C语言难点分析整理！[转贴]

这篇文章主要是介绍一些在复习 C 语言的过程中笔者个人认为比较重点的地方，较好的掌握这些重点会使对 C 的运用更加得心应手。此外会包括一些细节、易错的地方。涉及的主要内容包括：变量的作用域和存储类别、函数、数组、字符串、指针、文件、链表等。一些最基本的概念在此就不多作解释了，仅希望能有只言片语给同是 C 语言初学者的学习和上机过程提供一点点的帮助。

变量作用域和存储类别：

了解了基本的变量类型后，我们要进一步了解它的存储类别和变量作用域问题。

变量类别	子类别
局部变量	静态变量（离开函数，变量值仍保留）
	自动变量
	寄存器变量
全局变量	静态变量（只能在本文件中用）
	非静态变量（允许其他文件使用）

换一个角度

变量类别	子类别

静态存储变量	静态局部变量（函数）
	静态全局变量（本文件）
	非静态全局/外部变量（其他文件引用）
动态存储变量	自动变量
	寄存器变量
	形式参数

extern 型的存储变量在处理多文件问题时常能用到，在一个文件中定义 extern 型的变量即说明这个变量用的是其他文件的。顺便说一下，笔者在做课设时遇到 out of memory 的错误，于是改成做多文件，再把它 include 进来（注意自己写的*.h 要用“”不用<>），能起到一定的效用。static 型的在读程序写结果的试题中是个考点。多数时候整个程序会出现多个定义的变量在不同的函数中，考查在不同位置同一变量的值是多少。主要是遵循一个原则，只要本函数内没有定义的变量就用全局变量（而不是 main 里的），全局变量和局部变量重名时局部变量起作用，当然还要注意静态与自动变量的区别。

函数：

对于函数最基本的理解是从那个叫 main 的单词开始的，一开始总会觉得把语句一并写在 main 里不是挺好的么，为什么偏择出去。其实这是因为对函数还不够熟练，否则函数的运用会给我们编程带来极大的便利。我们要知道函数的返回值类型，参数的类型，以及调用函数时的形式。事先的函数说明也能起到一个提醒

的好作用。所谓形参和实参，即在调用函数时写在括号里的就是实参，函数本身用的就是形参，在画流程图时用平行四边形表示传参。

函数的另一个应用例子就是递归了，笔者开始比较头疼的问题，反应总是比较迟钝，按照老师的方法，把递归的过程耐心准确的逐级画出来，学习的效果还是比较好的，会觉得这种递归的运用是挺巧的，事实上，著名的八皇后、汉诺塔等问题都用到了递归。

例子：

```
long fun(int n)
{
long s;
if(n==1||n==2) s=2;
  else s=n-fun(n-1);
return s;
}
main()
{
printf("%ld",fun(4));
}
```

数组：

分为一维数组和多维数组，其存储方式画为表格的话就会一目了然，其实就是把相同类型的变量有序的放在一起。因此，在处理比较多的数据时（这也是大多数的情况）数组的应用范围是非常广的。

具体的实际应用不便举例，而且绝大多数是与指针相结合的，笔者个人认为学习数组在更大程度上是为学习指针做一个铺垫。作为基础的基础要明白几种基本操作：即数组赋值、打印、排序（冒泡排序法和选择排序法）、查找。这些都不可

避免的用到循环，如果觉得反应不过来，可以先一点点的把循环展开，就会越来越熟悉，以后自己编写一个功能的时候就会先找出内在规律，较好的运用了。另外数组做参数时，一维的[]里可以是空的，二维的第一个[]里可以是空的但是第二个[]中必须规定大小。

冒泡法排序函数：

```
void  bubble(int  a[],int  n)
{
int  i,j,k;
for(i=1,i<n;i++)
  for(j=0;j<n-i;j++)
  if(a[j]>a[j+1])
   {
    k=a[j];
     a[j]=a[j+1];
     a[j+1]=k;
     }
}
```

选择法排序函数：

```
void  sort(int  a[],int  n)
{
int  i,j,k,t;
for(i=0,i<n-1;i++)
  {
  k=i;
  for(j=i+1;j<n;j++)
    if(a[k]<a[j])  k=j;
    if(k!=i)
      {
```

```
        t=a[i];
        a[i]=a[k];
        a[k]=t;
        }
   }
}
```

