目 录

第一章 C 语言概述

1.1 C 程序结构特点 16

1、C 程序的基本构件——函数。

2、一个函数由函数首部和函数体两部分构成。

- 函数首部一般包括函数类型、函数名、函数参数等。
- 函数体一般包括声明部分和执行部分。其中：在声明部分中定义所用到的变量；执行部分则由若干个语句组成。

3、C 程序只有一个 main 函数，且总是从 main 函数开始执行。

4、C 语言语句必须以“；”结束。

5、用/*　　*/作为注释。

6、C 编译器一般自顶向下顺序编译 C 源程序，如果被调函数定义在主调函数之后位置时，要在主调函数前，给出被调函数的原型说明。以便编译器在编译被调函数的调用语句时，对调用语句进行参数检查。

如果不进行原型说明，则无法通过编译检查。

原型说明：类型说明 函数名（参数类型，参数类型，……）

7、头文件——头文件包含了 C 语言的标准函数库的原型说明。

C 语言通过使用#include <>预处理命令，将库函数的原型说明插入到源文件中。

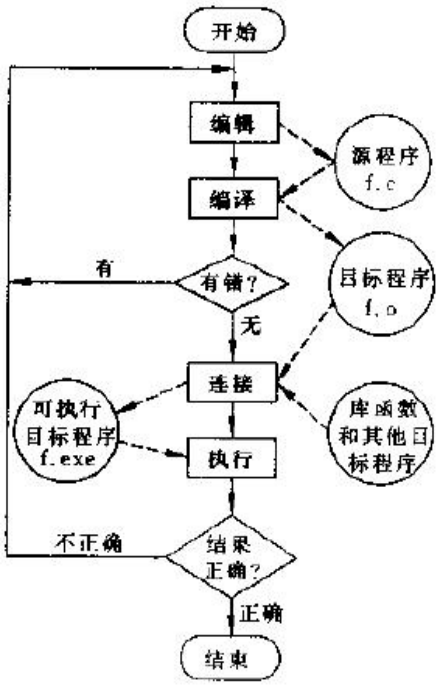
1.2 C 程序上机步骤 17

1、编辑源文件 .c；

2、编译成目标文件.obj；

3、连接——将目标程序和库函数及其他目标程序连接起来，生产可执行文件，文件扩展名为.exe；

4、执行。



结构体、共用体、枚举型。

3、指针类型：用于存储地址的一类数据类型。

4、空类型：

编译器根据变量的数据类型，给变量分配存储单元。

3.3 常量与变量 48

1、符号常量——符号常量在其作用域内不能改变，也不能被赋值。

#define 符号常量（大写） 常量

2、变量——变量由变量名和变量值标识。

数据类型 变量；

- 变量名实际上是一个符号地址，编译连接程序时给每个变量名分配一个内存地址，当程序访问变量值时，实际上是通过变量名找到相应的内存地址，从其存储单元中读取数据。

3.4 基本类型

1、整形数据在内存中的存放，是以补码形式存放的。

2、实型数据：单精度 双精度。

- 在内存中以指数形势存放。
- 若数据超过有效位，则超过 C 语言规定有效位的数据将被舍去，故产生误差。

3、字符型数据：用一个字节存储单元存储。即将字符的 ASCII 码存储到内存单元中。

- 用单引号括起来的一个字符。
- 转义字符——特殊字符常量或者控制字符常量，它们都以“\”开头。
- Char、unsigned char 区别：char 用 7 位数表示，最大表示编码为 127 的字符；unsigned char 用 8 位数表示，最大表示编码为 255 的字符。
- 字符数据与整型数据可以相互赋值。
- 字符数据可以以字符数据输出，也可以以整型数据形式输出。

4、字符串常量：C 语言中，必须是用字符数组来保存字符串常量。在内存中顺序存储。

- 用一对双引号括起来的字符序列。

- 每个字符串常量结尾以一个字符\0 作为结束标记。（一般由系统自动加上）。

3.5 变量 63

1、自动变量——在函数体内或复合语句中定义的非静态变量称为自动变量。

- C 语言编译时，不对自动变量赋初值。
- 当程序执行到自动变量的作用域时，程序才为自动变量分配空间。当定义自动变量的函数或复合语句执行结束后，程序释放自动变量的存储空间。
- 自动变量保存在程序的动态存储空间。

2、静态局部变量——在函数体内或复合语句中用 static 定义的变量称为静态局部变量。

- C 语言编译时，对静态局部变量赋初值。
- 静态局部变量存储在程序的静态存储空间。
- 静态局部变量在程序的整个运行期间均占用程序的静态存储空间，直到程序退出后才释放存储空间。

