# C程序设计

# 1顺序结构

## 1. 1数据的表现形式及其运算

### 1.1.1常量和变量

（1）常量：

整形常量：

实型常量：十进制，指数形式

字符常量：普通字符，转义字符

字符串常量

符号常量：#define PI 3.1516

符号常量不占内存，只是一个临时符号，在预编译后这个符号就不存在了，故不能对符号常量赋予新值

（2）变量

必须先定义后使用

（3）常变量

Const float pi=3.1416

与符号常量区别：

符号常量是预编译指令，只是用符号常量代表一个字符串，在预编译时仅是进行字符替换，在预编译后，符号常量就不存在了

常变量占用存储空间，有变量值，只是该值不会改变

（4）标识符

字母数字下划线，第一个字符必须是字母或者下划线

### 1.1.2数据类型

注意没有字符串类型

### 1.1.3整型数据

基本整型int 2/4字节，

双长整型 8字节

Short<int<long<longlong

|  |  |  |
| --- | --- | --- |
| 类型 | 字节数 | 取值范围 |
| Int基本整型 | 4 | - |
| Unsigned int无符号基本整型 | 4 | 0～ |
| Short短整型 | 2 | - |
| Unsigned short无符号短整型 | 2 | 0～ |
| Long长整型 | 4 | -～ |
| Unsigned long无符号长整型 | 4 | 0～ |
| Long long双长型 | 8 | -～ |
| Unsigned long long | 8 | 0～ |

### 1.4字符型数据

字符‘1’与数字1不同，字符1以ascii形式存储，占用1字节，整数1以二进制存储，占用2或4个字节

字符型数据的存储空间与值得范围

|  |  |  |
| --- | --- | --- |
| 类型 | 字节数 | 取值范围 |
| Signed char有符号字符型 | 1 | -～ |
| Unsigned char无符号字符型 | 1 | 0- |

### 1.1.5浮点型数据

实数是以指数形式存放在存储单元中的

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 字节数 | 有效数字 | 数值范围 |
| Float | 4 | 6 | 0以及1.2\*～13.4\* |
| Double | 8 | 15 | 0以及2.3\*～1.7\* |
| Long double | 8 | 15 | 0以及2.3\*～1.7\* |
|  | 16 | 19 | 0以及3.4\*～1.1\* |

默认浮点型变量都是按照双精度double处理

Turbo c是16字节数

Visual c++是8字节数

### 1.1.6运算符和表达式

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 含义 | 举例 | 结果 |
| + | 正号运算符单目运算符运算符 | +a | A的值 |
| - | 负号运算符单目运算符 | -a | A的算数负值 |
| \* | 乘法运算符 | A\*b |  |
| / | 除法运算符 | a/b |  |
| % | 求余运算符 | A%b |  |
| + | 假发运算符 | A+b |  |
| - | 减法运算符 | a-b |  |

%：要求参加运算的是整数，结果也会i是整数

除%外都可以是任意类型

不同类型见混合运算

float+double——double

int+float/double——double

char——数字/double

%c——ascii形式

%d——十进制整数

强制类型转换：（类型名）（表达式）

c运算符

算数运算符：+ - \* / % ++ --

关系运算符：〉〈== >= <= !=

逻辑运算符：!&&||

位运算符：<< >> ~ | ^ &

赋值运算符 =

条件运算符？：

逗号运算符，

指针运算符\*&

求字节数运算符sizeof

强制类型转换运算符（（类型））

成员运算符 ->

## 1.2数据输入输出

### 1.2.1Printf(格式控制，输出表列)

格式控制

d格式符：十进制

c格式符：一个字符

s格式符：一个字符串

f格式符：基本型%f，指定宽度%m.nf，左对齐-%m.nf

e格式符：指数

若想要输出%，printf(“%f%%\n”,1.0/3)

### 1.2.2Scanf(格式控制，地址表列)

注意：格式控制之后，变量地址而不是变量名

Scanf(“%f”,&a)

如果在格式控制字符串中除了格式声明以外还有其他字符，则在输入数据对应位置上应输入与这些字符相同的字符

Scanf(“a=%f,b=%f”,&a,&b)

a=1,b=3

输入数据时，输入空格，回车，tab键或遇到非法字符认为结束

### 1.2.3字符数据的输入输出

putchar输出一个字符

getchar输入一个字符

# 2选择结构

关系运算符及优先次序

算数运算符(+-\*/)>关系运算符(> < <= >=)>赋值运算符(== !=)

5>3&&8<4-!0------------------------------(5>3)&&8<(4-(!0))----------------------------(5>3)&&8<(4-1)

(Java 不可以写!0)

Switch表达式里整数类型（包括字符型）

Case加常量（或者常量表达式）

最后一个case（今为default）可不加break

# 3循环结构

for里面三个表达式都可以省略

while/do。。while/for都可用break和continue区别

break：跳出循环

continue：结束本次循环

# 4数组

数组特点：

一组有序数据的集合，每个元素都属于同一个数据类型

## 4.1 一维数组

类型符 数组名[常量表达式]

注意下标从0开始，定义10个元素不存在a[10]

Int a[n]不合法，c语言不允许对数组的大小做动态定义

如果指定数组为静态存储static，则不能用可变长数组

Static int a[2\*n] //不合法，a数组指定为statci存储方式

Int a[10]={1，2，3，4，5}

未被初始化的数组元素会自动初始化为0，如果是字符型数组，则初始化为‘\0’，如果是指针数组，则初始化为null即空指针

排序冒泡法

|  |
| --- |
| #include<stdio.h>  int main(){  int a[10];  int i,j,t;  printf(“input 10 number:\n”);  for (i=9;i<10;i++){  scanf(“%d”,&a[i]);  }  printf(“\n”);  for(j=0;j<n;j++){  for(i=0;i<9-j;i++){  if(a[i]>a[i+1]){  t=a[i];a[i]=a[i+1];a[i+1]=t;  }  }  }  printf(“the sorted number:\n”);  for(i=0;i<10;i++){  printf(“%d”,a[i]);  }  printf(“\n”);  return 0;  } |

## 4.2 二维数组

类型说明符 数组名[常量表达式][常量表达式]

二维数组排列顺序是按行存放的，即内存中先顺序存放第一行的元素，接着存放第二行的元素

|  |  |
| --- | --- |
| 2000 | a[0][0] |
| 2004 | a[0][1] |
| 2008 | a[1][0] |
| 2012 | a[1][1] |

典型错误

|  |
| --- |
| Int a[3][4]  a[3][4]=3 //错误，不存在这个元素 |

二维数组初始化时，可以省略第一维的长度，但第二维不能省略

Int a[][4]={{0，0，3}，{}，{0，10}}

存放下来的长这样

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | 3 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 10 | 0 | 0 |

## 4.3字符数组

### 4.3.1定义

c语言中没有字符串类型，字符串是放在字符数组中的

char c[10]={‘i’,’a’,’m’,’,’,’h’,’a’,’p’,’p’,’y’}

char diamond[5][5]={{‘’,’’,’\*’},{‘’,’\*’,’’,’\*’},{‘\*’,’’,’’,’\*’},{‘’,’\*’,’’,’\*’},{‘’,’’,’\*’}}

\*

\* \*

\* \*

\* \*

\*

c语言在存储字符串常量会自动加’\0’作为结束符，程序中往往判断是否有这个标识代表结束，而不是根据数组长度来决定字符串长度

char c={“I am happy”}

数组c长度不是10而是11，因为最后一位存储’\0’

Char c[10]={“china”}

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | H | i | N | a | \0 | \0 | \0 | \0 | \0 |

如果存储了以下的字符数组

Char c[]={“c program。”’}

若想要用新字符串代替老字符串，从键盘输入hello，替代，如果不加’\0’字符数组中字符如下

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| H | E | L | L | O | G | R | A | m | 。 | \0 |

hellogram。

如果在hello后面加\0就取代第六个字符g在数组中

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| H | E | L | L | O | \0 | R | A | m | 。 | \0 |

输出hello

多个’\0’，遇到第一个就结束输出

### 4.3.2字符数组输入输出

%c输入输出一个字符

%s整个字符串一次输入或输出

printf函数中，%s输出的字符数组名，而不是数组元素名

printf(“%s”,c[0]) //错误

printf(“%s”,c) //正确

scanf输入一个字符串

scanf输入项如果是 字符数组名，不需要加&地址符，因为数组名代表数组的起始地址

|  |
| --- |
| char str1[5],str2[5],str3[5]  scanf(“%s%s%s”,str1,str2,str3)  输入数据 how are you  结果，三个数组分别how，are，you |
| char str[13]  scanf(“%s”,str)  输入元素how are you  结果how后面遇到空格代表结束输入，输出元素how |

### 4.3.3使用字符串处理函数

#### puts函数——输出字符串的函数

puts(字符数组)

#### gets函数——输入字符串函数

注意gets(str)，键盘输入computer回车键

数组中存储有9个字符，而不是8个

‘\0’

Getchar 存的是’\n’

#### strcat函数——字符串连接函数

strcat(字符数组1，字符数组2)

作用是把两个字符数组中的字符串连接起来，把字符串2接到字符串1的后面，结果放在字符数组中，函数调用后得到一个数值——字符数组1值得地址

注意：

字符串1必须足够大，以便能容纳连接后面的新字符串

连接前两个字符串的后面都有’\0’,连接时，字符串1后面的’\0’取消，只在最后保留’\0’

#### Strcpy 和strncpy函数——字符串复制函数

Strcpy(字符数组1，字符数组2)

将字符数组2复制到字符数组1种

注意：字符数组1必须足够大

字符数组1必须写成数组名形式，字符串2可以是字符数组名，也可使是一个字符串常量

Strcpy(str,‘china’)

如果在复制前未对str1数组初始化或者赋值，则str1各字节中的内容是无法预知的，复制时将str2中的字符串和其后面’\0’一起复制到字符数组1种，取代字符数组1种的前面6个字符，最后4个字符并不一定是’\0’，而是str1中后4个元素

不能直接通过赋值形式，将一个字符串常量或者字符数组直接给一个字符数组

Str1=”china”

St1=str2

只能用strcpy函数将一个字符串复制到另一个字符数组中，用复制语句只能将一个字符复制给一个字符型变量或字符数组元素

Char [5],c1,c2

C1=’a’,c2=’b’

A[0]=’c’,a[1]=’j’,a[2=[i]

可以用strncpy函数将字符串2种的前面n个字符复制到字符数组1中去

#### Strncpy(str1,str2,n)

作用是将str2种最前面的n个字符，取代str1中原有的前面的n个字符，但复制的字符个数n不应多余str1中原有的字符（不包括’\0’）

#### strcmp函数——字符串比较函数

strcmp(字符串1，字符串2)

比较规则：将两个字符串自左到右逐个字符比较按照ascii值大小比较，直到出现不同的字符或者遇到’\0’为止

如果全部字符相同，则认为两个字符相同str1=str2，0

若出现不同的字符，则以第1对不同的字符比较结果为准

str1>str2 函数值为正数

str1<str2函数值为负数

#### strlen——测字符串长度的函数

#### strlwr——转为小写的函数

#### strupr——转为大写的函数

这些函数都需要加入头文件#include <string.h>

例子：输入一行字符，统计其中有多少个单词，单词之间用空格间隔开

|  |
| --- |
| #include<stdio.h>  int main(){  char string[81]={"i am boy"};  int i,num=0,word=0;  char c;  for (i=0;(c=string[i])!='\0';i++){  if(c ==' ') word=0;  else {  if(word ==0){  word=1;  num++;  }  }  }  printf("there are %d words in this line\n",num);  return 0;  } |

