谭浩强 C 语言程序设计(第三版)

谭浩强 C 语言程序设计

1. C 语言概述

1.1 C 语言的发展过程

C 语言的原型 ALGOL 60 语言。（也称为 A 语言）

1963 年，剑桥大学将 ALGOL 60 语言发展成为 CPL(Combined Programming Language)语言。

1967 年，剑桥大学的 Matin Richards 对 CPL 语言进行了简化，于是产生了 BCPL 语言。

1970 年，美国贝尔实验室的 Ken Thompson 将 BCPL 进行了修改，并为它起了一个有趣的名字"B 语言"。意思是将 CPL 语言煮干，提炼出它的精华。并且他用 B 语言写了第一个 UNIX 操作系统。

而在 1973 年，B 语言也给人"煮"了一下，美国贝尔实验室的 D.M.RITCHIE 在 B 语言的基础上最终设计出了一种新的语言，他取了 BCPL 的第二个字母作为这种语言的名字，这就是 C 语言。

为了使 UNIX 操作系统推广，1977 年 Dennis M.Ritchie 发表了不依赖于具体机器系统的 C 语言编译文本《可移植的 C 语言编译程序》。即是著名的 ANSI C。

1978 年由美国电话电报公司(AT&T)贝尔实验室正式发表了 C 语言。同时由 B.W.Kernighan 和 D.M.Ritchit 合著了著名的"THE C PROGRAMMING LANGUAGE"一书。通常简称为《K&R》，也有人称之为《K&R》标准。但是，在《K&R》中并没有定义一个完整的标准 C 语言，后来由美国国家标准协会（American National Standards Institute）在此基础上制定了一个 C 语言标准，于一九八三年发表。通常称之为 ANSI C。

1987 年，随着微型计算机的日益普及,出现了许多 C 语言版本。由于没有统一的标准,使得这些 C 语言之间出现了一些不一致的地方。为了改变这种情况,美国国家标准研究所(ANSI)为 C 语言制定了一套 ANSI 标准，成为现行的 C 语言标准 3.C 语言的主要特点 。C 语言发展迅速，而且成为最受欢迎的语言之一，主要因为它具有强大的功能。许多著名的系统软件，如 DBASE III PLUS、DBASE Ⅳ都是由 C 语言编写的。用 C 语言加上一些汇编语言子程序，就更能显示 C 语言的优势了,象 PC- DOS 、WORDSTAR 等就是用这种方法编写的。

1990 年，国际化标准组织 ISO（Intrernational StandardOrganization）接受了 87 ANSI C 为 ISO C 的标准（ISO9899-1990）。1994 年，ISO 修订了 C 语言的标准。目前流行的 C 语言编译系统大多是以 ANSI C 为基础进行开发的，但不同版本的 C 编译系统所实现的语言功能和语法规则有略有差别。

1.2 当代最优秀的程序设计语言

1.3 C 语言版本

Microsoft C,或称 MSC

Borland Turbo C,或称 Turbo C

AT&TC

1.4 C 语言的特点

1.简洁,紧凑,方便,灵活.

ANSI 有 32 个关键字,9 种控制语句.

auto, break, case, char, const, continue, default
do, double, else, enume, extern, flost, for
goto, if, int, long, register,return, short
signed,static, sizof, struct,switch, typedef, union
unsigned, void, volatile, while

Turbo C 扩充了 11 个关键字:

asm, _cs, _ds, _es, _SS, cdecl, far, hugeinterrupt near, pascal

2.运算符丰富.

34 种运算符,包括算术,关系,逻辑,位等等.

3.数据结构类型丰富.

4.具有结构化的控制语句

5.语法限制不太严格,程序设计自由度大.

6.C 语言允许直接访问物理地址,能进行位(bit)操作,能实现汇编语言的大部分功能,可能直接对硬件直接进行操作,程序执行效率高.

7.生成目标代码质量高,程序执行效率高.

8.与汇编语言相比,用 C 语言写的程序可移植性好.

1.5 面向对象的程序设计语言

1.6 C 和 C++

1.7 简单的 C 程序介绍

1.8 输入和输出函数

scanf(format_string,vlaue_addr_list)

printf(format_string,value_list)

1.9 C 源程序的结构特点

每个源程序由一个或多个源文件组成.

file_1

...

file_n

每个源文件由一个或多个函数组成.

function_1

...

function_n

第个函数由函数原型与函数体组成.

```
return_value function_name(parameter_list)
{
     function_body;
```

}

每个函数体由声明部分,执行部分,返回值部分(可选)组成:

```
{
   declaration;
   execution;
   return_value;
}
```

注意:

每个源程序必须有且只有一个 main 函数.

每个语句必须以分号结尾,但预处理命令,函数头和花括号之后不能加分号.

每个标识符,关键字必须用空格间隔.

1.10 书写程序时应遵循的规则

采用 C 语言格式

1.11 C 语言的字符集

C 语言字符集由字母,数字,下划线,空白符,标点和特殊字符组成.

1.字母:

小写字母 a~z 共 26 个

大写字母 A~Z 共 26 个

2 数字:

0~9 共 10 个

3.空白符:

空格符,制表符,换行符等统称为空白符.空白符只在字符常量和字符串常量中起作用.

4.标点和特殊字符

1.12 C 语言的词汇

在 C 语言中使用的词汇分为六类:标识符,关键字,运算符,分隔符,常量,注释符等.

1.标识符

C 规定,标识符只能是字母,数字,下划线组成的字符,并且第一个字符必须是字母或下划线,或者是关键字.ANSI C 不限制标识符的长度,但受各种版本 C 编译系统限制,同时受具体机器的限制.

2.关键字

C 语言的关键字分为:

(1)类型说明符

(2)语句定义符

(3)预处理命令字

3.运算符

4.分隔符

5.常量

6.注释符
C 语言注释必须是/**/

2. 程序的灵魂--算法
2.1 算法的概念
2.2 简单算法举例
2.3 算法的特性
2.4 怎样表示一个算法
2.4.1 用自然语言表示算法
2.4.2 用流程图表示算法
2.4.3 三种基本结构与改进的流程图
2.4.4 用 N-S 流程图表示算法
2.4.5 用伪代码表示算法
2.4.6 用计算机语言表示算法
2.5 结构化程序的设计方法

3. 数据类型,运算符与表达式
3.1 C 语言的数据类型
3.2 常量与变量
3.2.1 常量和符号常量
3.2.2 变量
3.3 整型数据
3.3.1 整形常量的表示方法
3.3.2 整形变量
3.4 实型数据
3.4.1 实型变量的表示方法
3.4.2 实型变量
3.4.3 实型常数的类型
3.5 字符型数据
3.5.1 字符常量
3.5.2 转义字符
3.5.3 字符变量
3.5.4 字符数据在内存中的存储形式及使用方法
3.5.5 字符串常量
3.5.6 符号常量
3.6 变量赋初值
3.7 各类数值型数据之间的混合运算
3.8 算法运算符和算术表达式
3.8.1 C 运算简介
3.8.2 算术运算符和算术表达式
3.9 赋值运算符和赋值表达式
3.10 逗号运算符和逗号表达式
3.11 小结

==========
第 2 章 程序的灵魂--算法

Nikilaus Wirth 提出的公式: 程序=算法+数据结构.
教材认为: 程序=算法+数据结构+程序设计方法+语言工具和环境

1. 算法的概念
 算法指计算机解决问题的步骤.
 算法分为数值算法与非数值算法.
2. 算法的简单举例
3. 算法的特性
 3.1 有穷性
 3.2 有效性
 3.3 确定性
 3.4 零个或多个输入
 3.5 一个或多个输出
4. 怎样表示一个算法
 4.1 使用流程图表示算法
  4.1.1 传统流程图
  4.1.2 N-S 流程图
  4.1.3 三种基本结构
   I. 顺序结构
   II.选择结构
   III.循环结构
 4.2 使用伪代码表示算法
 4.3 使用计算机语言表示算法
5. 结构程序设计的方法
 5.1 自顶向下,逐步细化,模块化设计,结构化编程
 5.2 自底向上,逐步增加,模块化设计,结构化编程

==========
第 3 章 数据类型,运算符和表达式

1. C 语言的数据类型:
 1.1 C 语言的变量: 先定义,后使用
 1.2 变量的定义包括三方面:
  存储类别
  数据类型
  作用域与生命期
 1.3 所谓数据类型实质是内存的存储表示.
 1.4 C 语言的数据类型可分为四类:
  基本类型: 整形,实型,字符型,枚举型
  构造类型: 数组类型,结构体类型,共用体类型
  指针类型
  空类型

2. 常量与变量

常量指值不会改变的量.

