第九章

變數等級

前言

- C語言的變數依據視野與生命週期可以區分為5種等級，在本章中我們將做詳細的討論。

- 程式設計師必須徹底了解變數以便正確地使用變數完成各種需求。

大綱

- 9.1 變數等級
  - 9.1.1　依照宣告位置判斷變數等級
  - 9.1.2　明確宣告變數等級
- 9.2 自訂程式區段
- 9.3 區域變數
  - 9.3.1　auto變數等級
  - 9.3.2　static 區域變數等級
  - 9.3.3　register變數等級

大綱

- 9.4 全域變數與外在變數
  - 9.4.1 external變數等級
  - 9.4.2 static external變數等級
  - 9.4.3 4種變數等級的區
- 9.5 本章回顧

9.1 變數等級

- C語言提供了5 種變數儲存等級。
- 5種變數等級分別是auto、static auto、extern、static extern、register等。
- 若由視野（scope）與生命週期（lifetime）來分類，則可以將變數分為全域變數與區域變數，分述如下：
  - 全域變數：
    - 全域變數宣告於所有函式之外，程式中所有的函式都可以使用該變數（但必須考慮相對位置）。
    - 全域變數的生命週期與程式的執行時間相同，必須等到程式結束執行，全域變數佔用的記憶體才會被釋放。
  - 區域變數：
    - 區域變數宣告於區段（例如函式）內部，只有定義該變數的區段可以使用這個變數，若非宣告為static auto，則該區段執行完畢，生命週期也宣告結束。

9.1.1 依照宣告位置判斷變數等級

- 判斷全域變數與區域變數的最簡單方式，可以從宣告變數的位置來加以識別。
- 全域變數定義於所有函式之外，而區域變數則定義於函式內部，如下範例。
- 【範例】：

```
 int a;
void func1(void)
{
    int b;
}
void main(void)
{
int c;
}
```

9.1.1 依照宣告位置判斷變數等級

- 說明：
  - a是一個全域變數，b是一個區域變數，c是一個區域變數。各函式可使用的變數如下表格。

	變數a	變數b	變數c
func1()	可使用	可使用	不可使用
main()	可使用	不可使用	可使用

9.1.1依照宣告位置判斷變數等級

- 【觀念範例9-1】全域變數與區域變數差別。
- 範例9-1：ch9_01.c

```
1  /*****************************
2       檔名:ch9_01.c
3       功能:全域變數與區域變數
4   *****************************/
5   #include <stdio.h>
6  #include <stdlib.h>
7  int a=10;
8  void func1(void)
9  {
10   int b=5;
11   a=a+1;
12   b=b+1;
13   printf("b=%d\n",b);
14 }
```

```
15
16 void main(void)
17 {
18   int c=20;
19    a=a+1;
20   /* b=b+1; */   /*   這是錯的敘述   */
21   c=c+1;
22   printf("a=%d\n",a);
23   func1();
24   printf("a=%d\n",a);
25   printf("c=%d\n",c);
26   /*   system("pause");   */
   }
```

9.1.1依照宣告位置判斷變數等級

- 執行結果：

```
a=11
b=6
a=12
c=21
```

- 範例說明：
  - 宣告全域變數a，它可以被任何函式存取。
  - 宣告func1()函式的區域變數b，它只能被func1()函式內的敘述存取。
  - 宣告main()函式的區域變數c，它只能被main()函式內的敘述存取。
  - 由執行結果中，可以得知第11行與第18行都會改變a的變數值，因為它是一個全域變數。

9.1.1依照宣告位置判斷變數等級

- 全域變數的宣告位置
  - 在前面我們曾經提過，全域變數宣告於所有函式之外，程式中所有的函式都可以使用該變數。
  - 但我們仍須考慮宣告全域變數及使用全域變數之敘述的相對位置。
    - 就如同函式宣告一樣，除非您在呼叫函式之前，事先宣告了函式否則將無法使用該函式，同樣地，全域變數也必須在使用前先宣告，否則無法使用。

