國立屏東大學 資訊工程學系 程式設計(二)

15 使用者自定資料型態

15.1 Structures

我們可以將相關的資料項目集合起來，定義為一個Structure(結構體)。從程式的角度來看，所謂的結構體是一個包含有多個變數的資料集合，可用以宣告變數－稱為結構體變數，進行相關的程式處理。

15.1.1 結構體變數

結構體變數的宣告在語法上以struct{ ... } structVarName;定義一個結構體，其中可包含一個或一個以上的欄位(field)定義。每個欄位定義其實就是一個變數的宣告，不可包含初始值的給定，其語法結構如下：

```
struct
{
   [type fieldName;]+
} structVarName[,structVarName]*;
```

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.xp1.yp2.x與p2.y來存取這些結構體中的欄位(變數)。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

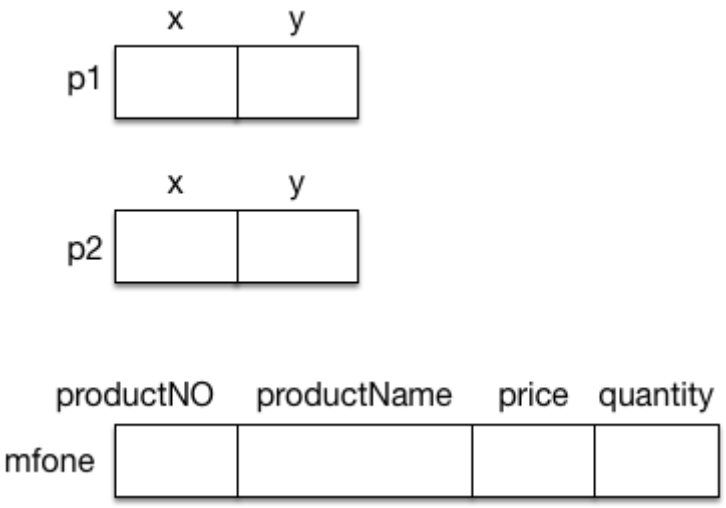

Fig. 1

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。

<這邊要進行一些討論...structdef.c>

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 = {,},
  p2 = {100, 100};

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

從C99開始，還支援了指定初始子(designated initializers)的方法，讓我們可以在宣告其初始值的同時，給定欄位的名稱，且不受原本順序的限制，請參考下例:

```
struct
```

```
{
   int x;
   int y;
} p1 = {.x=, .y=},
  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
```

15.1.2 結構體變數的操作

結構體變數的操作相當簡單，你可以「結構體變數名稱.欄位名稱」來存取其值，例如：

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

p1.x = 100;

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

p2.x+=5;

scanf("%d", &p1.x);
```

但結構體變數最方便的地方在於可以使用「=」運算子，讓兩個結構體間可以互相給定其值，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;
```

15.1.3 結構體型態

我們也可以將單獨地宣告結構體，而不用同時將其變數加以宣告。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宣告獨立於某個檔案，爾後其它程式可以直接以該定義來宣告其變數，其語法如下：

```
struct structName
{
   [type fieldName;]+
};
```

有了先宣告好的結構體之後，就可以在需要時再宣告之變數，請參考下面的程式：

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


struct point p1, p2;
struct point p3 ={6,6};
```

當然，你也可以這樣寫：

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

struct point p3;
```

另一種方式，則是直接將結構體定義為一種新的資料型態，其語法如下：

```
struct structName
{
   [type fieldName;]+
};

typedef struct structName typeName;

typedef struct
{
   [type fieldName;]+
} typeName;
```

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 <stdio.h>

struct point
{
  int x,y;
};

typedef struct point Point;

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

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

  printf("p1=(%d,%d) p2=(%d,%d)\n", p1.x, p1.y, p2.x, p2.y);

  return ;
}
```

15.1.4 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：

funarg.c

```
#include <stdio.h>

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

```
typedef struct point Point;

void showPoint(Point p)
{
  printf("(%d,%d)\n", p.x, p.y);
}

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

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

  showPoint(p1);
  showPoint(p2);

  return ;
}
```

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：

funargModify.c

```
#include <stdio.h>

struct point
{
  int x,y;
};

typedef struct point Point;

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

void showPoint(Point p)
{
  printf("(%d,%d)\n", p.x, p.y);
}

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

  p1=p2;
```

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

  showPoint(p1);
  showPoint(p2);

  resetPoint(p1);
  showPoint(p1);

  return ;
}
```

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？

<在這裡要討論一下 … >

funargPointer.c

```
#include <stdio.h>

struct point
{
  int x,y;
};

typedef struct point Point;

void resetPoint(Point *p)
{
  p->x=p->y=;
}

void showPoint(Point p)
{
  printf("(%d,%d)\n", p.x, p.y);
}

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

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

  showPoint(p1);
  showPoint(p2);

  resetPoint(&p1);
  showPoint(p1);
```

```
  return ;
}
```

讓我們做一個總結：若採用call-by-reference的方式，將變數所在的記憶體位址傳入函式，那麼在函式中，做為一個指向記憶體中某個結構體的指標，其要存取結構體內的欄位時，須以「→」運算子為之。

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：

funreturn.c