3、寄存器变量——用 register 声明的变量。

4、外部变量（全局变量）——在函数的外部定义的变量。它的作用域是从定义处开始，到本程序文件的末尾结束，在此作用域内，全局变量可以为程序的各个函数引用。

- C 语言编译时，对全局变量赋初值。
- 全局变量存储在程序的静态存储空间。
- 全局变量在程序的整个运行期间均占用程序的静态存储空间，直到程序退出后才释放存储空间。

注意：

（1）当引用本源文件后面定义的全局变量或引用在其他源文件中定义的全局变量是，应在引用位置前，利用 extern 声明该全局变量，以告诉编译器编译时，引用的是一个外部变量，在编译器连接时，将引用的外部变量的作用域扩展到本文件 extern 声明处。

（2）用 static 声明的全局变量，不能被其他文件引用。

注意：

（1）变量声明分为定义性声明和引用性声明。

（2）一般把建立存储空间的声明称为变量定义，把不需要建立存储空间的声明成为变量声明。

3.6 不同类型数据间的混合运算

- 不同类型数据进行混合运算时，不同类型的数据要先转换成同一类型，按照类型级别由低到高（char，short—int——unsigned—long—double；float—double）的顺序进行转换。
- 强制类型转换——（类型名） 表达式。强制类型转换也就是将存储数据的内存单元强制转换为另一种数据类型的单元大小。即强制将存放数据的内存单元改变。
- 赋值时进行类型转换：将数据复制给变量时，将会将数据强制转换为要赋值变量的类型。一般短类型转换为长类型时，进行符号扩展；长类型转换为短类型时，仅赋值低位，难以保证数据的正确性。

3.7 函数的调用过程（补充）

- C 函数其实就是一个程序模块。
- C 函数在编译时，单独编译成一个指令模块，在函数模块开始处定义保护现场指令，将用到的 CPU 寄存器压入堆栈。在返回时定义了恢复现场指令，将堆栈数据恢复到 CPU 寄存器。
- 在调用函数时，一般利用堆栈传递输入参数；利用 EAX 传递输出参数，注意在函数调用完成后，要维持堆栈平衡，且函数返回输出参数在 EAX 中，在使用输出参数前，不要改变 EAX 的值。

第四章 最简单的 C 程序设计——顺序程序设计 77

4.1 字符数据的输入输出

1、C 语言没有输入输出语句，IO 操作通过调用系统函数实现。

2、在程序的开头，要有：#include “stdio.h”或 #include <stdio.h>，预定义语句，用来引用头文件，在编译时将头文件中的函数原型声明添加到源文件中。

stdio.h 输入输出语句

string.h 字符串操作函数

math.h 定义数字函数

ctype.h 字符函数库

intrins.h 内部函数

stdlib.h 标准函数库

absacc.h 绝对地址访问

reg.h 专用寄存器文件

默认输出设备——显示屏，默认输入设备——键盘。

第五章 选择结构的程序设计

第六章 循环结构程序设计

6.1 语句标号

语句标号——用于定义程序中的某个位置，用标识符表示，不能只用数字。

6.2 break 语句和 continue 语句

1、break 语句结束循环语句和 switch 语句。

2、continue 结束本次循环，即忽略循环体中剩余的语句。

第七章 数组 132

7.1 构造类型

构造类型——是由基本类型按照一定规则构造而成的。（如数组、结构体、共同体、枚举型）

构造类型的每个分量（元素），是一个变量，它可以是一个简单类型或者构造类型。

构造类型的分量占用相邻的存储空间。

7.2 数组 133

1、数组——是有序数据的集合，数据元素类型相同，顺序存储，占用相邻的存储空间。

2、数组——数组必须先定义后引用。静态数组变量定义时编译器自动初始化数据元素为 0，动态数组变量在程序执行时分配存储空间，在未被赋值初始化之前其值随机。

3、C 语言只能逐个引用数组元素，不能一次引用整个数组。

4、数组引用是“数组名+下标”引用，数组下标均为整数。如 a[2]。

7.3 二维数组

1、数组元素为数组。

2、在内存中，C 语言的二维数组中数组元素的存放顺序是按行存放的。

3、二维数组引用是“数组名+下标+下标”。如 a[1][2]。

7.4 字符串——字符数组

1、一般用\0 来标识字符串结尾。\0 占用一个字符变量空间。

2、用字符串赋值字符数组时，C 编译器在字符串后自动加\0 赋给字符数字。

3、字符数组可以一次引用整个字符串数组。如整个字符串（数组）的输入输出，用%s 格式，且在输入字符数组时，用数组名代表数组的首地址，对于二维数组，仅仅写行下标不写列下标，也可以代表给行数组的首地址。

