# 链表+习题

Wang Houfeng
EECS,PKU
wanghf@pku.edu.cn

# 内容

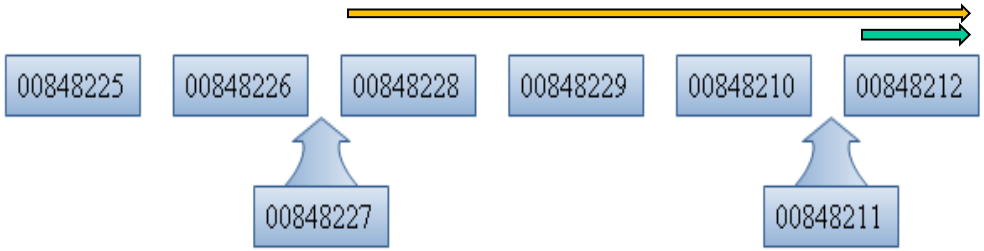
- **链表**
- 例题

# 为什么需要链表

- 如何表示多个相同对象？
  - 数组
- 如何表示一个复杂对象？
  - 结构
- 如何表示多个相同的复杂对象？
  - 例如：学生信息处理（学号，姓名，性别，出生日期...）
  - 结构数组

# 结构数组的问题

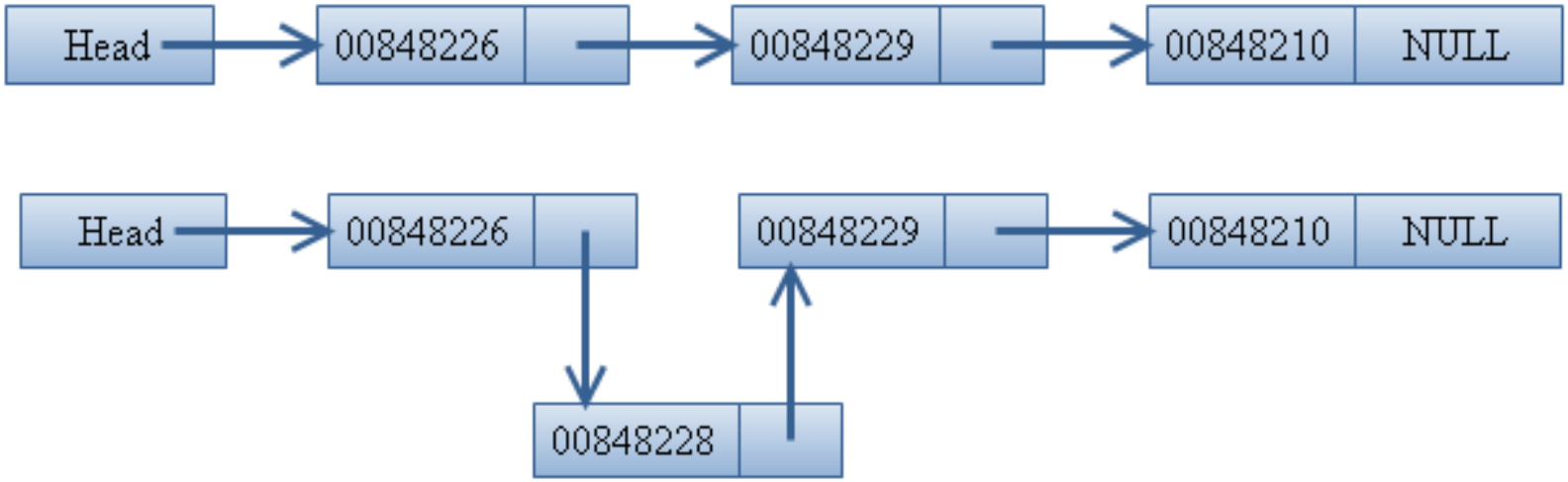
- 表示的对象个数事先不确定怎么办？
  - 例如：每年高考人数不同，不能因为人数变化每年更换系统
    - 定义过多会导致空间浪费
    - 定义过少可能空间不够用
- 插入与删除操作带来问题
  - 插入意味着后面的元素需要再向后移动
  - 删除意味着后面的元素需要向前移动
- 数组会永久性占用存储空间




