HOPE

（共二册）

Microsof C6.0 技术丛书

高级程序设计技术

邹然军 等译
吴 双 审校

中国科学院希望高级电脑技术公司

2

共二册

Microsotf C 6.0

技 术 丛 书

—— 高级程序设计技术

配套的书还有

- 最新Microsoft Quick C使用大全
  （V2.5版，共四册）
- 最新Quick C Microsoft C高级编程指南
  （共 二 册）

欢迎广大朋友选购

2

中国科学院希望高级电脑技术公司

一九九一年十二月

简　介

《高级程序设计技术》叙述的是怎样利用Microsoft新的集成开发环境（Microsoft程序员工作台）和源程序级调试工具（Code View调试程序）来最大限度地使用Microsoft C专用开发系统。

通过阅读本手册，你就会明白Microsoft C专用开发系统各组成部分之间是怎样协调工作以提供目前功能最强的开发环境。专用开发系统功能是否强大，关键在于你是否懂得怎样将专用开发系统进行组合，以适应你作为程序员的个人需要。

本书是按照叙述主题组织的，所以它回答了有关使用Microsoft C 6.0版的问题而没有提供可选项列表。如果你对Code View调试程序菜单项、程序员工作台包括专用开发系统在内的命令行实用程序有特殊疑问，可利用Microsoft C顾问(在线帮助)或《C参考手册》一书。

《高级程序设计技术》将使你看到工具和实用程序是如何组织在一起的。

本书内容

《高级程序设计技术》划分成四个部分。第一部分“改进程序性能”帮助你编写更为有效的程序。这部分包含关于优化的特殊内容，即：何时、为何采用不同的优化选择项。在这里还对新的存贮器管理选择项及用法进行了解释。例如，第三章叙述了内嵌汇编程序，该新特征可用来将汇编语言和C源代码混在一起。

第二部分“提高程序员的编程效率”会使你编程更快更有效。第八章解释你组合新程序员工作台（PWB）的不同方式——编辑程序和集成开发环境使你能够：

①建造新程序　②修改现有程序　③浏览源文件　④获得关于PWB、C语言和C运行库的帮助　⑤设置程序建立表　⑥建立程序　⑦用CodeView调试程序来调试程序

第八章叙述了怎样利用键盘指派、记录或编写宏和编写C扩展来改变PWB以适应你的程序设计风格。

在第二部分中花了一个章节来讲解Microsoft程序维护实用程序NMAKE。NMAKE是一个新的程序维护实用程序。它以程序清单作为输入，在程序建立过程中带来了更大的灵活性。它是Microsoft XENIX MAKE实用程序的超集，而且比MAKE的以前的版本功能要强得多。

第二部分第九章描述了Codeview调试程序，该程序比以前的版本在功能上要强。Code View的3.0版具有许多新特征，如，它可以记录下调试过程，然后供回顾之用（历史及动态回顾）。

第三部分是“特殊环境”，描述了新的图形功能。它还示范了用混合语言编程的方法和缩短程序的方法。Microsoft C可协助你方便地创建应用程序。Microsoft C的运行库包含的图形函数用于低级图形操作，如画线、矩形和圆。库中还包含了用来创建表这种形（如讲形图和直方图）的函数。

第四部分是“对OS/2的支持”，描述了专用开发系统如何帮你建立OS/2应用程序。第四章用三个章节的篇幅叙述双模式应用、创建多线索应用和动态连接库。

符 号 的 定

注意：下文中用述语“OS/2”表示OS/2系统——Microsoft操作系统/2（MS OS/2）以及IBM OS/2。同样，述语“DOS”指MS-DOS和IBM个人计算机DOS操作系统。当需要指出系统特有的特征时，则使用一操作系统的具体名字。

举例	解释
STDIO.H	大写字母表示用在DOS或者OS/2命令级的文件名、段名、寄存器和述语。
_cdecl	黑体字母表示C关键字、操作符、特定语言字符库函数及OS/2函数。在讨论语法范围内，黑体则表示必须如所示输入。
[可选项]	双方括弧中的项是可选项。
#pragma pack{1|2}	大括弧和垂线表示从两或多个项中选一项。除非大括弧外套方括弧，否则就必须选一项。
CL A.C B.C C.OBJ	这种字体用于举例、用户输入、程序输出和文本中的错误信息。
CL可选项[文件…]\	在某项后的水平省略号表示后面可能命跟着更多具有同样形式的项。

while ( )
{
.
.
.
}
	垂直省略号表示举例程序部分是有意省去的。

CTRL+ENTER	小体的大写字母表示键盘上的键名。在两键名之间若带有加号（+），则表示应按下第一个键不放手，再按下第二个键。回车键（有时以弯箭头形式出现在键盘上，叫做ENTER。光标移动键（有时又叫方向键）称为ARROW键。单个键用它们的方向（LEFT，UP）或键名（PGUP）表示。
“变量”	在书中第一次定义的述语用引号括起。
增强型，图形适配器（EGA）	第一次使用某个首字母缩略语时，往往将其全称拼写出来。

