第3章　最简单的C程序设计
——顺序程序设计

有了前两章的基础，现在可以开始由浅入深地学习C语言程序设计了。

为了能编写出C语言程序，必须具备以下的知识和能力：

(1) 要有正确的解题思路，即学会设计算法，否则无从入手。

(2) 掌握C语言的语法，知道怎样使用C语言所提供的功能编写出一个完整的、正确的程序。也就是在设计好算法之后，能用C语言正确表示此算法。

(3) 在写算法和编写程序时，要采用结构化程序设计方法，编写出结构化的程序。

算法的种类很多，可以说是无底洞，不可能等到把所有算法都学透以后再来学习编程序。C语言的语法规定很多、很烦琐，孤立地学习语法不但枯燥乏味，而且即使倒背如流，也不一定能写出一个好的程序。必须找到一种有效的学习方法。

本书的做法是：以程序设计为主线，把算法和语法紧密结合起来，引导读者由易及难地学会编写C程序。对于简单的程序，算法比较简单，程序中涉及的语法现象也比较简单(一般只用到简单的变量、简单的输出格式)。对于比较复杂的算法，程序中用到的语法现象也比较复杂(例如要使用数组、指针和结构体等)。本章先从简单的程序开始，介绍简单的算法，同时介绍最基本的语法现象，使读者具有编写简单的程序的能力。在此基础上，逐步介绍复杂一些的程序，介绍比较复杂的算法，同时介绍较深入的语法现象，把算法与语法有机地结合起来，步步深入，由浅入深，由简单到复杂，使读者很自然地、循序渐进地学会编写程序。

3.1　顺序程序设计举例

例3.1　有人用温度计测量出用华氏法表示的温度(如69°F)，今要求把它转换为以摄氏法表示的温度(如20℃)。

解题思路：这个问题的算法很简单，关键在于找到二者间的转换公式。根据物理学知识，知道以下转换公式：

$$c = \frac{5}{9}(f-32)$$

其中f代表华氏温度，c代表摄氏温度。据此可以用N-S图表示算法，见图3.1。

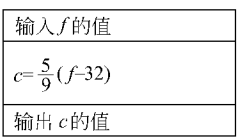

图　3.1

算法由3个步骤组成，这是一个简单的顺序结构。

编写程序：有了N-S图，很容易用C语言表示，写出求此问题的C程序。

```
#include <stdio.h>
int main()
```

```
{
  float f,c;                                    //定义 f 和 c 为单精度浮点型变量
  f=64.;                                        //指定 f 的值
  c=(5.0/9)*(f-32);                             //利用公式计算 c 的值
  printf("f=%f\nc=%f\n",f,c);                   //输出 c 的值
 return 0;
 }
```

运行结果：

```
f=64.000000
c=17.777778
```

读者应能看懂这个简单的程序。

例 3.2 计算存款利息。有 1000 元，想存一年。有 3 种方法可选：(1)活期，年利率为 r1；(2)一年期定期，年利率为 r2；(3)存两次半年定期，年利率为 r3。请分别计算出一年后按 3 种方法所得到的本息和。

解题思路： 关键是确定计算本息和的公式。从数学知识可知，若存款额为 p0，则：

活期存款一年后本息和为 p1＝p0(1＋r1)。

一年期定期存款，一年后本息和为 p2＝p0(1＋r2)。

两次半年定期存款，一年后本息和为 p3＝$p0\left(1+\frac{r3}{2}\right)\left(1+\frac{r3}{2}\right)$。

画出 N-S 流程图，见图 3.2。

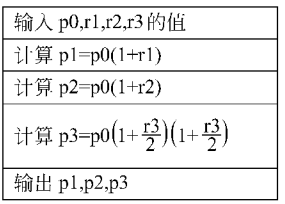

图 3.2

编写程序： 按照 N-S 图所表示的算法，很容易写出 C 程序。

```
#include <stdio.h>
int main()
 {
   float p0=1000, r1=0.0036, r2=0.0225, r3=0.0198, p1, p2, p3;
                                                //定义变量
   p1=p0*(1+r1);                                //计算活期本息和
   p2=p0*(1+r2);                                //计算一年定期本息和
   p3=p0*(1+r3/2)*(1+r3/2);                     //计算存两次半年定期的本息和
   printf("p1=%f\np2=%f\np3=%f\n",p1, p2, p3);          //输出结果
   return 0;
 }
```

