第一篇

C 语言程序设计基础

第一章 C语言概述

1.1 C语言的由来和发展

60年代，随着计算机科学的迅速发展，高级程序设计语言得到了广泛的应用。然而还没有一种可以用于书写操作系统和编译器等系统程序的高级语言，人们不得不用汇编语言来书写。但汇编语言存在着不可移植、可读性差、描述问题效率不高级语言等缺点，给编程带来很多不便，这就大大限制了人们在这门新兴科学中的活动。为此，人们对能用于系统设计的高级语言的开发就变得势在必行。

1969年，英国剑桥大学的 Maitin Richards 和美国贝尔实验室的 K Thompson 和 D. M. Ritche 先后设计并实现了 BCPL(Basic Computing Programming Language)语言和 B 语言(类似于 BCPL 语言)。1971 年，K Thompson 在 PDP—11/20 上实现了 B 语言，并用 B 语言书写了 UNIX 操作系统和实用程序。由于 BCPL 和 B 语言都是无类型语言，无法支持多种数据类型，导致描述许多事物时遇到困难，为此在 1972 到 1973 年间，贝尔实验室的 D. M. Ritchie 在 B 语言的基础上重新设计了一种取名为 C 的语言，用它重写了 UNIX 操作系统(定为第五版，这是当今世界上最著名的分时操作系统之一，在美国有 90%以上的大学都在使用该系统)，并在 PDP—11 机上实现。该版本加进多道程序设计功能，为 UNIX 系统的发展奠定了基础。

二十多年来，C 语言得到不断改善和扩充。目前从微型机到大型计算机都配有 C 编译程序，不仅能在装配 UNIX 操作系统的机器上，而且在非 UNIX 操作系统支持的机器上也能实现 C 的编译程序。由于 C 语言本身具有许多特点，现在它已成了在微、小、大、巨型计算机及计算机网络上从系统设计到工程应用都能使用的一种高级语言。

1.2 C语言的特点

1.2.1 C语言的优点

C 语言的特点可以从多方面阐述，这里仅从使用者的角度加以讨论，并将其大致归纳为以下几个主要方面：

1. 表达能力强且灵活。

C 语言可以直接处理字符、数字、地址，可以完成通常要用硬件实现的普通的算术及逻辑运算，且可以表达数值处理、字处理功能。

2. 程序结构清晰且紧凑。

因为 C 语言程序通常由若干个函数组成，所以它是一种模块化程序设计语言。因此，C 语言的大程序可以用若干个较小的程序模块来组织，这种模化结构的程序不但清晰，而且紧凑。

3. 书写简单、易学。

C 语言书写起来于分简洁。例如，C 语言用复合语句标号{}来代替一般语言中的 begin 和

end，花括号中可以包含 C 语言的任何一种语句，对语句列的长度也不加限制。另外，C 语言中还将各种运算符尽量地缩写以便于编程。

4. 生成的目标代码质量高。

用 C 语言编写的程序生成的代码质量高。实践证明，在实现相同功能的前提下，其他高级语言相对汇编语言的代码效率要低得多；而用 C 语言描述，其代码效率只比汇编语言低 10%～20%，但 C 语言在描述问题时的编程迅速、可读性好、表达能力强等优点是汇编语言无法比拟的。

5. 可移植性好。

对 C 语言编写的程序可以不加改动或稍加改动就可从一个环境搬到另一个环境中使用。统计资料表明，不同机器上的 C 语言编译程序 80%的代码可以公用，这样就带来了很大的方便。

1.2.2　C 语言的不足

世上没有十全十美的事物，C 语言也不例外。C 语言虽然具有表达能力强、生成代码质量高、可移植性好等优点，但是，传统 C 语言也存在着一些不足之处。比如说：求值顺序存在二义性，外部连接规则不明确，不能实现递归算法等等。

近些年来，以 C 语言产品为主的各厂家先后推出了一些新的 C 语言编译器，它们都以 ANSI C[1]或某种扩充 C 为标准对语言进行了不同程度的扩充，目的在于补传统 C 语言的缺陷。例如：Borland 公司的 C 编译器产品就能对 C 语言求值顺序的二义性提示警告信息；MetaWare 公司在其 Hight C 中引入了迭代子的概念来实现递归算法。