在用%s 输入输出字符串数组时，遇到\0 结束。

7.5 字符串处理函数 #include <string.h>

1、gets（字符数组名）

从键盘输入一个字符串（以回车结束），并返回字符数组的起始地址。

如 get（str）。

2、puts（字符数组名/字符串）

将数组中的字符串（\0 结尾的字符序列）输出到终端上，输完换行。

如 puts（str），puts（“ok”）。

3、strcpy（目的字符数组 1 名，源字符串/字符数组 2 名）

拷贝时，将“\0”一起拷贝过去。

4、strcat（字符数组 1 名，字符串/字符数组 2 名）

将字符串/字符数组 2 连接到字符数组 1 中。

连接时，编译器去掉字符数组 1 的“\0”。

5、strcmp（字符串/字符数组 1 名，字符串/字符数组 2 名）

比较字符串/字符数组 1 名 和字符串/字符数组 2 名的大小。

字符串/字符数组 1>字符串/字符数组 2，返回正数

字符串/字符数组 1=字符串/字符数组 2，返回 0

字符串/字符数组 1<字符串/字符数组 2，返回负数。

1、一个 C 程序可以分为若干个函数。

2、每个程序只能有一个主函数，C 程序的执行从 main 函数开始，从 main 函数结束。

3、函数间可以互相调用，但主函数不能被调用。

4、从用户角度看，函数可分为标准函数（库函数）和自定义函数。其中，编译器仅编译自定义函数，在连接时才将标准库函数的目标代码连接到程序。

5、一个 C 源程序由一个或多个文件构成，一个源程序文件是一个编译单位。

8.2 函数调用参数传递

1、值传递——如数值形参。此时，将实参值复制压栈，被调函数对复制到栈中的数值进行操作，不改变原来实参值。

2、地址传递——如数组形参（指针形参），此时，将实参数组的首地址压栈，被调函数引用实参数组的首地址，找到实参数组，对实参数组进行操作，改变实参数组值。即形参数组和实参数组共享同一单元。

8.3 函数变量的作用范围

1、局部变量——在定义局部变量的范围内有效。当局部变量重名时，有效范围小的优先。

- 在函数内部定义局部变量
- 函数的形式参数

- 在某个复合语句中定义的变量。

2、全局变量——在函数之外定义的变量。有效范围是从定义变量的位置开始到源文件结束。

8.4 变量的存储类别

变量的存储类别，即生存期。内存中供用户使用的存储空间包括：程序区、静态存储区、动态存储区。

1、静态存储区——在编译时分配空间，在程序运行完后才释放存储空间。存储静态局部变量和全局变量。

- 局部静态变量在编译时赋初值，在执行时可改变该值，但该存储空间一直保存到程序结束。
- 定义局部静态变量，如果没有赋初值，编译时会自动赋默认初值。
- 局部静态变量只能在定义它的函数中使用。
- 全局变量都是静态的。
- 利用 extern 外部变量方式表示变量的定义在别的文件中，提示编译器遇到此变量时，在其他模块中寻找其定义。而函数则是利用函数原型来声明。
- 用 static 关键字说明一个不能在其他源文件中引用的全局变量。即静态全局变量在声明它的整个文件都是可见的，但是在文件之外是不可见的。

2、动态存储区——仅在在执行时分配空间，用完后释放存储空间。存放自动变量和形式参数。

- 寄存器变量——用 register 关键字说明。寄存器变量对寄存器的占用是动态的。

第九章 预处理命令 197

9.1 预编译命令作用

预编译命令主要完成宏定义、文件包含、条件编译三种功能。

1、宏定义——指用一个指定的标识符（名字）来代表一个字符串。在预编译时，将宏名替代成字符串的过程称为宏展开。如：

- # define PI 3.1415926 定义宏，
  # undef PI 终止宏定义的作用域。
- #define V(a,b,c) a*b*c 定义带参数的宏。当宏展开时，将引用宏名语句中的实参字符串代替宏定义的形参字符串。
  int v =V(2,3,4) 则宏展开后为：int v= 2*3*4；

