知识点1【作业讲解】

知识点2【字符数组】（重要）

字符数组的初始化

字符数组的遍历

逐个字符初始化 和 字符串初始化的区别

获取键盘的字符串：

scanf和%s使用的时候 有个缺点 遇到 空格会结束输入

gets 缺点： 获取键盘输入的时候 不会管 buf的大小 容易造成 内存 污染

fgets函数 既可以获取 带空格的字符串 也可以保证 buf的不越界

提高案例：字符串的大小写 转换

作业：键盘获取一个字符串 大小自定义 将字符串的大写字母变小写 小写字母变大写 其他符号保持不变

知识点3【二维字符数组】(了解)

二维字符数组获取键盘输入的字符串：

作业：

知识点4【函数概述】

1、函数的定义

2、函数声明：

3、函数的调用

可以省略函数声明：函数的调用 在 函数定义的下方 就可以省略函数声明

知识点5【函数参数】（重要）

1、如果函数的形参 啥都不写 在调用的时候 可以传实参 只是实参得不到使用

2、如果函数没有参数 请将形参写成 void

知识点1【作业讲解】

```
void test01()
{
 int arr[3][4]={0};

 //获取键盘输入
 int i=0,j=0;
 for(i=0;i<3;i++)
 {
 for(j=0;j<4;j++)
  {
  scanf("%d", &arr[i][j]);
  }
  }

  for(i=0;i<3;i++)
  {
  //求每一行的平均值
  int sum = 0;

  for(j=0;j<4;j++)
  {
  sum += arr[i][j];
  }

  printf("第%d行的平均值%d\n",i, sum/4);
  }
}
int main(int argc,char *argv[])
{
```

```
 test01();
 return 0;
}
```

运行结果：

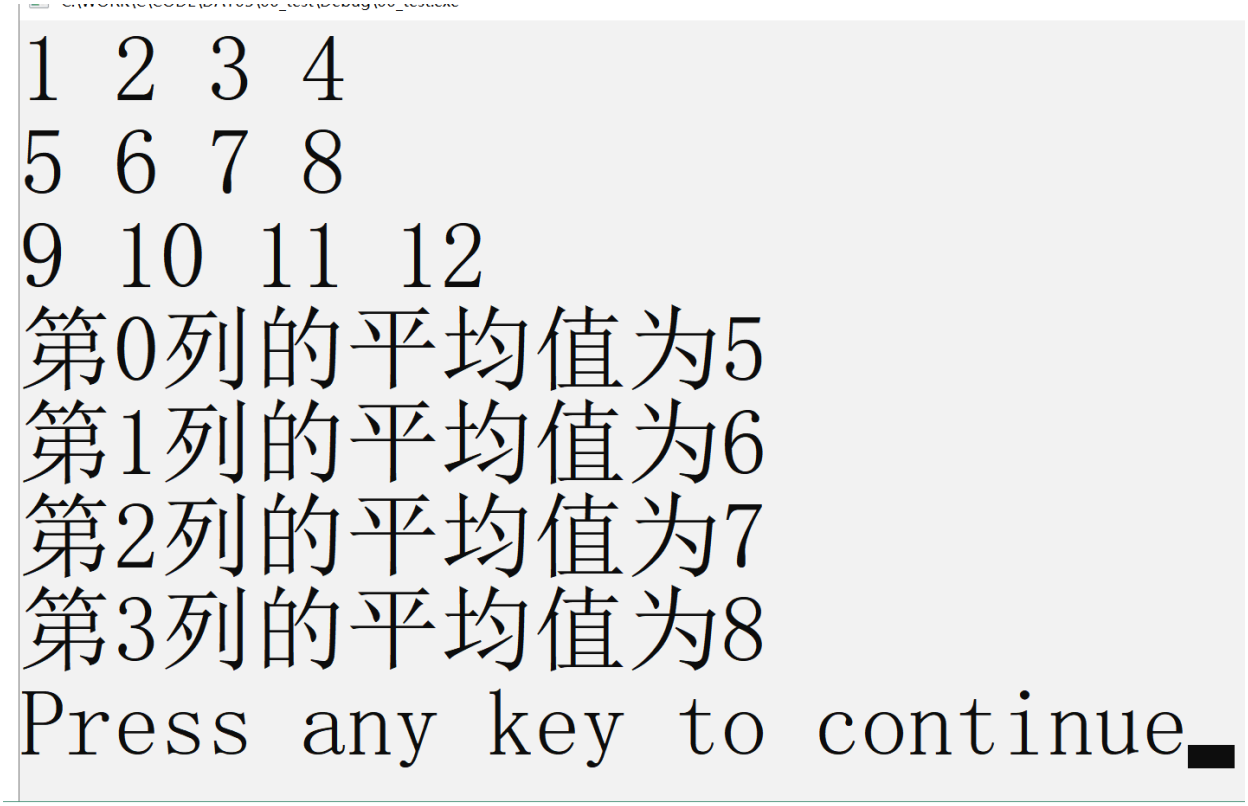
```
1 2 3 4
5 6 7 8
9 10 11 12
第0行的平均值2
第1行的平均值6
第2行的平均值10
Press any key to continue
```

案例：

```
#include<stdio.h>
void test01()
{
 int arr[3][4]={0};

 //获取键盘输入
 int i=0,j=0;
 for(i=0;i<3;i++)
 {
  for(j=0;j<4;j++)
  {
  scanf("%d", &arr[i][j]);
  }
  }

  for(j=0;j<4;j++)
  {
```

```
18  //计算第j列 0j 1j 2j
19  int sum = 0;
20  for(i=0;i<3;i++)
21  {
22  sum += arr[i][j];
23  }
24  printf("第%d列的平均值为%d\n",j, sum/3);
25  }
26
27
28 }
29 int main(int argc,char *argv[])
30 {
31  test01();
32  return 0;
33 }
```

运行结果：



知识点2【字符数组】（重要）

字符数组:本质是数组 只是数组的每个元素 是字符。

字符数组的初始化

```
1  //初始化：逐个字符初始化 不推荐
```

```
char str[6]={'h','e','l','l','o'};
//str[0] == 'h'的ASCII值
//初始化：以字符串的形式 初始化 推荐
char str[6]="hello";
```

字符数组的遍历

```
void test02()
{
 char str[6]="hello";

 //逐个字符遍历
 int i=0;
 for(i=0;i<6;i++)
 {
 printf("%c",str[i]);
  }
  printf("\n---------------\n");

  //字符数组 可以整体遍历 推荐
  printf("%s\n", str);
}
```

运行结果：

重要：一维数组名 代表的是 数组的第0个元素的地址。必须记住

```
void test03()
{
    char str[6]="hello";
    //%p是十六进制输出地址
    printf("第0个元素的地址:%p\n", &str[0] );
    printf("数组名:%p\n",str);
}
```

```
"C:\WORK\C\CODE\DAY05\00_test\Debug\00_test.exe"
第0个元素的地址:0019FED4
数组名:0019FED4
Press any key to continue
```

逐个字符初始化 和 字符串初始化的区别

```
1  void test04()
2  {
3   char str1[]={'h','e','h','e'};//逐个字符初始化 系统不会添加'\0'
4   char str2[]="hehe";//以字符串初始化 系统会给字符串添加'\0'
5
6   //空间大小
7   printf("sizeof(str1)=%d\n",sizeof(str1));//4
8   printf("sizeof(str2)=%d\n",sizeof(str2));//5
9
10   //%s输出的内容 从数组的第0个元素开始逐个元素输出 直到遇到'\0'结束
11   printf("str1 = ##%s##\n",str1);
12   printf("str2 = ##%s##\n",str2);
13  }
14  int main(int argc,char *argv[])
15  {
16   test04();
17   return 0;
18  }
```

运行结果:

```
sizeof(str1)=4
sizeof(str2)=5
str1 = ##hehe0   ##
str2 = ##hehe##
Press any key to continue
```

获取键盘的字符串:

定义一个字符数组 有足够大的空间

```
void test05()
{
 char buf[128]="";
 printf("请输入一个字符串\n");
 scanf("%s",buf);

 printf("buf=%s\n",buf);

}
```

运行结果:

```
请输入一个字符串
hello
buf=hello
Press any key to continue
```

scanf和%s使用的时候 有个缺点 遇到 空格会结束输入

```
void test05()
{
 char buf[128]="";
 printf("请输入一个字符串\n");
 //scanf("%s",buf);//不能获取带空格的字符串
 gets(buf);//获取带空格的字符串
 printf("buf=%s\n",buf);
}
```

运行结果:

```
请输入一个字符串
hello world
buf=hello world
Press any key to continue
```

gets 缺点: 获取键盘输入的时候 不会管 buf的大小 容易造成 内存 污染

案例:

```
void test05()
{
 char buf[10]="";
 printf("请输入一个字符串\n");
```

```
5
6   gets(buf);
7
8   printf("buf=%s\n",buf);
9
10  }
```

运行结果：

```
请输入一个字符串
hhhhhhhhhhhhhhhhhhhhhh
buf=hhhhhhhhhhhhhhhhhhhhhh
Press any key to continue
```

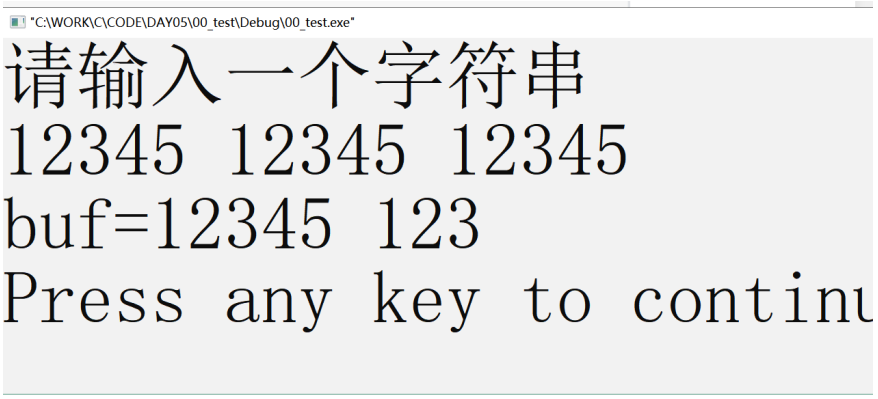
fgets函数 既可以获取 带空格的字符串 也可以保证 buf的不越界

```
1   #include<stdio.h>
2   char *fgets(char *s, int size, FILE *stream)
3   //s表示存放字符串的空间地址
4   //size 能够提取字符串的最大长度 size-1
5   //stream stdin 表示标准输入设备
6   返回值：就是获取到的字符串的首元素地址
```

案例：

```
1  void test05()
2  {
3   char buf[10]="";
4   printf("请输入一个字符串\n");
5
6   fgets(buf, sizeof(buf), stdin);//推荐
7
8   printf("buf=%s\n",buf);
9
10  }
11 int main(int argc,char *argv[])
12 {
13  test05();
14  return 0;
15 }
```

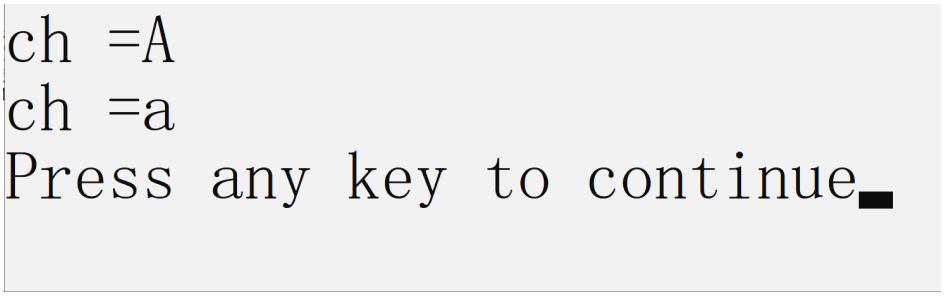
运行结果：



提高案例：字符串的大小写 转换

```c
void test06()
{
 //'a' 97 'b' 98 'c' 99 ~ 'z' 122
 //'A' 65 'B' 66 'C' 67 ~ 'Z' 90
 //说明小字母 和 大写字母的差值 是32

 char ch = 'a';
 //小写 变 大写 -32
 ch = ch -32;//ch = ch - ('a'-'A');
  printf("ch =%c\n",ch);

  //大写 变 小写 +32
  ch = ch+32;//ch = ch + ('a'-'A');
  printf("ch =%c\n",ch);
}
```

运行结果：



作业：键盘获取一个字符串 大小自定义 将字符串的大写字母变小写 小写字母变大写 其他符号保持不变

1、键盘获取一个字符串

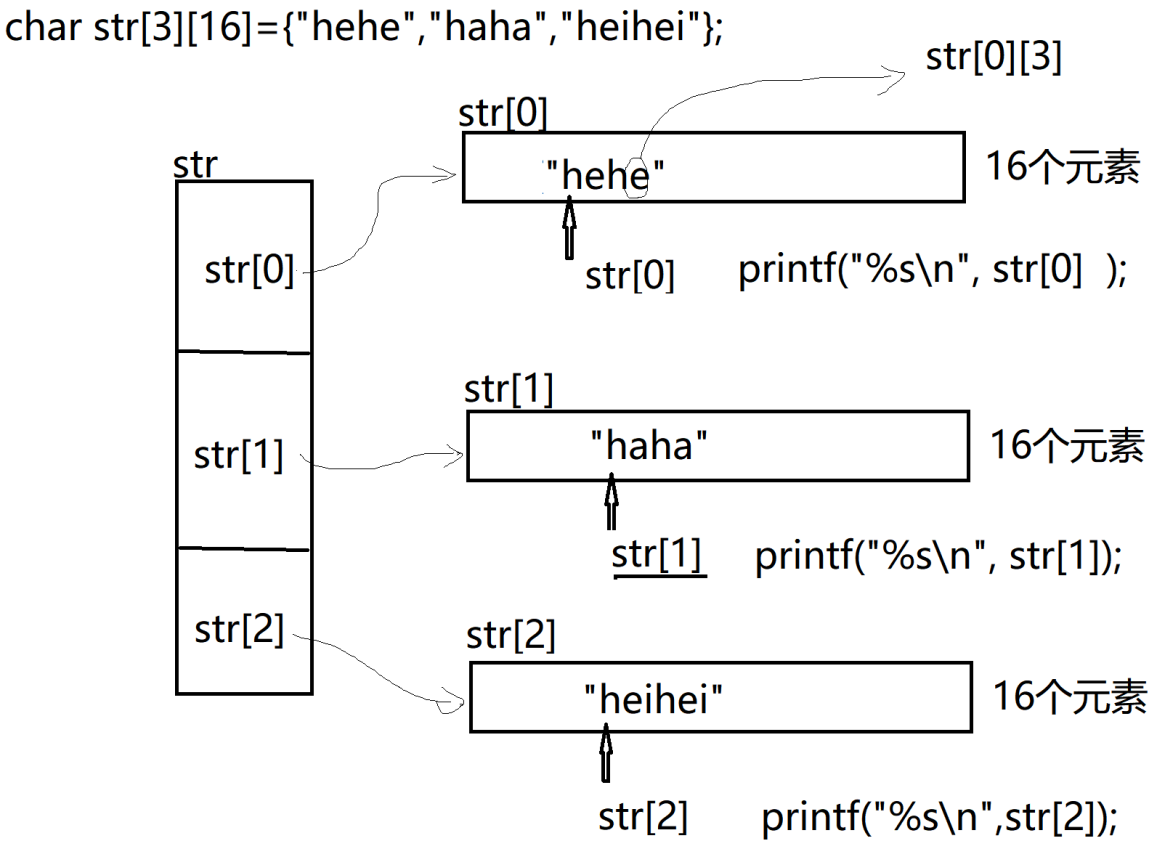
2、用for循环 将字符串的逐个字符 进行大小写转换

知识点3【二维字符数组】(了解)

一维字符数组 是存放多个字符的

二维字符数组 是存放 多个字符串 每个字符串占一行

```
//不管是数值还是字符 的二维数组 在初始化的时候 是可以省略行标的 行数由具体初始化决定
char str[3][16]={"hehe","haha","heihei"};
str[0] ="hehe"; str[1]="haha"; str[2] ="heihei";

```




```
void test07()
{
 char str[3][16]={"hehe","haha","heihei"};

 //输出一个字符串 仅仅使用行标
 printf("%s\n",str[0]);
 printf("%s\n",str[1]);
 printf("%s\n",str[2]);

  //输出的是 字符串中的某个字符 必须用行标 和列表
  printf("%c\n", str[1][3]);
}
```

运行结果：

```
hehe
haha
heihei
a
Press any key to continue
```

二维字符数组获取键盘输入的字符串：

应用场景：将键盘 需要输入 多个独立的字符串 用户 必须单独存储好 请选择二维字符数组 输入的字符串个数 决定了二维数组的行数 输入字符串中的最大长度 决定了二维字符数组的列数。

```
void test08()
{
 //hehe haha xixi heihei
 char buf[4][16]={""};
 int i=0;

 //获取键盘的字符串
 for(i=0;i<4;i++)
 {
  //scanf("%s",&buf[i][0]);
  scanf("%s", buf[i]);//buf[i]已经代表是第i行的首元素地址
  }

 //输出字符串
 for(i=0;i<4;i++)
 {
 printf("buf[%d]=%s\n",i,buf[i]);
 }

}
int main(int argc,char *argv[])
{
 test08();
 return 0;
}
```

运行结果：

```
呵呵 哈哈  嘻嘻 啦啦
buf[0]=呵呵
buf[1]=哈哈
buf[2]=嘻嘻
buf[3]=啦啦
Press any key to continue
```

作业：

1、键盘输入10个int数据，对其 从小-->大排序 （封装成函数）

2、键盘输入一个字符串 “abcdef” 进行前后的颠倒（逆置）-->"fedcba" （封装成函数）

知识点4【函数概述】

1、函数的定义：实现函数功能、确定函数体、返回值类型、形参类型。 让函数存在

2、函数的声明：不是实现函数功能 仅仅是说明改函数有返回值类型、形参类型、函数名。

3、函数的调用：函数的执行。

1、函数的定义

```
//返回值类型：函数将来返回值的类型
//函数名：函数的入口地址
//形参：函数外部数据 传递到 函数内部的 桥梁
//函数体：具体的函数功能带
返回值类型 函数名(形参类型 形参)
{
 函数体;
}
```

2、函数声明：

```
返回值类型 函数名(形参类型 形参);
```

3、函数的调用

```
//函数外部的实际数据
函数名(实参);
```

案例：

```
//函数声明：告诉编译器 该函数存在 请通过编译。
void my_fun();

```

```c
int main(int argc,char *argv[])
{
 //函数的调用：函数名+()
 my_fun();
 return 0;
}

//函数的定义
void my_fun()
{
 printf("my fun\n");
 return;//1、返回函数结果 2、结束函数
}
```

可以省略函数声明：函数的调用 在 函数定义的下方 就可以省略函数声明

```c
#include<stdio.h>

//函数的定义
void my_fun()
{
 printf("my fun\n");
 return;
}

int main(int argc,char *argv[])
{
 //函数的调用：函数名+()
 my_fun();
 return 0;
}
```

知识点5【函数参数】（重要）

1、如果函数的形参 啥都不写 在调用的时候 可以传实参 只是实参得不到使用

```c
void my_fun();

int main(int argc,char *argv[])
{

```

```
 my_fun(10,20);
 return 0;
}

//如果函数的形参 啥都不写 在调用的时候 可以传实参 只是实参得不到使用
void my_fun()
{
 printf("my fun\n");
 return;
}
```

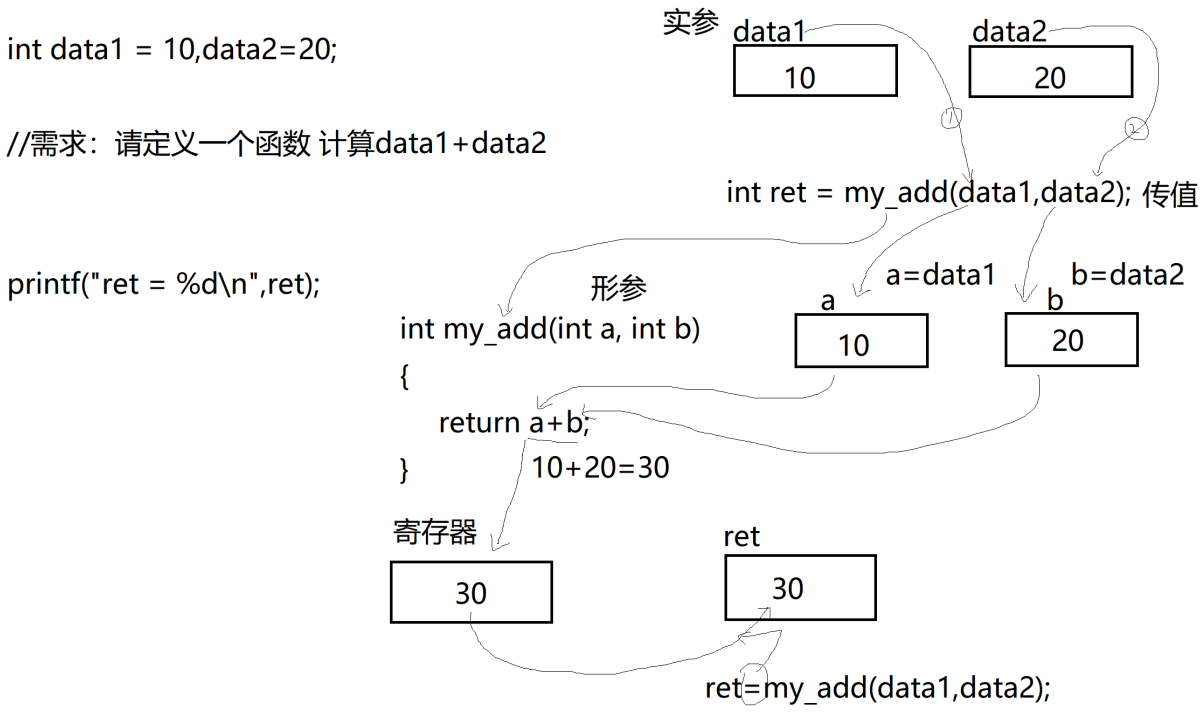
2、如果函数没有参数 请将形参写成 void

```
void my_fun(void);

int main(int argc,char *argv[])
{
 //my_fun(10,20);//err
 my_fun();
 return 0;
}
//如果函数的参数 为void 在调用的时候 就不要给函数传递实参
void my_fun(void)
{
 printf("my fun\n");
 return;
}
```

3、函数参数的传递

```
#include<stdio.h>

int my_add(int a, int b);
int main(int argc,char *argv[])
{
 int data1 = 10,data2=20;

 //需求：请定义一个函数 计算data1+data2
 int ret = my_add(data1,data2);

 printf("ret = %d\n",ret);//30

 return 0;
```

```
14 }
15 int my_add(int a, int b)
16 {
17  return a+b;
18 }
19
20
```



注意：

1、函数的形参 本质 是函数的局部变量。

2、形参 在函数定义的时候不会开辟空间，只在函数调用的时候才开辟空间。

3、形参 在函数结束的时候 才被释放。

```
1 int my_add(int a, int b) //a b就是形参
2 {
3  return a+b;
4 }
```

4、函数名代表的是函数的入口地址。

```
1 #include<stdio.h>
2
```

```c
int my_add(int a, int b);
int main(int argc,char *argv[])
{
 int data1 = 10,data2=20;

 //需求：请定义一个函数 计算data1+data2
 int ret = ((int(*)(int,int))(0x0040100F))(data1,data2);
  printf("%p\n", my_add);//my_add代表函数的入口地址

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

  return 0;
}
int my_add(int a, int b)
{
  return a+b;
}
```

5、函数的返回值 <=4字节 存放寄存器 >4字节 存放在栈区。

案例：自定义对应函数 求数组的最大值 和 最小值

```c
#include<stdio.h>
void my_input_arr(int a[5], int len)
{
 int i=0;
 printf("请输入%d个int数据\n",len);
 for(i=0;i<len;i++)
 {
 scanf("%d", &a[i]);
 }
  return;
}
void my_print_arr(int arr[5], int len)
{
  int i=0;
  for(i=0;i<len;i++)
  {
  printf("%d ",arr[i]);
```

```
 }
 printf("\n");
 return;
}

int my_max(int arr[5], int n)
{
 int tmp_max = arr[0];//假设第0个元素是最大的
 int i=0;
 for(i=0;i<n;i++)
 {
 if(tmp_max < arr[i])
 tmp_max = arr[i];
 }

 return tmp_max;
}

int my_min(int arr[5], int n)
{
 int tmp_min = arr[0];//假设第0个元素是最小的
 int i=0;
 for(i=0;i<n;i++)
 {
 if(tmp_min > arr[i])
 tmp_min = arr[i];
 }

 return tmp_min;
}
void test01()
{
 int arr[5]={0};
 int n = sizeof(arr)/sizeof(arr[0]);
 int max = 0,min = 0;

 //键盘给数组赋值
 my_input_arr(arr, n);

 //遍历该数组
```

```
 my_print_arr(arr, n);

 //计算数组的最大值
 max = my_max(arr, n);
 printf("max = %d\n",max);

 //计算数组的最小值
 min = my_min(arr, n);
 printf("min = %d\n",min);

 return;
}

int main(int argc,char *argv[])
{
 test01();
 return 0;
}
```

案例：分文件(了解)

main.c

```
#include<stdio.h>
#include"my_fun.h"
void test01()
{
 int arr[5]={0};
 int n = sizeof(arr)/sizeof(arr[0]);
 int max = 0,min = 0;

 //键盘给数组赋值
 my_input_arr(arr, n);

 //遍历该数组
 my_print_arr(arr, n);

 //计算数组的最大值
 max = my_max(arr, n);
 printf("max = %d\n",max);

 //计算数组的最小值
 min = my_min(arr, n);
```

```
 printf("min = %d\n",min);

 return;
}


int main(int argc,char *argv[])
{

 test01();
 return 0;
}
```

my_fun.c

```
#include<stdio.h>
void my_input_arr(int a[5], int len)
{
 int i=0;
 printf("请输入%d个int数据\n",len);
 for(i=0;i<len;i++)
 {
 scanf("%d", &a[i]);
 }
}
void my_print_arr(int arr[5], int len)
{
 int i=0;
 for(i=0;i<len;i++)
 {
 printf("%d ",arr[i]);
 }
 printf("\n");
}

int my_max(int arr[5], int n)
{
 int tmp_max = arr[0];//假设第0个元素是最大的
 int i=0;
 for(i=0;i<n;i++)
 {
 if(tmp_max < arr[i])
```

```
 tmp_max = arr[i];
 }

 return tmp_max;
}

int my_min(int arr[5], int n)
{
 int tmp_min = arr[0];//假设第0个元素是最小的
 int i=0;
 for(i=0;i<n;i++)
 {
 if(tmp_min > arr[i])
 tmp_min = arr[i];
 }

 return tmp_min;
}
```

my_fun.h

```
//声明函数
//extern 是告诉编译器 该函数 来自于 其他文件（extern 声明函数外部可用）
extern void my_input_arr(int a[5], int len);
extern void my_print_arr(int arr[5], int len);
extern int my_max(int arr[5], int n);
extern int my_min(int arr[5], int n);
```

运行结果：

```
请输入5个int数据
10 20 50 40 30
10 20 50 40 30
max = 50
min = 10
Press any key to continue
```

知识点6【变量的存储类别】

1、普通局部变量

2、静态局部变量

3、普通全局变量

4、静态全局变量