一、printf()函数  
printf()函数是格式化输出函数, 一般用于向标准输出设备按规定格式输出  
信息。在编写程序时经常会用到此函数。printf()函数的调用格式为:  
printf(”<格式化字符串>”, <参量表>);  
其中格式化字符串包括两部分内容: 一部分是正常字符, 这些字符将按原  
样输出; 另一部分是格式化规定字符, 以”%”开始, 后跟一个或几个规定字符,  
用来确定输出内容格式。  
参量表是需要输出的一系列参数, 其个数必须与格式化字符串所说明的输出  
参数个数一样多, 各参数之间用”,”分开, 且顺序一一对应, 否则将会出现意想  
不到的错误。  
1. 格式化规定符  
Turbo C2.0提供的格式化规定符如下:  
━━━━━━━━━━━━━━━━━━━━━━━━━━  
符号 作用  
——————————————————————————  
%d 十进制有符号整数  
%u 十进制无符号整数  
%f 浮点数  
%s 字符串  
%c 单个字符  
%p 指针的值  
%e 指数形式的浮点数  
%x, %X 无符号以十六进制表示的整数  
%0 无符号以八进制表示的整数  
%g 自动选择合适的表示法  
━━━━━━━━━━━━━━━━━━━━━━━━━━  
说明:  
(1). 可以在”%”和字母之间插进数字表示最大场宽。  
例如: %3d 表示输出3位整型数, 不够3位右对齐。  
%9.2f 表示输出场宽为9的浮点数, 其中小数位为2, 整数位为6,  
小数点占一位, 不够9位右对齐。  
%8s 表示输出8个字符的字符串, 不够8个字符右对齐。  
如果字符串的长度、或整型数位数超过说明的场宽, 将按其实际长度输出。  
但对浮点数, 若整数部分位数超过了说明的整数位宽度, 将按实际整数位输出;  
若小数部分位数超过了说明的小数位宽度, 则按说明的宽度以四舍五入输出。  
另外, 若想在输出值前加一些0, 就应在场宽项前加个0。  
例如: %04d 表示在输出一个小于4位的数值时, 将在前面补0使其总宽度  
为4位。  
如果用浮点数表示字符或整型量的输出格式, 小数点后的数字代表最大宽度,  
小数点前的数字代表最小宽度。  
例如: %6.9s 表示显示一个长度不小于6且不大于9的字符串。若大于9, 则  
第9个字符以后的内容将被删除。  
(2). 可以在”%”和字母之间加小写字母l, 表示输出的是长型数。  
例如: %ld 表示输出long整数  
%lf 表示输出double浮点数  
(3). 可以控制输出左对齐或右对齐, 即在”%”和字母之间加入一个”-” 号可  
说明输出为左对齐, 否则为右对齐。  
例如: %-7d 表示输出7位整数左对齐  
%-10s 表示输出10个字符左对齐  
2. 一些特殊规定字符  
━━━━━━━━━━━━━━━━━━━━━━━━━━  
字符 作用  
——————————————————————————  
\n 换行  
\f 清屏并换页  
\r 回车  
\t Tab符  
\xhh 表示一个ASCII码用16进表示,  
其中hh是1到2个16进制数  
━━━━━━━━━━━━━━━━━━━━━━━━━━  
例1  
#include<stdio.h>  
#include<string.h>  
int main()  
{  
char c, s[20], \*p;  
int a=1234, \*i;  
float f=3.141592653589;  
double x=0.12345678987654321;  
p=”How do you do”;  
strcpy(s, “Hello, Comrade”);  
\*i=12;  
c=’\x41′;  
printf(”a=%d\n”, a); /\*结果输出十进制整数a=1234\*/  
printf(”a=%6d\n”, a); /\*结果输出6位十进制数a= 1234\*/  
printf(”a=%06d\n”, a); /\*结果输出6位十进制数a=001234\*/  
printf(”a=%2d\n”, a); /\*a超过2位, 按实际值输出a=1234\*/  
printf(”\*i=%4d\n”, \*i); /\*输出4位十进制整数\*i= 12\*/  
printf(”\*i=%-4d\n”, \*i); /\*输出左对齐4位十进制整数\*i=12\*/  
printf(”i=%p\n”, i); /\*输出地址i=06E4\*/  
printf(”f=%f\n”, f); /\*输出浮点数f=3.141593\*/  
printf(”f=6.4f\n”, f); /\*输出6位其中小数点后4位的浮点数  
f=3.1416\*/  
printf(”x=%lf\n”, x); /\*输出长浮点数x=0.123457\*/  
printf(”x=%18.16lf\n”, x);/\*输出18位其中小数点后16位的长浮点  
数x=0.1234567898765432\*/  