运行结果：

```
p1=1003.599976
p2=1022.500000
p3=1019.898010
```

第 1 行是活期存款一年后本息和，第 2 行是一年期定期存款一年后本息和，第 3 行是两次半年定期存款一年后本息和。

程序分析： 第 4 行，在定义实型变量 p0，p1，p2，p3，r1，r2，r3 的同时，对变量 p0，r1，r2，r3 赋予初值。

第 8 行，在输出 p1，p2 和 p3 的值之后，用\n 使输出换行。

注意：在使用 Visual C++ 6.0 编译系统时，对以上两个程序在进行编译时，会显示出“警告”信息：“在初始化时把一个双精度常量赋给一个 float 型变量”。这是因为有的 C 编译系统(如 Visual C++)把所有实数都作为双精度数处理。因此提醒用户：把双精度常量赋给 float 型变量会造成精度损失。对这类“警告”，用户知道是怎么回事就可以了。承认此现实，让程序继续进行连接和运行，不影响运行结果。如果用 GCC 编译系统，则不会出现此“警告”信息。

3.2 数据的表现形式及其运算

有了以上写程序的基础，本节对程序中最基本的成分作必要的介绍。

3.2.1 常量和变量

在计算机高级语言中，数据有两种表现形式：常量和变量。

1. 常量

在程序运行过程中，其值不能被改变的量称为常量。如例 3.1 程序中的 5，9，32 和例 3.2 程序中的 1000，0.0036，0.0225，0.0198 是常量。数值常量就是数学中的常数。

常用的常量有以下几类：

(1) **整型常量**。如 1000，12345，0，－345 等都是整型常量。

(2) **实型常量**。有两种表示形式：

① 十进制小数形式，由数字和小数点组成。如：123.456，0.345，－56.79，0.0，12.0，0.0 等。

② 指数形式，如 12.34e3(代表 12.34×10^3)，－346.87e－25(代表 -346.87×10^{-25})，0.145E25(代表 0.145×10^{-25})等。由于在计算机输入或输出时，无法表示上角或下角，故规定以字母 e 或 E 代表以 10 为底的指数。但应注意：e 或 E 之前必须有数字，且 e 或 E 后面必须为整数。如不能写成 e4，12e2.5。

(3) **字符常量**。有两种形式的字符常量：

① **普通字符**，用单撇号括起来的一个字符，如：'a'，'Z'，'3'，'?'，'#'。不能写成'ab'或'12'。请注意：单撇号只是界限符，字符常量只能是一个字符，不包括单撇号。'a'和'A'是不同的字符常量。字符常量存储在计算机存储单元中时，并不是存储字符(如 a，z，#等)本身，而是以其代码(一般采用 ASCII 代码)存储的，例如字符'a'的 ASCII 化代码是 97，因此，在存储单元中存放的是 97(以二进制形式存放)。ASCII 字符与代码对照表见附录 B。[①]

---

① C 语言并没有指定使用哪一种字符集，由各编译系统自行决定采用哪一种字符集。C 语言只是规定：基本字符集中的每个字符必须用一个字节表示；空字符也占一个字节，它的所有二进位都是 0；对数字 0～9 字符的代码，后面一个数字的代码应比前一个数字的代码大 1(如在 ASCII 字符集中，数字'2'的代码是 50，数字'3'的代码是 51，后者比前者的代码大 1，符合要求)。中小型计算机系统大都采用 ASCII 字符集，ASCII 是 American Standard Code for Information Interchang(美国标准信息交换代码)的缩写。

