Operating System Labs Project 0

Project 0a: Get familiar with Linux

Part 1

Task1: s1.sh

创建文件夹 mkdir xxx
$1 命令行参数：因为$0 是脚本文件名，所以用$1
>> 输出重定向：将输出方向从屏幕变为指定文本文件
>为清空输入，而>>则是追加，若文件名不存在，还会自动创建
所以综合代码为 s1.sh 所展示：

```
#!/bin/bash
mkdir $1
echo "Wu Zijing">>$1/name.txt
echo "10185102141">>$1/stno.txt\
```

Task2:s2.sh

此次脚本代码量较大，所以在代码中也写了注释，在这里也做一次详细解释，因为任务如果用 C 语言去执行是比较简单的，因此最初的思路便是写出 C 语言的代码，然后参照语法去逐个翻译成 shell 脚本。

C 语言的思路：遍历文件夹下的所有文件，取出文件名，如果首字母为 b，进行处理，get 到他的权限信息和 owner，然后输出到输出文件。因为遍历的话，文件本身就是按照字典序的，所以省去了排序的部分。

实践为 shell 脚本：首先是如何遍历整个文件夹，查询知道语法是：for file in /usr/bin/*,那么 file 就是每次循环中的文件名。在循环中，我们首先坐第一件事，就是去除路径，对于文件 f,此时 file 并不是 f,而是/usr/bin/f，所以要先去除/usr/bin，用到 substr 的字串语法 ${file:9}，紧接着我们首先用一个 filename 存下这个去除路径后的文件名，作为第一部分信息，我们还需要 owner 和 permission 信息，分别称作第二部分和第三部分。

继续按照 C 的思路，先查看 filename 第一个字母是否是 b，于是用 if 语句进行判断，对于取出 filename 的第一个字母，我们仍旧采用 substr 的语法：if [[${file:9:1} == "b"]]来筛选。

然后我们便得到了全部首字母为 b 的文件名，下一件事就是如何得到 owner 和 permission 的信息，上课知道 ls -l 的指令是可以得到的，那么如何将这部分信息变成字符串呢？通过搜索引擎得知：var＝$(instruct)，那么 var 就存储了指令的内容，所以我们把 ls -l filename 放入$()中并用一个 information 来接收，就实现了将 permission 和 owner 存储在其中。

那么现在只剩下最后一个问题，如何从 information 中提取 owner 和

permission？我的想法是通过空格分隔，我们并不知道 owner 的下标，也不知道长度，但是我知道它在是 information 的第三个信息，于是 owner 就一定在第二个空格和第三个空格之间，我们假设第二个空格的下标是 begin,第三个空格的下标是 end，那么开始下标是 begin+1,长度是 end-begin-1，通过字串语法就可以得到 owner 的信息了，所以最终任务变成了找到第二个空格和第三个空格的下标。需要知道：如何遍历字符串，如何判断字符串等于空格。
遍历字符串：for i in $(seq 1 ${#information})
判断等于空格：if [[${information:i-1:1} == " "]]
再设置一个 count 变量来统计空格数量，那么就可以得到 begin,end
permission 同理可获得

最终用字符串拼接将 filename,owner,permission 进行拼接：
result=$filename" "$owner" "$permission #connect
echo $result>>out.txt
输出到 out.txt 中

以上为 s2.sh 的全部思路，完整代码附录在文件夹中

Part2 debug set_operation.c

首先统计一下全部的 bug:

21-31 行：check 函数，对于链表，没有判断 head->number 是否等于 num
26 行：语法错误，多写了一个 ->符号
89 行：for(i=0;i<=A_size;i++)循环进行了 A_size+1 次，应该去掉等于号
107 行：check 返回 1 代表在链表中，因此应该去掉感叹号
133 行：同理，也要去掉 check 前面的感叹号

再描述 debug 过程：

断点设置：63 行（A 链表输入完）
83 行（B 链表输入完）
102 行（A 到 A2 复制完）
127 行（A=A-B 计算完）
153 行（B=B-A 计算完）
变量监视：
A_head,B_head,A2_head
因为链表的后续节点不方便检测，所以再源码中添加了几个 output 来输出链表。

debug 过程：
编译失败：去除 26 行多余的 ->

输入：使用了测试样例
3 3 4 5
2 4 1

63 行断点触发：查看整个 A 链表的 number，正常
83 行断点触发：查看整个 B 链表的 number，正常
102 行断点触发：查看 A2，异常，检测整个 A2 的地址，发现有 p3=NULL，访问 number 异常，查看代码，发现 89 行多了=号

重新运行代码：
102 行断点触发：A2 正常
127 行断点触发：结果异常，查看 check 函数，发现没有比较 head 节点本身，去除 next

重新运行代码：
127 行断点触发：结果异常，发现保留的是 A 与 B 的交集，查看代码，发现 check 前面多了一个感叹号，去除

重新运行代码：
127 行断点触发：结果正常
153 行断点触发：结果异常，发现与 127 类似，去除感叹号

重新运行代码：全部正常，退出 debug

Bonus1.sh

NR 表示行数
S1 是第二个参数，也就是文件名
awk 'NR==10{print}' $1 :输出文件中行数等于 10 的信息

Bonus2.sh

同样，对于 C 语言，这道题也有清晰的思路，故尝试能按照 C 的流程来进行翻译。但是在创建数组/二维数组时发现难度极大，然后查询了 awk 的语法，发现 awk 可以通过空格数量来计算列数，以 NF 表示，通过换行符的数量计算行数，以 NR 表示，并且在 awk 内部是支持循环和字符串拼接的，所以抛去二维数组的想法，改用一维数组进行存储，从第二行开始，每一列的结果与前面的结果进行拼接。

即 if(NR==1){res[i]=$i}
else{res[i]=res[i]" "$i}
如果行数为 1，那么赋值，否则拼接
最后通过循环输出：for(j=1;j<=NF;j++){print res[j]}

以上为 Bonus2.sh 的全部思路，完整代码附录在文件夹中

Project 0b: Sorting

此部分因为是 C 语言的部分，所以都是比较熟悉的做法：建立数组存储数据，排序，写回。唯一需要注意的点就是数据的组数是未知的，这影响到我们开数组的规模取舍，本份代码的做法是对输入数据进行两次读入，第一次仅计算数据组数，即

```
while (1) {
    rc = read(fd, &r, sizeof(rec_t));
        if (rc == 0)
            break;
        if (rc < 0) {
            perror("read");
            exit(1);
        }
        count++;
  }
```

那么 count 就成功存储了数据组数

然后就可以通过 malloc 创建动态数组，并进行第二次读入

```
rec_t *array;
array = (rec_t*) malloc (sizeof(rec_t)*count);
 if(array == NULL){
    perror("No Memory!");
    exit(1);
 }

fd = open(inFile, O_RDONLY);
for(i = 0;i < count;i++)
    read(fd, &array[i], sizeof(rec_t));
```

然后通过自定义一个 cmp 函数进行排序，调用 qsort 之后进行写回，整体思路比较简单，对于各种 error 的错误信息可以直接仿照 generate.c 中的格式。
完整代码请参照文件夹

实验总结：本次实验时间主要花费在 shell 的部分，因为对于语法的了解太少了，需要查阅大量的博客。此外，shell 对于空格实在是太严格了，最初 for 和 if 中总是多加空格或者少加，以为和 c 一样不会对其敏感，但没想到 if[a == b]的空格不能多也不能少，因为空格花费了大量时间调试，要引以为戒……