



北京大学

《计算概论A》课程 程序设计部分

结构体与链表

李 戈

北京大学信息科学技术学院

lige@sei.pku.edu.cn



本节内容

■ 结构体

- ◆ 什么是结构体
- ◆ 结构体的使用

■ 链表

- ◆ 创建与操作
- ◆ 双向链表

■ 轻叩面向对象之门





经常遇到的情况

用一组变量来描述同一个“事物”

```
int    id;        //声明学号为int型;  
char   name[20];  //声明姓名为字符数组;  
char   sex;       //声明性别为字符型;  
int     age;      //声明年龄为整型;  
float  score;     //声明成绩为实型;  
char   addr[30];  //声明地址为字符数组
```



经常遇到的情况

用一组变量来描述同一个“事物”

```
int    id;        //声明学号为int型;  
char   name[20];  //声明姓名为字符数组;  
char   sex;       //声明性别为字符型;  
int     age;      //声明年龄为整型;  
float  score;     //声明成绩为实型;  
char   addr[30];  //声明地址为字符数组
```

构造一个新的数据类型 —— 结构体

```
struct student          //名字为 student 的结构体类型;
{
    int    id;           //声明学号为int型;
    char   name[20];     //声明姓名为字符数组;
    char   sex;          //声明性别为字符型;
    int    age;          //声明年龄为整型;
    float  score;        //声明成绩为实型;
    char   addr[30];     //声明地址为字符数组
};                       //注意大括号后的“;”
```

定义结构体类型的 变量

■ 定义结构体变量的方式

(1) 直接用已声明的结构体类型定义变量名

student **student1, student2;**

(结构体类型名) (结构体变量名) ;

◆ 对比:

int a; (student 相当于 int)

float a; (student 相当于 float)

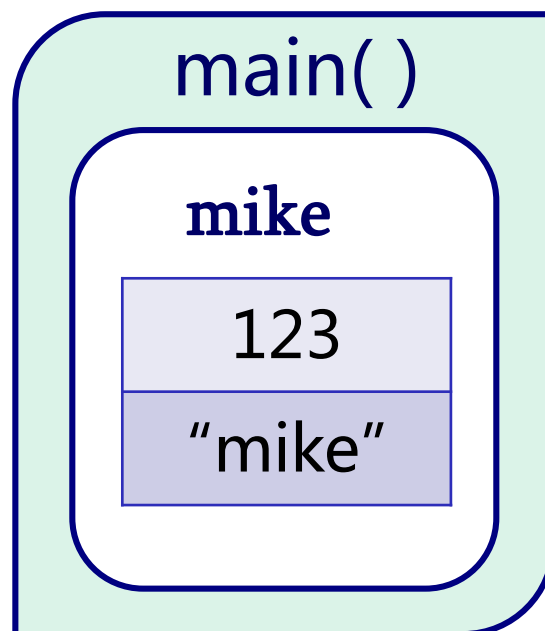
定义结构体类型的 变量

(2) 在声明类型的同时定义变量

```
struct student          \\结构体的名字为“student”;  
{   int    id;          \\声明学号为int型;  
    char   name[20];    \\声明姓名为字符数组;  
    char   sex;         \\声明性别为字符型;  
    int    age;         \\声明年龄为整型;  
    float  score;       \\声明成绩为实型;  
    char   addr[30];    \\声明地址为字符数组  
} lige_1, lige_2;      \\注意最后的“;”
```

定义结构体类型的 变量

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
int main()
{
    student mike = { 123, { 'm', 'i', 'k', 'e', '\0' } };
    mike.id_num = 20130000 + mike.id_num;
    for (int i = 0; mike.name[i] != '\0'; i++)
        mike.name[i] = toupper(mike.name[i]);
    cout << mike.id_num << " " << mike.name << endl;
    return 0;
}
```



2
0
1
3
1
2
3

M

I

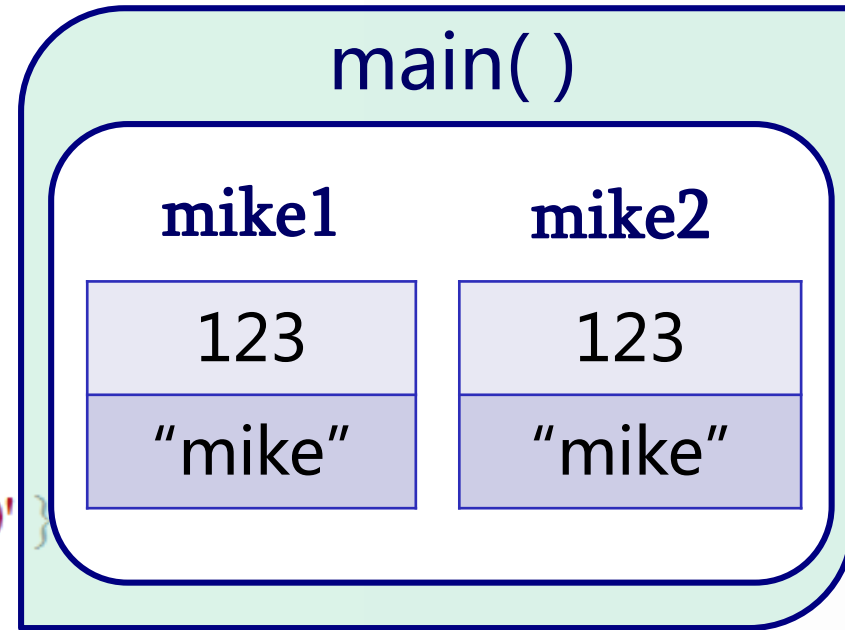
K

E

\0

结构体变量赋值

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
int main()
{
    student mike1 = { 123, { 'm', 'i', 'k', 'e', '\0' } };
    student mike2;
    mike2 = mike1;
    mike2.id_num = 20130000 + mike2.id_num;
    for (int i = 0; mike2.name[i] != '\0'; i++)
        mike2.name[i] = toupper(mike2.name[i]);
    cout << mike1.id_num << " " << mike1.name << endl;
    cout << mike2.id_num << " " << mike2.name << endl;
    return 0;
}
```