目　　录

第一部分　改进程序性能

第二部分　提高程序员的效率

第三部分 特殊环境

第四部分 OS/2支持程序

第一部分　改进程序性能

Microsoft C专用开发系统帮助你使用其高级优化程序和增强型存贮器管理功能。

第一章谈谈何时采用某种优化以及Microsoft C怎样产生执行速度快、长度短的代码。第二章叙述了Microsoft C为你提供用来分配和管理程序存贮器（包括新的_based类型）的高级工具。如果程序需要局部优化，可采用第三章描述的内嵌汇编程序。这样可生成最紧凑的代码。如果应用中需要浮点数学运算，你会发现第四章在解释Microsoft C数学软件包可选项方面是很有帮助的。它解释了哪个浮点可选项产生最快、最小、最灵活的代码。

第一章　优化C程序

Microsoft C编译程序将C源语句翻译成机器可执行的指令。另外，汇编程序重写或者"优化"程序中某些部分，使之更为有效，而在源代码级看不出来的。

编译程序执行的优化有三大类型：

1．修改或移动部分代码，以便使用更少的指令或更为有效地使用处理器。2．移动代码并合并运算，以便最大限制地利用寄存器。因为对存放在处理器寄存器中数据的运算远远快于对存在存贮器中数据同样的运算。3．删除多余的和无用的代码。

本章介绍了控制Microsoft C编译程序优化代码的各种途径。

1.1 从程序员工作台控制优化

程序员工作台（PWB）是一个集成开发环境，用来编辑、建立和调试Microsoft C编写的应用程序。关于PCW更详尽的内容，参阅《安装和使用Microsoft C专用开发系统》。

在程序员工作台内有两种方式编译：

1．Debug编译。若缺省，编译程序不执行任何优化。

2．Release编译。若缺省，编译程序进行最佳优化。

用修改C Global Build、Options、C Debug Build Option和C Release Build Options（在Options菜单）的方式，可使能或非使能编译程序执行的任何优化，这样可进行细微优化。

在Build Options每个对话框的每个优化对应于CL的一个命令行可选项。（实际上，PWB从你的输入建立一命令行传递给CL。）

注解：在本章中，优化可选项的讨论是按照优化的效果、调用优化的命令行可选项和控

制优化的杂注（pragmas）进行的。所有这些优化均可在Build Options对话框的编译单位（文件）级控制。

1.2 从命令行控制优化

要想从命令行控制优化，你必须先确定应用程序需要哪种优化。然后用以/O（有些情况下用/G）打头的命令行可选项说明这些优化。

如果可选项之间发生冲突，编译程序就采用命令行说明的最后一个可选项。命令行

```
CL /Oa /O1 /Ot TEST.C
```

对程序TEST.C编译。它说明编译程序可以：

1) 优化的前提是未做别名使用（/Oa）
2) 进行循环优化（/O1）
3) 进行加速总优化（/Ot）

前述的命令行也可写作：

```
CL /Oalt TEST.C
```

1.3 用编译指令控制优化

有时，你需要更精细地控制优化范围。命令行可选择使你能够以整个编译单位（文件）控制优化。另外，Microsoft C支持几种编译指令，使你能在每个功能的基础上进行控制。

本章所描述的控制优化的编译指令可以分为以下类型。

在C6.0中，每个优化指令都使用Optimize，跟随一个优化开关。

- 关于别名的优化（a和w）
- 清除局部公用子表达式（c）
- 清除全局公共子表达式（g）
- 全局寄存器的分配（e）
- 循环优化（l）
- 猜测式优化（z）
- 关闭不安全的优化（n）
- 浮点结果的一致性（p）
- 生成更小的代码或更快的速度（t）

任何优化或可选项的结合均可用optimize编译指令控制。例如，你要对某个函数使用，这时如果进行无别名优化，就可能出问题，你这时不需要使用这个优化选择。为此，按下述使用optimize：

```
/* Function (s) that do not do aliasing.*/
.
.
.
#pragma optimize ( "a" , off)
/* Function (s) that do aliasing.*
```

```
.
.
#pra ma optimize ( "a" , on)
/* More function (s) that do not do aliasing.*/
```