# 5函数

## 5.1函数定义

定义无参

类型名 函数名（）

{

函数体

}

定义有参

类型名 函数名（形参）

{

函数体

}

定义空函数

类型名 函数名（）

## 5.2函数调用

函数名 （实参列表）

函数调用过程中实参形参传递

实参可以是常量，变量，或者表达式

实参与形参的类型应相同或者赋值兼容，例如

C=max(a,b)

Int max(int x,int y){

Int z;

Z=x>y?x:y;

Return z

}

如果实参为int型，而形参为float，则按照不同类型数值的赋值规则转换，如果a为float值为3.5 ，形参x为int，则在传递时先将实数3.5转成整数3，然后传给形参x

return语句返回值类型应与函数类型一致，如果不一致，则以函数返回为主，对于数值型的数据，可以自动进行类型转换，即函数类型决定返回值的类型

形参未出现函数调用时，不占用内存单元，函数调用被临时分配内存单元

调用结束时，形参单元被释放，但实参单元仍保留并维持着原来的值没有改变

例如在执行max函数中，x和y的值变为10和15，a和b仍旧为2和3，因为形参和实参是两个不同的存储单元

注意：实参向形参传递为值传递，单向传递，只能由实参传递给形参，不能由形参传递给实参，形参实参占用不同单元，实参不能得到形参的值

函数返回值

return语句返回值类型应与函数类型一致，如果不一致，则以函数返回为主，对于数值型的数据，可以自动进行类型转换，即函数类型决定返回值的类型

|  |
| --- |
| 如果函数返回值类型与指定的函数类型不同，按照赋值规则处理  #include<stdio.h>  int main(){  int max(float x,float y);  float a=1.6,b=2.6;  float c;  c=(float)max(a,b);  printf("max is %f \n",c);  return 0;  }  int max(float x,float y){  float z;  z=x>y?x:y;  return z;  } |

Max函数形参是float型，实参也是float，今传入a=1.6和b=2.6，调用max函数，出现矛盾z值为2.6，函数定义为int型，return语句z为float型

处理方式：按赋值规则处理，先将z的值转换成为int型，得到2，就是函数得到的返回值。最后max(x,y)带回一个整型值2返回主调函数main

如果将main函数中c改成float型，用%f输出，输出2.000000.因为调用max函数得到的是int型，函数值为整数2

## 5.3函数的声明和函数原型

main函数位置在add函数前面，因为执行编译时自上而下，如果没有对函数add的声明，当编译到flaot add时无法确定是不是函数名，也无法判断实参和个数是否正确，无法做出正确性检验

函数声明简单照着写函数定义首行加分号

定义和声明不是一回事：

定义是针对函数功能的确立，包括指定函数名，函数值类型，形参以及其类型以及函数体是一个完整的，独立的函数单位

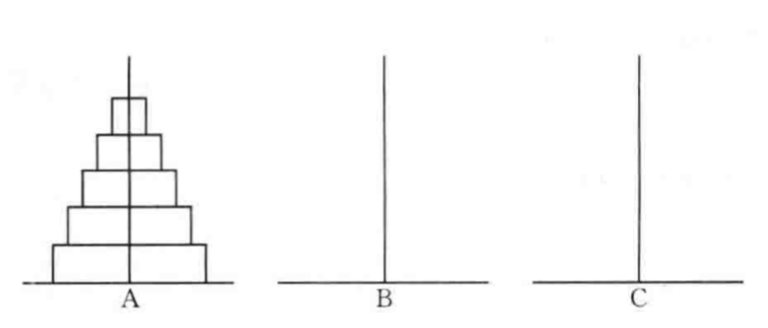
声明：则是把函数的名字，函数类型，以及形参的类型，个数和顺序通知编译系统，以便在调用该函数时系统按此进行对照检查（例如函数名是否正确，实参形参类型个数是否一致）不包含函数体

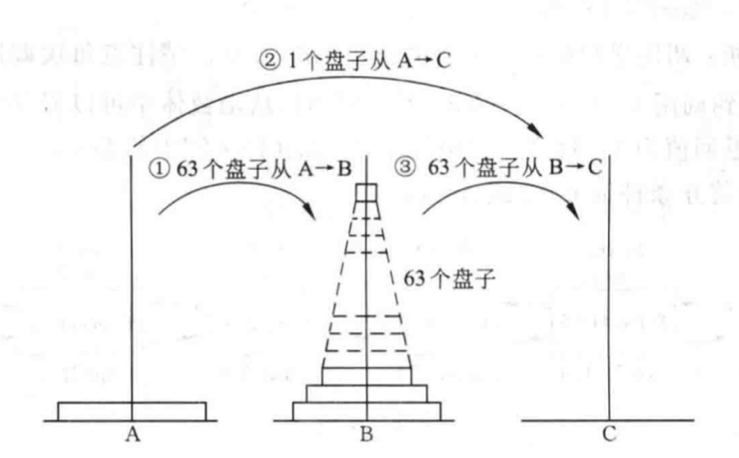
## 5.4函数嵌套使用

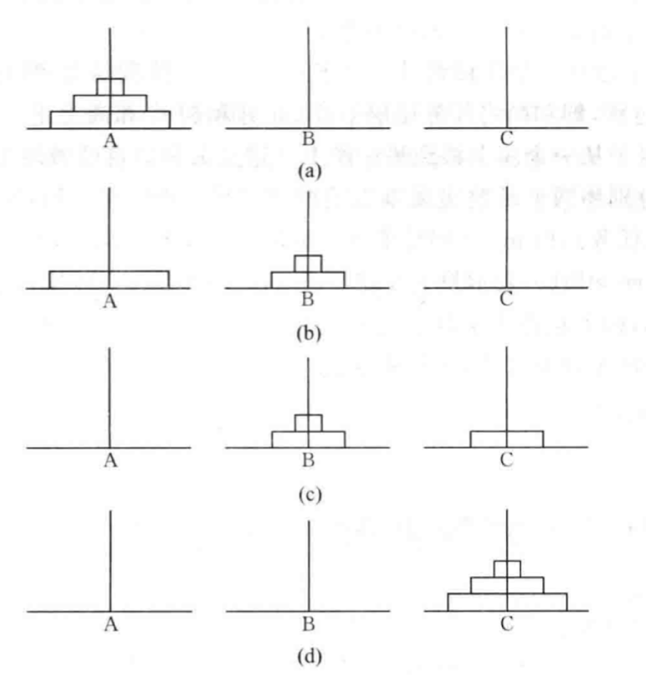
## 5.5函数递归

经典问题：汉诺塔问题

例题：汉诺塔问题，这是一个古典的数学问题，用一个用递归方法解题的典型例子。问题是这样的：古代有个梵塔，塔内有3个座，a，b，c。开始时a座上有64个盘子，盘子大小不等，大的在下，小的在上，有一个老和尚想把这64个盘子从a座移到c座，但规定每次只允许移动一个盘，且在移动过程中在3个座上都始终保持大盘在下，小盘在上，在移动过程中可以利用b座。要求编程序移动盘子的步骤







|  |
| --- |
| #include<stdio.h>  int main(){  void hanoi(int n,char one,char two,char three);  int m;  printf(“input the number of diskes:”)  scanf(“%d”,&m)  printf(“ the step to move %d diskes:\n”.m)  hanoi(m,’a’,’b’,’c’)    }  void hanoi(int n ,char one,char two,char three){  //将n个盘从one座，移到two座，移到three座  void move(char x,char y);  if(n==1){  move(one,three)  }else{  hanoi(n-1,one,three,two)  move(one,three)  hanoi(n-1,two,one,three)    }  }  void move(char x,char y){  printf(“%c->%c\n”,x,y)  }  } |

## 5.6数组作为函数参数

### 5.6.1数组元素作函数实参

数组元素可以做函数实参，不能做形参

因为形参是在函数被调用时临时分配存储单元的，不可能为一个数组元素单独分配存储单元（数组是一个整体，在内存中占连续的一段存储空间），在用数组元素作函数实参时，把实参的值传给形参，是值传递的方式，数据传递方向是从实参道形参，单向传递

例输入10个数，要求输出其中最大的元素和该数是第几个数

|  |
| --- |
| #include<stdio.h>  int main(){  int max(int x, int y);  int a[10],m,n,i;  printf(“enter 10 integer numbers:”)  for(i=0;i<10,i++){  scanf(“%d”,&a[i])  }  printf(“\n”)  for(i=1,m=a[0],n=0;i<10;i++){  if(max(m,a[i])>m){  m=max(m,a[i])  n=i; //把此数组元素的序号记下来，放在n中  }  }  printf(“the largest number is%d\n it is the %d th number\n”,m,n+1)  }  int max(int x,int y){  return x>y?x:y  } |

### 5.6.2数组名作为函数参数

用数组元素作实参，向形参变量传递的是数组元素的值，而用数组名作函数实参时，向形参（数组名或指针变量）传递的是数组首元素的地址

例有一个一维数组score ，内放10个学生成绩，求平均

|  |
| --- |
| #include<stdio.h>  int main(){  float average(float array[10])  float score[10],aver;  int i;  printf(“enter 10 integer numbers:”)  for(i=0;i<10,i++){  scanf(“%d”,&score[i])  }  aver =average(score);  printf(“average score is %5.2f\n”,aver)  }  float average(float array[10]){  float sum = 0;  for (i=1;i<10;i++){  sum=sum+a[i]  }  aver = sum/10  return aver  } |

注意：用数组名作函数实参时，不是把数组元素的值传递给形参，而是把实参数组的首元素地址传递个形参数组，这样两个数组就共占同一段内存单元，形参数组中铬元素的值如果发生改变会使实参数组元素的值同时发生变化。

例：用选择法对数组中10个整数从小到达排序

|  |
| --- |
| #include<stdio.h>  int main(){  void sort(int array[],int n);  int a[10],i;  printf(“enter array:\n”)  for(i=0;i<10,i++){  scanf(“%d”,&score[i])  }  sort(a,10);  for(i=0;i<10,i++){  printf(a[i])  }  }  void sort(int array[],int n){  int max = 0;  for (i=1;i<n;i++){  k=i;  for(j=i+1;<n,j++){  if(array[j]<array[k]){  k=j;  }  }  t=array[k];array[k]=array[j];array[j]=t;  }  } |

指针长/数组长

|  |
| --- |
| #include <stdio.h>  int max(int a[]){//a相当于一个指针  printf("---------\n");  printf("%d\n",sizeof(a)/sizeof(a[0]));  for (int i = 0;i<8;i++){  printf("%d ",a[i]);  }  return 1;  }  int main(){  int c;  int a[] = {4,3,2,1,2,3,4,5};  printf("%d\n",(sizeof(a)/sizeof(a[0])));  c=max(a);  return 0;  } |

