第 4 章　函　　数

内容提要

IAP15W4K58S4 单片机的程序设计主要采用两种语言：汇编语言和高级语言（C51）。函数是 C 语言程序的基本单元，一个 C 语言程序就是由一个个具有特定功能的 C 函数构成的。

在本章中，一是学习 C 语言程序结构，从宏观上了解 C 语言程序设计；二是学习 C 函数的定义与调用，从框架上理解 C 函数以及 C 函数在 C 程序设计中的重要性。

4.1　C 语言程序的结构

1．C 语言程序结构形式

```
#include<reg51.h>                          //包含命令
#include<intrins.h>
#define   uchar unsigned char              //宏定义
#define   uint unsigned int
uint time;                                 //全局变量定义
float fun_1(uint a, uint b);               //函数声明
/*－－－自定义函数 1－－－－*/
float fun_1(uint a, uint b)                //函数首部
{                                          //函数体
    声明部分
    执行部分
}
    ┆
/*－－－自定义函数 n－－－－*/
int fun_2(uchar x, uchar y)                //函数首部
{                                          //函数体
    声明部分
    执行部分
}
/*－－－主函数－－－－*/
void main(void)                            //函数首部
{                                          //函数体
    声明部分
    执行部分
}
```

2．C 程序结构说明

（1）一个 C 程序包括 3 大部分：预编译命令部分、全局声明部分和函数定义部分。

① 预编译命令包括文件包含命令（#include）、宏定义与宏定义的撤销(#define、#undef)和条件编译（#if、#else、#endif）。

② 全局声明包括变量声明和函数声明，全部变量声明是指在函数之外进行变量声明，即在函数外定义的变量为全局变量，反之在函数内部定义的变量称为局部变量；当一个函数调用另一个函数时，被调用函数必须先声明，被调用函数的声明既可以在调用函数中声明，也可以在调用函数的前面进行声明，在函数外部声明时，一般放在预编译命令之后，函数定义之前声明。

③ 函数是 C 语言程序的基本单位，一个 C 语言程序可包含多个不同功能的函数，但一个 C 语言程序中只能有一个且必须有一个名为 main()的主函数。主函数的位置可在其他功能函数的前面、之间或最后。当功能函数位于主函数的后面位置时，在主函数调用时，必须“先声明”。

C 语言程序总是从 main()主函数开始执行。主函数可通过直接书写语句或调用功能子函数来完成任务。功能子函数可以是 C 语言本身提供的库函数，也可以是用户自己编写的函数。

（2）注释：注释不是 C 程序所必需的，只是为了便于阅读而设置，有两种注释方式。

① 以//开始的单行注释，这种注释可以单独占一行，也可以出现在一行中其他内容的右侧。

② 以/*开始、以*/结束的块式注释，这种注释可以包含多行内容。编译系统会将一个/*开始符与下一个*/结束符之间的内容作为注释。

3. 函数结构

一个函数包括两部分：函数首部与函数体。

（1）函数首部。函数首部即为函数的第一行，包括函数类型、函数名、函数的参数列表（函数参数类型、函数参数名）。

（2）函数体。函数体是指函数首部下方花括号内的部分，又分为声明部分和执行部分。

① 声明部分包括定义在本函数中所用到的变量和对本函数所调用函数的声明。

② 执行部分由若干个语句组成，指定在函数中所进行的操作。

4. 库函数与自定义函数

库函数是针对一些经常使用的算法，经前人开发、归纳、整理形成的通用功能子函数。ANSI C 提供了一百多个标准库函数，不同的 C 编译系统除提供标准库函数外，还提供一些专门的应用函数，如 Keil C 则包含了针对 8051 单片机应用编程的库函数。

自定义函数是用户自己根据需要而编写的子函数。

4.2 预处理命令

所谓编译预处理，是编译器在对 C 语言源程序进行正常编译之前，先对一些特殊

的预处理命令作解释，产生一个新的源程序。编译预处理主要是为程序调试、程序移植提供便利。

在源程序中，为了区分预处理命令和一般的 C 语句的不同，所有预处理命令行都以符号“#”开头，并且结尾不用分号。预处理命令可以出现在程序任何位置，但习惯上尽可能地写在源程序的开头，其作用范围从其出现的位置到文件尾。