折半查找函数（原数组有序）：

```
void search(int a[],int n,int x)
{
int left=0,right=n-1,mid,flag=0;
while((flag==0)&&(left<=right))
   {
   mid=(left+right)/2;
   if(x==a[mid])
      {
      printf("%d%d",x,mid);
      flag =1;
      }
      else if(x<a[mid]) right=mid-1;
                else left=mid+1;
   }
}
```

相关常用的算法还有**判断回文，求阶乘，Fibanacci 数列，任意进制转换，杨辉三角形计算**等等。

字符串：

字符串其实就是一个数组（指针），在 scanf 的输入列中是不需要在前面加“&”符号的，因为字符数组名本身即代表地址。值得注意的是字符串末尾的‘\0’，如果没有的话，字符串很有可能会不正常的打印。另外就是字符串的定义和赋值问题了，笔者有一次的比较综合的上机作业就是字符串打印老是乱码，上上下下找了一圈问题，最后发现是因为

```
char *name;
```

而不是

```
char name[10];
```

前者没有说明指向哪儿，更没有确定大小，导致了乱码的错误，印象挺深刻的。

另外，字符串的赋值也是需要注意的，如果是用字符指针的话，既可以定义的时候赋初值，即

```
char *a="Abcdefg";
```

也可以在赋值语句中赋值，即

```
char *a;
a="Abcdefg";
```

但如果是用字符数组的话，就只能在定义时整体赋初值，即 char a[5]={"abcd"}; 而不能在赋值语句中整体赋值。

常用字符串函数列表如下，要会自己实现：

函数作用	函数调用形式	备注

字符串拷贝函数	strcpy(char*,char *)	后者拷贝到前者
字符串追加函数	strcat(char*,char *)	后者追加到前者后，返回前者，因此前者空间要足够大
字符串比较函数	strcmp(char*,char *)	前者等于、小于、大于后者时，返回 0、正值、负值。注意，不是比较长度，是比较字符 ASCII 码的大小，可用于按姓名字母排序等。
字符串长度	strlen(char *)	返回字符串的长度，不包括'\0'.转义字符算一个字符。
字符串型->整型	atoi(char *)	
整型->字符串型	itoa(int,char *,int)	做课设时挺有用的
	sprintf(char *,格式化输入）	赋给字符串，而不打印出来。课设时用也比较方便

注：对字符串是不允许做==或！=的运算的，只能用字符串比较函数

指针：

指针可以说是 C 语言中最关键的地方了，其实这个“指针”的名字对于这个概念的理解是十分形象的。首先要知道，指针变量的值（即指针变量中存放的值）是指针（即地址）。指针变量定义形式中：基本类型 *指针变量名 中的“*”代表的是

这是一个指向该基本类型的指针变量，而不是内容的意思。在以后使用的时候，如*ptr=a 时，“*”才表示 ptr 所指向的地址里放的内容是 a。

指针比较典型又简单的一应用例子是两数互换，看下面的程序，

```
swap(int c,int d)
{
int t;
t=c;
c=d;
d=t;
}
main()
{
int a=2,b=3;
swap(a,b);
printf(“%d,%d”,a,b);
}
```

这是不能实现 a 和 b 的数值互换的，实际上只是形参在这个函数中换来换去，对实参没什么影响。现在，用指针类型的数据做为参数的话，更改如下：

```
swap(#3333FF *p1,int *p2)
{
int t;
t=*p1;
*p1=*p2;
*p2=t;
}
main()
```

```
{
int  a=2,b=3;
int  *ptr1,*ptr2;
ptr1=&a;
ptr2=&b;
swap(prt1,ptr2);
printf("%d,%d",a,b);
}
```

这样在 swap 中就把 p1,p2 的内容给换了，即把 a，b 的值互换了。

指针可以执行**增、减运算**，结合++运算符的法则，我们可以看到:

***+** **+s**	取指针变量加 1 以后的内容
***s** **++**	取指针变量所指内容后 s 再加 1
(* **s)** **++**	指针变量指的内容加 1

指针和数组实际上几乎是一样的，数组名可以看成是一个常量指针，一维数组中 ptr=&b[0]则下面的表示法是等价的：

a[3]等价于*(a+3)

ptr[3]等价于*(ptr+3)

下面看一个用指针来自己实现 atoi（字符串型->整型）函数：

```
int  atoi(char  *s)
{
int  sign=1,m=0;
if(*s=='+'||*s=='-')  /*判断是否有符号*/
sign=(*s++=='+')?1:-1;  /*用到三目运算符*/
while(*s!='\0')  /*对每一个字符进行操作*/
  {
  m=m*10+(*s-'0');
  s++;  /*指向下一个字符*/
  }
return  m*sign;
}
```