## 5.7局部变量和全局变量

定义变量可能的3种情况

在函数的开头定义——局部变量

在函数内的复合语句内定义——局部变量

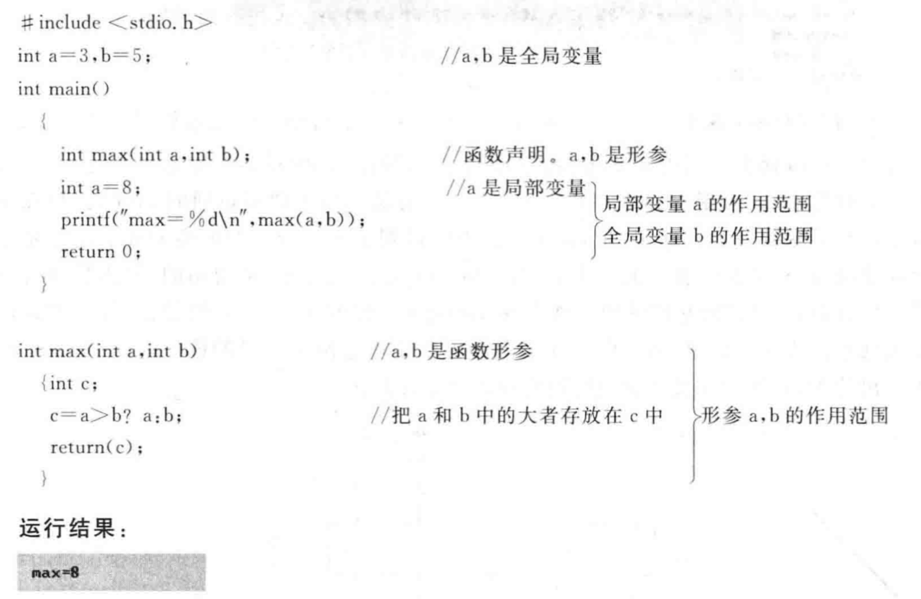
在函数的外部定义——全局变量

注意：主函数中定义的的变量，也只在主函数中有效，并不因为在主函数中定义而在真个文件或程序中有效

例题：有一个一维数组，内放10个学生成绩，写一个函数，当主函数调用此函数后，能求出平均分，最高分和最低分

|  |
| --- |
| #include<stdio.h>  float max=0;min=0;  int main(){  float average(float array[],int n);  float ave,score[10];  int i;  printf(“please enter 10 socres:”);  for(int i=0;i<10;i++){  scanf(“%f”,&score[i]);  }  aver = average(score,10);  printf(“max=6.2%f\n,min=6.2%f\naverage=%6.2f\n”,max,min,aver);  }  flaot aver(float array[],int n){  int i;  float aver,sum=array[0];  for (i=1;i<n;i++){  if(array[i]>max){  max=array[i];  }else if (array{i}<min){  min=array[i];  }  sum=sum+araay[i];  }  aver = sum/n;  return (aver)  } |

例：若外部变量和局部变量同名，分析结果



## 5.8变量的存储方式和生存期（易混淆）

### 5.8.1动态存储方式和静态存储方式

从变量作用域分为，全局变量和局部变量

从变量过的存在时间分为静态存储方式和动态存储方式

静态存储方式：在程序运行期间由系统分配的固定的存储空间的方式

动态存储方式：在程序运行期间根据需要进行动态的分配存储空间的方式

存储区分为

程序区：

静态存储区：

全局变量，在程序开始执行时为全局变量分配存储区，结束释放，占用固定的存储单元

动态存储区：

函数形式参数：在调用函数给形参分配空间

函数中定义的没有static关键字声明的变量，即自动变量

函数调用时的现场保护和返回地址

C语言中每一个变量和函数都有两个属性，数据类型和数据的存储类别。数据类型指整型，浮点型，存储类别指数据在内存中的存储方式：静态存储动态存储

C的存储类别：自动的auto，静态的static，寄存器的register，外部的extern

例子 回收，函数中加入static变量，返回存在

|  |
| --- |
| #include <stdio.h>  int\* func(){  int static aa[]={1,2,3,4,};;  return aa;  }  int main(void) {  int \*a = func();  printf("line\n");  printf("%d\n",a);  return 0;  } |

### 5.8.2局部变量的存储类别

（1）自动变量auto

如果不是声明static都是动态分配存储空间的，数据存储在动态存储区中，函数形参和函数中定义的局部变量都属于此类。在调用该函数时，系统会给这些变量分配存储空间，在函数调用结束时，自动释放

Auto可以省略，隐含指定为自动存储类别

（2）static静态局部变量

局部变量值调用结束后仍旧存在

例题考察静态局部变量的值

|  |
| --- |
| #include<stdio.h>  int main(){  int f(int );//函数声明  int a=2,i;//自动局部变量  for(int i=0;i<3;i++){  printf("%d\n",f(a));//输出f（a）的值  }  return 0;  }  int f(int a){  auto int b=0;//自动局部变量  static c=3;//静态局部变量  b=b+1;  c=c+1;  return(a+b+c);  } |

输出

|  |
| --- |
| 7  8  9 |

静态局部变量属于静态存储类别，在静态存储去分配存储单元。程序整个运行期间不释放，自动变量属于动态存储类别，分配在动态存储区空间，函数调用结束释放

对静态局部变量是在编译时赋初值，只赋值一次，再次调用函数时保留上次调用结束的值

虽然静态局部函数在调用结束后仍然存在，但只能呢个本函数引用，不能被其他函数引用

例题：输出1到5的阶乘值

|  |
| --- |
| #include<stdio.h>  int main(){  int fac(int );  int i;  for(int i=0;i<=5;i++){  printf(“%d！=%d\n”,f(a));  }  return 0;  }  int fac(int a){  static int f=1;  f=f\*n  return(f);  } |

（3）寄存器变量register变量

使用频繁的，直接从寄存器取出参加运算，不倒内存中去取

Register int i；

### 5.8.3全局变量的存储类别

全局变量都是存放在静态存储区中的

（1）在一个文件中扩展外部变量的作用域

例题：调用函数，求三个整数中最大者

|  |
| --- |
| #include<stdio.h>  int main(){  int max( );  extern int a,b,c;//把外部变量a，b，c的作用域扩展到从此处开始  printf("max is %d\n",max());  return 0;  }  int a=2,b=3,c=4;  int max(){  int m;  m=a>b?a:b;  if(c>m) m=c;  return (m);  } |

由于定义的外部变量的a，b，c的位置在函数main以后，本来main中是不能饮用外部变量的，现在加入extern对变量进行声明，把a，b，c的作用域扩展到该位置。

由于a，b，c是外部变量，调用max函数用不到参数传递，在max函数中可直接使用外部变量a，b，c的值

提倡将外部变量的定义放在引用他的所有函数之前，这样可以避免函数中多加一个extern声明

（2）将外部变量的作用域扩展到其他文件

例题：给定b的值，输入a和m求a\*b和a的m次幂

|  |
| --- |
| 文件file1.c  #include<stdio.h>  int a；  int main(){  int power( int);  int b=3,c,d,m;  printf(“enter the number a and its power m:\n”);  scanf(“%d%d”,&a,&m);  c=a\*b;  printf(“%d\*%d=%d\n”,&a,&b,&c)  d=power(m)  scanf(“%d%d%d”,&a,&b,&c);  printf(“%d\*\*%d=%d\n”,&a,&m,&d)  return 0;  }  文件file2.c  extern a;  int power(int n){  int i,y=1;  for(i=1;i<=n;i++){  y\*=a;  }  return (y);  } |

注意：extern既可以扩展全局变量在本文件中的作用域，又可以使外部变量的作用域从一个文件扩展到程序中的其他文件，那么系统中怎么区别处理

编译时遇到extern，现在本文件中找外部变量的定义，如果找到，就在本文件中扩展作用域，如果找不到，就在连接时从其他文件中找外部变量的定义，如果从其他文件中找到了，就将作用域扩展到本文件，如果找不到，按照出错处理

（3）将外部变量的作用域限制在本文件中

Static 声明只能用于本文件的外部变量称为静态外部变量

不要误认为外部变量加static才采取静态存储方式，不加static是动态存储方式

声明局部变量的存储类型和声明全局变量的存储类型的含义是不同的

局部变量：声明存储类型的作用是指定变量存储的区域（静态存储区或动态存储区）以及由此产生的生存期的问题

全局变量：在编译时分配内存的，存放在静态存储区中，声明存储类型作用是变量作用域的扩展问题

用static声明一个变量的作用

对局部变量用static声明，分配在静态存储区，该变量执行期间不释放，始终存在

对全局变量static，该变量的作用域仅限于本文件模块，即被声明的文件中

用auto，register，static声明时，都是在定义变量基础上加这些字，不能单独使用

Int a

Static int a

Static a 错误

### 5.8.4存储类别小结

对数据定义，需要指定数据类型，存储类别

Static int a//静态局部变量或者静态外部变量

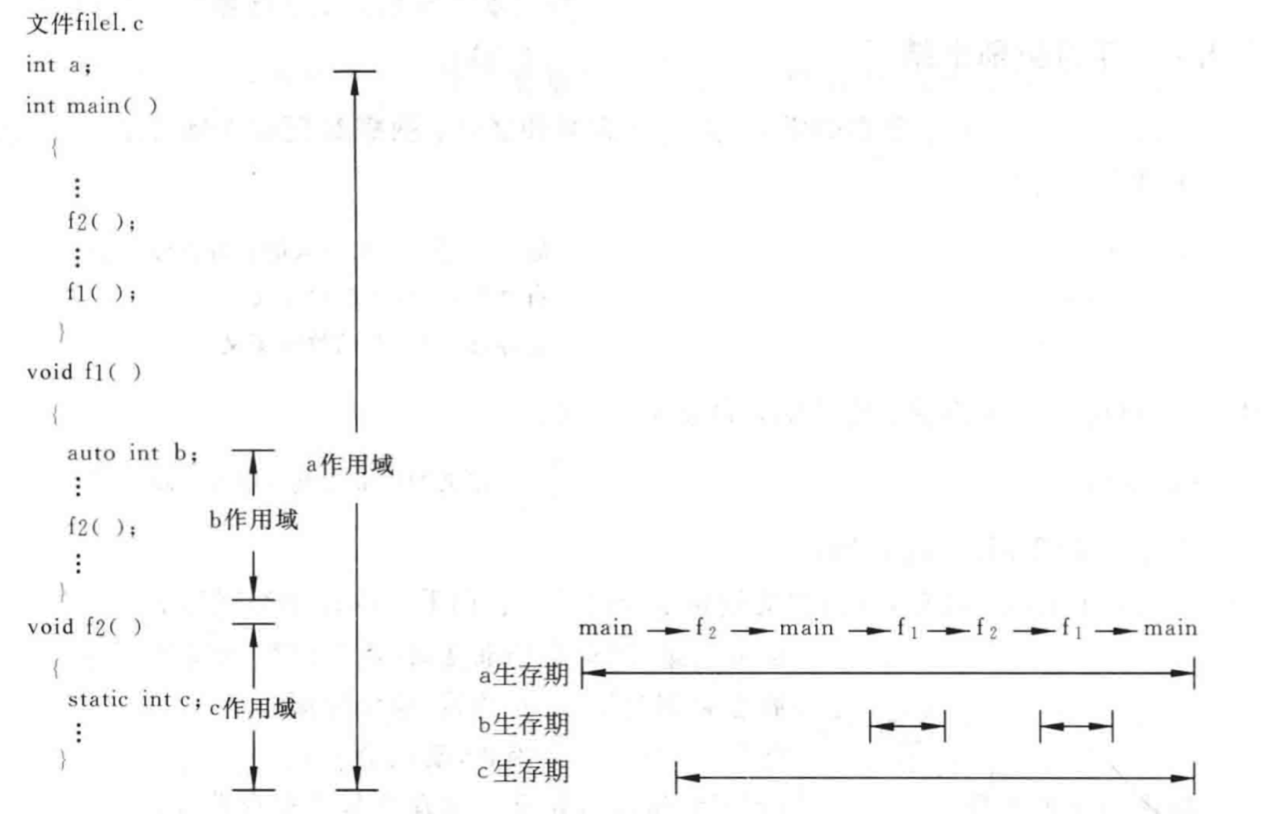
Auto char c//自动变量，函数内定义

Register int d//寄存器变量，函数内定义

此外用extern声明已定义的外部变量

Extern b//

1. 从作用域角度，局部变量和全局变量
2. 从生存周期区分
3. 从变量存放位置区分
4. 关于作用域和生存期的概念



各种类型变量的作用域和存在性情况

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 变量存储类别 | 函数内 |  | 函数外 |  |
|  | 作用域 | 存在性 | 作用域 | 存在性 |
| 自动变量和寄存器变量 | 对 | 对 | ❌ | ❌ |
| 静态局部变量 | 对 | 对 | ❌ | 对 |
| 静态外部变量 | 对 | 对 | 对仅限本文件 | 对 |
| 外部变量 | 对 | 对 | 对 | 对 |

