昨日作业

1.在终端输入一个文件名，判断文件是否为空，如果不为空，判断文件是否是普通文件，如果是普通文件，就判断是否具有写权限，没有写权限，就加上写权限，并在文件最后追加一行”hello world “

```
#!/bin/bash
read -p "请输入文件名: "  file
#文件不为空和是普通文件两个条件同时满足执行第一个语句块
if [ -s "$file" -a -f "$file" ]
then
#判断文件是否有写权限
        if [ -w "$file" ]
        then
                echo 普通文件有写权限
        else
                echo 普通文件无写权限，添加写权限:
                chmod u+w $file
                echo hello world >> $file
        fi
else
        echo 2
fi
```

2.在家目录/home/ubuntu 下创建两个目录文件 file_dir 和 dir_dir,如果家目录下有这两个文件夹，就不需要创建，询问用户是否要对目录清空[Y/N], 输入一个文件路径，判断这个文件路径是否存在，如果存在，把这个目录下的目录文件复制到 dir_dir 中，如果是其他文 件，复制到 file_dir 下，统计复制的文件和目录文件的个数，并打印出来

```
#!/bin/bash
path1=/home/ubuntu/file_dir
path2=/home/ubuntu/dir_dir
#小判断是否是目录文件
if [ -d "$path1" ]
then
        echo "file_dir文件存在"
        read -p "是否清空文件(Y=yes,N=no)" answer
    if [ "$answer" = "Y" ]
        then
        echo 清空file_dir文件
           rm -r  "$path1"/*
```

```
   fi
else
        echo "file_dir文件不存在，创建文件"
        mkdir "$path1"
fi
if [ -d "$path2" ]
then
        echo "dir_dir文件存在"
        read -p "是否清空文件(Y=yes,N=no)" answer
    if [ "$answer" = "Y" ]
    then
        echo 清空dir_dir文件
           rm -r "$path2"/*
    fi
else
        echo "dir_dir文件不存在，创建文件"
        mkdir "$path2"
fi
#path3=$1 #通过命令行传参
read -p "请输入目标路径" path3  #这个路径一定输入绝对路径
count1=0
count2=0
if [ -d "$path3" ]
then
        echo "$path3 存在"
        for file in `ls "$path3"`
        do
                echo $file
                if [ -d "$path3"/"$file" ]
                then
                        echo $file是文件夹
                        cp  $path3/$file  $path2 -r
                        ((count1++))
                else
                        echo $file 是其他文件
                        cp   $path3/$file  $path1
                        ((count2++))
                fi
        done
        echo "复制的目录文件有$count1个复制的其他文件有 $count2个"
```

```
else
        echo "$path3"不存在
fi
```

shell脚本中的case语句

格式:

case $变量名 in

匹配项1)

语句1

；；

匹配项2)

语句2

；；

。。。

*)

语句n

;;

esac

```
#!/bin/bash

read -p "请输入" var
case $var in
        "22091")
                echo 正在上预科
                ;;
        "22081")
                echo 正在上linux
                ;;
        "22071")
                echo 正在上c++
                ;;
        *)
                echo 其他课程
                ;;
esac

```

使用注意事项：

1.*为了给整个语句托底，其他匹配项匹配不到的都交给*对应的语句块

2.一个匹配项里面可以有多个匹配内容，用|来分开，表示或

```
#!/bin/bash

read -p "请输入" var
case $var in
        "上海中心")
                echo 坐标在上海浦东新区
                ;;
        "济南中心")
                echo 济南高新区
                ;;
        "北京中心"|"研发中心"|"创客中心")
                echo 北京海淀区
                ;;
        *)
                echo 五湖四海
                ;;
esac

```

3.可以用[x-y]表示匹配的一个范围

4.[字符列表]可以用来匹配在[]内部的任意字符

```
#!/bin/bash

read -p "请输入" var
case $var in
        [0123456789])
                echo 数字
                ;;
        [A-Za-z])
                echo 字母
                ;;
        [,.?!])
                echo 标点符号
```

```
                ;;
        *)
                echo 其他字符
                ;;
esac

```

shell脚本里的while循环语句

格式：

while [循环条件]
do
    循环体
done
解释：当循环条件被满足时，循环体执行

```
#!/bin/bash

i=0
sum=0
while [ $i -le 100 ]
do
        ((sum+=i))
        ((i++))
done
echo $sum
```

shell脚本里的while死循环

while [1/0]
while :
while ["a" == "a"]
while true `

shell脚本里的for循环

格式1：

c语言风格

for((表达式1；表达式2；表达式3))

{

    循环体

}

```
#!/bin/bash

i=0
sum=0
for((i=0;i<=100;i++))
        {
                ((sum+=i))
        }
        echo $sum
```

