第25讲：顺序表专题

目录

正文开始

课前预备

1. 课程目标

C语言语法基础到数据结构与算法，前面已经掌握并具备了扎实的C语言基础，为什么要学习数据结构课程？——**通讯录项目**

2. 需要的储备知识

简单了解，通讯录具备增加、删除、修改、查找联系人等操作。要想实现通讯录项目有两个技术关键：

1）C语言语法基础

2）数据结构 之 顺序表/链表

3. 数据结构相关概念

> 1、什么是数据结构

数据结构是由“数据”和“结构”两词组合而来。

什么是数据？常见的数值1、2、3、4.....、教务系统里保存的用户信息（姓名、性别、年龄、学历等等）、网页里肉眼可以看到的信息（文字、图片、视频等等），这些都是数据

什么是结构？

当我们想要使用大量使用同一类型的数据时，通过手动定义大量的独立的变量对于程序来说，可读性非常差，我们可以借助数组这样的数据结构将大量的数据组织在一起，结构也可以理解为组织数据的方式。

想要找到草原上名叫“咩咩”的羊很难，但是从羊圈里找到1号羊就很简单，羊圈这样的结构有效将羊群组织起来。

概念：**数据结构是计算机存储、组织数据的方式。**数据结构是指相互之间存在一种或多种特定关系的数据元素的集合。数据结构反映数据的内部构成，即数据由那部分构成，以什么方式构成，以及数

据元素之间呈现的结构。

总结：

1）能够存储数据（如顺序表、链表等结构）

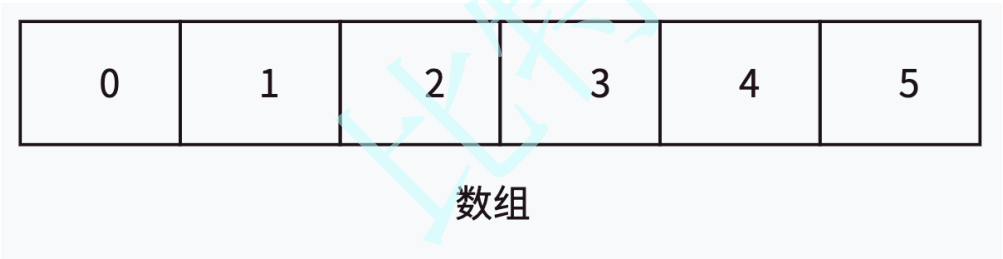
2）存储的数据能够方便查找

2、为什么需要数据结构？

如图中所示，不借助排队的方式来管理客户，会导致客户就餐感受差、等餐时间长、餐厅营业混乱等情况。同理，程序中如果不对数据进行管理，可能会导致数据丢失、操作数据困难、野指针等情况。

通过数据结构，能够有效将数据组织和管理在一起。按照我们的方式任意对数据进行增删改查等操作。

最基础的数据结构：数组。



【思考】有了数组，为什么还要学习其他的数据结构？

假定数组有10个空间，已经使用了5个，向数组中插入数据步骤：

求数组的长度，求数组的有效数据个数，向下标为数据有效个数的位置插入数据（注意：这里是否要判断数组是否满了，满了还能继续插入吗）.....

假设数据量非常庞大，频繁的获取数组有效数据个数会影响程序执行效率。

结论：最基础的数据结构能够提供的操作已经不能完全满足复杂算法实现。

顺序表

1、顺序表的概念及结构

1.1 线性表

> 线性表（*linear list*）是n个具有相同特性的数据元素的有限序列。 线性表是一种在实际中广泛使用的数据结构，常见的线性表：顺序表、链表、栈、队列、字符串...
>
> 线性表在逻辑上是线性结构，也就说是连续的一条直线。但是在物理结构上并不一定是连续的，线性表在物理上存储时，通常以数组和链式结构的形式存储。
>
> 案例：蔬菜分为绿叶类、瓜类、菌菇类。线性表指的是具有部分相同特性的一类数据结构的集合
>
> 如何理解逻辑结构和物理结构？

2、顺序表分类

- **顺序表和数组的区别**
  - 顺序表的底层结构是数组，对数组的封装，实现了常用的增删改查等接口
- **顺序表分类**
  - **静态顺序表**

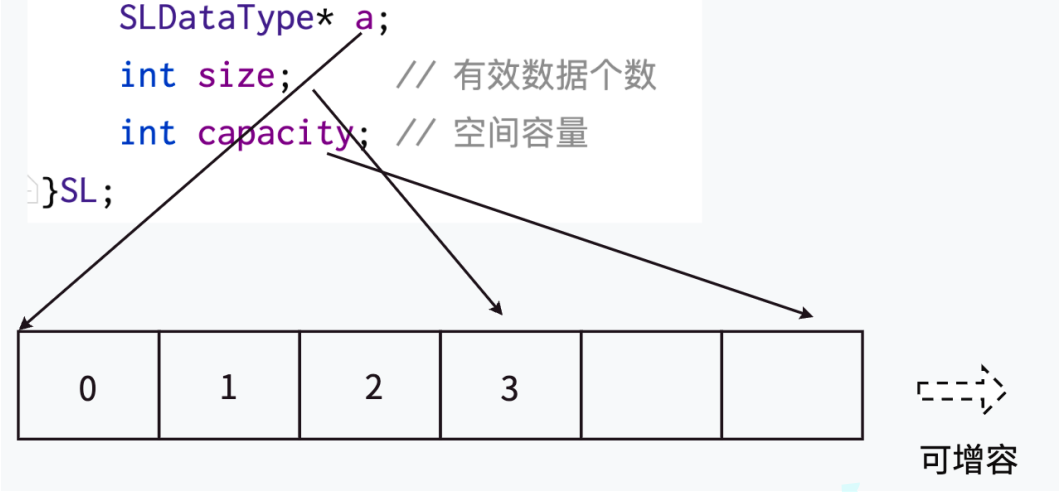
    概念：使用定长数组存储元素

```
//静态顺序表
typedef int SLDataType;
#define N 7
typedef struct SeqList{
    SLDataType a[N];    //定长数组
    int size;           //有效数据个数
}SL;
```

    静态顺序表缺陷：空间给少了不够用，给多了造成空间浪费

  - **动态顺序表**

```
// 动态顺序表 -- 按需申请
typedef struct SeqList
{
    SLDataType* a;
    int size;     // 有效数据个数
    int capacity; // 空间容量
}SL;
```



3、动态顺序表的实现

```c
#define INIT_CAPACITY 4
typedef int SLDataType;
// 动态顺序表 -- 按需申请
typedef struct SeqList
{
    SLDataType* a;
    int size;     // 有效数据个数
    int capacity; // 空间容量
}SL;

//初始化和销毁
void SLInit(SL* ps);
void SLDestroy(SL* ps);
void SLPrint(SL* ps);
//扩容
void SLCheckCapacity(SL* ps);

//头部插入删除 / 尾部插入删除
void SLPushBack(SL* ps, SLDataType x);
void SLPopBack(SL* ps);
void SLPushFront(SL* ps, SLDataType x);
void SLPopFront(SL* ps);

//指定位置之前插入/删除数据
void SLInsert(SL* ps, int pos, SLDataType x);
```

```
26 void SLErase(SL* ps, int pos);
27 int SLFind(SL* ps, SLDataType x);
```

完