@TOC

🌟如何构思?

根据C语言所存在的数据类型，我们很直观的联系我们手机上的通讯录，那么通讯录肯定是以每个联系人为单位的，而联系人是一个复合的数据集合体，他不能单独由整形或者字符串等简单表示，我们自然想到C语言结构体类型，`struct PeoInfo 就可以代表一个联系人的信息`。我们创建这样的结构体：

```
//表示人的信息
struct PeoInfo {
    char name[20];
    char sex[5];
    char tele[12];
    int age;
    char address[30];
};
```

这样就`依靠结构体来实现联系人的信息表示`。那么进一步考虑一个通讯录是什么样子，大的通讯录是以N个联系人为单位组成的，这是显然的，那么我们可以增加一个计数变量来保存当前的通讯录有多少个联系人，这样一来就可以实现通讯录的定义，同样使用结构体：

```
//封装通讯录
struct Contact {
    struct PeoInfo data[100];  //
    int sz;
};
```

😊 **考虑联系人属性**：

- `姓名`
- `性别`
- `电话号码`
- `年龄`
- `地址`

考虑设计时能不能更加灵活，比如我们通讯录的数据上限个数能不能将来好根据需求变化，数组的大小能否变化，`可以考虑将这些变量的大小采用宏定义：`

```
//使用宏定义,提高后续拓展性
#define MAX 100
#define MAX_NAME 20
#define MAX_SEX 5
#define MAX_TELE 12
#define MAX_ADDRESS 30
//表示人的信息
```

```
struct PeoInfo {
    char name[MAX_NAME];
    char sex[MAX_SEX];
    char tele[MAX_TELE];
    int age;
    char address[MAX_ADDRESS];
};

//封装通讯录
struct Contact {
    struct PeoInfo data[MAX];
    int sz;
};
```

至此，我们考虑主函数main.c的设计

- 简易的交互提示菜单

```
//简易菜单
void menu() {
    printf("---------------------------\n");
    printf("----1.Add--------2.Del----\n");
    printf("----3.Search-----4.Modify-\n");
    printf("----5.Show-------6.Sort---\n");
    printf("-----------0.Exit---------\n");
    printf("---------------------------\n");
}
```

这里注意要实现的功能：

```
1.增加联系人
2.删除联系人
3.查找联系人
4.修改联系人
5.排序
6.退出程序
```

- **初始化通讯录（即将通讯录‘0’化）**

```
//创建通讯录
    struct Contact con;
    //初始化通讯录
    InitContact(&con);
```

创建一个通讯录（结构体变量）con，并且通过结构体指针传参的方式来进行功能操作，这很重要，指针传参可以有效减少程序内存消耗开支，传指针更加直观。

- **switch分支语句实现功能控制**

```
menu();
```

```
        printf("请输入你想要的操作：");

        scanf("%d", &input);
        switch (input)
        {
        case 1:  //增加信息
            AddContact(&con);
            break;
        case 2:
            DelContact(&con);
            break;
        case 3:
            Search(&con);
            break;
        case 4:
            Modify(&con);
            break;
        case 5:
            ShowContact(&con);
            break;
        case 6:
            Sort(&con);
            break;
        case 0:
            printf("退出程序.");
            break;
        default:
            break;
        }
```

- **循环实现多次功能**

main.c

将功能整合一下即可，这是主函数的实现：

```
# define _CRT_SECURE_NO_WARNINGS 1
#include "contact.h"

//简易菜单
void menu() {
    printf("--------------------------\n");
    printf("----1.Add--------2.Del----\n");
    printf("----3.Search-----4.Modify-\n");
    printf("----5.Show-------6.Sort---\n");
    printf("-----------0.Exit---------\n");
    printf("--------------------------\n");
}
//通讯录实现
int main() {
    int input = 0;
    //创建通讯录
    struct Contact con;
    //初始化通讯录
```

```
    InitContact(&con);
    do {
        menu();
        printf("请输入你想要的操作：");

        scanf("%d", &input);
        switch (input)
        {
        case 1:  //增加信息
            AddContact(&con);
            break;
        case 2:
            DelContact(&con);
            break;
        case 3:
            Search(&con);
            break;
        case 4:
            Modify(&con);
            break;
        case 5:
            ShowContact(&con);
            break;
        case 6:
            Sort(&con);
            break;
        case 0:
            printf("退出程序.");
            break;
        default:
            break;
        }
    } while (input);
}
```

