C/C++/算法复习

liz*

April 15, 2009

Contents

*Email: shengyan1985@gmail.com

1 C基础知识

1.1 数组和字符数组

1. 数组声明和初始化 float b[1][2][3] = 0.0;
2. 字符数组 char str[] = "china"; // 字符串常量尾部都有字符\0作为字符串结束符
3. getchar()和putchar() putchar(getchar())
4. puts(s) 输出字符串s // stdio.h
5. gets(s) 从标准输入中断接受以\n结束的串转换为\0后存入s,并返回串s的指针. // stdio.h
6. strcat(s1, s2) 将s1和s2合并为s1, 并返回s1地址 //string.h
7. strcpy(s1, s2) 将s2拷入s1中, 并返回s1地址 //string.h
8. strcpy(s1, s2, n) 将s2的前n个字符拷入s1中, 并返回s1地址 //string.h
9. strcmp(s1, s2) 比较s1, s2大小, 若相等, 返回0; 若不等, 返回第一个不等字符的ASCII码差值
10. strlen(s) 测试s的实际长度

1.2 函数

1. 函数参数从右到左计算, 形实结合是值传递.
2. 函数返回值类型(函数类型)缺省为整型, void定义无(空)类型, 表示函数不返回值.
3. 如果return返回类型于函数类型不一致, 以函数类型为准.
4. 但数组是地址传递, 因为数组名本质是一个地址. 这里的形参数组长度可缺省说明, 形实数组长度可不等, 但形参数组长不能大于实参数组长.
5. 函数允许嵌套调用, 不允许嵌套定义.

1.3 function相关算法整理

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

/********************
模拟strcat(s1, s2)功能
********************/
char * mystrcat(char s1[], char s2[])
{
    int i = 0, j = 0;
    while (s1[i] != '\0') i++; //while (s[i]) i++;
    while ((s1[i]=s2[j]) != '\0')
    {
        i++;
        j++;
    }
    return s1;
}

/********************
模拟strcpy(s1, s2)功能
********************/
char * mystrcpy(char s1[], char s2[])
{
    int i;
    for (i=0; (s1[i]=s2[i]) != '\0'; i++); // for (i=0; (s1[i]=s2[i]); i++);
```

```
    return s1;
}

/*******************
字符串反转功能
*******************/
char * reverse(char ss[])
{
    int i, j, c;
    for (i=0, j=strlen(ss)-1; i<j; i++, j--)
    {
        c = ss[i];
        ss[i] = ss[j];
        ss[j] = c;
    }
    return ss;
}

/*******************
统计字符串中单词个数，以空格为分隔符
碰到空格后的第一个非空格字符，形如 _ A，表示出现一个单词
*******************/
int get_words_count(char ss[])
{
    int i, num=0, word=0;
    for (i=0; ss[i]; i++)
    {
        if (ss[i] == '\40') word = 0;
        else if (word == 0)
        {
            word = 1;
            num++;
        }
    }
    return num;
}