Static对局部变量使变量由动态存储方式改为静态存储方式

Static对全局变量来讲，使变量局部化（局部于本文件），但仍为静态存储方式

从作用域角度看，凡是有static声明的，其作用域都是局限的，或者局限本函数内（静态局部变量），或者局限于本文件内（静态外部变量）

## 5.9变量声明和定义

建立存储空间的声明称定义

不建立存储空间的声明为声明

简单方法，在函数中出现对变量的声明，除了用extern声明意外，都是定义，在函数

|  |
| --- |
| #include<stdio.h>  int main(){  extern a;//是声明不是定义，声明将已定义的外部变量a的作用域扩展到此  }  int a；//是定义，定义a为整型外部变量 |

## 5.10内部函数和外部函数

内部函数：一个函数智能被本文件中其他函数所引用，定义函数时

Static 类型名 函数名（形参表）

外部函数：可供其他文件调用

Extern int fun(int a,int b)

例题：有一个字符串，内有若干个字符，现输入一个字符，要求程序将字符串中该字符删去，用外部函数实现

|  |
| --- |
| #include<stdio.h>  int main(){  extern void enter\_string(char str[]);//对函数声明  extern void delete\_string(char str[],char ch);//对函数声明  extern void print\_string(char str[]);//对函数声明  //以上三行声明了在本函数  char c，str[80];  enter\_string(str);  scanf(“%c”,&c);  delete\_string(str,c);  printf\_string(str);  return 0;  }  文件file2.c  void enter\_string(int n){//定义外部函数enter\_string  gets(str);//向字符数组输入字符串  }  文件file3.c  void delete\_string(char str[],char ch){  int i,j;  for(i=i=0;str[i]!=‘\0’,i++){  if(str[i]!=ch){  str[j++]=str[i]    }  str[j]!=’\0’  }  }  文件file4.c  void printf\_string(char str[]){  printf(“%s\n”,str);  } |

# 6指针

## 6.1指针是什么

地址指向该变量单元

区分存储单元的地址和存储单元的内容

\*i\_pointer

一个变量的地址称为指针

例如地址2000时变量i的指针

指针变量：一个变量专门用来存放另一变量的地址（即指针）

指针变量就是地址变量，用来存放地址，指针变量的值是地址

i\_pointer为指针变量

## 6.2指针变量

### 6.2.1定义指针变量

类型名 \*指针变量名

Int \* i\_pointer =&a

或者

int \* i\_pointer

i\_pointer =&a

（1）\*表示该变量的类型为指针型变量，指针变量名是i\_pointer，

（2）不能写成\* i\_pointer =&a，因为a的地址是赋值给指针变量i\_pointer，不是付给\* i\_pointer

定义

（3）定义指针变量时必须指定基类型

（4）一个变量的指针的含义包含：一是以存储单元编号表示的地址，一是窒息那个存储单元的数据类型（int，char，float等）

（5）int \* 表示为指向int的指针

（6）指针变量中只能存放地址（指针），不要将一个整数赋值一个指针变量

\* i\_pointer=100 错误

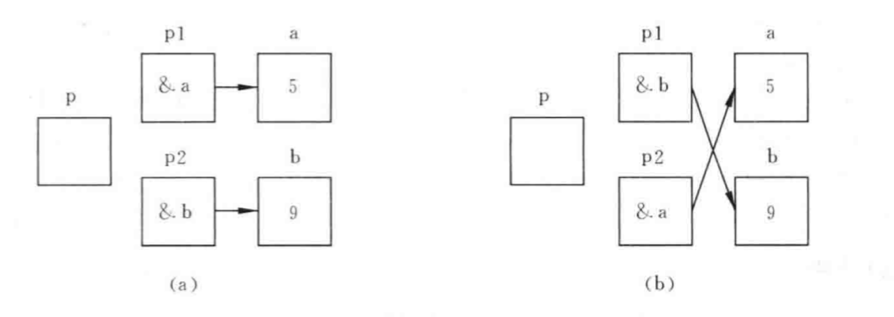
### 6.2.2怎样引用指针变量

&取地址符，&a是变量a的地址

\*指针运算符，或称间接访问运算符，\*p代表指针变量p指向的对象

例题6.1输入a和b两个整数，按先大后小的顺序输出a和b

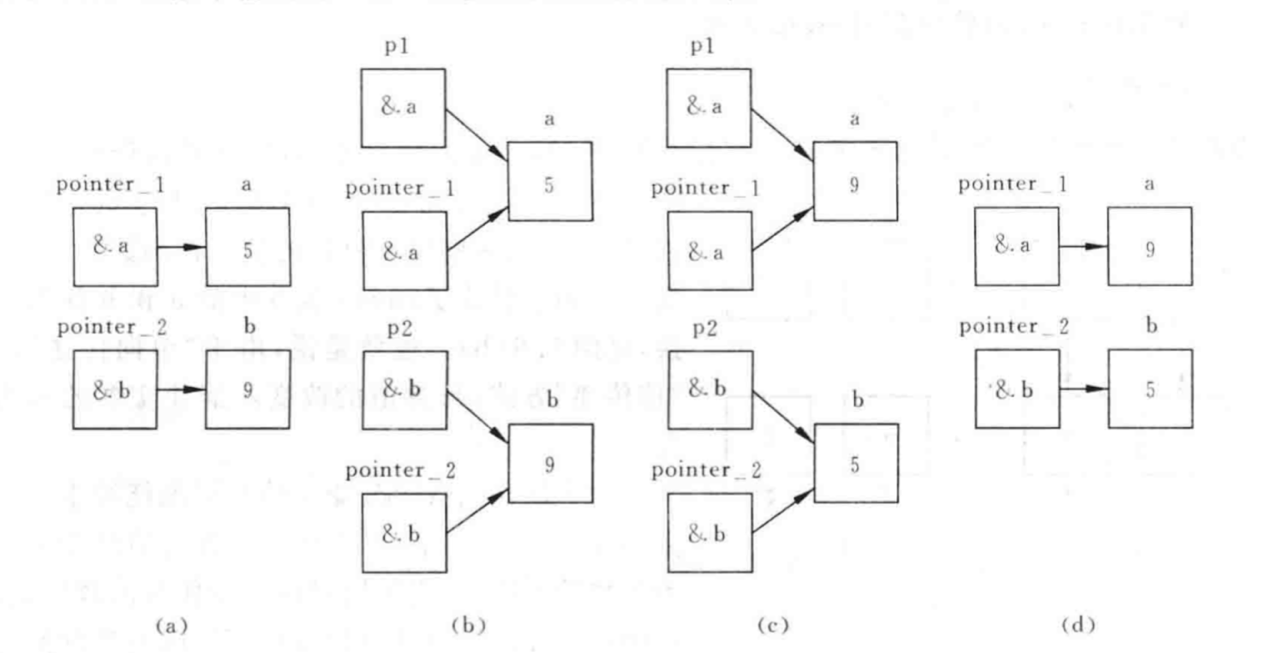
|  |
| --- |
| #include<stdio.h>  int main(){  int \*p1,\*p2,\*p,a,b;  printf("please enter two integer numbers:");  scanf("%d,%d",&a,&b);  p1=&a;  p2=&b;  if(a<b){  p=p1;  p1=p2;  p2=p;    }  printf("a=%d,b=%d\n",a,b);  printf("max=%d,min=%d\n",\*p1,\*p2);  return 0;  } |



### 6.2.3指针变量作函数参数

例题6.2，同上，对输入的两个整数按大小顺序输出，现用函数处理，且用指针类型的数据作函数参数

|  |
| --- |
| #include<stdio.h>  int main(){  void swap(int \*p1,int \*p2);  int a,b;  int \*pointer\_1,\*pointer\_2;  printf("please enter two integer numbers:");  scanf("%d,%d",&a,&b);  pointer\_1=&a;  pointer\_2=&b;  if(a<b){  swap(pointer\_1,pointer\_2);  }  printf("max=%d,min=%d\n",a,b);  return 0;  }  void swap(int \*p1,int \*p2){  int temp;  temp = \*p1;  \*p1=\*p2;  \*p2=temp;    } |



本例是交换a和b的值，而p1p2值不变，和上例相反

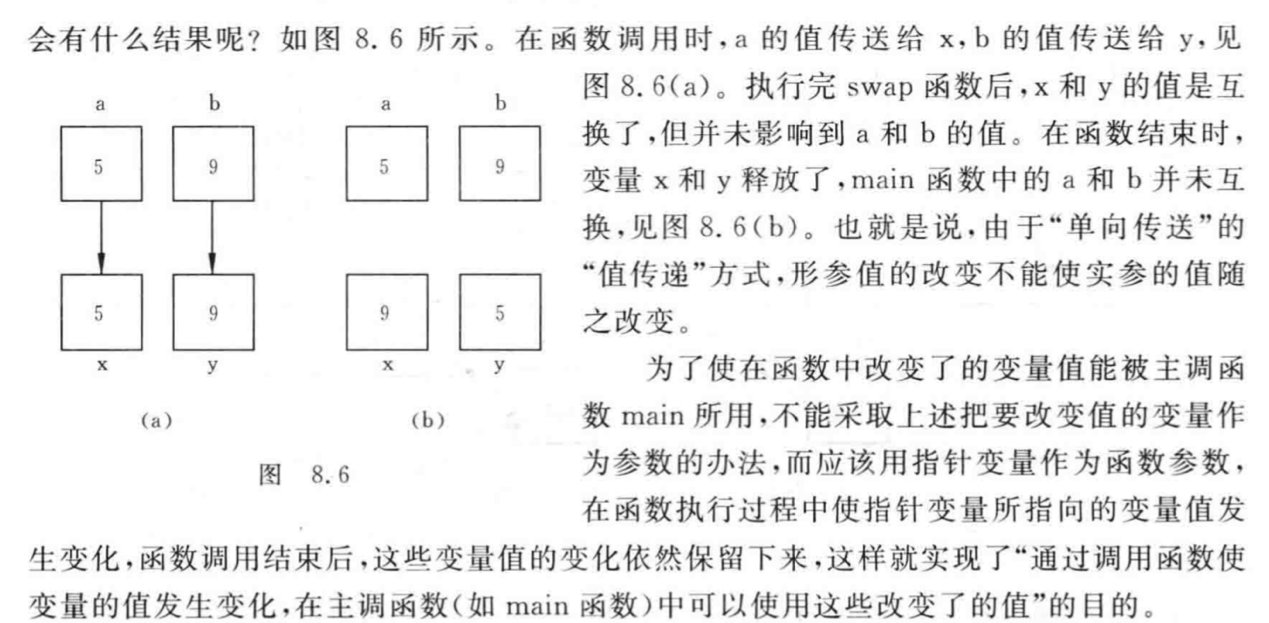
请注意，如果写成下面这样就有问题了

|  |
| --- |
| void swap(int \*p1,int \*p2){  int \*temp;  \*temp = \*p1;//此语句有问题  p1=\*p2;  p2=temp;  } |