在实际编程过程中，由于程序员所使用的 C 语言编译器大多数或多或少地在传统 C 上作了改进，因此可能会自动避开 C 的一些下足之处，不会因此而导致编程上的麻烦。

本节中之所以列出传统 C 中的这些不足之处，目在在于帮助读者更深入地对 C 语言进行了解，加深对 C 语言的认识，以便在实际应用中避免出错。

● 不精确的分类

C 语言中，字符型是个非常普遍的概念，而且常和整数等同，字符和整数还可以相互转换，C 语言还允许这两种类型间相互赋值，这种情况导致了 C 语言中不得不进行各种不太严格的硬行规定。请看下面这个例子：

例：变量在字符型和整型之间隐式转换

```
#include (stdio.h)
main()
{
  int c;
  while ((c=getchar()) ! =EOF)
        if ('a'<=c &&c<='y')
            putchar(c+'A'-'a');
        else
            putchar(c);
}
```

上例目的是把小写字母转换成大写字母，字母输入是ASCII的形式，变量c在概念上讲是字符型，但它却必须被定义成整型，因为它的域中必须包含判断文件是否结束的一1值(EOF)，这样就混淆了字符型与整型之间的类型差异。

● 副作用运算符

在C中，副作用运算符普遍存在，如加1、减1运算符，逗号运算符，各种赋值操作符等。这种运算符可以紧凑程序，提高代码效力，但它牺牲了程序的可读性，且可能诱发多种可能性。

● 求值顺序的二义性

C程序中许多动作与编译器相关，这样做，一是对于特定的计算机，被选择的语义特别有效；二是能够为一种特定的机器选择一种最简单的实现方法。但是当这些部分使用时，会影响到代码的移植性。如：

```
i=1;
j=i++ + ++i--;
```

对于不同的编译器，j值计算结果可能为1，2或3。更重要的是，同一个编译器下也可能算得不同的值，这取决于优化的状态。C不能保证求值顺序，即使是有括号的情况，这是因为编译器对某个表达式求值时，会按照它自身规定的某种方便和有效的方式进行计算。

● 参数传递不严谨

C不对参数作类型检查，函数实参的值按某类型压入堆栈上来传递给函数，被调过程根据自己的变量说明或者是整数缺省值来取用这些参数，例：

```
#include (math.h)
main()
{
  printf("sin(%d) is %e",1,sin(1));
  printf("sin(%e) is %e",1,sin(1.0));
}
```

在XENIX操作系统下运行结果为：

```
sin(1) is 9.996802e-01
sin(1.000000e+00) is 8.414710e-01
```

● 缺乏真正的多维数组

C语言的内存模式是一维的，多维数组也必须转化成一维数组按线性的顺序依次存入内存。这样，在进行程序设计时程序员必须知道数组对界关系。比如说：一个二维数组实质上是一个指针到指针的类型，若不在说明中规定第二个界限，那么就无法计算出元素在内存中的映象关系，从而不能进行正确调用。

● 缺乏基本类型

C中没有布尔类型的定义，这导致程序对整数类型的过份依赖，C语言用零代表false，非零代表true，这样使得布尔公理失效。在C中，也没有集合类型和常数说明。

● 语句的缺陷

C语言中，提供的goto语句就破坏了程序的结构化特性；多路开关语句中需要break语句来构造控制流，break，continue，goto，return这些语句都可能终止一个普通循环，其潜在的结合使一般的运算分析结果难以预料。

● 外部连接规则不明确

C 程序中，在一个文件中说明的外部变量一定要能够在另一个文件中找到。为了确定在哪一个说明是一个变量定义的情况，有四个法则：

① 除一个外部变量说明外，所有同名变量都要有 extern 关键字。

② 第一次初始化作为当前的定义，预置的值为变量初始化的值。

③ 初始化程序仅执行一次。

④ 任一个初始化都是允许的，选择哪一个，依赖于实现。

```
/* file1 */
int i=4;
/* file2 */
extern i=3;
/* file3 */
main()
{
  extern i;
  printf("%d-i",i);
}
```