C 语言提供的预处理命令主要有：文件包含、宏定义和条件编译。

1. 文件包含

文件包含实际上就是一个源程序文件可以包含另外一个源程序文件的全部内容。文件包含不仅可以包含头文件，如#include < reg51.h>，还可以包含用户自己编写的源程序文件，如#include“MY__PROC.C”。

C51 文件中首先必须包含有关 8051 单片机特殊功能寄存器地址以及位地址定义的头文件，如#include < reg51.h >。针对增强型 51 单片机，可以采用传统 8051 单片机的头文件，然后再用 sfr、sfr16、sbit 对新增特殊功能寄存器和可寻址位进行定义；也可将用 sfr、sfr16、sbit 对新增特殊功能寄存器和可寻址位进行定义的指令添加到 REG51.H 头文件中，形成该款单片机的头文件，预处理时，将 REG51.H 换成该单片机的头文件即可。

温馨提示：STC-ISP 在线编程软件中提供了 STC 单片机头文件生成工具，只需选择好所使用单片机的系列，就能自动生成该单片机系列的头文件，命名保存即可。

Keil C51 编译器中提供了许多库函数，这些库函数里的函数往往是最常用的、高水平的、经过反复验证过的，所以应尽量直接调用，以减少程序编写的工作量并降低出错的概率。为了使用现成的库函数，一般应在程序的开始处用预处理命令#include< >将有关函数说明的头文件包含进来，这样就不用再另外说明了。Keil C51 中常用库函数如表 4.1 所示。

表 4.1　Keil C51 中常用库函数

头文件名称	函数类型	头文件名称	函数类型
CTYPL .H	字符函数	ABSACC.H	绝对地址访问函数
STDIO.H	一般 I / O 函数	INTRINS.H	内部函数
STRING.H	字符串函数	STDARG.H	变量参数表
STDLIB.H	标准函数	SETJMP.H	全程跳转
MATH.H	数学函数	REG51.H	8051 单片机特殊功能寄存器地址定义的头文件

（1）文件包含预处理命令的一般格式。文件包含预处理命令的一般格式为：

```
#include <文件名>
```

或

```
# include "文件名"
```

上述两种方式的区别是：前一种形式的文件名用尖括弧括起来，系统将在包含 C 语言库函数的头文件所在的目录（通常是 KEIL 目录中的 include 子目录）中寻找文件；后一种形式的文件名用双引号括起来，系统先在当前目录下寻找，若找不到，再到其他路径中寻找。

（2）文件包含使用注意。

① 一个#include 命令只能指定一个被包含的文件。

② 如果文件 1 包含了文件 2，而文件 2 要用到文件 3 的内容，则在文件 1 中用两个#include 命令分别包含文件 2 和文件 3，并且文件 3 包含要写在文件 2 的包含之前，即在 file1.c 中定义：

```
# include<file3.c>
# include<file2.c>
```

③ 文件包含可以嵌套。 在一个被包含的文件中又可以包含另一个被包含文件。

包含文件包含命令为多个源程序文件的组装提供了一种方法。在编写程序时，习惯上将公共的符号常量定义、数据类型定义和 extern 类型的全局变量说明构成一个源文件，并以“.H”为文件名的后缀。如果其他文件用到这些说明时，只要包含该文件即可，无须再重新说明，减少了工作量。而且这样编程使得各源程序文件中的数据结构、符号常量以及全局变量形式统一，便于程序的修改和调试。

2. 宏定义

宏定义分为带参数的宏定义和不带参数的宏定义。

（1）不带参数的宏定义。不带参数宏定义的一般格式为：

```
#define 标识符   字符串
```

它的作用是在编译预处理时，将源程序中所有标识符替换成字符串。例如，

```
#define   PI   3.148                  //PI 即为 3.148
#define uchar unsigned   char        //在定义数据类型时，uchar 等效于 unsigned   char
```

当需要修改某元素时，只要直接修改宏定义即可，无须修改程序中所有出现该元素的地方。所以，宏定义不仅提高了程序的可读性，便于调试，同时也方便了程序的移植。

无参数的宏定义使用时，要注意以下几个问题：

① 宏名一般用大写字母，以便于与变量名的区别。当然，用小写字母也不为错。

② 在编译预处理中宏名与字符串进行替换时，不进行语法检查，只是简单的字符替换，只有在编译时才对已经展开宏名的源程序进行语法检查。

③ 宏名的有效范围是从定义位置到文件结束。如果需要终止宏定义的作用域，可以用#undef 命令。例如，

```
#undef   PI    //该语句之后的 PI 不再代表 3.148，这样可以灵活控制宏定义的范围
```

