第 3 章　语法规则

在人类交流的过程中，如果没有统一的语法规则，彼此之间将不能理解对方的意思。计算机没有理解语言的能力，必须分毫不差地按照语法规则编写的程序才能被它执行。如果想编写出合格的程序，必须对该程序设计语言的语法规则有充分的了解。

3.1　注释和空格

3.1.1　注释

注释是在编写程序时，写程序的人对一条语句、程序段或是函数等的解释或提示。例如：

```
task main()
{
    OnFwd(OUT_B, 75);                    //控制 B 端口电动机 75%功率正向旋转
    OnFwd(OUT_C, 75);

    Wait(2000);                          //延时 2s

    OnRev(OUT_BC, 75);

    Wait(2000);

    Off(OUT_BC);                         //关闭 B、C 端口
}
```

这里用到了 C++ 语言中的单行注释“//”。单行注释的特点是简单，注释以符号“//”开始，新的一行到来为结束。意思是单行注释只对本行要注释的内容有效。

还有一种注释格式，代码如下：

```
/*
    名称：主函数
    功能：控制 B、C 端口电动机先前进 2s 再后退 2s
*/
task main()
{
```

```
    OnFwd(OUT_B, 75);
    OnFwd(OUT_C, 75);

    Wait(2000);

    OnRev(OUT_BC, 75);                    /* 控制 B、C 端口电动机以 75%功率正向旋转 */

    Wait(2000);

    Off(OUT_BC);
}
```

较早版本的 C 语言中，只能用这种类型的注释，以“/*”开始、“*/”结束。所以说“/*”和“*/”是成对出现的。在写这种注释的时候通常先打出一对“/**/”，然后再在里面添加注释内容。

注意：“/**/”注释中间不能嵌套该类型的注释，如/* ABCD/* EDG */HIJK */。这段注释的有效部分是/* ABCD/* EDG */，后面的 HIJK */不属于注释部分。这样编译，将会导致编译错误。所以在注释一段标有注释的代码时要格外注意这一点。

前面所写的程序均比较简单，即使没有注释也不会有特别难懂的地方。随着以后学习的知识越来越多，编写的程序越来越复杂，要养成添加注释的好习惯。因为这样不仅可以使别人更容易理解代码的意思，也便于自己以后维护。

3.1.2　空格

在编写程序时，一定要注意代码的规范格式。请看下面代码。

```
task main(){OnFwd(OUT_B, 75);OnFwd(OUT_C, 15);
Wait(5000);OnFwd(OUT_B, 15);OnFwd(OUT_C, 75);
Wait(5000);Off(OUT_BC);}
```

上面代码是习题 1-2 中的走 S 形的代码。这段代码可以编译通过。对于计算机来说，这两种代码是一样的，唯一不同的是代码中空格和回车的多少。不过对于编译器来说，语句间的回车和空格符统统被忽略掉。

虽然说程序语法正确与否与程序语句间的空格多少并无关系，但是可以看到，上面代码阅读起来非常费劲。所以通常在书写代码时除了要在难以理解的代码段添加注释外，还要特别注意编程规范。下面就对这段代码中需要注意的相关格式及编程规范做一下注释，供大家参考。

【例 3-1】　编程规范及格式示范代码。

```
task main()
{   //花括号顶头写，且要和下面的花括号对齐

    OnFwd(OUT_B, 75);                          //语句前要用 Tab 键进行缩进
```

```
    OnFwd(OUT_C, 15);                    //语句中出现逗点","符号通常后面留一个空格
    Wait(5000);

    //程序段分块写,用换行隔开
    OnFwd(OUT_B, 15);
    OnFwd(OUT_C, 75);
    Wait(5000);

    Off(OUT_BC);
}
```

在书写程序的时候,通常将成对的符号先打出来,然后在里面填写代码,如{}、()、/**/。还要注意,代码中除了注释中会出现中文符号外,代码部分全部都是英文半角符号。

