第一章 AutoCAD 应用程序的 C 开发环境

1.1 引 言

AutoCAD Development System(ADS)是一种开发 AutoCAD 应用程序的 C 语言程序设计环境。对 AutoCAD 而言,使用 C 语言编写基于 ADS 的应用程序与使用 AutoLISP 语言来编写程序的效果相同。C 语言程序运行速度较 AutoLISP 快,且功能强大,它也可以被 AutoLISP 调用,因此,在速度和效率方面要求高的应用程序一般用 C 来编写。

如同标准 C 函数库一样,ADS 环境由库函数和头文件所定义,它几乎可以在所有支持 AutoCAD 的平台上使用。开发 AutoCAD 的 C 应用程序,除了引用 ADS 库函数外,还必须引用当前开发工具(如 Turbo C、High C)的库函数。

熟悉 C 语言,熟悉 AutoCAD,了解 AutoLISP 对开发 ADS 应用程序是必要的。如果您刚开始学习生成 ADS 应用程序,建议从 AutoCAD 软件包提供的实例开始,循序渐进地熟悉建立新应用程序的方法。

1.2 一般原理

建立一个成功的 ADS 应用程序,且适应于各种平台的要求,须做到如下几点:

- 在一个应用程序中,必须包含有头部文件"adslib.h"。

应用程序中必须包含有一条预处理指令,该预处理指令将指示编译程序,把预处理指令所指的 ADS 头部文件"adslib.h"嵌入预处理指令所在的程序中。

- 一个应用程序必须与 ADS 库进行链接。

在用编译程序编译源程序,或者在随后使用链接程序进行链接时,都必须使应用程序与 ADS 目标文件库相连。

- 应用程序能够并且必须使用 C 语言的标准函数库。

由 ADS 定义的函数、类型和变量,与 C 语言和标准库函数之间不会有冲突。

- 应用程序(经编译链接后)目标代码,必须在装有数学协处理器的机器上运行。只有带数学协处理器的机器支持 AutoCAD 软件包。如果您使用的编译程序依赖于数学协处理器的代码生成能力,那么就必须使用数学协处理器。

1.3 640K DOS (AutoCAD 286)环境

在 640K 内存的 DOS 环境中,ADS 应用程序是作为 DOS 的可执行文件而编译的,其运行文件扩展名为.EXE。在 AutoCAD 12.0 环境中,应用程序一装入内存就立即执行。其主要开发平台如下:

- 编译程序(compiler)

编译程序使用 Microsoft C 5.0 版(或更高的版本),Turbo C 或者 MetaWare 的 High c 1.6 版(这是一种 16 位的编译系统)。

- 链接程序(Linker)

链接程序使用 Microsoft 的覆盖型链接程序(Link)。

编译批处理可在 AutoCAD R12.0 软件中找到。

由于这种应用程序不能充分利用内存,它生成的是 16 位运行程序,而现在硬件发展很快,微机的 CPU 已到 586,因此这里不再详细讨论其编译和链接方法。

1.4　DOS Extender (AutoCAD 386)环境

在 DOS Extender 环境中(386 以上微机),ADS 应用程序作为保护模式下的可执行文件而编译链接的,其扩展名为.EXP。在 AutoCAD R12.0 环境中,应用程序一装入内存并不立即执行。

在 DOS Extender(386)环境,要求如下软件工具:

- 编译程序(compiler)

编译程序使用 MetaWare High C 或 WATCOM C .

- 链接程序(Linker)

链接程序使用 Phar Lap 386 | Link

- 调试程序(Debugger)

调试程序使用 Phar Lap 386 | DEBUG

- 虚拟存储管理程序(Virtual memory manager)

虚拟存储管理程序使用 Phar Lap 386 |VMM

- 汇编程序(Assembler)

汇编程序使用 Phar Lap 386 | ASM。

- 链接程序和汇编程序是作为 Phar Lap 386 |DOS-Extender 软件开发包的一部分提供的,虚拟存储器管理程序和调试程序是该软件开发包的任选项,编译程序是作为一个独立的产品提供的。