格式2

shell脚本风格

for 变量 in 单词列表

do

    循环体

done

单词列表写法

1.for 变量 in 单词1 单词2 .。。

```
#!/bin/bash

for var in 北京 上海 南京 重庆
do
        echo $var
done
```

2.for 变量 in `ls 路径`

```
#!/bin/bash
#指定一个操作路径
```

```
3
4 path=/home/ubuntu/1
5 #在当前路径下创建一个文件夹1
6 mkdir ./1
7 #循环遍历指定的路径
8 for var in `ls $path`
9 do
10 #进行文复制
11         cp -r "$path"/$var ./1
12 done
```

3.for 变量 in {起始字符..终止字符}

```
1 #!/bin/bash
2 #循环遍历a-z
3
4 for var in {a..z}
5 do
6         echo $var
7 done
```

4.如果把in和单词列表省略，那么在命令行传递单词列表参数

```
1 #!/bin/bash
2
3 for var
4 do
5         echo $var
6 done
```

5.可以把数组当作一个单词列表

```
1 #!/bin/bash
2 arr=(北京 上海 南京 重庆)
3 for var  in ${arr[@]}
4 do
5         echo $var
6 done
```

补充

break:结束当前循环

continue：结束当前循环进入下次循环

注意：break和continue后面可以加一个数字，表示跳出n层循环

break n == contine n+1

sleep:睡眠命令

sleep n：睡眠n秒

shell脚本里的select循环

格式：

select 变量 in 单词列表

do

循环体

done

特点：增强了人机的交互，一般select是死循环，没有办法结束，只能强制结束，或者遇到break结束

select循环一般和case语句联合使用

```
#!/bin/bash
arr=(北京 上海 南京 重庆)
select var in ${arr[@]}
do
        case $var in
                "北京")
                        echo 北京烤鸭
                        ;;
                "上海")
                        echo 上海生煎
                        ;;
                "南京")
                        echo 南京鸭血粉丝汤
                        ;;
                "重庆")
                        echo 重庆鸡公煲
                        ;;
                *)
                        echo 其他;;
        esac

        #echo haha

```

```
done

```

多行注释

1.<<字符串
注释内容
字符串
2.
: '
注释内容
'

函数

格式:

function 函数名（）
{
    函数体
    返回值
}

性质

1.shell脚本里面声明函数使用function
2.shell脚本函数内部返回值没有数据类型
3.shell脚本函数没有参数列表，传参使用位置变量传参
4.shell脚本函数体也是在{}内部
5.shell脚本通过ruturn去返回的数值只能在0-255范围内部，如果想要返回超过这个范围的数值，需要echo实现
6.函数正常不会执行，被调用时函数执行

函数的调用

格式:
函数名 参数1 参数2 参数3.。

```
#!/bin/bash
function aaa()
```

```
{
        #$0固定是脚本名
        echo $0
        #$1是函数传递的第1个参数
        echo $1
        echo $2
}

#函数的调用
aaa hello nihao hahha
```

函数返回值的获取

1.通过return返回返回值，$?获取返回值：这种方式只能获取0-255范围的数值

```
#!/bin/bash
function add()
{
        #$0固定是脚本名
        #$1是函数传递的第1个参数
        sum=$(($1+$2))
        return $sum
}

#函数调用
add 300 50
#返回值通过$?接收
echo $?

```

2.可以通过全局变量获取返回值，在shell脚本里面变量属性默认是全局可用的，如果想变成程局部的，得有一个标识符声明local

local 变量名：声明变量为局部变量

```
#!/bin/bash
function add()
{
        #$0固定是脚本名
        #$1是函数传递的第1个参数
        #全局变量接收计算的结果
        sum=$(($1+$2))
```

```
8
9  }
10
11 #函数调用
12 add 300 50
13 #函数执行的结果通过全局变量接收
14 echo $sum
```

3.通过echo返回返回值

```
1  #!/bin/bash
2  function add()
3  {
4          #$0固定是脚本名
5          #$1是函数传递的第1个参数
6          #全局变量接收计算的结果
7          sum=$(($1+$2))
8          #通过echo返回结果
9          echo $(($1+$2))
10 #        return $sum
11 }
12
13 #把返回结果赋值给变量var
14 var=`add 300 50`
15 echo $var
16
```

c语言里的重要关键字

1.static:静态关键字

作用：延长局部变量的生命周期直到程序结束

限制全局变量和函数的作用域，只能在当前文件使用

被static修饰的局部变量只能初始化一次,static修饰的局部变量不初始化默认数值为0

```
1  int main()
2  {
3      int i;
4      for(i=0;i<10;i++)
```

```
    {
            static int a;
            a++;
            printf("%d\n",a);
    }
}
结果：1 2 3 4 5 6 7 8 9 10
```