2、文件包含——指一个源文件可以将另一个源文件的全部内容包含进来。如：

- #include “文件名”或
  #include <文件名>
- 编译预处理时，将包含文件的全部内容复制到源文件中。在编译时作为一个源程序来编译。

3、条件编译——在预编译处理时，确定编译时要编译的部分。如：

- # ifdef 标识符

  程序段 1

  # else

  程序段 2

  # endif

- # if 表达式

  程序段 1

  # else

  程序段 2

  # endif

第十章 指针 211

10.1 变量的访问方式

1、直接访问——如：int a = 10；

2、间接访问——定义一个指针变量 p，存放变量 a 的首地址，通过 p 访问变量 a。则称指针变量 p 指向变量 a。

如：int a=10；int *p1; p=&a；b=*p1； 或*p1=100；等价于 a=100；

10.2 指针变量

1、指针变量的类型——是指该变量指向的内存数据的数据类型。

2、必须用引用变量的地址给指针变量赋值。

3、可以给指针变量赋值空值 null，防止指针变量存储随机值导致系统错误。

4、数组名代表数据的首地址。数组指针或数组名+1，指向下一个数组元素的存储地址。

声明格式：数据类型 *p ；

赋值格式：p=a；或 p=&a[0]；

引用格式：*p=常量；等价于 p[0]=常量；等价于 a[0]=常量；等价于*a=常量；

*（p+1）=常量；等价于 p[1]=常量；等价于 a[1]=常量；

等价于*（a+1）=常量；

5、数组名和指针变量虽然都可以代表数组的首地址，但是指针变量值可以改变，

而数组名的值不可以改变。

6、函数的指针专门用来存放函数的入口地址，当把函数的地址赋值给它时，它就指向该函数的入口地址。

声明格式：数据类型（*指针变量名）（）如：int （*P）（）；

赋值格式：p=max； 注 max 为定义的函数名；函数名代表该函数的入口地址。

引用格式：c=（*p）（a，b）；等价——c=max（a，b）；

7、只能将变量已分配的地址赋值给指针变量，不能直接将整数赋值给指针变量。

8、指针变量可以有 null 值，防止指针误作用。

第十一章 结构体 270

11.1 结构体 270

1、作用——将不同类型的数据组合成一个有机的整体。

2、结构体的定义——结构体是一种数据结构，按照某种规则定义，将若干个不同数据类型（也可相同）的数据项的组合成的一个有机整体。

3、声明结构体类型的形式：struct 结构体类型名字 {成员列表}；

- 成员列表形式：类型符 成员名 如：int num；

11.2 声明结构体类型变量的方法 271

1、先定义结构体类型：struct 结构体类型名字 {成员列表}；再定义结构体变量：struct 结构体类型名字 结构体变量名。

2、可在定义结构体类型时，定义结构体类型变量。struct 结构体类型名字 {成员列表} 结构体类型变量 1，结构体类型变量 2，……；

11.3 结构体变量引用 273

1、不能将一个结构体变量作为一个整体进行输入输出，只能对结构体变量成员分别赋值。

2、结构体变量成员引用方式：结构体变量名.成员名

3、如果结构体变量成员是另一个结构体变量，则要用若干个成员运算符，一级一级找到最低一级的成员。

11.4 结构体变量初始化

1、在声明变量时整体赋值。

2、复制相同结构的变量赋值。

3、除以上两种情况下可以对结构体类型变量整体赋值，其余情况下只能对变量成员分别赋值。

11.5 结构体数组 275

1、结构体数组在内存中连续存放。

2、声明方式：与定义结构体变量方法相同。见 11.2，只是在变量名后+[n]。也可在定义结构体数组时直接初始化结构体数组，此时可采用+[]的形式。

3、初始化方式：同定义结构体变量方法。

11.6 结构体类型指针

结构体类型指针——是指指向结构体变量所占据的内存的起始位置的指针。

声明形式：struct 结构体类型名 *指针名。如：struct student *p；

赋值形式：指针名=&结构体变量 。如：p=&studengt1；

引用形式：（*P）.成员名 或 p->成员名。*p 表示指针 p 指向的结构体变量。

注意：

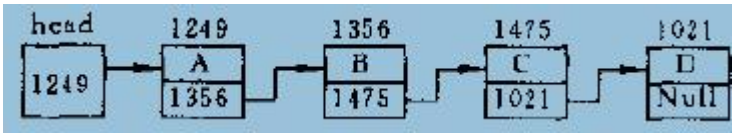
1、定义的结构体类型，和声明结构体变量时不分配空间。

2、结构体可以嵌套定义。

3、结构体成员名可以与变量名相同。

11.7 链表 283

1、单项链表结构：链表有一个头指针和表尾 null 指针。每个结点包含实际数据和下一个结点的地址。



2、可以用结构体类型指针，将多个结构体变量链接起来形成结构体链表。如：

```
#define NULL 0
struct student
{long num;
 float score;
 struct student *next;
};
main()
{ struct student a,b,c, *head, *p;
  a. num=99101; a. score=89. 5;
  b. num=99103; b. score=90;
  c. num=99107; c. score=85;   /*对结点的 num 和 score 成员赋值*/