int main(void)
{
    char s1[10], s2[20];
    //scanf("%s", s1);
    //scanf("%s", s2);
    //puts(mystrcat(s1, s2));
    //puts(mystrcpy(s1, s2));
    //puts(reverse(s2));
    //printf("%d\n", get_words_count(gets(s2)));
    printf("%d %d\n", 12/7, 12%7);
    return 0;
}
```

1.4 变量存储

1. 变量生存期： 变量存储单元存在的时效.
2. 变量作用域： 变量值可被访问的范围.
3. 全局变量(extern)： 在任何函数体外定义的变量. 生存期为整个程序执行期，作用域为从定义点到本源文件结束(隐含规则). 如果是同一源文件中访问，在定义点前访问,需使用extern属性说明外部变量；在定义点后访问，可缺省extern说明. 如果是其他文件中访问，在引用外部变量的源文件开头处用extern说明外部变量.
4. 静态变量： 在变量定义前使用static属性说明
5. 静态局部变量： 在局部变量类型定义前用static定义. 作用域仅限于定义它的函数内访问(同auto)，生存期同整个程序执行期. 于是乎，在多次函数调用中，静态局部变量保持值的连续性.
6. 静态全局变量： 在全局变量定义处用static说明，生存期： 同整个程序执行期，作用域： 仅限于本编译单位(源文件，避免冲突??).

7. 寄存器变量： 使用register定义的变量，仅使用auto变量，不适用全局变量和静态变量． 它的值存防御cpu通用寄存器中，不在普通内存中，提高速度．

8. 内部函数： 在函数类型定义前加static属性说明，使得该函数的作用域为本文件内(编译单位内，避免冲突)

9. 外部函数： 在函数类型定义前加extern属性说明或缺省． 一个文件想要调用另一个文件中的函数，需对所调外部函数进行说明．

10. 变量的定义形式，完整为： 存储类别 数据类型 变量列表; //存储类别为： auto, static, register, extern
    函数的定义形式，完整为： 存储类别 数据类型 函数名(形参列表) 函数体
    静态变量或外部变量如未初始化，系统自动使其初值为0(对数值类型)或空(对字符型数据)，而自动变量或寄存器变量，则其初值为随机数据．

1.5 宏

1. # define 宏名标识符 字符串
   编译预处理包括的工作： 宏定义，文件包含，条件编译．

2. # undef 宏名 用于终止宏名作用域，宏的默认作用域是从定义点到本源文件结束．
   带参宏,,,#define 宏名(参数表) 含参数字符串 ,,, 宏调用时提供实际参数． 宏展开用实参替代宏名中形参，对#define行中的字符串从左到右进行置换，将置换后的串在宏调用处宏展开...

1.6 条件编译

```
1) # ifdef 标识符 //若标识符已用#define定义过，则只编译段1
2)       程序段1
3) [# else
4)       程序段2]
5) # endif
```

```
1) # if 表达式 //表达式值为真(非零)，则编译段1，否则编译段2
2)       程序段1
3) [# else
4)       程序段2]
5) # endif
```

附上所有预处理命令:

```
# define 宏名 字符串
# define 宏名(参数1, 参数2, ..., 参数n) 字符串
# undef 宏名
# include "文件名" (或<文件名>)
# if 常量表达式
# ifdef 宏名
# ifndef 宏名
# else
# endif
```

1.7 指针

1. 变量指针与直接访问 变量指针： 变量的内存起始地址
   直接访问： 按变量指针存取变量值的方式．

2. 指针变量和间接访问 指针变量: 存放指针值的变量
   间接访问: 通过访问指针变量获得所访变量指针再访问变量的方式.
   类型说明符 * 指针变量名1, *指针变量名2; 指针变量的类型特指其所指向变量的类型.
   取址运算符&: 取出变量(常量是没有地址的)地址; 不能用整型量和任何非地址类型的数给指针变量赋值.
   间接访问(指针)运算符* ,, *和&满足有结合性. 严禁使用未初始化指针变量.

3. 数组和指针 数组名代表数组指针, 是指针常量.
   ++p, p++, --p, p--, 若p=&a[i], 则++p, p++等价于&a[i+1], 表示指针变量加(减)所指元素内存字节数.
   p+i等价于p=p+i*sizeof(arraytype), 即指针变量向前/后移i个数组元素.
   arraytype a[n], *p=a; 则p+i=a+i=&a[i],,,*(p+i)=*(a+i)=a[i]=*&a[i],,,[]又称为变址运算符.
   若二指针变量指向同一数组, 则可进行减法(获得相差个数)和比较运算.

4. 数组元素的访问

   (a) 下标法,,a[i]形式 &a[i]等价于a+i,, a[i]等价于*(a+i)

   (b) 指针变量法,,*(a+i)或*(p+i)形式 同上

   (c) 带下标的指针变量,,p[i]形式 (a+i)获得数组a的第i个元素地址,,,p[i]虽然p是指针, 但仍然可以使用[]取第i个元素值.

   (d) 指针变量增1运算 如程序中...

   (e) 数组名作为函数参数 它传递的是地址, 实参和形参可以是数组名和指针变量...

   (f) 多维数组和指针 对于二维数组, 数组名a代表的是首行首地址, a+1代表第一行的首地址, 因为现在的首元素是一个由4个整型元素所组成的一维数组. a+1的含义是a[1]的地址, 即a+4*2=...而a[0],a[1]既然是一维数组名, 则它们代表一维数组a[0]中第0列元素的地址, 即&a[0][0],&a[1][0], a[0]等价于*(a+0), a[0]+1等价于*(a+0)+1, 它们值为&a[0][1], a[i][j]可以通过*(a[i]+j), *(*(a +i)+j)取值.,,一定要记住*(a+i)和a[i]是等价的. a[i]从形式上看是a数组的第i个元素, 如果a是一维数组名, 则a[i]代表a数组第i个元素所占内存单元的内容, a[i]有有物理地址的, 占内存单元. 但如果a是二维数组, a[i]代表一维数组名, 它只是一个地址,a,a+i, a[i], *(a+i), *(a+i)+j, a[i]+j都是地址.
   a为二维数组名, 指向一维数组a[0], 即第0行首地址. a[0], *(a+0), *a为第0行第0列元素a[0][0]的地址. 纵向的 a+i, &a[i]为第1行首地址,,,横向的.
   所以a+1和*(a+1)是相同的地址,,,但一个行上的, 一个列上看的.
   一维数组名是指向列元素的,,在指向行的指针前面加一个*, 就转换为指向列的指针. 反之, 在指向列的指针前面加上&就称为指向行的指针.
   不要把&a[i]简单得理解为a[i]单元的物理地址, 因为不存在a[i]这样一个实际的变量, 他只是一种地址的计算方法, 能得到第i行的首地址, &a[i]和a[i]的值是一样的,但它们的含义不同. &a[i]或a+i指向行, 而a[i]或*(a+i)指向列.

5. 函数与指针 int (*p)(); // 声明p是一个指向函数的指针变量, 此函数带回整型的返回值, 不能写成 int *p(); 这样就是声明了一函数了.
   p = max; 赋给p为一个函数, 在使用时可以c=(*p)(a,b); 这相当于c=max(a,b);
   对指向函数的指针变量进行p+n, p++等运算是无意义的.

6. 用指向函数的指针作为函数参数 例子在代码中.

7. 返回指针值的函数 类型名 * 函数名(参数列表),,,一样用, 只是返回的指针不要是局部的, 也就是不要过期了.

8. 指针数组和指向指针的指针 类型名 * 数组名[数组长度]; 主要用在字符串数组中,如:
   char * name[] = {"follow me", "Basic", "other"}; // name就是为char **, 这和main函数的参数类表(int argc, char * argv[])的argv是一个道理.
   char ** p;// *的从右到左的结合性, 可以看成是*(*p) 表示指针变量p是指向一个字符指针变量, p = name+2, 表示name[2].

1.8 array相关算法整理

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

/*******************
统计s数组中串的实际长度
*******************/
int get_string_real_count(char *ss)
{
    char * ps=ss;
    while (*ps != '\0') ps++;
    return ps-ss;
}

/*******************
字符串与指针
*******************/
int string(void)
{
    char * string="some string"; // 这个只是将"some string"的首字符地址赋给string指针，其指向的是一字符类型.
    char * string1;
    char str[20]={"some string"}; // 数组初始化，之后就能整体赋值了. str[]="some string"是不对的.
    //*string1 = "some string"; // 这才是真正的将字符串赋给string1指针. 不过测试出来不对阿...
    printf("%s\n", string);
    printf("%d\n", sizeof(string));
    //printf("%s\n", string1);

    char * a, str2[10];
    a = str2;
    scanf("%s", a);  //这样做比较安全，比不加a = str2;好，因为光光char * a中a的值是随机的，有可能指向已用的内存段.

}

// 调用时可以sub(f1, f2);这在每次调用sub函数时，要调用的函数不是固定的，就很方便了.
void sub(int (* x1)(int), int (* x2)(int, int)) //x1指向的函数有一个整型形参，x2指向的函数有两个整型形参
{
    int a, b, i, j;
    a = (* x1)(i);
    b = (* x2)(i,j);
}

//在举个函数指针的例子
int max(int, int); //两个数的较大者 先作声明，因为在process中使用max作为形参，表明max是函数，而不是普通变量.
int min(int, int); //两个数的较小者
int add(int, int); //两个数的和

//现在定义一个process，要求可以实现不同功能...调用时process(10, 10, max), process(10, 10, add);
void process(int x, int y, int (* func)(int, int))
{
    int result;
    result = (*func)(x,y);
    printf("%d\n", result);
}

void mainmain(int argc, char * argv[]) // 这个argv值可变，而不是像之前的数组名常量，因为形参是变量
{
    while (argc-->1) printf("%s\n", *++argv);
    while (--argc>0) printf("%s%c", *++argv, (argc>1)?' ':'\n');
}
int main(void)
{
    char s1[10];
    int a[10], i, *p;

    /*scanf("%s", s1);
    printf("%d\n", get_string_real_count(s1));

    //指针变量增1运算,,, 运行速度会比(a+i)等快
```

```
68      for (i=0; i<10; i++) scanf("%d", (a+i));
69      for (p=a; p<(a+10); p++) printf("%d", *p);
70      p = a;
71      for (i=0; i<10; i++) printf("%d", *p++); */
72      string();
73      printf("\n");
74      return 0;
75  }
```