这段代码没有语法错误。但是，这是不是一段可接受的代码呢？实际上，它取决于特定系统的编译器和连接器，这显然影响了可移植性。

● 无函数定义嵌套

所有 C 函数都处于同一等级，函数之间能够相互调用，也可递归调用，但是，函数定义不能嵌套，因此 C 中有模块结构和局部堆栈环境，但没有局部函数。

● 不能直接返回数组

数组可作为结构的成员返回，却不能直接作函数值返回。

● 缺乏基本 I/O 定义

以上列出了传统 C 中存在的不足，实际上，这些看法并不是绝对的。如果你以不同的角度出发来看这些问题，可能会得出在大相径庭的结论。

1.3　C 语言程序的基本结构

上一节中已经介绍过，C 语言具有良好的模块化结构。在 C 语言中，模块化结构是通过函数来实现的，函数是按一定的格式排列并能完成某些功能的独立的语句集合。通常，在编程时，若出现以下三种情况就可以考虑去编制单独的函数。

1. 程序中有许多完全相同，或者算法和语句排列都完全相同，仅仅是可涉及的某些变量有所不同的程序段，且因为这些程序段之间或者距离较远，或者涉及的变量太多，或者变化的规律性不强等等原因而不能用循环语句来实现的，就可以考虑用函数来实现。

2. 当程序的某一点需要有选择地跳转去执行其他几个程序段，而这几个程序段都比较长，且相互的距离也比较远，以至仅靠 goto 语句无法控制时，就应当考虑把这些程序段分别编成函数，而去有选择地调用。这样做虽未能缩短程序，但却可以使程序变得通顺、流畅。

3. 当程序较长，不易分析和调试时，也应当考虑编制一些函数，这样会使程序简洁、清晰、便于调试，可读性好。

在程序设计中，使用函数有以下优点：

1. 引入函数后，使程序化为若干小的功能块，块与块之间相对独立，这就使程序的各部分之间结构清晰，逻辑关系明确，可读性大大提高，而且便于查错和修改。

2. 引入函数后，就可以在需要的地方直接引用函数名来代替整个函数，尤其是将那些需要多次运行的程序部分编写成函数后，将会大大减小整个程序的长度，节省内存和磁盘空间。

3. 函数可以把那些不需要知道的运算和操作细节隐藏起来，只留下非常简单的接口(参数)供人们使用，这就使编程者可以在他人以前所做工作的基础上着手工作，而不感到太吃力，从而提高了编程效率。

C语言本身的特点就是使编程者将大的问题化为许多小的问题，逐一编成函数，使整个程序模块化。

图1.1给出了一个常见的C语言程序结构图：

C程序的一般结构	例1.1 求两数的平均数。
预处理语句	#include <stdio.h>
外部变量说明	int ave;

main()
{
说明语句
可执行语句
}
	mian()
	{
	int n=10,m=8;
	ave=average(n,m);
	printf("The average of %d and
	%d is %d\n",n,m,ave);
	}

函数1()
{
说明语句
可执行语句
}
	average(int x,int y)
	{
	int z;
	z=x+y;
	z=y/2;
	return(z);
	}

函数2()
{
说明语句
可执行语句
}	

⋮	

C 程序的一般结构	例 1.1　求两数的平均数。

函数 n()
{
说明语句
可执行语句
}	

图 1.1　C 语言程序结构

说明：

1. 预处理语句是 C 语言在开始编译前就先处理的语句。它们通常是一些编译命令，如嵌入文件、定义符号常量或宏以及条件编译等。

2. 外部变量说明定义了一个公用的数据区，用来完成各函数之间的通信，外部变量只要定义在任何函数之外就可以，并不一定非要在程序的一开始就定义。但请注意，只有位于外部变量定义之后的函数才能直接使用，若在其前面或者其他文件中，则一定要说明后才能使用。

3. main()函数是 C 语言程序的一个特殊函数，称之为主函数。任何一个可执行的 C 语言程序都必须有一个且只能有一个 main 函数。程序总是从 main()函数的开始处执行，而不在乎它在整个程序中的位置。