1.4.1　设置编译环境

如果你准备了如下的环境变量和目录结构,就能简单地执行所使用的编译程序。如下的环境变量适用于 Meta ware High C 和 WATCOM C 两种编译程序:

环境变量 IPATH 内包含有你的编译程序的包含文件的路径名。

例:set ipath =\highc\inc

环境变量 LIST 指向列表程序文件的路径名。

例:set list=\listings\list1

如下的环境变量仅适用于 Meta ware High C:

HIGHC ——包含有 highc 编译程序的目录和路径名。在该目录下必须有一个名为\library 的子目录,且在\library 子目录下还必须有名为\library\source 的孙目录,其中包含有

随 HIGH C 编译程序一起提供的源文件。

.如下的环境变量仅适用于 WATCOM C 编译程序：

WATCOM ——包含有 WATCOM C 编译程序的目录的路径名。在该目录下必须有\h，\bin，\src 和 lib 四个孙目录。

386INC ——包含有 WATCOM C 编译程序的包含文件的目录路径名。

386LIB ——包含有 WATCOM C 的库的目录文件的目录路径名。

1.4.2 用 MetaWare High C 进行编译

为编译源代码，在命令行执行 hc 386 命令，编译一个 ADS 应用程序，需在命令行中指定一定的任选参数。下面是编译 example.c 的编译命令行实例：

```
hg386 example -ip\highc\inc\ -pr ads386.pro -ob example.obj
```

在以上实例中，选项参数意义如下：

-ip \highc\inc\ 告诉编译程序在什么地方查找它的标准定义文件。如果 IPATH 设置正确，则不需该选项。

-pr ads386.pro 指定预置文件 ads386.pro(必须自己生成)，该选项可不要。

-ob example.obj 指定目标文件的文件名，该选项也能重定向目标文件于另一目录中。

- 生成预置文件(profile)

如果在编译命令行参数中指定了预置文件名，那么，编译程序在读源代码之前要读取预置文件。通常，预置文件用于设置编译条件限定和杂注，预置文件中至少应包含如下说明：

```
# define HIGHC 1
pragma On (Floating-point)
```

宏定义 #define 直接定义了由 ADS 头文件 adslib 所使用的符号 HIGHC。第二行是杂注，用于接通数学协处理器的代码生成能力。

1.4.3 用 WATCOM C 编译程序进行编译

为了编译 C 语言源代码，可在命令行中调用 WC386 命令。编译一个 ADS 应用程序时，要求在命令行中指定一定的选项。下面是编译 example.c 的编译命令行实例：

```
WC386 example /d1 /7 /ml
```

在以上实例中，选项参数意义如下：

/d1 告诉编译程序为调试程序(dcbugger)生成行号信息。

/7 告诉编译程序生成由数学协处理器使用的指令。

/ml 规定由编译程序生成的代码所使用的模式。在本例中为大型模式，ADS 应用程序现仅支持使用存储器的大型模式。

1.4.4 用 Phar Lap 386|ASM 汇编程序进行汇编

运行 386|ASM 汇编程序的命令 386asm，可以汇编用汇编语言书写的源代码。需要注意的是：必须确保汇编语言程序的代码段有一个类别指定符 CODE，数据段有一个类别指定符

DATA。任何其他的类别名将会使得对应的段装入到堆的中部，当试图在 AutoCAD 软件包中运行该程序时，将会产生不良后果。

1.4.5 用 Phar Lap 386 |Link 链接程序进行链接

目标文件经过 386|LINK 链接程序的链接后就可产生可执行的程序，链接是通过执行 386Link 命令来完成的。下面是一个实例：

```
386 link ads. lib example. obj doe. obj -lib \highc\hce -exe exa
```