那么这时候需要考虑头文件的构建，在项目需要多个文件时，不妨将头文件整合到一起，同时将一些全局定义的类型、变量、函数声明也放入头文件。

😊 Contact.h

```
# define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
#include <string.h>
#include <assert.h>

//使用宏定义,提高后续拓展性
#define MAX 100
#define MAX_NAME 20
#define MAX_SEX 5
#define MAX_TELE 12
#define MAX_ADDRESS 30
//表示人的信息
struct PeoInfo {
    char name[MAX_NAME];
```

```
    char sex[MAX_SEX];
    char tele[MAX_TELE];
    int age;
    char address[MAX_ADDRESS];
};

//封装通讯录
struct Contact {
    struct PeoInfo data[MAX];
    int sz;
};

//声明初始化通讯录函数
void InitContact(struct Contact* con);

//声明增加联系人函数
void AddContact(struct Contact* con);

//声明显示通讯录
void ShowContact(const struct Contact* con);

//删除通讯录
void DelContact(struct Contact* con);

//查找联系人
void Search(const struct Contact* con);

//修改联系人
void Modify(struct Contact* con);

//可以按照姓名索引、年龄等排序
void Sort(struct Contact* con);
```

显然，在头文件里声明这几个功能函数。

😊 Contact.c 函数功能

至此，我们就要构建这些个主题功能如何实现，首先我们实现将通讯录初始化，这是功能实现前的必要操作。

```
//实现初始化通讯录
void InitContact(struct Contact* con) {
    //断言这个指针非空
    assert(con);
    con->sz = 0;
    memset(con->data, 0, 100 * sizeof(struct PeoInfo));  //初始化100个数据为0
}
```

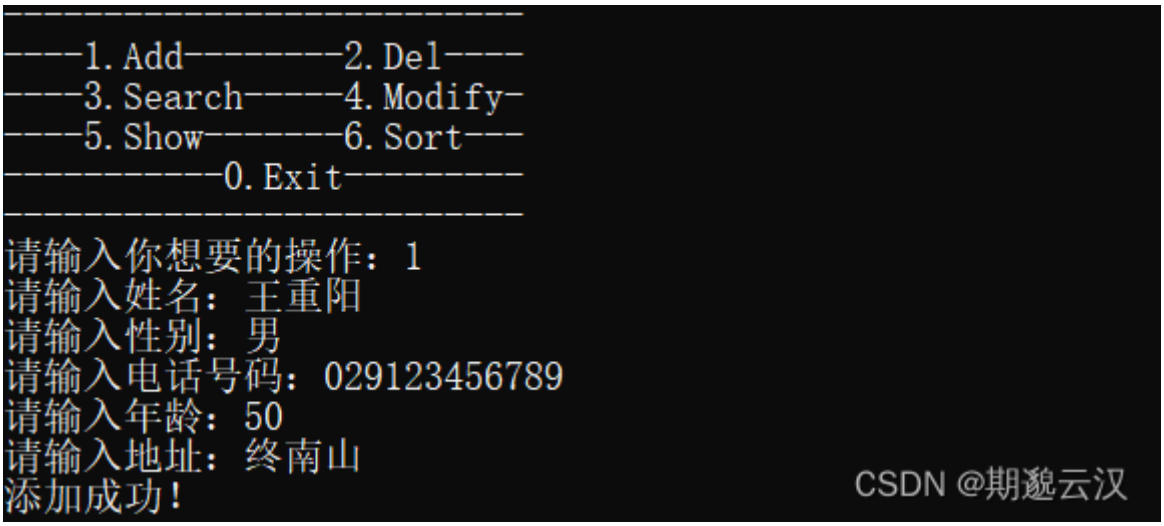
考虑Add函数，向通讯录增加联系人：

```
void AddContact(struct Contact* con) {
    if (con->sz == MAX) {
```

```
        printf("当前通讯录已满！\n");  //结束
        return;
    }
    //增添信息
    //以sz为下标，sz表示人数
    printf("请输入姓名：");
    scanf("%s", con->data[con->sz].name);
    printf("请输入性别：");
    scanf("%s", con->data[con->sz].sex);
    printf("请输入电话号码：");
    scanf("%s", con->data[con->sz].tele);
    printf("请输入年龄：");
    scanf("%d", &(con->data[con->sz].age));
    printf("请输入地址：");
    scanf("%s", &(con->data[con->sz].address));

    con->sz++;
    printf("添加成功！\n");
}
```

