國立屏東大學 資訊工程學系 C++程式設計入門教材

13. 使用者自定資料型態

程式設計的目的就是為了解決或滿足特定應用領域的問題，其中又以資料處理為最常見的需求。所以我們在設計程式時，往往需要宣告很多變數來代表真實（或抽象、虛擬）的世界中的人、事、時、地、物。以一個特定的應用題目來說，許多的變數間其實是具有相關性的，因此本章將以使用者自定資料型態（user-defined data type）來定義更為符合真實應用需求的複合資料型態（composite data type）。例如我們可以定義一個學生的型態，其中包含有學生的學號、姓名、班級、成績等資訊，或是一個產品的型態，其中包含產名代碼、名稱、單價與規格等資訊。本章將就此種複合資料型態的定義、變數宣告與操作等議題加以說明。

13.1 結構體

在進行程式設計時，我們可以將相關的資料項目集合起來，定義為一個結構體(Structure)。從程式的角度來看，結構體是一個包含有多個變數的資料集合，可做為使用者自定的資料型態，並用以宣告變數(稱之為結構體變數（Structure Variable）進行相關的處理與操作。

13.1.1 結構體變數

結構體變數(Structure Variable)的宣告，是以struct保留字進行相關的定義 — 可包含一個或一個以上的欄位(Field)（也就是相關的資料項目，又稱為資料成員Data Member)的定義，每個欄位其實就是一個變數的宣告，但是不可以包含初始值的給定，其語法如下：

結構體變數宣告語法定義

struct
{
  [資料型態 欄位名稱;]$^{+}$
}結構體變數名稱 [,結構體變數名稱]*;

例如，下面的程式碼片段定義了兩個平面上的點，每個點包含有x與y軸座標：

```
struct
{
   int x;
   int y;
} p1, p2;
```

當然，你也可以這樣寫：

```
struct
{
   int x,y;
} p1, p2;
```

在上述程式碼片段中，p1與p2為所宣告的結構體的變數，我們可以p1.x，p1.y，p2.x與p2.y來存取這些結構體中的欄位。下面提供另一個例子：

```
struct
{
   int  productNo;
   char *productName;
   float price;
   int   quantity;
} mfone;
```

這個例子宣告了一個名為mfone的結構體變數，其中包含有產品編號、名稱、單價與庫存數量。現在讓我們來看看在記憶體中的空間配置，如figure 1。

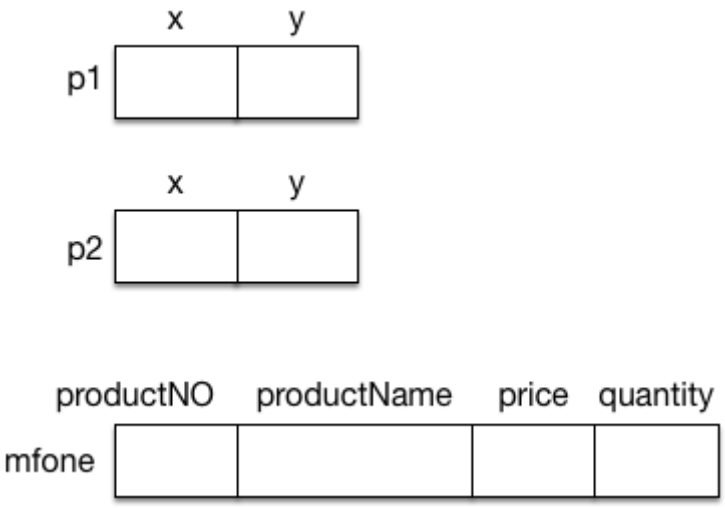

Fig. 1

現在請同學動動腦，想想看在前述例子中的p1, p2與mfone各佔用了多少的記憶體空間呢？

結構體所佔用的記憶體空間 在大部份的32位元與64位元的系統上的C語言編譯器的實作上，在結構體的空間配置上分別是以4與8個位元組的倍數為基礎。以64位元為例，在空間配置時，每次會分配8個位元組，逐一分配給結構體的欄位，當剩餘空間不足時，剩餘空間將會被閒置，然後再配置新的8個位元組給後續的欄位。