下面给出一个典型的 C 语言程序。

例 1.2　求两数的平均值。

```
/* 文件名:pro11.c */
#include (stdio.h)
main()
  {
    int n=10,m=78,ave;
    ave=average(n,m);
    printf("The average of %d and %d is %d\n",n,m,ave);
  }
average(int x,int y)
  {
    int z;
    z=x+y;
    z=y/z;
    return(z);
  }
```

运行结果：

```
C)pro11↙
The average of n and m is 44.
```

上例中，“#include”语句是预处理语句，main()是主函数，而 average()则是普通函数，它被主函数调用。

1.4 C语言程序的编译和连接

C语言是一种编译型语言，它与解释型的Basic和LISP语言不同，用户通过编辑器编写完成的C语言源程序并不能马上在目标机上运行。用户必须借助于编译器对自己的源程序进行编译和连接，才能够生成在目标机中运行的代码模块。

1.4.1 C语言程序的编译和连接过程

C语言的程序的编译和连接过程如下：首先它要通过编译器的编译，检查其语法、语义上是否合乎语法规则，并对源程序中的预处理语句进行预编译，再对整个模形编译之后，便形成目标文件，然后由连接器把一个程序的所有源文件编译后形成的目标文件连接起来，形成可执行文件，这时，运行了可执行文件，编程的预期效果才会显示出来。

1.4.2 C语言的编译和连接命令

1.4.2.1 Microsoft C环境

Microsoft C编译器适用于MS—DOS操作系统，在MS—C中，cl命令是用来编译和连接C源文件的唯一的命令，cl按三遍(pass)执行编译程序，然后自动调用LINK(Microsoft覆盖式连接器)来连接文件。

用户可以通过使用cl提供的选择项，来控制和修改由cl命令进行的任务。

cl命令具有如下形式：

CL [option] …file…[option…file…][/link [link—libinfo]]

命令中option表示选择项，Microsoft C编译器为编译源文件提供了大量的命令行选择项，详见具体的系统参考手册。

每个file指定了一个要被处理的源文件或目标名，或指定了一个在连接时要查找的库文件名。

CL命令自动指定在连接时使用的相应的库，但是，也可以使用带有可选择的连接库信息的/link选择项来指定另外的或不同的库，库查找路径及连接时使用的选择项，也可在link option域中指定连接程序选择项。

CL命令能处理源文件、目标文件、库文件或它们的任意组合，CL使用扩展文件名来决定该文件需要作何种处理，如下表所示：

- 如果文件扩展名是.C，则CL编译该文件；
- 如果文件扩展名是.OBJ，则CL调用连接器处理该文件；
- 如果文件扩展名是.LIB，则CL将把文件传给连接器去查找，除非命令行给出选择项抑制连接；
- 如果扩展名省略，则CL假定扩展名为.OBJ；如果扩展名是除.C、.OBJ、.LIB之外的任意形式，则CL假定该文件名为一个目标文件，除非该文件名被选择项指明为C的源文件；
- 如果文件扩展名是.ASM，则CL显示出错信息，指出它不能调用Microsoft宏汇编器。

