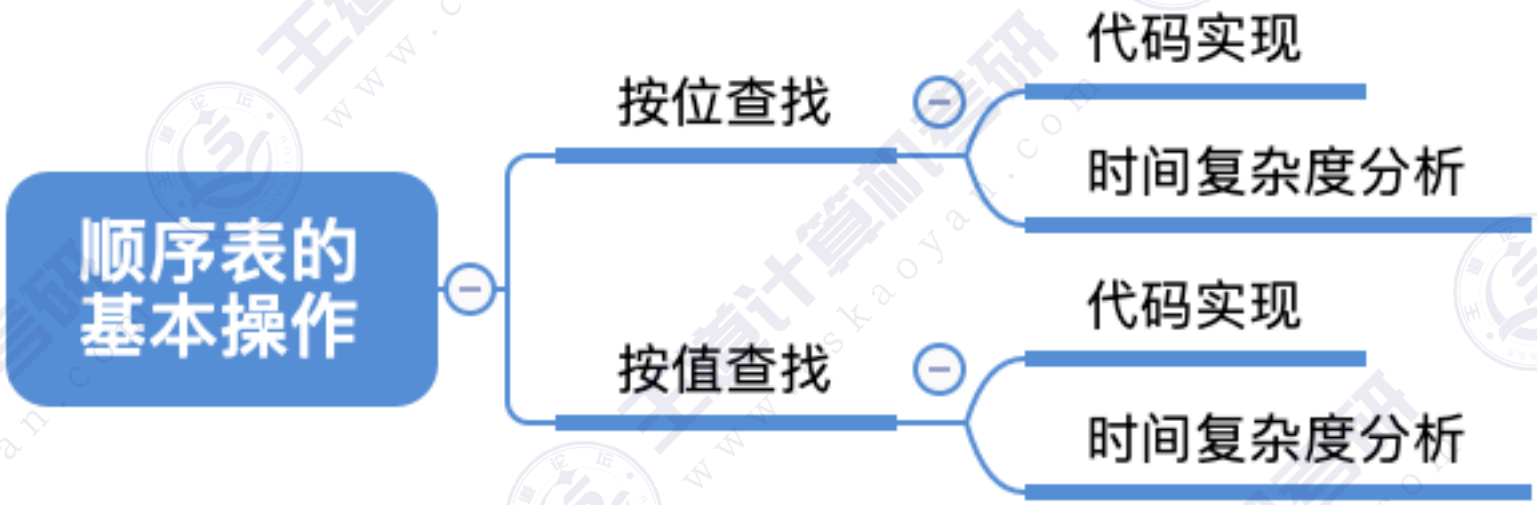
本节内容

顺序表

查找

知识总览



顺序表的按位查找

GetElem(L,i)：按位查找操作。获取表L中第i个位置的元素的值。

```
#define MaxSize 10                  //定义最大长度
typedef struct{
    ElemType data[MaxSize];         //用静态的“数组”存放数据元素
    int length;                     //顺序表的当前长度
}SqList;                            //顺序表的类型定义（静态分配方式）
```

静态分配

```
ElemType GetElem(SqList L, int i){
    return L.data[i-1];
}
```

顺序表的按位查找

GetElem(L,i)：按位查找操作。获取表L中第i个位置的元素的值。

```
#define InitSize 10          //顺序表的初始长度
typedef struct{
    ElemType *data;          //指示动态分配数组的指针
    int MaxSize;             //顺序表的最大容量
    int length;              //顺序表的当前长度
} SeqList;                   //顺序表的类型定义（动态分配方式）

ElemType GetElem(SeqList L, int i){
    return L.data[i-1];
}
```