② **转义字符**，除了以上形式的字符常量外，C 还允许用一种特殊形式的字符常量，就是以字符\开头的字符序列。例如，前面已经遇到过的，在 printf 函数中的'\n'它代表一个“换行”符。'\t'代表将输出的位置跳到下一个 tab 位置(制表位置)，一个 tab 位置为 8 列。这是一种在屏幕上无法显示的“控制字符”，在程序中也无法用一个一般形式的字符来表示，只能采用这样的特殊形式来表示。

常用的以“\”开头的特殊字符见表 3.1。

表 3.1　转义字符及其作用

转义字符	字符值	输出结果
\'	一个单撇号(')	具有此八进制码的字符
\"	一个双撇号(")	输出此字符
\?	一个问号(?)	输出此字符
\\	一个反斜线(\)	输出此字符
\a	警告(alert)	产生声音或视觉信号
\b	退格(backspace)	将当前位置后退一个字符
\f	换页(form feed)	将当前位置移到下一页的开头
\n	换行	将当前位置移到下一行的开头
\r	回车(carriage return)	将当前位置移到本行的开头
\t	水平制表符	将当前位置移到下一个 tab 位置
\v	垂直制表符	将当前位置移到下一个垂直制表对齐点
\o、\oo 或\ooo 其中 o 代表一个八进制数字	与该八进制码对应的 ASCII 字符	与该八进制码对应的字符
\xh[h…] 其中 h 代表一个十六进制数字	与该十六进制码对应的 ASCII 字符	与该十六进制码对应的字符

表 3.1 中列出的字符称为**“转义字符”**，意思是将“\”后面的字符转换成另外的意义。如“\n”中的“n”不代表字母 n 而作为“换行”符。

表 3.1 中倒数第 2 行是一个以八进制数表示的字符，例如'\101'代表八进制数 101 的 ASCII 字符，即'A'(八进制数 101 相当于十进制数 65，从附录 B 可以看到 ASCII 码(十进制数)为 65 的字符是大写字母'A')。'\012'代表八进制数 12(即十进制数的 10)的 ASCII 码所对应的字符“换行”符。表 3.1 中倒数第 1 行是一个以十六进制数表示的 ASCII 字符，如'\x41'代表十六进制数 41 的 ASCII 字符，也是'A'(十六进制数 41 相当于十进制数 65)。用表 3.1 中的方法可以表示任何可显示的字母字符、数字字符、专用字符、图形字符和控制字符。如'\033'或'\x1B'代表 ASCII 代码为 27 的字符，即 ESC 控制符。'\0'或'\000'是代表 ASCII 码为 0 的控制字符，即“空操作”字符，它常用在字符串中。

(4) **字符串常量**。如"boy","123"等，用双撇号把若干个字符括起来，字符串常量是双撇号中的全部字符(但不包括双撇号本身)。注意不能错写成'CHINA','boy','123'。单撇

号内只能包含一个字符，双撇号内可以包含一个字符串。

说明：从其字面形式上即可识别的常量称为“字面常量”或“直接常量”。字面常量是没有名字的不变量。

(5) **符号常量**。用#define指令，指定用一个符号名称代表一个常量。如：

```
#define  PI  3.1416                    //注意行末没有分号
```

经过以上的指定后，本文件中从此行开始所有的PI都代表3.1416。在对程序进行编译前，预处理器先对PI进行处理，把所有PI全部置换为3.1416。这种用一个符号名代表一个常量的，称为**符号常量**。在预编译后，符号常量已全部变成字面常量(3.14159)。使用符号常量有以下好处。

① 含义清楚。看程序时从PI就可大致知道它代表圆周率。在定义符号常量名时应考虑“见名知意”。在一个规范的程序中不提倡使用很多的常数，如：sum＝15＊30＊23.5＊43，在检查程序时搞不清各个常数究竟代表什么。应尽量使用“见名知意”的变量名和符号常量。

② 在需要改变程序中多处用到的同一个常量时，能做到“一改全改”。例如在程序中多处用到某物品的价格，如果价格用一常数30表示，则在价格调整为40时，就需要在程序中作多处修改，若用符号常量PRICE代表价格，只须改动一处即可：

```
#define PRICE 40
```

