函数指针

函数: 返回值 函数名 （形参列表）

指针函数： 返回值（\*标识符）（形参类型列表） = 函数名；

用途1 调用函数 用途2 作为函数形参，灵活导入一系列函数标志完全相同的函数。

1. 调用函数

int f1(int x,int y) {

return x - y;

}

int f2(int x,int y) {

return x + y;

}

int f3(int(\*fff)(int,int),int x,int y) {

printf("%d\n",fff(x,y));

}

向f3传入f1或f2（只传函数名），fff就是谁。

int f(char c) {

}

调用方式一

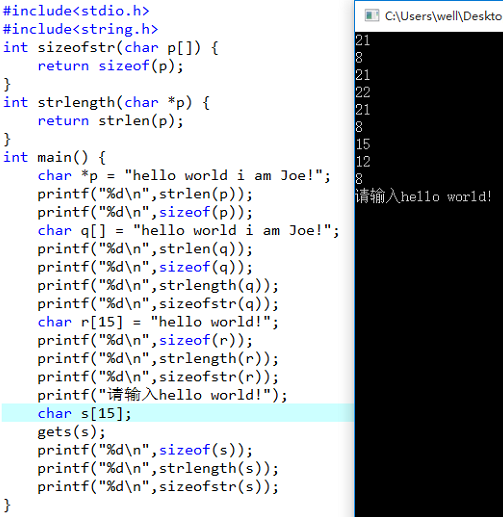
f('g');

调用方式二

int (\*ff)(char) = f;

ff('g');

字符串



char str[3] = "ai";//ok

char str[3];

str[3] = "ai"; //error str是个const常量

strcpy(str,"ai");

字符串长度

size\_t strlen(const char \*string)

返回字符串的长度，实参是个任意字符指针，一直扫到\0,才结束统计，所以字符数组表达的字符串末尾没有\0，数组越界且统计不可能正确。

复制字符串

char \*strcpy(char \*dst,char const \*src)

dst必须是字符数组的基地址！dst的容量要足够大。

扫描src一直到\0结束，连同\0一起复制到dst，如果src末尾没有\0，也会错误。

连接字符串

char \*strcat(char \*dst,char const \*src)

把src复制到dst中，要保证dst剩余的单元足够多！

字符串比较

int strcmp(const char \*s1,const char \*s2)

s1与s2相同，返回0。

s1和s2第一个不匹配的字符假设是a,b;如果a的ASII码大于b的，返回值大于0，否则小于0；

s1和s2末尾都要有\0。

查找一个字符

char\* strchr(const char \*str,int ch) 返回第一次出现的指针

char\* strrchr(const char \*str,int ch) 返回最后一次出现的指针

查找一个子串

char \*strstr(const char \*s1,const char \*s2)

返回在s1匹配的第一个字符的指针，没有返回NULL，s2是空，返回s1。

static

static修饰的全局变量，只能在本文件使用，避免了不同文件的全局变量相同引起的冲突

static修饰的函数，只能在本文件内使用，同上。

static修饰的局部变量，只被初始化一次，自动初始化成0.函数运行结束后不释放，下次运行该函数，在上次的基础值上执行代码。但是变量的作用域仍然于函数内部。

void f() {

static int a;

a = 1;

printf("a = %d\n",a++);

printf("a = %d\n\n",a);

}

int main() {

f();

f();

return 0;

} 打印结果：1 2 1 2

void f() {

static int a;

printf("a = %d\n",a++);

printf("a = %d\n\n",a);

}

int main() {

f();

f();

return 0;

}

打印结果：0 1 1 2

const

const修饰的变量是只读模式，一旦初始化，不能再修改。

const修饰的变量必须显示初始化，定义行直接初始化。

const int a = 1；a的值不能再被修改。

const int \*p；p的指向可变，p指向的变量不能修改。

int \*const p；p的指向不能变，p指向的变量可修改。

const int \*const p；都不能变。

const修饰形参效果同修饰局部变量。

const修饰函数返回值，接收函数返回值的变量必须也是const修饰的变量。

const char \*str，字符串每个单元的字符不能变化。

编译器通常不为普通const常量分配存储空间，而是将它们保存在符号表中，这使得它成为一个编译期间的常量，没有了存储与读内存的操作，使得它的效率也很高。预编译指令只是对值进行简单的替换，不能进行类型检查。