9.1.1依照宣告位置判斷變數等級

- 【觀念範例9-2】：外在變數的宣告位置所造成的影響。
- 範例9-2：ch9_02.c

9.1.1依照宣告位置判斷變數等級

```
1  /*******************************
2      檔名:ch9_02.c
3      功能:全域變數與區域變數
4   *******************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  int a=10;
8  void func1(void);
9   void main(void)
10 {
11  int c=20;
12  printf("a=%d\n",a);
13  /* printf("b=%d\n",b); */
14  func1();
15  /*  system("pause");  */
16 }
```

```
17 int b=100;
18  void func1(void)
19 {
20  printf("a=%d\n",a);
21  printf("b=%d\n",b);
22 }
```

- 執行結果：

```
a=10
a=10
b=100
```

9.1.1依照宣告位置判斷變數等級

- 全域變數與區域變數同名
  - 全域變數與區域變數可能會出現同名的現象，此時若在函式內部存取一個變數，首先會存取到區域變數。
  - 若不存在該區域變數，則編譯器會視為存取全域變數。
- 【觀念範例9-3】：全域變數與區域變數同名，視為不同的變數。
- 範例9-3：ch9_03.c

9.1.1依照宣告位置判斷變數等級

```
1  /*********************************
2       檔名:ch9_03.c
3       功能:全域變數與區域變數同名
4   *********************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  int a=10;
8   void func1(void)
9  {
10   int a=5;
11   a=a+1;
12   printf("func1的 a=%d\n",a);
13 }
14  void func2(void)
15 {
16   a=a+1;
17   printf("全域的 a=%d\n",a);
18 }
```

```
19 void main(void)
20 {
21   int a=20;
22   a=a+1;
23   printf("main的 a=%d\n",a);
24   func1();
25   func2();
26   /*  system("pause");  */
27 }
```

9.1.1依照宣告位置判斷變數等級

- 執行結果

```
main的 a=21
func1的 a=6
全域的 a=11
```

- 範例說明：
  - 明顯地，全域變數a、main()區域變數a、func1()區域變數a同名，若未特別指定，依照同名變數的取用規則，會先試圖抓取區域的變數，若找不到區域變數，才會抓取全域變數。

9.1.2 明確宣告變數等級

- 之前我們所介紹的變數宣告都是隱含式的變數宣告方式，也就是僅僅宣告變數的資料型態而未指定變數的等級。
- 此時，編譯器將自動指定內定的變數等級給該變數，例如在函式內使用隱含式變數宣告的變數等級內定為auto。
- 除了隱含式的變數宣告之外，我們也可以明確地在宣告變數時，同時指定該變數的等級。下列是明確宣告變數等級的詳細語法：