显然，在添加之前肯定要判断通讯录是否已经存满。



Show函数来打印通讯录：

```
//实现打印函数
void ShowContact(const struct Contact* con) {
    printf("%-10s\t%-5s\t%-12s\t%-5s\t%-20s\n\n","姓名","性别","电话号码","年龄","地址");
    for (int i = 0; i < con->sz; i++) {
        printf("%-10s\t%-5s\t%-12s\t%-5d\t%-20s\n", con->data[i].name,
            con->data[i].sex,
            con->data[i].tele,
            con->data[i].age,
            con->data[i].address);
    }
}
```

Del删除联系人：

```
static int FindName(struct Contact* con,char name[]) {
    for (int i = 0; i < con->sz; i++) {
        if (strcmp(con->data[i].name, name) == 0) {
            return i;
        }
    }
    return -1;
}
//删除指定联系人
void DelContact(struct Contact* con) {
    char name[MAX_NAME];
    printf("请输入要删除的联系人姓名：");
    scanf("%s", name);
    //查找该联系人
    int ret = FindName(con,name);
    if (ret == -1) {
        printf("该联系人不存在！\n");
    }
    else {
        for (int j = ret; j < con->sz - 1; j++) {
            con->data[j] = con->data[j + 1];
            //也可以使用memmove
        }
        con->sz--;
        printf("删除联系人成功！\n");
    }
}
```

在这里我们设计了一个静态查找姓名的函数来实现通过姓名发现联系人，来决定能否删除。当然，后续的Find查找联系人、和修改联系人也同理都要根据查找姓名来判断该联系人是否存在。

```
//查找指定联系人
void Search(const struct Contact* con) {
    char name[MAX_NAME];
    printf("请输入要查找的联系人的姓名：");
    scanf("%s", name);
    int ret = FindName(con,name);
    if (ret == -1) {
        printf("要查找的联系人不存在！\n");
    }
    else {
        printf("%-10s\t%-5s\t%-12s\t%-5s\t%-20s\n\n", "姓名", "性别", "电话号码",
"年龄", "地址");
        printf("%-10s\t%-5s\t%-12s\t%-5d\t%-20s\n", con->data[ret].name,
            con->data[ret].sex,
```

```
            con->data[ret].tele,
            con->data[ret].age,
            con->data[ret].address);
    }
}
```

```
//实现修改联系人
void Modify(struct Contact* con) {
    char name[MAX_NAME];
    printf("请输入要修改的联系人的姓名：");
    scanf("%s", name);
    int ret = FindName(con, name);
    if (ret == -1) {
        printf("要修改的联系人不存在\n");
    }
    else {
        //修改信息

        printf("请输入姓名：");
        scanf("%s", con->data[ret].name);
        printf("请输入性别：");
        scanf("%s", con->data[ret].sex);
        printf("请输入电话号码：");
        scanf("%s", con->data[ret].tele);
        printf("请输入年龄：");
        scanf("%d", &(con->data[ret].age));
        printf("请输入地址：");
        scanf("%s", &(con->data[ret].address));
        printf("修改成功\n");
    }
}
```