注意：要区分符号常量和变量，不要把符号常量误认为变量。符号常量不占内存，只是一个临时符号，在预编译后这个符号就不存在了，故不能对符号常量赋以新值。为与变量名相区别，习惯上符号常量用大写表示，如PI，PRICE等。

2. 变量

如例3.1程序中的c，f和例3.2程序中的p0，p1，p2，p3，r1，r2，r3等是变量。变量代表一个有名字的、具有特定属性的一个存储单元。它用来存放数据，也就是存放变量的值。在程序运行期间，变量的值是可以改变的。

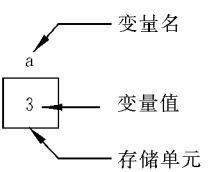

图 3.3

变量必须**先定义，后使用**[①]。在定义时指定该变量的名字和类型。一个变量应该有一个名字，以便被引用。请注意区分变量名和变量值这两个不同的概念，图3.3中a是变量名，3是变量a的值，即存放在变量a的内存单元中的数据。变量名实际上是以一个名字代表的一个存储地址。在对程序编译连接时由编译系统给每一个变量名分配对应的内存地址。从变量中取值，实际上是通过变量名找到相应的内存地址，从该存储单元中读取数据。

3. 常变量

C 99允许使用**常变量**，如：

---

① 定义变量的位置：一般在函数开头的声明部分定义变量，也可以在函数外定义变量(即外部变量、全局变量，见第7章)。C 99允许在函数中的复合语句(用一对花括号包起来)中定义变量。

```
const int a=3;
```

表示 a 被定义为一个整型变量，指定其值为 3，而且在变量存在期间其值不能改变。

常变量与常量的异同是：常变量具有变量的基本属性：有类型，占存储单元，只是不允许改变其值。可以说，常变量是有名字的不变量，而常量是没有名字的不变量。有名字就便于在程序中被引用。

请思考：常变量与符号常量有什么不同？如：

```
#define Pi 3.1415926                    //定义符号常量
const float pi=3.1415926;               //定义常变量
```

符号常量 Pi 和常变量 pi 都代表 3.1415926，在程序中都能使用。但二者性质不同：定义符号常量用 #define 指令，它是预编译指令，它只是用符号常量代表一个字符串，在预编译时仅是进行字符替换，在预编译后，符号常量就不存在了（全置换成 3.1415926 了），对符号常量的名字是不分配存储单元的。而常变量要占用存储单元，有变量值，只是该值不改变而已。从使用的角度看，常变量具有符号常量的优点，而且使用更方便。有了常变量以后，可以不必多用符号常量。

4. 标识符

在计算机高级语言中，用来对变量、符号常量名、函数、数组、类型等命名的有效字符序列统称为**标识符**(identifier)。简单地说，标识符就是一个对象的名字。前面用到的变量名 p1，p2，c，f，符号常量名 PI，PRICE，函数名 printf 等都是标识符。

C 语言规定标识符只能由字母、数字和下划线 3 种字符组成，且第 1 个字符必须为字母或下划线。下面列出的是合法的标识符，可以作为变量名：

sum，average，_total，Class，day，month，Student_name，lotus_1_2_3，BASIC，li_ling。

下面是不合法的标识符和变量名：

M. D. John，￥123，#33，3D64，a>b

注意：编译系统将大写字母和小写字母认为是两个不同的字符。因此，sum 和 SUM 是两个不同的变量名，同样，Class 和 class 也是两个不同的变量名。一般而言，变量名用小写字母表示，与人们日常习惯一致，以增加可读性。

3.2.2 数据类型

在例 3.1 和例 3.2 中可以看到：在定义变量时需要指定变量的类型。如例 3.1 中变量 f 和 c 被定义为单精度(float)型。C 语言要求在定义所有的变量时都要指定变量的类型。常量也是区分类型的。

