实　　验　　目　　的　　要　　求

实验 1：　顺序表的基本操作及应用

一、实验目的

1. 掌握线性表的顺序表示与实现

2. 实现顺序表的基本操作，会用这些基本操作解决实际问题

3. 加深对顺序表的理解，逐步培养解决实际问题的能力

二、实验内容

1、实现线性表的顺序存储定义，完成顺序表的创建、插入、删除、查找、排序等常用操作，完成两个有序线性表的合并，要求同样的数据元素只出现一次。

思路：

首先，明确需求，确定需要实现的功能，如初始化、添加、删除、查找、排序和合并顺序表。接着，设计合适的数据结构，选择数组作为顺序表的存储方式，并定义顺序表结构体。然后，将功能模块化，分别实现每个操作，如添加、删除、查找、排序和合并，确保每个函数专注于单一任务。

在排序功能中，引入枚举类型，使得升序和降序的选择更加直观。合并逻辑方面，先将第一个顺序表的元素全部添加到新表，再检查第二个表的元素是否存在，以避免重复。最后，增加错误处理机制，处理如超出最大存储限制和元素未找到等情况。

源码：已提交到github
data-structure-experiment/Experiment01/242040286_SeqList/242040286_SeqList/SeqList.c at master · Auart/data-structure-experiment

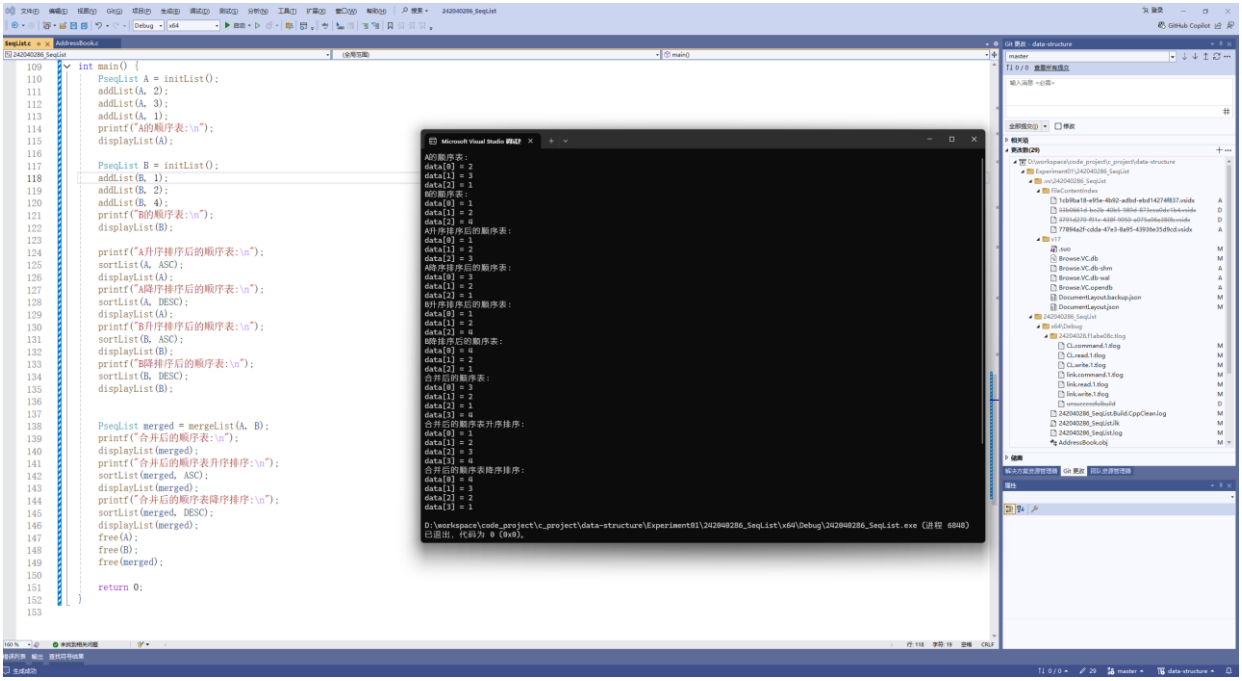
```c
#include <stdio.h>
#include <stdlib.h>
#define MAXLEN 100

typedef enum {
    ASC, DESC
} SortType;

typedef struct {
    int data[MAXLEN];
    int length;
} SeqList, * PseqList;


PseqList initList() {
    PseqList PL = malloc(sizeof(SeqList));
```

```
    if (PL) {
        PL->length = 0;
    }
    return PL;
}


void addList(PseqList PL, int item) {
    if (PL->length < MAXLEN) {
        PL->data[PL->length++] = item;
    }
    else {
        printf("顺序表已达到最大存储范围\n");
    }
}


int deleteList(PseqList PL, int item) {
    for (int i = 0; i < PL->length; i++) {
        if (PL->data[i] == item) {
            // 找到元素，进行删除
            for (int j = i; j < PL->length - 1; j++) {
                PL->data[j] = PL->data[j + 1];
            }
            PL->length--;
            return 1;
        }
    }
    printf("元素未找到\n");
    return -1;
}

int findList(PseqList PL, int element) {
    if (PL == NULL || PL->length == 0) {
        return -1;
    }

    for (int i = 0; i < PL->length; i++) {
        if (PL->data[i] == element) {
            return i;
        }
    }
    return -1;
}
```

```c
// 排序
void sortList(PseqList PL, SortType type) {
    if (PL == NULL || PL->length <= 1) {
        return;