關於結構體的變數其初始值可以在結構變數宣告時以「={…}」方式，依欄位的順序進行給定，請參考下面的例子：

```
struct
{
   int x;
   int y;
} p1 = {0,0},
  p2 = {100, 100};
struct
{
   int  productNo;
   char *productName;
   float price;
   int   quantity;
} mfone = {12, "mPhone 6s", 15000, 100};
```

讓我們也可以在宣告其初始值的同時，使用「指定初始子(Designated Initializer)」來給定欄位的名稱，就可以不受原本順序的限制，請參考下例:

```
struct
{
   int x;
   int y;
} p1 = {.x=0, .y=0},
  p2 = {.y=100, .x=100};
struct
{
   int  productNo;
   char *productName;
   float price;
   int   quantity;
} mfone = {.productName="mPhone 6s", .price=15000, .quantity=100};
//productNo並沒給定初始值
```

從C++11開始，還支援了「匿名結構體(Anonymous Structure)」可以在某些情況下省略結構體的名稱。例如：

```
struct point
{
   int x, y;
};


struct         // 這個宣告是錯誤的，因為沒有定義結構體的名稱
{     // 或是使用 -std=gnu++11參數，因為<nowiki>C++</nowiki>11已支援
   int x, y;
} s1, s2;
```

```
int main()
{
   struct point p1;
   point p2;          // <nowiki>C++</nowiki>允許不用寫struct
   point p3 = {5,5};
   point p4 {10,10}; // <nowiki>C++</nowiki>11 允許不用寫=，但編譯時記得要加 -
std=gnu++11 參數
   struct { int i, j; } t1; //在區域內允許定義匿名的結構體
   ...
}
```

13.1.2 結構體變數的操作

結構體變數的操作相當簡單，我們可以使用「結構體變數名稱.欄位名稱」的方式來存取其欄位，其中 . 被稱為「直接成員選取運算子(Direct Member Selection Operator)— 唸做dot點）」，不過筆者更建議你將它唸做 “ 的 ”，表示你要存取的是結構體變數 “ 的 ” 欄位，例如我們可以使用p1.x或p2.y來分別存取p1與p2結構體變數裡的x與y欄位，可唸做“p1的x”與 “ p2的y”要特別注意的是，直接成員選取運算子是一個二元運算子，其運算元就是由其左右兩側的結構體變數以及欄位名稱；其運算結果就是幫我們傳回在結構體變數內的特定欄位之數值。具體的使用範例，可參考以下的程式碼片段：

```
struct
{
   int x;
   int y;
} p1 = {0,0},
  p2 = {100, 100};
p1.x = 100;

float z;
z = sqrt(p1.x*p1.x + p1.y*p1.y);

p2.x+=5;

cin >> p1.x;
```

但結構體變數最方便的地方在於可以使用賦值運算子(也就是 😀)，讓兩個結構體變數可以互相給定其值，包含了其中所有的欄位，甚至也包含了字串的值。請參考下面的程式：

```
struct
{
   int  productNo;
   char *productName;
   float price;
   int   quantity;
} mfone = {12, "mPhone 6s", 15000, 100},
  mfone2;
```

```
mfone2=mfone;
```

13.1.3 結構體型態

我們也可以單獨地定義結構體，而不用像前述的例子都是定義結構體的同時，還要“ 順便 ”進行其變數的宣告。這樣做有很多的好處，尤其是當程式碼需要共享時，我們可以將結構體的定義獨立於某個檔案，爾後其它程式可以直接載入該檔案來得到相關的定義，其語法如下：

結構體定義語法

struct 結構體名稱
{
  [資料型態 欄位名稱;]+
};

