
}
main(){
      Cprofile student("小花" ,150,45);

      student.showBMI();
      system("pause");
      return 0;
 }
```

解構函數—將物件所占資源釋放

解構函數和建構函數的功能正好相反，解構函數的功能為當物件變數消失時，才會執行此一函數；該函數被執行時，系統會徹底的將原本物件所占用的資源皆釋放掉。

```
class   類別名稱{
                .
  public:
    ~類別名稱();
                .
};
```

```
類別名稱::~類別名稱(){
        程式敘述;
}
```

解構函數範例

```
#include<iostream>
using namespace std;
class Cprofile{
   private:
      string name;
      float h,w,bmi;
    public:
      Cprofile(string,float,float);
      ~Cprofile();
      void showBMI();
};
Cprofile::Cprofile(string n,float h1,float w1){
      name= n;
      h=h1;w=w1;
      bmi=w/(h*h/10000);
      cout<<"物件初始化完成 \n";
}
```

```
Cprofile::~Cprofile(){
      cout<<"物件資源已釋放\n";
}
void Cprofile::showBMI(){
      cout<<name<<" 的BMI為：";
      cout<<bmi<<endl;
}
main(){
      Cprofile student("小花" ,150,45);

      student.showBMI();
      system("pause");
      return 0;
 }
```