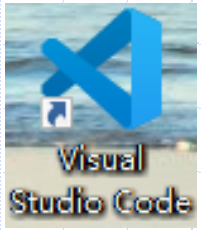
在开始之前：

- 实验课课程内容安排
  - 题目讲解+分析
  - 代码讲解
- 关于编译环境：



轻便简洁，
适合小代码编程

注意.cpp
文件尽量
不要中文
命名！

工程性
功能强大
界面美观
适合大项目开发

程序要求

1.注意异常处理的使用，比如删除空链表时需要抛出异常；

2.注意内存的动态申请和释放，是否存在内存泄漏

3.优化程序的时间性能；

4.保持良好的编程风格：

- ➢代码要简洁；
- ➢代码段与段之间要有空行和缩近；
- ➢标识符名称应该与其代表的意义一致；
- ➢函数名之前应该添加注释说明该函数的功能；
- ➢关键代码应说明其功能。

数据结构与算法

实验一　线性表

实验目的

- 通过了解下面题目并进行相应的实现，掌握如下内容：

➢熟悉C++语言的基本编程方法，掌握集成编译环境的调试方法；

➢学习指针、模板类、异常处理的使用；

➢掌握线性表的操作的实现方法；

➢学习使用线性表解决实际问题的能力。

题目一：线性表的实现

- 根据线性表的抽象数据类型的定义，选择下面任一种链式结构实现线性表，并完成线性表的基本功能。

线性表存储结构（五选一）：

1、带头结点的单链表

2、不带头结点的单链表

3、循环链表

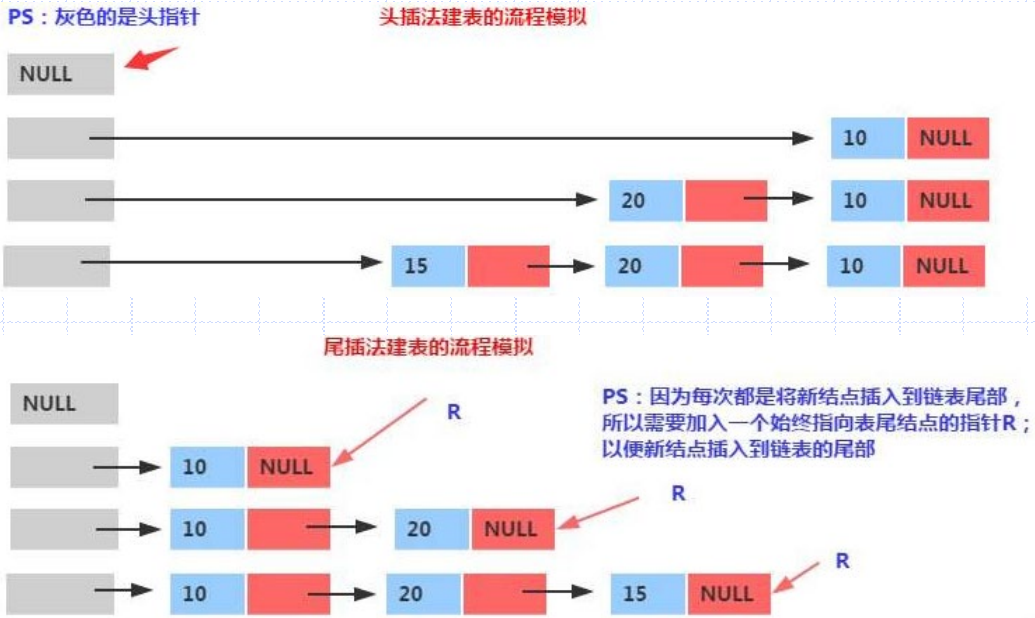
4、双链表

5、静态链表

题目一：线性表的实现

要求：实现线性表的基本功能

- **构造：使用头插法、尾插法两种方法；**
- **插入：要求建立的链表按照关键字从小到大有序（静态链表不要求该项）**



题目一：线性表的实现

要求：实现线性表的基本功能

- **删除；**
- **查找；**
- **获取链表长度；**
- **销毁；**
- **其他：可自行定义；**
- **编写测试main()函数测试线性表的正确性。**