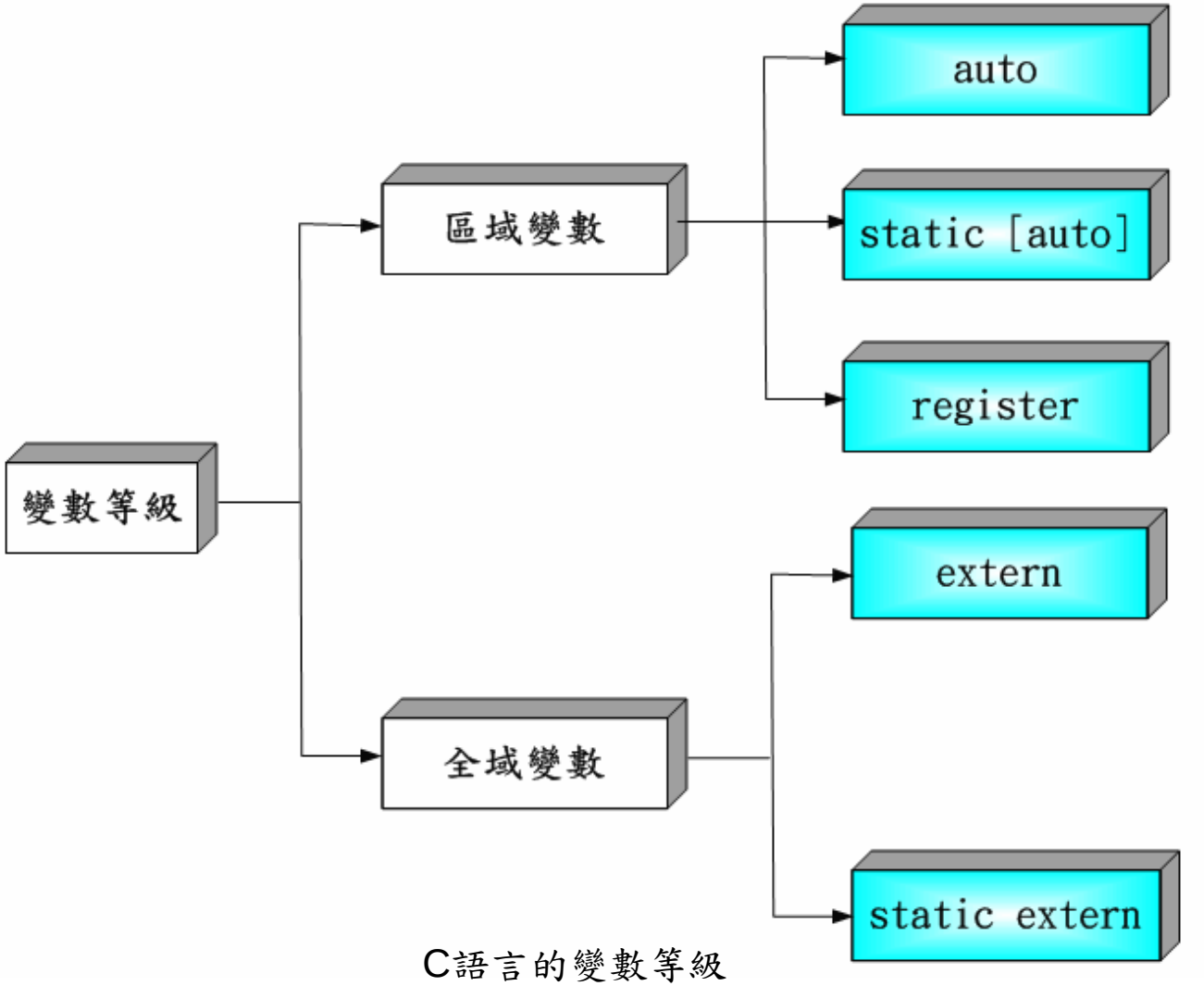
```
變數等級  變數資料型態  變數名稱;
```

9.1.2 明確宣告變數等級

- 【範例】：
  - static int a;
  - extern double b;
  - auto char c;
  - register d;

- 明確宣告變數等級，一共有5種等級可供選擇，分別是auto、static auto、extern、static extern、register。
- 將這五種等級加以分類為全域變數與區域變數
  - 全域變數包含extern、static extern兩種變數等級。
  - 區域變數包含auto區域變數、static [auto]區域變數、register三種變數等級。

9.1.2 明確宣告變數等級



C語言的變數等級

9.2 自訂程式區段

- 還記得我們在前面介紹過C語言的敘述分為很多種嗎？其中我們將可將眾多敘述組合成複合敘述，也就是由『{』與『}』包裝的程式區段。

- 這些區段將決定一個變數的視野與生命週期，傳統上，我們會使用區段符號來代表一個函式、迴圈...等等的程式碼，由於複合敘述也是敘述的一種，所以我們可以在函式內自訂程式區段。
  - 舉例來說，在下面這個範例中，我們在func1()函式中，另外定義了一個程式區段，這個範例仍然是一個合法的C語言程式。

9.2 自訂程式區段

- 【觀念範例9-4】：自訂程式區段。

- 範例9-4：ch9_04.c

9.2 自訂程式區段

```
1  /************************
2       檔名:ch9_04.c
3       功能:自訂程式區段
4   ************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7   int var5=50;
8   void func1(void)
9  {
10   int var1=10;
11    printf("var1=%d\n",var1);
12   {
13      int var2=20;
14      printf("var2=%d\n",var2);
15   }
16   printf("var5=%d\n",var5);
17 }
```

```
18 void main(void)
19 {
20   int var3=30;
21   func1();
22   printf("var3=%d\n",var3);
23   {
24     int var4=40;
25     printf("var4=%d\n",var4);
26   }
27   printf("var5=%d\n",var5);
28   /*  system("pause");  */
29 }
```