插入元素示意图

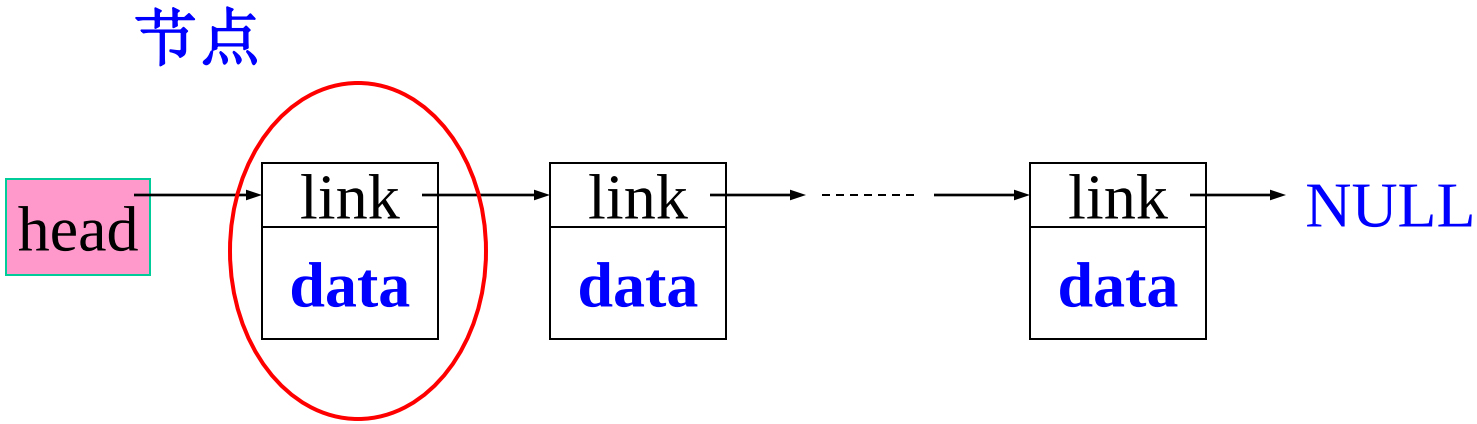
# 一种可能的解决办法：链表

- 链表结构



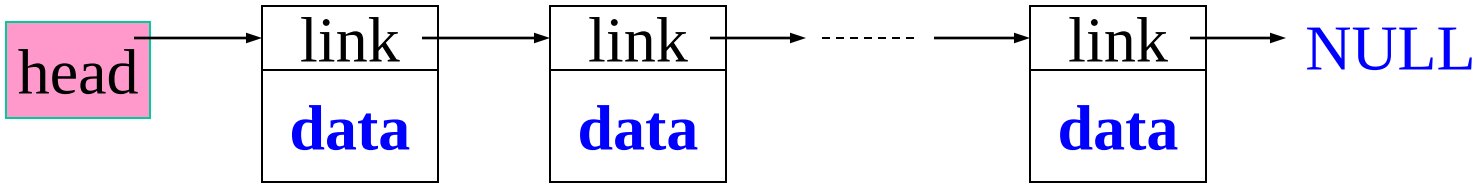

- 链表动态分配内存空间
  - 用多少申请多少

# 单向链表的形式




- 一种常用的动态数据结构
- 节点的有序集合;
- 每个节点包括两个大的部分： 数据部分和指向其它节点的指针， 节点由结构体描述；
- 指针表示节点之间的顺序关系,由节点中的一个成员（如，link）表示：该成员为指针， 起链接作用。

# 链表的特点




- 动态数据结构

关键是节点表示

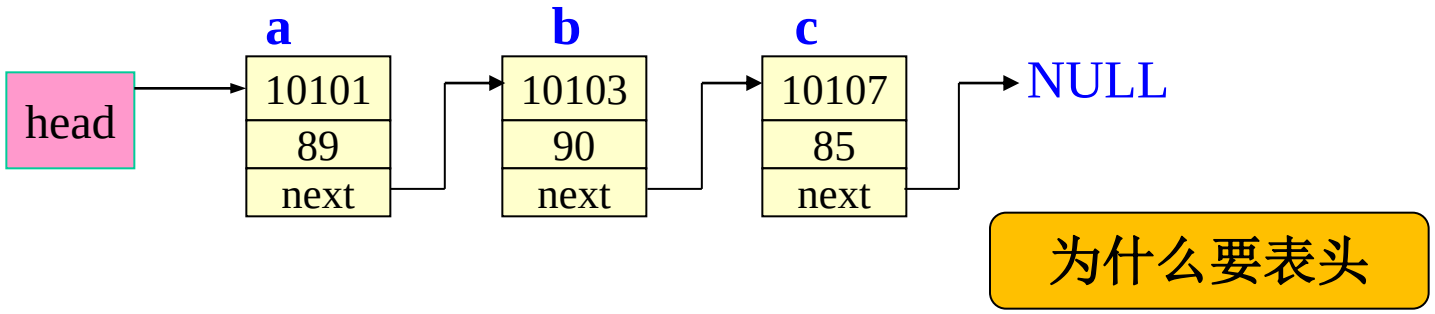
  - **链表头：**指向链表第一个节点的指针；
  - **链表节点**：链表中的每一个元素，包括：
    - 当前节点的数据
    - 下一个结点的地址
  - **链表尾：**不再指向其他结点的结点，其地址部分放一个NULL，表示链表到此结束