题目二：有序链表合并

- 设有两条有序链表（即data域元素的关键字由前往后不断增大），试设计算法，将这两条链表合并为一条新的有序链表，原链表不变。两条链表中data域**关键字相同的元素只选取一个**存储到新的有序链表中，不同的元素都存储到新的有序链表中。

要求：

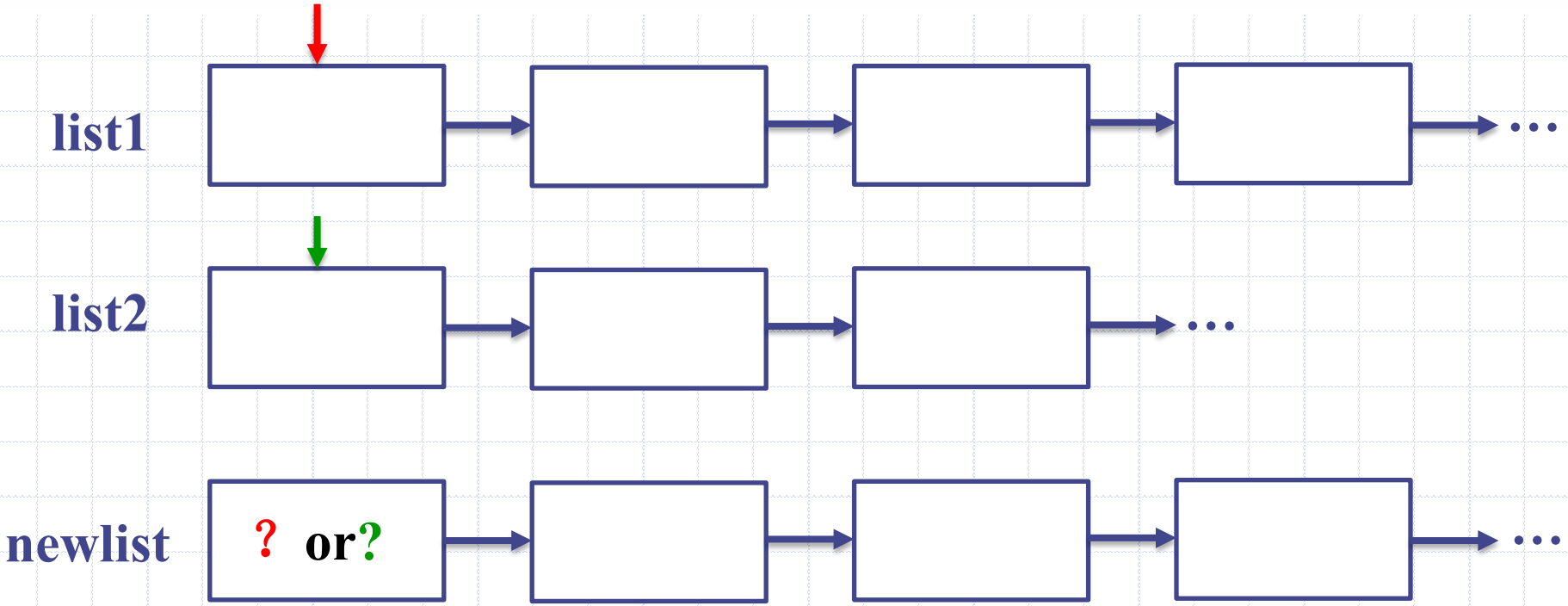
➢直接编写链表的**友元函数**完成该功能；

➢链表的data域可存储用户自定义类对象；

➢编写测试main()函数测试线性表的正确性。

题目二：有序链表合并

顺序从大到小/从小到大均可：



考虑如下情况：（假如按照从小到大）

- list1.data >(<) list2.data, new.data = ?
- list1.length != list2.length, 如何合并?

