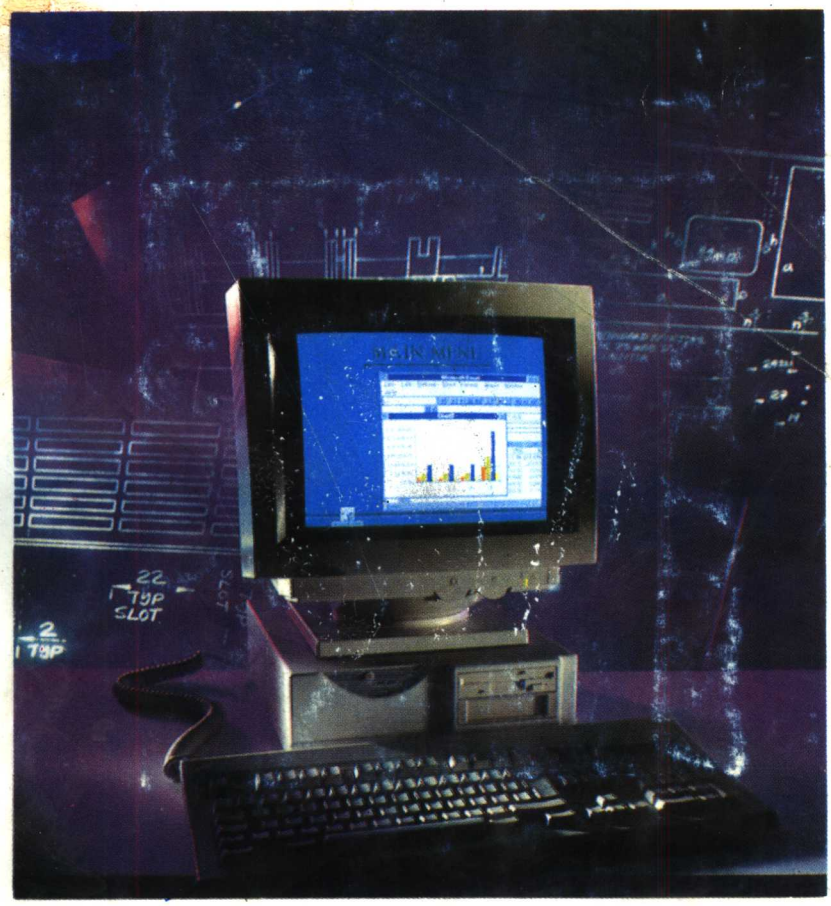
计算机基础教育系列教材

计算机语言接口编程与技巧

雷方桂
王云宜
吴耀斌 编 著



中南工业大学出版社

计算机语言接口编程与技巧

雷方桂　吴耀斌　编著

王云宜　　主审

中南工业大学出版社

【湘】新登字 010 号

计算机语言接口编程与技巧

雷方桂　吴耀斌　编著

王云宜　主审

责任编辑：肖梓高

*

中南工业大学出版社出版发行

长沙市东方印刷厂印装

新华书店总店北京发行所经销

*

开本：787×1092　1/16　印张：14.25　字数：356 千字

1995 年 7 月第 1 版　1995 年 7 月第 1 次印刷

印数：0001—6000

ISBN 7-81020-772-5/TP・050

定价：13.50 元

前　言

微型计算机的发展日新月异，随着微机硬件技术的不断更新换代，软件技术的发展也突飞猛进。计算机语言的种类越来越多，其功能也越来越强，应用范围也越来越广。同时，计算机语言各具特色，在各种系统软件、软件工具及实用程序的开发过程中，应用多种语言联合编程可以缩短软件的开发周期，节省大量人力物力，因此，计算机语言之间的接口程序设计技术和技巧是各级程序员所必须掌握和熟悉的关键技术之一。为此我们编写了这本书。