动态分配

和访问普通数组的方法一样

ElemType *data

如果一个 ElemType 占 6B，即 sizeof(ElemType)==6
指针 data 指向的地址为 2000

顺序表的按位查找

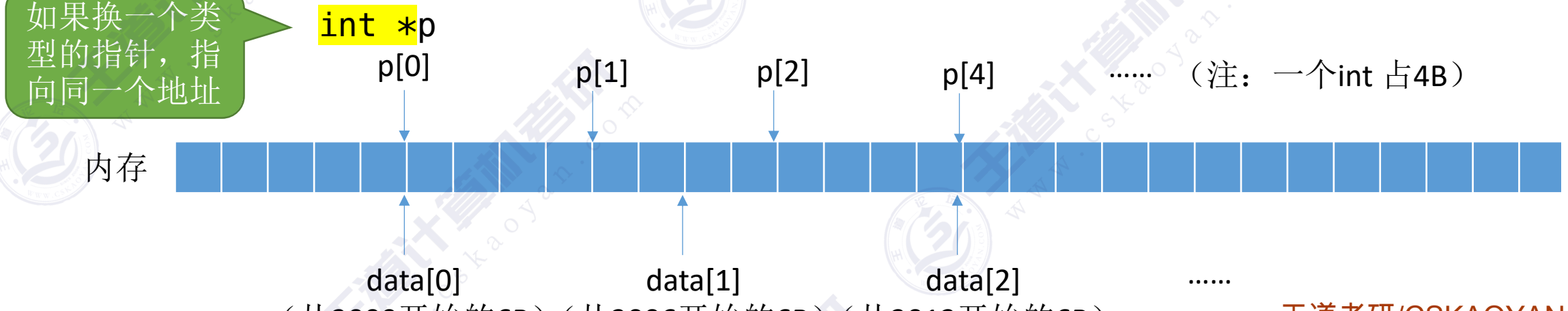
GetElem(L,i)：按位查找操作。获取表L中第i个位置的元素的值。

```
#define InitSize 10          //顺序表的初始长度
typedef struct{
    ElemType *data;          //指示动态分配数组的指针
    int MaxSize;             //顺序表的最大容量
    int length;              //顺序表的当前长度
} SeqList;                   //顺序表的类型定义（动态分配方式）

ElemType GetElem(SeqList L, int i){
    return L.data[i-1];
```