9.2 自訂程式區段

- 執行結果

```
var1=10
var2=20
var5=50
var3=30
var4=40
var5=50
```

9.2 自訂程式區段

- 範例說明：
  - 我們在func1()函式中另外定義了一個由『{』與『}』包裝的程式區段。
  - 我們在main()函式中另外定義了一個由『{』與『}』包裝的程式區段。
  - 本範例去除兩個{}都不會影響程式正確性及執行結果。
  - 但是實際上卻會影響了var2、var4的視野與生命週期。

9.3 區域變數

- 定義於區段（例如：函式或自訂區段）內部的變數稱為區域變數，其有效範圍僅限於區段之中，因此只有在定義該變數的區段中，才可以使用這個變數。
- 在C語言中，區域變數又分為auto、static兩種。
- 9.3.1 auto變數等級
  - 當我們在區段內定義某一個區域變數時，若未宣告變數等級，則該變數內定為auto變數等級。
  - 因此下列兩種語法，都是定義auto變數的方式：

9.3.1 auto變數等級

- 語法一：隱含宣告auto變數等級

```
{
變數資料型態  變數名稱;        /*   必須在區段內宣告   */
……
}
```

- 語法二：明確宣告auto變數等級

```
{
auto   變數資料型態  變數名稱;   /*   必須在區段內宣告   */
……
}
```

  - auto等級的變數，其生命週期與視野僅限於宣告時所在的區段內。
    - 【區段】：所謂程式的區段，代表由『{』到『}』之間，例如一個函式內。

9.3.1 auto變數等級

- 【範例1】

```
func1()
{
int a,b;
.....
}
main()
{
char c;
.....
}
```

【範例2】

```
func1()
{
auto int a,b;
.....
}
main()
{
auto char c;
.....
}
```

- 說明：
  - 上面兩個範例的a,b,c都是auto等級的變數，其中變數a,b的生命週期與視野為func1()函式，變數c的生命週期與視野為main()函式。

9.3.1 auto變數等級

- auto變數等級的生命週期
  - 變數的生命週期及視野與區段有非常緊密的關係。
    - 以函式區段為例，由於auto等級的變數必須在區段一開始就宣告，因此auto等級變數的生命週期是由該函式開始被執行時，直到函式執行完畢返回時。
- auto變數等級的視野
  - 既然auto變數僅存在於區段開始被執行到區段執行完畢，自然也只有該區段內的敘述可以存取auto變數。
  - 所以auto變數的視野，也僅限於宣告該變數的區段。

- 【觀念範例9-5】：auto變數的生命週期與視野。
- 範例9-5：ch9_05.c

```
1  /**********************
2      檔名:ch9_05.c
3      功能:auto變數
4   **********************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  void func1(void)
8  {
9   auto int var2=30;
10  printf("區段外var2 = %d\n",var2);
11  {
12     auto int var3 = 40;
13     printf("區段內var3 = %d\n",var3);
14     printf("區段內var2 = %d\n",var2);
15  }
16 }
17 void main(void)
18 {
19  auto int var1 = 10;
20  printf("區段外var1 = %d\n",var1);
21  {
22     auto int var1 = 20;
23     printf("區段內var1 = %d\n",var1);
24  }
25  func1();
26 }
```

9.3.1 auto變數等級

- 執行結果：

```
區段外var1 = 10
區段內var1 = 20
區段外var2 = 30
區段內var3 = 40
區段內var2 = 30
```

- 範例說明：
  - 宣告main函式的區域auto變數var1，只要是在main裡面的敘述都可以存取它。
  - 宣告main函式內自訂區段的區域auto變數var1，只要是在該區段內的敘述都可以存取它。