本书主要介绍了C语言与汇编语言、C语言与DOS之间以及高级语言与C语言、高级语言与汇编语言之间的接口程序设计方法和技巧。全书分9章：C语言与汇编语言的接口，包括了其接口编程的规则和约定以及参数传递方法，通过实例阐述了从C语言程序调用汇编以及从汇编程序调用C语言函数的方法和技巧；高级语言与C语言之间的接口，讲述了PASCAL语言、BASIC语言、FORTRAN语言、PROLOG语言以及dBASE Ⅲ、FOXBASE$^+$数据库管理系统与C语言的接口编程方法和技巧；汇编语言与高级语言的接口，介绍了Turbo PASCAL、Turbo BASIC、Turbo PROLOG与汇编语言的接口以及汇编语言与FROTRAN语言的接口程序实例；C语言与DOS的接口，讲述了伪变量等基本概念、C语言与BIOS的接口以及C语言对DOS服务功能的调用方法；高级语言与DOS的接口，讲述了Turbo PASCAL、Turbo PROLOG与DOS的接口；高级语言之间的接口介绍了Turbo PROLOG与MS-FORTRAN、Turbo PROLOG与dBASE Ⅲ的接口；接口编程程序的调试介绍了Turbo Debugger调试程序的基本方法及实例；设备驱动程序的编写介绍了用C语言编写设备驱动程序规则和技巧；最后介绍了有关混合编程的实用程序。

本书在讲述规则和方法的同时，为读者提供了一些经典的实用程序，通过这些实例来理解接口编程的方法和技巧，同时这些实例也可以在程序开发中直接调用。本书主要面向各级程序员，同时也可作为计算机专业大专以上学生的教学参考书，也是非计算机专业本科以上学生学习计算机语言的一本有效的工具书。

参加本书编写的还有王建新、吴迪文、牛丽娜、王烟波、陈建华、朱磊、向期中等同志。由于作者时间和水平有限，书中有错误和不当之处，敬请广大同仁和读者批评指正。

编者

1994年12月20日

目　录

1　C语言与汇编语言的接口

1.1　C语言接口编程基础

(一) 接口编程概述

在实际开发和编制软件的过程中，人们常常需要使用多种语言混合编程，从而充分利用各种语言的特色，使开发和编程过程达到事半功倍的效果。在进行混合语言编程时，一项作业被分成若干功能模块，每个模块以函数或过程的形式存在，针对每一功能模块的特点选用适合的语言独立编程。例如，涉及低级处理和中断操作的功能模块通常选用汇编语言编制。每个模块编制好后，就要使用相应的语言编译程序对其进行编译形成目标文件，最后将多个目标文件连接在一起形成一个完整的可执行文件。

在进行接口语言程序设计时，要注意两个关键问题：

(1) 要选择一个恰当的主模块。主模块是主程序所在的模块。要形成最后的可执行代码，没有主程序是不行的。一般，总是选择一个最大的模块作为主模块。这样，在进行接口编程时，总会有一个主语言，当所选主语言编程很难实现或无法实现其功能时，才选用其它合适的语言针对该功能独立编程。而某些语言由于自身编译的限制，使其在进行接口编程时，必须以主模块身份存在。

(2) 要严格遵守语言之间的调用约定，即建立语言之间的正确接口。不同的语言之间或多或少存在着差异，如数据类型、函数参数的传递方式等等，这就要求在进行接口编程时，除了掌握所用语言之外，还必须熟悉各种语言之间的命名约定、参数传送协议以及调用约定，并在实际编程中遵守这些要求。

(二) C语言的编译模式

在进行C语言接口编程前，首先要了解C语言的各种编译模式。这是因为，选择适当的编译模式对提高接口编程效率是很重要的，有时甚至是接口编程成功与否的决定因素。

Microsoft C与Turbo C都提供了六种编译模式，下面逐一介绍这六种模式。

1、**微模式**　在该模式下，所用四个段寄存器（CS、DS、SS、ES）都为同一个值，程序和数据、堆栈都在同一段内，即所有的寻址都是16位。相应地，C源程序中所有指针都是近指针（near），所有的调用都是近调用。这种模式所生成的程序，在连接时加参数/t可转化为.COM文件。

2、**小模式**　在小模式中，数据段和代码段分离，即最大可用空间为128K，这种模式适合于大多数程序。由于寻址仍是16位，所以所有调用仍是近调用，所有指针仍是近指针，但它同时允许对个别不在代码段内的函数用far关键字来调用，对个别不在数据段内的数据也可以用far或huge关键字来修正指针。

3、**中模式**　中模式中允许有多个代码段，但数据段仍只有一个。代码段采用20位寻址，数据段仍采用16位寻址。在一个代码段中缺省调用是远调用，但也可以使用near关键字来