```
#include <stdio.h>
#include <math.h>

struct point
{
  int x,y;
};

typedef struct point Point;

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)
{
  printf("(%d,%d)\n", p.x, p.y);
}

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 ;
}
```

現在，讓我們看看下面這個程式：

funreturnPointer.c

```
#include <stdio.h>

struct point
{
  int x,y;
};

typedef struct point Point;

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

void showPoint(Point p)
{
  printf("(%d,%d)\n", p.x, p.y);
}

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 ;
}
```

15.1.5 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：

contact.c

```
#include <stdio.h>

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

typedef struct
{
  Name name;
  int phone;
} Contact;

void showName(Name n)
{
  printf("%s, %s", n.lastname, n.firstname);
}

void showContact(Contact c)
{
  showName(c.name);
  printf(" %d\n", c.phone);
}

int main()
{
  Contact someone={.phone=12345, .name={.firstname="Jun",
.lastname="Wu"}};

  showContact(someone);
  return ;
}
```

15.1.6 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] = { {,}, {1,1}, {2,2}, {3,3}, {4,4}, {5,5}, {6,6}, {7,7},
{8,8}, {9,9} };

twoPoints[].x=5;
```

```
twoPoints[].y=6;
twoPoints[1] = a;
```

15.2 Unions

Union與結構體非常相像，但Union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：

union.c

```
#include <stdio.h>

union
{
  int i;
  double d;
} data;


int main()
{
  data.i=5;
  printf("%d %f\n", data.i, data.d );

  data.d=10.5;
  printf("%d %f\n", data.i, data.d );

  return ;
}
```

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 = {} //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;
```

15.3 Enumerations

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，然後再宣告變數:

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

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 };
```

15.4 綜合演練

請參考下面這個比較完整的例子：

deadline.c

```
#include <stdio.h>

typedef enum { Sunday=, Monday=1, Tuesday=2, Wednesday=3, Thursday=4,
               Friday=5, Saturday=6, Undef=-1 } Weekday;

char weekdayString[][10] ={"Sunday", "Monday", "Tuesday", "Wednesday",
                           "Thursday", "Friday", "Saturday" };

Weekday getWeekday(char day[])
{
  Weekday i=Sunday;

  for(i=Sunday;i<=Saturday;i++)
  {
    if(strcmp(day, weekdayString[i])==)
      return i;
  }
  return Undef;
}

int main()
{
  Weekday startDay, endDay;
  char today[10];
  int days;

  printf("What day (of the week) is it today? ");
  scanf(" %s", today);

  if((startDay = getWeekday(today))!=Undef)
  {
    printf("How many days after today? ");
    scanf(" %d", &days);
    endDay = (startDay+days)%7;
    printf("%d days later is %s.\n", days, weekdayString[endDay]);
    if(endDay==Sunday) // how about if(endDay==0)?
      printf("The end day is Sunday. We extend it to Monday!\n");
  }
  else
    printf("unknown weekday");
```

```
  return ;
}
```

union也常和結構體一起使用，用以定義更有彈性的結構體，請參考下例：

product.c

```
#include <stdio.h>
#include <string.h>

typedef enum {BOOK, KEYCHAIN, T_SHIRT} Type;
enum color {red, gold, silver, black, blue, brown};
enum _size {XS, S, M, L, XL, XXL, XXXL};
typedef enum _size Size;

typedef struct {
  int stock_number;
  float price;
  Type type;
  union
  {
    char author[20];
    enum color color;
    Size size;
  } attribute;
} Product;

int main()
{
  Product littlePrince;
  Product bubbleBear;
  Product transformer;

  littlePrince.stock_number=10;
  littlePrince.price = 280.0;
  littlePrince.type = BOOK;
  strcpy(littlePrince.attribute.author, "Antoine");

  bubbleBear.stock_number=20;
  bubbleBear.price = 55.0;
  bubbleBear.type = KEYCHAIN;
  bubbleBear.attribute.color = gold;

  transformer.stock_number=23;
  transformer.price = 350.0;
  transformer.type = T_SHIRT;
  transformer.attribute.size = L;
```

```
  return ;
}
```

下面這個例子，應用到了本章所介紹的相關方法：

product2.c

```
#include <stdio.h>
#include <string.h>

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)
{
  printf("Stock Number: %d\n", p.stock_number);
  printf("Price: %.2f\n", p.price);
  switch(p.type)
  {
    case BOOK:
      printf("Author: %s\n", p.attribute.author );
      break;
    case KEYCHAIN:
      printf("Color: %s\n", colorName[p.attribute.color]);
      break;
    case T_SHIRT:
      printf("Size: %s\n", sizeName[p.attribute.size]);
      break;
  }
  printf("\n");
}

int main()
{
```

```
  Product p[3];

  p[].stock_number=10;
  p[].price = 280.0;
  p[].type = BOOK;
  strcpy(p[].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[]);
  showInfo(p[1]);
  showInfo(p[2]);

  return ;
}
```