执行上述命令后生成的可执行文件为 exa. exp。各选项意义如下：

-lib \highc\hce　　指定库文件名

-exe exa　　指定可执行文件名

需要指出的是：链接程序的首趟扫描必须是扫描模块 ads. lib，因为该库所包含的初始化代码必须出现在可执行文件的映像的开始处。

1.4.6 调试说明

环境变量 DOSX 用于从 DOS Extender 中，设置调试信息的等级。

```
SET DOSX=DEBUG Y
```

式中 y 为一整数，范围 1～5，通常取 3。

环境变量 EXPDEBUG 应设置为 ADS，SET EXPDEBUG=ADS。

当用 386|DEBUG 调试程序调试应用程序在 AutoCAD 环境中的运行情况时，除了指定应用程序名，AutoCAD 软件包的主执行程序名(即 acad. EXE)外还要指定-symfile 开关(该开关也可缩写为-sf)以便指向你的应用程序。开关-sf 告诉 386|DEBUG 调试程序，使用被调试程序中的符号表，像所有 386|DEBUG 调试程序的其他开关那样，这个开关也必须置于可执行文件的前面。假设被调试程序名为 adsapp. exp，那么调试命令行格式如下：

```
386debug -sf adsapp acad. exe -minswfwize 4000000 -swapdefdisk -swapchk off -intmap 8 -vscan
20000
```

命令行中附加开关的含意，请参见 Phar Lap 386|DEBUG 调试手册。

需要指出的是：在调试前，必须检查 Debugger 环境文件是否如下设置：

- KEEPHEAP 至少为 8 个 4K 页
- FILP 必须设置为 ON
- MAIN 必须设置为 OFF

详细的调试方法请参看调试手册。

1.4.7 内存管理说明

在 DOS Extender 环境，AutoCAD 软件包仅仅使用由 386|VMM 管理的与虚拟控制程序接口(VGPI)相一致的扩充内存。如果你要使用 386|VMM，就应该使用 386|VMM 的 MAXVCPI 开关将所有可利用的内存分配为扩充存储器，一点都不要将内存作为扩展存储器。

1.5 有关 High C 的一些说明

目前，国内大多使用 High C 来开发 32 位的 AutoCAD 应用程序。本节对 High C 作一些简单介绍。

High C 是为高效的专业化编程所设计的语言，它适用于多种处理器，包括配置 80387 数字协处理器的 80386 微处理器。High C 支持 ANSI 标准及其一些增补。

High C 提供了一些新的由标准 C 所指定的强有力的类型检查，提高了程序的安全性和有效性。

重要的 High C 语言扩充功能有：

- 嵌套函数由顶层寻址完成，如同在 Pascal 中一样。
- 嵌套函数可作为参数传递给其他函数，如同在 Paseal 中一样。
- 允许内部函数如_abs，_min 和_max 的使用。
- 重述：一种横跨数据结构的有力结构。
- 在线库函数调用，如 strlen()，memset()等。
- 通过函数调用访问物理层的指令，如_inb()和_outb()。

对于 high C 编译器，具有如下重要性质：

- 支持 Intel 80387 和 Westek ABAcus 协处理器。
- 支持嵌入式应用程序库的可固化于 ROM 的代码。
- 为应用程序初始化段。
- 支持近、远目标和指针。
- 三种整数范围三种 IEEE 浮点精度数。
- UNIX 类型的 I/O 调用。
- 类似"亚麻布"式的诊断以及可将其关闭的 PCC_msgs 开关项。
- 支持 AT &T 的便携式 C 编译器(PCC)和 PCC 开关项以及-Hpcc 命令行选择项间兼容性。
- 支持许多种编译器选择项和控制项，包括一种严格的标准检查。
- 大量的警告信息——可以使用开关项打开或关闭。

多种优化操作，其中一些只有在很高级的编译器中才有，它们包括：

- 全局溢出分析
- 死/活分析
- 全局命令子表达式删除
- 全局寄存器分配
- 死代码删除
- 跳转指令代码长度的最小化
- 常量封固(folding)
- 常量及其副本繁殖
- 数字长度最小化
- 变量到寄存器的自动映像

- 交叉跳转(结尾合并)
- 从循环中移出不变表达式

在 High C 中，杂注(pragma)用于控制编译器开关项的设置、对外部函数的链接、数据存储、存储器模型、源程序的包含和列表以及其他一些功能。每一项杂注都有默认值，用户可使用轮廓描述文件来替代这些默认值以使编译器与当前环境匹配。

