C语言学习笔记

# 函数printf()与getchar();的解析与使用:

printf()就相当于是Java中system.out.println();语句,而

getchar()函数时再使用visual studio c++ 2013时使用本地调试器是让其弹出的输出窗口停住.

# 对#include<stdio.h> 的解释:

#include<stdio.h> //包含的作用,#预处理,stdio.h标准输入输出库

# 如何用记事本编写c语言程序

1. 首先新建一个点c(.c)结尾的文件.
2. 然后在改文件中编写正确的可执行的c程序,完成后保存
3. 最后用visual studio c++ 2013 (我用的是2013版本)中自带的 开发者编译工具将我们编写的代码编译成exe文件即可,操作是:打开该编译工具,该编译工具是一个命令行工具,在该命令行工具中找到我们写的c程序所在的目录,然后输入 编译语句 cl1.c 然后回车即可(1.c是c程序的文件名),对于编译后还生成了一个.obj的文件,我们可以将其删除掉

# 在c程序语言中想要执行window系统中的命令行所要用的头文件

|  |
| --- |
| #include<stdlib.h>  void main(){  system("ipconfig"); //显示ip  system("calc"); //打开计算器  system("pause"); //暂停  } |

|  |
| --- |
| #include<stdlib.h>  void main(){  for (int i = 0; i < 20;i++){  //system("ipconfig"); //显示ip  system("start calc"); //打开计算器//添加了start后表示并行打开,若不加start,则只能在打开一个计算器需要关闭后才能打开第二打开  //system("pause"); //暂停  }    } |

# 关于Windows.h的头文件下的对话框的案例:

|  |
| --- |
| #include<Windows.h>  void main(){  /\*  第一个参数表示为0,表示弹出的是系统对话框  第二个参数表对话框弹出的内容  第三个参数表示对话框的标题  第四个参数表示对话框的类型,0表示alert对话框,1表示confirm对话框,2表示有三个按钮的对话框,3也表示有三个按钮的对话框,但是显示的按钮字不一样  \*/  MessageBoxA(0,"嘿嘿!","提示",0);  } |

# 关于windows.h的头文件下的函数ShellExecuteA()函数的使用

|  |
| --- |
| #include<Windows.h>  void main(){  /\*  第一个参数:0 表示系统启动  第二个参数: "open" 表示打开  第三个参数: 表示要打开的进程命令  第四个参数: 默认为 0  第五个参数: 默认为 0  第六个参数: 0->隐藏打开,1->整除打开,3->最大化打开,6->最小化打开  \*/  ShellExecuteA(0, "open", "notepad", 0, 0, 3);  Sleep(1000); //让线程睡1秒钟  ShellExecuteA(0, "open", "notepad", 0, 0, 1);  } |

# 关于malloc();函数的使用

注意:这个函数甚至不需任何的头文件就可以运行

|  |
| --- |
| void main(){  while (1){  int \*p = (int \*)malloc(10000); //这个函数的意思是吃内存的意思(即分配内存),如果用一个死循环将其不断的执行,电脑会很快死掉  //其实就是每一秒钟分配10k的内存  }  } |

# malloc();函数在哪个头文件下

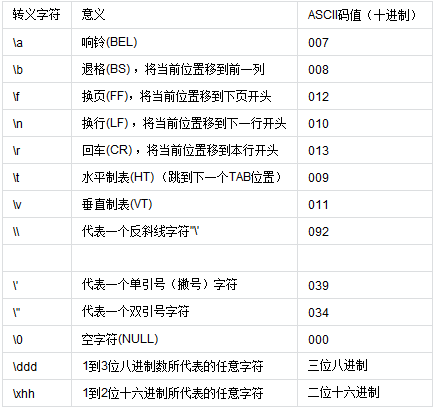
解析:malloc();函数在#include<stdlib.h>这个头文件中

|  |
| --- |
| #include<Windows.h>  #include<stdlib.h>  void main(){  while (1){  malloc(10000);  sleep();  }  } |

# 在c语言中关于printf()与putchar()的使用案例:

|  |
| --- |
| #include<stdio.h>  void main(){  printf("helloword"); //这是双引号,这接收的是一个字符串  putchar('\n');  putchar('h'); //这是单引号,只能接收一个字符  putchar('e');  putchar('l');  putchar('l');  putchar('o');  getchar();  } |

# c语言中 转义字符列表



# 使用八进制出差h字母,使用十六进制输出字母h,使用十进制输出h字母

注意:使用putchar();函数的参数是数字事,前面带0表示是八进制数

使用putchar();函数的参数是数字事,前面带0x表示是十六进制数

使用putchar();函数的参数是数字事,前面不带任何的数字表示十进制数

|  |
| --- |
| #include<stdio.h>  void main(){  //这是使用八进制数出h字母  putchar(0150); //使用putchar();函数的参数是数字时,前面带0表示式八进制数  putchar(0x68); //使用putchar();函数的参数是数字时,前面带0x表示式十六进制数  putchar(104); //使用putchar();函数的参数是数字时,前面不带如何的数字表示十进制数  putchar('\150'); //使用putchar();也可以用单引号将八进制引起来,用斜杠替代第一个0,这样也可以输出h字母  putchar('\x68'); //使用putchar();也可以用单引号将十六进制引起来,用斜杠替代第一个0,这样也可以输出h字母  getchar();  } |

# puts();函数的使用案例:

解析:puts()函数时标准的输出字符串的函数

|  |
| --- |
| #include<stdio.h>  void main(){  puts("helloword");  getchar();  } |

# 关于sprintf();函数的使用案例

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS //关闭安全检查\_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  #include<Windows.h>  void main(){  char str[50];  sprintf(str,"%s%s","note","pad"); //若要是使用这个sprintf();函数,就必须要关闭安全检查,因为这个函数太老了.  //sprintf(C, "%s%s", "A", "B"); 将A与B字符拼接起来付给C字符变量,之所以这样运行是按格式"%s%s"运行的.  printf(str); //向黑屏控制台打印字符  system(str); //执行该字符串命令  system("pause"); //让打印出来的黑屏停住不要关闭  } |

# 使用printf();函数输出变量的十六进制地址,同时输出变量的十进制的值的案例:

|  |
| --- |
| #include<stdio.h>  void main(){  int num = 1;  printf("%x",&num); //按十六进制打印出num变量的地址  printf("\n");  printf("%d", num); //按十进制打印出num变量的值  printf("\n%d", num); //先换行,在按十进制打印出num变量的值  getchar();  } |

# scanf();函数的用法及Windows系统如何用命令来做for循环的用法案例:

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS //当我们的程序中使用了scanf(),sprintf();等函数我们都要加上#define \_CRT\_SECURE\_NO\_WARNINGS 这个头  #include<stdio.h>  #include<stdlib.h>  void main(){  int num;  scanf("%d", &num); //在黑屏控制台中接收字符的输入,并将该字符存放到num变量中  char str[50];  sprintf(str, "for /l %%i in (1,1,%d) do start calc", num);//这是Windows命令行佛如循环的语句,儿 for /l 中的l是英文字母l,不是数字1,记住:C预言中两个%%表示一个%  system(str);  system("pause");  } |

# 关于@echo隐式执行的案例:

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  void main(){  int num;  scanf("%d", &num); //在黑屏控制台中接收字符的输入,并将该字符存放到num变量中  char str[50];  //sprintf(str, "for /l %%i in (1,1,%d) do start calc", num);//这是Windows命令行佛如循环的语句,儿 for /l 中的l是英文字母l,不是数字1,记住:C预言中两个%%表示一个%  sprintf(str,"for /l %%i in (1,1,%d) do @echo china" ,num); //再echo前面加上@,表示隐式的执行,去掉@表示显示的执行,隐式的执行只会看到一个结果,而显示的  //执行表示会看到所有的执行过程  system(str);  system("pause");  } |

# c语言中常量的用法案例:

|  |
| --- |
| #include<stdio.h>  #define X "12"  #define Y 25  void main(){  printf("这个X常量的值为:%s",X);  printf("\n这个Y常量的值为:%d", Y);  const int i = 5; //这也是常量,值不可以修改,但却可以修改内存中的值  printf("\n这是常量i的值:%d",i);  //printf("\n%x",&Y); //当一个常量没有用双引号 "" 将其引用时,是不能取出地址的值,  getchar();  } |

# #define 的高级用法案例

|  |
| --- |
| #include<stdio.h>  #define X "12"  #define Y 25  #define 我来替代main main //这是易语言编程的一种实现方式  #define 我来替代void void  我来替代void 我来替代main(){  printf("这个X常量的值为:%s",X);  printf("\n这个Y常量的值为:%d", Y);  const int i = 5; //这也是常量  printf("\n这是常量i的值:%d",i);  getchar();  } |

# c语言编程中,对于头文件,什么时候用#include<xxx.h>,什么时候用#include”xxx.h”?

解析:当这个头文件是系统自带的时,就用include<xxx.h>,如果这个头文件是我们自己编写的时,我们就用#include”xxx.h”.

# \_declspec(dllexport)的使用案例:

**解析:打成dll文件后,外部可以调用,没有不可以调用**

|  |
| --- |
| #include<stdlib.h>  #include<windows.h>  // \_declspec(dllexport) 外部可以调用，没有不可调用  \_declspec(dllexport) void goA()  {  while (1)  {  MessageBoxA(0, "你在与间谍聊天", "请注意安全", 0);  }  } |

# %d,%f …及printf()函数的使用案例:

|  |
| --- |
| #include<stdio.h>  void main(){  //printf();这个函数的本质就是按照自己方式将变量转换为字符串,不管变量以前是什么类型,一就是说值必须与类型匹配.  printf("%d",1); //%d整数,输出1  printf("\n%f", 1); //%f实数,输出0.0000000  printf("\n%f", 1.0); //%f实数,输出1.0000000  printf("\n%u",-10); //%u无符号整数 ,输出 随机数  printf("\n%x",10); //%x无符号十六进制  printf("\n%o",22); //%o无符号八进制  getchar();  } |

# 用数字表示bool值的案例:

|  |
| --- |
| #include<stdlib.h>  void main(){  //在c语言中如果用数字来表示bool型的值时,0表示false,而非零数字表示true,这非零数字包括正数和负数  0 ? system("calc") : system("notepad"); //打开的noetpad  } |

# int 型随机函数rand();的使用案例:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<Windows.h>  void main(){    for (int i = 0; i < 10;i++){  Sleep(1000);  int r = rand(); //这里获取到是int型的随机整数  printf("\n\n%d", r);  }  getchar();  } |

# c语言格式转换说明符.

|  |
| --- |
| 1．转换说明符  %a(%A) 浮点数、十六进制数字和p - (P - )记数法(C99)  % c 字符  %d 有符号十进制整数  %f 浮点数(包括float和doulbe)  % e(%E) 浮点数指数输出[e - (E - )记数法]  % g(%G) 浮点数不显无意义的零"0"  %i 有符号十进制整数(与%d相同)  % u 无符号十进制整数  %o 八进制整数 e.g. 0123  % x(%X) 十六进制整数0f(0F) e.g. 0x1234  % p 指针  %s 字符串  %% "%"  2．标志  左对齐："-" e.g. "%-20s"  右对齐："+" e.g. "%+20s"  空格：若符号为正，则显示空格，负则显示"-" e.g. "% 6.2f"  #：对c, s, d, u类无影响；对o类，在输出时加前缀o；对x类，在输出时加前缀0x；  对e, g, f 类当结果有小数时才给出小数点。  3．格式字符串（格式）  〔标志〕〔输出最少宽度〕〔．精度〕〔长度〕类型  "％-md" ：左对齐，若m比实际少时，按实际输出。  "%m.ns"：输出m位，取字符串(左起)n位，左补空格，当n>m or m省略时m = n  e.g. "%7.2s" 输入CHINA  输出" CH"  "%m.nf"：输出浮点数，m为宽度，n为小数点右边数位  e.g. "%3.1f" 输入3852.99  输出3853.0  长度：为h短整形量, l为长整形量  printf的格式控制的完整格式：  % -0 m.n l或h 格式字符  下面对组成格式说明的各项加以说明：  ①%：表示格式说明的起始符号，不可缺少。  ② - ：有 - 表示左对齐输出，如省略表示右对齐输出。  ③0：有0表示指定空位填0, 如省略表示指定空位不填。  ④m.n：m指域宽，即对应的输出项在输出设备上所占的字符数。N指精度。用于说明输出的实型数的小数位数。为指定n时，隐含的精度为n = 6位。  ⑤l或h : l对整型指long型，对实型指double型。h用于将整型的格式字符修正为short型。  －－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－  格式字符  格式字符用以指定输出项的数据类型和输出格式。  ①d格式：用来输出十进制整数。有以下几种用法：  %d：按整型数据的实际长度输出。  %md：m为指定的输出字段的宽度。如果数据的位数小于m，则左端补以空格，若大于m，则按实际位数输出。  %ld：输出长整型数据。  ②o格式：以无符号八进制形式输出整数。对长整型可以用"%lo"格式输出。同样也可以指定字段宽度用“%mo”格式输出。  例：  main()  {  int a = -1;  printf("%d, %o", a, a);  }  运行结果： - 1, 177777  程序解析： - 1在内存单元中（以补码形式存放）为(1111111111111111)2，转换为八进制数为(177777)8。  ③x格式：以无符号十六进制形式输出整数。对长整型可以用"%lx"格式输出。同样也可以指定字段宽度用"%mx"格式输出。  ④u格式：以无符号十进制形式输出整数。对长整型可以用"%lu"格式输出。同样也可以指定字段宽度用“%mu”格式输出。  ⑤c格式：输出一个字符。  ⑥s格式：用来输出一个串。有几中用法  %s：例如:printf("%s", "CHINA")输出"CHINA"字符串（不包括双引号）。  %ms：输出的字符串占m列，如字符串本身长度大于m，则突破获m的限制, 将字符串全部输出。若串长小于m，则左补空格。  %-ms：如果串长小于m，则在m列范围内，字符串向左靠，右补空格。  %m.ns：输出占m列，但只取字符串中左端n个字符。这n个字符输出在m列的右侧，左补空格。  %-m.ns：其中m、n含义同上，n个字符输出在m列范围的左侧，右补空格。如果n>m，则自动取n值，即保证n个字符正常输出。  ⑦f格式：用来输出实数（包括单、双精度），以小数形式输出。有以下几种用法：  %f：不指定宽度，整数部分全部输出并输出6位小数。  %m.nf：输出共占m列，其中有n位小数，如数值宽度小于m左端补空格。  %-m.nf：输出共占n列，其中有n位小数，如数值宽度小于m右端补空格。  ⑧e格式：以指数形式输出实数。可用以下形式：  %e：数字部分（又称尾数）输出6位小数，指数部分占5位或4位。  %m.ne和%-m.ne：m、n和” - ”字符含义与前相同。此处n指数据的数字部分的小数位数，m表示整个输出数据所占的宽度。  ⑨g格式：自动选f格式或e格式中较短的一种输出，且不输出无意义的零。  －－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－  关于printf函数的进一步说明：  如果想输出字符"%", 则应该在“格式控制”字符串中用连续两个%表示，如:  printf("%f%%", 1.0 / 3);  输出0.333333%。  －－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－  对于单精度数，使用%f格式符输出时，仅前7位是有效数字，小数6位．  对于双精度数，使用%lf格式符输出时，前16位是有效数字，小数6位．  ######################################拾遗 ########################################  由高手指点  对于m.n的格式还可以用如下方法表示（例）  char ch[20];  printf("%\*.\*s\n", m, n, ch);  前边的\*定义的是总的宽度，后边的定义的是输出的个数。分别对应外面的参数m和n 。我想这种方法的好处是可以在语句之外对参数m和n赋值，从而控制输出格式。  --------------------------------------------------------------------------------  今天(06.6.9)又看到一种输出格式 %n 可以将所输出字符串的长度值赋绐一个变量, 见下例:  int slen;  printf("hello world%n", &slen);  执行后变量被赋值为11。 |

# %ld格式符,srand()函数,time()函数的使用案例:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<Windows.h>  #include<time.h>  void main(){    time\_t t = (long)time(NULL);  printf("\n%ld",t); //这个%ld格式符我也没有用过..  srand((unsigned int)time(&t)); //初始化随机数发生器    int r = rand(); //随机数发生器  printf("\n%d",r);  getchar();  } |

# 关于sizeof(),FLT\_MAX,FLT\_MIN,INT\_MAX,INT\_MIN的使用案例:

|  |
| --- |
| #include<float.h>  #include<limits.h>  void main(){  printf("%d,%d",sizeof(float),sizeof(int));  printf("\n\n%f,%f",FLT\_MAX,FLT\_MIN); //使用FLT\_MAS与FLT\_MIN需要使用#include<float.h>这个头文件  printf("\n\n%f,%.100f", FLT\_MAX, FLT\_MIN); //使用FLT\_MAS与FLT\_MIN需要使用#include<float.h>这个头文件 ,同时%.100f表示该值后面的小数位数为100  printf("\n\n%d,%d",INT\_MAX,INT\_MIN); //使用INT\_MAX与INT\_MIN需要使用#include<limits.h>这个头文件    getchar();  } |

# sqrt();计算平方根

|  |
| --- |
| #include <math.h>  #include <stdio.h>  int main(void)  {     double x = 4.0, result;     result = sqrt(x);     printf("The square root of %lf is %lf\n", x, result);     return 0; } |

# 如何将两个char[]数组字符串连接起来的案例:

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS //关闭安全检查\_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  int main(void)  {  char str1[10] = "note";  char str2[10] = "pad";  char str3[10];  sprintf(str3,"%s%s",str1,str2);  printf("%s",str3);  system(str3);  } |

# c语言中求模运算的的规则:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  int main(void)  {  // c语言中求模只能在整型与char型中进行,其他类型不能够用于求模运算  float i = 9.0;  float j = 3.0;  int k = (int)i%(int)j;  printf("%d",k);    char c1 = 23;  char c2 = 2;  char c3 = c1%c2;  printf("\n\n%d",c3);  getchar();  } |

# 将一个整数转换为不同的进制,使用\_itoa();函数实现的案例:

\_itoa(value,out,x);

value:要进行进制转换的值,例如int型的

out:输出的值,char out[];

x:进制,例如:2进制,10进制,16进制等.

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS //关闭安全检查\_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  int main(void)  {  char out[100];  int i;  scanf("%d",&i);  \_itoa(i,out,2);  printf("输出int型的二进制--> %s",out);  \_itoa(i, out, 16);  printf("\n\n输出int型的十六进制--> %s",out);  printf("\n\n");  system("pause");  } |

# 如何在C语言代码中插入汇编代码:

|  |
| --- |
| #include<stdio.h>  void main(){  int num = 10;  num = num + 5;  printf("这是没执行汇编语言之前的输出值:%d",num);  \_asm{  mov eax, num //eax是一个存储器,将num的移存到eax中  add eax,5 //eax的值加5  mov num,eax //将eax赋给num  }  printf("\n\n这是在执行汇编语言之后的输出值:%d", num);  getchar();  } |

# 如何输入一个char数组:

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  void main(){  int num; //次数  char strurl[30]; //指令  scanf("%d%s",&num,strurl);  printf("num=%d,strurl=%s",num,strurl);  char str[50];  sprintf(str,"for /l %%i in (1,1,%d) do %s",num,strurl);  printf("%s",str);  system(str);  system("pause");  } |

# 如何将执行命令行输出的内容输出到指定的文件:

1. 先进入指定的路径:

|  |
| --- |
| C:\Users\Administrator>f:  F:\>cd F:\tempTest\ll  F:\tempTest\ll> |

1. 在改路径目录下创建一个1.txt空文件
2. 然后输出如下的命令:

|  |
| --- |
| F:\tempTest\ll>tasklist<1.txt>2.txt |

这样命令tasklist执行输出的内容就输入到2.txt中了

# 如何利用sprintf()函数截取字符数组前面指定长度的的字符:

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  void main(){  char str[50]="notepad1234567";  sprintf(str,"%.7s",str); //截取str字符数组前7个字符  printf("%s",str);  system(str);  system("pause");  } |

# printf(“%m.nf”,10.234567);的使用案例:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  void main(){  printf("%f",10.234567);  printf("\n%.10f",10.234567); //小数点后面保留10位  printf("\n%20.10f",10.234567); //20表示显得宽度  printf("\n%-20.10f",10.234567); //-表示左边对其  printf("\n%020.10f",10.234567); //0表示当实际的宽度小于定义的宽度时,用0填充  printf("\n%1.10f",10.234567); //1小于实际的宽度,就用实际的宽度来填充  system("pause");  } |

# 格式符”%e”(指数的输出方式)的用法案例:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  void main(){  printf("%e",10000000000); //这里输出是错误的.因为printf()这个函数不会数据类型进转换  printf("%e",10000000000.0); //这里的输出是正确的  printf("%e",.00000000001); //这里输出也是正确的,因为小数点前面的零我们可以将其省掉  system("pause");  } |

# gets();与puts();函数的用法以及用模仿cmd窗口:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  void main(){  while (1){  char str[50];  gets(str);  puts(str);  system(str);  }  } |

# 如何用scanf();函数接收double型数据的输入:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  void main(){  double d = 1.0;  scanf("%lf",&d);  printf("%f",d);  system("pause");  } |

# c语言总是否有bool类型变量:

解析:c语言中是没有bool类型变量的

# 块语句的案例:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  void main(){  {  char str[] = "1.";  printf("%s这是块语句!!",str);  }  double d = 1.0;  scanf("%lf",&d);  printf("%f",d);  system("pause");  } |

# 模拟键盘的按键:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<Windows.h>  void main(){  // 注意:如果自己电脑本身安装有360,要把360退掉后才可以 ,因为360会把这个操作给拦截掉  //window+E组合建  keybd\_event(0x5B, 0, 0, 0); //Windows键的按下  keybd\_event(0x45, 0, 0, 0); //E键的按下  keybd\_event(0x45, 0, KEYEVENTF\_KEYUP, 0); //E键的松开  keybd\_event(0x5B, 0, KEYEVENTF\_KEYUP, 0); //windows键的松开    } |

# c语言中关于自定义函数的声明与定义:

解析:在c语言中如果一个函数时自定义的,如果该函数在main()函数之前就不需要声明,直接定义即可,如果该函数在main()函数之后,则就需要现在main()函数之前声明,在main()之后定义

# c语言如何在一个文件中引用另一个文件中的函数:

1. 首先要新建一个头文件,例如run.h,在该run.h中只编写函数的申明,
2. 还有新建一个与头文件与之对应的源文件,例如run.c或run.cpp文件,该文件只编写与头文件与之对应的函数定义.
3. 再新建一个测试的源文件.例如test.c或test.cpp.在该文件中来调用run.h中声明的函数.源码如下
4. 这是run.h的文件

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  //这是一个runing函数的声明  void runing(); |

1. 这是run.c或run.cpp文件

|  |
| --- |
| #include"run.h"  void runing(){  printf("I am runing");  } |

1. 这是test.c或test.cpp文件

|  |
| --- |
| #include"run.h"  void main(){  runing();  getchar();  } |

# 如何配合模拟键盘,鼠标等打开一百度并搜索的案例:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<Windows.h>  //打开百度  void open(){  ShellExecuteA(0,"open","http://www.baidu.com",0,0,3);  }  //关闭百度  void close(){  system("taskkill /f /t /im 360se.exe");  }  //在打开的百度输入框中输入xianhua,并回车  void search(){  keybd\_event('x', 0, 0, 0);  keybd\_event('x', 0, 2, 0);  //我也还不知道为什么这里要加上如上的两行代码  keybd\_event('X',0,0,0);  keybd\_event('X',0,2,0);  keybd\_event('I', 0, 0, 0);  keybd\_event('I', 0, 2, 0);  keybd\_event('A', 0, 0, 0);  keybd\_event('A', 0, 2, 0);  keybd\_event('N', 0, 0, 0);  keybd\_event('N', 0, 2, 0);  keybd\_event('H', 0, 0, 0);  keybd\_event('H', 0, 2, 0);  keybd\_event('U', 0, 0, 0);  keybd\_event('U', 0, 2, 0);  keybd\_event('A', 0, 0, 0);  keybd\_event('A', 0, 2, 0);  Sleep(1000);  keybd\_event(0x0D, 0, 0, 0);  keybd\_event(0x0D, 0, 2, 0);  }  //模拟鼠标点击搜索到的内容  void click(){  SetCursorPos(200,270);    mouse\_event(MOUSEEVENTF\_LEFTDOWN,0,0,0,0);  mouse\_event(MOUSEEVENTF\_LEFTUP, 0, 0, 0, 0);  //mouse\_event(MOUSEEVENTF\_LEFTDOWN, 0, 0, 0, 0);  Sleep(100);  SetCursorPos(200, 310);  mouse\_event(MOUSEEVENTF\_LEFTDOWN, 0, 0, 0, 0);  mouse\_event(MOUSEEVENTF\_LEFTUP, 0, 0, 0, 0);  }  void main(){  for (int i = 0; i < 5;i++){  Sleep(1000);  open();  Sleep(3000);  search();  Sleep(3000);  click();  Sleep(7000);  close();  Sleep(3000);  }    } |

# 在C语言如何在堆区分配一块可以指定大小的内存和栈区自动分配内存的案例:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<Windows.h>  void main(){  int i = 1; //这系统自动分配栈区的内存  //C语言中如何在堆区分配一块内存  int \*p = (int \*)malloc(1024\*1024\*10);  getchar();  } |

# 空类型指针:

解析:空类型的指针可以用于接收任何类型变量的地址:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<Windows.h>  void main(){  int a = 10;  void \*p = &a; //空类型的指针可以接受如何类型的地址,但空类型指针不能取出内容,因为它知道是什么类型,也不知道大小  printf("%x,%x",&a,p);  //printf("\n%d",\*p); // error C2100: 非法的间接寻址  getchar();  } |

# 函数指针的使用案例:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<Windows.h>  int add(int a,int b){  return a + b;  }  void main(){  int addr = add(1,2);  printf("这是add函数的返回值:%d",addr);  printf("\n\n这是add函数返回值的地址:%x",&addr);  int (\*functionPointer)(int,int); //声明一个函数指针类型  functionPointer = (int (\*)(int, int))add; //这里add是获得add()函数的地址  addr = functionPointer(2,4);  printf("\n\n-----以下是用户函数指针的方式来调用的------");  printf("\n\n这是add()函数的地址%x",add);  printf("\n\n这是add函数的返回值:%d", addr);  printf("\n\n这是add函数返回值的地址:%x", &addr);  getchar();  } |

# 指针的指针的用法:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<Windows.h>  void main(){  int \*p = 0;  int \*\*pp = 0;  int i = 10;  p = &i;  pp = &p; //将指针p的地址赋予给pp指针,pp指针的值就p指针的地址,所以\*\*pp的值就等于\*p的值,为10  printf("通过\*\*pp等于\*p的值%d ",\*\*pp);  getchar();  } |

# C语言如何使用malloc()分配和free()来释放的内存空间:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<Windows.h>  void main(){  //使用malloc()函数来合理的分配内存空间,并用指针来替代数组来利用malloc所分配的内存  //malloc()函数的返回的是一片内存的首地址  //malloc()函数的返回值类型是void \* ,即空指针类型  int i = 20;  int \*p = (int \*)malloc(i\*sizeof(int));  printf("返回的首地址:%x",p);  //使用for循环来来初始化这块已经分配的内存  for (int j = 0; j < 20;j++){  p[j] = j; //p[j]类似于\*p,但p[j]是动态的,而\*p静态的  printf("\n这是p[j]的值:%d , 这是\*p的值:%d ,只是p的地址: %x , 这是p[j]的地址:%x",p[j],\*p,p,&p[j]);  printf("\n\*(p+j)的值%d,(p+j)地址的值%x\n",\*(p+j),p+j);  }  free(p); //将使用完毕的堆内存释放掉  system("pause");  } |

# 文件的打开与关闭案例:

1. 文件的打开
2. c语言问的打开是通过stdio.h函数库的fopen()函数实现的.fopen()函数的原理是:

FILE \*fopen(char \*filename,char \*mode);

其中filename是要打开的文件名,mode是说明处理文件的方式.它的调用方式一般为:

文件指针变量=fopen(文件名,处理文件的方式);

例如:fopen(“a:\\exam.txt”,”r”);

1. 注意:”文件名”是要打开的文件的文件名字,可以是包含盘符,路径,文件名,扩展名的文件标识符.但在属性要符合c语言的规定,例如文件名”a:\tc\1.txt”,在改函数中的文件名应写成”a:\\tc\\1.txt”.

二:打开的模式(mode)

mode:

r:读取,w:写入,a:追加,r+:读取/写入,w+:写入/读取,a+:读取/追加

三.文件的关闭

当c语言文件使用完后,应该立即关闭文件,放在由于无操作等原因破坏已经打开的文件.文件的关闭通过stdio.h中的fclose()函数实现.具体用法是:

fclose(文件的指针); 例如 fclose(fp);

则程序将文件类型指针fp所指向的文件关闭.之后fp将不再指向该文件.

# 如何打开和关闭一个二进制的文件:

1. 如果是二进制文件,在使用时只要在模式后添加字符b即可,如”rb”,”rb+”分别表示读取二进制文件和以读取/写入打开二进制文件.如果由于文件不存在等原因造成不能打开文件,则调用fopen()后将返回一个空指针NULL

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  void main(){  FILE \*fp = fopen("C:\\Users\\Administrator\\Desktop\\11.txt","r");  // printf("%p",fp);  if (fp == NULL){  printf("connot open thi file.\n");  exit(1);  }  system("pause");  } |

一般exit(0)表示程序正常退出,exit(非零值)表示程序出错后退出.至于关闭二进制文件与关闭一个正常的文件时一样的.没有什么其他的区别.

# 如何用printf()输出一个指针案例:

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  void main(){  FILE \*fp = fopen("C:\\Users\\Administrator\\Desktop\\1.txt","w"); //这里修改为w  printf("%p",fp); //这里输出的是一个指针  if (fp == NULL){  printf("connot open thi file.\n");  exit(1);  }  char w = fputc('a',fp);  printf("%c,%d",w,EOF);  fclose(fp);  system("pause");  } |

# 文件的读写操作:

1. 文件打开后,最常见的操作就是读取和写入.C语言提供多种对文件的读取和写入的函数,这里主要介绍四种常用文件的读写操作函数,他们都在文件头stdio.h中定义的.
2. 按字符读写的函数fgetc(),fputc();
3. 按字符串读写的函数fgets(),futs();
4. 按格式要求读写的函数fprintf();,fscanf();
5. 按数据块读写的函数fread(),fwrite();

# 按字符读写的函数fgetc(),fputc():

1. fputc()函数
2. 函数fputc()的作用是向文件写入一个字符,其条用的形式为:

fputc(字符,文件型指针变量);

例如:fputc(‘A’,fp);

fp为一文件类型指针变量,上式将字符常量’A’(也可以是字符型变量)写入文件当前位置,并且使文件位置指针下移一个字节.如果写入操作成功,则返回值是该字符,否则返回EOF.

EOF:EOF为c语言的一个常量,其值为-1

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  void main(){  FILE \*fp = fopen("C:\\Users\\Administrator\\Desktop\\1.txt","r");  // printf("%p",fp);  if (fp == NULL){  printf("connot open thi file.\n");  exit(1);  }  int w = fputc('a',fp);  printf("%d,%d",w,EOF);  fclose(fp);  system("pause");  } |

文件的写入1.

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  void main(){  FILE \*fp = fopen("C:\\Users\\Administrator\\Desktop\\1.txt","w"); //这里修改为w  // printf("%p",fp);  if (fp == NULL){  printf("connot open thi file.\n");  exit(1);  }  char w = fputc('a',fp);  printf("%c,%d",w,EOF);  fclose(fp);  system("pause");  } |

文件的写入2.

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  void main(){    printf("\n\t请输入要新建的文件名称:");  char name[50];  char filePath[100]="C:\\Users\\Administrator\\Desktop\\";  char suffix[20]=".txt";  char context;  scanf("%s",name);  sprintf(filePath,"%s%s%s",filePath,name,suffix);  FILE \*fp = fopen(filePath, "w+");  printf("%p", fp); //这里输出的是一个指针  if (fp == NULL){  printf("connot open thi file.\n");  exit(1);  }  printf("请输入内容,当输入一个\*字符时结束");  context = getchar();  while (context !='\*'){  fputc(context,fp);  context = getchar();  }  fclose(fp);  system("pause");  } |

fgetc()函数,结构体,文件的读出等用法的综合案例:

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  struct Path{  char filePath[1000];  };  struct Path getFilePath();  /\*返回1表示拷贝成功!0表示失败\*/  int copyFile(char filePath[]);  void main(){  struct Path path = getFilePath();  printf("%s",path.filePath);  int flag = copyFile(path.filePath);  if (flag){  printf("拷贝成功!");  }else{  printf("拷贝失败!");  }  system("pause");  }  struct Path getFilePath(){  printf("\n\t请输入要新建的文件名称:");  char name[50];  char filePath[100] = "C:\\Users\\Administrator\\Desktop\\";  char suffix[20] = ".txt";  char context;  scanf("%s", name);  sprintf(filePath, "%s%s%s", filePath, name, suffix);  FILE \*fp = fopen(filePath, "w+");  printf("%p", fp); //这里输出的是一个指针  if (fp == NULL){  printf("connot open thi file.\n");  exit(1);  }  printf("请输入内容,当输入一个\*字符时结束");  context = getchar();  while (context != '\*'){  fputc(context, fp);  context = getchar();  }  fclose(fp);  //system("pause");  struct Path path;  sprintf(path.filePath,"%s",filePath);  return path;  }  int copyFile(char filePath[]){  FILE \*fp = fopen(filePath,"r+");  FILE \*copyFile = fopen("C:\\Users\\Administrator\\Desktop\\copyFile.txt","w+");  if (fp == NULL){  printf("connot open the file.\n");  return 0;  }  char context = fgetc(fp);  while(context != -1){  fputc(context,copyFile);  context = fgetc(fp);  }  fclose(fp);  fclose(copyFile);  return 1;  } |

# 按字符串读写的函数fgets(),fputs()

1. fputs()函数
2. 函数fputs()的作用是向文件写入字符串,其调用形式为:

fputs(字符串,文件型指针变量);

其中字符串可以是字符串常量,指向字符串的指针变量,存放字符串数组的数组名.写入文件成功,函数返回值为0,否则为EOF

1. 注意:字符串的结束标志’\0’不写入.例如

fputs(“hello”,fp)

fp为一个文件类型指针变量,上式将字符串中的字符h,e,l,l,o写入文件指针的当前位置.

fgets()与fputs()的案例:

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  struct Path{  char filePath[1000];  };  struct Path getFilePath();  /\*返回1表示拷贝成功!0表示失败\*/  int copyFile(char filePath[]);  void main(){  struct Path path = getFilePath();  printf("%s",path.filePath);  int flag = copyFile(path.filePath);  if (flag){  printf("拷贝成功!");  }else{  printf("拷贝失败!");  }  system("pause");  }  struct Path getFilePath(){  printf("\n\t请输入要新建的文件名称:");  char name[50];  char filePath[100] = "C:\\Users\\Administrator\\Desktop\\";  char suffix[20] = ".txt";  char context[1024];  scanf("%s", name);  sprintf(filePath, "%s%s%s", filePath, name, suffix);  FILE \*fp = fopen(filePath, "w+");  printf("%p", fp); //这里输出的是一个指针  if (fp == NULL){  printf("connot open thi file.\n");  exit(1);  }  printf("请输入内容,当输入一个\*字符时结束\n");  scanf("%s", context);    while (context[0] != '\*'){  fputs(context, fp);  scanf("%s", context);    }  fclose(fp);  struct Path path;  sprintf(path.filePath,"%s",filePath);  return path;  }  int copyFile(char filePath[]){  FILE \*fp = fopen(filePath,"r+");  FILE \*copyFile = fopen("C:\\Users\\Administrator\\Desktop\\copyFile.txt","w+");  if (fp == NULL){  printf("connot open the file.\n");  return 0;  }  char context[1024];  fgets(context, 1024, fp);  //while (context[0] != -1){  fputs(context,copyFile);  fgets(context,1024,fp);  //}  /\*  char context = fgetc(fp);  while(context != -1){  fputc(context,copyFile);  context = fgetc(fp);  } \*/  fclose(fp);  fclose(copyFile);  return 1;  } |

1. fgets()函数:

函数fgets()的作用是从一个文件中读取一个字符串.其调用形式为:

fgets(str,n,fp);

其作用是从fp指向的文件的当前位置开始读取n-1个字符,并加上字符串结束’\0’一起放入字符数组str中.如果从文件读取字符时遇到换行符或文件结束标示EOF,读取结束.函数返回值为字符数组Str的首地址.

1. ….

# 使用fgets()与fputs()函数实现文件内容输入与拷贝:

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  struct Path{  char filePath[1000];  };  struct Path getFilePath();  /\*返回1表示拷贝成功!0表示失败\*/  int copyFile(char filePath[]);  void main(){  struct Path path = getFilePath();  printf("%s",path.filePath);  int flag = copyFile(path.filePath);  if (flag){  printf("拷贝成功!");  }else{  printf("拷贝失败!");  }  system("pause");  }  struct Path getFilePath(){  printf("\n\t请输入要新建的文件名称:");  char name[50];  char filePath[100] = "C:\\Users\\Administrator\\Desktop\\";  char suffix[20] = ".txt";  char context[1024];  scanf("%s", name);  sprintf(filePath, "%s%s%s", filePath, name, suffix);  FILE \*fp = fopen(filePath, "w+");  printf("%p", fp); //这里输出的是一个指针  if (fp == NULL){  printf("connot open thi file.\n");  exit(1);  }  printf("请输入内容,当输入一个\*字符时结束\n");  scanf("%s", context);    while (context[0] != '\*'){  fputs(context, fp);  scanf("%s", context);    }  fclose(fp);  struct Path path;  sprintf(path.filePath,"%s",filePath);  return path;  }  int copyFile(char filePath[]){  FILE \*fp = fopen(filePath,"r+");  FILE \*copyFile = fopen("C:\\Users\\Administrator\\Desktop\\copyFile.txt","w+");  if (fp == NULL){  printf("connot open the file.\n");  return 0;  }  char context[1024];  while (fgets(context,1024,fp) != NULL){  fputs(context, copyFile);  }  //fgets(context, 1024, fp);  //fputs(context, copyFile);  fclose(fp);  fclose(copyFile);  return 1;  } |

# 按格式要求读写函数fprintf(),fscanf():

1. fprintf()函数

函数fprintf()的作用与printf()相似,只是输出对象不是标准输出设备而是文件,即按照格式要求将数据写入文件.它调用的一般形式为:

fprintf(文件型指针变量,格式控制符,输出列表)

例如:

fprintf(fp,”%ld,%s,%5.1f”,num,name,score);

它的作用是将变量num,name,score按照%ld,%s,%5.1f的格式写入fp指向的文件的当前位置.

1. fprintf()函数的使用案例:\

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  struct Path{  char filePath[1000];  };  //定义一个学生的结构体  struct Student{  int id; //学生的编号  char name[200]; //学生的姓名  char gender[10]; //学生的性别  char end; //结束符 \*  };  struct Path getFilePath();  /\*返回1表示拷贝成功!0表示失败\*/  int copyFile(char filePath[]);  void main(){  struct Path path = getFilePath();  printf("%s",path.filePath);  int flag = copyFile(path.filePath);  if (flag){  printf("拷贝成功!");  }else{  printf("拷贝失败!");  }  system("pause");  }  struct Path getFilePath(){  printf("\n\t请输入要新建的文件名称:");  char name[50];  char filePath[100] = "C:\\Users\\Administrator\\Desktop\\";  char suffix[20] = ".txt";  //char context[1024];  struct Student student;  scanf("%s", name);  sprintf(filePath, "%s%s%s", filePath, name, suffix);  FILE \*fp = fopen(filePath, "w+");  printf("%p", fp); //这里输出的是一个指针  if (fp == NULL){  printf("connot open thi file.\n");  exit(1);  }  //printf("请输入内容,当在student.gender[0]输入一个\*字符时结束\n");  //scanf("%d,%s,%s,%c", &student.id,&student.name,&student.gender,&student.end);    //while (student.gender[0] != '\*'){  student.id = 1;  sprintf(student.name,"%s","傅雷家书附件是分开了");  sprintf(student.gender,"%s","男");  fprintf(fp,"%d,%s,%s",student.id,student.name,student.gender);  //scanf("%d,%s,%s", &student.id, student.name, student.gender);    //}  fclose(fp);  struct Path path;  sprintf(path.filePath,"%s",filePath);  return path;  }  int copyFile(char filePath[]){  FILE \*fp = fopen(filePath,"r+");  FILE \*copyFile = fopen("C:\\Users\\Administrator\\Desktop\\copyFile.txt","w+");  if (fp == NULL){  printf("connot open the file.\n");  return 0;  }  char context[1024];  while (fgets(context,1024,fp) != NULL){  fputs(context, copyFile);  }  //fgets(context, 1024, fp);  //fputs(context, copyFile);  fclose(fp);  fclose(copyFile);  return 1;  } |

1. fscanf()函数

函数scanf()从通过标准输入设备读取数据,同样函数fscanf()按照格式要求从文件中读取数据.它条用的一般形式为:

fscanf(文件型指针变量,格式控制,输入表列);

例如:

fscanf(fp,”%ld,%s,%5.1f”,&num,name,&score);

它的作用是从fp指向的文件的当前位置开始,按照%ld,%s,%5.1f的格式取出数据,赋给变量num,name和score.

主要用于数据文件的读写,即可以使用ASCII文件也可以使用二进制文件.

案例:

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  struct Path{  char filePath[1000];  };  //定义一个学生的结构体  struct Student{  int id; //学生的编号  char name[200]; //学生的姓名  char gender[10]; //学生的性别  char end; //结束符 \*  };  struct Path getFilePath();  /\*返回1表示拷贝成功!0表示失败\*/  int copyFile(char filePath[]);  void main(){  struct Path path = getFilePath();  printf("%s",path.filePath);  int flag = copyFile(path.filePath);  if (flag){  printf("拷贝成功!");  }else{  printf("拷贝失败!");  }  system("pause");  }  struct Path getFilePath(){  printf("\n\t请输入要新建的文件名称:");  char name[50];  char filePath[100] = "C:\\Users\\Administrator\\Desktop\\";  char suffix[20] = ".txt";  //char context[1024];  struct Student student;  scanf("%s", name);  sprintf(filePath, "%s%s%s", filePath, name, suffix);  FILE \*fp = fopen(filePath, "w+");  printf("%p", fp); //这里输出的是一个指针  if (fp == NULL){  printf("connot open thi file.\n");  exit(1);  }  //printf("请输入内容,当在student.gender[0]输入一个\*字符时结束\n");  //scanf("%d,%s,%s,%c", &student.id,&student.name,&student.gender,&student.end);    //while (student.gender[0] != '\*'){  student.id = 1;  sprintf(student.name,"%s","傅雷家书附件是分开了");  sprintf(student.gender,"%s","男");  fprintf(fp,"%d,%s,%s",student.id,student.name,student.gender);  //scanf("%d,%s,%s", &student.id, student.name, student.gender);    //}  fclose(fp);  struct Path path;  sprintf(path.filePath,"%s",filePath);  return path;  }  int copyFile(char filePath[]){  FILE \*fp = fopen(filePath,"r+");  FILE \*copyFile = fopen("C:\\Users\\Administrator\\Desktop\\copyFile.txt","w+");  if (fp == NULL){  printf("connot open the file.\n");  return 0;  }  struct Student student;  char context[1024];  fscanf(fp, "%d,%s,%s", &student.id, student.name, student.gender);  fprintf(copyFile, "%d,%s,%s", student.id, student.name, student.gender);  ///while (fgets(context,1024,fp) != NULL){  // fputs(context, copyFile);  //}  //fgets(context, 1024, fp);  //fputs(context, copyFile);  fclose(fp);  fclose(copyFile);  return 1;  } |

1. .

# 按数据块读写的函数fread(),fwrite()

1. fwrite()函数

函数fwrite()的作用是成批的数据块写入文件.

fwrite(写入文件的数据块的存放地址,一个数据块的字节数,数据块的个数,文件类型指针变量);

如果函数fwrite()操作成功,则返回值为实际写入文件的数据块的个数.

例如:已知struct student类型的数组str[20],则语句

fwrite(&str[1],sizeof(struct student),2,fp);

从结构体数组元素stu[1]存放的地址开始,以一个结构体struct student类型变量所占字节数为一个数据块,工写入文件类型指针fp指向的文件2个数据块,即str[1],stu[2]的内容写入文件.如果操作成功,函数的返回值为2.

1. fread()函数

函数fread()的作用是从文件中读出成批的数据块.fread(从文件读取的数据块的存放地址,一个数据块的字节数,数据块的个数,文件类型指针变量);

如果函数fread()操作成功,则返回是为实际从文件中读取数据块的个数.

例如:已知stu1是一个结构体truct student 变量,则

fread(&tru1,sizeof(struct student),1,fp);

从文件类型指针fp指向的文件的当前位置开始,读取一个数据块,该数据块为结构体struct student类型变量所占字节数,然后将读取的内容放入变量stu1中.

主要:fwrite()和fread()函数读写文件时,只有使用二进制方式,才可以读写任何类型的数据,最常用与读写数组和结构体类型的数据.

1. fwrite()与fread()的案例:

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  struct Path{  char filePath[1000];  };  //定义一个学生的结构体  struct Student{  int id; //学生的编号  char name[200]; //学生的姓名  char gender[4]; //学生的性别  };  struct Path getFilePath();  /\*返回1表示拷贝成功!0表示失败\*/  int copyFile(char filePath[]);  void main(){  struct Path path = getFilePath();  printf("%s",path.filePath);  int flag = copyFile(path.filePath);  if (flag){  printf("拷贝成功!");  }else{  printf("拷贝失败!");  }  system("pause");  }  struct Path getFilePath(){  printf("\n\t请输入要新建的文件名称:");  char name[50];  char filePath[100] = "C:\\Users\\Administrator\\Desktop\\";  char suffix[20] = ".txt";  //char context[1024];  struct Student student[3] = {  1,"张三", "男\n",  2, "李四", "男\n",  3, "王五", "女\n"  };  scanf("%s", name);  sprintf(filePath, "%s%s%s", filePath, name, suffix);  FILE \*fp = fopen(filePath, "wb+");  printf("%p", fp); //这里输出的是一个指针  if (fp == NULL){  printf("connot open thi file.\n");  exit(1);  }  fwrite(student,sizeof(struct Student),3,fp);  fclose(fp);  struct Path path;  sprintf(path.filePath,"%s",filePath);  return path;  }  int copyFile(char filePath[]){  FILE \*fp = fopen(filePath,"rb+");  FILE \*copyFile = fopen("C:\\Users\\Administrator\\Desktop\\copyFile.txt","wb+");  if (fp == NULL){  printf("connot open the file.\n");  return 0;  }  struct Student student[3];  char context[1024];  fread(student, sizeof(struct Student), 3, fp);  fwrite(student, sizeof(struct Student), 3, copyFile);  //fgets(context, 1024, fp);  //fputs(context, copyFile);  fclose(fp);  fclose(copyFile);  return 1;  } |

1. .

# 文件的其他常用函数

1. 文件在使用时,指向打开的文件的指针可以指示文件的当前位置,当每次读取或写入数据时,是从文件指针的当前位置开始读取与写入数据的,然后指针自动移到读写下一个数据的位置.所以文件指针的定位非常重要.
2. 在c语言中的stdio.h头文件中定义的一些文件操作中常用的库函数.
3. 函数feof(),rewind(),fseek(),ftell()
4. .

# 函数feof()

1. 函数feof()用来检测一个指向文件的指针是否已经指到了文件最后的结束标志EOF,调用的一般形式:

feof(文件型指针变量);

如果文件型指针指向的文件的当前位置为结束标志EOF,则函数返回一个非零值,否则返回0值.

# 函数ftell()

1. 函数ftell()用于测试指向文件的指针的当前位置,它的调用方式为:

ftell(文件型指针变量);

函数的返回值是一个常整数,如果测试成功,则返回指向文件的指针当前指向的位置距离文件开头的字节数,否则否后-1L

# 在windows系统中如何用SetCurrentDirectory(路径);设置当前路径

说明:设置当前目录

返回值:Long，非零表示成功，零表示失败。会设置GetLastError

参数表:参数 类型及说明

lpPathName String，新目录的路径

Example案例

CString g\_strDirect = "\\vcpp32\\ex34a\\WebSite";

SetCurrentDirectory(g\_strDirect);

# 在windows系统中如何用GetCurrentDirectory()获取当前路径

函数功能:获取当前进程的当前目录。

注意当前目录这东西有点怪异，本来确实是可执行文件的所在目录，但是当用文件对话框成功打开一个文件后，该目录就被修改为被打开的文件所在目录。比如在程序中使用"123.txt"这样的文件名，希望将它存储在exe文件所在目录中，但是当用户用文件对话框打开一个文件"F:\456.txt"后，这个123.txt会被存储到F盘下。(VC2005下的情况，其他VC版本待验证)

参数说明:参数 类型及说明

nBufferLength 缓冲区的长度

lpBuffer 指定一个预定义字串，用于装载当前目录

返回值

调用成功 返回装载到lpBuffer的字节数。

使用GetLastError函数可获得错误信息

# 在linux中关于设置与获取当前路径的案例

|  |
| --- |
| 相关函数：getcwd, chroot  头文件：#include <unistd.h>  定义函数：int chdir(const char \* path);  函数说明：chdir()用来将当前的工作目录改变成以参数path 所指的目录.  返回值执：行成功则返回0, 失败返回-1, errno 为错误代码.  范例  #include <unistd.h>  main(){  chdir("/tmp");  printf("current working directory: %s\n", getcwd(NULL, NULL));  }  执行：  current working directory :/tmp |

# c语言的函数怎样返回数组案例(用指针的方式)

1. Test.h的头文件

|  |
| --- |
| #include<stdio.h>  #include<Windows.h>  #include<stdlib.h>  /\*用户封装目录路径\*/  struct Path{  char directoryPath[400];  };  /\*路径后缀\*/  struct PathSuffix{  char directorySuffix[1500];  };  /\*将所有的后缀都装置到这个结构体中\*/  struct AllSuffix{  struct PathSuffix pathSuffix[];  };  /\*这是目录前缀\*/  struct Path directoryPrefix;  /\*通过传人的目录路径得到完整目录路径\*/  struct Path getDirectoryPath(struct Path path);  /\*获取输入的父亲目录路径\*/  struct Path setDirectoryPath();  /\*为所有路径后缀的值设置值\*/  struct PathSuffix \*setAllPathSuffix();    /\*逻辑函数\*/  void logicFuncation(); |

1. Test.cpp文件

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<Windows.h>  #include<stdlib.h>  #include<string.h>  #include "Test.h"  int main(){  logicFuncation();  system("pause");  return 0;  }  /\*逻辑函数\*/  void logicFuncation(){  char path[1000];  directoryPrefix = setDirectoryPath();  struct PathSuffix \*pathSuffix = NULL;  pathSuffix = setAllPathSuffix();  for (int i = 0; i < 50; i++){  // printf("\t\n %d : %s", i, pathSuffix[i].directorySuffix);  sprintf(path, "%s%s", directoryPrefix.directoryPath, pathSuffix[i].directorySuffix);  printf("\n %d\t%s",i+1,path);  SetCurrentDirectory(path);  system("install.bat");  }  system("pause");  }  /\*获取输入的父亲目录路径\*/  struct Path setDirectoryPath(){  char directory[2000];  struct Path path;  scanf("%s", directory);  sprintf(path.directoryPath,"%s",directory);  return path;  }  /\*为所有路径后缀的值设置值\*/  struct PathSuffix \*setAllPathSuffix(){  //struct AllSuffix allSuffix;  struct PathSuffix pathSuffix[50];  sprintf(pathSuffix[0].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[1].directorySuffix, "%s", "\\com.bjhy.platform.process");  sprintf(pathSuffix[2].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[3].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[4].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[5].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[6].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[7].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[8].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[9].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[10].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[11].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[12].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[13].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[14].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[15].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[16].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[17].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[18].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[19].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[20].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[21].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[22].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[23].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[24].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[25].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[26].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[27].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[28].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[29].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[30].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[31].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[32].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[33].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[34].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[35].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[36].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[37].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[38].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[39].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[40].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[41].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[42].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[43].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[44].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[45].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[46].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[47].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[48].directorySuffix, "%s", "\\com.bjhy.platform.dao");  sprintf(pathSuffix[49].directorySuffix, "%s", "\\com.bjhy.platform.dao");  //allSuffix.pathSuffix = pathSuffix;  //sprintf(allSuffix.pathSuffix, '%a', pathSuffix);  return pathSuffix;  } |