9.3.1 auto變數等級

- 【觀念範例9-6】：auto變數的生命週期
- 範例9-6：ch9_06.c

```
1  /*********************************
2      檔名:ch9_06.c
3      功能:auto變數
4   *********************************/
5
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  void func1(void)
10 {
11  auto int var1;
12
13  printf("var1 = %d\n",var1);
14  var1=100;
15  printf("var1 = %d\n",var1);
16  var1=var1+1;
17  printf("var1 = %d\n",var1);
18 }
```

```
19
20 void func2(void)
21 {
22  auto int var2;
23  var2=0;
24  var2++;
25 }
26
27 void main(void)
28 {
29  func1();
30  func2();
31  printf("==========\n");
32  func1();
33  /*  system("pause");  */
34 }
```

9.3.1 auto變數等級

- 執行結果：

```
var1 = 895
var1 = 100
var1 = 101
==============
var1 = 1
var1 = 100
var1 = 101
```

- 範例說明：
  - 第29、32行各呼叫func1()函式一次。
  - 第一次呼叫func1()函式，執行到第13行時，var1由於未設定初值，因此無法掌控變數值，在執行結果中，出現的是『895』，經由運算，在離開func1()函式之前，var1變數值為101。

9.3.1 auto變數等級

- 由func1()函式返回時，所有func1函式內被宣告為auto的變數將會被釋放記憶體空間，因此var1變數也將被釋放。

- 第二次呼叫func1()函式，執行到第13行時，var1的值不會是101，雖然第一次呼叫func1()時，var1最後的值為101，但由於func1()執行完畢返回時，已經釋放var1，因此，當我們再次呼叫該函式時，只會得到一個新配置的變數var1。

- 如果我們想要保留var1變數值，不因函式返回而消失的話，就必須將var1宣告為靜態變數。

9.3.2 static 區域變數等級

- static區域變數和auto變數差不多，唯一的差別僅在於static區域變數不會因為區段執行完畢而被釋放，因此，上一次區段執行完畢所留下的變數值，將可以保留到下一次執行區段時（例如下一次的函式呼叫）仍然能夠繼續使用。
  - 語法：宣告static區域變數等級

```
{
static  變數資料型態  變數名稱;   /*   必須在區段內定義  */
……
}
```

9.3.2 static 區域變數等級

- static 區域變數等級的生命週期
  - static區域變數的生命週期將由宣告後直到整個程式執行完畢。
- static區域變數等級的視野
  - static區域變數視野與auto區域變數相同，也就是僅限於宣告該變數的區段。
- 【觀念範例9-7】：static區域變數的生命週期
- 範例9-7：ch9_07.c

9.3.2 static 區域變數等級

```
1  /*****************************
2      檔名:ch9_07.c
3      功能:static區域變數
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  void func1(void)
8  {
9   static int var1=100;
10
11  printf("var1 = %d\n",var1);
12  var1=var1+1;
13  printf("var1 = %d\n",var1);
14 }
15 void main(void)
16 {
17  func1();
18  printf("==========\n");
19  func1();
20  /*  system("pause");  */
21 }
```

– 執行結果：

```
var1 = 100
var1 = 101
==========
var1 = 101
var1 = 102
```

9.3.3 register變數等級

- 某些常被運算的區域變數，我們可以將之宣告為register變數，以便指定存放在CPU的暫存器中，加快資料的存取速度。但並不是所有的變數都可以將之宣告為register變數。它必須有下列限制：
  - 必須是auto區域變數，不能是static區域變數。
  - 必須是整數類的資料型態，例如int、char。
- 雖然我們將某個變數宣告為register變數，但仍無法保證它將被放在暫存器中被執行。
- 因此使用register宣告變數，並不一定會獲得較快的運算速度。
- 語法:

```
register  變數資料型態  變數名稱;   /*  資料型態必須是整數類  */
```