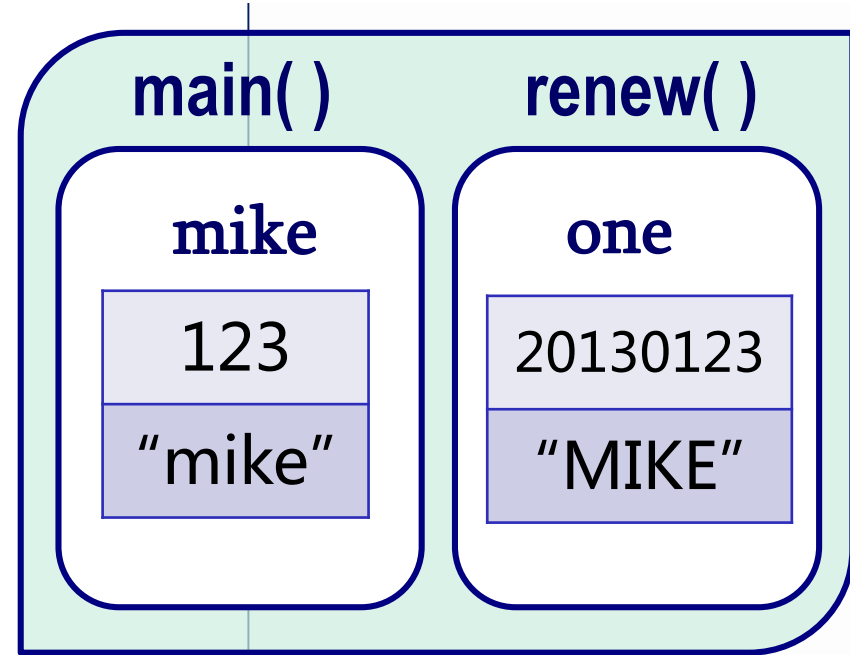


结构体赋值
相当于

Copy一份
给对方

结构体做函数参数

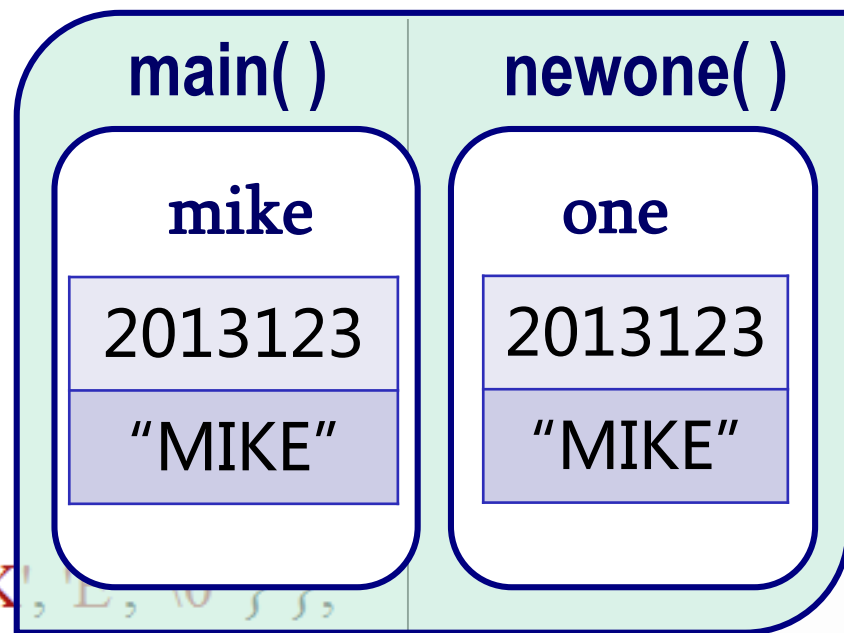
```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
void renew(student one)
{
    one.id_num = 20130000 + one.id_num;
    for (int i = 0; one.name[i] != '\0'; i++)
        one.name[i] = toupper(one.name[i]);
    cout << one.id_num << " " << one.name << endl;
}
int main()
{
    student mike = { 123, { 'm', 'i', 'k', 'e', '\0' } };
    renew(mike);
    cout << mike.id_num << " " << mike.name << endl;
    return 0;
}
```



结构体做参数
相当于
Copy一份
给函数

结构体做函数返回值

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
student newone()
{
    student one = { 20130123, { 'M', 'I', 'K', 'E', ' ' } };
    return one;
}
int main()
{
    student mike = newone();
    cout << mike.id_num << " " << mike.name << endl;
    return 0;
}
```



结构体做返回值
相当于

Copy一份
给调用者

指向结构体的指针

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
int main()
{
    student mike = { 123, { 'm', 'i', 'k', 'e', '\0' } };
    student *one = &mike;
    cout << (*one).id_num << " " << (*one).name;
    return 0;
}
```

A terminal window with a black background and white text. The text '123 mike' is displayed on a single line, representing the output of the provided C++ code.

123 mike

指向结构体的指针

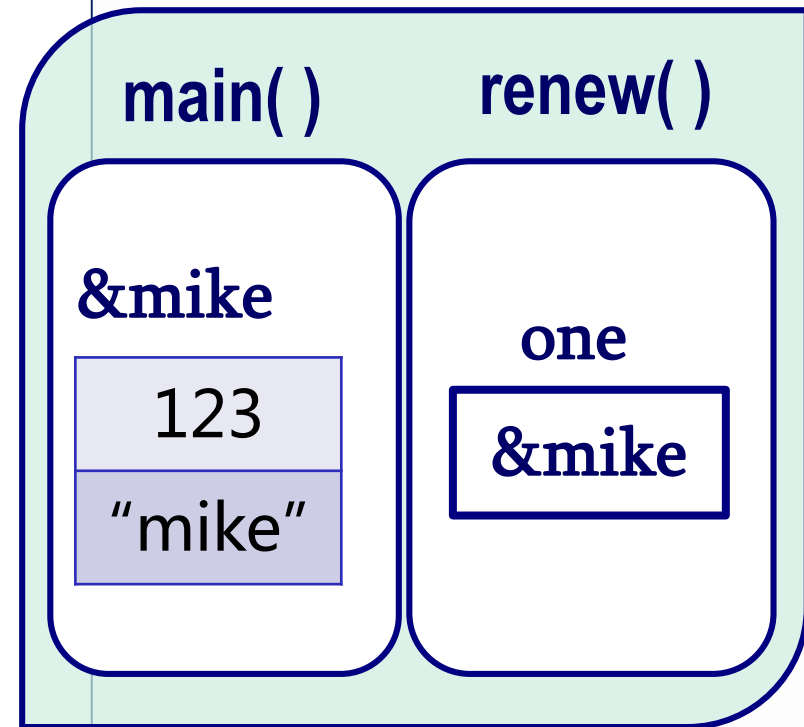
```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
int main()
{
    student mike = { 123, { 'm', 'i', 'k', 'e', '\0' } };
    student *one = &mike;
    cout << one->id_num << " " << one->name;
    return 0;
}
```

A terminal window with a black background and white text displaying the output of the C++ program: "123 mike".