推翻缺省约定。通常，一个C语言函数就形成一个独立的代码段。

4、紧凑模式 紧凑模式与中模式相反，它只有一个代码段，却允许有多个数据段，也就是说代码仍是以16位进行寻址，而数据却要用20位进行寻址。

5、大模式 大模式允许有多个数据段和代码段，但全部静态数据仍限制在一个段(64K)内。

6、巨模式 巨模式与大模式唯一不同之处是巨模式不再要求静态数据必须在一个段内。

显然，这六种编程模式适用不同的场合，其中微模式所产生的文件执行速度最快而巨模式所产生的文件执行速度最慢，在进行单纯的C语言程序设计时，如果代码段和数据段都不是很大（64K内），则通常选用小模式，这时所产生的文件执行速度接近于微模式。但在进行接口程序设计时，数据和代码都有可能不在同一段内。因而要选择中模式甚至大模式进行编译。

(三) C语言外部接口约定原则

1、C语言关键词extern extern用于说明一个变量或函数是外部，即该变量或函数存在于另一个独立的程序中。格式如下：

extern<变量名>；　　或　　extern<函数原型>

例如：extern int outv;　　extern viod outf (int a);

表明outv是一个外部变量，而outf函数是一个外部函数。外部变量和外部函数只要说明正确，C语言就可按内部变量来使用或按内部函数来调用。

2、C语言的编译模式 要符合与其接口的语言的要求。一般情况下，C语言与高级语言(如PASCAL、PROLOG和BASIC等)所采用的编译模式通常是大模式。

1.2 调用约定与参数传递

(一) C语言编译调用约定

调用约定是指C语言编译程序将其信息传递给被调函数并从被调函数返回值的方法。包括：

(1) 要传递给被调函数的参数放在何处。

(2) 用什么汇编指令来调用函数。

(3) 被调函数得到控制权后系统处于何种状态，哪些寄存器可以破坏，哪些寄存器需要保存。

(4) 被调函数的返回值（如果有的话）应放在何处。

在一般情况下，C语言是利用堆栈将参数传递给被调函数的。如果参数是七种内部数据类型之一（即char、short int、int、long int、unsigned int、float、double）或者是一个结构，那么数据的实际值就被置于栈顶。如果参数是一个数组，那么就把数组的地址置于栈顶。表1-1给出了各种变量在栈中所占用的字节数。

Microsoft与Turbo C参数是由右至左顺序压入栈中，接着压入调用函数的返回地址。函数调用是通过call指令实现的。可能是近调用，也可能是远调用，这要依C程序所选模式而定。如果是near近调用，则只需压入偏移地址；如果是far远调用，则需压入段地

表 1-1 C 语言函数在传递变量时各种数据类型在堆栈中所占的字节数

类型	字节数	类型	字节数
char	2	short	2
signed char	2	signed short	2
unsigned char	2	unsigned short	2
int	2	signed int	2
unsigned int	2	long	4
unsigned long	4	float	8
double	8	(near)pointer	2(offset)
(far)pointer	4(segment and offset)		

址及偏移地址。

被调函数开始执行时，便从栈中取出参数的值进行相应的运算，而当被调函数运行结束时，函数将返回值传给调用函数（如果有的话），这个值通常被放在 AX 寄存器或 DX 寄存器中。表 1-2 给出了 C 语言函数的返回值的寄存器使用情况。其中，float、double、struct、union 的回送方法是将值放入静态数据区，然后返回其地址指针；在小模式下，地址指针放入 AX 中，大模式下地址指针 DX: AX 中。

表 1-2 C 语言函数返回值的寄存器使用情况

类型	寄存器与含义	类型	寄存器与含义
char	AX	long	低位在 AX 中，高位在 DX 中
unsigned char	AX	float&double	低位在 AX 中，高位在 DX 中
short	AX	struct&union	指向静态存储区，指针在 AX 中
unsigned short	AX	unsigned long	指向静态存储区，指针在 AX 中
int	AX	(near)pointer	偏移量在 AX 中
unsigned int	AX	(far)pointer	偏移量在 AX 中，段地址在 DX 中

在进行用汇编语言编写的被调用函数时，首先 BP（基指针）必须压入栈中保存，然后 SP（堆栈指针）的当前值放入 BP 中，以便子程序可以使用堆栈、任意存取数据。另外两个需要考虑的寄存器是 SI 和 DI。如果在汇编子程序中需要使用它们，则在子程序入口处亦需保存它们。这样，在汇编语言子程序返回之前，需恢复 BP、SI 和 DI 并且让堆栈指针复位。