    }

    for (int i = 0; i < PL->length - 1; i++) {
        for (int j = 0; j < PL->length - i - 1; j++) {
            if ((type == ASC && PL->data[j] > PL->data[j + 1]) ||
                (type == DESC && PL->data[j] < PL->data[j + 1])) {
                int temp = PL->data[j];
                PL->data[j] = PL->data[j + 1];
                PL->data[j + 1] = temp;
            }
        }
    }
}

// 合并
PseqList mergeList(PseqList P1, PseqList P2) {
    PseqList mergedList = initList();
    if (!mergedList) {
        return NULL;
    }

    for (int i = 0; i < P1->length; i++) {
        addList(mergedList, P1->data[i]);
    }

    for (int i = 0; i < P2->length; i++) {
        if (findList(mergedList, P2->data[i]) == -1) {
            addList(mergedList, P2->data[i]);
        }
    }

    return mergedList;
}

void displayList(PseqList PL) {
    for (int i = 0; i < PL->length; i++) {
        printf("data[%d] = %d\n", i, PL->data[i]);
    }
}
```

```c
int main() {
    PseqList A = initList();
    addList(A, 2);
    addList(A, 3);
    addList(A, 1);
    printf("A的顺序表:\n");
    displayList(A);

    PseqList B = initList();
    addList(B, 1);
    addList(B, 2);
    addList(B, 4);
    printf("B的顺序表:\n");
    displayList(B);

    printf("A升序排序后的顺序表:\n");
    sortList(A, ASC);
    displayList(A);
    printf("A降序排序后的顺序表:\n");
    sortList(A, DESC);
    displayList(A);
    printf("B升序排序后的顺序表:\n");
    sortList(B, ASC);
    displayList(B);
    printf("B降排序后的顺序表:\n");
    sortList(B, DESC);
    displayList(B);


    PseqList merged = mergeList(A, B);
    printf("合并后的顺序表:\n");
    displayList(merged);
    printf("合并后的顺序表升序排序:\n");
    sortList(merged, ASC);
    displayList(merged);
    printf("合并后的顺序表降序排序:\n");
    sortList(merged, DESC);
    displayList(merged);
    free(A);
    free(B);
    free(merged);

    return 0;
}
```

