1. 代码实践

```
#include <iostream>
#include <string>
#define MAX 1000
using namespace std;

//联系人结构体
struct Person
{
    string m_name; //姓名
    int m_Sex; //性别 1：男 2：女
    int m_Age; //年龄
    string m_phone; //电话
    string m_Addr; //住址
};

//通讯录结构体
struct Addressbooks
{
    struct Person personArray[MAX];
    int m_size; //通讯录中当前的人数
};

void addPerson(Addressbooks *abs)
{
    //先判断通讯录是否已满，如果满了就不再添加
    if(abs->m_size==MAX)
        cout << "通讯录已满，请勿继续添加" << endl;
    else
    {
        //添加具体联系人
        string name;
        cout << "请输入姓名：  " << endl;
        cin >> name;
        abs->personArray[abs->m_size].m_name=name;

        int sex; //性别 1：男 2：女
        while(true)
        {
            cout << "请输入性别：  " << endl;
            cin >> sex;
            if(sex==1||sex==2)
            {
```

```
                abs->personArray[abs->m_size].m_Sex=sex;
                break;
            }
            else
               cout << "输入有误，请重新输入" << endl;
        };

        int age; //年龄
        cout << "请输入年龄：  " << endl;
        cin >> age;
        abs->personArray[abs->m_size].m_Age=age;

        string phone; //电话
        cout << "请输入电话：  " << endl;
        cin >> phone;
        abs->personArray[abs->m_size].m_phone=phone;

        string addr; //住址
        cout << "请输入住址：  " << endl;
        cin >> addr;
        abs->personArray[abs->m_size].m_Addr=addr;

        abs->m_size++;
        cout << "添加成功" << endl;

        system("pause"); //请按任意键继续
        system("cls"); //清屏
    }
}

void showPerson(Addressbooks *abs)
{
    if(abs->m_size==0)
    {
        cout << "当前记录为空" << endl;
    }
    else
        for(int i=0;i<abs->m_size;i++)
        {
            cout << "姓名：  " << abs->personArray[i].m_name << "\t" ;
            cout << "性别：  " << (abs->personArray[i].m_Sex==1?"男":"女
") << "\t" ;
            cout << "年龄：  " << abs->personArray[i].m_Age << "\t" ;
            cout << "电话：  " << abs->personArray[i].m_phone << "\t" ;
```

```
            cout << "地址：  " << abs->personArray[i].m_Addr << "\t" ;
            cout << endl;
        }
    system("pause"); //请按任意键继续
    system("cls"); //清屏
}

int isExist(Addressbooks *abs,string name)
{
    for(int i=0;i<abs->m_size;i++)
    {
        if(abs->personArray[i].m_name==name)
        {
            return i;   //如果找到的话，返回下标
        }
    }
    return -1;
}


void delectPerson(Addressbooks *abs)
{
    cout << "请输入删除联系人的姓名：  " << endl;
    string name;
    cin >> name;
    if(isExist(abs,name)==-1)
    {
        cout << "查无此人" << endl;
    }
    else
    {
        for(int i=isExist(abs,name);i<abs->m_size;i++)
        {
            abs->personArray[i]=abs->personArray[i+1];
        }
        abs->m_size--;
        cout << "删除成功" << endl;
    }
    system("pause"); //请按任意键继续
    system("cls"); //清屏
}

void searchPerson(Addressbooks *abs)
{
    cout << "请输入您要查找的联系人：  " << endl;
```

```
    string name;
    cin >> name;
    int ret=isExist(abs,name);
    if(ret!=-1)
    {
        cout << "姓名：  " << abs->personArray[ret].m_name << "\t" ;
        cout << "性别：  " << (abs->personArray[ret].m_Sex==1?"男":"女
") << "\t" ;
        cout << "年龄：  " << abs->personArray[ret].m_Age << "\t" ;
        cout << "电话：  " << abs->personArray[ret].m_phone << "\t" ;
        cout << "地址：  " << abs->personArray[ret].m_Addr << "\t" ;
        cout << endl;
    }
    else
    {
        cout << "该联系人不存在" << endl;
    }
    system("pause"); //请按任意键继续
    system("cls"); //清屏
}
```

2. 计算机基础知识整理

2.1 死锁的四个必要条件：

• 互斥条件：一个资源一次只能被一个进程使用

• 请求保持条件：一个进程因请求资源而阻塞时，对已经获得资源保持不放

• 不可抢占条件：进程已获得的资源在未使用完之前不能强行剥夺

• 循环等待条件：若干进程之间形成一种头尾相接的循环等待资源的关系

2.2　死锁处理：

• 预防死锁：破坏产生死锁的 4 个必要条件中的一个或者多个；实现起来比较简单，但是如果限制过于严格会降低系统资源利用率以及吞吐量

• 避免死锁：在资源的动态分配中，防止系统进入不安全状态(可能产生死锁的状态)-如银行家算法

• 检测死锁：允许系统运行过程中产生死锁，在死锁发生之后，采用一定的算法进行检测，并确定与死锁相关的资源和进程，采取相关方法清除检测到的死锁。实现难度大

• 解除死锁：与死锁检测配合，将系统从死锁中解脱出来（撤销进程或者剥夺资源）。对检测到的和死锁相关的进程以及资源，通过撤销或者挂起的方式，释放一些资源并将其分配给

处于阻塞状态的进程，使其转变为就绪态。实现难度大

3. 当日工作总结

① 熟悉了一些 Git 相关操作

② 上传学习笔记至小组仓库。