④ 宏定义时可以引用已经定义的宏名。例如，

```
#define   X   2.0
#define   PI   3.14
```

```
#define   ALL   PI*X
```

⑤ 对程序中用双引号括起来的字符串内的字符，不进行宏的替换操作。

（2）带参数的宏定义。为了进一步扩大宏的应用范围，在定义宏时还可以带参数。带参数的宏定义的一般格式为：

```
# define   标识符（参数表）   字符串
```

它的作用是在编译预处理时，将源程序中所有标识符替换成字符串，并且将字符串中的参数用实际使用的参数替换。例如，

```
# define   S(a, b）  (a*b) / 2
```

若程序中使用了 S(3, 4)，在编译预处理时将替换为(3*4) / 2。

3. 条件编译

条件编译命令允许对程序中的内容选择性地编译，即可以根据一定的条件选择是否编译。条件编译命令主要有以下几种形式：

（1）形式 1。

```
# ifdef 标识符
程序段 l
# else
程序段 2
# endif
```

它的作用是当“标识符”已经由#define 定义过了，则编译“程序段 1”，否则编译“程序段 2”。其中，如果没有“程序段 2”，则上述形式可以变换为：

```
# ifdef 标识符
程序段 l
# endif
```

（2）形式 2。

```
# ifndef 标识符
程序段 1
# else
程序段 2
# endif
```

它的作用是当“标识符”没有由#define 定义过，则编译“程序段 1”，否则编译“程序段 2”。同样，若无“程序段 2”时，则上述形式变换为：

```
# ifndef 标识符
程序段 1
# endif
```

（3）形式 3。

```
# if 表达式
程序段 1
#else
程序段 2
# endif
```

它的作用是当“表达式”值为真时，编译“程序段 1”，否则编译“程序段 2”。同样当无“程序段 2 " 时，则上述形式变换为：

```
#if 表达式
程序段 1
#endif
```

以上 3 种形式的条件编译预处理结构都可以嵌套使用。当# else 后嵌套#if 时，可以使用预处理命令# elif，它相当于# else　# if。

在程序中使用条件编译主要是为了方便程序的调试和移植。

2. 全局变量定义与函数声明

（1）全局变量的定义。全局变量是指在程序开始处或各个功能函数的外面所定义的变量，在程序开始处定义的变量在整个程序中有效，可供程序中所有的函数共同使用；在各功能函数外面定义的全局变量只对定义处开始往后的各个函数有效，只有从定义处往后的各个功能函数可以使用该变量。

当有些变量是整个程序都需要使用的，例如，LED 数码管的字形码或位码。这时，有关 LED 数码管的字形码或位码的定义就应放在程序开始处。

（2）函数声明。一个 C 语言程序可包含多个不同功能的函数，但一个 C 语言程序中只能有一个且必须有一个名为 main()的主函数。主函数的位置可在其他功能函数的前面、之间或最后。当功能函数位于主函数的后面位置时，在主函数调用时，必须对各功能函数“先声明”，一般放在程序的前面。例如，

```
#include <REG51.H>
void   delay(void);                    //声明子函数
void   light1(void);                   //声明子函数
void   light2(void);                   // 声明子函数
/*－－－－－－－－－－主函数－－－－－－－－－－－－*/
void main(void)
{
    while(1)
    {
        light1();
        delay();
        light2();
        delay();
    }
```

```
}
/*－－－－－－－－－－各功能函数略－－－－－－－－－－－－*/
```

主函数调用了 light1()、delay()、light2()，而且 light1()、delay()、light2()三个功能函数在主函数的后面，在主函数前必须对 light1()、delay()、light2()先进行声明。

若功能函数位于主函数的前面位置时，就不必对各功能函数“声明”。

（3）函数的连接。实际上，函数的连接也是一种函数声明。当程序中子函数与主函数不在同一个程序文件时，要通过连接的方法实现有效的调用。一般有两种方法，即外部声明与文件包含。

① 外部声明。例如，

```
#include <REG51.H>
extern void delay(void);              //声明该函数在其他文件中/
extern void light1(void);             //声明该函数在其他文件中/
extern void light2(void);             //声明该函数在其他文件中/
/*－－－－－－－－－－－－－－－－－－－－－－－－*/
void main(void)
{
    while(1)
    {
        light1();
        delay();
        light2();
        delay();
    }
}
```