有了先宣告好的結構體之後，就可以在需要時以「struct 結構體名稱」或「結構體名稱」做為型態，以進行變數的宣告。請參考下面的程式：

```
struct point
{
   int x;
   int y;
};
struct point p1, p2; // 以struct point做為型態
point p3 ={6,6};     // 以point做為型態
```

當然，你也可以這樣寫：

```
struct point
{
   int x,y;
} p1, p2;

point p3;
```

儘管我們可以在C++語言裡，直接將結構體的名稱做為資料型態，但基於語法的相容性(與C語言的語法兼容)，仍然有些人習慣“ 明確地 ”進行型態定義，其語法如下：

結構體型態定義語法1

struct 結構體名稱
{
  [資料型態 欄位名稱;]+
};
typedef [struct]? 結構體名稱 型態名稱;

結構體型態定義語法2

typedef struct
{
  [資料型態 欄位名稱;]+
} 型態名稱;

如此一來，我們就可以利用這個新的資料型態來宣告變數，請參考下面的程式：

```
struct point
{
   int x,y;
};

typedef struct point Point;
```

或是

```
typedef struct
{
   int x,y;
} Point;
```

接著你就可以在需要的時候，以Point做為資料型態的名稱來進行變數的宣告，例如：

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

struct point
{
  int x,y;
};

typedef struct point Point;
```

```
int main()
{
  Point p1;
  Point p2 = {5,5};

  p1=p2;
  p1.x+=p1.y+=10;

  cout << "p1=(" << p1.x << "," << p1.y << ") "
       << "p2=(" << p2.x << "," << p2.y << ")" << endl;

  return 0;
}
```

13.1.4 結構體與函式

在結構體與函式方面，我們將先探討以結構體變數做為引數的方法，請參考下面的例子：

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

struct point
{
  int x,y;
};

void showPoint(point p)
{
  cout << "(" << p.x << "," << p.y << ")" << endl;
}

int main()
{
  point p1;
  point p2 = {5,5};

  p1=p2;
  p1.x+=p1.y+=10;

  showPoint(p1);
  showPoint(p2);

  return 0;
}
```

請看下一個程式，我們設計另一個函式，讓我們修改所傳入的point的值：

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

struct point
{
  int x,y;
};

void resetPoint(point p)
{
  p.x=p.y=0;
}

void showPoint(point p)
{
  cout << "(" << p.x << "," << p.y << ")" << endl;
}

int main()
{
  point p1;
  point p2 = {5,5};

  p1=p2;
  p1.x+=p1.y+=10;

  showPoint(p1);
  showPoint(p2);
  resetPoint(p1);
  showPoint(p1);
  return 0;
}
```

請先猜猜看其執行結果為何？然後再實際執行看看其結果為何？

再換成下面這個改用「傳址呼叫(Call By Address)」的程式試試看：

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

struct point
{
  int x,y;
};
```

```
void resetPoint(point *p)
{
  p->x=p->y=0;
}

void showPoint(point p)
{
  cout << "(" << p.x << "," << p.y << ")" << endl;
}

int main()
{
  point p1;
  point p2 = {5,5};

  p1=p2;
  p1.x+=p1.y+=10;

  showPoint(p1);
  showPoint(p2);
  resetPoint(&p1);
  showPoint(p1);
  return 0;
}
```

在此要提醒讀者，若是以指標來操作結構體，在存取其欄位時，必須改用 -> 運算子才行。 當我們在呼叫某個函式時，也可以直接產生一個新的結構體做為引數，例如：

```
showPoint( (Point) {5,6}  );
```

我們也可以讓結構體做為函式的傳回值，請參考下面的例子：

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

struct point
{
  int x,y;
};

point addPoints(point p1, point p2)
{
  point p;
  p.x = p1.x + p2.x;
  p.y = p1.y + p2.y;
```

