可变参数即表示参数个数可以变化，可多可少，也表示参数的类型也可以变化，可以是int，double还可以是char\*，类，结构体等等。可变参数是实现printf()，sprintf()等函数的关键之处，也可以用可变参数来对任意数量的数据进行求和，求平均值带来方便（不然就用数组或每种写个重载）。在C#中有专门的关键字parame，但在C,C++并没有类似的语法，不过幸好提供这方面的处理函数，本文将重点介绍如何使用这些函数。

第一步 可变参数表示

用三个点…来表示，查看printf()函数和scanf()函数的声明：

int printf(const char \*, ...);

int scanf(const char \*, ...);

这三个点用在宏中就是变参宏（Variadic Macros），默认名称为\_\_VA\_ARGS\_\_。如：

#define WriteLine(...) { printf(\_\_VA\_ARGS\_\_); putchar('\n');}

再WriteLine("MoreWindows");

考虑下printf()的返回值是表示输出的字节数。将上面宏改成：

#define WriteLine (...) printf(\_\_VA\_ARGS\_\_) + (putchar('\n') != EOF ? 1: 0);

这样就可以得到WriteLine宏的返回值了，它将返回输出的字节数，包括最后的’\n’。如下例所示i和j都将输出12。

       int i = WriteLine("MoreWindows");

       WriteLine("%d", i);

       int j = printf("%s\n", "MoreWindows");

       WriteLine("%d", j);

第二步 如何处理va\_list类型

函数内部对可变参数都用va\_list及与它相关的三个宏来处理，这是实现变参参数的关键之处。

在<stdarg.h>中可以找到va\_list的定义：

typedef char \*  va\_list;

再介绍与它关系密切的三个宏要介绍下：va\_start()，va\_end()和va\_arg()。

同样在<stdarg.h>中可以找到这三个宏的定义：

#define va\_start(ap,v)  ( ap = (va\_list)&v + \_INTSIZEOF(v) )

#define va\_end(ap)      ( ap = (va\_list)0 )

#define va\_arg(ap,t)    ( \*(t \*)((ap += \_INTSIZEOF(t)) - \_INTSIZEOF(t)) )

其中用到的\_INTSIZEOF宏定义如下：

#define \_INTSIZEOF(n) ( (sizeof(n) + sizeof(int) - 1) & ~(sizeof(int) - 1) )

来分析这四个宏：

va\_end(ap)这个最简单，就是将指针置成NULL。

va\_start(ap,v)中ap = (va\_list)&v + \_INTSIZEOF(v)先是取v的地址，再加上\_INTSIZEOF(v)。\_INTSIZEOF(v)就有点小复杂了。( (sizeof(n) + sizeof(int) - 1) & ~(sizeof(int) - 1) )全是位操作，看起来有点麻烦，其实不然，非常简单的，就是取整到sizeof(int)。比如sizeof(int)为4，1,2,3,4就取4，5,6,7,8就取8。对x向n取整用[**C语言**](http://lib.csdn.net/base/c)的算术表达就是((x+n-1)/n)\*n，当n为2的幂时可以将最后二步运算换成位操作——将最低 n - 1个二进制位清 0就可以了。

va\_arg(ap,t)就是从ap中取出类型为t的数据，并将指针相应后移。如va\_arg(ap, int)就表示取出一个int数据并将指针向移四个字节。

因此在函数中先用va\_start()得到变参的起始地址，再用va\_arg()一个一个取值，最后再用va\_end()收尾就可以解析可变参数了。

第三步 vfprintf()函数和vsprintf()函数

vfprintf()这个函数很重要，光从名字上看就知道它与经常使用的printf()函数有很大的关联。它有多个重载版本，这里讲解最常用的一种：

函数原型

int vfprintf(

   FILE \*stream,

   const char \*format,

   va\_list argptr

);

第一个参数为一个FILE指针。FILE结构在C语言的读写文件必不可少。要对屏幕输出传入stdout。

第二个参数指定输出的格式。

第三个参数是va\_list类型，这个少见，但其实就是一个char\*表示可变参参数的起始地址。

返回值：成功返回输出的字节数（不包括最后的’\0’），失败返回-1。

vsprintf()与上面函数类似，就只列出函数原型了：

int vsprintf(

   char \*buffer,

   const char \*format,

   va\_list argptr

);

还有一个int \_vscprintf(const char \*format, va\_list argptr );可以用来计算vsprintf()函数中的buffer字符串要多少字节的空间。

代码范例

下面就给出了自己实现的printf()函数(注1)与WriteLine()函数

int Printf(char \*pszFormat, ...)

{

       va\_list   pArgList;

       va\_start(pArgList, pszFormat);

       int nByteWrite = vfprintf(stdout, pszFormat, pArgList);

       va\_end(pArgList);

       return nByteWrite;

}

int WriteLine(char \*pszFormat, ...)

{

       va\_list   pArgList;

       va\_start(pArgList, pszFormat);

       int nByteWrite = vfprintf(stdout, pszFormat, pArgList);

       if (nByteWrite != -1)

              putchar('\n'); //注2

       va\_end(pArgList);

       return (nByteWrite == -1 ? -1 : nByteWrite + 1);

}

调用与printf()函数相同。

再给出一个用可变参数来求和，遗憾的在C,C++中无法确定传入的可变参数的个数（printf()中是通过扫描'%'个数来确实参数的个数的），因此要么就要指定个数，要么在参数的最后要设置哨兵数值：

**设置哨兵数值：**

const int GUARDNUMBER = 0; //哨兵标识

//变参参数的个数无法确定，在printf()中是通过扫描'%'个数，在这通过设置哨兵标识来确定变参参数的终止

int MySum(int i, ...)

{

       int sum = i;

       va\_list argptr;

       va\_start(argptr, i);

       while ((i = va\_arg(argptr, int)) != GUARDNUMBER)

              sum += i;

       va\_end(argptr);

       return sum;

}

可以这样的调用：   printf("%d\n", MySum(1, 3, 5, 7, 9, 0));

但不可以直接传入一个0：   printf("%d\n", MySum(0)); //error

**指定个数：**

int MySum(int nCount, ...)

{

       if (nCount <= 0)

              return 0;

       int sum = 0;

       va\_list argptr;

       va\_start(argptr, nCount);

       for (int i = 0; i < nCount; i++)

              sum += va\_arg(argptr, int);

       va\_end(argptr);

       return sum;

}

调用时第一个参数表示后面参数的个数如：

       printf("%d\n", MySum(5, 1, 3, 5, 7, 9));

       printf("%d\n", MySum(0));

代码所用的头文件：

#include <stdarg.h>

#include <stdio.h>

可变参数的使用方法远远不止上述几种，不过在C,C++中使用可变参数时要小心，在使用printf()等函数时传入的参数个数一定不能比前面的格式化字符串中的’%’符号个数少，否则会产生访问越界，运气不好的话还会导致程序崩溃。

可变参数的原形理涉及到调用函数时参数的入栈问题，这个下次再开一篇进行专门的探讨。

注1．    网上有不用vfprintf()自己解析参数来实现printf()的，但很少能将功能做到与printf()相近（实际上能完全熟悉printf()的人已经就不多，不信的话可以先看看《C陷阱与缺陷》了解printf()很多不太常用的参数，再去Microsoft Visual Studio\VC98\CRT\SRC中查看OUTPUT.C对printf()的实现）。

注2．    如果输出单个字符 putchar(ch)会比printf(“%c”, ch)效率高的多。在字符串不长的情况下，多次调用putchar()也会比调用printf(“%s\n”, szStr);的效率高。在函数大量调用时非常明显。