为什么在用计算机运算时，要指定数据的类型呢？在数学中，数值是不分类型的，数值的运算是绝对准确的，例如：78 与 97 之和为 175，1/3 的值是 0.33333333…（循环小数）。数学是一门研究抽象的学科，数和数的运算都是抽象的。而在计算机中，数据是存放在存储单元中的，它是具体存在的。而且，存储单元是由有限的字节构成的，每一个存储单元中存放数据的范围是有限的，不可能存放“无穷大”的数，也不能存放循环小数。例如用 C 程序

计算和输出 1/3：

```
printf("%f",1.0/3.0);
```

得到的结果是 0.333333，只能得到 6 位小数，而不是无穷位的小数。

注意：用计算机进行的计算不是抽象的理论值的计算，而是用工程的方法实现的计算，在许多情况下只能得到近似的结果。

所谓类型，就是对数据分配存储单元的安排，包括存储单元的长度（占多少字节）以及数据的存储形式。不同的类型分配不同的长度和存储形式。

C 语言允许使用的类型见图 3.4，图中有 * 的是 C 99 所增加的。

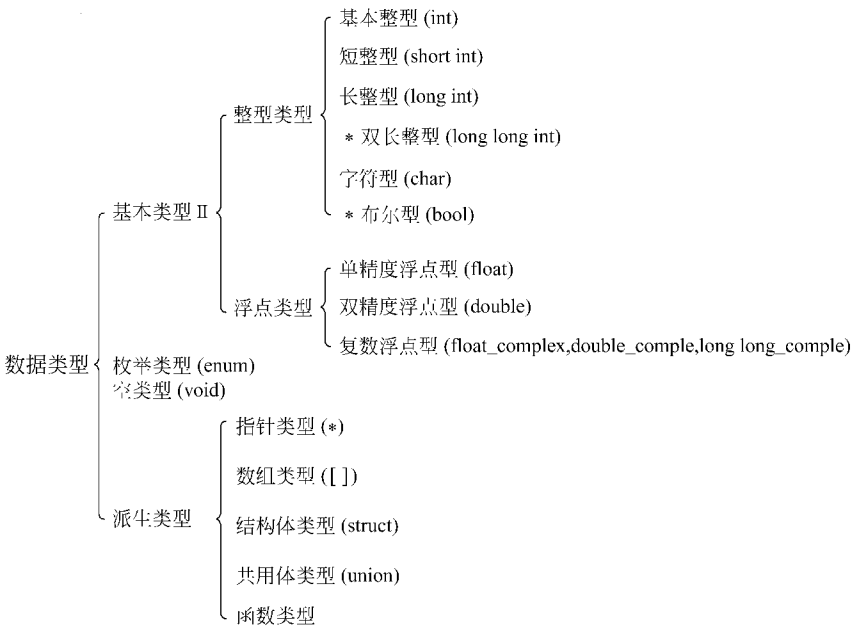

图 3.4

其中基本类型（包括整型和浮点型）和枚举类型变量的值都是数值，统称为算术类型（arithmetic type）。算术类型和指针类型统称为纯量类型（scalar type），因为其变量的值是以数字来表示的。枚举类型是程序中用户定义的整数类型。数组类型和结构体类型统称为组合类型（aggregate type），共用体类型不属于组合类型，因为在同一时间内只有一个成员具有值。函数类型用来定义函数，描述一个函数的接口，包括函数返回值的数据类型和参数的类型。

不同类型的数据在内存中占用的存储单元长度是不同的，例如，Visual C++ 6.0 为 char 型（字符型）数据分配 1 个字节，为 int 型（基本整型）数据分配 4 个字节，存储不同类型数据的方法也是不同的。

本书不孤立地、枯燥地叙述以上各种类型的规则，而是结合编程介绍怎样使用各种数据类型。本章及第 4～5 章介绍基本数据类型的应用，第 6 章介绍数组，第 7 章介绍函数，第 8 章介绍指针，第 9 章介绍结构体类型、共用体类型和枚举类型。

3.2.3 整型数据

1. 整型数据的分类

本节介绍最基本的整型类型。

(1) **基本整型**(int 型)