  head=&a;                     /*将结点 a 的起始地址赋给头指针 head*/
  a. next=&b;                  /*将结点 b 的起始地址赋给 a 结点的 next 成员*/
  b. next=&c;                  /*将结点 c 的起始地址赋给 b 结点的 next 成员*/
  c. next=NULL;                /*c 结点的 next 成员不存放其他结点地址*/
  p=head;                      /*使 p 指针指向 a 结点*/
  do
    {printf("%ld %5. 1f\n",p->num,p->score);
                               /*输出 p 指向的结点的数据*/
     p=p ->next;               /*使 p 指向下一结点*/
    } while(p! =NULL);         /*输出完 c 结点后 p 的值为 NULL*/
}
```

11.8 共用体 297

1、共用体指将几种不同类型的变量存储在同一段内存单元中。

2、共用体变量的存储单元大小等于最长成员变量所占内存的长度

3、共用体变量中起作用的是最后一次存放成员。

4、共用体类型声明方式：union 公用体类型名称 {成员变量列表};

5、共用体变量声明方式：（1）union 公用体类型名称 {成员列表}共用体变量；

（2）先声明共用体类型，然后声明共用体变量，union 公用体类型名称 共用体变量。

11.9 枚举类型 301

1、枚举类型——指将变量的值一一列举出来，变量的值只限于列举出来的值的范围内的一个。

2、声明枚举类型：enum 枚举类型名称 {枚举常量列表};

3、声明枚举变量：enum 枚举类型名称 枚举类型变量;

- 普通文件——指磁盘文件。

2、从操作系统的角度看：每一个与主机相关联的输入输出设备都可看作一个文件。

（1）根据文件的组织形式分为：顺序存取文件 和 随机存取文件。

（2）根据文件的存储形式分为：ASCII 文件 和 二进制文件。

13.3 C 语言对文件的处理方法 319

C 语言中对文件的存取是以字符（字节）为单位的，一个输入输出流就是一个字节流或二进制流。

文件的存储方式分为缓冲文件系统和非缓冲文件系统。区别是缓冲文件系统是系统自动开辟缓冲区，非缓冲文件系统是由程序为每个文件设定缓冲区。

ANSI C 标准只采用 缓冲文件系统 来处理文件。

13.4 文件结构体类型 321

缓冲文件系统中，每个被使用的文件都在内存中开辟一个 FILE 结构体类型的区，用来存放文件的有关信息（文件名字、文件状态、当前位置、缓冲区等有关信息），

FILE 结构体类型原型：

- typedef struct｛

```
short level;     /*缓冲区“满”或“空”的程度*/
unsigned flags;  /*文件状态标志*/
char fd;         /*文件描述符*/
unsigned char hold;  /*如无缓冲区不读取字符*/
short bsize;     /*缓冲区的大小*/
unsigned char *buffer; /*数据缓冲区的位置*/
unsigned char *curp; /*指针，当前的指向*/
unsigned istemp; /*临时文件，指示器*/
short token;/*用于有效性检查*/｝FILE；
```

13.5 文件结构体数组和指针 321

FILE *fp——声明了一个指向 FILE 类型结构体的指针变量。

FILE f［5］——声明了一个文件结构体数组 f，它有 5 个元素，存放 5 个文件的信息。

FILE 变量——声明了一个文件结构体变量。

13.6 文件的操作 321

C 语言要求，在文件读写之前要“打开”文件，在使用结束后要“关闭”文件。

1、打开文件：

- FILE *fp;
- fp＝fopen（“文件名”，“使用文件方式”）；指针变量指向被打开的文件。

例如：fp＝fopen（“a1”，“r”）。

2、关闭文件：

fclose（文件指针）；返回值：关闭成功返回值为 0；否则返回 EOF(-1) 。

使文件指针变量不指向该文件，也就是文件指针变量与文件“脱钩”，此后不能再通过该指针对原来与其相联系的文件进行读写操作。

3、将字符写到文件中：

fputc (ch，fp) ；返回值：如果输出成功，则返回值就是输出的字符；如果输出失败，则返回一个 EOF（-1）。

将字符（ch 的值）输出到文件指针 fp 所指向的文件中去。该文件必须是以写或读写方式打开的。

4、将字符从文件中读出：

ch＝fgetc（fp）；返回值：读取成功一个字符，赋给 c h 。如果遇到文件结束符，返回一个文件结束标志 EOF（-1）。

从指定的文件读入一个字符，该文件必须是以读或读写方式打开的。

注意：ANSI C 提供一个 feof（）函数来判断文件是否真的结束。如果是文件结束，函数 feof（fp）的值为 1 （真）；否则为 0 （假）。以上也适用于文本文件的读取。