在调用CL进行编译连接时，用户可以用MS—DOS的通配符(*或?)来对一批文件进行

处理;但是,不要在带有文件名参数的 CL 选择项中使用通配符,也不要在由 CL 环境变量给出的 CL 选择项中使用通配符。

在 CL 命令行中任何 filespec 可以包括完全的或部分的路径说明,完全的路径说明包括驱动器号,而部分路径说明则不包括。

如果由于某种原因想中断编译和连接过程,可按〈CTRL〉+C 或〈CTRL〉+〈BREAK〉,则 CL 将终止运行并返回 MS-DOS 命令行状态。

例:例 1.2 已经给出 C 的源文件,要想一次性完性编译和连接产生可执行文件,命令如下:

```
C)CL pro11.c <CR>
```

用户也可以将编译和连接过程单步进行,命令如下:

```
C)CL /c pro11.c <CR>
C)LINK pro11 <CR>
```

在 Microsoft C 推出之后,Microsoft 公司又推出了 Microsoft CodeView 调试器,它是强有力的基于窗口的工具,可以跟踪程序中的逻辑错误。CodeView 调试器可显示源代码或汇编代码,指示将要运行的行,还可以动态检查变量值,切换屏幕以查看程序输出以及执行其他相关的功能。调试器中的实用程序对软件开发的不同阶段都很重要。在用编译器或汇编器生成一个或多个目标文件时,要用 LINK 来产生一个可执行文件,在 LINK 过程中,LIB 文件可使你生成检查和维护软件库,用实用程序 MAKE 可在相当大的程度上自动实现编译和连接,MAKE 保持对改变过的源文件的追踪并只执行必要的命令来修改程序。具体使用方法请参见有关的系统参考手册。

注意,CodeView 调试器并非对所有的文件都能支持,表 1-1 中列出了这些文件的类型。

表 1-1 调试器不能直接支持的文件类型

文件名	解 释
包含文件	不能用 CodeView 调试器去调试 include file 中的源代码
压缩文件	CodeView 的符号信息不能送入压缩文件中
.COM 文件	扩展名为.COM 的文件可在汇编状态下调试,它们不能包含符号调信息
常驻内存程序	调试器只能和驻在磁盘的.EXE 和.COM 文件一起工作,不支持常驻内存文件
改变环境的程序	在调试器下运行的程序可以读 DOS 环境,但不能永久地修改它。在退出 CodeView 时,所有对环境的修改遗失
程序段前缀 PSP	CodeView 调试器自动预处理程序的 PSP

在 MSC 6.0 以上的版本中,新增了程序员工作台(PWB)的功能,它是对程序设计和调试器的集成化。PWB 是一个面向窗口的程序设计环境,它把正文编辑、编译、连接、调试、MAKE 功能、源程序浏览和联机帮助数据库结合起来。在启动 PWB 之后,可以用菜单系统迅速找到所需要的命令。在集成环境下编辑程序,可利用系统提供的宏定义、书签功能、键盘重定义功能及强大的搜索功能来减少代码的书写量。同时,也可以在不退出编辑器的情况下,编译连接并运行应用程序,在编译连接程序时,可以在选择菜单中指示编译参数,这些参数包括存储模式、处理器类型及影响编译过程的其它全程设置,在 PWB 集成环境中编译程序之前,要指定构造

程序的类型，如.EXE或.COM等，这可通过选择菜单中的构造选择项来完成。

当然，您也不必局限于PWB环境中构造程序，源程序编辑完毕后，可在操作系统的环境下使用命令行编译器，即用CL命令编译和连接程序。CL命令的使用方式在前面已叙述过了。

1.4.2.2　Microsoft Quick C环境

Microsoft Quick C编译器集效率高、可移植性强、灵活性好于一体，并为C语言程序设计提供了一套完善的开发环境，其中主要包括一个集成化程序设计环境。

在快速C编译器中编译一个C程序用如下语句：

QCL [option]…file…[option|file]…[/link[lib…lintopt]]

命令中的信息说明如下：

option是选择项，用于控制编译进程中的各种操作，详细内容请见具体的系统参考手册。为快速查看常用选择项，可打入QCL/HELP，并按ENTER键。

file是要处理的源文件格式目标文件名，或者要传递给连接器的库文件名，用户在进行编译连接时，命令行上至少必须给出一个文件名，QCL根据文件的扩展名决定如何处理相应文件：

表1－2　QCL命令能处理的文件扩展名

扩展名	动　作
.C	假定为C源文件，编译之
.OBJ	假定为目标文件，并把它传送给连接器处理
.LIB	假定为独立库文件，并传送给连接程序，连接器将该文件和生成的目标文件以及命令行中给出的目标文件相连
其他扩展名或无扩展名	假定该文件为目标文件，将它传送给连接器处理

任何文件名可以包含一个完整或部分的路径名。

lib是要传递给连接器的库文件名。

link－opt是一个或更多的选择项，QCL命令会自动将这些选择项传递给连接器，通常不必给出该选择项，除非希望连接器完成特殊操作。

命令行要求长度不得超过128个字符(这是由MS－DOS环境规定的)，在这个限制内可给出任意个选择项，文件名和独立的库文件名。

如果要单独进行连接，则首先可用LINK命令直接调用连接器，其次在QCL命令中的文件名用目标文件名标识。

在进入目标文件的当前目录后，通过以下方式使用LINK命令：

LINK objfile…[,[exefile][,[mapfile][,[lib[…]]] [link－opt]…[;]