printf(”c=%c\n”, c); /\*输出字符c=A\*/  
printf(”c=%x\n”, c); /\*输出字符的ASCII码值c=41\*/  
printf(”s[]=%s\n”, s); /\*输出数组字符串s[]=Hello, Comrade\*/  
printf(”s[]=%6.9s\n”, s);/\*输出最多9个字符的字符串s[]=Hello,  
Co\*/  
printf(”s=%p\n”, s); /\*输出数组字符串首字符地址s=FFBE\*/  
printf(”\*p=%s\n”, p); /\* 输出指针字符串p=How do you do\*/  
printf(”p=%p\n”, p); /\*输出指针的值p=0194\*/  
getch();  
retunr 0;  
}

二、scanf()函数  
scanf()函数是格式化输入函数, 它从标准输入设备(键盘) 读取输入的信息。  
其调用格式为:  
scanf(”<格式化字符串>”, <地址表>);  
格式化字符串包括以下三类不同的字符;  
1. 格式化说明符: 格式化说明符与printf()函数中的格式说明符基本相同。  
2. 空白字符: 空白字符会使scanf()函数在读操作中略去输入中的一个或多  
个空白字符。  
3. 非空白字符: 一个非空白字符会使scanf()函数在读入时剔除掉与这个非  
空白字符相同的字符。  
地址表是需要读入的所有变量的地址, 而不是变量本身。这与printf()函数  
完全不同, 要特别注意。各个变量的地址之间同”,”分开。  
例2:  
main()  
{  
int i, j;  
printf(”i, j=?\n”);  
scanf(”%d, %d”, &i, &j);  
}

上例中的scanf()函数先读一个整型数, 然后把接着输入的逗号剔除掉, 最  
后读入另一个整型数。如果”,”这一特定字符没有找到, scanf()函数就终止。若  
参数之间的分隔符为空格, 则参数之间必须输入一个或多个空格。  
说明:  
(1). 对于字符串数组或字符串指针变量, 由于数组名和指针变量名本身就  
是地址, 因此使用scanf()函数时, 不需要在它们前面加上”&”操作符。  
例3  
mian()  
{  
char \*p, str[20];  
scanf(”%s”, p); /\*从健盘输入字符串\*/  
scanf(”%s”, str);  
printf(”%s\n”, p); /\*向屏幕输出字符串\*/  
printf(”%s\n”, str);  
}

(2). 可以在格式化字符串中的”%”各格式化规定符之间加入一个整数, 表示  
任何读操作中的最大位数。  
如例3中若规定只能输入10字符给字符串指针p, 则第一条scanf() 函数语句  
变为  
scanf(”%10s”, p);  
程序运行时一旦输入字符个数大于10, p就不再继续读入, 而后面的一个读  
入函数即scanf(”%s”, str)就会从第11个字符开始读入。  
实际使用scanf()函数时存在一个问题, 下面举例进行说明:  
当使用多个scanf()函数连续给多个字符变量输入时, 例如:  
main()  
{  
char c1, c2;  
scanf(”%c”, &c1);  
scanf(”%c”, &c2);  
printf(”c1 is %c, c2 is %c”, c2\1, c2);  
}

运行该程序, 输入一个字符A后回车 (要完成输入必须回车), 在执行scanf  
(”%c”, &c1)时, 给变量c1赋值”A”, 但回车符仍然留在缓冲区内, 执行输入语句  
scanf(”%c”, &c2)时, 变量c2输出的是一空行, 如果输入AB后回车, 那么输出结  
果为: c1 is A, c2 is B。  
要解决以上问题, 可以在输入函数前加入清除函数fflush()修改以上程序变成:  
#include<stdio.h>  
main()  
{  
char c1, c2;  
scanf(”%c”, &c1);  
fflush(stdin);  
scanf(”%c”, &c2);  
printf(”c1 is %c, c2 is %c”, c1, c2);  
}

1.1.2 非格式化输入输出函数  
非格式化输入输出函数可以由上面讲述的标准格式化输入输出函数代替, 但  
这些函数编译后代码少, 相对占用内存也小, 从而提高了速度, 同时使用也比较  
方便。下面分别进行介绍。  
一、puts()和gets()函数  
1. puts()函数  
puts()函数用来向标准输出设备(屏幕)写字符串并换行, 其调用格式为:  
puts(s);  
其中s为字符串变量(字符串数组名或字符串指针)。  
puts()函数的作用与语printf(”%s\n”, s)相同。  
例4:  
main()  
{  
char s[20], \*f; /\*定义字符串数组和指针变量\*/  
strcpy(s, “Hello! Turbo C2.0″); /\*字符串数组变量赋值\*/  
f=”Thank you”; /\*字符串指针变量赋值\*/  
puts(s);  
puts(f);  
}

说明:  
(1). puts()函数只能输出字符串, 不能输出数值或进行格式变换。  
(2). 可以将字符串直接写入puts()函数中。如:  
puts(”Hello, Turbo C2.0″);

2. gets()函数  
gets()函数用来从标准输入设备(键盘)读取字符串直到回车结束, 但回车符  
不属于这个字符串。其调用格式为:  
gets(s);  
其中s为字符串变量(字符串数组名或字符串指针)。  
gets(s)函数与scanf(”%s”, &s)相似, 但不完全相同, 使用scanf(”%s”, &s)  
函数输入字符串时存在一个问题, 就是如果输入了空格会认为输入字符串结束,  
空格后的字符将作为下一个输入项处理, 但gets() 函数将接收输入的整个字符  
串直到回车为止。  
例5  
main()  
{  
char s[20], \*f;  
printf(”What’s your name?\n”);  
gets(s); /\*等待输入字符串直到回车结束\*/  
puts(s); /\*将输入的字符串输出\*/  
puts(”How old are you?”);  
gets(f);  
puts(f);  
}  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*c++\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
cin与cout   
一：标准输入函数cin   
    不知道说它是个函数对还是不对，它是代表标准的输入设备--键盘。他是属于流的，他的用法和流的用法是一样的。也就是：cin>>变量;   
小小的说明一下，输入多个变量可以写在一行，如:cin>>x>>y>>z;   
这样写不是不允许，而是不好看，如果是不同的变量类型，那就更是没头没脑了。除了你，人家是不知道该输入什么的，所以，一般在输入语句的前面，我们一般都   
要做一个提示，请输入×××，让人家心里有个底，知道这个变量是做什么的。   
    另外，这个函数是不用带地址符号"&"的，也不用写明变量类型，千万不要跟scanf混淆。当然他就也不检查变量输入是否合法。如：   
int i;   
cout<<"please input a number:"   
cin>>i;   
cout<<"i="<<i<<endl;   
如果你输入的是一个字符如’a’那么他也不检查，但你输出的结果不是正确的，这要是手工进行检查。当然他也跟scanf一样，如果在循环内部输入不合法的变量值，那么也将陷入死循环。如下：   
/\*一个输入不合法变量陷入死循环的例子\*/   
#include <iostream.h>   
main()   
{   
     int i;   
     while(i!=-1)   
     {   
         cout<<"i="   
         cin>>i;    /\*请输入不是一个字符如’a’试试\*/   
         cout<<endl;   
     }   
}   
    如上一个程序，如果你输入的不合法，那就将陷入死循环。解决的办法有个一，把cin>>i;语句移到判断循环的语句中，那么，你输入的如果是不合法的变量，他将跳出循环。   
    cin是用空格来分隔输入的。请看看如下的例子：   
/\*一个空格分隔使输入的变量达不到希望的值\*/   
#include <iostream.h>   
main()   
{   
     char str[20];   
     cout<<"please input a string:";   
     cin>>str;    /\*你试着输入"hello word"\*/   
     cout<<endl<<"str="<<str;   
}   
看得到是什么结果呢？得到的仅仅是str=hello，为什么呢？因为cin是以空格为分隔的，当你输入一个空格时，那他就认为后面的输入不属于这里了，   
认为应该给后面的变量了。另外，当你输入的字符串大于分配的空间时，还会出现溢出现象。当然，还有整行输入的函数，包括空格也一起输入了，以后也会学到。   
二、标准输出函数cout

/\*一个按进制输出的例子\*/   
#include<iostream.h>   
void main()   
{   
    int x=30, y=300, z=1024;   
    cout<<x<<’ ’<<y<<’ ’<<z<<endl;                  //按十进制输出   
    cout.setf(ios::showbase | ios::uppercase);   //设置基指示符输出和数值中的字母大写输出   
    cout<<x<<’ ’<<y<<’ ’<<z<<endl;   
    cout.unsetf(ios::showbase | ios::uppercase); //取消基指示符输出和数值中的字母大写输出   
    cout.setf(ios::oct);                                    //设置为八进制输出,此设置不取消一直有效   
    cout<<x<<’ ’<<y<<’ ’<<z<<endl;                   //按八进制输出   
    cout.setf(ios::showbase | ios::uppercase);    //设置基指示符输出和数值中的字母大写输出   
    cout<<x<<’ ’<<y<<’ ’<<z<<endl;   
    cout.unsetf(ios::showbase | ios::uppercase); //取消基指示符输出和数值中的字母大写输出   
    cout.unsetf(ios::oct);                                 //取消八进制输出设置，恢复按十进制输出   
    cout.setf(ios::hex);                                    //设置为十六进制输出   
    cout<<x<<’ ’<<y<<’ ’<<z<<endl;   
    cout.setf(ios::showbase | ios::uppercase); //设置基指示符输出和数值中的字母大写输出   
    cout<<x<<’ ’<<y<<’ ’<<z<<endl;   
    cout.unsetf(ios::showbase | ios::uppercase); //取消基指示符输出和数值中的字母大写输出   
    cout.unsetf(ios::hex);                       //取消十六进制输出设置，恢复按十进制输出   
    cout<<x<<’ ’<<y<<’ ’<<z<<endl;   
}   
我们用cout.setf()设置输出的格式，用cout.unsetf()取消格式。可以看出10进制在输出的时候不管有没有设置基指示符ios::   
showbase，都没用，8进制再输出的时候在前面加0，而16进制是在前面加0X。而对于数值中字母大写输出，只对16进制有用，以后我们就应该看情   
况使用了。当然，我们前面已经说了，还有一种方法也可以实现格式化输出，那就是使用操纵算子，如下，   
/\*一个按进制输出的例子\*/   
#include<iomanip.h>   
void main()   
{   
    int x=30, y=300, z=1024;   
    cout<<x<<’ ’<<y<<’ ’<<z<<endl;       //按十进制输出   
    cout<<oct<<x<<’ ’<<y<<’ ’<<z<<endl; //按八进制输出   
    cout<<setiosflags(ios::showbase);    //设置基指示符   
    cout<<x<<’ ’<<y<<’ ’<<z<<endl;       //仍按八进制输出   
    cout<<resetiosflags(ios::showbase); //取消基指示符   
    cout<<hex<<x<<’ ’<<y<<’ ’<<z<<endl; //按十六进制输出   
    cout<<setiosflags(ios::showbase | ios::uppercase);   
                  //设置基指示符和数值中的字母大写输出，   
    cout<<x<<’ ’<<y<<’ ’<<z<<endl; //仍按十六进制输出   
    cout<<resetiosflags(ios::showbase | ios::uppercase);   
                  //取消基指示符和数值中的字母大写输出   
    cout<<x<<’ ’<<y<<’ ’<<z<<endl; //仍按十六进制输出   
    cout<<dec<<x<<’ ’<<y<<’ ’<<z<<endl; //按十进制输出   
}

/\*一个使用填充，宽度，对齐方式的例子\*/   
#include <iostream.h>   
void main()   
{   
    cout<<"第一章"<<endl;   
    cout<<"    ";   
    cout.setf(ios::left);        //设置对齐方式为left   
    cout.width(7);               //设置宽度为7，不足用空格填充   
    cout<<"1.1";   
    cout<<"什么是C语言";   
    cout.unsetf(ios::left);      //取消对齐方式，用缺省right方式   
    cout.fill(’.’);              //设置填充方式   
    cout.width(30);              //设置宽度，只对下条输出有用   
    cout<<1<<endl;   
    cout<<"    ";                   
    cout.width(7);               //设置宽度   
    cout.setf(ios::left);        //设置对齐方式为left   
    cout.fill(’ ’);              //设置填充，缺省为空格   
    cout<<"1.11";   
    cout<<"C语言的历史";   
    cout.unsetf(ios::left);      //取消对齐方式   
    cout.fill(’.’);   
    cout.width(30);   
    cout<<58<<endl;   
    cout.fill(’ ’);   
    cout<<"第二章"<<endl;   
}   
    我们多次设置了宽度，为的是使我们的间距能一致，也使用了对齐方式，为的是使我们的数据能对齐显示，看起来美观。我们还使用了填充方式。我们下面用操纵算子来实现也是可以的。   
/\*一个使用填充，宽度，对齐方式的例子\*/   
#include <iomanip.h>   
void main()   
{   
    cout<<"第一章"<<endl;   
    cout<<"    ";   
    cout<<setiosflags(ios::left)<<setw(7);         //设置宽度为7，left对齐方式   
    cout<<"1.1";   
    cout<<"什么是C语言";   
    cout<<resetiosflags(ios::left);                //取消对齐方式   
    cout<<setfill(’.’)<<setw(30)<<1<<endl;         //宽度为30，填充为’.’输出   
    cout<<setfill(’ ’);                            //恢复填充为空格   
    cout<<"    ";   
    cout<<setw(7)<<setiosflags(ios::left);         //设置宽度为7，left对齐方式   
    cout<<"1.11";   
    cout<<"C语言的历史";   
    cout<<resetiosflags(ios::left);                //取消对齐方式   
    cout<<setfill(’.’)<<setw(30)<<58<<endl;        //宽度为30，填充为’.’输出         
    cout<<setfill(’ ’)<<"第二章"<<endl;   
}   
    我们输出了同样的效果，不过依我的性格，我更喜欢用操纵算子来进行格式化输出。最后我们看看浮点数的格式输出，如下例：   
/\*关于浮点数的格式\*/   
#include <iostream.h>   
void main()   
{   
    float f=2.0/3.0,f1=0.000000001,f2=-9.9;   
    cout<<f<<’ ’<<f1<<’ ’<<f2<<endl;      //正常输出   
    cout.setf(ios::showpos);              //强制在正数前加+号   
    cout<<f<<’ ’<<f1<<’ ’<<f2<<endl;   
    cout.unsetf(ios::showpos);            //取消正数前加+号   
    cout.setf(ios::showpoint);            //强制显示小数点后的无效0   
    cout<<f<<’ ’<<f1<<’ ’<<f2<<endl;   
    cout.unsetf(ios::showpoint);          //取消显示小数点后的无效0   
    cout.setf(ios::scientific);           //科学记数法   
    cout<<f<<’ ’<<f1<<’ ’<<f2<<endl;   
    cout.unsetf(ios::scientific);         //取消科学记数法   
    cout.setf(ios::fixed);                //按点输出显示   
    cout<<f<<’ ’<<f1<<’ ’<<f2<<endl;   
    cout.unsetf(ios::fixed);              //取消按点输出显示   
    cout.precision(18);                   //精度为18，正常为6   
    cout<<f<<’ ’<<f1<<’ ’<<f2<<endl;   
    cout.precision(6);                    //精度恢复为6   
}   
    同样，我们也一样能用操纵算子实现同样的功能：   
/\*关于浮点数的格式\*/   
#include <iomanip.h>   
void main()   
{   
    float f=2.0/3.0,f1=0.000000001,f2=-9.9;   
    cout<<f<<’ ’<<f1<<’ ’<<f2<<endl;      //正常输出   
    cout<<setiosflags(ios::showpos);      //强制在正数前加+号   
    cout<<f<<’ ’<<f1<<’ ’<<f2<<endl;   
    cout<<resetiosflags(ios::showpos);    //取消正数前加+号   
    cout<<setiosflags(ios::showpoint);    //强制显示小数点后的无效0   
    cout<<f<<’ ’<<f1<<’ ’<<f2<<endl;   
    cout<<resetiosflags(ios::showpoint); //取消显示小数点后的无效0   
    cout<<setiosflags(ios::scientific);   //科学记数法   
    cout<<f<<’ ’<<f1<<’ ’<<f2<<endl;   
    cout<<resetiosflags(ios::scientific); //取消科学记数法   
    cout<<setiosflags(ios::fixed);        //按点输出显示   
    cout<<f<<’ ’<<f1<<’ ’<<f2<<endl;   
    cout<<resetiosflags(ios::fixed);       //取消按点输出显示   
    cout<<setprecision(18);               //精度为18，正常为6   
    cout<<f<<’ ’<<f1<<’ ’<<f2<<endl;   
    cout<<setprecision(6);                //精度恢复为6   
}   
   在c/c++系统中除了标准的输入输出外，还提供了更多的输入函数。这写函数主要有getch(),getche(),   
getchar   
(),cin.get(),putch(),putchar(),cout.put(),gets(),cin.getline(),puts()。另外   
还有些为了让缓冲区不影响程序的正确操作的缓冲去的操作，如：cin.putback(),fflush(stdin),cout.flush().我们   
做一下简单的说明。   
    1、getch()和getche(),非缓冲式输入，从键盘读入一个字符。getch()读入字符不显示。有conio.h支持。   
    2、cin.get(),getchar()，缓冲式输入，从键盘读入一个字符，并显示。getchar()由stdio.h支持，cin.get()由iostream.h支持。   
    3、putch()和putchar()，非缓冲式输出，输出一个字符到显示器。putch()由conio.h支持，putchar()由stdio.h支持。   
    4、cout.put()，缓冲式输出，输出一个字符到显示器。由iostream.h支持。   
    5、gets()和cin.geline()，缓冲式输入，读入一字符串（包括空格，不包括最后的回车），gets()由stdio.h支持，cin.getline()由iostream.h支持。   
    6、puts()，非缓冲输出，输出一个字符串，由stdio.h支持。   
    7、cin.putback()，把一个字符送回输入缓冲区。   
    8、fflush(stdin)，清除输入缓冲区操作。无法清除cin.get()等带来的操作。   
    9、cout.flush()，清楚输出缓冲区。   
    在这里我们稍微说一下输入/输出缓冲区，这是为了减少程序访问io带来中断而设的一段空间。当程序满足某个刷新条件时，那就将清理缓冲区。具体条件为：   
    1、输入缓冲区   
a，程序要求输入时，按下了回车键。   
b，遇到程序结束。   
c，遇到下一条输入语句。   
d，遇到清除缓冲区操作   
e，缓冲区溢出   
    2、输出缓冲区   
a，输出缓冲区溢出   
b，遇到下一条输入语句   
c，使用刷新缓冲区迫使清除   
d，程序结束。   
    缓冲区操作有时会带来程序的不正确的输入，如前面说的scanf()，在连续输入的时候，会把一个回车赋给下一个字符变量。我们操作的时候一定要注意。