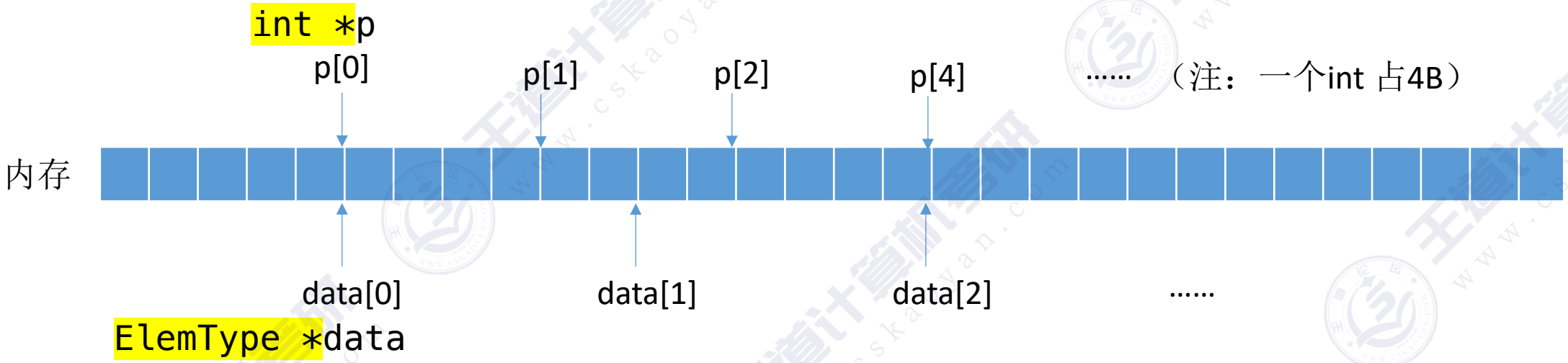
动态分配

和访问普通数组的方法一样

如果换一个类型的指针，指向同一个地址



顺序表的按位查找

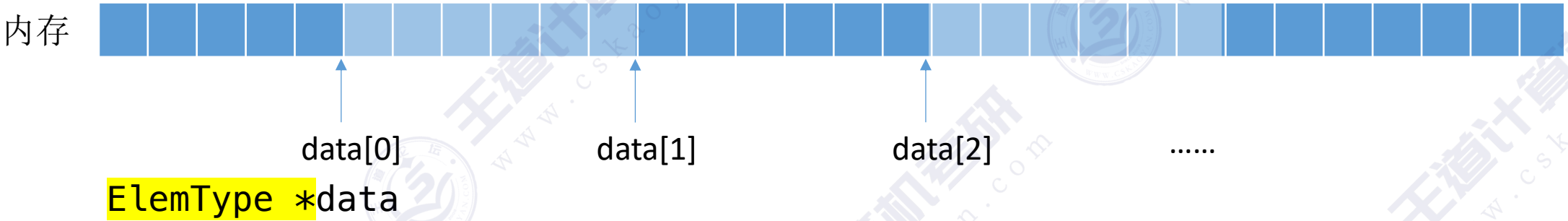



```
#define InitSize 10 //默认的最大长度
typedef struct{
    int *data;          //指示动态分配数组的指针
    int MaxSize;        //顺序表的最大容量
    int length;         //顺序表的当前长度
}SeqList;
```

```
void InitList(SeqList &L){
    //用 malloc 函数申请一片连续的存储空间
    L.data=(int *)malloc(InitSize*sizeof(int));
    L.length=0;
    L.MaxSize=Ini
}
```

再次理解，为何malloc函数返回的存储空间起始地址要转换为与数据元素的数据类型相对应的指针

按位查找的时间复杂度




```
ElemType GetElem(SeqList L, int i){
    return L.data[i-1];
}
```

时间复杂度：O(1)

由于顺序表的各个数据元素在内存中连续存放，因此可以根据起始地址和数据元素大小立即找到第 i 个元素——“随机存取”特性

顺序表的按值查找

LocateElem(L,e)：按值查找操作。在表L中查找具有给定关键字值的元素。

```
#define InitSize 10          //顺序表的初始长度
typedef struct{
    ElemType *data;          //指示动态分配数组的指针
    int MaxSize;             //顺序表的最大容量
    int length;              //顺序表的当前长度
} SeqList;                   //顺序表的类型定义（动态分配方式）


//在顺序表L中查找第一个元素值等于e的元素，并返回其位序
int LocateElem(SeqList L,ElemType e){
    for(int i=0;i<L.length;i++)
        if(L.data[i]==e)
            return  i+1;     //数组下标为i的元素值等于e，返回其位序i+1
    return  0;               //退出循环，说明查找失败
}
```

顺序表的按值查找

LocateElem(L,e)：按值查找操作。在表L中查找具有给定关键字值的元素。

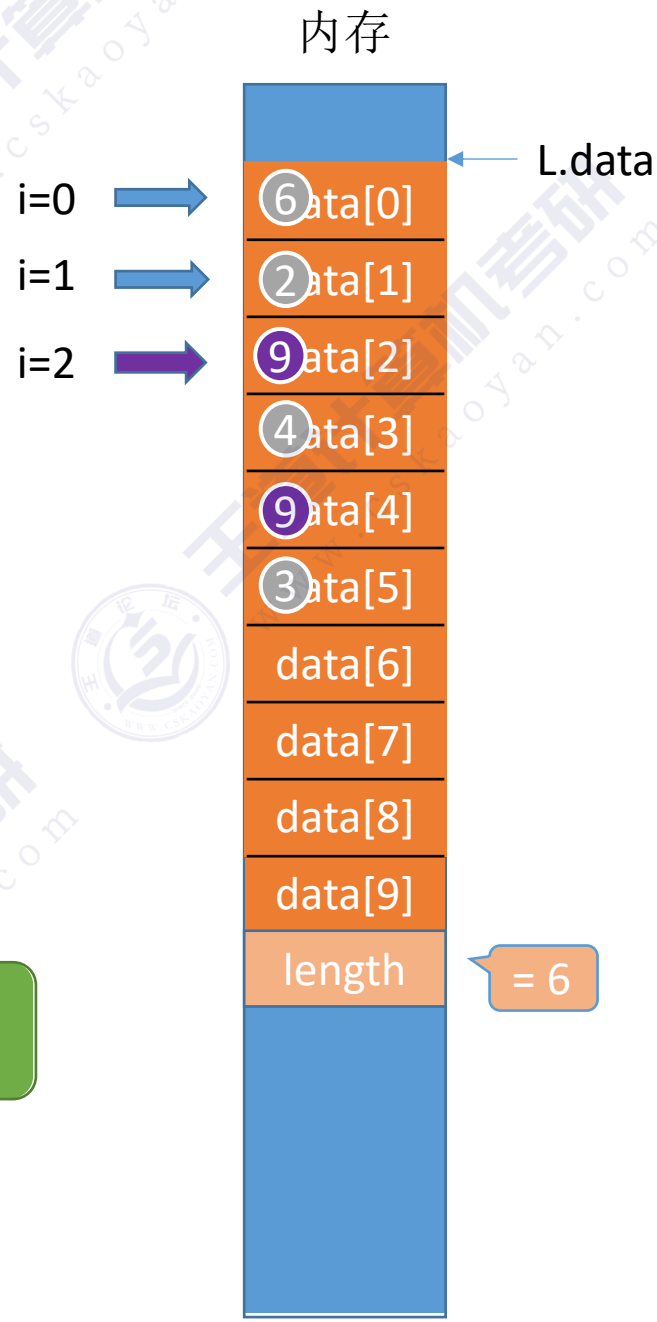
```
typedef struct{
    int *data;          //指示动态分配数组的指针
    int MaxSize;        //顺序表的最大容量
    int length;         //顺序表的当前长度
}SeqList;

//在顺序表L中查找第一个元素值等于e的元素，并返回其位序
int LocateElem(SeqList L,int e){
    for(int i=0;i<L.length;i++)
        if(L.data[i]==e)
            return  i+1;
    return  0;
}
```