（二）参数传递

C 语言支持两种函数参数传递方式：标准 C 语言方式和 Pascal 语言方式。其区别为：

(1) 标准 C 语言方式函数的参数自右至左次序压栈，而 Pascal 语言方式函数的参数是自左至右压栈。

(2) 标准 C 语言方式无需知道栈上所压参数的个数，它只假设所有参数均在栈中，而 Pascal 语言方式需知道向其传递的参数个数。

(3) 标准 C 语言方式在被调用子程序中无需将参数弹出栈，只需在主程序中调用指令后加入 add sp, n（n 为参数所占用的字节数）指令或其它指令来调整栈。而 Pascal 语言方

式是在被调子程序中根据参数的个数，在子程序最后用 ret n 清栈，这样，在主程序中调用指令后不再出现调整栈操作。

在 C 语言中，所有函数都默认使用 C 语言参数调用方式，在使用 Pascal 编译选择项时，所有函数才采用 Pascal 调用方式（如在 Turbo C 中使用 -P 编译选择项，在 Microsoft C 中使用 / GC 编译选择项），但在这种情况下，又可用 cdecl 修饰符在 Pascal 语言方式中强行使某个函数使用 C 语言参数传递方式，而用 Pascal 修饰符又可在 C 语言方式下强行使某个函数使用 Pascal 语言参数传递方式，如：

```
void cdecl func (int a, int b);
void pascal func (int a, int b);
```

（三）汇编子程序编写格式

在编写汇编子程序时，完全可以按照汇编语言的格式书写。然而，C 语言编译器由 C 语言转换的汇编代码有其固定的格式（Turbo C 用 -S 编译选择项，Microsoft C 用 / F 编译选择项）。

1、书写格式 首先对各个段进行说明或定义，如下列出了 Microsoft C 下各段存放的内容，Turbo C 下各段的存放内容也基本一致。

__BSS：未初始化的静态数据（那些用far关键字加以强制说明的除外）。

C__common：包含小模式的所有未初始化的全局变量。在紧凑模式或大模式中，这种类型数据是放在别名为 FAR__BSS 的数据段中。

__DATA：已经初始化的全局数据和静态数据（除far说明外），是缺省数据段。

Data segmeents：用far强制说明的全局数据和静态数据。已初始化的数据项是类别名为 FAR__DATA，未初始化的为 FAR__BSS。

STACK：自动变量，局部数据项。

CONST：只读常数（浮点常数，远数据的各个段值）。

__TEXT：码。

在编写汇编模块时，必须遵守：①保证连接程序能得到必要的信息；②保证文件格式符合 C 语言程序所用的存储模式。如下是汇编程序的一般格式。

```
标识符          名              文件名
< text>         SEGMENT         BYTE  PUBLIC 'CODE'
                ASSUME          CS: < text>, DS: < dseg>
                <……代码段……>
< text>         ENDS
< dseg>         GROUP           __DATA,  __BSS
< data>         SEGMENT         WORD PUBLIC 'BSS'
                <……初始化数据段……>
__BSS           SEGMENT         WORD PUBLIC "BSS"
                <……非初始化数据段……>
__BSS           ENDS
                END
```

其中：< text>、< data> 和 < dseg> 标识符根据所选用的存储模式可换成相应的符

号。表 1-3 列举了各种存储模式下的替换情况，其中 filename 为模块名。这些标识符的替换在整个程序中应保持一致。注意，在特大存储模式中，没有__BSS 段，GROUP 也可以不同。一般情况下，__BSS 是任选项，只需要用时才定义。

表 1-3 标识符替换及存储模式

模式	标识符替换	代码和数据指针
极小，小	<code> = __TEXT <data> = __DATA <dseg> = DGROUP	代码: DW__TEXT: * * * 数据: DW DGROUP: * * *
紧凑	<code> = __TEXT <data> = __DATA <dseg> = DGROUP	代码: DW__TEXT: * * * 数据: DW DGROUP: * * *
中	<code> = filename__TEXT <data> = __DATA <dseg> = DGROUP	代码: DD__TEXT: * * * 数据: DD DGROUP: * * *
大	<code> = filename__TEXT <data> = __DATA <dseg> = DGROUP	代码: DD__TEXT: * * * 数据: DD DGROUP: * * *
特大	<code> = filename__TEXT <data> = filename__DATA <dseg> = filename DGROUP	代码: DD: * * * 数据: DD: * * *