编程规范是具备编写漂亮代码最基本的要求,所以希望读者在一开始就养成良好的编程习惯,千万不要吝啬在代码中添加空格和回车。

3.2 常量与变量定义

在接触一门新技术时,往往第一感觉都是很难。但用得多了,理解它的本意后就会觉得不过如此。任何一门技术都不难,难的是接触一个新的领域时,首先要接受各种陌生的词汇。

3.2.1 常量和符号常量

在程序运行过程中,其值不能改变的量称为常量。在 C 语言中常量有不同类型,如 1、2、3 为整型常量,1.23、11.3 则为实型常量,在编程语言中也称小数为浮点数。当然也有字符型常量,如 a、B、C。如果想在屏幕上输出字母信息,就会用到字符型常量。

下面是我们写过的第一个程序。大家可以发现,这里面用了 5 个常量,即 75 和 2000 共出现 5 次。

```
task main()
{
    OnFwd(OUT_B, 75);
    OnFwd(OUT_C, 75);

    Wait(2000);

    OnRev(OUT_BC, 75);

    Wait(2000);

    Off(OUT_BC);
}
```

这样使用常量有一个弊端,如果对相同的常量进行修改,往往得修改多个地方。假设

现在想让电动机功率都输出为 80，这样得修改 3 个地方。用一个标识符代表一个常量，可以解决上面大量使用相同常量需要修改时所带来的麻烦。

【例 3-2】　用标识符来代替常量代码。

```
#define RS 75
#define TIM 2000

task main()
{
    OnFwd(OUT_B, RS);
    OnFwd(OUT_C, RS);

    Wait(TIM);

    OnRev(OUT_BC, RS);

    Wait(TIM);

    Off(OUT_BC);
}
```

程序中用 # define 命令行定义 RS 和 TIM 代表常量 75 和 2000，之后程序中只要出现 RS 都代表 75，出现 TIM 都代表 2000。这样编译的程序和之前程序实现的功能是一样的。这种用一个标识符代表一个常量的符号，称为符号常量。

请注意符号常量和变量不同。符号常量在其作用域内（本例中为主函数）是不能改变的，也不能再被赋值。后面会介绍变量。例如，下面的赋值操作是错误的。

```
RS=80;                                    //错误
```

或许读者还不理解赋值的含义。在 NXC 语言中，把该符号“=”称为赋值号。在此“=”不是数学上的等号。赋值号主要用于给变量赋值。初次理解这样的概念或许会有些抽象，不过往后慢慢就会习惯。

注意：习惯上，符号常量名用大写，变量名用小写，以示区别。

使用符号常量时，只需改动一处就能将程序中所有相同常量都进行修改。例如，将 # define RS 75 改为

```
#define RS 80
```

程序中所有用到 RS 的地方都用 80 代替。用 # define 命令行定义符号常量也称为宏定义。当然，它的用法不仅如此，以后用到的时候再做讨论。

3.2.2　变量

理解变量之前，首先得理解内存中数据的存储。读者不必害怕，理解它要比你想象的简单得多。可以把内存想象成一列格子，用来存放数据，每个格子都有唯一的编号地址。

图 3.1 所示为地址 0000 到 FFFFF 的内存。

变量代表内存中特定属性的一个存储单元。它用来存放数据，也就是变量的值。可以想象成变量代表内存中的一个格子，可以往这个格子里面放数，这个数就表示变量的值。在程序运行期间，变量的值是可以改变的。

一个变量应该有一个名字，以便引用。变量名、变量值和存储单元的关系如图 3.2 所示。

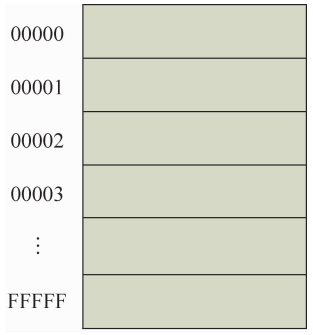


图 3.1 存储单元

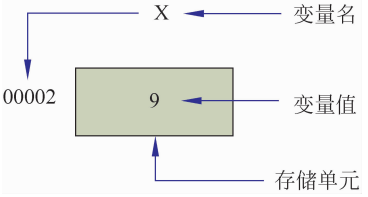


图 3.2 变量和存储单元

变量名实际上是以一个名字对应一个内存单元的地址。在图 3.2 中，可以看成 X 代表地址为 00002 的内存单元。该内存单元中储存了一个数，该数的值为 9。从变量中取值，实际上就是通过变量名找到对应的内存地址，从该存储单元中读取数据。