1.9 一些关于指针的注意点

1. 指针类型变量和整型变量不能相互赋值
2. 指针变量可以为空值，即该指针变量不指向任何变量，p=NULL，这个NULL在stdio.h中符号常量为0，可以与NULL值进行相等性比较
3. 两个指针变量比较，同一数组中的两个指针p，进行<，>比较． 不在同一数组中的两个指针比较是没有意义的．
4. 关于void *，就是不指定它是指向哪一种类型数据的． "ANSI C标准规定用动态存储分配函数时返回void指针，它可以用来指向一个抽象的类型的数据，在将它的值赋给另一指针变量时要进行强制类型转换使之适合于被赋值的变量的类型．"
   p1 = (char *)p2; or p2 = (void *)p1; 强制转换成需要的类型．

1.10 结构体

struct 结构体名
{ 成员列表 };
struct 结构体名 // 或者结构体名直接省略
{ 成员列表 } 变量名表列;
指向结构体变量的指针 struct student *p = &stu; 使用p->name，或者(*p).name，或者stu.name都可以访问成员变量．

1.11 共用体

现在想起来，这个东西是多么稀有的东西．
union 共用体名
{ 成员表列 } 变量表列;
共用体是使几种不同类型的变量存放到同一段内存单元中． 变量所占内存的字节数不同，但都是从同一地址开始，使用覆盖技术，几个变量相互覆盖，使几个不同的变量共占同一段内存的结构．
a.i访问共用体a中的变量i． 多个变量在一个时间点上只有一个有意义，比如，a.f= 0.0，将会冲掉之前的a.i值，所以这时访问a.i是没有多大意义的．

1.12 枚举类型

声明: enum weekday { sun, mon, tue, wed, thu, fri, sat }; {}中的为枚举常量，顺序从0开始
定义枚举变量: enum weekday myweekday=mon;//赋为mon值
也可以直接: enum { sun, mon, tue, wed, thu, fri, sat } myweekday;
改变枚举元素的值: enum { sun=7, mon=1, tue, wed, thu, fri, sat } sw;
if (sw == mon) ...
if (sw < sun) ...
sw = (enum weekday)2; //将序号为2的枚举元素赋给sw，相当于sw = tue，sw = (enum weekday)(5-3);

1.13 用typedef定义类型

```
typedef int INTEGER;
typedef struct ... DATE;
typedef int NUM[100]; NUM n;
typedef char * STRING; STRING p, s[10];
typedef int (* POINTER)(); POINTER  p1;
```

typedef和#define的不同: typedef是编译时处理的, #define是在预编译时处理的,作简单字符串替换; 前者是语句, 后者是宏命令.