123 mike

指向结构体的指针

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
void renew(student *one)
{
    one->id_num = 20130000 + one->id_num;
    for (int i = 0; one->name[i] != '\0'; i++)
        one->name[i] = toupper(one->name[i]);
}
int main()
{
    student mike = { 123, { 'm', 'i', 'k', 'e', '\0' } };
    renew(&mike);
    cout << mike.id_num << " " << mike.name;
    return 0;
}
```



20130123 MIKE

结构体数组

- 数组名相当于指向数组第一个元素的指针；
- 指向元素的指针++，则跨过一整个结构体

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
int main()
{
    student myclass[3] =
    { 123, { 'm', 'i', 'k', 'e', '\0' },
      133, { 't', 'o', 'm', '\0' },
      143, { 'j', 'a', 'c', 'k', '\0' } };
    student * one = myclass;
    cout << one->id_num << " " << one->name << endl;
    one++;
    cout << one->id_num << " " << one->name << endl;
    return 0;
}
```

143

"jack"



小结

- 由上述分析可以看出：

**结构体数据类型的特性
与
普通数据类型的特性
是一致的。**



生日相同问题

■ Description

- ◆ 在一个有100人的大班级中，存在多个生日相同的同学概率非常大。现给出每个学生的学号，出生月日。请列出所有生日相同的同学。

■ Input

- ◆ 第一行为整数 n ，表示有 n 个学生， $n < 100$ 。
- ◆ 此后每行包含一个字符串和两个整数，分别表示学生的学号（字符串长度小于10）和出生月($1 \leq m \leq 12$)日($1 \leq d \leq 31$)。
- ◆ 学号、月、日之间用一个空格分隔。

■ Output

- ◆ 对每组生日相同的学生，输出一行，
- ◆ 其中前两个数字表示月和日，后面跟着所有在当天出生的学生的学号，数字、学号之间都用一个空格分隔。
- ◆ 对所有的输出，要求按日期从前到后的顺序输出。
- ◆ 对生日相同的学号，按输入的顺序输出。

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]	a[16]	a[17]	a[18]	a[19]	a[20]	a[21]	a[22]	a[23]	a[24]	a[25]	a[26]	a[27]	a[28]	a[29]	a[30]	a[31]
------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

[illegible]

生日相同问题

数组统计:

学号	学号	学号	学号	学号	学号	学号	学号	学号	学号
月	月	月	月	月	月	月	月	月	月
日	日	日	日	日	日	日	日	日	日
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

```
struct student
{
    char ID[10];
    int month;
    int day;
}stu[100];
```

```
void main ()
{
```

```
    int i, j, k, n, flag, count[100]={0};
    cout << "how many students ?";
    cin>>n;
    for(int i=0; i<n; i++)
```

```
        cin>>stu[i].ID >> stu[i].month>> stu[i].day;
    for(int m=1; m<=12; m++)
        for(int d=1; d<=31; d++)
        {
```

```
            flag=0; j=0;
```

```
            for(int i=0; i<n; i++)
```

```
                if (stu[i].month == m && stu[i].day == d)
                    {count[++j] = i; flag++; }
```

```
            if(flag>1) //count[j]用于记录生日相同同学的学号
```

```
            {
                cout<<m<<" "<<d<<" ";
```

```
                for(k=1; k<=j; k ++)
```

```
                    cout << stu[count[k]].ID << " "<< endl;
```

```
            }
```

```
        }
```

```
    }
```

```
struct student
```

```
{
```

```
    char ID[10];
```

```
    int month;
```

```
    int day;
```

```
}stu[100];
```



本节内容

■ 结构体

- ◆ 什么是结构体
- ◆ 结构体的使用

■ 链表

- ◆ 创建与操作
- ◆ 双向链表

■ 轻叩面向对象之门



结构体数组

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
int main()
{
    student myclass[3] =
    { 123, { 'm', 'i', 'k', 'e', '\0' },
      133, { 't', 'o', 'm', '\0' },
      143, { 'j', 'a', 'c', 'k', '\0' } };
    student * one = myclass;
    cout << one->id_num << " " << one->name << endl;
    one++;
    cout << one->id_num << " " << one->name << endl;
    return 0;
}
```

123

"Mike"

133

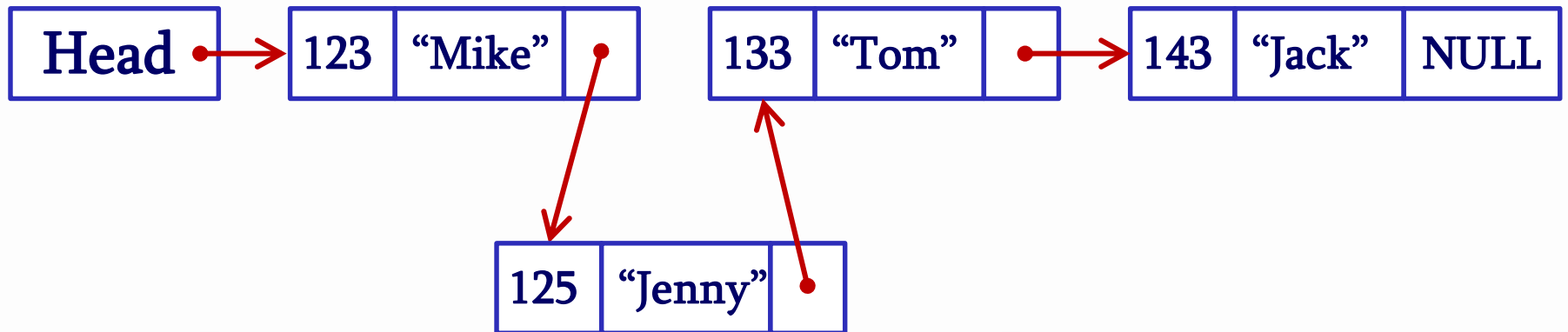
"Tom"

143

"Jack"

可以采取的办法

■ 用指针把结构体链起来！



链表



■ 一种非常常用的数据结构

- ◆ 链表头：指向第一个链表结点的指针；
- ◆ 链表结点：链表中的每一个元素，包括：
 - 当前节点的数据
 - 下一个结点的地址
- ◆ 链表尾：不再指向其他结点的结点，其地址部分放一个NULL，表示链表到此结束。

链表可以 动态地 创建

■ 动态地 申请内存空间

◆ `int *pint = new int(1024);` `delete pint;`

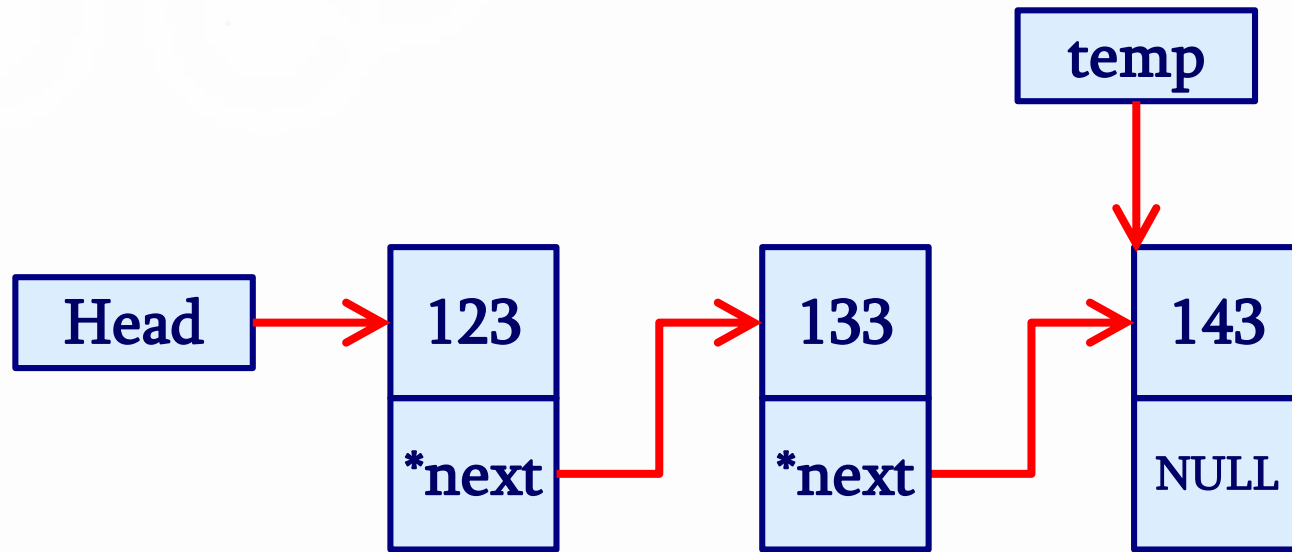
◆ `int *pia = new int[4];` `delete [] pia;`

■ 动态地 建立链表节点