编译器工作时需要至少 1.5M 的可用 RAM，而 Phar Lap 运行环境大约要占用 170K DOS 的内存空间。

High C 支持 C 字符串中的双字节的汉字字符。为确保这种支持，需要在驱动程序命令行上或设备驱动程序配置文件中说明 HKanji。High C 的库，尤其是字符串函数，不支持汉字。

下面讨论一下 High C 的存储器模型。

80386 存储器具有一个分段式体系结构。一个段是一个存储器区域，此处段寄存器的内容是固定的。其寻址是相对于这段寄存器的位移，一个段可大至 4G 字节。

段寄存器共有 6 个：

cs——代码段
ds——数据段
ss——堆栈段
es——附加段
gs——附加段
fs——附加段

代码存取有两种不同方式的模型：小和大。

数据存取有三种不同方式的模型：小、中等和大。

存储器模型是数据存取和代码存取模型的特定组合，根据不同的组合，细分为：小(Small)、紧缩(Compact)、中等(Medium)、大(Big)和超大(Large)，这就是 High C 的存储器模型，由于一个 80386 的段可以大到 4G，High C 编译程序只支持小(small)存储器模型，这跟我们使用 Trubo C，Microsoft C 等编译器中的小模式(small)含义是不同的，传统上所讲的小模式是 64K。

还有一点需要指出的是，只要内存足够，High C 可以分配多达几兆的数组，但这个数组必须作为全局变量，也就是说只有全局堆才能分配大空间，局部堆不能分配大空间。使用 malloc()等函数进行动态分配时不受此限制。

第二章　ADS 应用程序的基本框架

每一个 ADS 应用程序必须能支持由 ADS 环境所定义的 AutoLISP 的接口。接口程序要求应用程序按一定的次序，使用确定的值调用确定的 ADS 库函数。下面给出一个典型的 ADS 应用程序。

2.1　一个典型的 ADS 应用程序

```
/* example.c */
#include <stdio.h>
#include "adslib.h"
#define ELEMENTS(array) (sizeof(array)/sizeof(array[0]))
        int fact(struct resbuf *);
        int squareroot(struct resbuf *);
        int funcload();
        int dofunc();
        ads_real rfact(int x);
        ads_real rsq(ads_real x);
        struct{
            char *cmdname;
            int (*cmdfun)();
        } cmdtab[] = {
        {"fact", fact}, {"sqr", squareroot},
        };
        // MAIN--the main routine
        void main(int argc, char *argv[])
        {
            int stat;
            short scode=RSRSLT;
            ads_init(argc, argv);
            for (;;) {
                if ((stat=ads_link(scode))<0) {
                    ads_printf("Error when run ads_link, code=%d\n", stat);
                    exit(1);
                }
                scode=RSRSLT;
                switch(stat) {
                    case RQXLOAD:
                        scode=funcload()==RTNORM? RSRSLT:SRSERR;
```

```
                    break;
                case RQSUBR:
                    scode=dofunc()==RTNORM? RSRSLT:RSERR;
                    break;
                default:
                    break;
            }
        }
    }
//FUNCLOAD
    static int funclod()
{
}
static int funcload()
{
    int i;
    char ccbuf[40];
    for (i=0; i<ELEMENTS(cmdtab); i++) {
        strcpy(ccbuf, cmdtab[i].cmdname);
        if (! ads_defun(ccbuf,i))
        return RTERROR;
    }
    return RTNORM;
}
//DOFUN
static int dofunc()
{
    int value;
    struct resbuf *rb;
    if ((value=ads_getfuncode())<0||value>=ELEMENTS(cmdtab)) {
        ads_fail("UNKNOWN:Received non-existent function code. \n");
        return RTERROR;
    }
    rb=ads_getargs();
    return(*cmdtab[value].cmdfunc)(rb);
}
//FACT
static int fact (struct resbuf *rb)
{
    int x;
    if(rb==NULL)
    retrun RTERROR;
    if (rb->restype==RTSHORT) {
```

```
        x=rb->resval.rint;
        } else {
        ads_fail("Argument should be an integer.");
        return RTERROR;
    }
    if(x<0) {
        ads_fail("Argument should be positve.");
        return RTERROR;
    }
    ads_retreal(rfact(x));
    return RTNORM;
}
//This is implementation of the actual external factorial function
static ads_real rfact(int n)
{
    ads_real ans=1.0;
    while(n)
    ans *=n--;
    return ans;
}
//SQUAREROOT
static int squareroot(struct resbuf * rb)
{
    ads_real x;
    if (rb==NULL)
    return RTERROR;
    if(rb->restype==RTSHORT) {
        x=(ads_real)rb->resval.rint;
        } else if(rb-> restype==RTREAL) {
        x=rb->resval.rreal;
        } else {
        ads_fail("Argument should be a real or an integer.");
        return RTERROR;
    }
    if(x<0) {
        ads_fail("Argument should be positve.");
        return RTERROR;
    }
    ads_retreal(rsqr(x));
    return RTNORM;
}
//This is the implementation of the actual external function
```

```
static ads_real rsqr(ads_real x)
{
    int n=50;
    ads_real y,c,c1;
    if (x==0.0) return 0.0;
    y=(x*2+1.)/(x+1.);
    c=(y-x/y)/2;
    c1=0.0;
    while ((c!=c1)&&(n-->0)) {
        y=c;
        c1=c;
        c=(y-x/y)/2;
    }
    return y;
}
```