变量指值可以改变的量.

3. 整型数据

3.1 整型常量的表示方法

十进制常数,无须前缀

八进制常数,必须以 0 为前缀

十六进制常数,必须以 0x 为前缀

长整形常数,必须以 L 或 l 为后缀

无符号常数,必须以 U 或 u 为后缀

3.2 整型变量

整型数据在内存中的形式: 补码,正数的补码是本身,负数的补码按位取反再加 1.

整型变量的分类: 短整型,整型,长整型,无符号型.

整型变量的定义: 存储类型 数据类型 变量标识符 1,...;

4. 实型数据

4.1 实型变量的表示方法

小数形式

指数形式: 必须加阶码标志 e 或 E.

单精度常数: 必须以 F 或 f 为后缀.

4.2 实型变量

实型数据在内存中的形式: 符号|小数部分|指数部分

实型变量的分类: 单精度,双精度,长双精度

实型数据的舍入误差: 实型数据会自动截舍超长部分.

4.3 实型常数的类型

实型常数都按 double 型处理.

5. 字符型数据

5.1 字符常量: 字符常量指用单引号括起来的一个字符.

5.2 转义字符: 转义字符以反斜线"\"开头,表示特殊的字符常量.

常用转义字符其其含义:

\n,换行,10

\r,回车,13

\t,水平制表符,9

\v,垂直制表符,

\b,退格符,8

\f,换页符,12

\\,反斜线,92

\',单引号,39

\",双引号,34

\a,鸣铃,7

\ddd,3 位八进制数表示的 ASCII 字符

\xhh,2 位十六进制数表示的 ASCII 字符

5.3 字符变量：存储字符的变量
5.4 字符数据在内存中的存储方式及使用方法
每个字符变量被分配一个字节的内在空间.
C 语言允许字符变量参与数值运算,即用字符的 ASCII 码参与运算.
5.5 字符串常量：字符串常量是用双引号括起来的字符序列,字符串常量占内存的字节数等于字符串中的字节数加 1.

6. 变量赋初值：存储类别 数据类型 变量 1=值 1,...;
7. 各类数值型数据之间的混合运算：
7.1 自动类型转换：char, short-->int-->long-->double<--float
7.2 强制类型转换：(类型说明符)(表达式)
8. 运算符和表达式：运算符有二个特性,优先级和结合性
C 语言的运算符可以分为 10 类：
8.1 算术运算符：+,-,*,/,%,++,--
8.2 关系运算符：>,<,>=,<=,==,!=
8.3 逻辑运算符：!,&&,||
8.4 位运算符：~,&,|,^,算术左移<<,算术右移>>
8.5 赋值运算符：=,算术复合赋值符,位复合赋值符
8.6 条件去处符：?:
8.7 逗号运算符：,,...
8.8 指针运算符：取内容*,取地址&
8.9 求字节数运算符：sizeof
8.10 特殊运算符：类型运算符(),下标运算符[],成员运算符(-->,.)

==========

第 4 章 最简单的 C 程序设计--顺序程序设计
1. 从流程角度看，程序可以分为三种基本结构：顺序结构，选择结构，循环结构.
2. C 源程序的结构
C 源程序
|__C 源程序文件 1,C 源程序文件 2,...
|__预处理命令,
全局变量声明,
函数 1,...
函数 n
|__函数首部,
函数体
|__局部变量声明,
执行部分
|__语句 1,...
语句 n
3. C 语言中的语句可以分为五类：
3.1 表达式语句
3.2 函数调用语句

3.3 控制语句: 顺序,选择,循环,跳转
3.4 复合语句
3.5 空语句
4. 数据输入输出的概念及在C语言中的实现
C语言中所有的输入输出都通过标准库函数来实现.
标准库文件stdio.h包含所有的标准输入输出函数.
4.1 字符输入输出函数:
getchar()
putchar(字符变量)
4.2 格式输出函数printf:
4.2.1 printf函数的一般形式: printf("格式控制串",输出表列)
4.2.2 格式控制串的一般形式: %[标志][宽度][.精度]类型
1.类型:
d表示十进制
o表示八进制
x,X表示十六进制
u,U表示无符号整型
f表示小数形式
e,E表示指针形式
g,G表示%f,%e中宽度较小的.
c表示字符
s表示字符串
2.标志:
-表示左对齐
+表示输出符号
空格表示输出正数时冠以空格,输出负数时冠以负号
#对c,s,d,u类无影响,对o,x类输出前缀,对e,g,f类当结果有小数时才显示小数点
3.宽度:
超宽时忽略
4.长度:
h表示短整型量,l表示长整型量
4.3 格式输入函数scanf:
4.3.1 scanf函数的一般形式: scanf("格式控制串",地址表列)
4.3.2 格式控制串的一般形式: %[*][宽度][长度]类型
1.类型:
d表示十进制
o表示八进制
x,X表示十六进制
u,U表示无符号型
f表示小数形式
e表示指数形式
c表示字符形式
s表示字符串形式

2.*表示跳过输入值
3.宽度
4.长度：h 表示短整型量，l 表示长整型量

==========

第 5 章 分支选择程序

1. 关系运算符和关系表达式

比较二个量的运算符称为关系运算符.

1.1 关系运算符的优先级和结合性：关系运算符为双目运算符,其优先级低于算术运算符,结合性为左结合.

1.2 关系表达式的一般形式：表达式 关系运算符 表达式

2. 逻辑运算符和逻辑表达式

C 语言提供三种逻辑运算符：逻辑非!,逻辑与&&,逻辑或||

2.1 逻辑运算符的优先级和结合性:

逻辑运算符的优先级为：逻辑非!>算术运算符>关系运算符>逻辑与&&,逻辑或||>赋值运算符

逻辑运算符的结合性为：逻辑非为单目运算符,结合性为右结合. 逻辑与,逻辑或为双目运算符,结合性为左结合.

2.2 逻辑运算的值:

逻辑运算遵循短路逻辑,其值为"真"或"假",分别用"1"和"0"表示.即非 0 为真,0 为假,真值为 1,假值为 0;

2.3 逻辑表达式的一般形式：表达式 逻辑运算符 表达式

3. if 语句

3.1 C 语言的 if 语句有 3 种形式:

第一种形式 if:

```
if(表达式) 语句;
```

第二种形式 if-else:

```
if(表达式)
  语句 1;
else
  语句 2;
```

第三种形式 if-else-if:

```
if(表达式 1)
  语句 1;
else if(表达式 2)
  语句 2;
...
else if(表达式 n)
  语句 n;
else
  语句 n+1;
```

3.2 if 语句的嵌套

C 语言规定, else 语句总是与前面最近的 if 配对.

3.3 条件运算符和条件表达式:

一般形式: 表达式 1?表达式 2:表达式 3

条件表达式 2,条件表达 3 的返回值必须具有相同的数据类型.

4. switch 语句:

C 语言提供多分支选择的 switch 语句, 一般形式为:

```
switch(整型表达式){
 case 整型常量表达式 1: 语句 1;
 case 整型常量表达式 2: 语句 2;
 ...
 case 整型常量表达式 n: 语句 n;
 default: 语句 n+1;
}
```

switch-case 从匹配的 case 往下执行,default 是默认执行语句.使用 break 语句可以跳出 switch-case;

示例: int main(int argc, char* argv[])

```
 {
     int input=0;
     scanf("%d",&input);
     switch(input){
  case 1:
      printf("start from 1\n");
  case 2:
      printf("start from 2\n");
  case 3:
      printf("start from 3\n");
  default:
      printf("start from default");
     }
 }
 ----------------------------------------
 1
 start from 1
 start from 2
 start from 3
 start from default
```

==========

第 6 章 循环结构

1. 概述

C 语言提供多种循环语句,可以组成不同形式的循环结构.