```
struct student
{
    int id;
    student *next;
};
```

```
student *head;
head = new student;
```

链表结构——逐步建立链表



■ Step1:

- ◆ head = new student;
- ◆ student *temp = head;

■ Step2:

- ◆ Continue?

■ Y:

- ◆ temp->next = new student;
- ◆ temp = temp->next
- ◆ goto Step2

■ N:

- ◆ temp->next = NULL

```
student *create()
```

```
{
```

```
    student *head, *temp;    int num, n = 0;
```

```
    head = new student;
```

```
    temp = head;
```

```
    cin >> num;
```

```
    while (num != -1)
```

```
    {
```

```
        n++;
```

```
        temp->id = num;
```

```
        temp->next = new student;
```

```
        temp = temp->next;
```

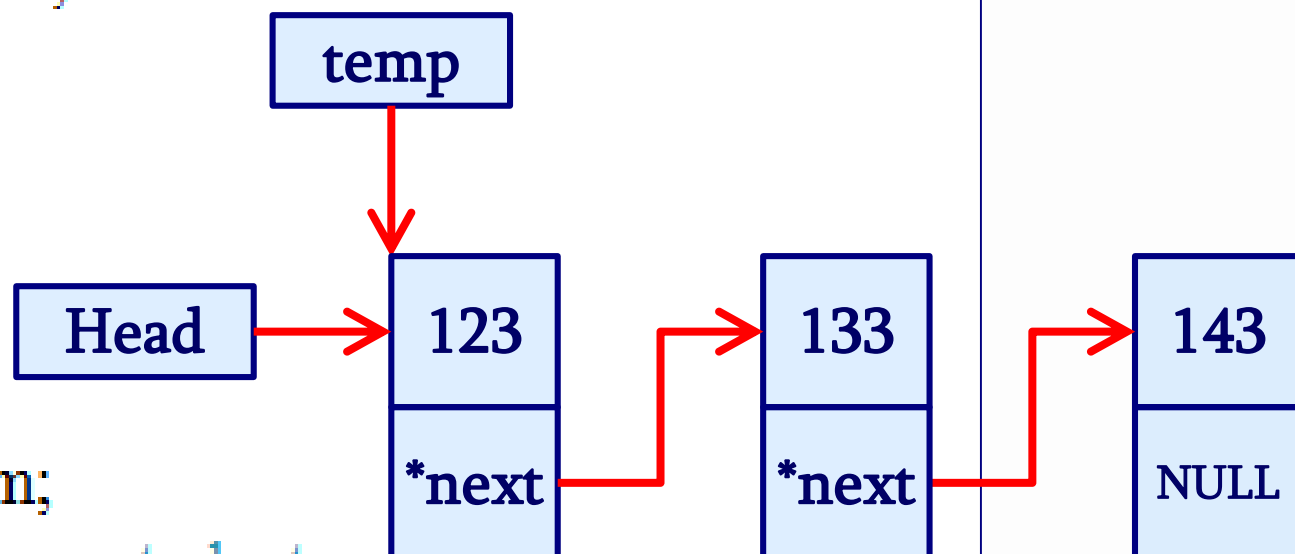
```
        cin >> num;
```

```
    }
```

```
    if (n == 0) head = NULL; else temp->next = NULL;
```

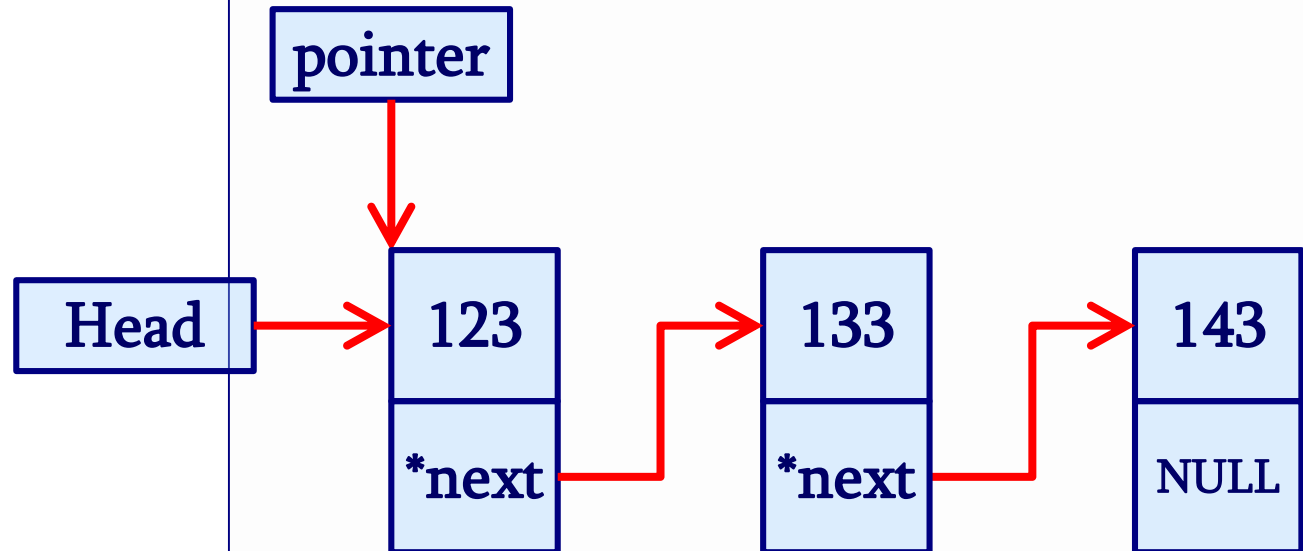
```
    return head;
```