**volatile**

内存访问速度远不及CPU处理速度，为提高机器整体性能，在硬件上引入硬件高速缓存Cache，加速对内存的访问。编译器可能优化读取和存储，把变量从内存读进缓冲区，当其他线程修改了在内存中的此变量，但是本进程都是从高速缓存中读取此变量，无法得知此变量已经被修改。

C语言关键字volatile表明某个变量的值可能在外部被改变，因此对这些变量的存取不能缓存到寄存器，每次使用时需要重新从内存存取。该关键字在多线程环境下经常使用，因为在编写

 一般说来，volatile用在如下的几个地方：  
1、中断服务程序中修改的供其它程序检测的变量需要加volatile；  
2、多线程的程序时，同一个变量可能被多个线程修改，而程序通过该变量同步各个线程。  
3、存储器映射的硬件寄存器通常也要加volatile说明，因为每次对它的读写都可能由不同意义；

extern

引用另外一个文件的全局变量和函数，当在本文件中检测到extern，意味着此变量或函数在其他文件中，会在其他文件中找此变量或函数。如果本文件也有相同名字的函数和变量，会执行此文件的，不会执行其他文件的。

结构体

（1）

struct tag {

成员；

}；

struct tag x；

（2）

typedef struct{

成员；

}TAG；

TAG x；

定义结构体的两种形式

无自引用时建议的

定义方法：

结构体的自引用

结构体的成员不能含有本结构体类型的变量，可以含有指向本结构体类型变量的指针；

此时声明变量**必须含有标签**，如下

typedef struct tag {

int data;

struct tag \*node;

}TAG;

TAG x;

struct tag {

int data;

struct tag \*node;

};

struct tag x;

结构体赋值

typedef struct tag {

int data;

char str[3];

double \*d;

struct tag \*t;

}Tag;

（1）定义时变量时赋值

Tag x = { 1, "ha", &s,&zz};

其他形式都是错误；

（2）定义变量后赋值

Tag k;

k.data = 2;

//k.str[3] = "ai";//error;

strcpy(k.str, "ai");

//或 char tmp[3] = “ai”,一个一个的复制到str[3]里面去

k.d = &s;

k.t = &(zz);

定以变量后不能这样赋值

Tag x; x = { 1, "ha", &s,&zz};//错误！！！必须分开逐个赋值。

结构体之间相互赋值

Tag x = { 1, "ha", &s,&zz}; Tag y = x;

判断结构体之间是否相等？

if(x == y) 错误，不能这样判断

结构体的存储方式

struct C {

char a;

int b;

char c;

};

sizeof(C) = 12

struct C {

int a;

char b;

char c;

};

sizeof(C) = 8

struct B{  
  long a1;  
  short a2;  
};

sizeof(B) = 8

struct A{  
   short a1;  
   short a2;  
   short a3;  
};

sizeof(A) = 6

求结构体所占总的字节数方法：

1. 数组不是整体，当做n个独立单元看待。
2. 找到占字节数最多（假设是k个字节）的成员。
3. k个单元一个区
4. 从上往下分配成员，紧挨着分，如果一个成员踩住区的分界线，则不要这个区了，从新区开始。
5. 所占字节总数是k的整数倍，不够补齐。

struct A{  
     int a;  
     double b;  
      float c;  
};

sizeof（A） = 24

sizeof（B） = 48

sizeof不适用于结构体！

struct B{  
   char e[2];  
   int f;  
   double g;     
   short h;  
    struct A i;

};

位段

位段的优点是节省存储空间，缺点是移植性变差。

short char int 最好显示unsigned或signed。

使用位域的主要目的是压缩存储，其大致规则为：    
1) 如果相邻位域字段的类型相同，且其位宽之和小于类型的sizeof大小，则后面的字段将紧邻前一个字段存储，直到不能容纳为止；    
2) 如果相邻位域字段的类型相同，但其位宽之和大于类型的sizeof大小，则后面的字段将从新的存储单元开始，其偏移量为其类型大小的整数倍；    
3) 如果相邻的位域字段的类型不同，则各编译器的具体实现有差异，VC6采取不压缩方式，Dev-C++采取压缩方式；    
4) 如果位域字段之间穿插着非位域字段，则不进行压缩；    
5) 整个结构体的总大小为最宽基本类型成员大小的整数倍。

struct A{    
    char f1 : 3;    
    char f2 : 4;    
    char f3 : 5;    
 };

因此sizeof(A)的结果为2。

struct B{    
   char f1 : 3;    
   short f2 : 4;    
   char f3 : 5;    
 };