(1) 用 if-goto 语句.
(2) 用 for 语句.
(3) 用 while 语句.
(4) 用 do-while 语句.

2. if-goto 语句

goto 是一种无条件转移语句.

一般形式: goto label;

label 由标识后带:组成.

3. for 语句

一般形式: for(initialization;condition;increasement)
statement

4. while 语句

一般形式: while(condition) statement

5. do-while 语句

一般形式: do
statement
while(condition)

6. 循环的嵌套

C 语言允许循环的嵌套,而且对嵌套层次不作限制.

7. 几种循环的比较:

不同情景使用不同循环结构.

8. break 语句和 continue 语句

8.1 break 语句用于跳出本次循环或跳出开关语句.

一般形式: break;

8.2 continue 语句用于跳出本层循环.

一般形式: continue;

==========

第 7 章 数组

C 语言中使用数组处理向量.

1. 一维数组的定义和引用

1.1 一维数组的定义方式: ElementType ArrayName[ArrayBound];

1.2 一维数组元素的引用: ArrayName[index];

1.3 一维数组的初始化: 使用数组直接量初始化,或逐个元素初始化. 数组初始化可以不指定第一维数组的边界.

2. 二维数组的定义和引用

2.1 二维数组的定义方式: ElementType ArrayName[ArrayBound1][ArrayBound2];

在 C 语言中,二维数组是按行排列的.

2.2 二维数组元素的引用: ArrayName[ArrayBound1][ArrayBound2];

2.3 二维数组的初始化: 二维数组可以分段赋值,也可以连续赋值.初始化可以

不指定首维边界,因为 C 语言中张灯数组元素按列排列,首维边界自动计算,但是,一旦指定,将以此作为边界.

3. 字符数组
 3.1 字符数组的定义: char ArrayName[ArrayBound1][ArrayBound2][...];
 3.2 字符数组的初始化:
  字符数组可以使用字符串初始化,但最后一个元素是字符串的结束标志'\0';
 3.3 字符数组的引用: ArrayName[index1][index2][...];

4. 字符串和字符串结束标志
 C 语言使用字符数组存储字符串,并以'\0'作为结束标志.
 C 语言允许使用字符串对数组作初始化赋值.

5 字符串的处理函数:
 C 语言标准库提供字符串的处理函数可以分为输入,输出,连接,复制,比较,求长,修改,转换,搜索等.
 5.1 输入:
  scanf(format_string,array)
  gets(array)
 5.2 输出:
  printf(format_string,array)
  puts(array)
 5.3 连接:
  strcat(array1,array2);
  将 array2 连接到 array1,同时删除 array1 的结束标志.
 5.4 复制:
  strcpy(array1,array2);
  将 array2 复制到 array1,同时复制结束标志.
 5.5 比较:
  strcmp(array1,array2);
  比较 array1 和 array2,并返回负数,0,正数.
 5.6 求长:
  strlen(array);
  返回 array 的长度,但不包括结束标志.

==========

递归简述

递归算法的特点

递归过程一般通过函数或子过程来实现。

递归算法：在函数或子过程的内部，直接或者间接地调用自己的算法。

递归算法的实质：是把问题转化为规模缩小了的同类问题的子问题。然后递归调用函数(或过程)来表示问题的解。

递归算法解决问题的特点：

(1) 递归就是在过程或函数里调用自身。

(2) 在使用递增归策略时，必须有一个明确的递归结束条件，称为递归出口。

(3) 递归算法解题通常显得很简洁，但递归算法解题的运行效率较低。所以一般不提倡用递归算法设计程序。

(4) 在递归调用的过程当中系统为每一层的返回点、局部量等开辟了栈来存储。递归次数过多容易造成栈溢出等。所以一般不提倡用递归算法设计程序。

递归算法所体现的“重复”一般有三个要求：

一是每次调用在规模上都有所缩小(通常是减半)；

二是相邻两次重复之间有紧密的联系，前一次要为后一次做准备(通常前一次的输出就作为后一次的输入)；

三是在问题的规模极小时必须用直接给出解答而不再进行递归调用，因而每次递归调用都是有条件的(以规模未达到直接解答的大小为条件)，无条件递归调用将会成为死循环而不能正常结束。

例子如下：

描述：把一个整数按 n(2<=n<=20)进制表示出来，并保存在给定字符串中。比如 121 用二进制表示得到结果为：“1111001”。

参数说明：s: 保存转换后得到的结果。

n: 待转换的整数。

b: n 进制 (2<=n<=20)

```
void
numbconv(char *s, int n, int b)
{
int len;
if(n == 0) {
strcpy(s, "");
return;
}
/* figure out first n-1 digits */
numbconv(s, n/b, b);
/* add last digit */
len = strlen(s);
s[len] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"[n%b];
s[len+1] = '\0';
}
void
main(void)
{
char s[20];
int i, base;
FILE *fin, *fout;
fin = fopen("palsquare.in", "r");
fout = fopen("palsquare.out", "w");
assert(fin != NULL && fout != NULL);
fscanf(fin, "%d", &base);
/*PLS set START and END*/
```

```
for(i=START; i <= END; i++) {
numbconv(s, i*i, base);
fprintf(fout, "%s\n", s);
}
exit(0);
}
```

递归算法简析（PASCAL 语言）

递归是计算机科学的一个重要概念,递归的方法是程序设计中有效的方法,采用递归编写

程序能是程序变得简洁和清晰.

一 递归的概念

1.概念

一个过程(或函数)直接或间接调用自己本身,这种过程(或函数)叫递归过程(或函数).

如:

```
procedure a;
begin
.
.
.
a;
.
.
.
end;
```

这种方式是直接调用.

又如:

```
procedure b; procedure c;
begin begin
. .
. .
. .
c; b;
. .
. .
. .
end; end;
```

这种方式是间接调用.

例 1 计算 n!可用递归公式如下:

1 当 n=0 时

fac(n)={n*fac(n-1) 当 n>0 时

可编写程序如下:

```
program fac2;
var
```

```
n:integer;
function fac(n:integer):real;
begin
if n=0 then fac:=1 else fac:=n*fac(n-1)
end;
begin
write('n=');readln(n);
writeln('fac(',n,')=',fac(n):6:0);
end.
```

例 2 楼梯有 n 阶台阶,上楼可以一步上 1 阶,也可以一步上 2 阶,编一程序计算共有多少种不同的走法.

设 n 阶台阶的走法数为 f(n)

显然有

1 n=1

f(n)={

f(n-1)+f(n-2) n>2

可编程序如下:

```
program louti;
var n:integer;
function f(x:integer):integer;
begin
if x=1 then f:=1 else
if x=2 then f:=2 else f:=f(x-1)+f(x-2);
end;
begin
write('n=');read(n);
writeln('f(',n,')=',f(n))
end.
```

二, 如何设计递归算法

1.确定递归公式

2.确定边界(终了)条件

三, 典型例题

例 3 梵塔问题

如图:已知有三根针分别用 1,2,3 表示,在一号针中从小放 n 个盘子,现要求把所有的盘子

从 1 针全部移到 3 针,移动规则是:使用 2 针作为过度针,每次只移动一块盘子,且每根针上

不能出现大盘压小盘.找出移动次数最小的方案.

程序如下:

```
program fanta;
var
n:integer;
procedure move(n,a,b,c:integer);
begin
```

```
if n=1 then writeln(a,'--->',c)
else begin
move(n-1,a,c,b);
writeln(a,'--->',c);
move(n-1,b,a,c);
end;
end;
begin
write('Enter n=');
read(n);
move(n,1,2,3);
end.
```

例 4 快速排序

快速排序的思想是:先从数据序列中选一个元素,并将序列中所有比该元素小的元素都放到它的右边或左边,再对左右两边分别用同样的方法处之直到每一个待处理的序列的长度为 1, 处理结束.