Sort为通讯录的联系人进行排序，这里我们可以使用C语言通过的qsort函数

```
//实现qsort的如何比较
int CmpByAge(const void* e1, const void* e2) {
    //强制类型转化
    return ((struct PeoInfo*)e1)->age - ((struct PeoInfo*)e2)->age;
}
//按照姓名来排序
int CmpByName(const void* e1, const void* e2) {
    return strcmp(((struct PeoInfo*)e1)->name, ((struct PeoInfo*)e2)->name);
}
//实现按照年龄来排序
void Sort(struct Contact* con) {
    //使用qsort函数
    //qsort (void* base, size_t num, size_t size,int (*compar)(const void*, const
void*));
    qsort(con->data, con->sz, sizeof(struct PeoInfo), CmpByName);
}
```

关于比较规则，CmpBy***可以自行设计，这里根据年龄来排序，也可以按照姓名来排序。

程序结构

整个的项目结构已经很清晰：

- main.c
- Contact.c
- Contact.h

`对整个功能核心函数进行整合，在Contact.c中实现：`

Contact.c

```
# define _CRT_SECURE_NO_WARNINGS 1
# include "contact.h"

//实现初始化通讯录
void InitContact(struct Contact* con) {
    //断言这个指针非空
    assert(con);
    con->sz = 0;
    memset(con->data, 0, 100 * sizeof(struct PeoInfo));  //初始化100个数据为0
}

//
void AddContact(struct Contact* con) {
    if (con->sz == MAX) {
        printf("当前通讯录已满！\n");  //结束
        return;
    }
    //增添信息
    //以sz为下标，sz表示人数
    printf("请输入姓名：");
    scanf("%s", con->data[con->sz].name);
    printf("请输入性别：");
    scanf("%s", con->data[con->sz].sex);
    printf("请输入电话号码：");
    scanf("%s", con->data[con->sz].tele);
    printf("请输入年龄：");
    scanf("%d", &(con->data[con->sz].age));
    printf("请输入地址：");
    scanf("%s", &(con->data[con->sz].address));

    con->sz++;
    printf("添加成功！\n");
}


//实现打印函数
void ShowContact(const struct Contact* con) {
    printf("%-10s\t%-5s\t%-12s\t%-5s\t%-20s\n\n","姓名","性别","电话号码","年龄","地
址");
    for (int i = 0; i < con->sz; i++) {
        printf("%-10s\t%-5s\t%-12s\t%-5d\t%-20s\n", con->data[i].name,
```

```
            con->data[i].sex,
            con->data[i].tele,
            con->data[i].age,
            con->data[i].address);
    }
}

static int FindName(struct Contact* con,char name[]) {
    for (int i = 0; i < con->sz; i++) {
        if (strcmp(con->data[i].name, name) == 0) {
            return i;
        }
    }
    return -1;
}
//删除指定联系人
void DelContact(struct Contact* con) {
    char name[MAX_NAME];
    printf("请输入要删除的联系人姓名：");
    scanf("%s", name);
    //查找该联系人
    int ret = FindName(con,name);
    if (ret == -1) {
        printf("该联系人不存在！\n");
    }
    else {
        for (int j = ret; j < con->sz - 1; j++) {
            con->data[j] = con->data[j + 1];
            //也可以使用memmove
        }
        con->sz--;
        printf("删除联系人成功！\n");
    }
}

//查找指定联系人
void Search(const struct Contact* con) {
    char name[MAX_NAME];
    printf("请输入要查找的联系人的姓名：");
    scanf("%s", name);
    int ret = FindName(con,name);
    if (ret == -1) {
        printf("要查找的联系人不存在！\n");
    }
    else {
        printf("%-10s\t%-5s\t%-12s\t%-5s\t%-20s\n\n", "姓名", "性别", "电话号码",
"年龄", "地址");
        printf("%-10s\t%-5s\t%-12s\t%-5d\t%-20s\n", con->data[ret].name,
            con->data[ret].sex,
            con->data[ret].tele,
            con->data[ret].age,
            con->data[ret].address);
    }
}
```

```c
//实现修改联系人
void Modify(struct Contact* con) {
    char name[MAX_NAME];
    printf("请输入要修改的联系人的姓名：");
    scanf("%s", name);
    int ret = FindName(con, name);
    if (ret == -1) {
        printf("要修改的联系人不存在\n");
    }
    else {
        //修改信息

        printf("请输入姓名：");
        scanf("%s", con->data[ret].name);
        printf("请输入性别：");
        scanf("%s", con->data[ret].sex);
        printf("请输入电话号码：");
        scanf("%s", con->data[ret].tele);
        printf("请输入年龄：");
        scanf("%d", &(con->data[ret].age));
        printf("请输入地址：");
        scanf("%s", &(con->data[ret].address));
        printf("修改成功\n");
    }
}

//实现qsort的如何比较
int CmpByAge(const void* e1, const void* e2) {
    //强制类型转化
    return ((struct PeoInfo*)e1)->age - ((struct PeoInfo*)e2)->age;
}
//按照姓名来排序
int CmpByName(const void* e1, const void* e2) {
    return strcmp(((struct PeoInfo*)e1)->name, ((struct PeoInfo*)e2)->name);
}
//实现按照年龄来排序
void Sort(struct Contact* con) {
    //使用qsort函数
    //qsort (void* base, size_t num, size_t size,int (*compar)(const void*, const
void*));
    qsort(con->data, con->sz, sizeof(struct PeoInfo), CmpByName);
}
```