本程序文件只有主函数，主函数调用的 light1()、delay()、light2()都不在本文件中，因此，在主函数调用前需用关键字“extern”对被调用函数进行外部声明，告诉编译系统该函数在其他 C 程序文件中。子函数程序文件和主函数程序应在同一文件夹下。

② 文件包含。同上例，假设主函数调用的 light1()、delay()、light2()等函数在 test4b.c 文件中，除上述外部声明的方法对被调函数进行声明外，还可以采用文件包含的方法，对被调函数进行声明。实际上，文件包含就是把被包含文件放在包含命令位置处。

```
#include <REG51.H>
#include    " test4b.c "              //包含 light1()、delay()、light2()等函数所在文件
void main(void)
{
    while(1)
    {
        light1();
        delay();
        light2();
        delay();
    }
}
```

4.3 函数的定义

4.3.1 函数的分类

1. 从函数定义的角度看，函数可分为库函数和用户定义函数两种

（1）库函数：由C系统提供，用户无须定义，也不必在程序中作类型说明，只需在程序前包含有该函数原型的头文件即可在程序中直接调用。在前面用到的_crol_()、_nop_()等函数均属此类，有关_crol_()、_nop_()函数的声明就在 intrins.h 的头文件中。

（2）用户定义函数：由用户按需要写的函数。对于用户自定义函数，不仅要在程序中定义函数本身，而且在主调函数模块中还必须对该被调函数进行类型说明，然后才能使用。

2. 从函数是否有返回值的角度看，函数可分为有返回值函数和无返回值函数两种

（1）有返回值函数：此类函数被调用执行完后将向调用者返回一个执行结果，称为函数返回值。如数学函数即属于此类函数。由用户定义的这种要返回函数值的函数，必须在函数定义和函数说明中明确返回值的类型。

（2）无返回值函数：此类函数用于完成某项特定的处理任务，执行完成后不向调用者返回函数值。这类函数类似于其他语言的过程。由于函数无须返回值，用户在定义此类函数时可指定它的返回值为“空类型”， 空类型的说明符为“void”。

3. 从主调函数和被调函数之间数据传送的角度看，函数可分为无参函数和有参函数两种

（1）无参函数：函数定义、函数说明及函数调用中均不带参数。主调函数和被调函数之间不进行参数传送。此类函数通常用来完成一组指定的功能，可以返回或不返回函数值。

（2）有参函数：也称为带参函数。在函数定义及函数说明时都有参数，称为形式参数（简称为形参）。在函数调用时也必须给出参数，称为实际参数（简称为实参）。进行函数调用时，主调函数将把实参的值传送给形参，供被调函数使用。

注意，在C语言中，所有的函数定义，包括主函数 main 在内，都是平行的。也就是说，在一个函数的函数体内，不能再定义另一个函数，即不能嵌套定义。但是函数之间允许相互调用，也允许嵌套调用。习惯上把调用者称为主调函数。函数还可以自己调用自己，称为递归调用。main 函数是主函数，它可以调用其他函数，而不允许被其他函数调用。因此，C程序的执行总是从 main 函数开始，完成对其他函数的调用后再返回到 main 函数，最后由 main 函数结束整个程序。一个C源程序必须有，也只能有一个主函数 main。

4.3.2 函数的定义

1. 无参函数的定义形式

类型标识符 函数名()

```
{
    声明部分;
    语句部分;
}
```

其中类型标识符和函数名称为函数头。类型标识符指明了本函数的类型，函数的类型实际上是函数返回值的类型。该类型标识符与前面介绍的各种说明符相同。函数名是由用户定义的标识符，函数名后有一个空括号，其中无参数，但括号不可少。

{}中的内容称为函数体。在函数体中声明部分，是对函数体内部所用到的变量的类型说明。

在很多情况下都不要求无参函数有返回值，此时函数类型符可以写为 void。

我们可以书写一个函数的定义如下：

```
void Bio()                  //无参函数，无返回值，函数名为 Bio。
{
    unsigned char x;        //变量定义
    P1=0xff;                //执行语句
    x=P1;                   //读 P1 口输入数据，存变量 x 中
    P2=x;                   //变量 x 的数据送 P2 口输出
}
```