3.2.3 标识符

NXC 中用来对变量、符号常量、函数、数组等数据对象命名的有效字符序列，统称为标识符(Identifier)。简单地说，标识符就是一个名字。

NXC 中对标识符的命名规则和 C 语言是一样的，只能由字母、数字和下划线 3 种字符组成，且第一个字符必须是字母或下划线。

下面是合法的标识符：

Max、_myInt、max2、_2min 等

下面是不合法的标识符：

3judge、M&G、#%MAX、a<b 等

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 A 和 a 是两个不同的变量名，Max 和 max 也是两个不同的变量名。通常，变量名用小写字母表示。

在选择变量名和其他标识符时，要做到“见名知意”，即选择有一定含义的英文单词或缩写作为标识符，如 max、min、judge、cmp、total、count。除了数值计算外，为了增强函数的可读性，一般不要选择具有代数意义的符号作为变量名，如 x、y、a、b、c。

在 NXC 和 C 语言编程中，要求对所有用到的变量名要做到：先定义，后使用。凡是未被定义的，系统不把它当作变量名。每个变量被指定为一个确定类型，在编译时就为其

分配相应的存储单元。定义变量也称为声明变量。声明一个整型变量可使用以下语句：

```
int sum=30;
```

读者可能还不知道什么是整型变量，在下面数据类型一节具体介绍 NXC 中支持的各种数据类型。

3.3　数 据 类 型

数学里有小数、分数、整数、实数等不同的数据类型，在 NXC 语言中也提供了不同的数据类型，可以根据实际编程需要使用。且我们还可以使用各个数据类型组合一些复杂的组合类型。下面介绍 NXC 语言中的数据类型和具体使用。

3.3.1　变量类型

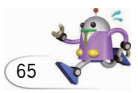
NXC 中有多种数据类型可供使用，如表 3.1 所示，这里罗列了 NXC 语言中所有的数据类型。对于读者来说，只需要大致扫一眼即可，不需要关心细节的含义。例如，8、16 位的含义，还有什么叫字节，多线程等。这里只是把所有的数据类型列出来，以便读者在以后编程时可以查阅方便而已。

表 3.1　数据类型

类型名	类 型 信 息
bool	布尔类型，8 位，无符号
byte	字节类型，8 位，无符号
char	字符类型，8 位，有符号
int	整型，16 位，有符号
short	短整型，16 位，有符号
long	长整型，32 位，有符号
unsigned	用作修饰和定义，无符号类型
float	单精度浮点类型，32 位
mutex	互斥量类型，被使用在多线程设计共享资源的代码中
string	字节为单位的字符串数组
struct	用户自定义的结构体
Arrays	任何变量类型的数组，Arrays 不是关键字，只表示数组的意思

这一小节的标题之所以叫变量类型，是因为在程序中使用的数据类型除了直接写数据外，都是通过变量体现的。例如，要使用一个整数计数，就需要根据计数范围选择是 int 类型还是 long 类型。假定选用了 int 类型，则需要用 int 类型声明一个变量，然后用这个变量完成程序的操作。

可以使用变量类型关键字加变量名定义变量，定义多个相同类型的变量可用逗号(,)隔开，用分号(;)结束定义。变量值可以在变量后面用赋值号(=)给定。



定义变量的例子如下：

```
int x;                                  //声明一个名叫 x 的整型变量
bool y,z;                               //声明 y 和 z 布尔型变量
long a=1, b;                            //声明 a 和 b 长整型变量,给 a 赋初始值 1
float f=1.15, g;                        //声明 f 和 g 为浮点型变量,并初始化 f
int data[10];                           //声明一个 data 数组,包含 10 个整型值且初始值都为 0
bool flags[]={true, true, false, false};
                                        //声明一个布尔类型数组,包含 4 个值且进行初始化
string msg="hello world";               //声明一个字符串类型,并给其赋初始值
```