9.4 全域變數與外在變數

- C語言的全域變數分為兩種，一種是單一檔案內的全域，另一種則是跨檔案的全域。

- 事實上，我們將C語言變數區分為全域變數與區域變數，所依據的是變數的生命週期與視野。

  - 當一個變數的生命週期或視野僅限於函式（或區段）之內時，我們將之稱為區域變數。

  - 而一個變數的生命週期或視野不限於函式之內時，則稱之為全域變數。

9.4 全域變數與外在變數

- 由函式的角度來看，一個在函式內部宣告的變數稱之為內在變數。
- 相對於內在變數而言，凡是在函式外部定義的變數，則稱之為外在變數（external variable）。
- 內在變數的視野，無論如何一定會被侷限在函式內部，但生命週期則不一定。
- 而外在變數的視野則可以跨越數個函式，甚至是不同檔案中的數個函式，而生命週期則由程式開始執行時或宣告變數時，直到程式執行完畢。

9.4.1 external變數等級

- 對於個體單位而言，如果要存取個體之外的外在變數，則必須先使用extern加以宣告。語法如下：

```
extern 變數資料型態 全域變數;
```

  - 【語法說明】：
    - 如果想要跨檔讀取其他檔案的全域變數，則必須在函式內使用上述語法，宣告該變數為外在變數。
    - 如果只是要讀取自身檔案的全域變數且內部無此變數，則可以省略上述宣告。
    - 變數宣告前加上extern關鍵字後，編譯器就會知道該變數已經在別的地方宣告過了，而不需要再保留記憶體空間給該變數。

9.4.1 external變數等級

- 【觀念範例9-10】：外在變數宣告，單一檔案示範。
- 範例9-10：ch9_10.c

9.4.1 external變數等級

```
1  /************************************
2      檔名:ch9_10.c
3      功能:外在變數宣告(單一檔案示範)
4   ************************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7   int i;
8  void func1(void)
9  {
10  extern int i;       /* 可省略此行 */
11  i++;
12 }
13  void main(void)
14 {
15  extern int i;      /* 可省略此行 */
16  printf("i=%d\n",i);
17  func1();
18  printf("i=%d\n",i);
19  /*  system("pause");  */
20 }
```

– 執行結果：

```
i=0
i=1
```

9.4.3 4種變數等級的區別

- 經由前面章節的介紹，我們已經學習過5種C語言變數的等級，除了register等級之外，可以分為兩大類：內在變數與外在變數。也可以由static的角度區分為兩大類：普通變數與static變數。我們將這四種變數等級的生命週期與視野以表格方式呈現如下。

變數等級	生命週期	視野
auto變數（普通內在變數）	從函式（或區段）執行開始，直到函式（或區段）執行結束。	僅限於函式（或區段）之內。
static auto變數（static內在變數）	直到程式結束為止。	僅限於函式（或區段）之內。
extern變數（普通外在變數）	從程式執行到程式結束為止。	跨檔案的所有函式。
static extern變數（static外在變數）	從程式執行到程式結束為止。	同一檔案內的所有函式。

9.6 本章回顧

- C語言的變數，依照生命週期與視野，約略可分為兩類：全域變數及區域變數。
  - 全域變數定義於所有函式之外，在變數宣告後面的所有函式都可以使用該變數。全域變數的生命週期與程式的執行時間相同，必須等到程式結束執行，全域變數佔用的記憶體才會被釋放。
  - 區域變數定義於區段（例如函式）內部，只有定義該變數的區段可以使用這個變數，若非宣告為static auto，則該區段執行完畢，生命週期也宣告結束。

9.6 本章回顧

- 對於區域變數而言，如果該變數常常被使用，我們可以將之宣告為register等級，提醒編譯器，將之使用暫存器來存放變數，以便加快程式執行速度。

變數等級	生命週期	視野
auto變數（普通內在變數）	從函式（或區段）執行開始，直到函式（或區段）執行結束。	僅限於函式（或區段）之內。
static auto變數（static內在變數）	直到程式結束為止。	僅限於函式（或區段）之內。
extern變數（普通外在變數）	從程式執行到程式結束為止。	跨檔案的所有函式。
static extern變數（static外在變數）	從程式執行到程式結束為止。	同一檔案內的所有函式。

本章习题