2．有参函数的定义形式

```
类型标识符 函数名(形式参数表列)
{
    声明部分;
    语句部分;
}
```

有参函数比无参函数多了一个内容，即形式参数表列。在形参表中给出的参数称为形式参数，它们可以是各种类型的变量，各参数之间用逗号间隔。在进行函数调用时，主调函数将赋予这些形式参数实际的值。形参既然是变量，必须在形参表中给出形参的类型说明。

例如，定义一个函数，用于求两个数中的大数，可写为：

```
int max(int a, int b)
{
    if (a>b) return a;
    else return b;
}
```

第一行说明 max 函数是一个整型函数，其返回的函数值是一个整数。形参为 a、b，均为整型变量。a、b 的具体值是由主调函数在调用时传送过来的。在{}中的函数体内，除形参外没有使用其他变量，因此只有语句而没有声明部分。在 max 函数体中的 return 语句是把 a（或 b）的值作为函数的值返回给主调函数，有返回值函数中至少应有一个 return 语句。

4.4 函数的调用

1. 函数调用的一般形式

C 语言中，函数调用的一般形式为：

```
函数名(实际参数表);
```

如果是对无参函数调用时则无“实际参数表”，但是括号不能省略。实际参数表中的参数可以是常数、变量或其他构造类型数据及表达式。如果有多个实参，各参数之间用逗号分隔。

（1）形参变量只有在被调用时才分配内存单元，在调用结束时，即刻释放所分配的内存单元。因此，形参只有在函数内部有效。函数调用结束返回主调函数后则不能再使用该形参变量。

（2）实参可以是常量、变量、表达式、函数等，无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值，以便把这些值传送给形参。因此应预先用赋值、输入等办法使实参获得确定值。

（3）实参和形参在数量上、类型上、顺序上应严格一致，否则会发生“类型不匹配”的错误。

（4）函数调用中发生的数据传送是单向的。即只能把实参的值传送给形参，而不能把形参的值反向地传送给实参。

2．函数调用的方式

按函数在程序中出现的位置来分，可有以下 3 种函数调用方式：

（1）函数表达式：函数作为表达式中的一项出现在表达式中，以函数返回值参与表达式的运算。这种方式要求函数是有返回值的。例如，

```
z=max(x,y);   // max 函数的返回值赋予变量 z
```

（2）函数语句：函数调用的一般形式加上分号即构成函数语句。例如，

```
delay(50);   //设 delay(50)是一个 50ms 的延时函数，调用时产生 50ms 的延时
```

（3）函数实参：函数作为另一个函数调用的实际参数出现。这种情况是把该函数的返回值作为实参进行传送，因此要求该函数必须是有返回值的。例如，

```
k= max(12,max(23,25)）; //调用 max(23,25)的返回值又作为 max 函数的一个参数
```

3. 函数的嵌套调用

C 语言中不允许作嵌套的函数定义。因此各函数之间是平行的，不存在上一级函数和下一级函数的问题。但是 C 语言允许在一个函数的定义中出现对另一个函数的调用，这样就出现了函数的嵌套调用，即在被调函数中又调用其他函数。这与其他语言的子程序嵌套的情形是类似的，其关系可表示如图 4.1 所示。

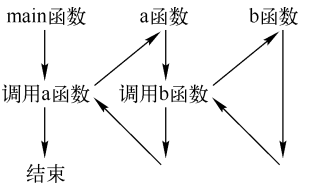

图 4.1　函数的嵌套调用

图 4.1 表示了两层嵌套的情形。其执行过程是：执行 main 函数中调用 a 函数的语句时，即转去执行 a 函数，在 a 函数中调用 b 函数时，又转去执行 b 函数，b 函数执行完毕返回 a 函数的断点继续执行，a 函数执行完毕返回 main 函数的断点处继续执行。

4. 被调用函数的声明

在主调函数中对被调用函数作说明的目的是使编译系统知道被调函数返回值的类型，以便在主调函数中按此种类型对返回值进行相应的处理。

其一般形式为：

```
类型说明符 被调函数名(类型 形参，类型 形参);
```

或者为：

```
类型说明符 被调函数名(类型，类型);
```

括号内给出了形参的类型和形参名，或只给出形参类型。这便于编译系统进行检错，以防止可能出现的错误。

如果使用库函数，还应该在程序开头用#include 命令将调用有关库函数所用到的信息包含进来，例如，

```
#include <reg51.h>
```

其中“reg51.h”是一个头文件，在 reg51.h 文件中包含了 8051 单片机特殊功能寄存器地址与特殊功能寄存器位地址的定义。同样，如要使用数学函数，应该用#include<math.h>在程序开头来声明。其中，.h 是头文件所用的后缀，表示该文件是头文件(header file)。