程序如下:

```
program kspv;
const n=7;
type
arr=array[1..n] of integer;
var
a:arr;
i:integer;
procedure quicksort(var b:arr; s,t:integer);
var i,j,x,t1:integer;
begin
i:=s;j:=t;x:=b;
repeat
while (b[j]>=x) and (j>i) do j:=j-1;
if j>i then begin t1:=b; b:=b[j];b[j]:=t1;end;
while (b<=x) and (i<j) do i:=i+1;
if i<j then begin t1:=b[j];b[j]:=b;b:=t1; end
until i=j;
b:=x;
i:=i+1;j:=j-1;
if s<j then quicksort(b,s,j);
if i<t then quicksort(b,i,t);
end;
begin
write('input data:');
for i:=1 to n do read(a);
writeln;
quicksort(a,1,n);
```

```
    write('output data:');
    for i:=1 to n do write(a:6);
    writeln;
    end.
```

==========

第 8 章 函数

1. 概述

1.1 C 源程序的结构

C 源程序
|__源程序文件 1..n
    |__预处理命令,
        全局变量声明,
        函数 1..n
        |__函数首部,
            函数体
    |__局部变量声明
        执行部分
        |__语句 1..n

1.2 语言中函数的分类

1)从函数定义的角度: 库函数与用户函数;
2)从函数功能的角度: 有返回值函数与无返回值函数;
3)从数据传递的角度: 有参数函数与无参数函数;

1.3 C 语言库函数分类:

1)字符类型分类函数;
2)转换函数;
3)目录路径函数;
4)诊断函数;
5)图形函数;
6)输入输出函数;
7)接口函数;
8)字符串函数;
10)数学函数;
11)日期和时间函数;
12)进程控制函数;
13)其它函数;

1.4 C 语言的函数定义是平行的,即函数不允许嵌套定义,但允许嵌套调用,main 函数除外.

2. 函数定义的一般形式

2.1 无参数的函数定义:

```
type function(parameter_list)
{
 declaration;
 execution;
}
```

2.2 有参数的函数定义:

```
type function(parameter_list)
{
 declaration;
 execution;
}
```

3. 函数的参数和函数的值

3.1 形式参数和实际参数

formal parameter list 和 actual parameter list 必须一一对应,包括数量,次序,类型.

3.2 函数的返回值

1)函数的值只能通过 return 语句返回主调函数.

return expression; 或 return (expression);

2)函数值的类型必须与函数的类型一致,否则,函数值自动类型转换为函数类型再返回.

3)函数类型为 int 时可以省略.

4)函数没有类型必须明确定义为 void 类型.

4. 函数的调用

4.1 函数调用的一般形式: function(actual_parameter_list);

4.2 函数调用方式:

1)函数表达式;

2)函数语句;

3)函数实参;

4.3 被调用函数的声明和函数原型

C 语言中的函数必须先声明,后调用. 正如变量必须先定义,后引用.

声明函数的方式:

```
type function(formal_type1 formal_value1,...)
   或 type function(formal_type1,...)
   即函数原型不包括形参的名字.
```

5. 函数的嵌套调用

前面说过,函数不允许嵌套定义,但允许嵌套调用,但 main 函数除外.

6. 函数的递归调用

函数直接或间接调用自身称为递归.C 语言允许函数递归调用.

递归的关键: 1)确定递归公式 2)确定递归边界

递归和递推的设计思路相似,关键都是确定公式与确定边界.
递归是剥,递推是裹.

7. 数组作为函数参数
数组作为函数参数有 2 种形式:
1)数组元素作为函数实参.
2)数组作为函数形参和实参.
数组形参忽略首维边界,即首维边界指定也没用.
数组形参与数组实参必须一一对应,包括元素类型,维数,维界. 但首维边界除外.

8. 局部变量和全局变量
C 语言中变量的作用域: 全局变量和局部变量.
8.1 局部变量在函数体内声明的,作用域自声明点起至被包含的"}"处.生存期由存储类别决定.
复合语句内可以定义局部变量,内层局部变量会屏蔽同名的外层局部变量或全局变量.

8.2 全局变量在文件内声明的,作用域自声明点起至文件尾处.生存期为程序执行期间.因为全局变量都保存在静态存储区.

9. 变量的存储类别
9.1 动态存储方式和静态存储方式
从变量的作用域可分为全局变量和局部变量;
从变量的生成期可分为静态存储方式和动态存储方式;

C 语言存储空间:

程序区

静态存储区

动态存储区

程序区保存程序数据.
静态存储区在程序执行前分配,在程序执行后释放,在程序执行期间不再动态分配和释放.全局变量保存在静态存储区.
动态存储区在程序执行期间动态分配和释放.函数形参,自动变量,函数调用现场和地址信息保存在动态存储区.

简言之,变量有二个因素:作用域和生存期;
变量的作用域由声明位置决定: 如果是全局变量,则从声明点到文件尾. 如果是局部变量,则从声明点到被包含的}处.
变量的生存期由存储类别决定: 如果是静态存储,则在程序执行期间有效.如果是动态存储,则在函数调用期间有效.

程序可以使用 extern 声明外部变量,即其它文件的全局变量.
但是,全局变量可被 static 限制在声明的文件内,即其他文件无法 extern 此全局变量.

程序可以使用 static,auto,regitser 声明局变量的存储类别,默认的存储类别是 auto;

==========

第 9 章 预处理命令

1. 概述

源程序的预处理指源程序在编译前进行的工作.
源程序的预处理由预处理程序负责完成.
源程序在预算后自动编译.

C 语言提供多种预处理功能: 宏定义, 文件包含, 条件编译.

2. 宏定义:

宏定义由源程序中的宏定义命令完成,宏展开(或宏替换)由预处理程序自动完成.

2.1 无参宏定义

#define macro_name macro_value
或 #define macro_name (expression)

2.2 带参宏定义

#define macro_name(formal_parameter_list) macro_value
或 #define macro_name(formal_parameter_list) (expression)

2.3 注意:

预处理命令后都没有分号,因为其是命令,不是语句.
如果 macro_value 是 expression,最好使用括号括住.
带参宏的形参没有类型,因其只是占位符,并非真正的参数.
宏定义可以嵌套,即后面的宏的定义可以使用前面已定义的宏.

3. 文件包含:

文件包含有二种形式:

#include <file_name>
或 #include "file_name"

二者的区别: <>只在库目录查找.
""先在前前目录查找,再在库目录查找.

4. 条件编译:

C 语言的条件编译有三种形式:

4.1 第一种形式:

#ifdef macro_name
程序段 1

```
#else
 程序段 2
#endif
```

4.2 第二种形式:

```
#ifndef macro_name
 程序段 1
#else
 程序段 2
#endif
```

4.3 第三种形式:

```
#if condition_1
 程序段 1
#elif condition_2
 程序段 2
...
#elif condition_m
 程序段 m
#else
 程序段 m+1
#endif
```

4.4 条件编译可以减少内存开销,提高程序效率.
源程序文件最好使用条件编译,避免多次被包含造成编译错误.

==========

第 10 章 指针

指针是 C 语言的核心.

1.地址指针的概念

1.1 地址指存储单元的编号

1.2 指针指存储空间的首地址

1.3 指针变量指用来存储指针的变量

1.4 地址和指针的区别: 地址是存储单元的编号,指针是数据结构存储空间的首地址.每个存储单元都有一个编号,地址即这个编号,但某个数据结构可能占据若干连续存储单元,指针即这块连续存储单元的首地址.

2.变量的指针与指向变量的指针变量.

变量指针指变量存储空间的首地址.

指针变量指存储指针的变量.

2.1 定义指针变量:

```
type *variable_name;
```

其中*表示 variable_name 是 type 类型的指针.