\*p1是a，\*temp是指针变量temp所指向的变量，但由于未给temp赋值，因此无确定值，可能破坏系统正常工作状态。

请注意，如果采用下面的方法，会发生什么情况

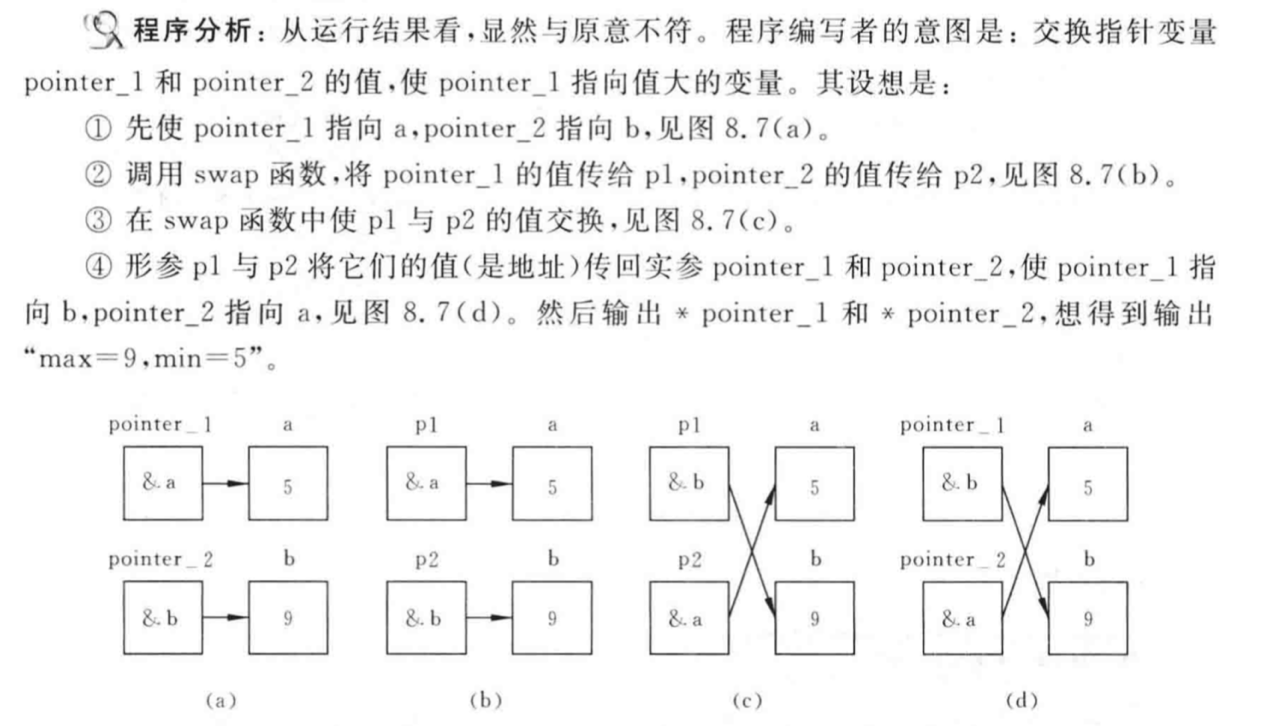
|  |
| --- |
| void swap(int x,int y){  int temp;  temp = x;  x=y;  y=temp;    } |



不要企图通过改变指针形参的值而使得指针实参的值该变,来看下面的例子

例题6.4：对输入的两个整数按大小顺序输出

|  |
| --- |
| #include<stdio.h>  int main(){  void swap(int \*p1,int \*p2);  int a,b;  int \*pointer\_1,\*pointer\_2;  printf("please enter two integer numbers:");  scanf("%d,%d",&a,&b);  pointer\_1=&a;  pointer\_2=&b;  if(a<b){  swap(pointer\_1,pointer\_2);  }  printf("max=%d,min=%d\n",a,b);  return 0;  }  void swap(int \*p1,int \*p2){  int \*temp;  temp = p1;  p1=p2;  p2=temp;    } |



用指针做函数参数要遵循一规则：不可能通过执行调用函数来改变实参指针变量的值，但是可以改变实参指针变量所指的变量的值

## 6.3通过指针引用数组

### 6.3.1数组元素的指针

数组元素指针就是数组元素地址

Int a[10]={1,3,5,7,9};

Int \*p;

P=&a[0];与p=a等价。p是数组a的手元素的地址

注意：数组名不代表整个数组，只代表首元素的地址。上述p=a的作用是把a数组的首元素的地址赋给指针变量p，而不是把数组a各元素的值赋给p

定义指针变量可以初始化

Int \*p=&a[0]等价于

Int \*p

P=&a[0]

或者写成

Int \*p=a

作用是将a数组首元素a[0]的地址赋给指针变量p，而不是赋给\*p

### 6.3.2在引用数组元素时指针的运算

p+1指向数组元素中下一个元素，实际上是p+1\*d

p-1指向数组元素中上一个元素

p++ ++p

p-- --p

p1-p2，如果p1和p2指向同一数组，结果是两个地址之差/数组元素长度

p+i，如果p初值为&a[0]，p+9和a+9都代表&a[9]

\*(p+i) \*(a+i) a[i]的值三者等价

### 6.3.3通过指针引用数组元素

下标法a[i]

例题6.6有一个整型数组a，有10个元素，要求输出数组中的全部元素

|  |
| --- |
| #include<stdio.h>  int main(){  int a[10];  int i;  printf("please enter 10 integer numbers:");  for(i=0;i<10;i++){  scanf("%d",&a[i]);  }  for(i=0;i<10;i++){  printf("%d",a[i]);  }  printf("\n");  return 0;  } |

（2）通过数组名计算数组元素地址，找出元素值

|  |
| --- |
| #include<stdio.h>  int main(){  int a[10];  int i;  printf("please enter 10 integer numbers:");  for(i=0;i<10;i++){  scanf("%d",&a[i]);  }  for(i=0;i<10;i++){  printf("%d",\*(a+i));  }  printf("\n");  return 0;  } |

指针法\*(a+i)或者\*(p+i)，其中a为数组名，p为指向数组元素的指针变量，初值p=a

(3)用指针变量指向数组元素

|  |
| --- |
| #include<stdio.h>  int main(){  int a[10];  int i,\*p;  printf("please enter 10 integer numbers:");  for(i=0;i<10;i++){  scanf("%d",&a[i]);  }  for(p=a;p<(a+10);p++){  printf("%d",\*p);  }  printf("\n");  return 0;  } |

例题6.7：通过指针变量输出整型数组a的10个元素

|  |
| --- |
| #include<stdio.h>  int main(){  int a[10];  int i,\*p;  p=a;  printf("please enter 10 integer numbers:");  for(i=0;i<10;i++){  scanf("%d",p++);  }  P=a;  for(i=0;i<10;i++,p++){  printf("%d",\*p);  }  printf("\n");  return 0;  } |

分析

1. p++使p指向下一元素a[1]

\*p得到下一元素a[1]的值

（2）\*p++

等价于\*(p++)先引用p的值，然后实现\*p的操作，然后p自增1

（3）\*(p++) 和\*(++p)

前者是先取\*p的值，然后使p加1，后者是先p加1再取\*p

如果p初值为&a[0]，则前者输出a[0]的值后者输出a[1]的值

（4）++(\*p)

表示p所指向的元素值加1

若p=a，相当于++a[0]。，若a[0]=3。最终值是4

### 6.3.4用数组名做函数参数

表以变量名和数组名做函数参数的对比

|  |  |  |
| --- | --- | --- |
| 实参类型 | 变量名 | 数组名 |
| 要求形参的类型 | 变量名 | 数组名或指针变量 |
| 传递的信息 | 变量的值 | 实参数组首元素的地址 |
| 通过函数调用能否改变实参的值 | 不能 | 能 |

用户可以认为有一个形参数组，它从实参数组哪里得到起始地址，因此形参数组和实参数组公用一段存储空间，在调用函数期间，如果改变形参数组的值，也就是改变实参数组的值。

注意：实参数组名代表一个固定的地址，或者说指针常量，形参数组名并不是一个固定的地址，而是按照指针变量处理

Void fun(arr[],int n){

Printf(“%d\n”,\*arr);

Arr=arr+3;

Printf(“%d\n”,\*arr);

}

例题6.8：将数组a中n个整数按照相反顺序存放

|  |
| --- |
| #include<stdio.h>  int main(){  void inv(int x[],int n);  int a[10]={3,7,9,11,0,6,7,5,4,2};  int i;  printf("the original array:\n");  for(i=0;i<10;i++){  printf("%d",a[i]);  }  printf("\n");  inv(a,10);  for(i=0;i<10;i++){  printf("%d",a[i]);  }  printf("\n");  return 0;  }  void inv(int x[],int n){  int temp,i,j,m=(n-1)/2;  for(i=0;i<=m;i++){  j=n-1-i;  temp=x[i];  x[i]=x[j];  x[j]=temp;  }  return ;  } |

|  |
| --- |
| #include<stdio.h>  int main(){  void inv(int x[],int n);  int a[10]={3,7,9,11,0,6,7,5,4,2};  int i;  printf("the original array:\n");  for(i=0;i<10;i++){  printf("%d",a[i]);  }  printf("\n");  inv(a,10);  for(i=0;i<10;i++){  printf("%d",a[i]);  }  printf("\n");  return 0;  }  void inv(int \*x,int n){  int \*p,temp,\*i,\*j,m=(n-1)/2;  i=x;  j=x+n-1;  p=x+m;  for(i=x;i<=p;i++){  temp=\*i;  \*i=\*j;  \*j=temp;  }  return ;  } |

例题6.9：将6.8改成指针变量做实参

|  |
| --- |
| #include<stdio.h>  int main(){  void inv(int x[],int n);  int a[10]={3,7,9,11,0,6,7,5,4,2};  int i,\*p=a;  printf("the original array:\n");  for(i=0;i<10;i++){  printf("%d",a[i]);  }  printf("\n");  inv(a,10);  for(p=a;p<a+10;p++){  printf("%d",\*p);  }  printf("\n");  return 0;  }  void inv(int \*x,int n){  int \*p,temp,\*i,\*j,m=(n-1)/2;  i=x;  j=x+n-1;  p=x+m;  for(i=x;i<=p;i++){  temp=\*i;  \*i=\*j;  \*j=temp;  }  return ;  } |

例题6.10 ：用指针方法对10个整数按由大到小的顺序排序

选择法

|  |
| --- |
| #include<stdio.h>  int main(){  void sort(int x[],int n);  void inv(int x[],int n);  int a[10]={3,7,9,11,0,6,7,5,4,2};  int i,\*p;  p=a;  printf("the original array:\n");  for(i=0;i<10;i++){  printf("%d",a[i]);  }  printf("\n");  sort(p,10);  for(p=a;p<a+10;p++){  printf("%d",\*p);  }  printf("\n");  return 0;  }  void sort(int x[],int n){  int i,j,k,t;  for(int i=0;i<n;i++){  k=i;  for(int j=i+1;j<n;j++){  if(x[j]>x[k]){  k=j;  }  if(k!=i){  t=x[i];  x[i]=x[k];  x[k]=t;  }  }  }  } |

### 6.3.5通过指针引用多维数组

#### （1）多维数组元素的地址

a[0]代表一维数组a[0]中第0列元素的地址即&a[0][0]

a[1]代表&a[1][0]

a[2]代表&a[2][0]

a[0]+0，a[0]+1分别代笔a[0][0]，a[0][1]元素的地址即&a[0][0]&a[0][1]

a[0]==\*(a+0) a[1]==\*(a+1)

a[0]+1==\*(a+0)+1==&a[0][1]

\*(\*(a+0)+1)/\*(\*a+1)==a[0][1]的值

\*(a[i]+j)/\*(\*(a+i)+j)==a[i][j]的值

\*(a+i)==a[i]