命令行长度不超过128个字符。

如果可执行文件名被省略，则最终所产生的可执行文件的文件名以第一个目标文件名为基本名，加上.EXE扩展名。

如果映象文件缺省，则以第一个目标文件名为基本名，加上.MAP扩展名。

如果库文件缺省，编译联接命令根据行上的存储模型选择项和浮点选择项，决定一个合适

的库文件名。

1.4.2.3　Turbo C 环境

Turbo C 是美国 Borland 国际公司在 IBM PC 机上实现的一个高效、优化的 C 编译器，其编译速度快，能支持极小、小、中、紧凑、大、巨型等六种存储模式，还可以使用远、近指针的混合模式。另外，Turbo C 为开发者提供了一个完整的交互式集成开发环境。

Turbo C 的命令行版本为 TCC，可编译 C 语言源文件，并把其连接成可执行文件，TCC 也可调用 MASM 汇编程序，以汇编. ASM 源文件，如果仅仅是编译源程序，只须在命令行中使用－C 选择项。

TCC 命令行格式为：

tcc［选择项　选择项　选择项…］文件名　文件名…

键入一个短横线（－）紧跟一个选择项字符就可设置命令行开关，每个选择项字符后再加一个短横线就关闭该选择项，例：－A 把 ANSI 关键字选择项置成“开”，－A－则将此选择项置成“关”，用此特性可在命令行中设置一些选择项，取代配置文件中相应的设置。

编译器按下列规则编译命令行中的文件名：

表 1－3　TCC 命令能处理的文件扩展名一览

文件名	动　作
filename	编译 filename. c
filename. c	编译 filename. c
filename. xyz	编译 filename. xyz
filename. obj	连接时的目标文件
filename. lib	连接时的库文件
filename. asm	调用 TASM 将之汇编成 OBJ 文件

在 Turbo C 的集成开发环境下，建立一个新的可执行文件的一般步骤为：

① 设置工作环境以指示编译器和连接器在何处寻找和保存有关文件。

② 把相应的源程序装入编辑窗口（注意，若可执行程序由多个模块组成，则需建立一个由所有模块名组成的工程文件）。

③ 建立可执行程序文件。

建立可执行文件的步骤取决于源文件是单个文件还是多个文件。

Turbo C 具体的命令行编译和集成编译参见相应的系统参考手册。

1.4.2.4　Borland C＋＋环境

Borland C＋＋是 Borland 国际公司为面向对象的程序设计提供的编译器。

Borland C＋＋编译器提供了两种开发环境：一种是交互集成开发环境（主文件名为 bc. exe），它集编辑、编译、连接、调试于一体，编译连接器的各种选择项和开关项可以从环境中直接进行设置与修改，大大的方便了开发人员；另一种是命令行开发环境，其编译器名为 bcc. exe，连接器为 tlink. exe，开发者也可以通过从命令行上输入命令对程序进行编译和连接操作。

Borland C++对文件操作等的规定与其 Turbo C 一致。

1.4.2.5 High C 环境

High C 是 MetaWare 公司推出的 32 位 C 语言编译器。除了对标准 C 语言进行了大量的扩充之外，该编译器的最大特点就是能编译出运行于 80386、80486 等微处理器保护模式下的可执行程序——.exp。

经过与 AutoCAD R11 版以上的 ADS 所提供的库函数 ads.lib 进行适当的连接之后，由 High C 编译出的保护模式可执行程序可直接加载到 AutoCAD 环境中运行。与 AutoLISP 语言编写出来的程序相比，由 High C 编译出的保护模式可执行程序具有保密性好、运行速度快等特点，因此，大量的 AutoCAD 程序开发者正在转向用 C 语言开发 AutoCAD 应用程序。