4.5 函数的递归调用与再入函数

C 语言中允许在调用一个函数过程中，又间接或直接地调用该函数自己，这称为函数的递归调用。递归调用可以使程序简洁，代码紧凑，但速度会稍慢，并且要占用较大的堆栈空间。

在 C51 中，采用一个扩展关键字 reentrant，作为定义函数时的选项，从而构造成再入函数，使其在函数体内可以直接或间接地调用自身函数，实现递归调用。需要将一个函数定义为再入函数时，只要在函数名后面加上关键字 reentrant 就可以了，格式如下：

```
函数类型 函数名（形式参数表）［reentrant］
```

C51 对再入函数有如下的规定：

（1）再入函数不能传送 bit 类型参数，也不能定义一个局部位标量。再入函数不能包括位操作以及 8051 单片机的可位寻址区。

（2）编译时在存储器模式的基础上为再入函数在内部或外部存储器中建立一个模拟堆栈区，称为再入栈。再入函数的局部变量及参数被放在再入栈中，从而使再入函数可以进行递归调用。而非再入函数的局部变量被放在再入栈之外的暂存区内，如果对非再入函数进行递归调用，则上次调用时使用的局部变量数据将被覆盖。

（3）在参数的传递上，实际参数可以传递给间接调用的再入函数。无再入属性的间接调用函数不能包含调用参数，但是可以定义全局变量来进行参数传递。

*4.6 中断函数

中断函数是一种特殊函数，针对单片机的硬件电路产生的一种操作。

1. 中断服务函数的定义

中断服务函数定义的一般形式为：

函数类型 函数名（形式参数表）［interrupt n］ ［using m］

其中，关键字 interrupt 后面的 n 是中断号，n 的取值范围为 0～31。编译器从 8n＋3 处产生中断向量，具体的中断号 n 和中断向量取决于不同的单片机芯片。

关键字 using 用于选择工作寄存器组，m 为对应的寄存器组号，m 取值为 0～3，对应 8051 单片机的 0～3 寄存器组。

2．8051 单片机中断源的中断号与中断向量

8051 单片机中断源的中断号与中断向量如表 4.2 所示。

表 4.2 8051 单片机中断源的中断号与中断向量

中断源	中断号 n	中断向量地址 8n+3
外部中断 0	0	0003H
定时/计数器中断 0	1	000BH
外部中断 1	2	0013H
定时/计数器中断 1	3	001BH
串行口中断	4	0023H

注：IAP15W4K58S4 单片机有更多的中断源，各中断源的中断号以及向量地址详见第 10 章。

3. 中断服务函数的编写规则

（1）中断函数不能进行参数传递，如果中断函数中包含任何参数声明都将导致编译出错。

（2）中断函数没有返回值，如果企图定义一个返回值将得到不正确的结果。因此，最好定义中断函数时将其定义为 void 类型，以明确说明没有返回值。

（3）在任何情况下都不能直接调用中断函数，否则会产生编译错误。因为中断函数的返回是由 8051 单片机指令 RETI 完成的，RETI 指令影响 8051 单片机的硬件中断系统。

（4）如果中断函数中用到浮点运算，必须保存浮点寄存器的状态，当没有其他程序执行浮点运算时可以不保存。

（5）如果在中断函数中调用了其他函数，则被调用函数所使用的寄存器组必须与中断函

数相同。用户必须保证按要求使用相同的寄存器组，否则会产生不正确的结果。如果定义中断函数时没有使用 using 选项，则由编译器选择一个寄存器组作为绝对寄存器组访问。

*：中断函数的学习可放在单片机中断系统章节中学习。

习 题 4

4.1 在 C 语言程序中，哪个函数是必须的？C 语言程序的执行顺序是如何决定的？

4.2 主函数的函数名是什么？当主函数与子函数在同一个程序文件时，调用时应注意什么？当主函数与子函数分属在不同的程序文件时，调用时有什么要求？

4.3 函数的调用方式主要有 3 种，请举例说明。

4.4 全局变量与局部变量的区别是什么？如何定义全局变量与局部变量？

4.5 说明下列宏定义、条件编译中关键字的含义：

include define ifdef ifndef undef endif

* 中断函数的学习可放在单片机中断系统章节中学习。