题目二：有序链表合并

考虑如下情况：

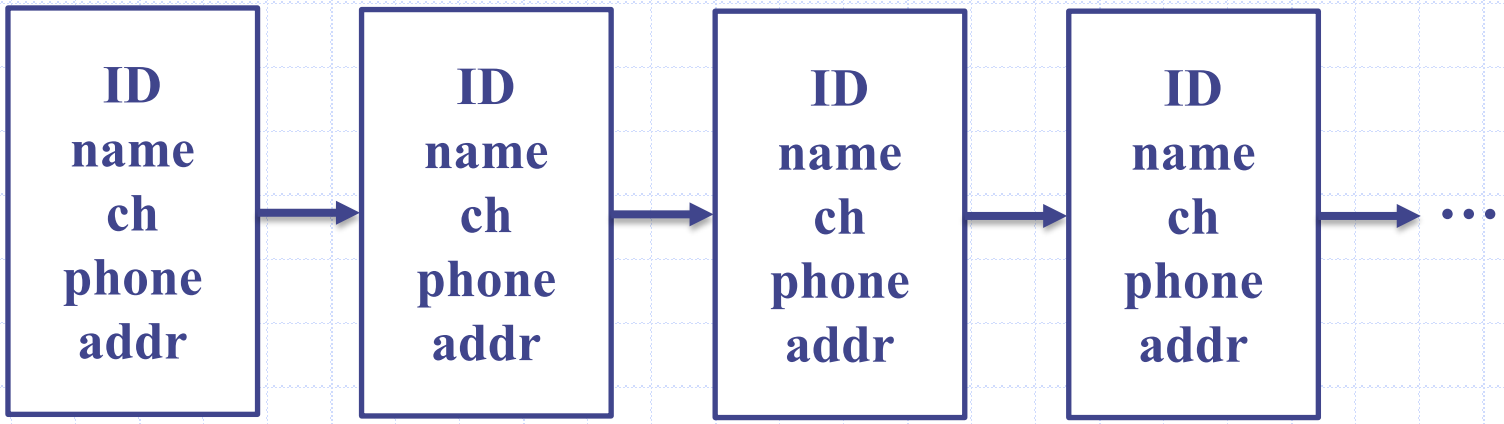
- list1.data > list2.data, new.data = ?
- list1.data < list2.data, new.data = ?
- 处理list1.next = null
- 处理list2.next = null

题目三：通讯录管理

- 利用线性表实现一个通讯录管理，通讯录的数据格式如下：
- struct DataType{

```
    int ID;       //编号
    char name[10];    //姓名
    char ch;    //性别
    char phone[13];    //电话
    char addr[31];    //地址
}
```

题目三：通讯录管理


ID
name
ch
phone
addr
ID
name
ch
phone
addr
ID
name
ch
phone
addr
ID
name
ch
phone
addr
…

题目四：一元多项式

- 利用线性表实现一个一元多项式Polynomial；
- $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \ldots + a_nx^n$
- Polynomial的结构特点如下：

```
struct term{
        float coef;    //系数
        int expn;    //指数
}
```