全局变量在程序块外面声明，它不专属于任何一个任务或函数，它可以在所有任务、函数和子程序中被使用。全局变量的作用域从定义开始到程序的结束。

局部变量被定义在函数和任务中，它的作用域只限定在定义该局部变量的程序块中。局部变量作用于从定义开始到程序块的结束。程序代码中被一组花括号“{}”括起的一组语句称为一个程序块。全局变量和局部变量的定义代码如下：

```
int x;                                  //x 为全局的整型变量
task fun()
{
    int y;                              //y 为 fun 任务中的局部变量
    x=y;                                //正确,x 可以使用 y 对其进行赋值
    {
        int z;                          //声明 z 为该花括号中的局部变量
        y=z;                            //正确,y 变量可以在该程序块中使用
    }
    y=z;                                //错误,z 变量只属于上面的程序块
}
task main()
{
    x=1;                                //正确,x 为全局变量
    y=2;                                //错误,y 是只属于 fun 任务的局部变量
}
```

3.3.2 类型详解

上一小节简单介绍了 NXC 中包含的变量类型，这一节对各变量进行详细讲解。

1. bool 布尔类型

在 NXC 中 bool 类型是无符号 8 位值。使用 bool 类型，通常为其赋值为 true 和 false。true 为 1，false 为 0。但是 bool 类型的取值可以是 0～255(UCHAR_MAX)。

2. byte 字节类型

NXC 中 byte 类型也为无符号 8 位值。该类型可以存储的取值范围为 0～255(UCHAR_MAX)。也可以用 unsigned char 来定义一个无符号 8 位变量，代码如下：

```
byte x=2;                 //定义一个字节类型变量并赋初值
unsigned char y=2;        //定义一个无符号字符类型变量并赋初值,这两个变量都为无符号 8 位
```

3. char 字符类型

NXC 中 char 类型变量为有符号 8 位值。该类型的取值范围为－128(SCHAR_MIN)～127(SCHAR_MAX)。char 类型通常只用来存储简单的 ASCII 码字符类型值，每个 ASCII 码对应一个特定的正整数值。定义字符型的语句如下：

```
char ch='A';              //定义字符型变量 ch,并给其赋值为'A',用来表示大写字母 A
char chx=65;              //定义字符型变量 chx,并给其赋整数值为 65,chx 也表示大写字母 A
                          //因为'A'对应的 ASCII 码值为 65
```

4. int 整型

在 NXC 中，整型变量为有符号 16 位值。该类型的取值范围为－32768(INT_MIN)～32767(INT_MAX)。如果想声明一个无符号 16 位变量，可以在 int 前面加 unsigned 关键字。无符号整型的取值范围是 0～65535(UINT_MAX)。定义整型变量的代码如下：

```
int x=0xfff;              //定义整型变量 x 并赋初值,0x 表示为十六进制数
int y=-23;                //定义整型变量 y 并为其赋一个负数初值
unsigned int z=62043;     //定义无符号整型变量 z,并为其赋初值
```

5. short 短整型

在 NXC 中，short 类型也为有符号 16 位值。该类型的取值范围是－32768(SHRT_MIN)～32767(SHRT_MAX)。在 NXC 中，short 类型其实就相当于 int 类型的别名，但是在目前一些主流 C/C++ 编译器中已经将 int 默认为 4 字节 32 位类型。定义 short 类型变量的代码如下：

```
short x=0xfff;            //定义短整型变量 x 并赋初值,注意最大正整数值不能超过 32767
short y=-23;              //定义短整型变量 y 并赋负数初值,注意负数最小不能低于-32768
```

6. long 长整型

在 NXC 中，long 类型为有符号 32 位值。该类型的取值范围是－2147483648(LONG_MIN)～2147483647(LONG_MAX)。如果想定义一个无符号 32 位值类型，可以在 long 前面加 unsigned 关键字。无符号长整型的取值范围为 0～4294967295(ULONG_MAX)。定义长整型的代码如下：

```
long x=2147000000;            //定义长整型变量 x,并赋初值
long y=-88235;                //定义长整型变量 y,并给其赋负数初值
unsigned long b=4294860000;   //定义无符号长整型变量 z,并赋一个比 LONG_MAX 大的初值
```