2.2 指针变量的引用

指针运算符有 2 种:
1) *表示取内容.
2) &表示取地址.

2.3 指针变量作为函数参数
C 语言中的参数传递有 2 种: 值传递和指针传递.
值传递,形参得到实参的拷贝.
指针传递,形参完全指向实参.
程序员可以灵活参数传递的方式.

2.4 指针变量的几个问题
1. 指针运算符
1) &取地址
2) *取内容
2. 指针变量的运算
1) 赋值运算: =
2) 加减算术运算: +,-,++,--
3) 两个指针变量之间的运算: 只有指向同一数组的指针,两个指针变量之间的运算才有意义. 否则, 两个指针变量之间的运算没有任何意义.
注意: 对指针变量赋 0 值和不赋值是不同的,指针赋 0 值后可以安全,只是不指向具体变量.指针未赋值前可能指向任意值,使用不安全.

3.指针的核心理念
3.1 变量指针与指针变量
变量指针是变量的存储空间的首地址.
指针变量是存储指针的变量.
3.2 数组指针与指针数组
数组指针是指向数组的指针.
指针数组是存储指针的数组.
3.3 函数指针与指针函数
函数指针是指向函数的指针.
指针函数是返回指针的函数.
3.4 指针的指针
指针的指针是指向指针变量的指针.
3.5 误区:指针的指针与二维数组名的关系
指针的指针: 指向指针变量的指针.
二维数组名: 指针二维数组的首地址.
可以当作二维数组首元素的指针.指针的指针与二维数组名是没有必然关联.
C 语言中,n 维数组可以展开成(n-1)维数组的数组.

3.6 void 指针
ANSI C 新增 void 指针类型,表示不确定类型的指针,即可以指向任意类型的指针.

注意：void 类型,表示空类型,但 void 指针,表示任意类型.使用 void 指针的时候,必须明确强制类型转换为目标类型,否则无法使用.

4. 指针参数：以下是二个关键程序,演示指针函数.

注意：1)传递数组指针的时候,必须明确指定数组的维界.否则数组指针无法进行指针变量的运算,因为类型不确定,指针无法知道偏移量.

2)把 n 维数组展开成一维数组,可以避免定义数组的维界.

```
#include<stdio.h>

/*
* invalid use of array with unspecified bounds|
* 出现此错误,原因在于形参是数组指针,但是数组的边界未指明,无法进行指针变量的移动.
* 类似,多维数组除首维之后没有指定维界一样.
*/
void test(int (*p)[2], int b1, int b2)
{
    int i,j;
    for(i=0;i<b1;i++)
    {
 for(j=0;j<b2;j++)
 {
     printf("%d  ",p[i][j]);
 }
 printf("\n");
    }
}

/*
* 把 n 维数组展开作为一维数组,可以避免硬性指定数组的维界.
*
*/
void test2(int *p, int b1, int b2)
{
    int i,j;
    for(i=0;i<b1;i++)
    {
 for(j=0;j<b2;j++)
 {
     printf("%d  ",*(p+b1*i+j));
 }
 printf("\n");
    }
}
```

```
int main()
{
    int a[][2]={{1,2},{3,4}};
    test2(a,2,2);
}
```

==========

---

realloc

原型：extern void *realloc(void *mem_address, unsigned int newsize);

用法：#include <stdlib.h> 有些编译器需要#include <alloc.h>

功能：改变 mem_address 所指内存区域的大小为 newsize 长度。

说明：如果重新分配成功则返回指向被分配内存的指针，否则返回空指针 NULL。

当内存不再使用时，应使用 free()函数将内存块释放。

注意：这里原始内存中的数据还是保持不变的。

举例：

```
// realloc.c
#include <syslib.h>
#include <alloc.h>
main()
{
char *p;
clrscr(); // clear screen
p=(char *)malloc(100);
if(p)
printf("Memory Allocated at: %x",p);
else
printf("Not Enough Memory!\n");
getchar();
p=(char *)realloc(p,256);
if(p)
printf("Memory Reallocated at: %x",p);
else
printf("Not Enough Memory!\n");
free(p);
getchar();
return 0;
}
```

详细说明及注意要点：

1、如果有足够空间用于扩大 mem_address 指向的内存块，则分配额外内存，并返回 mem_address

这里说的是“扩大”，我们知道，realloc 是从堆上分配内存的，当扩大一块内存空间时，realloc()试图直接从堆上现存的数据后面的那些字节中获得附加的字节，如果能够满足，自然天下太平。也就是说，如果原先的内存大小后面还有足够的空闲空间用来分配，加上原来的空间大小＝ newsize。那么就 ok。得到的是一块连续的内存。

2、如果原先的内存大小后面没有足够的空闲空间用来分配，那么从堆中另外找一块 newsize 大小的内存。

并把原来大小内存空间中的内容复制到 newsize 中。返回新的 mem_address 指针。（数据被移动了）。

老块被放回堆上。

例如：

```
#include <malloc.h>
void main()
{
char *p, *q;
p = (char * ) malloc (10);
q=p;
p = (char * ) realloc (p,20); //A
.....
}
```

在这段程序中我们增加了指针 q，用它记录了原来的内存地址 p。这段程序可以编译通过，但在执行到 A 行时，如果原有内存后面没有足够空间将原有空间扩展成一个连续的新大小的话，realloc 函数就会以第二种方式分配内存，此时数据发生了移动，那么所记录的原来的内存地址 q 所指向的内存空间实际上已经放回到堆上了!这样就会产生 q 指针的指针悬挂，如果再用 q 指针进行操作就可能发生意想不到的问题。所以在应用 realloc 函数是应当格外注意这种情况。

3、返回情况

返回的是一个 void 类型的指针，调用成功。（这就再你需要的时候进行强制类型转换）

返回 NULL，当需要扩展的大小(第二个参数)为 0 并且第一个参数不为 NULL，此时原内存变成了“freed（游离）”的了。

返回 NULL，当没有足够的空间可供扩展的时候，此时，原内存空间的大小维持不变。

4、特殊情况

如果 mem_address 为 null，则 realloc()和 malloc()类似。分配一个 newsize 的内存块，返回一个指向该内存块的指针。

如果 newsize 大小为 0，那么释放 mem_address 指向的内存，并返回 null。

如果没有足够可用的内存用来完成重新分配(扩大原来的内存块或者分配新的内存块），则返回 null.而原来的内存块保持不变。

==========

第 11 章 结构体与共用体

1. 结构体类型的定义与结构体变量的声明

1.1 先定义,再声明

```
struct StructType
{
 ElemType_1 elem_1;
 ElemType_2 elem_2;
 ...
 ElemType_n elem_n;
};
...
struct StructType variable[={intial_list}];
```

注意,这种声明必须在 StrutType 前加 struct 关键字,指明这是一个 struct 自定义类型.

1.2 定义同时声明

```
struct StructType
{
 ElemType_1 elem_1;
 ElemType_2 elem_2;
 ...
 ElemType_n elem_n;
}variable[={initial_list}];
```

注意,这种声明适用离散型变量(变量个数有限).

1.3 直接声明变量

```
struct
{
 ElemType_1 elem_1;
 ElemType_2 elem_2;
 ...
 ElemType_n elem_n;
}variable[={initial_list}];
```

注意,这种声明适用 singleton 情景.

2. 结构体变量元素的引用

```
variable.element;
  或   (*pointer).element;
pointer->element;
```

即变量使用.运算符引用元素,指针使用->运算符引用元素.

3. 结构体变量的初始化与赋值.

结构体变量的初始化可以使用结构体直接量.

但是结构体变量的赋值只能单独对成员赋值.

4. 结构体数组类型的定义和结构体数组变量的声明

结构体数组类型的定义和结构数组变量的声明类似,只是将变量类型改成数组类型.

5. 结构体指针变量的声明

5.1 结构体指针变量的声明

```
struct StructType *pointer;
```

5.2 结构体指针变量的引用

```
(*pointer).element;
```

或 `pointer->element;`

6. 结构体数组指针的声明

结构体数组指针指向结构体数组的首地址.