2.extern

外部引用关键字

```
1.c:
    #include <stdio.h>
//引用外部全局变量
extern int a;
int main(int argc, const char *argv[])
{
        printf("%d\n",a);
        return 0;
}
2.c
int a=10;
```

注意：被static关键字修饰的全局变量无法被外部引用，如果想用，定义一个全局指针指向该变量，把指针引用

3.const 常量化关键字

谁被const修饰了，谁就无法被操作修改

```
#include <stdio.h>
//引用外部全局变量
extern int *b;
int main(int argc, const char *argv[])
{
         int const a=10;
         //通过指针修改const变量的数值
        int *p=&a;
        *p=100;
        printf("%d\n",a);
```

```
        return 0;
}
```

```
const int *p;//指针指向的空间的数值无法被修改
int const *p;//指针指向的空间的数值无法被修改
int const *const p;//指针的指向不可以被修改，指针指向的空间的数值无法被修改
int * const p;//指针的指向不可以被修改
```

4.register 关键字

寄存器存储类型：被register关键字修饰的变量存储在寄存器中，寄存器没有地址，所以不可以对register修饰的变量取地址

如果寄存器存满了或者不想被申请，那么此变量和auto关键字修饰的变量得到一样的待遇

5.volatile关键字

防止编译器优化，每次cpu取数据都是在内存中直接取

c语言里的构造数据类型

1.结构体

定义：

struct 结构体名

{

成员类型1 成员变量1；

。。。

};

结构体变量定义：

struct 结构体名 结构体变量名；

结构体变量访问成员：

结构体变量名.成员名

结构体指针定义：

struct 结构体名 *结构体指针名

结构体指针访问结构体成员：

结构体指针名->成员名

结构体数组定义：

struct 结构体名 结构体数组名[长度];

结构体数组访问成员的成员变量：

结构体数组名[下标].成员名

字节对齐：

结构体为了在数据读写的时候效率高，我们默认采用了结构体字节对齐机制，给结构体变量申请空间时，按照成员中类型最大的成员的类型长度给每一成员个变量申请空间，32位系统结构体最大对齐是4字节对齐，64位系统中结构体最大8字节对齐

```
struct test
{
    int a;      //8
    char b;      //
    char e;
    short d;
    double c;  //8
};    16字节
typedef struct
{
        int a;
        char b;
        short d;
        char e;
        double c;
}A;  24字节

```

2.共用体

定义以及成员访问请参考之前的上课笔记

字节对齐：共用体也是遵循字节对齐规则

```
union a
{
    int a;
    char b[21];
    short c;
};
24字节
```

3.枚举

枚举的作用用来声明一组常量

一般一个月30/31天，一个星期有7天，一天有24四小时，一年有四季，可以将这些具有不同状态的量给封装成一个枚举类型

定义格式

enum 枚举名

{

    成员1,

    成员2,

    成员3

};

枚举变量的定义格式：

enum 枚举名 枚举变量名

性质

1.枚举成员的数值默认第一个成员为0,依次向下递增1

```
enum week
{
        Mon,
        Tue,
        Win,
        Thur,
        Fri,
        Sat,
        Sun,
};
int main(int argc, const char *argv[])
{
        printf("%d %d %d %d %d %d %d\n",Mon,Tue,Win,Thur,Fri,Sat,Sun);
        return 0;
}
```

2.枚举成员数值可以自己指定

```c
#include <stdio.h>
enum week
{
        Mon=1,
        Tue=2,
        Win=3,
        Thur=4,
        Fri=5,
        Sat=6,
        Sun=7,
};
int main(int argc, const char *argv[])
{
        //定义枚举变量
        enum week w1;
        w1=Mon;
        switch(w1)
        {
                case Mon:
                                printf("周一\n");
                                break;
                default:
                                printf("其他\n");
                                break;
        }
        printf("%d %d %d %d %d %d %d\n",Mon,Tue,Win,Thur,Fri,Sat,Sun);
        return 0;
}
```

3.可以给枚举某一成员指定数值，下面的成员从这个数值开始依次+1

```c
#include <stdio.h>
enum week
{
        Mon,
        Tue,
        Win,
        Thur=4,
        Fri,
        Sat,
```

```
10          Sun,
11  };
12
```

4.枚举成员是全局可以起作用的，不能定义和他同名的全局变量
5.枚举成员是常量，数值不可以被修改

任务
1.复习今天shell脚本内容，把出现的代码再回顾回顾
2.完成下发的c练习题