A,a+I,a[i],\*(a+i),\*(a+i)+j,a[i]+j都是地址

\*(a[i]+j)和\*(\*(a+i)+j)都是二维数组元素a[i][j]的值

表6.2二维数组a的有关指针

|  |  |  |
| --- | --- | --- |
| 表示形式 | 含义 | 值 |
| A | 二维数组名，指向一维数组a[0]，即0行起始地址 | 2000 |
| A[0]，\*(a+0),\*a | 0行0列地址 | 2000 |
| A+1，&a[1] | 1行起始地址 | 2016 |
| A[1],\*(a+1) | 1行0列元素a[1][0]的地址 | 2016 |
| A[1]+2,\*(a+1)+2,&a[1][2] | 1行2列元素a[1][2]的地址 | 2024 |
| \*(a[1]+2),\*(\*(a+1)+2),a[1][2] | 1行2列元素a[1][2]的值 | 是元素值，为13 |

a+1：序号为1的行的首地址，指向序号为1的行

\*(a+1)/a[1]/a[1]+0指向1行0列的元素

二维数组名是指向行的

一维数组名是指向列元素的a[0]a[1]，a[0]+1比表示一个a元素所代表的字节数，在指向行的指针前面加一个\*就转为指向列的指针。a和a+1都是指向行的指针，再他门前面加\*，成为指向列的指针，分别指向a数组元素0行0列和1行0列元素

反之列指针前面加&，得到&a[0]由于a[0]与\*(a+0)等价，因此&a[0]==&\*a指向二维数组的0行

注意

&a[i]和a[i]值相同，但含义不同。

&a[i]/a+i指向行

a[i]/\*(a+i)指向列

当j为0时值是相同的

一维数组中a+i指向的是一个数组元素的存储单元，在该单元中有具体值

二维数组中a+i指向行，a+i，a[i],\*(a+i),&a[i],&a[i][0]值相同代表同一地址

例题6.11输出二维数组的有关数据

|  |
| --- |
| #include<stdio.h>  int main(){  int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};  printf("%d,%d\n",a,\*a);  printf("%d,%d\n",a[0],\*(a+0));  printf("%d,%d\n",&a[0],&a[0][0]);  printf("%d,%d\n",a[1],a+1);  printf("%d,%d\n",&a[1][0],\*(a+1)+0);  printf("%d,%d\n",a[2],\*(a+2));  printf("%d,%d\n",&a[2],a+2);  printf("%d,%d\n",a[1][0],\*(\*(a+1)+0));  printf("%d,%d\n",\*a[2],\*(\*(a+2)+0));  } |
| 301132640,301132640  301132640,301132640  301132640,301132640  301132656,301132656  301132656,301132656  301132672,301132672  301132672,301132672  9,9  17,17 |

#### (2)指向多为数组元素的指针变量

指向数组元素的指针变量

例题6.12：有一个3\*4的二维数组，要求用指向元素的指针变量输出二维数组各元素的值

|  |
| --- |
| #include<stdio.h>  int main(){  int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};  int \*p;  for(p=a[0];p<a[0]+12;p++){  if((p-a[0])%4==0){  printf("\n");  }  printf("%4d",\*p);  }  printf("\n");  return 0;  } |
| 1 3 5 7  9 11 13 15  17 19 21 23 |

指向由m歌元素组成的一位数组指针变量

例题6.13：输出二维数组任一行任一列元素的值

|  |
| --- |
| #include<stdio.h>  int main(){  int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};  int (\*p)[4],i,j;  p=a;  printf("please enter row and column\n");  scanf("%d,%d",&i,&j);  printf("a[%d,%d]=%d\n",i,j,\*(\*(p+i)+j));  return 0;  } |

用指向数组的指针做函数参数

例题8.14：有一个班，3个学生4门课，计算各平均分数以及第n个学生的成绩

|  |
| --- |
| #include<stdio.h>  int main(){  void average(float \*p,int n);  void search(float (\*p)[4],int n);  float score[3][4]={{65,67,70,60},{80,87,90,81},{90,99,100,98}};  average(\*score,12);  search(score,2);  return 0;  }  void average(float \*p,int n){  float \*p\_end;  float sum=0,aver;  p\_end=p+n-1;  for(;p<p\_end;p++){  sum=sum+(\*p);    }  aver = sum/n;  printf("average=%5.2f\n",aver);  }  void search(float(\*p)[4],int n){  int i;  printf("the score of no%d are :\n",n);  for(int i=0;i<4;i++){  printf("%5.2f",\*(\*(p+n)+i));  }  printf("\n");  } |

例题6.15 在上题基础上，查找一门以上课程不及格学生，输出他们全部课程的成绩

|  |
| --- |
| #include<stdio.h>  int main(){  void search(float (\*p)[4],int n);  float score[3][4]={{65,67,70,60},{57,87,90,81},{90,99,100,98}};  search(score,3);  return 0;  }  void search(float(\*p)[4],int n){  int i,j,flag;  for(j=0;i<n;j++){  flag=0;  for(int i=0;i<4;i++){  if(\*(\*(p+j)+i)<60){  flag=1;  }  }  if(flag==1){  printf("no.%d fails his scores are:\n",j+1);  for(i=0;i<4;i++){  printf("%5.1f",\*(\*(p+j)+i));  }  printf("\n");  }  }  } |

## 6.4通过指针引用字符串

### 6.4.1字符串的饮用方式

（1）用%s输出该字符串

Char string[]=”I love china”

Printf(“%s”,string)

Printf(“%c”,string[7])

1. 用字符指针变量指向一个字符串常量，通过字符指针变量引用字符串常量

Char \*string=”I love china”

Printf(“%s”,string)

string指向字符串的第一个字符

相当于下面的两行

Char \*string;

String=”I love china”

说明： 通过字符数组名或字符指针变量可以输出一个字符串，而对一个数值型数组，是不能企图用数组名输出他的全部元素的

例题：6.18将字符串a复制到字符串b，然后输出字符串b

|  |
| --- |
| #include<stdio.h>  int main(){  char a[]="i am student",b[20];  int i;  for(i=0;\*(a+i)!='\0';i++){  \*(b+i)=\*(a+i);  }  \*(b+i)='\0';  printf("string a is：%s\n",a);  printf("string b is");  for(int i=0;b[i]!='\0';i++){  printf("%c",b[i]);  }  return 0;  } |

例题：6.19 用指针变量来处理6.18问题

|  |
| --- |
| #include<stdio.h>  int main(){  char a[]="i am student",b[20],\*p1,\*p2;  p1=a;  p2=b;  int i;  for(;\*p1!='\0';p1++,p2++){  \*p2=\*p1;  }  \*p2='\0';  printf("string a is：%s\n",a);  printf("string b is:%s",b);  return 0;  } |

### 6.4.2字符指针做函数参数

例题6.20：用函数调用实现字符串的复制

1. 用字符数组名做函数参数

|  |
| --- |
| #include<stdio.h>  int main(){  void copy\_string(char from[],char to[]);  char a[]="i am student";  char b[]="you are a student";  printf("string a=%s\n string b=%s\n",a,b);  printf("copy string a to b \n");  copy\_string(a,b);  printf("string a is：%s\n",a);  printf("string b is:%s",b);  return 0;  }  void copy\_string(char from[],char to[]){  int i=0;  while(from[i]!='\0'){  to[i]=from[i];  i++;  }  to[i]='\0';  } |

1. 用字符型指针变量做实参

|  |
| --- |
| #include<stdio.h>  int main(){  void copy\_string(char from[],char to[]);  char a[]="i am student";  char b[]="you are a student";  char \*from=a,\*to=b;  printf("string a=%s\n string b=%s\n",a,b);  printf("copy string a to b \n");  copy\_string(a,b);  printf("string a is：%s\n",a);  printf("string b is:%s",b);  return 0;  }  void copy\_string(char from[],char to[]){  int i=0;  while(from[i]!='\0'){  to[i]=from[i];  i++;  }  to[i]='\0';  } |

1. 用字符指针变量做形参和实参

|  |
| --- |
| #include<stdio.h>  int main(){  void copy\_string(char from[],char to[]);  char \*a="i am student";  char b[]="you are a student";  printf("string a=%s\n string b=%s\n",a,b);  printf("copy string a to b \n");  copy\_string(a,b);  printf("string a is：%s\n",a);  printf("string b is:%s",b);  return 0;  }  void copy\_string(char \*from,char \*to){  for(;\*from!='\0';from++,to++){  \*to = \*from;  }  \*to='\0';  } |

表6.3调用函数时实参与形参的对应关系

|  |  |  |  |
| --- | --- | --- | --- |
| 实参 | 形参 | 形参 | 形参 |
| 字符数组名 | 字符数组名 | 字符指针变量 | 字符指振变量 |
| 字符数组名 | 字符指针变量 | 字符指针变量 | 字符指针变量 |

### 6.4.3使用字符指针变量和字符数组的比较

（1）字符数组由若干个元素组成，每个元素中放一个字符，而字符指针变量存放的是地址（字符串第一个字符的地址），绝不是将整个字符串存入字符指针变量中

（2）赋值方式：可以对字符指针变量赋值，但不能对数组名赋值

Char \*a;

A=”I love china”

不能用以下办法对字符数组名赋值

Char str[14]

str[0]=”i”//合法

str=”I love china”//错误，数组名是地址是常量，不能被赋值

（3）初始化的含义，对字符指针变量赋初值

Char \*a = “I love china”

等价于

Char \*a;

A=”I love chian”

而对数组的初始化

Char str[14]=”I love china”

并不等价于

Char str[14];

Str[]=”I love china”

（4）存储单元内容。编译时为字符数组分配若干个存储单元，已存放各元素的值，而对字符指针变量，只分配一个存储单元

（5）指针变量的值可以改变，而数组名代表一个固定的值（数组首元素的地址），不能改变

Char \*a=”I love chian”

A=a+7

Printf(“%s”,a)

错误的如下

Char str[]=”I love chian”

A=a+7

Printf(“%s”,a)

（6）字符数组中各元素的值是可以改变的（可以对他们再赋值），但字符指针变量指向的字符串常量中的内容是不可以被取代的（不能对他们再赋值）

Char a[]=”house”

Char\*b=”house”

A[2]=”r”

B[2]=”r”

## 6.5指向函数的指针

### 6.5.1定义

存储空间的起始地址（入口地址），成为函数的指针

Int (\*P)(INT,INT)

### 6.5.2用函数指针变脸调用函数

例题6.22用函数求整数a和b中的大者

1. 通过函数名调用函数

|  |
| --- |
| #include<stdio.h>  int main(){  int a,b,c;  printf("please enter a and b");  scanf("%d,%d\n",&a,&b);  c=max(a,b);  printf("a=%d\nb=%d\nmax=%d\n",a,b,c);  return 0;  }  int max(int x,int y){  int z;  if(x<y){  z=x;  }else{  z=y;  }  return(z);  } |

(2) 通过指针变量访问它所指向的函数

|  |
| --- |
| #include<stdio.h>  int main(){  int a,b,c;  int max(int,int);  int (\*p)(int,int);  p=max;  printf("please enter a and b");  scanf("%d,%d\n",&a,&b);  c=(\*p)(a,b);  printf("a=%d\nb=%d\nmax=%d\n",a,b,c);  return 0;  }  int max(int x,int y){  int z;  if(x<y){  z=x;  }else{  z=y;  }  return(z);  } |

### 6.5.2怎样定义和使用指向函数的指针变量

类型名（\*指针变量名）（函数参数表列）

例题6.23输入两个整数，然后让用户选择1或2，选1时调用max函数，选2时调用min函数

|  |
| --- |
| #include<stdio.h>  int main(){  int a,b,c,n;  int max(int,int);  int min(int ,int);  int (\*p)(int,int);  printf("please enter a and b");  scanf("%d,%d\n",&a,&b);  printf("please choose 1 or 2");  scanf("%d,",&n);  if(n==1){  p=max;  }else {  if(n==2){  p=min;  }    }  c=(\*p)(a,b);  printf("a=%d\nb=%d\n",a,b);  if(n==1){  printf("max=%d\n",c);  }else {  if(n==2){  printf("min=%d\n",c);  }    }  return 0;  }  int max(int x,int y){  int z;  if(x<y){  z=x;  }else{  z=y;  }  return(z);  }  int min(int x,int y){  int z;  if(x<y){  z=y;  }else{  z=x;  }  return(z);  } |