5、数据块读写：

- fread (buffer,size,count，fp);
- fwrite(buffer,size,count,fp);

buffer：是一个指针。对 fread 来说，它是读入数据的存放地址。对 fwrite 来说，是要输出数据的地址（均指起始地址）。

size： 要读写的字节数。

count： 要进行读写多少个 size 字节的数据项。

fp： 文件型指针。

6、格式化读写函数：

- fprintf (文件指针，格式字符串，输出表列)；
- fscanf (文件指针，格式字符串，输入表列)；

注意：

用 fprintf 和 fscanf 函数对磁盘文件读写，使用方便，容易理解，但由于在输入时要将 ASCII 码转换为二进制形式，在输出时又要将二进制形式转换成字符，花费时间比较多。因此，在内存与磁盘频繁交换数据的情况下，最好不用 fprintf 和 fscanf 函数，而用 fread 和 fwrite 函数。

7、以“字”或者整数为单位读写函数：

- putw(int i,FILE * fp)；
- int i = getw(FILE * fp)；

8、以“字符串”为单位读写文件的函数：

- fgets(str,n,fp); 从 fp 指向的文件读出 n-1 个字符，在最后加一个’\0’。返回值：str 的首地址。如果遇到 EOF 则读入结束。
- fputs(“china”,fp); 把字符串写入到 fp 指向的文件。第一个参数可以是字符串常量、字符数组名或字符型指针。字符串末尾的′ \ 0 ′不输出。

13.7 文件的定位 333

1、将文件当前的位置指针重新返回到文件的开头位置：无返回值。

- rewind （fp)；执行后，将文件的位置指针重新定位到文件的开头。

2、随机读写：改变文件的位置指针，一般用于二进制文件。

- fseek(文件类型指针，位移量，起始点)；无返回值。

起始点：	文件开头	SEEK_SET	0
	文件当前位置	SEEK_CUR	1
	文件末尾	SEEK_END	2

位移量：以起始点为基点，向后（前，末尾时）移动的字节数。一般要求为 long 型。

3、获取流式文件当前的位置指针：返回当前位置——用相对于文件开头的位移量来表示。，出错时返回 EOF。

- i = ftell(fp);

13.8 出错检测 335

1、ferror（fp）；返回 0，表示未出错；返回非 0，表示出错。

注意：在调用一个输入输出函数后立即检查 ferror 函数的值，否则信息会丢失。在执行 fopen 函数时，ferror 函数的初始值自动置为 0 。

2、clearerr(fp)；使文件错误标志和文件结束标志置为 0。

只要出现错误标志，就一直保留，直到对同一文件调用 clearerr 函数或 rewind 函数，或任何其他一个输入输出函数。调用后 feeror（fp）的值变为 0，且文件结束标志置为 0。

13.9 小结 336

分类	函数名	功能
打开文件	fopen()	打开文件
关闭文件	fclose()	关闭文件
文件定位	fseek()	改变文件位置指针的位置
	Rewind()	使文件位置指针重新至于文件开头
	Ftell()	返回文件位置指针的当前值
文件状态	feof()	若到文件末尾，函数值为真
	Ferror()	若对文件操作出错，函数值为真
	Clearerr()	使 ferror 和 feof()函数值置零
文件读写	fgetc(),getc()从指定文件取得一个字符	
	fputc(),putc()把字符输出到指定文件	
	fgets()从指定文件读取字符串	
	fputs()把字符串输出到指定文件	
	getw()从指定文件读取一个字（int 型）	

putw()把一个字输出到指定文件

fread()从指定文件中读取数据项

fwrite()把数据项写到指定文件中

fscanf()从指定文件按格式输入数据

fprintf()按指定格式将数据写到指定文件中

1、C++保留了 C 语言原有的所有优点，增加了面向对象的机制。

2、C++源文件以.cpp 为后缀。

3、除了可以用 /*……*/ 形式的注释行外，还允许使用以// 开头的单行注释。

4、除了可以用 printf 函数输出信息外，还可以用 cout 进行输出。cout 的作用是将<<运算符右侧的内容送到输出设备中输出。使用 cout 需要用到头文件 iostream.h，在程序的第一行用#include 命令将该头文件“包含”进来。