后记

整个程序需要进行分块实现，分块为了让整个代码的思路更加清晰，函数独自封装，头文件整合在一起，主函数实现外部功能选择框架，这些都是需要注意。

思考结合C语言枚举类型的特性：
我们实现在input输入操作时，case进行匹配时可以更加直观地看到我们要进行的操作

```
//采用枚举来实现input的选择
enum input {
    Exit,   //0
    Add,    //1
    Del,    //2
    Search, //3
    Modify, //4
    Show,   //5
    Sort    //6
};
```

这样一来，在switch.....case语句里，我们case后面是需要对应操作名称，在input输入时就可以根据数字去匹配具体的操作。

```
        scanf( _Format: "%d", &input);
        switch (input)
        {
        case Add:  //增加信息
            AddContact(&con);
            break;
        case Del:
            DelContact(&con);
            break;
        case Search:
            SearchContact(&con);
            break;
        case Modify:
            ModifyContact(&con);
            break;
        case Show:
            ShowContact(&con);
            break;
        case Sort:
            SortContact(&con);
            break;
        case Exit:
            DestroyCon(&con);  //销毁通讯录
            printf( _Format: "退出程序! \n");
            break;
        default:
            break;
        }
```


修改后的代码如下：

main.c

```c
# define _CRT_SECURE_NO_WARNINGS 1
#include "contact.h"

//简易菜单
void menu() {
    printf("--------------------------\n");
    printf("----1.Add--------2.Del----\n");
    printf("----3.Search-----4.Modify-\n");
    printf("----5.Show-------6.Sort---\n");
    printf("-----------0.Exit---------\n");
    printf("--------------------------\n");
}
//采用枚举来实现input的选择
enum input {
    Exit,   //0
    Add,    //1
    Del,    //2
    Search, //3
    Modify, //4
    Show,   //5
    Sort    //6
};
//通讯录实现
int main() {
    int input = 0;
    //创建通讯录
    struct Contact con;
    //初始化通讯录
    InitContact(&con);
    do {
        menu();
        printf("请输入你想要的操作：");

        scanf("%d", &input);
        switch (input)
        {
        case Add:  //增加信息
            AddContact(&con);
            break;
        case Del:
            DelContact(&con);
            break;
        case Search:
            SearchContact(&con);
            break;
        case Modify:
            ModifyContact(&con);
            break;
        case Show:
            ShowContact(&con);
            break;
        case Sort:
            SortContact(&con);
            break;
        case Exit:
```

```
            DestroyCon(&con);  //销毁通讯录
            printf("退出程序！\n");
            break;
        default:
            break;
        }
    } while (input);
}
```