运行结果：



结论（总结）：

这段代码主要实现了一个顺序表的基本功能，特别是在排序和合并方面非常实用。在排序的部分，代码使用了一个叫做枚举的方式，让我们可以选择升序或降序来排列数字，这样就能直观地看到我们想要的顺序。通过简单的比较和交换，顺序表的内容就被整理得井井有条。

合并的部分则是将两个顺序表合成一个新的列表。它首先把第一个表的所有数字都放进新表，然后再把第二个表的数字逐个检查，如果新表中没有，就添加进去。这样合并后的顺序表不仅保留了所有的元素，还确保每个数字都是独一无二的，整合得非常好。

2、利用1中顺序表的基本操作参照教材2.8节完成两个多项式的加法运算或手机通讯录的设计与实现。

思路：

首选，确定系统的基本功能，包括建立通讯录、添加联系人、删除联系人、查找联系人和显示联系人。

然后，用户界面通过封装模块化呈现，通过函数数组指针存储相应功能函数,并通过输入序号执行相应的功能。整个设计注重模块化，便于维护和扩展.

源码： 已提交到github
data-structure-experiment/Experiment01/242040286_SeqList/242040286_SeqList/AddressBook.c at master · Auart/data-structure-experiment

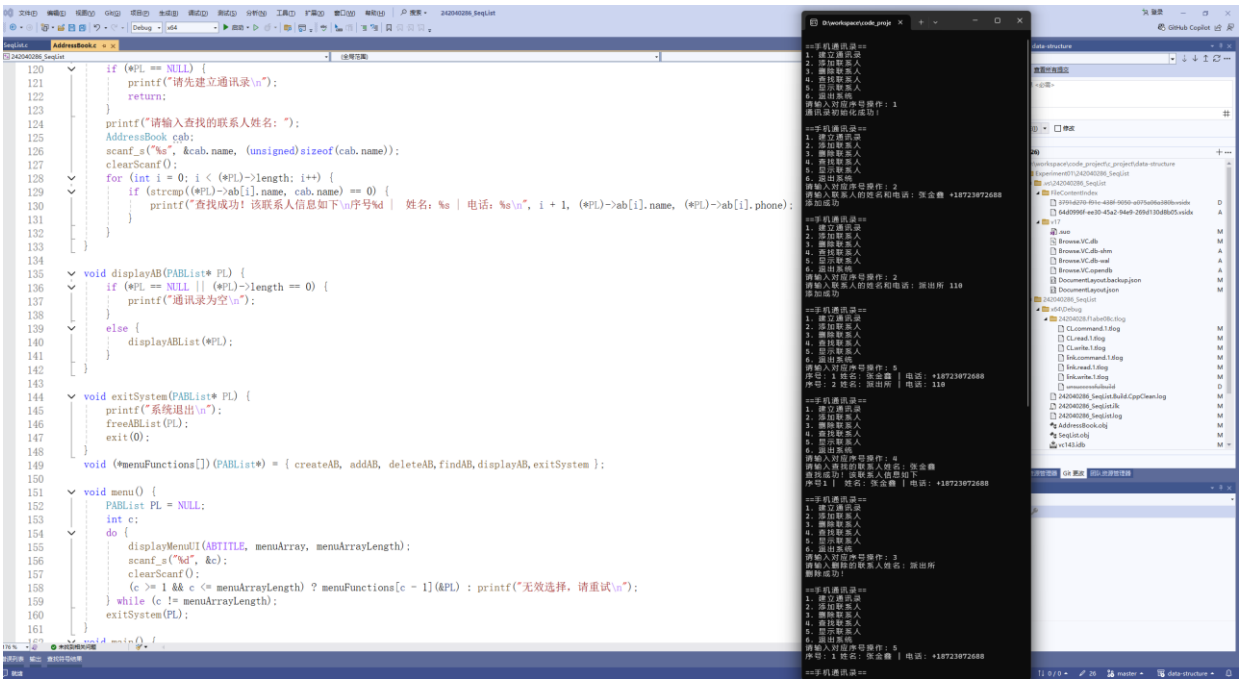
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXLEN 100
#define ABTITLE "\n==手机通讯录=="
typedef struct {
    char name[20];
    char phone[15];
} AddressBook;
typedef struct {
    AddressBook ab[MAXLEN];
    int length;
} ABList, * PABList;
void clearScanf();
PABList initABList();
void displayMenuUI(const char* title, char** menuArr, int arrLength);
void displayABList(PABList PL);
void freeABList(PABList* PL);
void createAB(PABList* PL);
void addAB(PABList* PL);
void deleteAB(PABList* PL);
void findAB(PABList* PL);
void displayAB(PABList* PL);
void exitSystem(PABList* PL);
void menu();

char* menuArray[] = {
    "建立通讯录",
    "添加联系人",
    "删除联系人",
    "查找联系人",
    "显示联系人",
    "退出系统"
};

int menuArrayLength = sizeof(menuArray) / sizeof(menuArray[0]);

// 清空输入缓冲区
void clearScanf() {
```

```
    while (getchar() != '\n');
}


PABList initABList() {
    PABList PL = malloc(sizeof(ABList));
    if (PL) {
        PL->length = 0;
        printf("通讯录初始化成功!\n");
    }
    return PL;
}


void displayABList(PABList  PL) {
    for (int i = 0; i < PL->length; i++) {
        printf("序号：%d 姓名：%s | 电话：%s\n", i + 1, PL->ab[i].name, PL->ab[i].phone);
    }
}


void freeABList(PABList* PL) {
    if (PL && *PL) {
        free(*PL);
        *PL = NULL;
    }
}

void displayMenuUI(const char* title, char** menuArr, int arrLength) {
    printf("%s\n", title);
    for (int i = 0; i < arrLength; i++) {
        printf("%d. %s\n", i + 1, menuArr[i]);
    }
    printf("请输入对应序号操作: ");
}


void createAB(PABList* PL) {
    if (*PL == NULL) {
        *PL = initABList();
    }
    else {
        printf("通讯录已建立\n");
    }
```