事实上，根据所选存储模式的不同，CS、DS、SS、ES 寄存器也进行相应的处理。

极小模式　　CS= DS= SS; ES= 暂存

极中模式　　CS! = DS; DS= SS; ES= 暂存

紧凑、大模式　CS! = DS! = SS; ES= 暂存（各模块有 CS）

特大模式　　CS! = DS! = SS; ES= 暂存（各模块有 CS 和 DS）

2、程序体的主要结构

```
push bp
mov bp, sp
……程序体
mov sp, bp
pop bp
ret
```

若为 Pascal 参数传递方式，则最后的"ret"改为：

ret * * *

其中：* * * 为需要栈的字节数。

3、常量与变量定义　存取模式也影响到如何定义作为代码、数据指针的数据常量。表 1-3 给出的指针格式，其中 * * * 表示指向的地址。其定义可以使用 DW(定义字)，也可以用 DD（定义双字），以表明指针产生的大小。数值和文本常量可按通常方法定义。

变量的定义方法与常量一样。若想说明非初始化变量，可在__BSS 段进行说明，并在

通常放值的位置上写上问号(?)。

4. 定义外部标识符 为了使标识符（子程序名或变量名）在汇编模块下可见，必须将它们说明为 PUBLIC 类型。例如要写一个含整型函数 total 和 level 以及整型变量 high、low 的模块，则必须在代码段中加入：

```
PUBLIC  _total, _level
```

在数据段中加入：

```
PUBLIC  _high, _low
_high DW 32767
_low DW 0
```

注意，这里 total、level、high、low 前都加了下划线。这是因为，C 语言程序在定义一个外部标识符时，其编译程序在默认状态下自动在其首部加上下划线。虽然可用-u-命令行选择项去掉下划线，但使用标准 C 语言库时，除非重构整个库，否则会出现问题。

然而，在使用 Pascal 方式时，情况有所不同，它们把所有标识符改为大写，且不冠以下划线。

5. 对 C 语言主程序的要求 汇编子程序对 C 语言主程序也有所要求。汇编子程序在 C 语言主程序中是通过外部函数调用实现的。在 C 语言子程序中，被引用的外部函数必须使用函数原型，即函数参数表的每个参数必须有明确的类型说明。

1.3 从 C 语言中调用汇编函数

掌握了各种调用约定、规则以及编写 C 语言函数与汇编子程序的格式，就可以编写一些汇编子程序。下面的例子分别是在 Microsoft C 6.0 和 Turbo 2.0 下编写求两个值中较大的程序。

（一）Microsoft 环境

【例 1】在 Microsoft C 6.0 下编写求两个值中较大的程序。

C 语言程序编写的主函数为（假设文件名为 MAXC.C）：

```
#include "stdio.h"
extern int max_val (int p1, int p2);
main( )
{
   int lmax;
   lmax= max_val (15, 30);
   printf ("%d", lmax);
}
```

假设是在小模式下运行，且使用 C 语言参数传递方式，不妨先看一看 MAXC.C 经 /Fa 编译选择项编译后生成的 AA.ASM 汇编代码，注意观察其参数的压栈次序与子程序调用返回后栈的调整。

```
                ; Static Name Aliases
                ; TITLE maxc.c
```

```
                .8087
INCLUDELIB SLIBCE
_TEXT   SEGMENT WORD PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA   ENDS
CONST   SEGMENT WORD PUBLIC 'CONST'
CONST   ENDS
_BSS    SEGMENT WORD PUBLIC 'BSS'
_BSS    ENDS
DGROUP  GROUP CONST, _BSS, _DATA
        ASSUME DS:DGROUP, SS:DGROUP
EXTERN  __acrtused:ABS
EXTERN  _printf:NEAR
EXTERN  ____aNchkstk:NEAR
EXTERN  __max__val :NEAR
_DATA   SEGMENT
$SG169  DB '&D',00H
_DATA   EBDS
_TEXT   SEGMENT
        ASSUME CS:TEXT
;Line 1
;Line 4
        PUBLIC main
_main   PROC NEAR
        push bp
        mov bp, sp
        mov ax, 2
        call ____aNchkstk
;               lmax= -2
; Line 7
        mov ax,30
        push ax
        mov ax,15
        push __max__val
        call ax
        add sp, 4
        push ax
        mov ax, OFFSET DGROUP: $SG169
        push ax
```

```
                call __printf
; Line 8
                mov sp, bp
                pop bp
                ret
                nop
_main           ENDP
_TEXT           ENDS
END
```