🌟 动态内存开辟扩容data通讯录

在C语言里，动态内存开辟的最大好处就是最大化合理利用内存空间，想象一下，如果我们一开始就给通讯录联系人的数据data开辟100个，那么实际会用到这么多的内存空间吗？又或者我们能不能通过增长式的内存空间申请来保存更多的联系人信息，那么realloc函数将会帮助我们实现动态增长的通讯录长度。

```
//封装通讯录（动态内存开辟）
struct Contact {
    struct PeoInfo *data;
    int sz;
    int capacity;  //通讯录容量
};
```

这里我们就要用结构体指针来指向联系人的内存空间块了。

考虑修改初始化通讯录这个函数，我们默认一开始通讯录可以存放3个联系人，其实DEFAULT_SIZE可以设定为想要的值，`data指针指向了malloc申请的 **3 * 每个联系人结构体字节大小** 的空间`

```
//实现初始化通讯录
void InitContact(struct Contact* con) {
    //断言这个指针非空
    assert(con);
    //con->sz = 0;
    //memset(con->data, 0, 100 * sizeof(struct PeoInfo));  //初始化100个数据为0

    con->data = (struct PeoInfo*)malloc(DEFAULT_SIZE * sizeof(struct PeoInfo));
    con->capacity = DEFAULT_SIZE;
    con->sz = 0;
}
```

考虑其实在这些功能里最受影响，最需要修改的功能就是Add增加联系人这个功能，考虑当当前通讯录满的时候就进行扩容，即`con->sz == con->capacity`

```
//检查容量
int check_capa(struct Contact * con) {

    if (con->sz == con->capacity) {

        //增加容量
        struct PeoInfo* ptr = (struct PeoInfo*)realloc(con->data, (con->capacity
+ INC_SZ)*sizeof(struct PeoInfo));
```

```
        if (ptr != NULL) {
            con->data = ptr;
            con->capacity += INC_SZ;
            printf("增容成功！\n");
            return 1;
        }
        else {
            perror("AddContact()");
            return 0;
        }
    }
    return 1;
}
```

```
//动态扩容增加联系人信息
void AddContact(struct Contact* con) {
    if (0 == check_capa(con)) {
        return;
    }
    //增添信息
    //以sz为下标，sz表示人数
    printf("请输入姓名：");
    scanf("%s", con->data[con->sz].name);
    printf("请输入性别：");
    scanf("%s", con->data[con->sz].sex);
    printf("请输入电话号码：");
    scanf("%s", con->data[con->sz].tele);
    printf("请输入年龄：");
    scanf("%d", &(con->data[con->sz].age));
    printf("请输入地址：");
    scanf("%s", &(con->data[con->sz].address));

    con->sz++;
    printf("添加成功！\n");
}
```