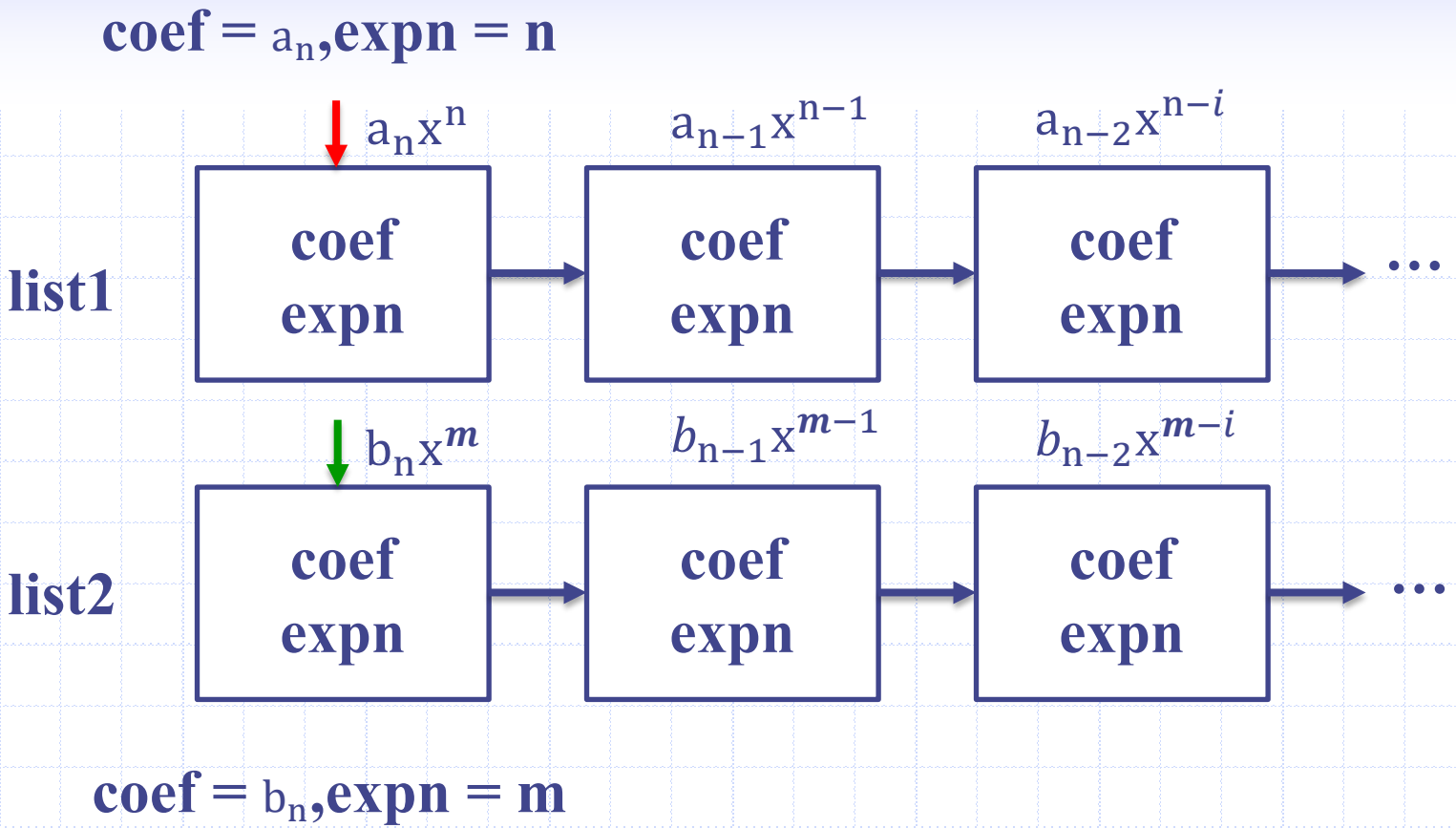
- 可以使用链表实现，也可以使用顺序表实现。

题目四：一元多项式

要求：

- 能够实现一元多项式的输入和输出；
- 能够进行一元多项式相加；
- 能够进行一元多项式相减；
- 能够计算一元多项式在x处的值；
- 能够计算一元多项式的导数；
- 能够进行一元多项式相乘；
- 编写测试main()函数测试线性表的正确性。

题目四：一元多项式



- 考虑系数为0的项是否需要存在链表当中。

题目四：一元多项式

- 相加：(指数相同)

list1.data.expn == list2.data.expn

处理list1.data.coef + list2.data.coef == 0

- 相减：(指数相同)

list1.data.expn == list2.data.expn

处理list1.data.coef - list2.data.coef == 0

- 相乘：(系数相乘，指数相加)

lst.data.coef = list1.data.coef * list2.data.coef

lst.data.expn = list1.data.expn + list2.data.expn

如何将计算的节点插入新生成的链表?