- cout <<" This is a c++ program.\ n" ；

C++为了方便使用，除了可以利用 printf 和 scanf 函数进行输出和输入外，还增加了标准输入输出流 cout 和 cin。cout 是由 c 和 out 两个单词组成的，代表 C++的输出流，cin 是由 c 和 in 两个单词组成的，代表 C++的输入流。它们是在头文件 iostream.h 中定义的。在键盘和显示器上的输入输出称为标准输入输出，标准流是不需要打开和关闭文件即可直接操作的流式文件。

14.3 C++的输出 cout

1、cout 必须和输出运算符<<一起使用。<< 在这里不作为位运算的左移运算符，而是起插入的作用，例如：cout<<"Hello!\n";的作用是将字符串“Hello!\n” 插入到输出流 cout 中，也就是输出在标准输出设备上。

2、也可以不用\n 控制换行，在头文件 iostream.h 中定义了控制符 endl 代表回车换行操作，作用与\n 相同。endl 的含义是 end of line，表示结束一行。

3、可以在一个输出语句中使用多个运算符<< 将多个输出项插入到输出流 cout 中，<<运算符的结合方向为自左向右，因此各输出项按自左向右顺序插入到输出流中。每输出一项要用一个<< 符号。

例如：

```
for (i=1; i<=3;i++)
  cout<<"count="<<i<<endl;
```

输出结果为:

```
    count=1
count=2
count= 3
```

4、用 cout 和<< 可以输出任何类型的数据。例如：

```
float a=3.45;
int b=5;
char c='A';
cout<<"a="<<a<<", "<<"b="<<b<<", "<<"c="<<c<<endl;
```

5、如果要指定输出所占的列数，可以用控制符 setw 设置(注意：若使用 setw,必须包含头文件 iomanip.h)，如 setw(5) 的作用是为其后面一个输出项预留 5 列，如输出项的长度不足 5 列则数据向右对齐，若超过 5 列则按实际长度输出。例如：

```
cout<<"a="<<setw(6)<<a<<endl
    <<"b="<<setw(6)<<b<<endl
    <<"c="<<setw(6)<<c<<endl;
```

输出结果为：

```
a=3.45
b=     5
c=     A
```

6、在 C++中将数据送到输出流称为“插入”(inserting) ，或“放到”(putting) 。<< 常称为 “插入运算符”。

14.4 C++的输入 cin 341

1、输入流是指从输入设备向内存流动的数据流。标准输入流 cin 是从键盘向内存流动的数据流。用>> 运算符从输入设备键盘取得数据送到输入流 cin 中，然后送到内

存。在 C++中，这种输入操作称为“提取”(extracting) 或“得到”(getting) 。>> 常称为“提取运算符”。

2、cin 要与 >> 配合使用。例如：

3、C++为流输入输出提供了格式控制，如：dec(用十进制形式)，hex(用十六进制形式)，oct(用八进制形式)， 还可以控制实数的输出精度等。

14.5 函数的重载 342

1、C++允许在同一作用域中用同一函数名定义多个函数，这些函数的参数个数和参数类型不同，而且函数类型也可不同，这就是函数的重载，即一个函数名多用。

2、系统会根据参数的类型和个数找到与之匹配的函数，并调用不同的函数。

14.6 带缺省参数的函数 344

C++允许实参个数与形参个数不同。办法是在形参表列中对一个或几个形参指定缺省值(或称默认值)。

例如某一函数的首部可用如下形式：

```
void fun(int a， int b， int c=100)
```

在调用此函数时如写成 fun(2，4，6)，则形参 a，b，c 的值分别为 2，4，6（这是与过去一样的）。如果写成 fun(2，4) ，即少写了最后一个参数，由于在函数定义时已指定了 c 的缺省值为 100，因此 a，b，c 的值分别为 2，4，100。请注意：赋予缺省值的参数必须放在形参表列中的最右端。例如：

```
void f1(float a， int b， int c=0， char d=′a′) (正确)
void f2(float a， int c=0， char d=′a′， int b)      (不正确)
```

注意：不要同时使用重载函数和缺省参数的函数，因为当调用函数时少写一个参数，系统无法判定是利用重载函数还是利用缺省参数的函数，会发生错误。

14.7 变量的引用类型 345

1、“引用”(reference)是 C++的一种新的变量类型，是对 C 的一个重要扩充。它的作用是为一个变量起一个别名。

2、假如有一个变量 a，想给它起一个别名 b，可以这样写：

```
int a;
```

int &b=a;