其中：

```
                mov ax, 30
                push ax
                mov ax, 15
                push ax
```

为参数入栈。P2（值为 30）首先入栈，其次 P1（值为 15））入栈，接着调用函数 call __max__val，call 指令后用指令 add sp, 4 来调整栈。

现在再来看用汇编语言编写的 max__val 函数的如下子程序：

```
_TEXT       SEGMENT WORD PUBLIC 'CODE'      ; L1
            ASSUME CS: _TEXT                ;L2
            PUBLIC __max__val               ;L3
__max__val  PROC NEAR                       ;L4
            push bp                         ;L5
            mov bp, sp                      ;L6
            xor ax,                         ;L7
            mov ax, WORD PTR [bp+4]         ;L8
            cmp WORD PTR [bp+6], ax         ;L9
            jle Loop1                       ;L10
            mov ax, WORD PTR [bp+6]         ;L11
Loop1:      mov sp, bp                      ;L12
            pop bp                          ;L13
            ret                             ;L14
__max__val  ENDP                            ;L15
_TEXT       ENDS                            ;L16
END                                         ;L17
```

程序说明：程序开始，建立各种段，段的形式随内存模式而变化。L3 行 PUBLIC __max__val 是为了使 max__val 子程序在汇编程序外可见，即为使 C 语言主函数可正确调用它，而说明成 PUBLIC 类型。L4 至 L15 行为程序部分。因为是小模式，所以在 L4 行采用 near 近调用，如果是微模式或紧模式亦为 near 近调用，其余模式为 far 远调用。L5、L6、L12、L13、L14 行为程序体的首尾框架。程序部分首先是保存 BP 寄存器（L5 行），并将栈指针的当前值放入 BP（L6 行），以便使用堆栈数据。因为假设用 C 语言方式

调用，故参数是自右至左压栈，[bp+6]存放的是 P2（即 30），[bp+4]存放的是 P1（即 15），[bp+2]存放的是返回地址。因为是近调用，所以只需两个字节存放返回地址。若为远调用，则[bp+8]存放的是 P2（即 30），[bp+6]存放的是 P1（即 15），[bp+2]和[bp+4]存放的是返回地址。L7～L11 行用于判断[bp+4]和[bp+6]中的数，取大的存入 AX 寄存器中。这些子程序中所做的工作基本完成。最后，返回主程序前恢复 bp（L12、L13）。

完成了汇编子程序与 C 语言主程序的编程，下一步就是使其正确编译、连接，生成 .EXE 文件。假设 C 语言主程序名为 MAXC.C，汇编子程序名为 MAXASM.ASM，则其步骤为：

(1) 对 MAXC.C，使用命令行参数生成 MAXC.OBJ。

C> CL / Fa / AS　MAXC.C

(2) 用 MASM 宏汇编对 MAXASM.ASM 进行汇编生成 MAXASM.OBJ。

C> MASM　MAXASM;

(3) 用标准 LINK 程序连接 MAXC.OBJ 与 MAXASM.OBJ 生成 MAXVAL.EXE。

C> LINK　MAXC MAXSAM，MAXVAL

运行 MAXVAL.EXE 程序可得正确结果 30。

若此例改用 Pascal 传递方式，则用子程序 MAXASM.ASM 应如何修改呢？

(1) Pascal 方式函数的参数是自左至右进栈的，故在[bp+6]中存放的是 P1（即 15），[bp+4]中存放的是 P2（即 30），[bp+2]存放的是返回地址。所以从栈中取数据的操作也应作相应的处理。MAXASM.ASM 程序作如下改动：

L8 行可改为：mov ax，WORD PTR [bp+6]

L9 行可改为：cmp WORD PTR [bp+4]，ax

L11 行可改为：mov ax,WORD PTR [bp+4]