2.2 示例分析

由给出的典型示例可以看出，AutoLISP 是按以下步骤来访问 ADS 应用程序的：

1. AutoLISP 在初始化或通过调用(xload)函数装入 ADS 应用程序。
2. ADS 应用程序通过调用 ads_init()初始化与 AutoLISP 的通信。
3. ADS 应用程序指出它正准备使用一个 RSRSLT 应用程序结果码(result code)来调用 ads_link()，以在 AutoLISP 中处理一个请求。
4. Autoslip 用结果码 RQXLOAD 调用函数 ads_link()实行返回。
5. 每调用 ads_defun()一次，应用程序将定义一个外部函数。
6. 应用程序结果码 RSRSLT 再一次调用函数 ads_link()。
7. 当用户或 AutoLISP 函数要判断它为应用程序的一个外部函数时，AutoLISP 将使用一个 RQSUBR 要求码从 ads_link()返回。
8. 当处理完此外部函数后，此应用程序会使用一个 RSRSLT 结果码来调用 ads_link()。

由上可知，一个 ADS 应用程序完全是 AutoLISP 的"奴隶"，一直处于等待交互状态，直到 AutoLISP 要求它动作为止。另一方面，当 ADS 应用程序正响应 AutoLISP 一个请求的时候，AutoCAD 和 AutoLISP 都处于空转状态，它们等着来自 ADS 函数库的请求，此时它们不响应任何用户的输入。

这个 ADS 应用程序实现了两个数学函数，一个是 rfact()，用以计算阶乘；另一个是 rsqr()，用以计算平方根。

这个应用程序可以作为 ADS 程序设计的一个模板。对于不同功能的 ADS 应用程序，main()稍作变动(改写功能函数定义部分)，funclood()及 dofun()函数都不用变动，重写功能实现函数及其定义，便可生成其他的 ADS 应用程序。

需要指出的是，如果想把 ADS 应用程序设计成 AutoLISP 中的命令，funcload()函数要

稍作改动如下：

```
static int funcload()
{
    int i;
    char cc buf[40];
    strcpy (ccbuf,"c:");
    for (i=0; i<=ELEMENTS (cmdtab); i++) {
        strcpy (ccbuf+2,cmdtab[i].cmdname);
        if (! ads_defun(ccbuf,i))
        return RTERROR;
    }
    return RTNORM;
}
```