这就声明了 b 是 a 的“引用”，即 a 的别名。经过这样的声明后，使用 a 或 b 的作用相同，都代表同一变量。注意：在上述声明中，&是“引用声明符”， 并不代表地址。不要理解为“把 a 的值赋给 b 的地址”。

3、声明引用并不另开辟内存单元，b 和 a 都代表同一变量单元。在声明一个引用型变量时，必须同时使之初始化，即声明它代表哪一个变量。 在声明一个变量的引用后，

4、在本函数执行期间，该引用一直与其代表的变量相联系，不能再作为其他变量的别名。下面的用法不对：

```
int  a1，a2;
int  &b=a1;
int &b=a2;(企图使 b 变成 a2 的引用（别名）是不行的)
```

5、C++之所以增加“引用”，主要是把它作为函数参数，以扩充函数传递数据的功能。 C++提供了向函数传递数据的第三种方法，即传送变量的别名。例如：

```
#include <iostream.h>
void swap(int &a，  int &b)
{int temp;
 temp=a;
 a=b;
 b=temp;
}
void main( )
{int i=3，j=5;
swap(i，  j);
cout<<"i="<<i<<"     "<<"j="<<j<<endl;
}
```

在 swap 函数的形参表列中声明变量 a 和 b 是整型的引用变量（和其他变量一样，既可以在函数体中声明变量的类型，也可以在定义函数时在形参表列中声明变量的类型）。

请注意：在此处&a 不是“a 的地址”，而是指“a 是一个引用型变量”。 但是此时并未对它们初始化，即未指定它们是哪个变量的别名。当 main 函数调用 swap 函数时由

实参把变量名传给形参。i 的名字传给引用变量 a，这样 a 就成了 i 的别名。同理，b 成为 j 的别名。a 和 i 代表同一个变量，b 和 j 代表同一个变量。在 swap 函数中使 a 和 b 的值对换，显然，i 和 j 的值同时改变了。

当读者看到&a 这样的形式时，怎样区别是声明引用变量还是取地址的操作呢？请记住，当&a 的前面有类型符时（如 int &a），它必然是对引用的声明；如果前面无类型符（如&a），则是取变量的地址。

14.8 内置函数 348

调用函数时需要一定的时间，如果有的函数需要频繁使用，则所用时间会很长，从而降低程序的执行效率。C++提供一种提高效率的方法，即在编译时将所调用函数的代码嵌入到主调函数中。这种嵌入到主调函数中的函数称为内置函数(inline function)，又称内嵌函数。

例如：

```
#include <iostream.h>
inline int max(int a，  int b，  int c)  //这是一个内置函数，求 3 个整数中的最大者
{ if (b>a) a=b;
  if (c>a) a=c;
  return a;
}
void main( )
{int i=7，  j=10，  k=25，  m ;
  m=max(i，  j，  k);
  cout<<"max="<<m<<endl;
  }
```

14.9 作用域运算符 349

1、作用域运算符：：，：：aa 表示全局作用域中的变量。

2、不能用：：访问函数中的局部变量。

14.10 动态分配运算符 new349

1、new 运算符使用的一般格式为：new 类型 ［初值］;

2、用 new 分配数组空间时不能指定初值。

3、如果由于内存不足等原因而无法正常分配空间，则 new 会返回一个空指针 NULL，用户可以根据该指针的值判断分配空间是否成功。

例子：

new int; （开辟一个存放整数的空间，返回一个指向整型数据的指针）

new int(100); （开辟一个存放整数的空间，并指定该整数的初值为 100）

new char［10］; （开辟一个存放字符数组的空间，该数组有 10 个元素，返回一个指向字符数据的指针）

new int［5］［4］; （开辟一个存放二维整型数组的空间，该数组大小为 5*4）

float *p=new float(3.14159) （开辟一个存放实数的空间，并指定该实数的初值为 3.14159，将返回的指向实型数据的指针赋给指针变量 p）

14.11 撤销内存运算符 delete250

1、delete 运算符使用的一般格式为：delete ［ ］ 指针变量;

例如：

要撤销上面用 new 开辟的存放实数的空间(上面第 5 个例子)，应该用 delete p;

前面用 new char［10］ 开辟的空间，如果把返回的指针赋给了指针变量 pt，则应该用以下形式的 delete 运算符撤销：delete ［ ］ pt;（在指针变量前面加一对方括号，表示对数组空间的操作）。