Variable Argument Lists

在 C 語言中，當函數的 argument list 出現 ellipsis(…)，表示該函數需要使用到 variable argument lists (不定參數串列)或稱 variable-length argument lists(不定長度參數串列)，要取得 list 中的 arguments 時，要透過 type(型態)、幾個 macro(巨集)及 va_list。Macro 名稱爲 va_start、va_arg 及 va_end，這是搭配著 va_list 使用的。以上的巨集可在 stdarg.h 中找到，若是較舊的編譯器時，可能是在 varargs.h 中。巨集定義如下(出至 Visual C++ 6.0)：

```
#define _INTSIZEOF(n)     ( (sizeof(n) + sizeof(int) - 1) & ~(sizeof(int) - 1) )
#define va_start(ap,v)    ( ap = (va_list)&v + _INTSIZEOF(v) )
#define va_arg(ap,t)      ( *(t *)((ap += _INTSIZEOF(t)) - _INTSIZEOF(t)) )
#define va_end(ap)        ( ap = (va_list)0 )
```

接下來，一一介紹這四個巨集的原理。

一. _INTSIZEOF

在 32-Bit protected mode 下，若資料位址是 double word(4 bytes) boundary，換句話說，若資料的位址能被 4 整除的話，CPU 讀取的速度較快。所謂一個位址爲 double word boundary，即位址爲 4 的倍數。爲了讓資料在 boundary 上，要利用到 memory alignment，巨集_INTSIZEOF 的目的就是如此。_INTSIZEOF 功用爲【無條件進位取 sizeof(int)的倍數】，怎麼說呢？假設有 n 個蘋果，無條件進位取 x 的倍數，算法爲：

(n + (x – 1)) – ((n + (x - 1)) % x) … (1)

假設 x 是 2 的次方(x = 1,2,4,8…)，上式可簡化爲：

(n + (x – 1) & ~(x – 1) … (2)

註：(1)式的概數觀念到(2)式的簡化過程，由工程師之家的coco「不定參數的 C 函式」所提出說明。

簡單的證明一下：

Let t = (n + (x – 1))

(1) 式 ➔ t – t % x

(2) 式 ➔ t & ~(x-1)

條件：x 是 2 的次方

❶ t % x 意思爲取得 t % x 的餘數，即 t 中，取得在 x 右側的 Bits

Ex:

假設 t = 19 , x = 4

➔19 % 4 = 3

➔ 19 的二進制： 0000 0000 0000 0000 0000 0000 0001 0011

4 的二進制： 0000 0000 0000 0000 0000 0000 0000 0100

保留 4 右側的 Bits 0000 0000 0000 0000 0000 0000 0000 0011

得 t – t % x 意思爲取得 t 中，保留在 x 左側的 Bits(含 x 本身的 bit)。

❷ t & ~(x – 1) 的意思取得 t 在 x 左側的 Bits

在條件爲 x 是 2 的次方， t – t % x 等於 t & ~(x – 1)得證。

把 sizeof(n)當作 n，sizeof(int)當作 x，無條件進位取 sizeof(int)的倍數，以達到 alignment 的效果。

Note:在 32-Bit 的平台中，int 的大小爲 4 bytes，double word boundary 對 CPU 讀取的速度較快，而在 16-Bit 的平台中，int 的大小爲 2 bytes，word boundary 對 CPU 讀取的速度較快。_INTSIZEOF 會根據不同的平台，做正確的 alignment。

二. va_start、va_arg 及 va_end

```
#define va_start(ap,v)      ( ap = (va_list)&v + _INTSIZEOF(v) )
#define va_arg(ap,t)        ( *(t *)((ap += _INTSIZEOF(t)) - _INTSIZEOF(t)) )
#define va_end(ap)          ( ap = (va_list)0 )
```

以上的巨集利用例子來說明會比較清楚恰當：

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

void dumpList(int n, ...){
    va_list arg_ptr;                    // -- (1)
    va_start(arg_ptr, n);               // -- (2)

    printf("count = %d: ", n);
    while (n-- > 0) {
        int i = va_arg(arg_ptr, int);   // -- (3)
        printf("%d ", i);
    }
    printf("\n");

    va_end(arg_ptr);                    // -- (4)
}

int main()
{
    dumpList(1, 1);
    dumpList(3, 1, 2, 3);
    return 0;
}
```

(1) 宣告一個指向不定參數串列的指標 arg_ptr，其實 va_list 定義爲 char * :