7. unsigned 无符号型

使用 unsigned 关键字主要用来修饰 char、int 和 long，用来定义无符号类型。因为无符号类型的数据不用拿出高位区分正负，所以它可以存储全 8、16 和 32 位数据。它可以存储相当于原先最大正值 2 倍的数值。定义无符号 char、int 和 long 类型的代码如下：

```
unsigned char uc=0xff;              //定义无符号字符型变量，并赋最大初值 UCHAR_MAX
unsigned int ui=0xffff;             //定义无符号整型变量，并赋最大初值 UINT_MAX
unsigned long ul=0xffffffff;        //定义无符号长整型变量，并赋最大初值 ULONG_MAX
```

8. float 浮点类型

NXC 浮点类型为 32 位 IEEE 754 单精度标准（IEEE 754 是浮点数表示的统一标准）。该类型的为 4 字节，即用 32 位二进制表示，重要的是它的精度为 24 位，还有 7 位为小数部分。

使用浮点数的算法要比使用整型的操作慢一些，但是如果想存储小数，使用浮点型数据会很方便。标准的 NXT 固件提供了 sqrt()二次求根函数。增强版的 NBC/NXC 固件包含了很多标准的 C math 库，用来对浮点数进行操作。

```
float pi=3.14159;                   //定义一个浮点变量并为其赋初值
float e=2.71828;
float s2=1.4142;
```

9. mutex 互斥量

在 NXC 中，互斥量为 32 位的值，它主要用来同步多线程之间共享资源的存取。所以，互斥量变量都是以全局变量的形式定义的。当有多个任务或函数去获取一个共享资源时，所有的函数和任务可以通过 Acquire 或 Release 去独占资源，从而实现多任务的同步。下面编写程序，以使用互斥来实现多任务同步。

【例 3-3】 使用互斥量实现多任务同步程序。

```
mutex motorMutex;                           //定义一个互斥量
task t1()
{
    while(true)
    {
        Acquire(motorMutex);

        //共享资源代码代码段，使用互斥量进行保护

        Release(motorMutex);
        Wait(MS_500);
    }
}
task t2()
{
    while(true)
```

```
    {
        Acquire(motorMutex);

        //共享资源代码代码段,使用互斥量进行保护

        Release(motorMutex);
        Wait(MS_200);
    }
}
task main()
{
    Precedes(t1, t2);              //启动线程 t1 和 t2
}
```

10. string 字符串

在 NXC 中提供的 string 变量类型可以很方便地定义和操作一个字符串，string 类型可以看成是以 0 或 null 结尾的字节数组。

可以将字符串信息输出到 NXC 邮箱、文件或者 LCD 屏幕。可以在定义或使用时通过常字符串形式初始化 string 类型字符串，代码如下：

```
string msg="Testing";              //定义字符串 msg,并对其进行初始化
string ff;
ff="Fred Flintstone";              //直接通过字符串加赋值号对 string 类型进行赋值
```

3.3.3　结构体

NXC 语言支持用户自定义集合类型，即结构体类型。通常用代码模拟一个任务的时候，任务里面的属性往往不止一个，所以将其属性集中到一起就变成了结构体类型。在 NXC 中定义结构体的方式是固定的，都是使用 struct 关键词声明并定义结构体类型代码，代码如下：

```
struct car                         //声明一个汽车结构体
{
    string car_type;
    int hp;
};
struct person                      //声明一个人结构体
{
    string name;
    int age;
    car vehicle;                   //人结构体中可以包含已经定义过的汽车结构体
};
person myPerson;                   //定义一个人的实体变量 myPerson
```

当定义了一个结构体类型后，可以用这个新的结构体类型定义变量，也可以将这个结构体类型嵌套在另一个结构体中定义。在上面代码中先定义了一个汽车类型的结构体，