High C 是 AutoDesk 公司为其 AutoCAD 产品指定的缺省的 C 语言开发编译器，本书将在"第二篇 High C 程序设计基础"中对 High C 语言的基本语法结构，它对 C 语言的基本扩充功能以及程序设计过程进行详细讲解。关于 High C 的有关内容请参见第二篇。

第二章　基本元素及数据类型

2.1　C语言的基本元素

C语言的基本元素是指构成C程序的各种名字、数字和符号。主要包括下述内容：

- 符号集
- 常数
- 标识符
- 关键字
- 注释
- 标志

2.1.1　符号集

C语言定义了两个可供使用的字符集，即C符号集和可表示符号集。可表示符号集包括所有的字母、数字以及用户用单字符可描绘出的任何符号，可表示符号集的范围依赖于终端、主控台或符号发生器的类型。C符号集是可表示符号集的一个子集，包括字母、数字和对C编译器有指定含义的标点符号。下面将介绍C符号集中的各种符号。

2.1.1.1　字母、数字、下划线

C符号集中包括英文符号表的所有大、小写字母，十进制阿拉伯数字系统和下划线：

- 大写字母：ABCDEFGHIJKLMNOPQRSTUVWXYZ；
- 小写字母：abcdefgbijklmnopqrstuvwxyz；
- 十进制数字：0123456789；
- 下划线：_。

其中大、小字字母被视为不同的符号，这在编程时要加以注意。

2.1.1.2　空白符号

空格符、制表符(tab)、垂直制表符(veitical tab)、回车符(carriege return)、换行符(line feed)和换页符(form feed)都称为空白符，它们在正文显示时，表示字与字之间、行与行之间的空白间隔，这些符号的内部值虽然不同，但都是对用户定义的项起分隔作用。

〈CTRL〉+Z是个特殊功能键，表示文件终结。这一功能键之后的字符将被忽略。

C编译器在编译时忽略这些空白符号，注释的内容也被视为空白符号。

2.1.1.3　标点符号和特殊记号

表2—1中列出了C语言中使用的标点符号和特殊记号。

表 2－1　标点符号与特殊符号表

符号	名称
,	逗号
.	圆点
;	分号
:	冒号
?	问号
'	单引号
"	双引号
(	左圆括号
)	右圆括号
[	左方括号
]	右方括号
{	左花括号
}	右花括号
<	左尖括号(小于号)
>	右尖括号(大于号)
!	感叹号
|	竖线
/	斜线
\	反斜线
~	波折号
#	#号
%	百分号
&	and 号
^	异或符号
*	乘号
-	减号
+	加法
=	等于号

注意：仅能出现于文字串、符号常数注释和#include 命令的文件名中的可表示符号集并没有列在上表中。

2.1.1.4　转义序列

转义序列又称换码序列，常用于字符串和常数中，以表示某种特定的动作。表 2－2 中给出了 ANSI 标准的转义序列。

表 2—2　转义序列表

符号	名称
\n	换行
\t	水平制表
\v	垂直制表
\b	退格
\r	回车
\f	换页
\a	响铃报警
\'	单引号
\"	双引号
\\	反斜线
\ddd	八进制 ASCII 代码值
\xddd	十六进制 ASCII 代码值

若反斜线后面的字符不包括在上表中，那么反斜线不起作用。\ddd 和\xddd 序列可表示 ASCII 码的任意字符，\后面的数字(d)或是三位八进制数或是三位十六进制数，该序列中虽不必一定要三位，但至少要出现一位数字。

2.1.1.5　操作符表

表 2—3 中给出了 C 语言的操作符及其含义。

表 2—3　操作符

操作符	含义
<	小于
>	大于
<=	小于等于
>=	大于等于
==	相等
!=	不等

2.1.2　常数

在程序中能作为常数值的有数字和字符串，常数值在执行时始终不能改变。C 语言中有如下几种常数：整数、浮点数、字符常数和字符串。

2.1.2.1　整数

整数可以是十进制，八进制，十六进制数字表示的整数值。

十进制常数形式为:digits(digits 可为从 0 到 9 的一个或多个十进制数位,首位不为 0);八进制常数形式为:0digits(这里 digits 为从 0 到 9 的一位或多位八进制数位,首位 0 是引导符);十六进制常数形式为:0xhdigits 或 0Xdigits(在此,hdigits 是从 0~9 的数字并从'a'到'f'的字母表示的一位或多位十六进制数,0x 是引导符,x 可用大写或小写)。