### 6.5.4用指向函数的指针做函数参数

指向函数的指针变量的一个重要用途就是把函数的地址作为参数传递到其他函数

例题6.24：有两个整数a和b，由用户输入1，2，3，如输入1，程序就给出a和b中大者，输入2，给出min，输出3，给出sum

|  |
| --- |
| #include<stdio.h>  int main(){  int fun(int x,int y,int (\*p)(int, int));  int a=34,b=-21,n;  int max(int,int);  int min(int ,int);  int add(int,int);  int (\*p)(int,int);  printf("please choose 1 or 2 or 3");  scanf("%d,",&n);  if(n==1){  p=max;  }else {  if(n==2){  p=min;  }    }  if(n==1) fun(a,b,max);  else if(n==2) fun(a,b,min);  else if(n==3) fun(a,b,add);  return 0;  }  int fun(int x,int y,int (\*p)(int,int)){  int result;  result=(\*p)(x,y);  printf("%d\n",result);  }  int max(int x,int y){  int z;  if(x<y){  z=x;  }else{  z=y;  }  return(z);  }  int min(int x,int y){  int z;  if(x<y){  z=y;  }else{  z=x;  }  return(z);  }  int add(int x,int y){  int z;  z=x+y;  return(z);  } |

## 6.6返回指针值得函数

一个函数返回一个整形值，可以返回指针型的数据即地址

类型名 \*函数名（参数表列）

例6.25 有a个学生，每个学生有b门课程，要求在用户输入学生序号后，能输出该学生全部成绩。用指针函数实现

|  |
| --- |
| #include<stdio.h>  int main(){  float \*search(float (\*p)[4],int n);  float score[3][4]={{65,67,70,60},{57,87,90,81},{90,99,100,98}};  float \*p;  int i,k;  printf("enter the number of students");  scanf("%d",&k);  printf("the scores of no%d are :\n",k);  p=search(score,k);  for(i=0;i<4;i++){  printf("%5.2f\t",\*(p+i));  }  printf("\n");  return 0;  }  float \*search(float(\*pointer)[4],int n){  float \*pt;  pt=\*(pointer+n);  return (pt);  } |

例题6.26对例6.25 中的学生，找出其中不及格的课程的学生及其学生号

## 6.7指针数组和多重指针

### 6.7.1定义

一个数组，若其元素均为指针类型数据，称为指针数组

Int \*p[4]

Int(\*p)[4]//这是指向一维数组的指针变量

类型名\*数组名[数组长度]

指针数组适合指向若干个字符串，使字符串处理更加方便

例题：6.27 将若干个字符串按字母顺序由小到大输出

|  |
| --- |
| #include<stdio.h>  #include<string.h>  int main(){  void sort(char \* name[],int n);  void print(char \* name[],int n);  char \* name[]={"follow me","basic","great wall","fortran","computer design"};  int n=5;  sort(name,n);  print(name,n);  return 0;    }  void sort(char \* name[],int n){  char \*temp;  int i,j,k;  for(i=0;i<n-1;i++){  k=i;  for(j=i+1;j<n;j++){  if(strcmp(name[k],name[j])>0)k=j;    }  if(k!=i){  temp = name[i];  name[i]=name[k];  name[k]=temp;  }    }  }  void print(char \* name[],int n){  int i;  for(i=0;i<n;i++){  printf("%s\n",name[i]);  }    } |

### 8.7.2指向指针数据的指针

Char \*\*p

\*(\*p)

\*p是指针变量

Char \*是p指向的是char\*型的数据，也就是说p指向一个字符指针变量，这个字符指针变量指向一个字符型数据。

p=name+2

printf(“%D”\*P)//NAME[2]的值，是地址

printf(“%s”,\*p)//输出great wall

例题6.28使用指向指针数据的指针变量

|  |
| --- |
| #include<stdio.h>  int main(){  char \*name[]={"follow me","basic","great wall","fortran","computer design"};  char \*\*p;  int i;  for(i=0;i<5;i++){  p=name+i;  printf("%s\n",\*p);  }  return 0;  } |

P指向char\*的指针变量，执行p=name+i使p指向name数组的0号元素\*p是name[0]的值，即第一个字符串首字母的地址，printf输出第一个字符串

说明指针型数组的元素也可以指向整形或者实型数据

Int a[5]={1,2,3,4,5};

Int \*name[5],I;

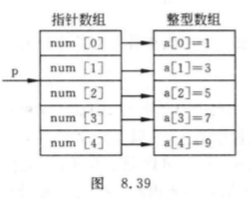
Int \*\*p;

For(i=0;i<5;i++){

Num[i]=&a[i];

}

为了得到a[2]中的数据3，可以先使p=num+2，然后输出\*\*p，\*p是num[2]的值，而num[2]的值是a[2]的地址，因此\*\*P是a[2]的值5



例题6.29有一个指针数组，其元素分别指向一个整形数组的元素，用指向指针数据的指针变量，输出整形数组各元素的值

|  |
| --- |
| #include<stdio.h>  int main(){  int a[5]={1,3,5,7,9};  int \*num[5]={&a[0],&a[1],&a[2],&a[3],&a[4]};  int \*\*p,i;  p=num;  for(int i=0;i<5;i++){  printf("%d\n",\*\*p);  p++;  }  return 0;  } |

## 6.8动态内存分配与指向它的指针变量

全局变量是分配在内存中的静态存储区

非静态的局部变量是分配在内存中的动态存储区（栈）

堆：内存动态分配区域，存放一些临时的数据，不必在程序的声明部分定义，也不必等到函数结束时释放

### 6.8.2怎样建立内存的动态分配

（1）Malloc开辟动态存储区

Void \*malloc(unsigned int size)

作用是在内存的动态存储区中分配一个长度为size 的连续空间。形参size的类型定为无符号整型，不匀速为负数。返回值是所分配区域的第一个地址

Malloc(100)//开辟100字节的临时分配域，函数值为其第一个字符的地址

注意，指针的类型void，不指向任何类型的数据，只提供一个纯地址。如果函数为能成功执行，返回null

（2）用calloc函数开辟动态存储区

Void \*calloc(unsigned n,unsigned size)

作用，内存中分配n个长度为size的连续空间

如果分配成功，函数返回指向所分配域的第一个字节的指针；如果分配失败，返回null

（3）用realloc重新分配动态存储区

Void \*callreallococ(void \*p，unsigned int size)

如果已经使用malloc或calloc获得了动态空间，想改变大小

（4）free释放动态存储区

Free(p)

### 6.8.3void指针类型

例题6.30 建立动态数组，输入5个学生成绩，另外用一个函数，检查其中有无低于60分的，输出不合格的成绩

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  int main(){  void check(int \*);  int \*p1,i;  p1=(int \*)malloc(5\*sizeof(int));//开辟动态内存区将地址转换成int \*型，然后放到p1中  for(int i=0;i<5;i++){  scanf("%d\n",p1+i);  }  check(p1);  return 0;  }  void check(int \*p){  int i;  printf("they are fail:");  for(int i=0;i<5;i++){  if(p[i]<60){  printf("%d\n",p[i]);  }  }  printf("\n");  } |

## 6.9指针小结

（1）指针就是地址

&a是变量a的地址，也可以称为变量a的指针

指针变量是存放地址的变量，也可以说是，指针变量是存放指针的变量

指针变量的值是一个地址（指针变量的值是一个指针）

指针变量也可称为地址变量，他的值是地址

&取地址，&a，a的地址，&是取指针运算符，&a是变量a的指针（指向a的指针）

数组名是一个地址，是元素首元素的地址（数组名是一个指针，是数组首元素的指针）

函数名是一个指针（指向函数代码去的首字节），也可以说，函数名是一个地址（函数代码区首字节的地址）

函数实参如果是数组名，传递给形参的是一个地址，也可以说，传递给形参的是一个指针

（2）要区别指针喝指针变量

指针是地址，指针变量是用来存放地址的变量

指针变量的值是一个地址

（3）什么叫指向

地址意味着指向，因为通过地址能找到具有该地址的对象。对于指针变量来说，把谁的地址存放在指针变量中，就说此指针变量指向谁

Int a,\*p

Float b

P=&a

P=&b；//错误，b是float类型，类型不匹配

（4）搞清楚指针指向

一维数组名代表数组首元素的地址

Int \*p,a[10];

P=a;

P是指向int型类型的指针变量，显然p只能指向数组中的元素（int型变量），而不是整个数组

对于p=a，准确的来说，p指向a数组的首元素

P指向字符串中的首字符

（5）有关指针变量的归纳比较

表6.4指针变量的类型及含义

|  |  |  |
| --- | --- | --- |
| 变量定义 | 类型表示 | 含义 |
| Int i | Int | 定义整型变量i |
| Int \*p | Int \* | 定义p为指向整型数据的指针变量 |
| Int a[5] | Int [5] | 定义整型数组a，它有5个元素 |
| Int \*p[4] | Int \* [4] | 定义指针数组p，它由4个指向整型数据的指针元素组成 |
| Int (\*p)[4] | Int (\*)[4] | P为指向包含4个元素的一维数组的指针变量 |
| Int f() | Int () | F为返回整型函数值的函数 |
| Int \*p() | Int \*() | P为返回一个指针的函数，该指针指向整型数据 |
| Int (\*p)() | Int (\*)() | P为指向函数的指针，该函数返回一个整型值 |
| Int \*\*p | Int \*\* | P是一个指针变量，它指向一个指向整型数据的指针变量 |
| Void \*p | Void \* | P是一个指针变量，基类型为void（空类型），不指向具体的对象 |

（6）指针运算

* 指针变量加（减）一个整数

例如，p++，p--，p+i，p-i，p+=i，p-=i，均是指针变量加（减）一个整数

* 指针变量赋值

P=&a//将a的地址赋值给p

P=array；//将数组array首元素地址赋给p

P=&array[i]；//将数组array第i个元素的地址赋给p

P=max；//max为已定义函数，将max的入口地址赋给p

P1=p2；//p1和p2是基类型相同的指针变量，将p2的值赋给p1

* 两个指针变量可以相减

如果两个指针变量都指向同一个数组中的元素，则两个指针变量值之差是两个指针之间的元素个数

* 两个指针变量比较

若两个指针指向同一个数组的元素，则可以比较。指向前面的元素的指针变量小于指向后面元素的指针变量。如果p1和p2不指向同意数组则比较无意义

* 指针变量可以有空值

指针变量不指向任何变量

P=null

其中null为一个符号常量，代表整数0

#define null 0

注意p的值为null与未对p赋值是两个不同的概念。前者是有值的，值为0，不指向任何变量。后者虽未对p赋值但不等于p 无值，只是他的值是一个无法预料的值，也就是p可能指向一个事先未指定的单元。这种情况很危险，因此在引用指针变量之前应该对他赋值

任何指针变量或地址都可以与null做相等或不相等的比较

If(p==null)…

# 7用户自己建立数据类型

## 7.1定义和使用结构体变量

### 7.1.1用户自己建立结构体类型

Struct 结构体名

{成员表列}

成员表列

{类型名 成员名}

### 7.1.2定义结构体类型变量

1先声明结构体类型，再定义该类型的变量

Struct student student1,student2

2在声明类型的同时定义变量

Struct 结构体名

{成员表列

}变量表列

3不指定类型名而直接定义结构体类型变量

Struct {

成员表列

}变量名表列

### 7.1.3结构体变量的初始化和引用

（1）定义结构体变量时，可以对它的成员变量初始化

（2）引用结构体变量中成员的值

结构体变量名，成员名

1. 对结构体变量成员可以像普通变量一样进行各种运算
2. 同类的结构体变量可以互相赋值
3. 可以引用结构体变量成员的地址，也可以引用结构体变量的地址

结构体变量地址主要用于函数参数，传递结构体变量的地址

Scanf(“%d”,&student.name);//输入student1.num的值

Scanf(“%d”,&student)//输出结构体变量student1的首地址

例如：输入两个学生的学号，姓名，成绩，输出成绩较高的学生的学号，姓名和成绩

## 7.2结构体数组

### 7.2.1定义结构体数组

{成员表列} 数组名 [数组长度]