(2) 汇编子程序返回子程序之前应有清栈操作，因为 P1、P2 占 4 个字节，所以 L14 行应改为 ret 4。清栈操作只能进行一次。

(3) 使用 Pascal 方式时，所有的外部标识符都应改为大写，且不冠以下划线，于是：

L3 行应改为：PUBLIC MAX_VAL

L4 行应改为：MAX_VAL PROC NEAR

L5 行应改为：MAX_VAL ENDP

修改后的 MAXASM.ASM 程序如下：

```
_TEXT       SEGMENT WORD PUBLIC 'CODE'
            ASSUME CS: _TEXT
            PUBLIC _MAX_VAL
MAX_VAL     PROC NEAR
            push  bp
            mov  bp, sp
            xor  ax, ax
            mov  ax, WORD PTR  [bp+6]
            cmp  WORD PTR  [bp+4],   ax
            jle  Loop1
            mov  ax, WORD PTR  [bp+4]
```

```
Loop1:      mov  sp, bp
            pop  bp
            ret  4
MAX__VAL    ENDP
_TEXT       ENDS
END
```

用 / Fa、 / Gc 编译选择项即按 Pascal 方式由 MAXC.C 生成的 MAXC.ASM 程序清单如下:

```
            ; Static  Name  Aliases
            ; TITLE  maxc.c
            .8087
INCLUDELIB SLIBCE
_TEXT       SEGMENT WORD PUBLIC 'CODE'
_TEXT       ENDS
_DATA       SEGMENT WORD PUBLIC 'DATA'
_DATA       ENDS
CONST       SEGMENT WORD PUBLIC 'CONST'
CONST       ENDS
_BSS        SEGMENT WORD PUBLIC 'BSS'
_BSS        ENDS
DGROUP      GROUP CONST, BAA, DATA
            ASSUME DS:DGROUP ,SS:DGROUP
EXTERN      ____artused:ABS
EXTERN      __printf:NEAR
EXTERN      ____aNchkstk:NEAR
EXTERN      MAX__VAL:NEAR
_DATA       SEGMENT
$ SG169     DB '%d', 00H
_DATA       ENDS
_TEXT       SEGMENT
            ASSUME CS:_TEXT
;Line 1
;Line 4
            PUBLIC MAIN
main        PROC NEAR
            push bp
            mov bp,sp
            mov ax,2
            call ____aNchkstk
            ; = lmax= -2
```

```
;Line 7
                mov ax,15
                push ax
                mov ax,30
                push ax
                call MAX__VAL
                push ax
                mov ax, CFFSET DGOUP:$ SG169
                push ax
                call __printf
;Line 8
                mov sp, bp
                pop bp
                ret
MAIN            ENDP
_TEXT           ENDS
END
```

在编译连接的第一步也需改动，需加入 / Gc 编译选择项（即 Pascal 方式选择项）：

C> CL / Fa / Gc / AS　MAXC.C

或者直接在 MAXC.C 程序中函数 max__val 原型定义前加修饰符 Pascal：

extern int pascal max__val (int p1, int p2);

（二）Turbo C 环境

下面介绍在 Turbo C 集成开发环境下编译、连接与执行程序的过程。假设主程序文件名为 SHOW.C，汇编子程序文件名为 SH.ASM，其步骤为：

(1) 利用宏汇编 MASM 生成 SH.ASM 的目标代码 SH.OBJ 文件。

(2) 在 Turbo C 的集成开发环境的 EDIT 状态下，建立一个工程文件 SHOWCH.PRJ。SHOWCH.PRJ 内容为：

SHOW.C

SH.OBJ

(3) 由于 Turbo C 对大小写字母非常敏感，而 MASM 汇编成的目标代码均为大小写字母，故须把集成开发环境下 LINKER 选择项中的 CASE SENSITIVE LINK（大小写敏感连接）开关置成关闭 (OFF) 状态。

(4) 在 Turbo C 集成开发环境下，从 Project name 为 SHOWCH.PRJ，激活此工程文件。

(5) 在 Turbo C 集成开发环境下，从 Compile 选择项中选择 Build all，即生成了 SHOWCH.EXE 可执行文件。

运行 SHOWCH.EXE，屏幕上将会依次输出 A～J10 字符。

同样，读者可试着用 Pascal 语言调用方式修改程序，来加深对调用约定的理解。

【例 2】在 Turbo C 2.0 环境下实现将一串字符送键盘缓冲区，然后显示在屏幕上。单