指向多维数组的指针变量也是一个比较广泛的运用。例如数组 a[3][4]，a 代表的实际是整个二维数组的首地址，即第 0 行的首地址，也就是一个指针变量。而 a+1 就不是简单的在数值上加上 1 了，它代表的不是 a[0][1]，而是第 1 行的首地址，&a[1][0]。

指针变量常用的用途还有把指针作为参数传递给其他函数，即**指向函数的指针**。

看下面的几行代码：

```
void  Input(ST  *);
void  Output(ST  *);
void  Bubble(ST  *);
void  Find(ST  *);
```

```
void  Failure(ST  *);

/*函数声明：这五个函数都是以一个指向 ST 型（事先定义过）结构的指针变量作为参数，无返回值。*/


void  (*process[5])(ST  *)={Input,Output,Bubble,Find,Failure};

/*process 被调用时提供 5 种功能不同的函数共选择(指向函数的指针数组）*/


printf("\nChoose:\n?");

scanf("%d",&choice);

if(choice>=0&&choice<=4)

(*process[choice])(a);  /*调用相应的函数实现不同功能*;/
```

总之，指针的应用是非常灵活和广泛的，不是三言两语能说完的，上面几个小例子只是个引子，实际编程中，会逐渐发现运用指针所能带来的便利和高效率。

文件：

函数调用形式	**说明**
fopen("路径","打开方式")	打开文件
fclose(FILE *)	防止之后被误用
fgetc(FILE *)	从文件中读取一个字符
fputc(ch,FILE *)	把 ch 代表的字符写入这个文件里

fgets(FILE *)	从文件中读取一行
fputs(FILE *)	把一行写入文件中
fprintf(FILE *,"格式字符串",输出表列）	把数据写入文件
fscanf(FILE *,"格式字符串",输入表列）	从文件中读取
fwrite（地址，sizeof（），n，FILE *）	把地址中 n 个 sizeof 大的数据写入文件里
fread（地址，sizeof（），n，FILE *）	把文件中 n 个 sizeof 大的数据读到地址里
rewind（FILE *）	把文件指针拨回到文件头
fseek（FILE *，x，0/1/2）	移动文件指针。第二个参数是位移量，0 代表从头移，1 代表从当前位置移，2 代表从文件尾移。
feof(FILE *)	判断是否到了文件末尾

文件打开方式	说明
r	打开只能读的文件
w	建立供写入的文件，如果已存在就抹去原有数据
a	打开或建立一个把数据追加到文件尾的文件
r+	打开用于更新数据的文件
w+	建立用于更新数据的文件，如果已存在就抹去原有数据
a+	打开或建立用于更新数据的文件，数据追加到文件尾

注：以上用于文本文件的操作，如果是二进制文件就在上述字母后加“b”。

我们用文件最大的目的就是能让数据保存下来。因此在要用文件中数据的时候，就是要把数据读到一个结构（一般保存数据多用结构，便于管理）中去，再对结构进行操作即可。例如，文件 aa.data 中存储的是 30 个学生的成绩等信息，要遍历这些信息，对其进行成绩输出、排序、查找等工作时，我们就把这些信息先读入到一个结构数组中，再对这个数组进行操作。如下例：

```
#include<stdio.h>

#include<stdlib.h>

#define  N  30
```

```
typedef struct student /*定义储存学生成绩信息的数组*/
{
char *name;
int chinese;
int maths;
int phy;
int total;
}ST;

main()
{
ST a[N]; /*存储 N 个学生信息的数组*/
FILE *fp;
void (*process[3])(ST *)={Output,Bubble,Find}; /*实现相关功能的三个函数*/
int choice,i=0;
Show();
printf("\nChoose:\n?");
scanf("%d",&choice);
while(choice>=0&&choice<=2)
  {
  fp=fopen("aa.dat","rb");
  for(i=0;i<N;i++)
    fread(&a[i],sizeof(ST),1,fp); /*把文件中储存的信息逐个读到数组中去*/
  fclose(fp);
  (*process[choice])(a); /*前面提到的指向函数的指针，选择操作*/
  printf("\n");
  Show();
  printf("\n?");
  scanf("%d",&choice);
```

```
  }
}

void Show()
{
printf("\n****Choices:****\n0.Display the data form\n1.Bubble it according to the total score\n2.Search\n3.
Quit!\n");
}

void Output(ST *a) /*将文件中存储的信息逐个输出*/
{
int i,t=0;
printf("Name Chinese Maths Physics Total\n");
for(i=0;i<N;i++)
  {
  t=a[i].chinese+a[i].maths+a[i].phy;
  a[i].total=t;
  printf("%4s%8d%8d%8d%8d\n",a[i].name,a[i].chinese,a[i].maths,a[i].phy,a[i].total);
  }
}

void Bubble(ST *a) /*对数组进行排序，并输出结果*/
{
int i,pass;
ST m;
for(pass=0;pass<N-1;pass++)
  for(i=0;i<N-1;i++)
    if(a[i].total<a[i+1].total)
      {
      m=a[i]; /*结构互换*/
```

```
        a[i]=a[i+1];
        a[i+1]=m;
        }
Output(a);
}

void Find(ST *a)
{
int i,t=1;
char m[20];
printf("\nEnter the name you want:");
scanf("%s",m);
for(i=0;i<N;i++)
  if(!strcmp(m,a[i].name)) /*根据姓名匹配情况输出查找结果*/
  {
  printf("\nThe result is:\n%s, Chinese:%d, Maths:%d,    Physics:%d,Total:%d\n",m,a[i].chinese,a[i].maths,a
[i].phy,a[i].total);
  t=0;
  }
if(t)
  printf("\nThe name is not in the list!\n");
}
```