结构体数组的首地址同时也是第一个数组元素的首地址.

7. 结构体指针参数

结构体指针参数与普通指针参数无异.

C 语言中参数传递的方式有二种:值传递与指针传递.

1)值传递,形参拷贝实参的值内容.

指针传递,形参指跟实参指向相同的首地址.

2)值传递需要值拷贝,损失性能,但形参的值修改不影响实参的值,保证实参安全.

指针传递不需要值拷贝,节省性能,但形参的值修改影响实参的值,不保证实参数安全.

8. 动态存储分配

1.内存分配函数

`void *malloc(unsigned size)`分配 1 块 size 大小的连续内存空间.成功返回内存空间首地址,失败返回 NULL.

`void *calloc(size_t n, size_t size)`分配 n 块 size 大小的连续内存空间.成功返回内存空间首地址,失败返回 NULL.

`void *realloc(void *ptr,unsigned size)`重新分配 size 大小的连续内存空间,原来的数据不变.成功返回内存空间首地址,失败返回 NULL.

2.内存释放函数

```
void free(void *ptf);
```

3.注意:

1)使用 malloc,calloc,realloc 分配的内存,必须明确由 free 释放,否则造成内存泄漏.

2)使用 malloc,calloc,realloc 分配内存时,成功返回内存空间首地址,失败返回 NULL.

如果使用 ptr=(Type *)realloc(ptf,newSize),在失败时修改原来的指针 ptr 为 NULL.原来的内存空间无法释放,造成内存泄漏.

9. 链表的概念
 1) 使用结构体定义链表结点数结构.
 2) 使用内存分配函数动态创建链表.

10. 枚举类型
 10.1 枚举类型的定义和枚举变量的声明
  1)枚举类型的定义
  enum EnumType{EnumElem_1[=IdxVal_1]...};
  注意: I. 元素索引值起点默认为 0,可以重新设定.
        II. C 语言允许枚举类型参与数值运算,即使用索引值进行数值计算.

  2)枚举变量的声明
  enum EnumType identifier[=EnumElem];
  注意: 枚举变量声明时需在枚举类型前加 enum 关键字指明 EnumType 是自定义的 enum 类型.

 10.2 枚举变量的赋值和引用
  1)枚举值是常量,不是变量.
  2)枚举元素由系统自动从 0 开始分配索引值,但可以重新设定索引值起点.

11. 类型定义符 typedef
 typedef Original_Type Derived_Type;

12. 共用体的类型定义和共用体变量声明
 共用体和结构体类似,但所有元素共用相同的内存空间.
 共用体的内存空间由最大内存空间的元素决定.
 共用体使用 union 关键字,而不是 struct 关键字.

==========

第 12 章 位运算
C 语言提供位运算功能,使 C 语言能像汇编语言一样编写系统程序.
1.位运算
 C 语言提供了 6 种位运算符:
 ~按位取反
 &按位与
 |按位或
 ^按位异或
 <<算术左移
 >>算术右移

```
int main()
{
   int a=1,b=-1;
   printf("a<<3=%d\n",a<<3);
```

```
    printf("a>>3=%d\n",a>>3);
    printf("b<<3=%d\n",b<<3);
    printf("b>>3=%d\n",b>>3);
}
/*
可见C语言中进行的移位运算都是算术移位运算.
a<<3=8
a>>3=0
b<<3=-8
b>>3=-1
*/
```

---

2. 位域

所谓位域是将一个字节划成几个区域,每个区域有自己的域名:位数,可以程序中操作.

C语言中提供位域的数据结构,其实是一种特殊的结构体.位域类型的定义:

```
struct BitZone
{
 BitType_1 field_name_1;
 ...
 BitType_n field_name_n;
};
```

注意: 1)位域只能存储在同一个字节,不能跨字节.

2)位域的最大长度是一个字节.

3)位域可以没有域名,表示用来填充或调整位置.

```
struct BitZone
{
    int a:4;
    unsigned b:5;
};

int main()
{
    struct BitZone bz;
    bz.a=15;
    bz.b=257;
    int c=257;
    printf("a=%d,\nb=%d,\nc=%d\n",bz.a,bz.b,c);

}
```

a=-1,
b=1,
c=257

==========
第 13 章 文件

c 文件操作函数
---
clearerr（清除文件流的错误旗标）
相关函数
    feof
表头文件
    #include<stdio.h>
定义函数
    void clearerr(FILE * stream);
函数说明
    clearerr（）清除参数 stream 指定的文件流所使用的错误旗标。
返回值

---
fclose（关闭文件）
相关函数
    close，fflush，fopen，setbuf
表头文件
    #include<stdio.h>
定义函数
    int fclose(FILE * stream);
函数说明
    fclose()用来关闭先前 fopen()打开的文件。此动作会让缓冲区内的数据写入文件中，并释放系统所提供的文件资源。
返回值
    若关文件动作成功则返回 0，有错误发生时则返回 EOF 并把错误代码存到 errno。
错误代码
    EBADF 表示参数 stream 非已打开的文件。
范例
    请参考 fopen（）。

---
fdopen（将文件描述词转为文件指针）
相关函数
    fopen，open，fclose

表头文件

#include<stdio.h>

定义函数

FILE * fdopen(int fildes,const char * mode);

函数说明

fdopen()会将参数 fildes 的文件描述词，转换为对应的文件指针后返回。参数 mode 字符串则代表着文件指针的流形态，此形态必须和原先文件描述词读写模式相同。关于 mode 字符串格式请参考 fopen()。

返回值

转换成功时返回指向该流的文件指针。失败则返回 NULL，并把错误代码存在 errno 中。

范例

```
#include<stdio.h>
main()
{
FILE * fp =fdopen(0,” w+” );
fprintf(fp,” %s\n” ,” hello!” );
fclose(fp);
}
```

执行

hello!

---

feof（检查文件流是否读到了文件尾）

相关函数

fopen，fgetc，fgets，fread

表头文件

#include<stdio.h>

定义函数

int feof(FILE * stream);

函数说明

feof()用来侦测是否读取到了文件尾，尾数 stream 为 fopen（）所返回之文件指针。如果已到文件尾则返回非零值，其他情况返回 0。

返回值

返回非零值代表已到达文件尾。

---

fflush（更新缓冲区）

相关函数

write，fopen，fclose，setbuf

表头文件

#include<stdio.h>

定义函数

int fflush(FILE* stream);

函数说明

fflush()会强迫将缓冲区内的数据写回参数 stream 指定的文件中。如果参

数 stream 为 NULL，fflush()会将所有打开的文件数据更新。
返回值
成功返回 0，失败返回 EOF，错误代码存于 errno 中。
错误代码
EBADF 参数 stream 指定的文件未被打开，或打开状态为只读。其它错误代码参考 write（）。

---

fgetc（由文件中读取一个字符）
相关函数
open，fread，fscanf，getc
表头文件
include<stdio.h>
定义函数
nt fgetc(FILE * stream);
函数说明
fgetc()从参数 stream 所指的文件中读取一个字符。若读到文件尾而无数据时便返回 EOF。
返回值
getc()会返回读取到的字符，若返回 EOF 则表示到了文件尾。
范例

```
#include<stdio.h>
main()
{
FILE *fp;
int c;
fp=fopen(“exist”,”r”);
while((c=fgetc(fp))!=EOF)
printf(“%c”,c);
fclose(fp);
}
```

---

fgets（由文件中读取一字符串）
相关函数
open，fread，fscanf，getc
表头文件
include<stdio.h>
定义函数
har * fgets(char * s,int size,FILE * stream);
函数说明
fgets()用来从参数 stream 所指的文件内读入字符并存到参数 s 所指的内存空间，直到出现换行字符、读到文件尾或是已读了 size-1 个字符为止，最后会加上 NULL 作为字符串结束。

返回值
    gets()若成功则返回 s 指针，返回 NULL 则表示有错误发生。
范例
    #include<stdio.h>
```
main()
{
char s[80];
fputs(fgets(s,80,stdin),stdout);
}
```
执行
    this is a test /*输入*/
this is a test /*输出*/