空白字符不可出现在整常数数字之间。

如需要表示负值,必须把'—'放在常数表达式之前。

八进制、十六进制常数可对应整型、无符号整型、长整型或无符号长整型,当表示成长型时,要在常数的尾部添加字母"l"或"L"。

2.1.2.2 浮点常数

浮点常数是一个十进制表示的符号实数,其语义表示如下:

[digits][. digits] [E|e[-|+]digits]

其中,digits 是一位或多位十进制数字(0—9),E(e)是指数符号,小数点之前是整数部分,小数点之后是尾数部分(可省)。

在浮点常数中不得出现任何空白符号。

所有的浮点常数视为双精类型。

2.1.2.3 字符常数

字符常数的语义表示为:

'char'

单字符常数是由单引号(')括住的可表示字符集中的单个字符,转义序列视为单个字符,因而也是有效的字符常数。注意,转义字符必须用转义序列表示,否则会出错。一个字符常数的值就是该字符在字符集中的数值。

在使用单引号和反斜线时,在前面须再用反斜线。如:'\''表示单引号。

字符常数总是整型,在类型转换中不带符号。

2.1.2.4 字符串

字符串的语义表示为:

"characters"["characters"]

"字符串"是括在双引号(")之中的可表示字符集中字符的序列。在字符串中双引号、反斜线、换行符要用它们各自的转义序列(\",\\,\n)表示,非打印字符也要用相应的转义序列表示。每个转义序列视为单个字符。

用户可在字符串中制造换行,其做法是在希望换行的地方放上转义序列"\n",例如:

"long strings can be broken into \n two or more pieces."

此行实行显示为:

```
Long strings can be broken into
two or more pieces.
```

如果程序中的某一行太长,用户可以在源程序中用续行符号(\)将一行形式上分为两行。

传统方法是在前一行的最后加一个反斜杠(\)并立即回车，例如：

```
"Long strings can be bro\
   ken into two or more pieces."
```

实际表示的字符串是："long strings can be broken into two or more pieces."

仅用空格隔开的两个或多个字符串将被联接成一个字符串。如：

```
printf ("This is the first half of the string,"
                    "this is the second half");
```

则显示为："This is the first half of the string,this is the second half."

字符串的字符按照从左到右的顺序依次存储在连续的存储单元中，空字符(用转义序列\0表示)自动添加到串尾表示每个字符串的结束。程序中的每个字符串通常视为彼此不同的，尽管如此，两个相同的字符串还是不能存储在不同的单元中。因此，程序在执行当中不允许修改字符串。

字符串的类型是char[ ]，即一个字符串是其每个元素都是字符型的数组。数组元素的个数是串中的字符数加1(最后的空字符也作为数组的元素被存储)。

2.1.3　标识符

标识符是在数据或一般信息处理中表示数据项目并予以命名的字符串，在给定程序中的名字。在程序中必须用说明的方式建立与标识符相关的变量或函数。说明后可在程序的语句中引用它们。标识符是以字母或下划线领头的字母、数字或下划线组成的序列。标识符的长度由编译器的识别能力决定。在使用下划线(_)时应小心，标识符开头使用下划线时可能会与隐蔽的系统标识符相冲突而出错。

2.1.4　关键字

关键字是C编译器在内部指定的、具有固定含义的标识符。用户定义的任何标识符不得与关键字冲突，且关键字不得重定义。不过，程序员可以在编译前用预处理命令指定标识符来代替关键字。

关键字的具体序列如下(带下划线的关键字是MSC 6.0版的扩充部分)：

_asm	_far	_segment
auto	_fastcall	_segname
_based	float	self
break	for	short
case	_fortran	signed
_cdecl	goto	sizeof
char	_huge	static
const	if	struct
continue	int	switch
default	_interrupt	typedef
do	_loadds	union
double	long	unsigned
else	_near	void