i+1=3

基本数据类型：int、char、double、float 等可以直接用运算符“==”比较

结构类型的数据元素也这样吗？

调用：LocateElem(L, 9);



结构类型的比较

```c
typedef struct {
    int num;
    int people;
} Customer;

void test () {
    Customer a;
    a.num = 1;
    a.people = 1;
    Customer b;
    b.num = 1;
    b.people = 1;
    if (a == b) {
        printf("相等");
    }else {
        printf("不相等");
    }
}
```

不能

Invalid operands to binary expression ('Customer' and 'Customer')

需要依次对比各个分量来判断两个结构体是否相等

```c
bool isCustomerEqual (Customer a, Customer b){
    if (a.num == b.num && a.people == b.people)
        return true;
    else
        return false;
}
```

更好的办法：定义一个函数，童叟无欺，用过都说好

注意：C语言中，结构体的比较不能直接用 “==”

```c
if (a.num == b.num && a.people == b.people) {
    printf("相等");
}else {
    printf("不相等");
}
```

顺序表的按值查找

LocateElem(L,e)：按值查找操作。在表L中查找具有给定关键字值的元素。

```
//在顺序表L中查找第一个元素值等于e的元素，并返回其位序
int LocateElem(SeqList L,ElemType e){
    for(int i=0;i<L.length;i++)
        if(L.data[i]==e)
            return  i+1;   //数组下标为i的元素值等于e，返回其位序i+1
    return  0;             //退出循环，说明查找失败
}
```

Tips：

《数据结构》考研初试中，手写代码可以直接用“==”，无论 ElemType 是基本数据类型还是结构类型

手写代码主要考察学生是否能理解算法思想，不会严格要求代码完全可运行

有的学校考《C语言程序设计》，那么...也许就要语法严格一些

按值查找的时间复杂度

```
//在顺序表L中查找第一个元素值等于e的元素，并返回其位序
int LocateElem(SeqList L,ElemType e){
    for(int i=0;i<L.length;i++)
        if(L.data[i]==e)
            return  i+1;    //返回其位序i+1
    return  0;
}
```

关注最深层循环语句的执行次数与问题规模 n 的关系

问题规模 n = L.length （表长）

最好情况：目标元素在表头

循环1次；最好时间复杂度 = O(1)

最坏情况：目标元素在表尾

循环 n 次；最坏时间复杂度 = O(n);

平均情况：假设目标元素出现在任何一个位置的概率相同，都是 $\frac{1}{n}$

目标元素在第1位，循环1次；在第2位，循环2次；……；在第 n 位，循环 n 次

平均循环次数 $= 1\cdot\frac{1}{n} + 2\cdot\frac{1}{n} + 3\cdot\frac{1}{n} + \ldots\ldots + n\cdot\frac{1}{n} = \frac{n(n+1)}{2}\frac{1}{n} = \frac{n+1}{2}$ ⇒ 平均时间复杂度 = O(n)

知识回顾与重要考点