```
}
void addAB(PABList* PL) {
    if (*PL == NULL) {
        printf("请先建立通讯录\n");
        return;
    }
    AddressBook ab = { 0 };
    printf("请输入联系人的姓名和电话: ");
    scanf_s("%s %s", ab.name, (unsigned)sizeof(ab.name), ab.phone, (unsigned)sizeof(ab.phone));
    clearScanf();
    if ((*PL)->length < MAXLEN) {
        (*PL)->ab[(*PL)->length++] = ab;
        printf("添加成功\n");
    }
    else {
        printf("通讯录已满\n");
    }
}
void deleteAB(PABList* PL) {
    if (*PL == NULL) {
        printf("请先建立通讯录\n");
        return;
    }
    AddressBook ab = { 0 };
    printf("请输入删除的联系人姓名: ");
    scanf_s("%s", ab.name, (unsigned)sizeof(ab.name));
    clearScanf();
    for (int i = 0; i < (*PL)->length; i++) {
        if (strcmp((*PL)->ab[i].name, ab.name) == 0) {
            (*PL)->ab[(*PL)->length--] = ab;
            printf("删除成功! \n");
        }

    }
}
void findAB(PABList* PL) {
    if (*PL == NULL) {
        printf("请先建立通讯录\n");
        return;
    }
    printf("请输入查找的联系人姓名: ");
    AddressBook cab;
    scanf_s("%s", &cab.name, (unsigned)sizeof(cab.name));
    clearScanf();
```

```
    for (int i = 0; i < (*PL)->length; i++) {
        if (strcmp((*PL)->ab[i].name, cab.name) == 0) {
            printf("查找成功！该联系人信息如下\n序号%d |  姓名：%s | 电话：%s\n", i + 1,
(*PL)->ab[i].name, (*PL)->ab[i].phone);
        }
    }
}

void displayAB(PABList* PL) {
    if (*PL == NULL || (*PL)->length == 0) {
        printf("通讯录为空\n");
    }
    else {
        displayABList(*PL);
    }
}

void exitSystem(PABList* PL) {
    printf("系统退出\n");
    freeABList(PL);
    exit(0);
}
void (*menuFunctions[])(PABList*) = { createAB, addAB, deleteAB,findAB,displayAB,exitSystem };

void menu() {
    PABList PL = NULL;
    int c;
    do {
        displayMenuUI(ABTITLE, menuArray, menuArrayLength);
        scanf_s("%d", &c);
        clearScanf();
        (c >= 1 && c <= menuArrayLength) ? menuFunctions[c - 1](&PL) : printf("无效选择，请重试
\n");
    } while (c != menuArrayLength);
    exitSystem(PL);
}
void main() {
    menu();
}
```

运行结果：



结论（总结）：

手机通讯录的设计与实现，使用函数数组指针的设计提高了代码的灵活性和可维护性。通过将各个功能函数存储在一个数组中，用户只需输入对应的序号即可快速调用所需功能。这种方法简化了菜单管理，使得代码结构更清晰，易于扩展和修改。

此外，函数数组指针允许动态调用不同功能，减少了冗余代码的编写，提升了程序的效率和可读性。这种模块化的设计不仅提高了用户体验，也为未来的功能拓展提供了便利。因此，使用函数数组指针是实现灵活和高效程序的重要策略。

三、实验总结（收获）

通过这次实验，我对顺序表的基本操作有了更深入的理解，并掌握了如何使用顺序存储结构来实现线性表的常用功能。这些操作包括创建、插入、删除、查找、排序和合并等。

1. 顺序表的基本操作：我学会了如何实现顺序表，并完成了对顺序表的增删改查操作。

2. 排序与合并功能：在排序部分，我使用了枚举类型来选择升序或降序，这使得排序逻辑更加清晰。同时，合并两个有序线性表的实现让我意识到如何有效地处理重复数据，确保合并后的数据结构保持唯一性。

3. 手机通讯录系统的设计：在实现通讯录的过程中，采用函数数组指针的设计提高了代码的灵活性和可维护性。通过将各个功能模块化，用户可以方便地通过输入序号来调用相应的功能，从而提升了用户体验。

4. 模块化设计的重要性：这次实验强调了模块化设计在程序开发中的重要性。通过将不同的功能拆分为独立的函数，代码的可读性和可维护性得到了极大提升。同时，减少了冗余代码，增强了程序的效率。

总之，这次实验不仅提高了我的编程能力，也让我更深入地理解了数据结构在实际应用中的重要性。我期待在未来的学习中能将这些知识应用到更复杂的项目中去。