1.14 位运算

&: 位与
|: 位或
^: 位异或
: 取反
<<: 左移
>>: 右移
相关组合, 可以形式循环移位等.

1.15 文件

1. fopen: FILE * fp; fp = fopen(文件名, 使用文件方式); 文件打开方式: r(只读), w(只写), a(追加)/文本文件, rb, wb, ab/二进制文件, r+/读写代开一个文本文件, w+/读写建立一个新的文本文件, a+/读写打开一个文本文件, rb+, wb+, ab+
   打开失败, fopen函数带回一个出错信息, fopen返回的是一个NULL.

2. fclose: fclose(文件指针);

3. fputc: 把一个字符写到磁盘文件上去.
   一般调用形式: fputc(ch, fp);
   作用: 将字符ch输出到fp所指向的文件中去. 如果输出成功, 返回这个输出的字符, 失败, 返回EOF(stdio.h中的符号常量, 值为-1).
   之前的putchar是由fputc派生出来的. #define putchar(c) fputc(c, stdout) 这将c输出到标准输出.

4. fgetc: 从指定文件读入一个字符, 该文件必须是以读或读写方式打开.
   调用形式: ch = fgetc(fp); 如果在执行fgetc读字符时遇到文件结束符, 函数返回一个文件结束标志EOF(-1), 这可作为判断文件结束条件.

5. fread和fwrite: 读写一个数据块
   调用形式: fread(buffer, size, count, fp); fwrite(buffer, size, count, fp); buffer是一个指针, 对fread来说,是读入数据的存放地址, 对fwrite来说, 是要输出数据的地址. size为读写的字节数, count要进行读写多个size字节的数据项. 如果文件以二进制形式打开, 用fread和fwrite就可以读写任何类型的信息. 如fread(f, 4, 2, fp); 表示从fp所指向的文件读入2个2个字节的数据并存储到数组f中. 对于复杂的结构体类型, 同样可以使用这种方式整个读写一个结构体.
   fread和fwrite调用成功, 返回值为count的值, 即输入或输出数据项的完整个数.

6. fprintf和fscanf: 和printf, scanf函数作用相仿, 都是格式化读写函数, 只是前者的读写对象不是终端而是磁盘文件.
   调用形式: fprintf(文件指针, 格式字符串, 输出列表); fscanf(文件指针, 格式字符串, 输入列表);
   使用fprintf和fscanf函数, 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(--难道只能是读写文本文件, 好像是的哦), 花费时间较多. 因此在内存与磁盘频繁交换数据的情况下,最好是使用fread和fwrite函数.

7. putw和getw: 用来对磁盘文件读写一个字(整数, 一般为2个字节, 但根据系统不同又所差别).
   putw(10, fp); 将整数10输出到fp指向的文件. 而i = getw(fp); 从磁盘文件读一个整数到内存, 赋给整型变量i.

8. fgets和fputs:从指定文件中读入一个字符串/向指定文件中输出一个字符串.
   fgets(str, n, fp); n为要求得到的字符, 但只从fp指向的文件输入n-1个字符, 最后加一个'\0', 放于字符数组str中, 如果在读完n-1个字符之前遇到换行符或EOF, 读入马上结束. fgets返回值为str的首地址.
   fputs("sring", fp); 字符串末尾的'\0'不输出到文件, 若输出成功, 函数值为0, 失败时, 为EOF.

9. rewind:使位置指针重新返回文件的开头, 没有返回值.
   rewind(fp); 它使文件的位置指针重新定位于文件开头, 并使feof函数的值恢复为0(假).