这个 ADS 应用程序求阶乘及计算平方根时需要向函数传递自变量，调用 ads_getargs ()函数返回从 AutoLISP 传来的值的结果缓冲器链表的一个指针，在求阶乘及计算平方根的函数中利用这个指针，从结果缓冲区中取出函数自变量的值。语句如下：

```
if (rb==NULL) return RTERROR;          //判断传来指针是否为空;
    if (rb->restype==RTSHORT) {        //判断传来的缓冲区内值的类型;
    x=rb->resval.rint;                 //取出自变量(符合要求)
    }
    else {                             //不合要求,进行出错处理
    ads_fail("Argrment should be an integer.");
    return RTERROR;
}
```

2.3 ADS 应用程序的请求码和结果码

2.3.1 AutoLISP 的请求码

ADS 应用程序在 ads_link()调用中的大部分时间是等待 AutoLISP 回答一个请求码，ADS 应用程序根据 AutoLISP 的请求码，采取相应的动作。

请求码以整数值定义在 adscodes.h 头文件中，所有的请求码如下：

RQXLOAD　　请求定义函数

通知 ADS 应用程序它已被 AutoLISP 装入，它必须定义它的外部函数。应用程序每定义一个外部函数需调用一次 ads_defun()。

RQSUBR　　请求调用一个外部函数

通知 ADS 应用程序，用户或一个 AutoLISP 函数已调用了应用程序的一个外部函数。应用程序必须调用 ads_getcode()，以取得函数的数字代码，然后使用此值选择所要求的函数并调用它。函数码由 ads_defun()调用产生。

RQXUNLD　　请求退出此程序

通知 ADS 应用程序，用户或 AutoLISP 函数已调用(xunload)请求 AutoLISP 退出 ADS

应用程序。在大多数情况下，应用程序将返回一个正常状态值，也可做一些清除工作，如释放动态分配空间，关闭打开的文件，释放选择集等。

RQSAVE AutoCAD 正要保存文件

通知发出 AutoCAD SAVE 命令的 ADS 应用程序

RQEND AutoCAD 正要结束

通知发出 AutoCAD END 命令的 ADS 应用程序

RQQUIT AutoCAD 正要退出

通知 ADS 应用程序，AutoCAD QUIT 命令已被使用。

2.3.2 应用程序的结果码

ads_link()函数有一个 ADS 应用程序，必须用它来指示其所属状态的整型自变量。这个自变必须等于在 adscodes.h 中定义的结果码之一，可能的结果码如下：

RSRSLT 通知 AutoLISP 应用程序已完成它的任务，正等待新的请求。

RSERR 通知 AutoLISP 在 ADS 应用程序中出现了一个错误。如果这个值出现在一个 RQXLOAD 请求之后，那么 AutoLISP 将终止应用程序并退出它。如果这个结果出现在一个 RQSUBR 请求之后，那么 AutoLISP 将取消这个外部函数的请求。

2.4 从 AutoLISP 中访问 ADS 应用程序

1. 装入一个 ADS 应用程序

用户可以用 AutoLISP 中的 xload 函数装入一个编译过的 ADS 应用程序。语法如下：

(xload filename [onfailure])

函数 xload 将搜索由 filename 指定的文件并将它装入内存中，同时立刻执行初始化操作，但并不立刻运行整个程序。

假如装入成功，则返回应用程序的名称字符串；否则给出一错误信息。

用户可同时装入多个 ADS 应用程序，最大值为 255。

2. 列出已装入的 ADS 应用程序

键入 AutoLISP 中的(ads)函数将返回字符串序列，每个字符串是一个已被装入的 ADS 程序名称。

3. 运行 ADS 函数

ADS 应用程序定义于 AutoLISP 中被称为外部函数的函数组中。由于此应用程序已使用 xload 装入，故用户可通过键入一个引用外部函数名的 AutoLISP 语句来运行一个用户定义的外部函数或定义用户的 AutoLISP 函数。AutoLISP 变量可以通过参数传给外部函数，同时外部函数将返回一个结果。

4. 卸载 ADS 应用程序

可使用 xunload 卸载 ADS 应用程序。

语法：(xunload filename)