Optimize的参数可以放在一个字符串中。例如，

```
#pragma optimize ( "lge" , off)
```

将禁止循环优化、全局公共子表达式优化和全局寄存器分配。

1.4 缺省优化

如果没有明显地在命令行上禁止优化（/Od），编译器会进行缺省的优化。它们包括：

- 小范围公共子表达式的删除
- 无用存储删除
- 常量传播

1.4.1 公共子表达式删除

在公共子表达式删除中，编译器如果在程序中找到重复的子表达式，就会修改代码，使公用的表达式只计算一次。公用子表达式的例子如下：

```
a=b+c*d;
x=c*d/y;
```

上面两行中都包含公共子表达式c * d。编译程序只计算一次c * d，并把结果放在一中间变量显（通常是一个寄存器）：

```
tmp=c*d;
a=b+tmp;
x=tmp/y;
```

1.4.2 无用存储的删除

无用存贮的删除是公共子表达式删除的推广。在小范围内的变量如果值不变，就被合并到临时变量中。

在下列代码段中，编译程序检测出表达式func(x) 等价于func(a+b)：

```
x=a+b;
x=func (x) ;
```

这样，编译程序可重写代码如下：

```
x=func (a+b) ;
```

1.4.3 常量传播

做常量传播时，编译程序分析变量赋值并确定是否可将它们改变成常量赋值。在下面的例子中，先把7赋给i，再把i赋给j：

```
i=7;
j=i;
```

它被改成：

```
j=7;
j=7;
```

尽管可以在源文件中做这些修改，但这样做可能会降低程序的可读性。在许多情况下，优化不仅提高了程序的效率，而且允许你写出易读的代码。

在某些情况下，你甚至想关闭缺省优化。因为优化可能在目标文件中重新组织代码。在调试中，你的某些代码可能就难以辨认了。因此，在使用符号优化程序之前通常最好去除一切优化。可以用/Od（关闭优化）达到这个目的。

对函数来说，可以用语句#pragma optimize (`` '', off) 关闭一切优化。要将优化恢复成以前状态，用语句#pragma optimize (`` '', on)。

1.5 调整优化

对许多应用来说，缺省的优化就足够了。但有时你还要更进一步地进行优化。优化任选项可以使编译器达到这个目的。

1.5.1 选择速度或尺寸（/Ot和/Os）

除缺省优化外，Microsoft C编译程序还自动使用/Ot可选项进行速度优化。任选项/Ot提高速度，但也可能增加程序的尺寸。如果你宁愿面向缩短程序长度进行优化，就采用/Os任选项。/Os任选项能缩短程序尺寸，但也可能会减慢程序的速度。

如果以函数为单位控制速度和尺寸，就使用带t的optimize。值为on表示进行速度优化；值为off表示生成更小的代码。。例如，

```
#pragma optimize ( "t" , off)        /* Optimize for smallest code.*/
.
.
.
#pragma optimize ( "t" , on) *       /* Optimize for fastest code.*/
```

1.5.2 生成内部函数（/Oi）

在某些常规函数调用的位置，C编译程序可以插入运行更快的“内部函数”。每执行一次函数调用，必须执行一组指令来存贮参数并为局部变量保存空间。在函数返回时，也要用一些指令释放局部变量和参数使用的空间，并将值返回到调用程序。这些指令的执行需要时间。对正常大小的函数来说，这些指令影响不大，但如果函数仅有一两行，这些指令几乎会占用函数一半的代码。

为防止这种类型的代码扩展，一种办法是不编写这样的短函数，尤其在速度关键的经常使用代码部分。不过许多库函数仅有一两行代码。编译程序提供两种库函数形式。一种是标准C函数，它进行函数调用的所有操作。另一种完成同样操作，但却不需要函数调用。第二种形式称为内部函数。内部函数总是比标准的函数调用快，它在优化时非常有用。

例如，函数strcpy实际上为：

```
int strcpy (char *dest, char *source)
{
```

```
        while (*dest++=*source++)
    }
```

编译程序含有一内部形式的strcpy。如果你指示了编译程序生成内部函数，任何对strcpy的调用均可用这个内部形式代替。

注解：尽管上例用C写的程序是为清晰起见的，大多数库函数采用汇编语言书写，以便充分利用80×86指令集的功能。内部函数不是定义成宏的简单库函数。

用任选项/Oi编译会使编译程序使用下列函数的内部形式：

abs	labs	outp	strcpy
_disable	lrotl	outpw	strlen
_enable	lrotl	rotl	strset
fabs	memcmp	rotr	
inp	memcpy	strcat	
inpw	memst	strcmp	