题目五：大整数加减法

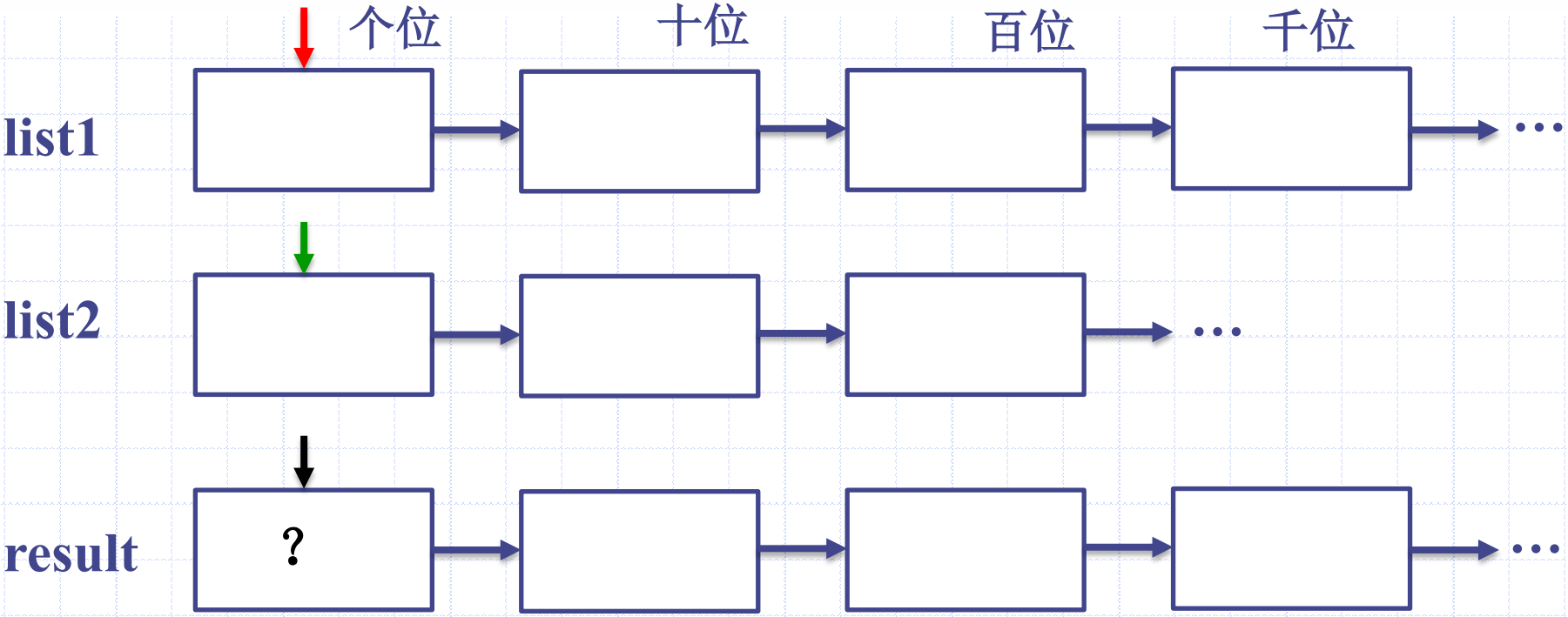
- 利用链表实现大整数加减法操作；
- 32位机器直接操作的数据最大为32个bit，若超过32bit，则需要单独设计算法。在这里，可以用链表的每个结点存储大整数的每一位的十进制数字，则可以进行大整数的算数运算，该实验仅实现加减法操作。

题目五：大整数加减法

要求：

- 随机产生2个1~50位的数字串，并存储到2个链表中；
- 进行加法或减法操作，结果存储到新的链表中；
- 打印运算结果。

题目五：大整数加减法



- 考虑链表结构的设计，是否有更节省空间的数据结构。

题目五：大整数加减法

考虑如下情况：

- 相加：list1.data + list2.data

list1.data < 0?

list2.data < 0?

list1.data[i] + list2.data[i] ≥ 10？如何进位

- 相减：list1.data - list2.data

list1.data < 0?

list2.data < 0?

list1.data[i] < list2.data[i]？如何借位

- 高位连续0如何处理?
- 计算结果为负如何处理

题目六：模拟内存管理

- 动态内存管理是操作系统的基本功能之一，用于响应用户程序对内存的申请和释放请求；
- 初始化时，系统只有一块**连续的空闲内存**；
- 当不断有用户申请内存时，系统会根据某种策略**选择一块合适的连续内存供用户程序使用**；
- 当用户程序释放内存时，系统将其回收，供以后重新分配，释放时需要计算该**内存块的左右是否也为空闲块**，若是，则需要**合并变成更大的空闲块**；
- 试设计用于模拟动态内存管理的内存池类。