```
}
```



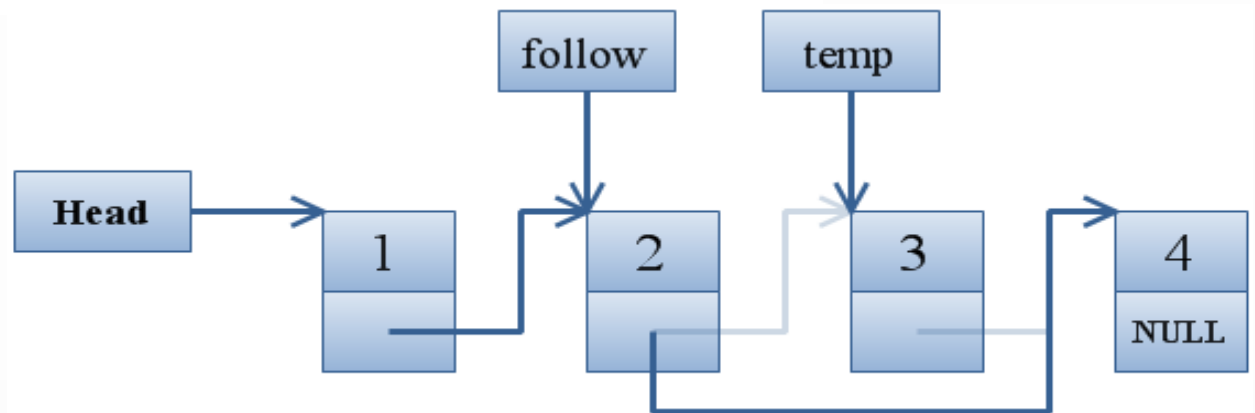
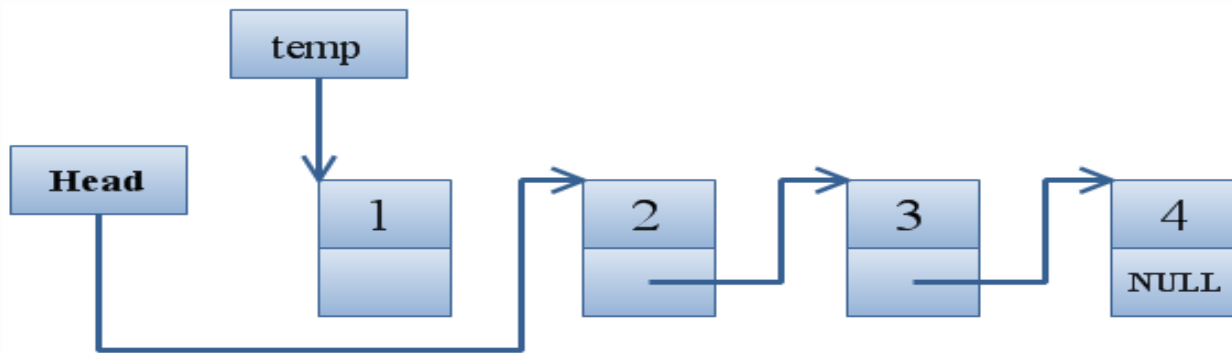
链表元素的遍历

```
#include<iostream>
using namespace std;
struct student
{
    int id;
    student *next;
};
student *create();
int main()
{
    student *pointer = create();
    while (pointer->next != NULL)
    {
        cout << pointer->id << endl;
        pointer = pointer->next;
    }
    return 0;
}
```



链表结构——删除结点

`temp=head; head = head→next; delete temp;`



`follow→next = temp→next; delete temp;`

在链表中将值为n的元素删掉

```
linker *dele(student *head; int n)
{
    student *temp,* follow;
    temp = head;
    if (head == NULL) {
        return(head);
    }
    if (head->num == n) {
        head = head->next;
        delete temp;
        return(head);
    }
    while(temp != NULL && temp->num != n){
        follow= temp;
        temp = temp->next;
    }
    if (temp == NULL) cout<<"not found";
    else {
        follow->next =temp->next;
        delete temp;
    }
    return(head);
}
```

//head为空，空表的情况

//第一个节点是要删除的目标；

//寻找要删除的目标；

//没寻到要删除的目标；

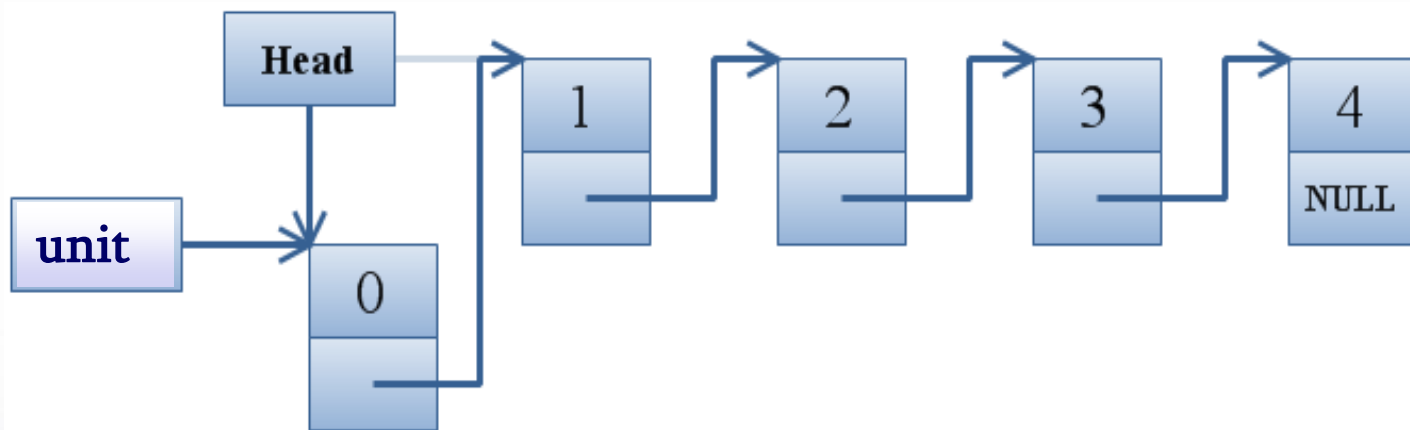
//删除目标节点；

链表结构——插入结点

■ 将结点unit插入链表：

插在最前面的情况