到这里就可以实现根据实际联系人个数动态开辟内存空间。

😊😊 完整源代码参考

`main.c`

```
# define _CRT_SECURE_NO_WARNINGS 1
#include "contact.h"

//简易菜单
void menu() {
    printf("--------------------------\n");
    printf("----1.Add--------2.Del----\n");
    printf("----3.Search-----4.Modify-\n");
    printf("----5.Show-------6.Sort---\n");
    printf("-----------0.Exit---------\n");
    printf("--------------------------\n");
```

```
}
//采用枚举来实现input的选择
enum input {
    Exit,   //0
    Add,    //1
    Del,    //2
    Search, //3
    Modify, //4
    Show,   //5
    Sort    //6
};
//通讯录实现
int main() {
    int input = 0;
    //创建通讯录
    struct Contact con;
    //初始化通讯录
    InitContact(&con);
    do {
        menu();
        printf("请输入你想要的操作：");

        scanf("%d", &input);
        switch (input)
        {
        case Add:  //增加信息
            AddContact(&con);
            break;
        case Del:
            DelContact(&con);
            break;
        case Search:
            SearchContact(&con);
            break;
        case Modify:
            ModifyContact(&con);
            break;
        case Show:
            ShowContact(&con);
            break;
        case Sort:
            SortContact(&con);
            break;
        case Exit:
            DestroyCon(&con);  //销毁通讯录
            printf("退出程序！\n");
            break;
        default:
            break;
        }
    } while (input);
    return 0;
}
```

Contact.c

```c
# define _CRT_SECURE_NO_WARNINGS 1
# include "contact.h"

//销毁通讯录
void DestroyCon(struct Contact* con) {
    free(con->data);
    con->data = NULL;
    con->capacity = 0;
    con->sz = 0;
}

//实现初始化通讯录
void InitContact(struct Contact* con) {
    //断言这个指针非空
    assert(con);
    //con->sz = 0;
    //memset(con->data, 0, 100 * sizeof(struct PeoInfo));  //初始化100个数据为0

    con->data = (struct PeoInfo*)malloc(DEFAULT_SIZE * sizeof(struct PeoInfo));
    con->capacity = DEFAULT_SIZE;
    con->sz = 0;
}

//检查容量
int check_capa(struct Contact * con) {

    if (con->sz == con->capacity) {

        //增加容量
        struct PeoInfo* ptr = (struct PeoInfo*)realloc(con->data, (con->capacity
+ INC_SZ)*sizeof(struct PeoInfo));
        if (ptr != NULL) {
            con->data = ptr;
            con->capacity += INC_SZ;
            printf("增容成功! \n");
            return 1;
        }
        else {
            perror("AddContact()");
            return 0;
        }
    }
    return 1;
}
//动态扩容增加联系人信息
void AddContact(struct Contact* con) {
    if (0 == check_capa(con)) {
        return;
    }
    //增添信息
    //以sz为下标，sz表示人数
    printf("请输入姓名：");
    scanf("%s", con->data[con->sz].name);
    printf("请输入性别：");
    scanf("%s", con->data[con->sz].sex);
```

```
    printf("请输入电话号码：");
    scanf("%s", con->data[con->sz].tele);
    printf("请输入年龄：");
    scanf("%d", &(con->data[con->sz].age));
    printf("请输入地址：");
    scanf("%s", &(con->data[con->sz].address));

    con->sz++;
    printf("添加成功！\n");
}


//实现打印函数
void ShowContact(const struct Contact* con) {
    printf("%-10s\t%-5s\t%-12s\t%-5s\t%-20s\n\n","姓名","性别","电话号码","年龄","地
址");
    for (int i = 0; i < con->sz; i++) {
        printf("%-10s\t%-5s\t%-12s\t%-5d\t%-20s\n", con->data[i].name,
            con->data[i].sex,
            con->data[i].tele,
            con->data[i].age,
            con->data[i].address);
    }
}
//实现通过联系人姓名来查找联系人是否存在
static int FindName(const struct Contact* con,char name[]) {
    for (int i = 0; i < con->sz; i++) {
        if (strcmp(con->data[i].name, name) == 0) {
            return i;
        }
    }
    return -1;
}
//删除指定联系人
void DelContact(struct Contact* con) {
    char name[MAX_NAME];
    printf("请输入要删除的联系人姓名：");
    scanf("%s", name);
    //查找该联系人
    int ret = FindName(con,name);
    if (ret == -1) {
        printf("该联系人不存在！\n");
    }
    else {
        for (int j = ret; j < con->sz - 1; j++) {
            con->data[j] = con->data[j + 1];
            //也可以使用memmove
        }
        con->sz--;
        printf("删除联系人成功！\n");
    }
}

//查找指定联系人
void SearchContact(const struct Contact* con) {
```

```
    char name[MAX_NAME];
    printf("请输入要查找的联系人的姓名：");
    scanf("%s", name);
    int ret = FindName(con,name);
    if (ret == -1) {
        printf("要查找的联系人不存在！\n");
    }
    else {
        printf("%-10s\t%-5s\t%-12s\t%-5s\t%-20s\n\n", "姓名", "性别", "电话号码",
"年龄", "地址");
        printf("%-10s\t%-5s\t%-12s\t%-5d\t%-20s\n", con->data[ret].name,
            con->data[ret].sex,
            con->data[ret].tele,
            con->data[ret].age,
            con->data[ret].address);
    }
}

//实现修改联系人
void ModifyContact(struct Contact* con) {
    char name[MAX_NAME];
    printf("请输入要修改的联系人的姓名：");
    scanf("%s", name);
    int ret = FindName(con, name);
    if (ret == -1) {
        printf("要修改的联系人不存在\n");
    }
    else {
        //修改信息

        printf("请输入姓名：");
        scanf("%s", con->data[ret].name);
        printf("请输入性别：");
        scanf("%s", con->data[ret].sex);
        printf("请输入电话号码：");
        scanf("%s", con->data[ret].tele);
        printf("请输入年龄：");
        scanf("%d", &(con->data[ret].age));
        printf("请输入地址：");
        scanf("%s", &(con->data[ret].address));
        printf("修改成功\n");
    }
}

//实现qsort的如何比较
int CmpByAge(const void* e1, const void* e2) {
    //强制类型转化
    return ((struct PeoInfo*)e1)->age - ((struct PeoInfo*)e2)->age;
}
//按照姓名来排序
int CmpByName(const void* e1, const void* e2) {
    return strcmp(((struct PeoInfo*)e1)->name, ((struct PeoInfo*)e2)->name);
}
//实现按照年龄来排序
void SortContact(struct Contact* con) {
```

```
    //使用qsort函数
    //qsort (void* base, size_t num, size_t size,int (*compar)(const void*, const void*));
    qsort(con->data, con->sz, sizeof(struct PeoInfo), CmpByName);
}
```