---

fileno（返回文件流所使用的文件描述词）
相关函数
    open，fopen
表头文件
    #include<stdio.h>
定义函数
    int fileno(FILE * stream);
函数说明
    fileno()用来取得参数 stream 指定的文件流所使用的文件描述词。
返回值
    返回文件描述词。
范例
    #include<stdio.h>
```
main()
{
FILE * fp;
int fd;
fp=fopen(“/etc/passwd”,”r”);
fd=fileno(fp);
printf(“fd=%d\n”,fd);
fclose(fp);
}
```
执行
    fd=3

---

fopen（打开文件）
相关函数
    open，fclose
表头文件

#include<stdio.h>

定义函数

FILE * fopen(const char * path,const char * mode);

函数说明

参数 path 字符串包含欲打开的文件路径及文件名，参数 mode 字符串则代表着流形态。

mode 有下列几种形态字符串:

r 打开只读文件，该文件必须存在。

r+ 打开可读写的文件，该文件必须存在。

w 打开只写文件，若文件存在则文件长度清为 0，即该文件内容会消失。若文件不存在则建立该文件。

w+ 打开可读写文件，若文件存在则文件长度清为零，即该文件内容会消失。若文件不存在则建立该文件。

a 以附加的方式打开只写文件。若文件不存在，则会建立该文件，如果文件存在，写入的数据会被加到文件尾，即文件原先的内容会被保留。

a+ 以附加方式打开可读写的文件。若文件不存在，则会建立该文件，如果文件存在，写入的数据会被加到文件尾后，即文件原先的内容会被保留。

上述的形态字符串都可以再加一个 b 字符，如 rb、w+b 或 ab＋等组合，加入 b 字符用来告诉函数库打开的文件为二进制文件，而非纯文字文件。不过在 POSIX 系统，包含 Linux 都会忽略该字符。由 fopen()所建立的新文件会具有 S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH(0666)权限，此文件权限也会参考 umask 值。

返回值

文件顺利打开后，指向该流的文件指针就会被返回。若果文件打开失败则返回 NULL，并把错误代码存在 errno 中。

附加说明

一般而言，开文件后会作一些文件读取或写入的动作，若开文件失败，接下来的读写动作也无法顺利进行，所以在 fopen()后请作错误判断及处理。

范例

```
#include<stdio.h>
main()
{
FILE * fp;
fp=fopen(“noexist”,”a+”);
if(fp= =NULL) return;
fclose(fp);
}
```

---

fputc（将一指定字符写入文件流中）

相关函数

fopen，fwrite，fscanf，putc

表头文件

#include<stdio.h>

定义函数
    int fputc(int c,FILE * stream);
函数说明
    fputc 会将参数 c 转为 unsigned char 后写入参数 stream 指定的文件中。
返回值
    fputc()会返回写入成功的字符，即参数 c。若返回 EOF 则代表写入失败。
范例
    #include<stdio.h>
main()
{
FILE * fp;
char a[26]=”abcdefghijklmnopqrstuvwxyz”;
int i;
fp= fopen(“noexist”,”w”);
for(i=0;i<26;i++)
fputc(a[i],fp);
fclose(fp);
}

---

fputs（将一指定的字符串写入文件内）
相关函数
    fopen，fwrite，fscanf，fputc，putc
表头文件
    #include<stdio.h>
定义函数
    int fputs(const char * s,FILE * stream);
函数说明
    fputs()用来将参数 s 所指的字符串写入到参数 stream 所指的文件内。
返回值
    若成功则返回写出的字符个数，返回 EOF 则表示有错误发生。
范例
    请参考 fgets（）。

---

fread（从文件流读取数据）
相关函数
    fopen，fwrite，fseek，fscanf
表头文件
    #include<stdio.h>
定义函数
    size_t fread(void * ptr,size_t size,size_t nmemb,FILE * stream);
函数说明
    fread()用来从文件流中读取数据。参数 stream 为已打开的文件指针，参数

ptr 指向欲存放读取进来的数据空间，读取的字符数以参数 size*nmemb 来决定。Fread()会返回实际读取到的 nmemb 数目，如果此值比参数 nmemb 来得小，则代表可能读到了文件尾或有错误发生，这时必须用 feof()或 ferror()来决定发生什么情况。

返回值

返回实际读取到的 nmemb 数目。

附加说明

范例

```
#include<stdio.h>
#define nmemb 3
struct test
{
char name[20];
int size;
}s[nmemb];
main()
{
FILE * stream;
int i;
stream = fopen(“/tmp/fwrite”,” r”);
fread(s,sizeof(struct test),nmemb,stream);
fclose(stream);
for(i=0;i<nmemb;i++)
printf(“name[%d]=%-20s:size[%d]=%d\n”,i,s[i].name,i,s[i].size);
}
```

执行

```
name[0]=Linux! size[0]=6
name[1]=FreeBSD! size[1]=8
name[2]=Windows2000 size[2]=11
```

---

freopen（打开文件）

相关函数

fopen，fclose

表头文件

```
#include<stdio.h>
```

定义函数

```
FILE * freopen(const char * path,const char * mode,FILE * stream);
```

函数说明

参数 path 字符串包含欲打开的文件路径及文件名，参数 mode 请参考 fopen()说明。参数 stream 为已打开的文件指针。Freopen()会将原 stream 所打开的文件流关闭，然后打开参数 path 的文件。

返回值

文件顺利打开后，指向该流的文件指针就会被返回。如果文件打开失败则返回 NULL，并把错误代码存在 errno 中。

范例

```
#include<stdio.h>
main()
{
FILE * fp;
fp=fopen(“/etc/passwd”,” r”);
fp=freopen(“/etc/group”,” r”,fp);
fclose(fp);
}
```

---

fseek（移动文件流的读写位置）

相关函数

rewind，ftell，fgetpos，fsetpos，lseek

表头文件

```
#include<stdio.h>
```

定义函数

```
int fseek(FILE * stream,long offset,int whence);
```

函数说明

fseek()用来移动文件流的读写位置。参数 stream 为已打开的文件指针，参数 offset 为根据参数 whence 来移动读写位置的位移数。

参数

whence 为下列其中一种:

SEEK_SET 从距文件开头 offset 位移量为新的读写位置。SEEK_CUR 以目前的读写位置往后增加 offset 个位移量。

SEEK_END 将读写位置指向文件尾后再增加 offset 个位移量。

当 whence 值为 SEEK_CUR 或 SEEK_END 时，参数 offset 允许负值的出现。

下列是较特别的使用方式:

1) 欲将读写位置移动到文件开头时:fseek(FILE *stream,0,SEEK_SET);

2) 欲将读写位置移动到文件尾时:fseek(FILE *stream,0,0SEEK_END);

返回值

当调用成功时则返回 0，若有错误则返回-1，errno 会存放错误代码。

附加说明

fseek()不像 lseek()会返回读写位置,因此必须使用 ftell()来取得目前读写的位置。

范例

```
#include<stdio.h>
main()
{
FILE * stream;
long offset;
fpos_t pos;
```

```
stream=fopen(“/etc/passwd”,” r”);
fseek(stream,5,SEEK_SET);
printf(“offset=%d\n”,ftell(stream));
rewind(stream);
fgetpos(stream,&pos);
printf(“offset=%d\n”,pos);
pos=10;
fsetpos(stream,&pos);
printf(“offset = %d\n”,ftell(stream));
fclose(stream);
}
```

执行

```
    offset = 5
offset =0
offset=10
```

---

ftell（取得文件流的读取位置）

相关函数

fseek，rewind，fgetpos，fsetpos

表头文件

#include<stdio.h>

定义函数

long ftell(FILE * stream);