```
  return p;
}

void showPoint(point p)
{
  cout << "(" << p.x << "," << p.y << ")" << endl;
}

int main()
{
  point p1;
  point p2 = {5,5};
  point p3;

  p1=p2;
  p1.x+=p1.y+=10;

  showPoint(p1);
  showPoint(p2);

  p3=addPoints(p1,p2);
  showPoint(p3);
  return 0;
}
```

現在，讓我們看看下面這個程式：

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

struct point
{
  int x,y;
};

point * addPoint(point *p1, point p2)
{
  p1->x = p1->x + p2.x;
  p1->y = p1->y + p2.y;
  return p1;
}

void showPoint(point p)
{
  cout << "(" << p.x << "," << p.y << ")" << endl;
}

int main()
{
```

```
  point *p1;
  point p2 = {5,5};
  point *p3;

  p1=&p2;
  p1->x += p1->y+=10;

  showPoint(*p1);
  showPoint(p2);

  p3=addPoint(p1,p2);
  showPoint(*p1);
  showPoint(*p3);
  return 0;
}
```

13.1.5 巢狀式結構體

我們也可以在一個結構體內含有另一個結構體做為其欄位，請參考下面的程式：

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

#include <stdio.h>

struct Name { char firstname[20], lastname[20]; };

struct Contact
{
  Name name;
  int phone;
};

void showName(Name n)
{
  cout << n.lastname << ", " << n.firstname << endl;
}

void showContact(Contact c)
{
  showName(c.name);
  cout << c.phone << endl;
}

int main()
```

```
{
  Contact someone={.phone=12345, .name={.firstname="Jun", .lastname="Wu"}};

  showContact(someone);
  return 0;
}
```

13.1.6 結構體陣列

我們也可以利用結構體來宣告陣列，請參考下面的片段：

```
Point a={3,5};
Point twoPoints[2];
Point points[10] = { {0,0}, {1,1}, {2,2}, {3,3}, {4,4}, {5,5}, {6,6}, {7,7},
{8,8}, {9,9} };

twoPoints[0].x=5;
twoPoints[0].y=6;
twoPoints[1] = a;
```

13.2 共有體

共有體(Union)與結構體非常相像，但共有體在記憶體內所保有的空間僅能存放一個欄位的資料，適用於某種資料可能有兩種以上不同型態的情況，例如：一個在程式中會使用到的數值資料，在某些應用時是整數，但在另一些應用時可能會是浮點數。在這種情況下，我們可以宣告一個union來解決此問題：

```
union
{
   int i;
   double d;
} data;
```

當我們需要它是整數時，就使用其i的欄位；若需要它是浮點數時，就使用其d的欄位，例如：

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

union
{
  int i;
  double d;
```

```
} data;


int main()
{
  data.i=5;
  cout << data.i << endl;
  cout << data.d << endl;

  data.d=10.5;
  cout << data.i << endl;
  cout << data.d << endl;

  return 0;
}
```

請執行看看這個程式，想想看為何會有這樣的執行結果，也想想看這樣的工具可以用在哪？

當然，前面在結構體時可以應用的宣告方式，定義方式等都可以適用在union上，例如下面的程式碼：

```
union
{
   int i;
   double d;
} data = {0} // 只有第一個欄位可以有初始值
```

```
union
{
   int i;
   double d;
} data = {.d=3.14} // 可以指定其它的欄位做為初始值
```

```
union num
{
   int i;
   double d;
};

typedef union num Num;
```

```
typedef union
{
   int i;
   double d;
} Num;
```

13.3 列舉

當我們在宣告變數時，最重要的事情是必須為變數選擇適合的資料型態，但除了在整數、浮點數、字元這些大範圍的資料型態裡去做選擇，有時候某些變數僅有非常有限的數值可能性，甚至是五根手指數得完的範圍 — 這種時候，你需要的是將所有可能的數值列舉出來，並把它們變為一個型態。例如:

- 一個代表撲克牌花色的變數，其可能的數值為Spade、Heart、Diamond與Club
- 一個代表服裝尺碼的變數，其可能的數值為XXL、XL、L、M、S與XS
- 一個代表血型的變數，其可能的數值為A型、B型、AB型與O型
- 一個代表成績等第的變數，其可能的數值為A++、A+、A、A-、B+、B、B-、C與F
- 一個代表訂單狀態的變數，其可能的數值為訂單成立、撿貨、理貨、出貨、已送達

針對上述這些需求，C++語言提供enum保留字讓我們明確的將可能的數值列舉出來，並且將其命名為一個新的型態，然後就可以用來宣告所需的變數了。請參考以下的語法：

列舉變數宣告（Enumeration Variable Declaration）語法之一

enum
{
  數值[,數值]*;
} 列舉變數名稱 [,列舉變數名稱]*;

我們可以用以下的宣告，來限制變數s1與s2可能的數值：

```
enum { SPADE, HEART, DIAMOND, CLUB } s1, s2;
```

也可以把enum定義好，然後將「enum 列舉名稱」或直接使用「列舉名稱」做為型態進行變數宣告:

列舉變數宣告（Enumeration Variable Declaration）語法之一

enum 列舉名稱
{
  數值[,數值]*;
}

[enum]? 列舉名稱 列舉變數名稱[,列舉變數名稱]*;

以下是一些宣告的例子：

```
enum suit { SPADE, HEART, DIAMOND, CLUB };
enum suit s1, s2;  // 使用enum suit做為型態
suit s3, s4;       // 直接將suit視為型態
```

當然，如同結構體與共有體，同樣也可以使用過去C語言所慣用的typedef將列舉定義為一個型態(C++語言不需要這樣做，可以直接使用列舉名稱做為型態名稱)，例如：

```
typedef enum { SPADE, HEART, DIAMOND, CLUB } Suit;
Suit s1,s2;
```

其實enum的實作是把列舉的數值視為整數，第一個數值視為0，第二個為1，依此類推。不過，我們也可以自行定義其數值：

```
enum suit { SPADE=1, HEART=13, DIAMOND=26, CLUB=39 };
```

13.4 綜合演練

請參考下面這個比較完整的例子：

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

typedef enum {BOOK, KEYCHAIN, T_SHIRT} Type;
typedef enum {red, gold, silver, black, blue, brown} Color;
char colorName[][10]={"red", "gold", "silver", "black", "blue", "brown"};
typedef enum {XS, S, M, L, XL, XXL, XXXL} Size;
char sizeName[][5]={"XS", "S", "M", "L", "XL", "XXL", "XXXL"};

typedef struct {
  int stock_number;
  float price;
  Type type;
  union
```

```
  {
    char author[20];
    Color color;
    Size size;
  } attribute;
} Product;

void showInfo(Product p)
{
  cout << "Stock Number: " << p.stock_number << endl;
  cout << "Price: " << fixed << setprecision(2) << p.price << endl;
  switch(p.type)
  {
    case BOOK:
      cout << "Author: " << p.attribute.author << endl;
      break;
    case KEYCHAIN:
      cout << "Color: " << colorName[p.attribute.color] << endl;
      break;
    case T_SHIRT:
      cout << "Size: " << sizeName[p.attribute.size] << endl;
      break;
  }
  cout << endl;
}

int main()
{
  Product p[3];

  p[0].stock_number=10;
  p[0].price = 280.0;
  p[0].type = BOOK;
  strcpy(p[0].attribute.author, "Antoine");

  p[1].stock_number=20;
  p[1].price = 55.0;
  p[1].type = KEYCHAIN;
  p[1].attribute.color = gold;

  p[2].stock_number=23;
  p[2].price = 350.0;
  p[2].type = T_SHIRT;
  p[2].attribute.size = L;

  showInfo(p[0]);
  showInfo(p[1]);
  showInfo(p[2]);

  return 0;
}
```