Contact.h

```
# define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <stdlib.h>
//使用宏定义,提高后续拓展性，将成员属性的字段长度灵活定义
#define MAX 100
#define MAX_NAME 20
#define MAX_SEX 5
#define MAX_TELE 12
#define MAX_ADDRESS 30

#define DEFAULT_SIZE 3  //初始化时默认的data的容量大小，即设置初始通讯录有3个联系人
#define INC_SZ 3  //每次扩容的大小
//表示人的信息
struct PeoInfo {
    char name[MAX_NAME];
    char sex[MAX_SEX];
    char tele[MAX_TELE];
    int age;
    char address[MAX_ADDRESS];
};

//封装通讯录（动态内存开辟）
struct Contact {
    struct PeoInfo *data;
    int sz;
    int capacity;  //通讯录容量
};


//声明初始化通讯录函数
void InitContact(struct Contact* con);

//销毁通讯录
void DestroyCon(struct Contact* con);

//声明增加联系人函数
void AddContact(struct Contact* con);

//声明显示通讯录
void ShowContact(const struct Contact* con);

//删除通讯录
void DelContact(struct Contact* con);
```

```
//查找联系人
void SearchContact(const struct Contact* con);

//修改联系人
void ModifyContact(struct Contact* con);

//可以按照姓名索引、年龄等排序
void SortContact(struct Contact* con);
```

> 关于qsort函数的更多用法可以参考· cplusplus