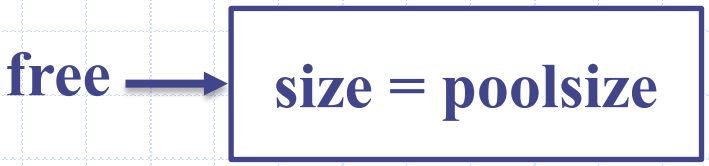
题目六：模拟内存管理

要求：

- **实现内存池MemoryPool(int size)的初始化；**
- **实现Allocate(int size)接口；**
- **实现Free(void *p)接口；**
- **实现内存池的析构；**
- **在分配内存空间时可选择不同的内存分配策略：最佳拟合策略、最差拟合策略或最先拟合策略，实现其中至少两种分配策略；**
- **编写测试main()函数对类中各个接口和各种分配策略进行测试，并实时显示内存池中的占用块和空闲块的变化情况。**

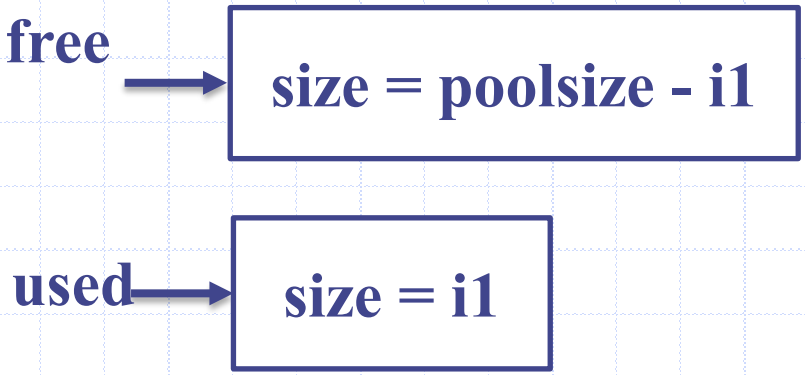
题目六：模拟内存管理

➢初始化：