下列浮点函数没有真正的内部形式，虽然它们不使用标准的栈传送参数，但却直接向浮点芯片传送数据。

acos	fmod	acosl	fmodl
asin	log	asinl	logl
atan	log10	atanl	log10l
atan2	pow	atan2l	powl
ceil	sin	ceill	sinl
cos	sinh	cosl	sinhl
cosh	sqrt	coshl	sqrtl
exp	tan	expl	tanl
floor	tanh	floorl	tanhl

警告：编译程序执行优化的前提是数学内部函数无副作用。如果你编写了自己的matherr函数，且该函数影响了全局变量，就要使用function编译，使编译器不生成内部函数。

当你不需要前面所列的所有函数都采用内部形式时，就用 intrinsic 代替任选项/Oi。编译指示intrinsic的形式如下：

```
#pragma intrinsic (function) ...
```

如果你想大部分函数都用内部形式，只有少数函数用标准形式，就在编译时用/Oi而在源文件中使用function编译指示。它具有下列格式：

```
#pragma function (function...)
```

下列代码显示了intrinsic编译指示的用法：

```
#pragma intrinsic (abs)

void main (void)
{
```

```
        int i, j;

        j=ibg_routine_1 ( ) ;
        j=abs (i) ;
        big_routine_2 (j) ;
}
```

这个程序中的abs函数用一段汇编代码计算。由于没有函数调用的开销，所以程序执行更快。

在上面的例子中，速度的损失不是很明显，因为只调用了一次abs。在下面的例子中，对abs的调用处在循环中，且被调用很多次，使用内部函数则可节省大量执行时间。

```
#pragma intrinsic (abs)
void main (void)
{
int i, j, x;
        for (j=0; j<1000; j++)
        {
             for (i=0; i<1000; j++)


                  x=abs (i-j) ;


    {
    printf ( "The value of x is %d\n" , x) ;
 {
```

下面列的是使用内部形式的函数调用的一些限制：

·在使用替换数学库时（MLIBCAY.LIB）不能使用内部形式的浮点函数。

·在Os/2 DLL中不能使用内部形式的浮点函数，因为LLIBCDLL.LIB实际上用了替换数学库。

·若使用/Ox（最大优化）任选项，就等于开放了/Oi（产生内部函数）。所以/Ox不要与上面所列的几点冲突。

注意：内部形式的_enable、_disable、inp、outp、inpw和outpw在OS/2下不能工作。必须使用库形式。可以用function编译指示强制说明。

1.5.3 没有别名（/Oa或/Ow）

“别名”是用来指示已经有另一个不同名字所指存贮器位置的名字。因为存贮器存取比起CPU寄存器存取占用更多时间，编译程序就设法将常用变量存入寄存器。然而，别名的使用减少了编译程序将变量保存在寄存器中的机会。

指针是引用存贮的一种方法。因为指针的值要到程序执行时才能确定，所以编译程序没快知道程序执行时哪些变量将通过指针修改，因此，编译器只好认为指针会修改任何位置的变量。这样使限制了编译程序在存贮器中长时间保存变量。

任选项/Oa告诉编译程序存贮器中没有多个别名。在下面所列几点中，赋值语句的左

右都可以是变量。任何调用函数的变量都使用的是变量本身。如果编译程序假定没有使用别名，任何没用volatile的变量，都必须遵守下列规则：

- 如果变量直接使用，就没有指向它的指针。
- 如果用指针引用变量，就不会直接引用变量。
- 如果一个指针指向某个区域，就不会用其他指针指向同一个区域。

要说清楚这些规则是怎样影响代码的，考虑下面例子：

```
char  p;
char  *ptr_p;

ptr_p=&p;          /* Take the address of p.*/
```

可以使用*ptr_p或p，但不能在同一个函数中使用。如果引用了两个变量名，便是使用了别名。例如，

```
char    *p_buf;
char    *p_alias;

if ((p_alias=p_buf=malloc (5000)) ==NULL)
     return;
else
{
      .
      .
      .
}
```

上面例子中的代码是很常见的。它示范了从堆中动态分配一块存贮器并将原来的地址保存在p_buf中。程序然后在别名p_alias上执行所有的指针算术。当函数用完了存贮器块后，对于free函数，p_buf是一有效参数，因为它仍然包含了原来的地址。

可选项/Oa和/Ow告诉编译程序你没有在代码中使用别名。可选项/Oa和/Ow之间的区别是，用/Oa表示你将不会使用别名（这样编译程序便可执行重要的优化，否则是不可能的）且函数调用是安全的。/Ow选择项类似于/Oa，例外是，函数调用之后，指针变量必须从存贮器重新装入。