```
unit->next = head;  
head = unit;
```

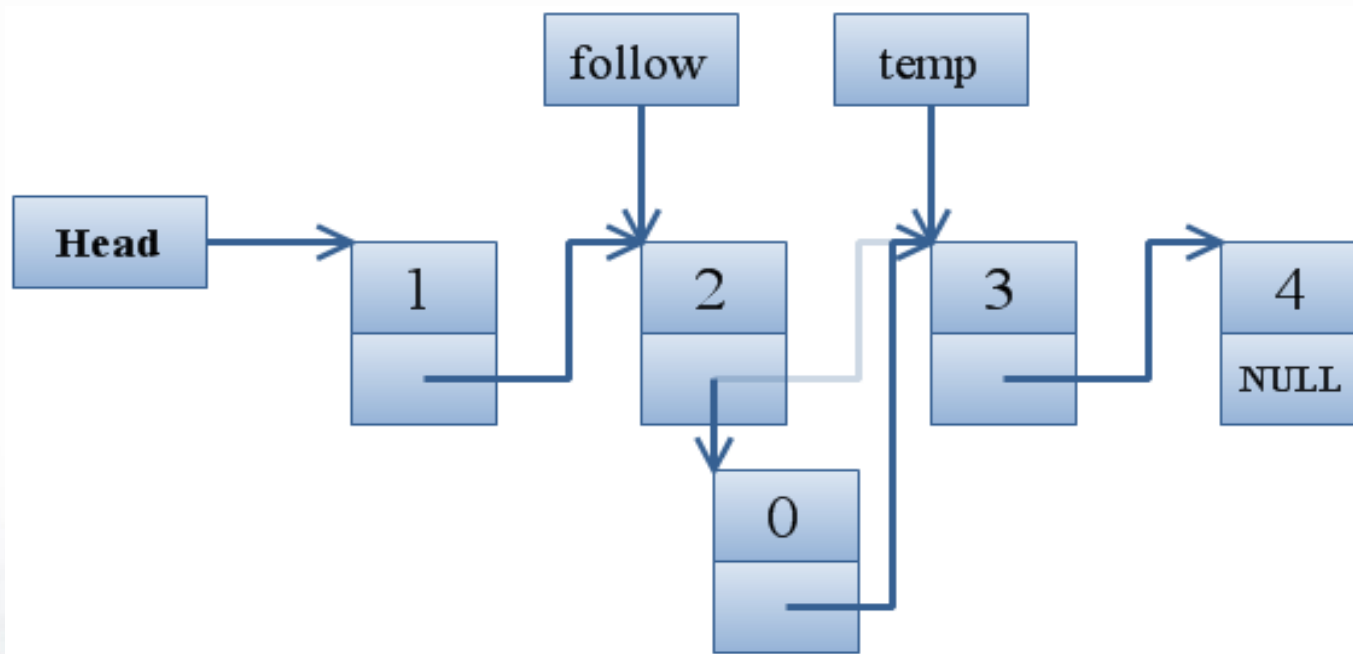


链表结构——插入结点

■ 将结点unit插入链表:

插在中间的情况

```
unit->next = temp;  
follow->next = unit;
```

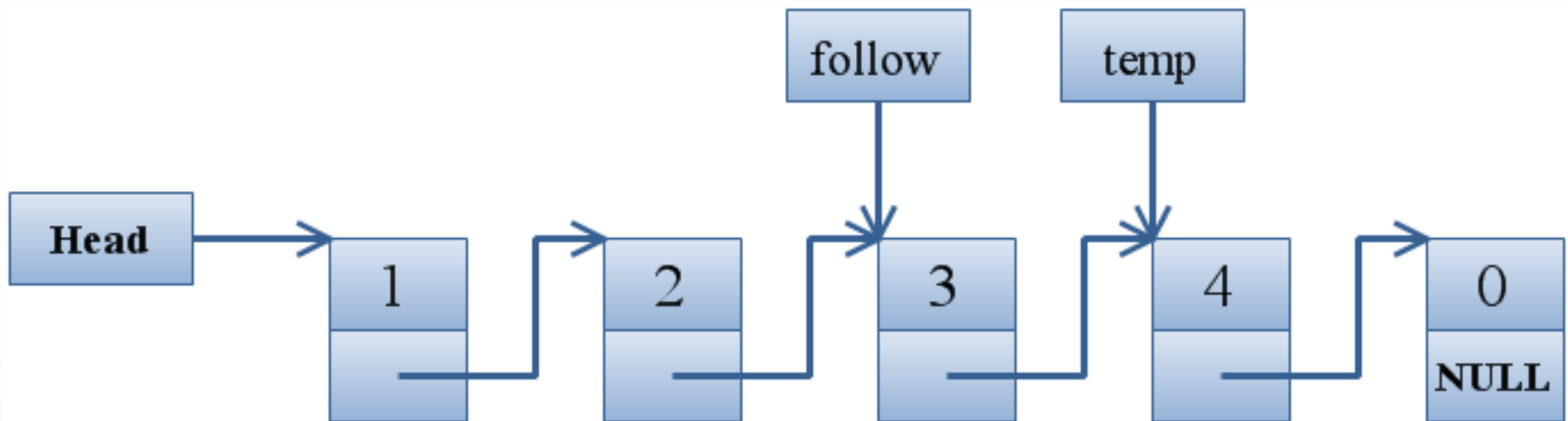


链表结构——插入结点

■ 将结点unit插入链表:

插在最后面的情况

```
temp->next = unit;  
unit->next = NULL;
```

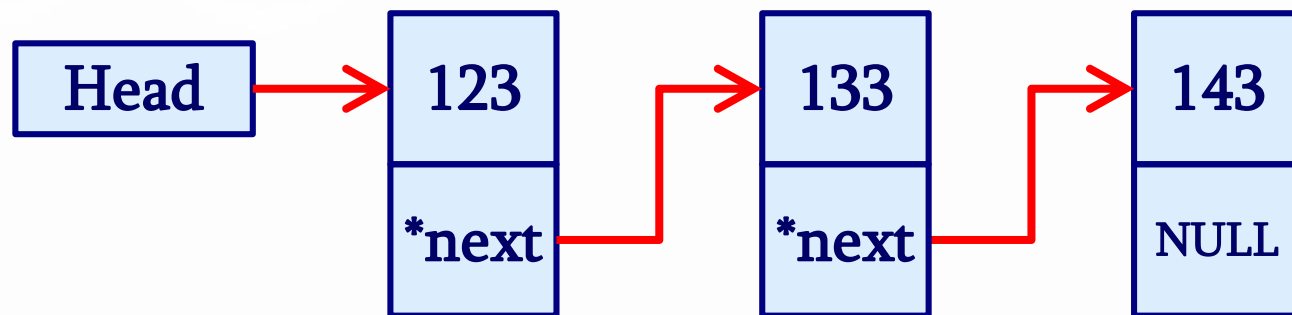