链表：

链表是 C 语言中另外一个难点。牵扯到结点、动态分配空间等等。用结构作为链表的结点是非常适合的，例如：

```
struct node
{
int data;
```

```
struct node *next;
};
```

其中 next 是指向自身所在结构类型的指针，这样就可以把一个个结点相连，构成链表。

链表结构的一大优势就是动态分配存储，不会像数组一样必须在定义时确定大小，造成不必要的浪费。用 malloc 和 free 函数即可实现开辟和释放存储单元。其中，malloc 的参数多用 sizeof 运算符计算得到。

链表的基本操作有：**正、反向建立链表；输出链表；删除链表中结点；在链表中插入结点**等等，都是要熟练掌握的，初学者通过**画图**的方式能比较形象地理解建立、插入等实现的过程。

```
typedef struct node
{
char data;
struct node *next;
}NODE; /*结点*/

正向建立链表：
NODE *create()
{
char ch='a';
NODE *p,*h=NULL,*q=NULL;
while(ch<'z')
  {
  p=(NODE *)malloc(sizeof(NODE)); /*强制类型转换为指针*/
  p->data=ch;
  if(h==NULL) h=p;
```

```
    else  q->next=p;
  ch++;
  q=p;
  }
q->next=NULL;  /*链表结束*/
return  h;
}
```

逆向建立：

```
NODE  *create()
{
char  ch='a';
NODE  *p,*h=NULL;
while(ch<='z')
  {
  p=(NODE  *)malloc(sizeof(NODE));
  p->data=ch;
  p->next=h;  /*不断地把 head 往前挪*/
  h=p;
  ch++;
  }
return  h;
}
```

用递归实现链表逆序输出：

```
void output(NODE *h)
{
if(h!=NULL)
  {
  output(h->next);
  printf("%c",h->data);
  }
}
```

插入结点（已有升序的链表）：

```
NODE *insert(NODE *h,int x)
{
NODE *new,*front,*current=h;
while(current!=NULL&&(current->data<x)) /*查找插入的位置*/
  {
  front=current;
  current=current->next;
  }
new=(NODE *)malloc(sizeof(NODE));
new->data=x;
new->next=current;
if(current==h) /*判断是否是要插在表头*/
  h=new;
else front->next=new;
return h;
}
```

删除结点：

```
NODE  *delete(NODE  *h,int  x)
{
NODE  *q,*p=h;
while(p!=NULL&&(p->data!=x))
  {
  q=p;
  p=p->next;
  }
if(p->data==x)  /*找到了要删的结点*/
  {
  if(p==h)  /*判断是否要删表头*/
  h=h->next;
     else  q->next=p->next;
  free(p);  /*释放掉已删掉的结点*/
  }
return  h;
}
```

经常有链表相关的程序填空题，做这样的题要注意看下面提到的变量是否定义了，用到的变量是否赋初值了，是否有给分配空间的没有分配空间，最后看看返回值是否正确。

笔者水平有限，难免有疏漏、错误的地方，浅显之处，还望指正见谅。上述内容仅是个提示作用，并不包括 C 语言的全部内容