下面的程序例子使我们对/Oa或/Ow优化可选项的使用略见一斑：

```
int g;

void main (void)
{
     add_em (&g) ;
}

int add_em (int *p)
```

```
{
    *p=2;          /*Assign a value to an alias for g.*/
    g=3;           /*Assign a value directly to g.*/
}
```

在函数add_em中，g和*p均指示同一存贮器位置。该位置先赋以值2，后值3。*p（g的别名）所指之值然后加上g，结果返回主程序。如果你不使用/Oa命令行可选项，编译程序便假定对*p的引用可以和g一样对同一存贮器位置引用，于是就不会用寄存器存放两者的值。不过，假如你说明了/Oa可选项，编译程序假定g和*p引用不同存贮器位置，于是将两者存放在不同寄存器中。在return语句处，g与*p具有不同值，尽管两个别名实际上应该包含同样值。

注意，编译程序是在有限时间内将值保存在寄存器中的。如果一个存贮器位置的不同别名发生在不同函数，它们就不会带来预料不到的结果。没有把握时就不要使用别名。

包含别名的错误难于发现。别名错误最常见的错误是数据的破坏。如果发现全局或局部变量被指定了似乎随机的值，那么采取下列步骤确定是否有优化和别名方面的问题：

- 用/Od（非使能优化）编译程序。
- 如果用/Od编译后程序能工作，对于/Oa可选项检查正常的编译可选项。
- 若使用了/Oa可选项，修改编译可选项使/Oa没有说明。

注意：用带a或w可选项的optimize杂注，你可以指示编译程序非使能带有别名使用的不安全代码的优化。

1.5.4 进行循环优化（/Ol）

任选项/Ol允许对循环进行优化。由于循环部分被多次重复，所以它们是优化的主要目标。这些优化都涉及到移动或重写代码以便执行起来更快。

循环优化可用任选项/Ol打开，也可以用Loop_opt编译指示控制。下列语句允许循环优化：

```
#pragma loop_opt (on)
```

而下列行则关闭循环优化：

```
#pragma loop_opt (off)
```

优化的循环中只能包括每次循环都改变的表达式。如果在循环中某个子表达式保持不变，就应该先计算它们。不幸的是，有些子表达式难以发现。优化程序的目的就是在编译时把这些子表达式移出循环体，

例如：

```
i=-100;
while (i<0)
{
    i+=x+y;
}
```

上例中表达式x+y在循环体中不改变。循环优化程序将这个子表达式从循环体中移出，使它只计算一次，这些优化包括移动代码或重写代码，使它们执行得更快。最后得到的代码如

下：

```
i=-100;
t=x+y;
while (i<0)
{
    i+=t;
}
```

当编译程序假定无别名使用时，循环优化更加有效。当然可以在循环优化时不用/Oa或/Ow，当使用了它，会得到更高的效率。

下面的代码段可能有别名使用的问题：

```
i=-100;
while (i<0)
{
    i+=x+y;
    *p=i;
}
```

如果没有用/Oa任选项，编译程序只好假定x或y可能被*p所修改。这时子表达式x+y就不是常量。如果你使用了任选项/Oa，编译程序就会假定*p不影响x或y，作为常量的子表达式就可以提出循环体之外。

注意：用/O1任选项或Loop_opt说明一切循环优化都是安全的。要进一步循环优化，必须使用猜测式优化（/Oz）。/O1和/Oz组合可以满足大部分程序的要求，但并不总是安全的。

1.5.5 禁止不安全的循环优化（/On）

关闭不安全的循环优化（/On）是一个已废弃的任选项，保留它的目的是与旧版本的兼容。缺省情况下的循环优化是安全优的。/On作为缺省值与/Oz作用相反。

1.5.6 允许猜测式优化（/Oz）

编译程序可进行极主动的优化，这种优化在速度和长短方面均产生高效的代码。然而某些程序不适合使用这种技术。对于它们就不能用任选项/Oz，但你仍然可以使用其它优化方法。

由于猜测式优化/Oz采用的方法非常富有进攻性，所以它们不属于最大优化（/Ox）的一部分。

下面用实例说明/Oz项的效果：

·循环优化（/O1）。循环优化使用的方法是分析程序流程，并在循环中提取不变表达式。当使用（/Oz）时，编译器在提取不变的表达式后可能会出错，这种错误常常导致循环体中的if语句所保护的条件出错。不变的表达式被提出循环体外，导致保护它的if语句无法检测错误。下面是两个引起这类问题的例子：

```
for (i=0; i<100; ++i)
    if (float_val!=0.0F)
```