10. fseek: 对流式文件可以进行顺序读写, 也可以进行随机读写. 关键在于控制文件的位置指针, 如果位置指针是按照字节位置顺序移动的, 就是顺序读写. 如果将位置指针按需要移动到任意位置, 就可以实现随机读写, 读写完上一个字符(字节)后, 并不一定要读写其后续的字符(字节), 可以读写文件中任意位置上所需要的字符(字节). fseek正是实现了改变文件的位置指针.
    调用形式: fseek(文件类型指针, 位移量, 起始点); 起始点有 文件开始(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, 向前移动的字节数, long型数据.
    fseek函数一般用于二进制文件, 因为文本文件要发生字符转换, 计算位置时往往会发生混乱.
    fseek(fp, i*sizeof(struct student_type), o); 然后fread(&stud[i], sizeof(struct student_type), 1, fp);

11. ftell: 获得流式文件中的当前位置, 用相对于文件开头的位移量来表示.
    i = ftell(fp); 若返回-1L表示出错.

12. ferror和clearerr: 在调用上述输入输出函数时, 如果出现错误, 除了返回数值反映外, 还可以用ferror来检查
    ferror(fp); 如果返回值为0(假), 表示未出错. 如果返回一个非零值, 表示出错. 对同一文件每一次调用输入输出函数, 均产生一个新的ferror函数值, 所以应当在调用一个输入输出函数立即检查ferror函数的值, 否则会丢失信息. 在执行fopen函数时, ferror函数的初始值置为0;
    clearerr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或rewind函数.

1.16 文件相关算法整理

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

/********************
读取一个文本文件
********************/
void read_text_file(void)
{
    char ch;
    FILE * fp;
    fp = fopen("test", "r");
    ch = fgetc(fp);
    while (ch != EOF)
    {
        putchar(ch);
        ch = fgetc(fp);
    }
}

/********************
顺序读取一个二进制文件
********************/
void read_text_file(void)
{
```

```
    char ch;
    FILE * fp;
    fp = fopen("test", "rb");
    while (!feof(fp)) //feof值为0时表示未到文件结束，读入一个字节的数据赋给整型变量c，遇到文件结束时，
       feof为1. 这种方法也适用于文本文件.
    {
        ch = fgetc(fp);
        //...
    }
    fclose(fp);
}

/********************
自定义的getw函数
********************/
int getw(FILE * fp)
{
    char * ch;
    int i;
    s = (char * )&i;
    s[0] = getc(fp);
    s[1] = getc(fp);
    return i;
}

/********************
自定义的putw函数
********************/
int putw(int i, FILE * fp)
{
    char * s;
    s = (char * )&i;
    putc(s[0],fp);
    putc(s[1],fp);
    return i;
}

/********************
在系统不提供fread和fwrite时，编写任何类型的读写文件函数，这里putfloat，将一个浮点数写入文件
********************/
int putfloat(float num, FILE * fp)
{
    char * s;
    int count;
    s = (char * )&num;
    for (count=0; count<4; count++) putc(s[count], fp);
    return 1;
}
```

2 C库函数使用

2.1 malloc函数

函数原型: void * malloc(unsigned int size);
作用: 在内存的动态存储区中分配一个长度为size的连续空间, 返回的是一个指向该分配域起始地址的指针(类型为void), 如果分配失败(如内存空间不足), 则返回空指针(NULL). 返回的void * 常被强制转换为自己需要的指针类型.

2.2 calloc函数

函数原型: void * calloc(unsigned n, unsigned size);
作用: 在内存动态存储区中分配n个长度为size的连续空间, 返回的是一个指向该分配域起始地址的指针(类型为void), 如果分配失败(如内存空间不足), 则返回空指针(NULL). 用calloc函数可以为一维数组开辟动态存储空间, n为数组元素个数, 每个元素长度为size.

2.3 free函数

函数原型: void free(void * p);
作用: 释放由p指向的内存区, p是最近一次调用calloc或malloc函数时返回的值. free函数无返回值.