```

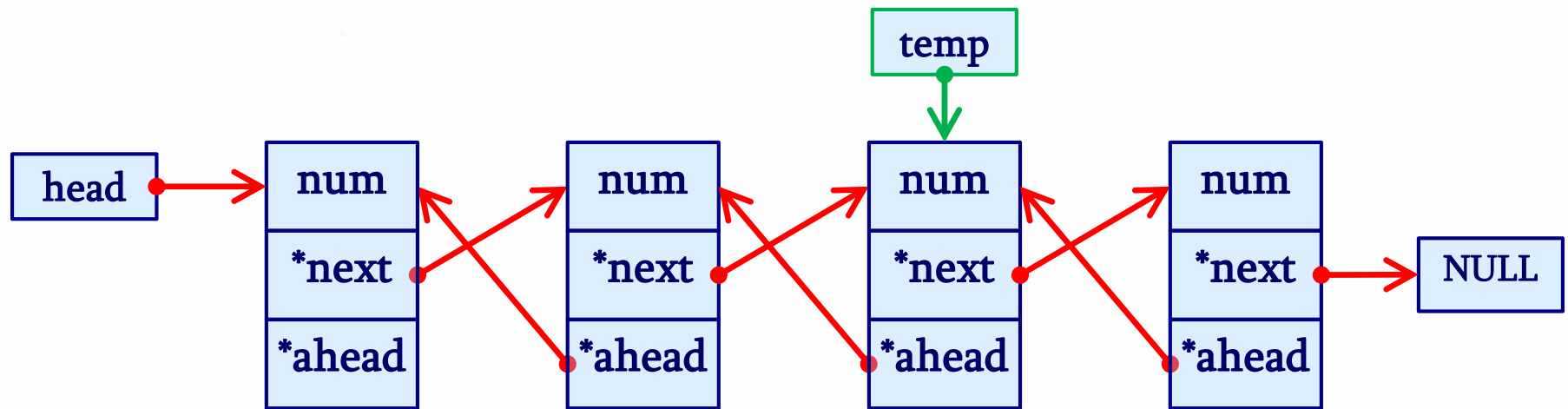
Student *insert(student *head; int n) {
    student *temp,*unit,*follow;           //插入结点值为n的结点
    temp = head; unit = new student;
    unit->num = n; unit->next = NULL;
    if (head==NULL) {                       //如果链表为空，直接插入
        head=unit;
        return(head);
    }                                       //寻找第一个不小于n或结尾的结点temp
    while((temp->next != NULL)&&(temp->num < n)){
        follow=temp; temp=temp->next;
    }
    if (temp==head){                        //如果temp为第一个结点
        unit->next=head; head=unit;
    }
    else{                                   //如果temp为最后一个结点
        if( temp->next == NULL) temp->next = unit;
        else{                             //如果temp为一个中间结点
            follow->next=unit; unit->next=temp;
        }
    }
    return(head);
}

```

单向链表

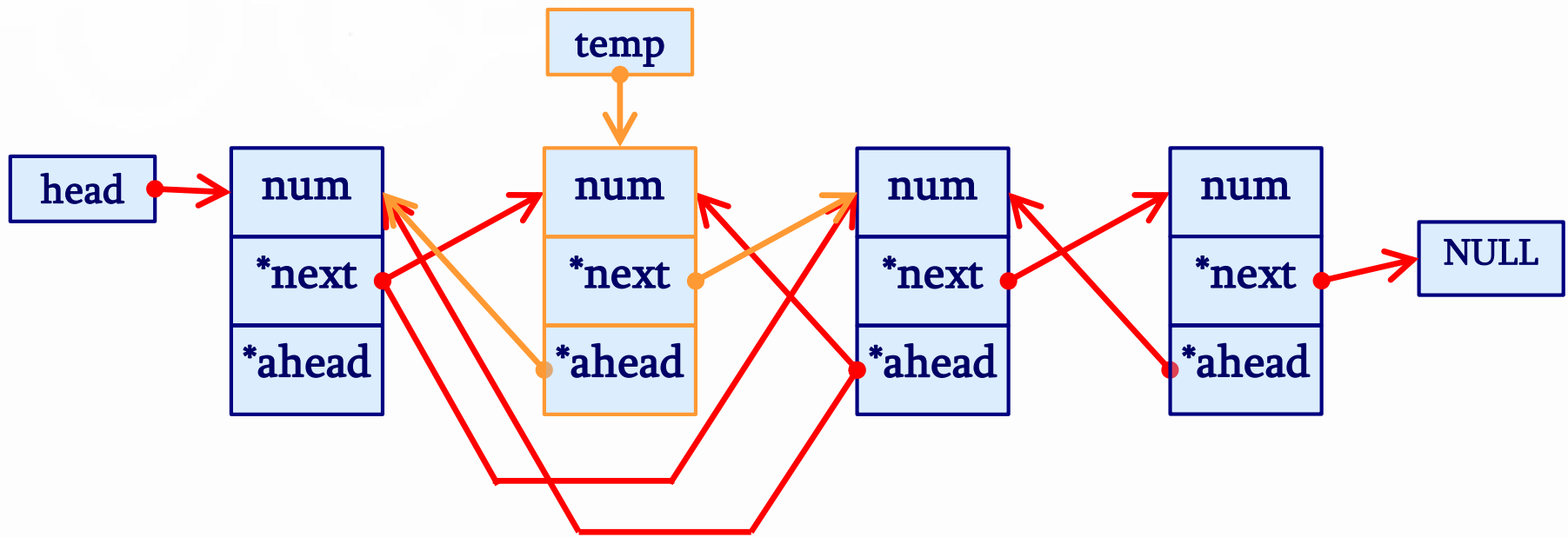


双向链表



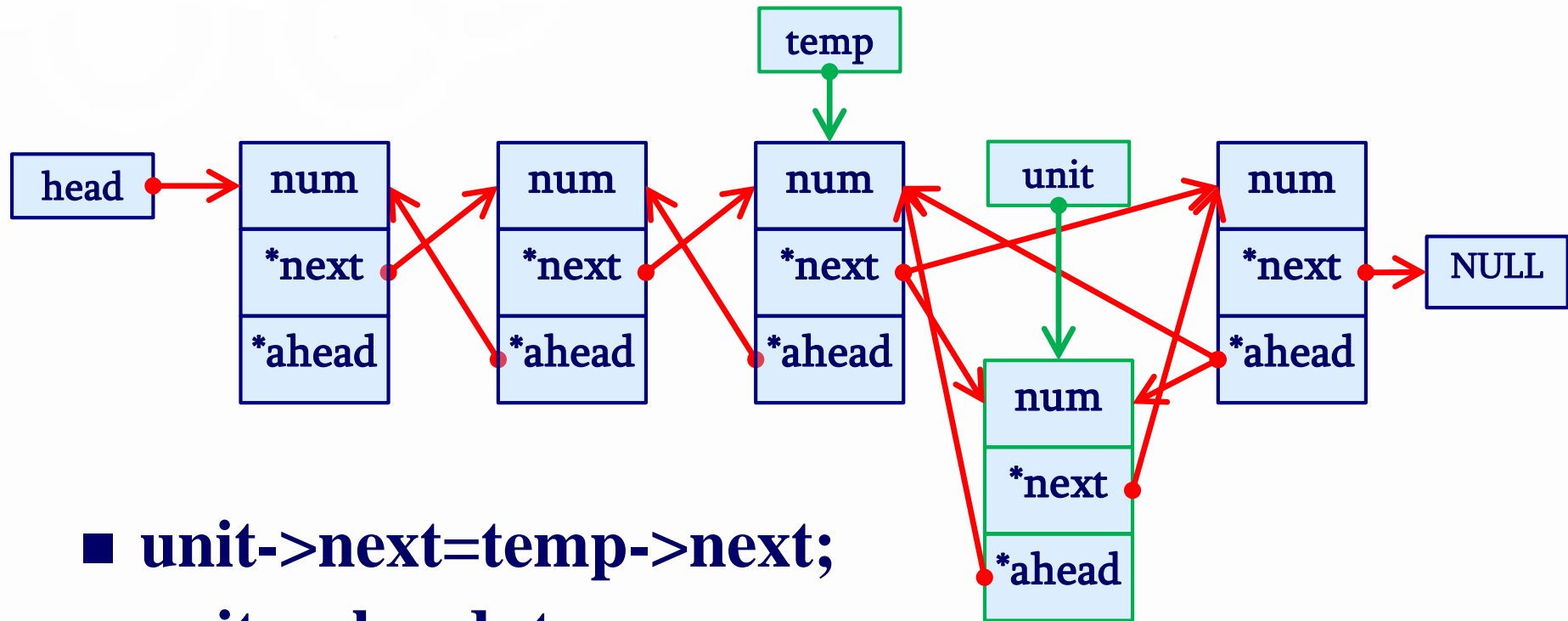
- `temp->num`: 存放数据
- `temp->next`: 指向下一个
- `temp->ahead`: 指向前一个

双向链表



- `temp->ahead->next=temp->next;`
- `temp->next->ahead=temp->ahead;`

双向链表

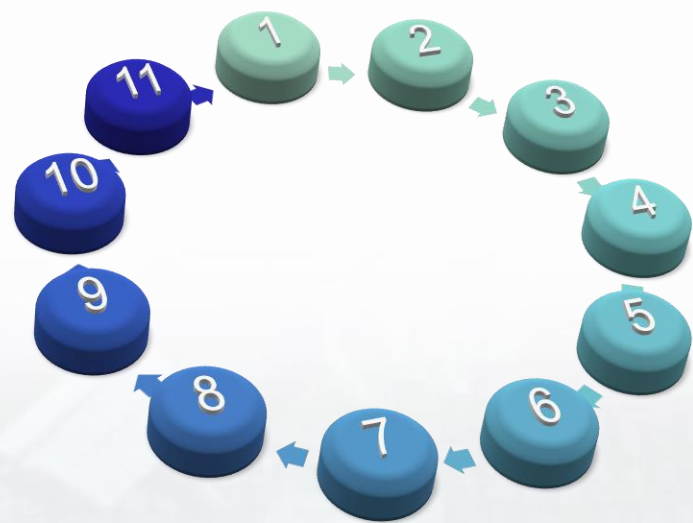


- `unit->next=temp->next;`
- `unit->ahead=temp;`
- `temp->next->ahead=unit;`
- `temp->next=unit;`

约瑟夫问题

■ 问题描述

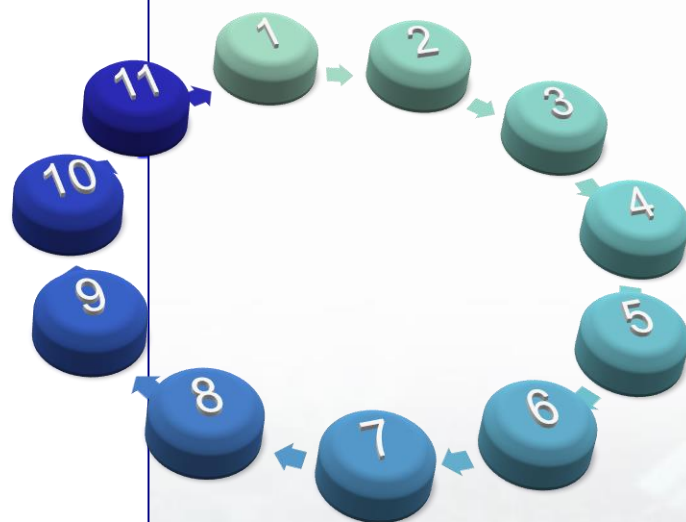
- ◆ 编号为1 – N的N个人围坐在一起形成一个圆圈，从第P个人开始，依次按照顺时针的方向报数，数到第M的人出列，直到最后剩下一个人。
- ◆ 请写一个程序，对于给定的 N, P, M ，计算并打印出依次出列的人的编号。



约瑟夫问题

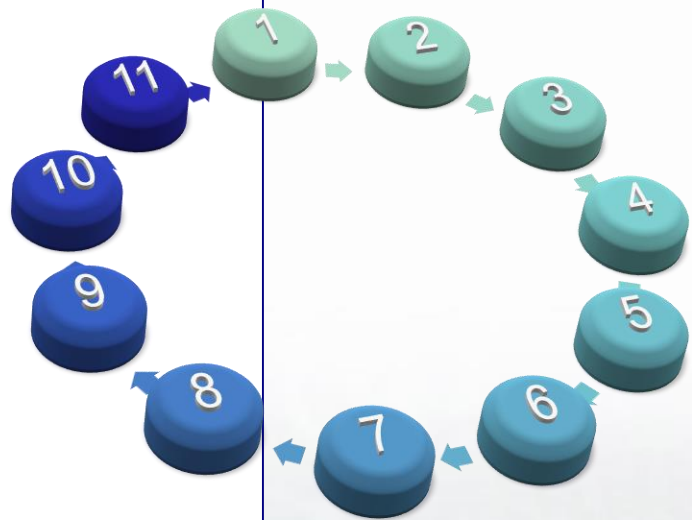
```
#include<iostream>
using namespace std;
struct Node
{
    int num;
    Node * ahead;
    Node * next;
};
Node *Create(int N);
Node *Search(Node *head, int P);
Node *Release(Node *head, int M);
int main()
{
    int N, P, M = 0;           //N-起始节点数, P-开始节点
    cin >> N >> P >> M;       //每次释放第M个节点
    Node * head = Create(N);    //创建N个节点的环
    head = Search(head, P);     //找到第P个节点
    while (head->next != head)  //不断释放第M个元素
    {                           //直到只剩一个元素
        head = Release(head, M); //释放第M个节点
    }
    cout << head->num;
    return 0;
}
```

```
11 3 7
9,5,2,11,10,1,4,8,3,6,7
```



约瑟夫问题

```
Node *Create(int N) //创建包含N个节点的双向循环链表
{
    int n = 1;
    Node * node = new Node;
    node->num = n;
    Node * head = node;    //指向第一个节点
    Node * tail = head;    //指向最后一个节点
    while (n++ < N)
    {
        node = new Node;    //建新节点
        node->num = n;        //赋值
        tail->next = node;    //接入新节点
        node->ahead = tail;
        tail = tail->next;    //尾巴后移
    }
    tail->next = head;        //尾巴处理
    head->ahead = tail;
    return head;
}
```



`Node *Search(Node *head, int P) //从Head开始寻找第P个节点`

```
{  
    while (head->num != P)  
    {  
        head = head->next;  
    }  
    return head;  
}
```

`Node *Release(Node *head, int M) //释放Head开始的第M个节点`

```
{  
    int count = 1;  
    Node *temp = head;  
    while (count < M)    //寻找第M个节点  
    {  
        temp = temp->next;  
        count++;  
    }  
    temp->next->next = temp->next;    //移除第M个节点  
    temp->next->next = temp->next;    //移除第M个节点  
    cout << temp->num << ", ";  
    head = temp->next;    //释放第M个节点所占内存空间  
    delete temp;  
    return head;  
}
```