由于相邻位域类型不同，在VC6中其sizeof为6，在Dev-C++中为2。

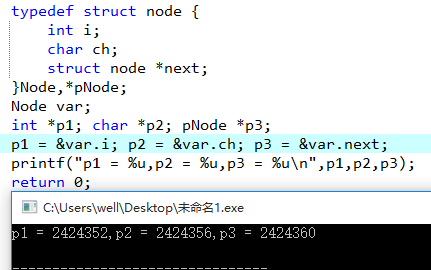
struct C{    
  char f1 : 3;    
  char f2;    
  char f3 : 5;    
};

非位域字段穿插在其中，不会产生压缩，在VC6和Dev-C++中得到的大小均为3。

结构体

尽量向函数传递结构体指针，而不是结构体。函数返回同理。

结构体的成员可以类比数组，单个成员也是变量，可以被取地址，也可以当实参传递给函数的形参，具有普通变量的性质。



函数

函数声明的形参名是无用的，只写形参类型即可，无论定义还是声明函数时，若函数无形参，建议 函数类型 函数名（void），带上void，显示说明函数无形参。

函数体执行前，要先检查形参值的合法性。

C99标准明确规定main函数的返回值类型一定是int，main函数标准定义形式：

int main（void）

说明：函数执行时，为形参分配存储单元；函数从外界接收的一定只是一个常量值，

把这常量值赋给存储单元，让形参变量在函数体中运转。

int f(x,y){

......;

}

printf("f(k++, ++s) = %d", f(k++, ++s));

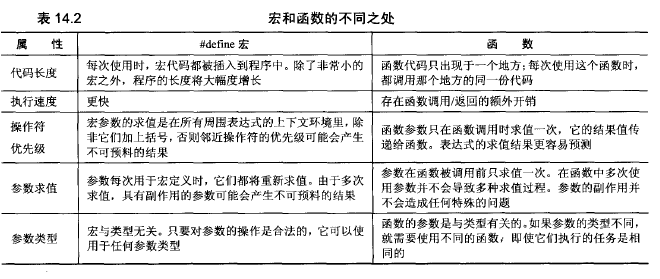
对于函数f，形参是x和y，【k不是】，只是把k和s的值传给x和y；

可拆成：

++s;

printf("f(k++, ++s) = %d", f(k,s));

k++;



待整理

表达式 = 操作符 + 操作数

语句 = 表达式 + ;

多个语句实现一定的功能。

每个表达式都有值和类型。

“Hello World！”，字符串常量，值是首地址，即'H'的地址，类型是const char\* 。

赋值表达式往往有类型转换，赋值时，编译器先把右值类型转换成左值类型，再赋值。

隐式类型转换，较少字节类型转换成较多字节类型，编译器不会警告。

较多字节类型转换成较少字节类型，有可能溢出，编译器会发出警告。

short int let = 6；（有类型转换，因为6默认的是 int 类型）

取一个小数的整数部分用类型转换实现。

强制类型转换！

数据运算中的类型转换（+ - \* / %）

unsigned | signed char , unsigned | signed short --->int ---->unsigned int ---->long

---->unsigned long --->long long ----> unsigned long long --->double --->long double

算数运算时，所有的float转换成double

**运算符两侧的操作数，低级别的数据类型转换成较高级别的数据类型再进行运算，运算结果是高级别数据类型。**

例题：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

float a = 10.0/3;

printf("%f",a);//输出3.333333

float a = 10/3;

printf("%f",a);//输出3.000000

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

C语言运算符优先级和结合性

大致优先级：（），->，.，[ ]， 大于 单目运算符 大于 算数运算符 大于 关系运算符 大于逻辑运算符 大于 赋值运算符

一句话总结自增自减运算符【在执行表达式所在语句前或后把自增或自减运算符的操作数加1或减1】

数据类型

整型 浮点型 指针 聚合类型

char float

数组

signed char unsigned float

结构体

unsigned char

double

union

short int unsigned

double

unsigned

short int

long double

int unsigned

long double

unsigned int

小数

long int 科学计数法的特大整数

unsigned long int

整型

sizeof(short int) <= sizeof(int) <= sizeof(long int)

char 1个字节 0 -- 127 signed char 1个字节 -128 -- 127

unsigned char 0 -- 255

理解 3u 8ul 10LL 056 0x56 的意思！

实际应用中，有些变量一定是正值，用unsigned可充分利用数据类型范围

unsigned和signed只能用于整型

无符号整型数据输出用u%

缺省情况下，整型字面值默认是最短类型但能完全容纳这个值。 枚举类型

enum Weekday

{

Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday

};