函数说明

ftell()用来取得文件流目前的读写位置。参数 stream 为已打开的文件指针。

返回值

当调用成功时则返回目前的读写位置，若有错误则返回-1，errno 会存放错误代码。

错误代码

EBADF 参数 stream 无效或可移动读写位置的文件流。

范例

参考 fseek()。

---

fwrite（将数据写至文件流）

相关函数

fopen，fread，fseek，fscanf

表头文件

#include<stdio.h>

定义函数

size_t fwrite(const void * ptr,size_t size,size_t nmemb,FILE * stream);

函数说明

fwrite()用来将数据写入文件流中。参数 stream 为已打开的文件指针，参数 ptr 指向欲写入的数据地址，总共写入的字符数以参数 size*nmemb 来决定。Fwrite()会返回实际写入的 nmemb 数目。

返回值

返回实际写入的 nmemb 数目。

范例

```
#include<stdio.h>
#define set_s (x,y) {strcoy(s[x].name,y);s[x].size=strlen(y);}
#define nmemb 3
struct test
{
char name[20];
int size;
}s[nmemb];
main()
{
FILE * stream;
set_s(0,”Linux!”);
set_s(1,”FreeBSD!”);
set_s(2,”Windows2000.”);
stream=fopen(“/tmp/fwrite”,”w”);
fwrite(s,sizeof(struct test),nmemb,stream);
fclose(stream);
}
```

执行

参考 fread（）。

---

getc（由文件中读取一个字符）

相关函数

read，fopen，fread，fgetc

表头文件

```
#include<stdio.h>
```

定义函数

```
int getc(FILE * stream);
```

函数说明

getc()用来从参数 stream 所指的文件中读取一个字符。若读到文件尾而无数据时便返回 EOF。虽然 getc()与 fgetc()作用相同，但 getc()为宏定义，非真正的函数调用。

返回值

getc()会返回读取到的字符，若返回 EOF 则表示到了文件尾。

范例

参考 fgetc()。

---

getchar（由标准输入设备内读进一字符）

相关函数

fopen，fread，fscanf，getc

表头文件

#include<stdio.h>

定义函数

int getchar(void);

函数说明

getchar()用来从标准输入设备中读取一个字符。然后将该字符从 unsigned char 转换成 int 后返回。

返回值

getchar()会返回读取到的字符，若返回 EOF 则表示有错误发生。

附加说明

getchar()非真正函数，而是 getc(stdin)宏定义。

范例

```
#include<stdio.h>
main()
{
FILE * fp;
int c,i;
for(i=0li<5;i++)
{
c=getchar();
putchar(c);
}
}
```

执行

```
1234 /*输入*/
1234 /*输出*/
```

---

gets（由标准输入设备内读进一字符串）

相关函数

fopen，fread，fscanf，fgets

表头文件

#include<stdio.h>

定义函数

char * gets(char *s);

函数说明

gets()用来从标准设备读入字符并存到参数 s 所指的内存空间，直到出现换行字符或读到文件尾为止，最后加上 NULL 作为字符串结束。

返回值

gets()若成功则返回 s 指针，返回 NULL 则表示有错误发生。

附加说明

由于 gets()无法知道字符串 s 的大小，必须遇到换行字符或文件尾才会结束输入，因此容易造成缓冲溢出的安全性问题。建议使用 fgets()取代。

范例

参考 fgets()

---

mktemp（产生唯一的临时文件名）

相关函数

tmpfile

表头文件

#include<stdlib.h>

定义函数

char * mktemp(char * template);

函数说明

mktemp()用来产生唯一的临时文件名。参数 template 所指的文件名称字符串中最后六个字符必须是 XXXXXX。产生后的文件名会借字符串指针返回。

返回值

文件顺利打开后，指向该流的文件指针就会被返回。如果文件打开失败则返回 NULL，并把错误代码存在 errno 中。

附加说明

参数 template 所指的文件名称字符串必须声明为数组，如:

char template[ ]=”template-XXXXXX”;

不可用 char * template=”template-XXXXXX”;

范例

```
#include<stdlib.h>
main()
{
char template[ ]=”template-XXXXXX”;
mktemp(template);
printf(“template=%s\n”,template);
}
```

---

putc（将一指定字符写入文件中）

相关函数

fopen，fwrite，fscanf，fputc

表头文件

#include<stdio.h>

定义函数

int putc(int c,FILE * stream);

函数说明

putc()会将参数 c 转为 unsigned char 后写入参数 stream 指定的文件中。虽然 putc()与 fputc()作用相同，但 putc()为宏定义，非真正的函数调用。

返回值
putc()会返回写入成功的字符，即参数 c。若返回 EOF 则代表写入失败。
范例
参考 fputc（）。

---

putchar（将指定的字符写到标准输出设备）
相关函数
fopen，fwrite，fscanf，fputc
表头文件
#include<stdio.h>
定义函数
int putchar (int c);
函数说明
putchar()用来将参数 c 字符写到标准输出设备。
返回值
putchar()会返回输出成功的字符，即参数 c。若返回 EOF 则代表输出失败。
附加说明
putchar()非真正函数，而是 putc(c，stdout)宏定义。
范例
参考 getchar()。

---

rewind（重设文件流的读写位置为文件开头）
相关函数
fseek，ftell，fgetpos，fsetpos
表头文件
#include<stdio.h>
定义函数
void rewind(FILE * stream);
函数说明
rewind()用来把文件流的读写位置移至文件开头。参数 stream 为已打开的文件指针。此函数相当于调用 fseek(stream,0,SEEK_SET)。
返回值

范例
参考 fseek()

---

setbuf（设置文件流的缓冲区）
相关函数
setbuffer，setlinebuf，setvbuf
表头文件
#include<stdio.h>
定义函数

```
    void setbuf(FILE * stream,char * buf);
```
函数说明

在打开文件流后，读取内容之前，调用 setbuf()可以用来设置文件流的缓冲区。参数 stream 为指定的文件流，参数 buf 指向自定的缓冲区起始地址。如果参数 buf 为 NULL 指针，则为无缓冲 IO。Setbuf()相当于调用:
setvbuf(stream,buf,buf?_IOFBF:_IONBF,BUFSIZ)
返回值

---

setbuffer（设置文件流的缓冲区）
相关函数

```
    setlinebuf, setbuf, setvbuf
```
表头文件

```
    #include<stdio.h>
```
定义函数

```
    void setbuffer(FILE * stream,char * buf,size_t size);
```
函数说明

在打开文件流后，读取内容之前，调用 setbuffer()可用来设置文件流的缓冲区。参数 stream 为指定的文件流，参数 buf 指向自定的缓冲区起始地址，参数 size 为缓冲区大小。
返回值

---

setlinebuf（设置文件流为线性缓冲区）
相关函数

```
    setbuffer, setbuf, setvbuf
```
表头文件

```
    #include<stdio.h>
```
定义函数

```
    void setlinebuf(FILE * stream);
```
函数说明

setlinebuf()用来设置文件流以换行为依据的无缓冲 IO。相当于调用:setvbuf(stream,(char *)NULL,_IOLBF,0);请参考 setvbuf()。
返回值

---

setvbuf（设置文件流的缓冲区）
相关函数

```
    setbuffer, setlinebuf, setbuf
```
表头文件

```
    #include<stdio.h>
```
定义函数

```
    int setvbuf(FILE * stream,char * buf,int mode,size_t size);
```
函数说明

在打开文件流后，读取内容之前，调用 setvbuf()可以用来设置文件流的缓冲区。参数 stream 为指定的文件流，参数 buf 指向自定的缓冲区起始地址，参数 size 为缓冲区大小，参数 mode 有下列几种
_IONBF 无缓冲 IO
_IOLBF 以换行为依据的无缓冲 IO
_IOFBF 完全无缓冲 IO。如果参数 buf 为 NULL 指针，则为无缓冲 IO。
返回值

---

ungetc（将指定字符写回文件流中）
相关函数
fputc，getchar，getc
表头文件
#include<stdio.h>
定义函数
int ungetc(int c,FILE * stream);
函数说明
ungetc()将参数 c 字符写回参数 stream 所指定的文件流。这个写回的字符会由下一个读取文件流的函数取得。
返回值
成功则返回 c 字符，若有错误则返回 EOF。