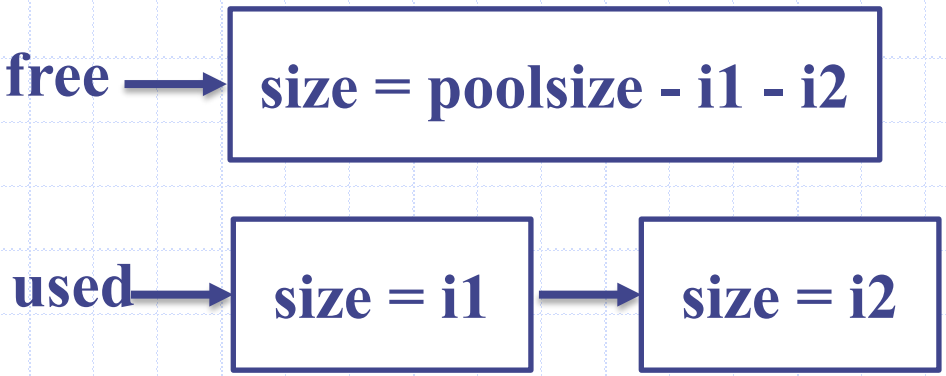


使用了i大小的内存

➢Allocate(int i1)：



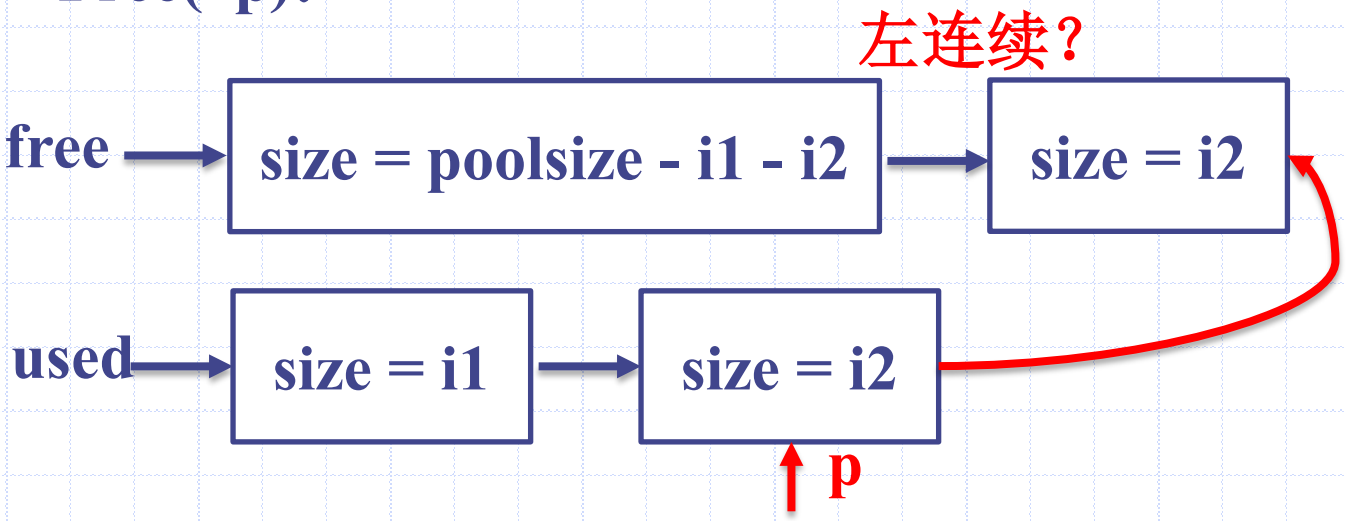
➢ Allocate(int i2)：



注意，不连续！

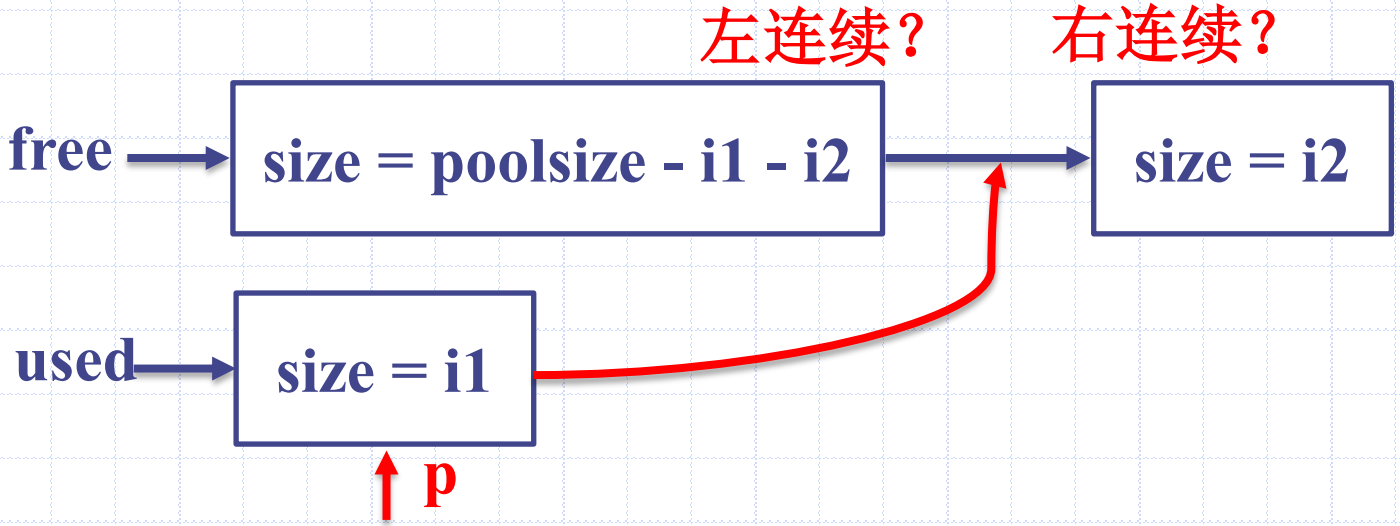
题目六：模拟内存管理

➢ Free(*p)：



题目六：模拟内存管理

➢ Free(*p)：



题目六：模拟内存管理

考虑如下情况：

- 初始化内存空间：

定义free、used?

- 分配内存空间：

最佳拟合？最差拟合？最先拟合?

- 回收内存空间：

找到插入位置?

和当前空闲空间左连续？ 如何合并

和当前空闲空间右连续？ 如何合并