编译系统分配给 int 型数据 2 个字节或 4 个字节(由具体的 C 编译系统自行决定)。如 Turbo C 2.0 为每一个整型数据分配 2 个字节(16 个二进位),而 Visual C++ 为每一个整型数据分配 4 个字节(32 位)。在存储单元中的存储方式是:用整数的补码(complement)形式存放。一个正数的补码是此数的二进制形式,如 5 的二进制形式是 101,如果用两个字节存放一个整数,则在存储单元中数据形式如图 3.5 所示。如果是一个负数,则应先求出负数的补码。求负数的补码的方法是:先将此数的绝对值写成二进制形式,然后对其后面所有各二进位按位取反,再加 1。如−5 的补码见图 3.6。

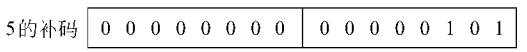

图 3.5

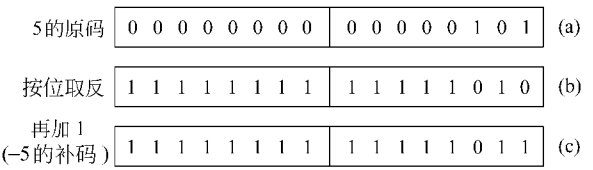

图 3.6

在存放整数的存储单元中,最左面一位是用来表示符号的,如果该位为 0,表示数值为正;如果该位为 1,表示数值为负。

有关补码的知识不属本书范围,在此不深入介绍,如需进一步了解,可参考有关计算机原理的书籍。

说明:如果给整型变量分配 2 个字节,则存储单元中能存放的最大值为 0111111111111111,第 1 位为 0 代表正数,后面 15 位为全 1,此数值是$(2^{15}-1)$,即十进制数 32767。最小值为 1000000000000000,此数是-2^{15},即−32768。因此一个整型变量的值的范围是−32768～32767。超过此范围,就出现数值的“溢出”,输出的结果显然不正确。如果给整型变量分配 4 个字节(Visual C++),其能容纳的数值范围为$-2^{31}\sim(2^{31}-1)$,即−2147483648～2147483647。

(2) **短整型**(short int)

类型名为 short int 或 short。如用 Visual C++ 6.0,编译系统分配给 int 数据 4 个字节,短整型 2 个字节。存储方式与 int 型相同。一个短整型变量的值的范围是−32768～32767。

(3) **长整型**(long int)

类型名为 long int 或 long。一个 long int 型变量的值的范围是$2^{31}\sim(2^{31}-1)$,即

−2147483648～2147483647(Visual C++ 6.0),编译系统分配给 long 数据 4 个字节。

(4) **双长整型**(long long int)

类型名为 long long int 或 long long,一般分配 8 个字节。这是 C 99 新增的类型,但许多 C 编译系统尚未实现。

说明:C 标准没有具体规定各种类型数据所占用存储单元的长度,这是由各编译系统自行决定的。C 标准只要求 long 型数据长度不短于 int 型,short 型不长于 int 型。即

sizeof(short)≤sizeof(int)≤sizeof(long)≤sizeof(long long)

sizeof 是测量类型或变量长度的运算符。在 Turbo C++ 中,int 型和 short 型数据都是 2 个字节(16 位),而 long 型数据是 4 个字节(32 位)。在 Visual C++ 6.0 中,short 数据的长度为 2 字节,int 数据的长度为 4 字节,long 数据的长度为 4 字节。通常的做法是:把 long 定为 32 位,把 short 定为 16 位,而 int 可以是 16 位,也可以是 32 位。读者应了解所用系统的规定。在将一个程序从 A 系统移到 B 系统时,需要注意这个区别。例如,在 A 系统,整型数据占 4 个字节,程序中将整数 50000 赋给整型变量 price 是合法的、可行的。但在 B 系统,整型数据占 2 个字节,将整数 50000 赋给整型变量 price 就超过整型数据的范围,出现"溢出"。这时应当把 int 型变量改为 long 型,才能得到正确的结果。