Weekday today;

today = Saturday;

today只能取花括号的某个值，且花括号里面都是整型字符常量；

部分初始化，则未初始化的值比它前面初始化的大1

都为初始化，则是0，1，2，3 ...... 浮点型

缺省情况下，浮点字面值默认是double。

小数部分占得位越多，数的有效数字也越多，精度也越大

指数部分占得位越多，表示的范围越大

float的有效数字是6或7位，第7位不一定有效，前6位一定有效//具体翻书 下面解释有效数字是6位的意思：

float f = 3.123456

printf("%f",f);

输出结果可靠数据是前6位，即，3.12345

float f = 123456.789

printf("%.2f",f);

因为要求输出结果保留两位，所以输出结果小数点后会有两位，但是！这两位小数保留的毫无意义，因为它不可靠，可靠的6位有效数字已经被整数部分的123456占完了。

根据以上，楼主可以自行给 f 赋各种不同值，多次调整整数部分与小数部分的位数和大小，查看输出，相信你会发现，输出结果，从左数，只有前6位是一定准确的！

如例1：

#include<stdio.h>

int main () {

float f = 1234560.991f;

printf("%f",f);

return 0;

}

输出结果是：1234561.000000

例2：

#include<stdio.h>

int main () {

float f = 1.913757f;

printf("%.10f",f);

return 0;

}

输出结果是1.9137569666，对比，只有5位小数是精确的，答案还是6位。

综上，所以精确度是6位，意思是前6位一定精确，6位以后的可能精确，可能不精确。

常量

1,3，-99,3000 默认是int类型。288L 长整型 288LL 长长整型

3.1415,1.2E3，浮点型。默认是double。 3.14f 单精度

'a'，‘1’，'\n' 字符常量

"BOY" 字符串常量

#define ok 1 符号常量 const 常变量用法

数据类型 const a = 右值；

a是只读变量，不可修改。

难点

int const \*p； 可以修改指针的值，不可以修改指针指向的值。

int \* const p； 可以修改指针指向的值，不可以修改指针的值。

int const \* const p； 二者都不能修改。

1. 函数外定义的变量，作用域是定义处到本源文件结束，存放在静态存储区，在程序开始执行时分配存储区，程序运行完毕后就释放。

2. 可以【 extern 函数外定义的变量】进行声明，把函数外定义的变量的在本文件的作用域扩展到定义处之上。提倡把外部变量定义在所有引用它的函数之前，这样可以避免使用 extern。在另一个文件中声明【 extern 函数外定义的变量】，也可以使函数外定义的变量扩展到另一个文件中。

3. 不建议使用函数外定义的变量：由于在各个函数执行时都可能改变函数外定义的变量的值，程序容易出错。

4. 【 static 函数外定义的变量】把函数外定义的变量的作用域限制在本文件中，即使其他文件有【 extern 函数外定义的变量】也不能引用本文件的函数外定义的变量，有利于程序员模块化工作。

5. 函数形参，函数体语句块外，函数体语句块中定义的变量函数内定义的变量，作用域分别是本函数，定义处到函数结束，定义处到语句块结束。

6. 函数内定义的变量只能在本函数内引用他它们，在次函数外是不能使用这些变量的。作用域外，无法通过变量名访问他们，可以给不同作用域的不同变量起同一个名字嵌套语句块标识符名最好不要重叠。

7. 在调用函数时，为形参和语句块外变量分配动态存储区间，函数结束时释放。在调用函数运行到语句块时，为语句块内变量分配动态存储空间，语句块运行完毕后立即释放。

8. statis 函数内定义的变量，存储在静态存储区，在函数调用结束后，存储单元不释放，继续保留原值，在下一次再调用该函数时，该变量已经有值，即，只定义初始化一次。注意：虽然静态函数内定义的变量，在函数运行结束后仍旧存在，但是仍不能被其他函数引用。函数内定义的变量只能被本函数引用。

9. statis 类型名 函数名（形参表）；内部函数，该函数只能在本文件内使用，其他文件不能使用。

UNION

**一、前言**

“联合体”(union）与“结构体”(struct)有一些相似之处。但两者有本质上的不同。在结构体中,各成员有各自的内存空间, 一个结构变量的总长度是各成员长度之和。而在“联合”中,各成员共享一段内存空间, 一个联合变量的长度等于各成员中最长的长度。应该说明的是, 这里所谓的共享不是指把多个成员同时装入一个联合变量内, 而是指该联合变量可被赋予任一成员值,但每次只能赋一种值, 赋入新值则冲去旧值。