```
typedef char* va_list
```

(2) 以指向不定參數串列的指標 arg_ptr 及函數的參數列中最後一個固定的參數，作爲巨集 va_start 的參數，而 va_start 主要用來初始化 arg_ptr。

va_start(arg_ptr, n);

等於 arg_ptr = (va_list)&n + _INTSIZEOF(v)

從上得知，在呼叫 va_start 巨集後，arg_ptr 指向第一個不定參數

(3) 傳回目前 arg_ptr 指向的參數，型態爲 int，同時 arg_ptr 指向下一個不定參數。

(4)va_end 巨集把指標 arg_ptr 指向 0
這個例子簡單的介紹不定參數的用法，更多的應用與細節會在另一篇《不定參數講座》說明。

三. Variable Promotions

在不定參數串列中，由於沒有宣告參數，無法從目前的資訊中去得到型態與轉換的方式，所以在不定參數中，應用《default argument promotions》:

(1) 型態爲 char 及 short int 會 promote 至 int

(2) 型態爲 float 會 promote 至 double

若沒有注意這樣的 promotion，會發生什麼樣的問題呢？以一個例子來說明：

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

void dumpList(int n, ...){
    va_list arg_ptr;
    va_start(arg_ptr, n);

    printf("count = %d: ", n);
    while (n-- > 0) {
        double i = va_arg(arg_ptr, double);
        printf("%f ", i);
    }
    printf("\n");

    va_end(arg_ptr);
}

int main()
{
    dumpList(1, 2.1);
    dumpList(3, 2.1, 4.3, 6.8);
    return 0;
}
```

輸出：

```
count = 1: 2.100000
count = 3: 2.100000 4.300000 6.800000
```

以上的例子 2.1、4.3.、6.8 會被視爲型態 double，所以在使用 va_arg 時，型態要選擇 double，千萬不要使用 float，在預設中，並不會有 float 型態在不定參數串列的型態中。。試試看以下：

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

```
void dumpList(int n, ...){
    va_list arg_ptr;
    va_start(arg_ptr, n);

    printf("count = %d: ", n);
    while (n-- > 0) {
        float i = va_arg(arg_ptr, float);
        printf("%f ", i);
    }
    printf("\n");

    va_end(arg_ptr);
}

int main()
{
    dumpList(1, 2.1);
    dumpList(3, 2.1, 4.3, 6.8);
    return 0;
}
```

結果：

```
count = 1: -107374184.000000
count = 3: -107374184.000000 2.012500 0.000000
```

因爲不定參數視 2.1、4.3、6.8 型態爲 double，而你卻當作爲 float 的參數，錯誤是預期的。不死心，再試試看把上面的程式，main 裡面改成：

```
float fNum = 2.1f;
dumpList(1,fNum);
```

翻成組合語言來看的話：

```
        float fNum = 2.1f;
004010C8 C7 45 FC 66 66 06 40 mov     dword ptr [ebp-4],40066666h
        dumpList(1,fNum);
004010CF D9 45 FC             fld     dword ptr [ebp-4]
004010D2 83 EC 08             sub     esp,8
004010D5 DD 1C 24             fstp    qword ptr [esp]
004010D8 6A 01                push    1
004010DA E8 26 FF FF FF       call    @ILT+0(dumpList) (00401005)
004010DF 83 C4 0C             add     esp,0Ch
```

發現，雖然 fNum 的型態爲 float，但 fNum 的值仍被視爲 double，再一次提醒，因爲 Variable Promotion 的關係，在使用 va_arg 時，要注意不定參數型態的問題。

以上的內容，對於不定參數串列做簡短初步的介紹，希望能幫助瞭解與應用。若以上有任何的問題，歡迎來信，若內容有誤或需要修正的地方，也歡迎接受糾正或批評。

Written By James(abc9250@yahoo.com.tw) 2004/9/3

相關連結：

1. Variable-Length Argument Lists Q&A
   http://www.eskimo.com/~scs/C-faq/s15.html
2. The code project – Variable Argument Functions
   http://www.codeproject.com/cpp/argfunctions.asp