2. 整型变量的符号属性

以上介绍的几种类型,变量值在存储单元中都是以补码形式存储的,存储单元中的第 1 个二进位制代表符号。整型变量的值的范围包括负数到正数(见表 3.2)。

表 3.2 整型数据常见的存储空间和值的范围

类 型	字节数	取值范围
int(基本整型)	2	−32768～32767,即-2^{15}～$(2^{15}-1)$
	4	−2147483648～2147483647,即-2^{31}～$(2^{31}-1)$
unsigned int(无符号基本整型)	2	0～65535,即 0～$(2^{16}-1)$
	4	0～4294967295,即 0～$(2^{32}-1)$
short(短整型)	2	−32768～32767,即-2^{15}～$(2^{15}-1)$
unsigned short(无符号短整型)	2	0～65535,即 0～$(2^{16}-1)$
long(长整型)	4	−2147483648～2147483647,即-2^{31}～$(2^{31}-1)$
unsigned long(无符号长整型)	4	0～4294967295,即 0～$(2^{32}-1)$
long long(双长型)	8	−9223372036854775808～9223372036854775807 即-2^{63}～$(2^{63}-1)$
unsigned long long (无符号双长整型)	8	0～18446744073709551615,即 0～$(2^{64}-1)$

在实际应用中，有的数据的范围常常只有正值(如学号、年龄、库存量、存款额等)。为了充分利用变量的值的范围，可以将变量定义为“无符号”类型。可以在类型符号前面加上修饰符 unsigned，表示指定该变量是“无符号整数”类型。如果加上修饰符 signed，则是“有符号类型”。因此，在以上 4 种整型数据的基础上可以扩展为以下 8 种整型数据。即

有符号基本整型　　[signed] int;
无符号基本整型　　unsigned int;
有符号短整型　　[signed] short [int];
无符号短整型　　unsigned short [int];
有符号长整型　　[signed] long [int];
无符号长整型　　unsigned long [int]
有符号双长整型*　　[signed] long long [int];
无符号双长整型*　　unsigned long long [int]

以上有“*”的是 C 99 增加的，方括号表示其中的内容是可选的，既可以有，也可以没有。如果既未指定为 signed 也未指定为 unsigned 的，默认为“有符号类型”。如 signed int a 和 int a 等价。

有符号整型数据存储单元中最高位代表符号(0 为正，1 为负)。如果指定 unsigned(为无符号)型，存储单元中全部二进位(b)都用作存放数值本身，而没有符号。无符号型变量只能存放不带符号的整数，如 123，4687 等，而不能存放负数，如－123，－3。由于左面最高位不再用来表示符号，而用来表示数值，因此无符号整型变量中可以存放的正数的范围比一般整型变量中正数的范围扩大一倍。如果在程序中定义 a 和 b 两个短整型变量(占 2 个字节)，其中 b 为无符号短整型：

```
short a;                          //a 为有符号短整型变量
unsigned short b;                 //b 为无符号短整型变量
```

则变量 a 的数值范围为－32768～32767，而变量 b 的数值范围为 0～65535。图 3.7(a)表示有符号整型变量 a 的最大值(32767)，图 3.7(b)表示无符号整型变量 b 的最大值(65535)。

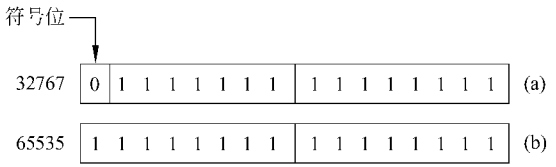

图　3.7

表 3.2 列出各种整型数据的存储空间和值的范围。

说明：

(1) 只有整型(包括字符型)数据可以加 signed 或 unsigned 修饰符，实型数据不能加。

(2) 对无符号整型数据用“%u”格式输出。%u 表示用无符号十进制数的格式输出。如：

```
unsigned short price = 50;                //定义 price 为无符号短整型变量
printf("%u\n",price);                     //指定用无符号十进制数的格式输出
```