结构体类型 数组名[数组长度]

Struct person Leader[3]

对结构体数组初始化的形式是在定义数组的后面加上={初值表列}

Student person leader[3]={“li“,0,”zhang”,0,”sun”,0};

例题 有n个学生，要求按照成绩的高低顺序输出各学生的信息

|  |
| --- |
| #include<stdio.h>  struct Student{  int num;  char name[20];  float score;  };  int main(){  struct Student stu[5]={{10101,"Zhang",78},{10103,"Wang",98.5},{10106,"Li",86},{10108,"Ling",73.5},{10110,"Sun",100}};  struct Student temp;  const int n=5;  int i,j,k;  printf("the order is:\n");  for(i=0;i<n-1;i++){  k=i;  for(j=i+1;j<n;j++){  if(stu[j].score>stu[k].score){  k=j;  }  temp = stu[k];  stu[k]=stu[i];  stu[i]=temp;  }  }  for(i=0;i<n;i++){  printf("%6d\n%8f\n%6.2f\n",stu[i].num,stu[i].name,stu[i].score);  printf("\n");  return 0;  }  } |

## 7.3结构体指针

### 7.3.1指向结构体变量的指针

Struct student \*pt;

例题通过指向结构体变量的指针变量输出结构体变量中成员的信息

|  |
| --- |
| #include<stdio.h>  #include<string.h>  int main(){  struct Student{  long num;  char name[20];  char sex;  float score;  };  struct Student stu\_1;  struct Student \*p;  p=&stu\_1;  stu\_1.num=10101;  strcpy(stu\_1.name,"li lin");  stu\_1.sex='m';  stu\_1.score=89.5;  printf("no.:%ld\nname:%s\nsex:%c\nscore:%5.1f\n",stu\_1.num,stu\_1.name,stu\_1.sex,stu\_1.score);  printf("\nno.:%ld\nname:%s\nsex:%c\nscore:%5.1f\n",(\*p).num,(\*p).name,(\*p).sex,(\*p).score);  return 0;  } |

\*p两侧的括号不可省略，\*p.num等价于\*（p.num）

如果p指向一个结构体变量stu，以下三种方式

* Stu.成员名
* (\*P).成员名
* P->成员名

### 7.3.2指向结构体数组的指针

例题：有3个学生的信息，放在结构体数组中，要球输出全部学生的信息

|  |
| --- |
| #include<stdio.h>  #include<string.h>  struct Student{  long num;  char name[20];  char sex;  float score;  };  struct Student stu[5]={{10101,"Zhang",78},{10103,"Wang",98.5},{10106,"Li",86},{10108,"Ling",73.5},{10110,"Sun",100}};  int main(){  struct Student \*p;  for(p=stu;p<stu+3;p++){  printf("no.:%ld\nname:%s\nsex:%c\nscore:%5.1f\n",p->num,p->name,p->sex,p->score);  }  return 0;  } |

注意：

1. 如果p的初值为stu，即指向stu的序号为0的元素

P+1就指向下一元素

(++p)->num//p自加1，得到p指向元素中的num成员值（即10102）

(p++)->num//先求p->num，（10101）然后p加1指向stu[1]

（2）p为指向struct student类型对象的指针变量，用来指向一个struct student对象，不能用于指向stu数组元素中的某一成员

### 7.3.3用结构体变量和结构体变量的指针做函数参数

（1）用结构体变量的成员做参数stu[1].num值传递

（2）用结构体变量做实参

（3）用指向结构体变量或数组元素的指针做实参，将结构体变量或数组元素的地址传给形参

例题：有n个结构体变量，内含学生学号，姓名和3门课程的成绩，要求输出平均成绩最高的学生的信息

|  |
| --- |
| #include<stdio.h>  #define N 3  struct Student{  int num;  char name[20];;  float score[3];  float aver;  };  int main(){  void input(struct Student stu[]);  struct Student max(struct Student stu[]);  void print(struct Student stu);  struct Student stu[N],\*p=stu;  input(p);  print(max(p));  return 0;  }  void input(struct Student stu[]){  int i;  printf("请输出各学生的信息：学号，姓名，3门课成绩\n");  for(i=0;i<N;i++){  scanf("%d %s %f %f %f",&stu[i].num,stu[i].name,&stu[i].score[0],&stu[i].score[1],&stu[i].score[2]);  } stu[i].aver = (stu[i].score[0]+stu[i].score[1]+stu[i].score[2])/3.0;    }  struct Student max(struct Student stu[]){  int i,m=0;  for(i=0;i<N;i++){  if(stu[i].aver>stu[m].aver){  m=i;  }  }  return stu[m];  }  void print(struct Student stud){  printf("\n成绩最高的学生是\n");  printf("学号：%d\n姓名:%s\n三门课成绩:%5.1f%5.1f%5.1f\n平均成绩:%6.2f\n",stud.num,stud.name,stud.score[0],stud.score[1],stud.score[2],stud.aver);  } |

## 7.4用指针处理链表

Struct student{

Int num；

Float score；

Struct Student \*next;

}

上面只是定义一个struct student类型，并未实际分配存储空间，只有定义了变量才分配存储单元

### 7.4.2建立简单的静态链表

例 建立一个简单的链表，由3个学生数据的结点组成，要求输出各节点的数据

|  |
| --- |
| #include<stdio.h>  #define N 3  struct Student{  long num;  float score;  struct Student \*next;  };  int main(){  struct Student a,b,c,\*head,\*p;  a.num=10101;  a.score=89.5;  b.num=10103;  b.score=90;  c.num=10107;  c.score=85;  head=&a;  a.next=&b;  b.next=NULL;  do{  printf("%ld %5.1f\n",p->num,p->score);  p=p->next;  }while(p!=NULL);  return 0;  } |

### 7.4.3建立动态链表

例题：写一个函数建立有3个学生数据的单向动态链表

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #define LEN sizeof(struct Student)  struct Student{  long num;  float score;  struct Student \*next;  };  int n;  struct Student \* creatlink(void){  struct Student \*head;  struct Student \*p1,\*p2;  n=0;  p1=p2=(struct Student \*)malloc(LEN);  scanf("%ld,%f",&p1->num,&p1->score);  head = NULL;  while(p1->num!=0){  n=n+1;  if(n==1){  head=p1;  }else{  p2->next=p1;  }  p2=p1;  p1=(struct Student \*)malloc(LEN);  scanf("%ld,%f",&p1->num,&p1->score);    }  p2->next=NULL;  return (head);  };  int main(){  struct Student \*p;  p=creatlink();  printf("\nnum:%ld\nscore:%5.1f\n",p->num,p->score);  return 0;  } |

## 7.5共用体类型

### 7.5.1什么事共用体类型

几个不同的变量共享同一段内存的结构

Union 共用体名

{

成员表列；

}变量表列

### 7.5.2引用共用体变量的方式

a.i（引用共用体变量的整形变量i）

### 7.5.3共用体类型的数据的特点

（1）同一内存段可以用来存放几种不同类型的成员，但在每一瞬间只能存放其中一个成员，而不是同时存放几个

（2）初始化表中只有一个常量

Union data

{

Int I;

Char ch;

Float f;

}a={1,’a’,1.5};

Union data a={16}

Union data a={，ch=‘j’}

（3）共用体变量中起作用的成员是最后一次被赋值的成员，在对共用体变量中的一个成员赋值后，原油变量存储单元中的值就被取代

a.ch=’a’

a.f=1.5

a.i=40

(4)共用体变量的地址和他的成员的地址都是同一地址。

例题：有若干个人员的数据，其中有老师有学生。学生数据包括，姓名，号码，性别，职业，班级，教师的数据包括：姓名，号码，性别，职业，职业，要求用一个表格处理

|  |
| --- |
| #include<stdio.h>  struct Student{  int num;  char name[10];  char sex;  char job;  union{  int class;  char position[10];  }category;  } person[2];  int main(){  int i;  for(i=0;i<2;i++){  printf("please enter the data of person");  scanf("%d %s %c %c",&person[i].num,&person[i].name,&person[i].sex,&person[i].job);  if(person[i].job=='s'){  scanf("%d",&person[i].category.class);  }else if(person[i].job =='t'){  scanf("%s",&person[i].category.position);  }else{  printf("input error!");  }  }  printf("\n");  printf("no. name sex job class/position\n");  for (int i=0;i<2 ; i++) {  if(person[i].job =='s'){  printf("%-6d%-10s%-4c%-4c%-10d\n",person[i].num,person[i].name,person[i].sex,person[i].job,person[i].category.class);  }else{  printf("%-6d%-10s%-4c%-4c%-10d\n",person[i].num,person[i].name,person[i].sex,person[i].job,person[i].category.position);  }  }  } |

## 7.6使用枚举类型

enum {枚举名} 枚举元素列表

不能赋值

enum{sun,Mon,tue,wed,thu,fri,sat}workday,weekend

Sum=0//错误

例题：口袋里有红，黄，蓝，白，黑5种颜色的球若干，每次从口袋中先后取出3个球，问得到3种不同颜色球的坑内的取法，输出每种排列的情况

|  |
| --- |
| #include<stdio.h>  int main(){  enum Color{red,yellow,blue,white,black};  enum Color i,j,k,pri;  int n,loop;  n=0;  for(i=red;i<=black;i++){  for(j=red;j<=black;j++){  if(i!=j){  for(k=red;k<=black;k++){  if((k!=i)&&(k!=j)){  n=n+1;  printf("%-4d",n);  for(loop=1;loop<=3;loop++){  switch(loop){  case 1:pri=i;break;  case 2:pri=j;break;  case 3:pri=k;break;  default:break;  }  }  switch(pri){  case red:printf("%-10s","red");break;  case yellow:printf("%-10s","yellow");break;  case blue:printf("%-10s","blue");break;  case white:printf("%-10s","white");break;  case black:printf("%-10s","black");break;  default:break;  }  }  printf("\n");  }  }    }  }  printf("\ntotal:%5d\n",n);  return 0;  } |

## 7.7用typeof声明新类型名

C直接提供的标准类型名int char float double long等。和程序者自己编写的结构体，共用体，枚举类型除外，还可以用typeof指定新的类型名代替以有的类型名

1. 简单的用一个新的类型名替换原有的类型名
2. 明明一个简单的类型名代替复杂的类型的表示方法

命名一个新的类型名date。代替上面的一个结构体类型，然后可以用新的类型名dat去定义变量

Typedef struct{

Int month

Int day。

Int year

}Date

1. 明明一个新的类型名代替指针类型
2. Typedef （char \* string）

String p，a[10]

1. 明明一个新策类型名代表提指针类型
2. 明明一个新的类型名代表指向函数的补偿补充

归纳起来，声明一个新的类型名方法

1. 先按定义变量的方法写出定义体，int i
2. 将变量名换成新类型名 ，例如将i换成count
3. 在最前面加typedef 例如typedef int count
4. 然后可以用新类型名去定义变量

Typedef int num[100]

同样对字符指针类型

Char \*p//定义变量p的方式

Char \* string//用新类型名string取代变量名p

Typedef char \* string//加typedef

String p//用新类型名string定义变量，相当char \*p