只有在下述两种情况下 AutoLISP 会自动卸载一个 ADS 应用程序：当应用程序本身通过 ads_abort()函数报告一个致命错误或当用户从 AutoCAD 退出时。

5. 在 AutoCAD 初始化时装入应用程序

初始化时，AutoCAD 会检查其目录中是否有一个名为 acad. ads 的文件。如果有，则它会试图装入每个名为 acad. ads 的文件。

acad. ads 文件是一个文本文件，它应包含一个或多个 ADS 应用程序文件名，每一行都有一个带或不带扩展名的文件名。

第三章 ADS的预定义

3.1 ADS的头文件

ADS有四个头文件，其名称和内容如下：

adslib.h 包含ADS的一般性定义。

adscodes.h 包含ADS函数库所返回(或传递)的程序码值的定义。

ads.h 包含ADS函数库类型定义和函数声明。
函数库函数及大部分数据类型定义是以ads_为前缀。

cl_errno.h 包含AutoCAD系统变量ERRNO所使用的错误码的符号值。

需要说明的是，adslib.h头文件中用#include命令包含了adscodes.h及ads.h两个头文件，每个应用程序资源仅需包含：

```
#include "adslib.h"
```

3.2 一般类型及其定义

1. 实数

AutoCAD中的实数总是双精度浮点数。ads.h中定义了对应的类型来保持此标准，如下所示：

```
typedef double ads_real;
```

用户应该对所有给AutoCAD或AutoCAD传回的实数值使用ads_real变量。

2. 点

AutoCAD定义点为一个数组类型(ads.h中)，如下所示：

```
typedef ads_real ads_point[3];
```

一个点总有3个值，当为二维点时，数组的第3个元素被忽略，最安全的做法是初始化为0值。

为了便于处理点，ads.h文件中还定义了：

```
#define X 0
#define Y 1
#define Z 2
```

必须逐个处理点中的元素；如：

```
newpt[X]=oldpt[X];
newpt[Y]=oldpt[Y]
newpt[Z]=oldpt[Z];
```

也可以通过使用定义于 ads.h 中的 ads_point_set()宏来拷贝一个点值。

由于 C 语言遵循引用传递约定，所以点不必引用地址操作来传递值，而且 C 总是通过一个指向数组第一个元素的指针来传送数组参数的。

示例

ads_osnap()函数将获得一个参数点，并返回一个点作为结束，它被声明如下：

```
int ads_osnap (pt,mode,result)
ads_point pt;
char * mode;
ads_point result;
```

因为是数组，pt 和 result 将自动地被间接传送。

3. 转换矩阵

函数 ads_draggen(),ads_grves(),ads_nentselp()以及 ads_xforms()也可使用一个定义为 4×4 实数值数组的转换矩阵：

```
typedef ads_real ads_matrix[4][4];
```

矩阵的前三行指定其缩放或旋转，第 4 行是矩阵的一个转换向量，ads.h 文件定义符号 T 来作为这个向量的坐标：

```
#define T 3
```

示例

下例函数可初始化一个恒等矩阵：

```
void indent_init(ads_matrix id)
{ int i,j;
  for (i=0;i<=3;i++)
     for (j=0;j<=3;j++)
        id [j][j]=0.0;
fox (i=0;i<=3;i++)
     id [i][i]=1.0;
}
```

下面列出了一些基本几何转换的转换矩阵：

$$\begin{bmatrix} 1.0 & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \end{bmatrix} \quad \begin{bmatrix} 1. & 0. & 0. & Tx \\ 0. & 1. & 0. & Ty \\ 0. & 0. & 1. & Te \\ 0. & 0. & 0. & 1. \end{bmatrix} \quad \begin{bmatrix} Sx & 0. & 0. & 0. \\ 0. & Sy & 0. & 0. \\ 0. & 0. & Sa & 0. \\ 0. & 0. & 0. & 1. \end{bmatrix}$$

恒等　　转换　　缩放

$$\begin{bmatrix} CosQ & -SinQ & 0. & 0. \\ SinQ & CosQ & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \end{bmatrix}$$