一个联合体类型必须经过定义之后, 才能使用它，才能把一个变量声明定义为该联合体类型。

关于结构体的更多内容请参考《[**结构体struct简介**](http://hubingforever.blog.163.com/blog/static/1710405792012223112153850/)》

**二、联合体的定义**

定义一个联合类型的一般形式为:

union 联合名

{

成员表

};

成员表中含有若干成员,成员的一般形式为: 类型说明符 成员名。成员名的命名应符合标识符的规定。

例如:

union perdata

{

   int class;

   char office[10];

};

　　定义了一个名为perdata的联合类型,它含有两个成员,一个为整型,成员名为class;另一个为字符数组,数组名为office。联合定义之后,即可进行联合变量说明,被说明为perdata类型的变量,可以存放整型量class或存放字符数组office。

**三、联合体变量的声明**

　　联合变量的说明和结构变量的声明方式相同, 也有三种形式。第一种，先定义联合体类型,再声明联合体变量;第二种，定义联合体类型的同时声明联合体变量；第三种，直接声明联合体（以匿名的形式定义联合体类型）。

以perdata类型为例,说明如下:

第一种，先定义联合体类型,再声明联合体变量

union perdata

{

   int class;

   char officae[10];

};

union perdata a,b; /\*说明a,b为perdata类型\*/

第二种，定义联合体类型的同时声明联合体变量:

union perdata

{

int class;

char office[10];

}a,b;

第三种，直接声明联合体（以匿名的形式定义联合体类型）:

union

{

int class;

char office[10];

}a,b

经说明后的a,b变量均为perdata类型。a,b变量的长度应等于 perdata 的成员中最长的长度, 即等于office数组的长度,共10个字节。对a,b变量如赋予整型值时,只使用了2个字节,而赋予字符数组时,可用10个字节。

**四、联合体变量的赋值和使用**

　　对联合体变量的赋值,使用都只能是对变量的成员进行。

联合变量的成员表示为: 联合变量名**.**成员名

例如,a被说明为perdata类型的变量之后,可使用 a.class，a.office

不允许只用联合变量名作赋值或其它操作。 也不允许对联合变量作初始化赋值,赋值只能在程序中进行。

一个联合体变量, 每次只能赋予一个成员值。一个联合变量的值就是联合变员的某一个成员值。

[例4.1]设有一个教师与学生通用的表格,教师数据有姓名,年龄,职业,教研室四项。学生有姓名,年龄,职业,班级四项。

编程输入人员数据, 再以表格输出。

main()

{

   struct

   {

      char name[10];

      int age;

      char job;

      union

      {

         int class;

         char office[10];

      } depa;

   }body[2];

   int n,i;

   for(i=0;i<2;i++)

   {

      printf("input name,age,job and department\n");

      scanf("%s %d %c",body[i].name,&body[i].age,&body[i].job);

      if(body[i].job=='s')

         scanf("%d",&body[i].depa.class);

      else

         scanf("%s",body[i].depa.office);

   }

   printf("name\tage job class/office\n");

   for(i=0;i<2;i++)

   {

   if(body[i].job=='s')

      printf("%s\t%3d %3c %d\n",body[i].name,body[i].age ,body[i].job,body[i].depa.class);

   else

      printf("%s\t%3d %3c %s\n",body[i].name,body[i].age, body[i].job,body[i].depa.office);

   }

}

　　本例程序用一个结构数组body来存放人员数据, 该结构共有四个成员。其中成员项depa是一个联合类型, 这个联合又由两个成员组成,一个为整型量class,一个为字符数组office。在程序的第一个for语句中,输入人员的各项数据,先输入结构的前三个成员name,age和job,然后判别job成员项,如为"s"则对联合depa·class输入(对学生赋班级编号)否则对depa·office输入(对教师赋教研组名)。

　　在用scanf语句输入时要注意,凡为数组类型的成员,无论是结构成员还是联合成员,在该项前不能再加"&"运算符。如程序第18行中

body.name是一个数组类型,第22行中的body.depa.office也是数组类型,因此在这两项之间不能加"&"运算符。程序中的第二个for语句用于输出各成员项的值: