

## 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

### 3、函数参数的传递

注意:

案例: 自定义对应函数 求数组的最大值 和 最小值

案例: 分文件(了解)

### 知识点6【变量的存储类别】

## 知识点1【作业讲解】

```
1 void test01()
2 {
3     int arr[3][4]={0};
4
5     //获取键盘输入
6     int i=0,j=0;
7     for(i=0;i<3;i++)
8     {
9         for(j=0;j<4;j++)
10        {
11            scanf("%d", &arr[i][j]);
12        }
13    }
14
15    for(i=0;i<3;i++)
16    {
17        //求每一行的平均值
18        int sum = 0;
19
20        for(j=0;j<4;j++)
21        {
22            sum += arr[i][j];
23        }
24
25        printf("第%d行的平均值%d\n",i, sum/4);
26    }
27 }
28 int main(int argc,char *argv[])
29 {
```

```
30 test01();  
31 return 0;  
32 }
```

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:

```
1 #include<stdio.h>  
2 void test01()  
3 {  
4     int arr[3][4]={0};  
5  
6     //获取键盘输入  
7     int i=0,j=0;  
8     for(i=0;i<3;i++)  
9     {  
10        for(j=0;j<4;j++)  
11        {  
12            scanf("%d", &arr[i][j]);  
13        }  
14    }  
15  
16    for(j=0;j<4;j++)  
17    {
```

```

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:

```

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

```

## 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

```

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

## 字符数组的遍历

```
1 void test02()
2 {
3     char str[6]="hello";
4
5     //逐个字符遍历
6     int i=0;
7     for(i=0;i<6;i++)
8     {
9         printf("%c",str[i]);
10    }
11    printf("\n-----\n");
12
13    //字符数组 可以整体遍历 推荐
14    printf("%s\n", str);
15 }
```

运行结果：

C:\WORK\CODE\DATA05\00\_test\Debug\00\_test.exe

hello

-----

hello

Press any ke

**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);
}

```

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

```
1 void test05()
2 {
3     char buf[128]="";
4     printf("请输入一个字符串\n");
5     scanf("%s",buf);
6
7     printf("buf=%s\n",buf);
8
9 }
```

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**

```
1 void test05()
2 {
3     char buf[128]="";
4     printf("请输入一个字符串\n");
5     //scanf("%s",buf); //不能获取带空格的字符串
6     gets(buf); //获取带空格的字符串
7     printf("buf=%s\n",buf);
8 }
```

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：

```
1 void test05()
2 {
3     char buf[10]="";
4     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
hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
buf=hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
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 }
```

```
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：



```
请输入一个字符串
12345 12345 12345
buf=12345 123
Press any key to continue
```

### 提高案例：字符串的大小写 转换

```
1 void test06()
2 {
3     //'a' 97 'b' 98 'c' 99 ~ 'z' 122
4     //'A' 65 'B' 66 'C' 67 ~ 'Z' 90
5     //说明小写字母 和 大写字母的差值 是32
6
7     char ch = 'a';
8     //小写 变 大写 -32
9     ch = ch -32;//ch = ch - ('a'-'A');
10    printf("ch =%c\n",ch);
11
12    //大写 变 小写 +32
13    ch = ch+32;//ch = ch + ('a'-'A');
14    printf("ch =%c\n",ch);
15 }
```

运行结果：

```
ch =A
ch =a
Press any key to continue_
```

**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

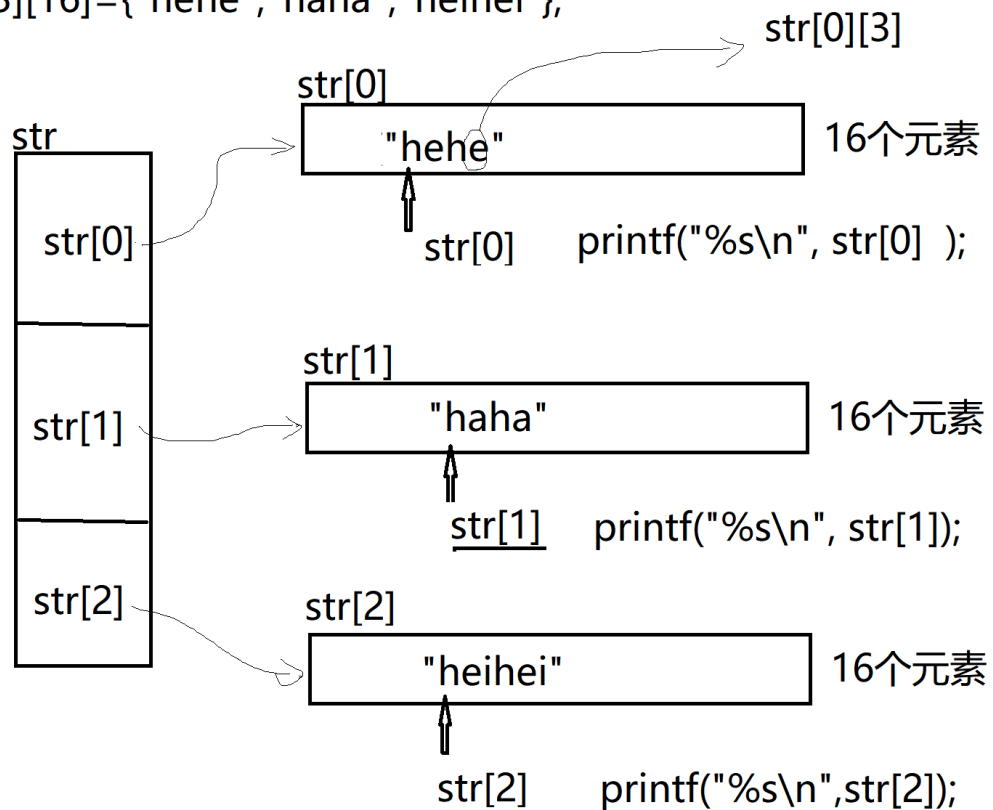
## 知识点3 【二维字符数组】（了解）

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

```
1 //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
2 char str[3][16]={"hehe","haha","heihei"};
3 str[0] ="hehe"; str[1]="haha"; str[2] ="heihei";
4
```

char str[3][16]={"hehe","haha","heihei"};



```
1 void test07()
2 {
3     char str[3][16]={"hehe","haha","heihei"};
4
5     //输出一个字符串 仅仅使用行标
6     printf("%s\\n",str[0]);
7     printf("%s\\n",str[1]);
8     printf("%s\\n",str[2]);
9
10    //输出的是 字符串中的某个字符 必须用行标 和列表
11    printf("%c\\n", str[1][3]);
12 }
```

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。

```
1 void test08()
2 {
3     //hehe haha xixi heihei
4     char buf[4][16]={""};
5     int i=0;
6
7     //获取键盘的字符串
8     for(i=0;i<4;i++)
9     {
10        //scanf("%s",&buf[i][0]);
11        scanf("%s", buf[i]); //buf[i]已经代表是第i行的首元素地址
12    }
13
14    //输出字符串
15    for(i=0;i<4;i++)
16    {
17        printf("buf[%d]=%s\n",i,buf[i]);
18    }
19
20 }
21 int main(int argc,char *argv[])
22 {
23     test08();
24     return 0;
25 }
```

运行结果：

```
1 呵呵 哈哈 嘻嘻 啦啦
2 buf[0]=呵呵
3 buf[1]=哈哈
4 buf[2]=嘻嘻
5 buf[3]=啦啦
6 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

```
1 //返回值类型：函数将来返回值的类型
2 //函数名：函数的入口地址
3 //形参：函数外部数据 传递到 函数内部的 桥梁
4 //函数体：具体的函数功能带
5 返回值类型 函数名(形参类型 形参)
6 {
7     函数体;
8 }
```

### 2、函数声明：

```
1 返回值类型 函数名(形参类型 形参);
```

### 3、函数的调用

```
1 //函数外部的实际数据
2 函数名(实参);
```

## 案例：

```
1 //函数声明：告诉编译器 该函数存在 请通过编译。
2 void my_fun();
3
```

```

4  int main(int argc, char *argv[])
5  {
6      //函数的调用：函数名+()
7      my_fun();
8      return 0;
9  }
10
11 //函数的定义
12 void my_fun()
13 {
14     printf("my fun\n");
15     return; //1、返回函数结果 2、结束函数
16 }

```

**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**

```

1  #include<stdio.h>
2
3  //函数的定义
4  void my_fun()
5  {
6      printf("my fun\n");
7      return;
8  }
9
10 int main(int argc, char *argv[])
11 {
12     //函数的调用：函数名+()
13     my_fun();
14     return 0;
15 }

```

## 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**

```

1  void my_fun();
2
3  int main(int argc, char *argv[])
4  {
5

```

```

6  my_fun(10,20);
7  return 0;
8  }
9
10 //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
11 void my_fun()
12 {
13     printf("my fun\n");
14     return;
15 }

```

## 2、如果函数没有参数 请将形参写成 void

```

1  void my_fun(void);
2
3  int main(int argc, char *argv[])
4  {
5      //my_fun(10,20); //err
6      my_fun();
7      return 0;
8  }
9  //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
10 void my_fun(void)
11 {
12     printf("my fun\n");
13     return;
14 }

```

## 3、函数参数的传递

```

1  #include<stdio.h>
2
3  int my_add(int a, int b);
4  int main(int argc, char *argv[])
5  {
6      int data1 = 10, data2=20;
7
8      //需求: 请定义一个函数 计算data1+data2
9      int ret = my_add(data1, data2);
10
11     printf("ret = %d\n", ret); //30
12
13     return 0;

```

```

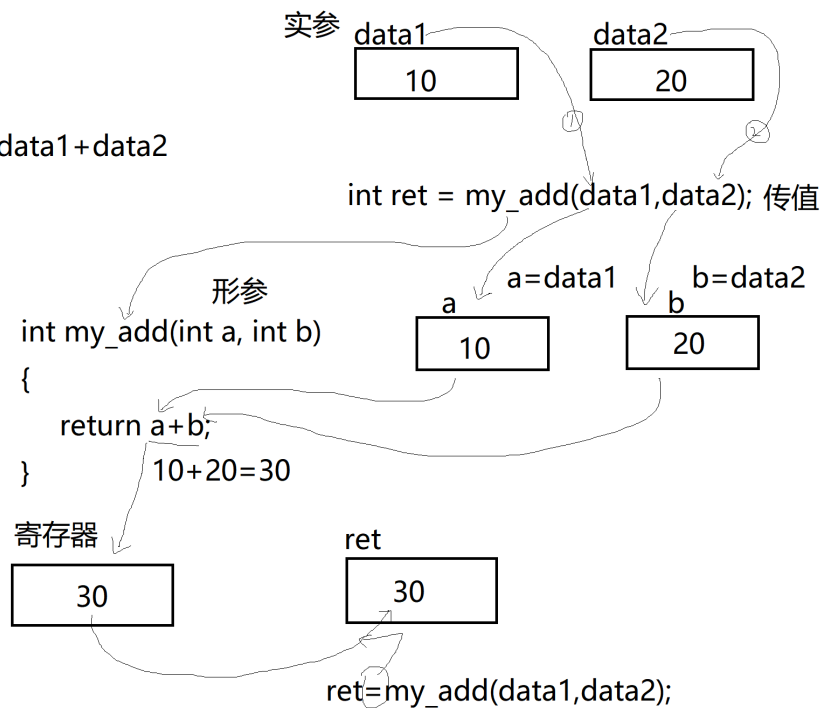
14 }
15 int my_add(int a, int b)
16 {
17     return a+b;
18 }
19
20

```

int data1 = 10,data2=20;

//需求: 请定义一个函数 计算data1+data2

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



### 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

```

```

3 int my_add(int a, int b);
4 int main(int argc, char *argv[])
5 {
6     int data1 = 10, data2=20;
7
8     //需求: 请定义一个函数 计算data1+data2
9     int ret = ((int (*)(int, int))(0x0040100F))(data1, data2);
10    printf("%p\n", my_add); //my_add代表函数的入口地址
11
12    printf("ret = %d\n", ret);
13
14    return 0;
15 }
16 int my_add(int a, int b)
17 {
18     return a+b;
19 }

```

5、函数的返回值  $\leq 4$ 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

```

1 #include<stdio.h>
2 void my_input_arr(int a[5], int len)
3 {
4     int i=0;
5     printf("请输入%d个int数据\n", len);
6     for(i=0; i<len; i++)
7     {
8         scanf("%d", &a[i]);
9     }
10    return;
11 }
12 void my_print_arr(int arr[5], int len)
13 {
14     int i=0;
15     for(i=0; i<len; i++)
16     {
17         printf("%d ", arr[i]);

```



```
18  }
19  printf("\n");
20  return;
21  }
22
23  int my_max(int arr[5], int n)
24  {
25      int tmp_max = arr[0]; //假设第0个元素是最大的
26      int i=0;
27      for(i=0;i<n;i++)
28      {
29          if(tmp_max < arr[i])
30              tmp_max = arr[i];
31      }
32
33      return tmp_max;
34  }
35
36  int my_min(int arr[5], int n)
37  {
38      int tmp_min = arr[0]; //假设第0个元素是最小的
39      int i=0;
40      for(i=0;i<n;i++)
41      {
42          if(tmp_min > arr[i])
43              tmp_min = arr[i];
44      }
45
46      return tmp_min;
47  }
48  void test01()
49  {
50      int arr[5]={0};
51      int n = sizeof(arr)/sizeof(arr[0]);
52      int max = 0,min = 0;
53
54      //键盘给数组赋值
55      my_input_arr(arr, n);
56
57      //遍历该数组
```

```

58  my_print_arr(arr, n);
59
60  //计算数组的最大值
61  max = my_max(arr, n);
62  printf("max = %d\n",max);
63
64  //计算数组的最小值
65  min = my_min(arr, n);
66  printf("min = %d\n",min);
67
68  return;
69 }
70
71 int main(int argc,char *argv[])
72 {
73     test01();
74     return 0;
75 }

```

## 案例：分文件(了解)

main.c

```

1  #include<stdio.h>
2  #include"my_fun.h"
3  void test01()
4  {
5      int arr[5]={0};
6      int n = sizeof(arr)/sizeof(arr[0]);
7      int max = 0,min = 0;
8
9      //键盘给数组赋值
10     my_input_arr(arr, n);
11
12     //遍历该数组
13     my_print_arr(arr, n);
14
15     //计算数组的最大值
16     max = my_max(arr, n);
17     printf("max = %d\n",max);
18
19     //计算数组的最小值
20     min = my_min(arr, n);

```

```

21     printf("min = %d\n",min);
22
23     return;
24 }
25
26
27 int main(int argc,char *argv[])
28 {
29
30     test01();
31     return 0;
32 }

```

## my\_fun.c

```

1  #include<stdio.h>
2  void my_input_arr(int a[5], int len)
3  {
4      int i=0;
5      printf("请输入%d个int数据\n",len);
6      for(i=0;i<len;i++)
7      {
8          scanf("%d", &a[i]);
9      }
10 }
11 void my_print_arr(int arr[5], int len)
12 {
13     int i=0;
14     for(i=0;i<len;i++)
15     {
16         printf("%d ",arr[i]);
17     }
18     printf("\n");
19 }
20
21 int my_max(int arr[5], int n)
22 {
23     int tmp_max = arr[0];//假设第0个元素是最大的
24     int i=0;
25     for(i=0;i<n;i++)
26     {
27         if(tmp_max < arr[i])

```

```

28  tmp_max = arr[i];
29  }
30
31  return tmp_max;
32 }
33
34 int my_min(int arr[5], int n)
35 {
36  int tmp_min = arr[0]; //假设第0个元素是最小的
37  int i=0;
38  for(i=0;i<n;i++)
39  {
40  if(tmp_min > arr[i])
41  tmp_min = arr[i];
42  }
43
44  return tmp_min;
45 }

```

## my\_fun.h

```

1 //声明函数
2 //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）
3 extern void my_input_arr(int a[5], int len);
4 extern void my_print_arr(int arr[5], int len);
5 extern int my_max(int arr[5], int n);
6 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