2D旋转(XY平面)

$$\begin{bmatrix} 1. & 0. & 0. & 0. \\ 0. & CosQ & -SinQ & 0. \\ 0. & SinQ & CosQ & 0. \\ 0. & 0. & 0. & 1. \end{bmatrix}$$

2D旋转(YZ平面)

$$\begin{bmatrix} CosQ & 0 & SinQ & 0. \\ 0. & 1. & 0. & 0. \\ -SinQ & 0. & ConQ & 0. \\ 0. & 0. & 0. & 1. \end{bmatrix}$$

2D旋转(XZ平面)

4. 实体和选择集名称

在 AutoLISP 中,实体和选择集是一对长整数;在 ads. h 中定义如下所示:

```
typedef long ads_name[2];
```

如同使用 ads_point 一样,ads_name 变量总是通过参数来传送,但必须逐个元素地赋值。

用户也可以通过使用定义于 ads. h 中的 ads_name_set()宏来拷贝一个实体或选择集名称,用 ads_name_equal()宏来比较两个名称。

示例

```
ads_name oldname, newname;
  ads_name_set (oldname, newname);          /* 将 newname 设置为 oldname */
```

可以通过 ads_name_clear()宏强制一名称拥有无效的值,并且通过使用 ads_name_nil()宏来测试一个实体或选择名称是否为定。

5. 只能在扩充实体数据中使用的类型

扩充实体数据可以包含二进制数据,并可组织成不同长度的集合块,这些将由 ads_binary 结构处理:

```
struct ads_binary {                 //Binary date chunk structure
    short len;                      // Length of chunk in bytes
    char * buf;                     //binary data
};
```

其中 len 数据段的值必须在 0 到 127 的范围内。如果应用程序要求的数据多于 127 字节,必须把数据组织成复合块。

扩展的实体数据也能包含长型整数。如果缓冲器的 resval 段的数据组织的集合以 ads_u_val 表示,它用于处理扩充数据,其中既包含了一个 ads_binary,也包含了一个 long 成分。下节将介绍结果缓冲器(区)。

AutoLISP 不会识别这些类型,所以它们不能传递给 AutoLISP 函数。它们可以通过 ads_invoke()函数传送给其他外部函数,但仅当它们属于一个实体扩展数据的组中时。用户不能传送独立的 long 整数或二进制的数据块。

6. 有用的值

ADS 的库函数头文件 adslib. h 中提供了如下的一些有用定义:

```
# define    TRUE 1
# define    FALSE 0
# define    EOS '\0'
# define    PAUSE "\\"          /* Panse in command argment list */
```

3.3 结果缓冲区和类型码

ADS 头文件 ads.h 中定义了结果缓冲区(resbuf)的结构如下：

```
union ads_u_val {
    ads_real rreal;
    ads_real rpoint[3];
    short rint;
    char * string;
    long rlname [2];
    long rlong;
    struct ads_binary rbinary;
};
struct resbuf {
        struct resbuf * rbnext;
        short restype;
        union ads_u_val resval;
};
```

由以上定义可以看出，此结构具有 3 个域：一个为指针域，指向别的结果缓冲区或为空；一个为类型域，该域的值表示了该域存储的值的类型，另一个为值域，为适应不同的数据类型，它定义成联合的形式，可以看出此结构适合于处理实体及符号集。由于结果缓冲区还可以结合成链表，因此它适于处理那些长度可变或包含了混合数据类型的对象，结果缓冲区列表用于说明实体、AutoCAD 符号表的入口和 AutoLISP 的表。大多数 ADS 库函数返回或接收的要么是单个的结果缓冲区，要么是结果缓冲区列表。

深入理解结果缓冲区对于编写 ADS 实用程序具有重要意义。

注意：long 整型段的 resval.rlong 和二进制数据段的 resval.rbinary 一样，只能用来处理扩充实体数据。

3.3.1 ADS 函数库定义的结果类型码

一个结果缓冲区的 restype 字段是一个指出哪个类型的值将被存储在此缓冲区 resval 字段内的 short 整数码，下表列出了结果类型码。