國立屏東商業技術學院 資訊工程系 程式設計(二)

17. 動態記憶體管理

不特別說明自動, 靜態….等

直接說明 變數的範圍與生命週期

本章將就C++的記憶體管理，進行一個簡要的介紹：

在程式中的資料C++提供四種不同的記憶體管理方法：

- 自動儲存(automatic storage)
- 靜態儲存(static storage)
- 動態儲存(dynamic storage)
- 緒儲存(thread storage)

17.1 自動儲存

通常，在函式內所宣告的變數就是用自動儲存的方式，並稱為自動變數(automatic variables)下面的程式碼中，變數a, b與temp皆屬之，它們只在foo函式的「{ … }」範圍內存在，一但函式執行結束並返回時，這些自動變數就不在存在。

```
int foo(int a, int b)
{
   int temp;
   temp = 2*a+3*b;
   return temp;
}
```

在實作上，自動變數是存放在程式所配置到的記憶體中，以Stack的方式管理，變數依序放入stack中，並以相反的順序從stack中移除。自動變數的生命週期限於其所位於的程式區塊中，例如上述程式foo函式由「{」開始至「}」結束的範圍就稱為一個程式區塊(block)。我們也可以在程式中視需要建立巢狀的程式區塊，例如：

```
int main()
{
   int i,j;

   scanf(" %d", &i);
   scanf(" %d", &j);

   {
      int x;
```

```
        x=2*i;
        if(j>x)
            j=x;
    }
}
```

在這個例子中，main()函式因為某些原因，暫時地需要一個變數x以進行相關的計算，因此，我們將需要x的地方標記為一個程式區塊，這樣一來，在這個區塊之外，就不存在x這個變數。若沒有了區塊，變數x將持續存在於記憶體內，直到main()函式結束為止。

17.2 靜態儲存

用static的方式修飾變數的宣告，將可使其生命週期橫跨整個程式的範圍，例如：

```
static int counter = 0;
```

17.3 動態儲存

動態儲存是以new來配置記憶體空間，並以delete來將空間刪除(事實上，記憶體空間不會被刪除的，只會被釋放並回收)，往往是搭配指標使用。

17.3.1 指標與動態記憶體管理

通常，我們都是用類似下面的程式碼的方式，在使用指標：

```
int *p;
int x;

p=&x;
```

在C++中，提供一個新的方式：

```
int *p;

p = new int;

或是

int *p = new int;
```

用這種方式，直接取得在記憶體中的一塊位置，而不需要另外宣告一個整數。同樣的，當你不再需要這塊位置時，可以delete將其刪除(意即，還給系統)。

```
int *p = new int;

delete p;
```

17.3.2 動態陣列

我們可以透過new與delete來動態地建立與刪除(或者說回收)陣列。請參考下面的例子：

```
int *p = new int [10]; //動態配置產生一塊存放10個整數的陣列空間, 並讓p指向它。

p[0]=3;
p[1]=2;

delete [] p;              //不再使用時將其刪除
```

17.3.3 動態結構體

我們也可以透過new與delete來動態地建立與刪除(或者說回收)結構體。請參考下面的例子：

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

Point *p = new Point;
p->x=5;
(*p).y=6;

delete p;
```

或是

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

point *p = new point;
p->x=5;
(*p).y=6;

delete p;
```