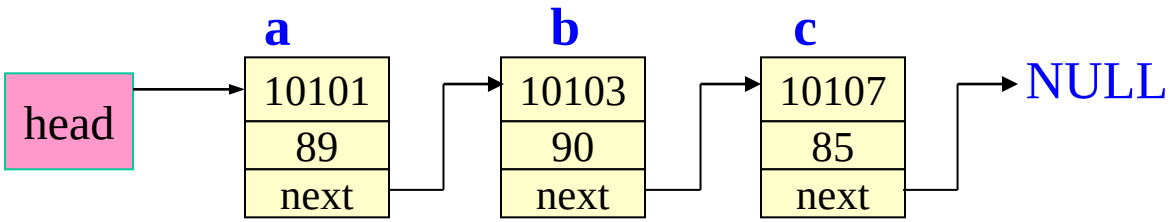
# 用结构体定义链节点

```
struct student
{   int num;
    int score;
    struct student *next ;
};
```

用 struct student 本身
说明成员 *next

- 链表的节点定义打破了先定义再使用的限制，即，可以用自己定义自己（回想递归函数的定义也违反了先定义再使用的限制）。






# 一个例子

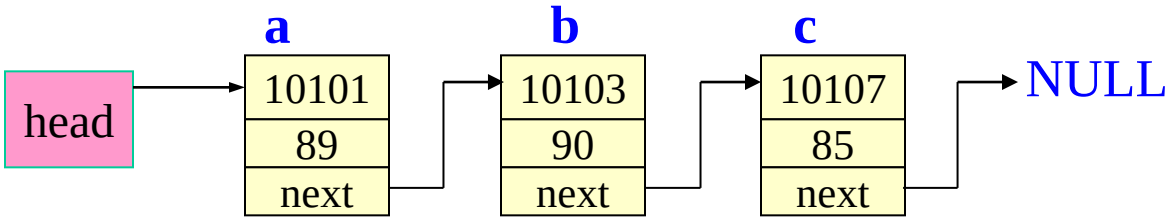
```
#include <iostream>
using namespace std;
#define NULL 0
struct student
    {int num;
     int score;
     struct student *next;  };
main()
{   struct student a,b,c,*head,*p;
    a. num=10101; a.score=89;
    b. num=10103; b.score=90;
    c. num=10107; c.score=85;
    head=&a; a.next=&b; b.next=&c; c.next=NULL; p=head;
    do{
          cout<<p->num<<p->score<<endl;
          p=p->next;
       } while(p!=NULL);
}
```

运行结果：

```
10101  89
10103  90
10107  85
```

**理解NULL的作用**

# 解释




- 开始时使head指向a节点，a.next指向b节点，b.next指向c节点；单向链表中的 head 非常重要，告诉了从哪里开始
- 通过指针的指向关系构成链表；
- c.next=NULL 的作用是使c.next不指向任何有用的存储单元。
- 在输出链表时要借助 p，
  - 先使p指向a节点，然后输出a节点中的数据，p=p->next 是为输出下一个节点作准备；
  - p->next的值是b节点的地址，因此执行p=p->next后p就指向b节点，所以在下一次循环时输出的是b节点中的数据。

# C++的内存申请与释放

- new：获得动态区域指定大小的内存，并将内存地址给指针变量（可以初始化所获内存的值）
- delete：将动态分配的内存空间释放后还给系统

```
int main()
{
    int *p = new int;
    *p = 10;
    cout<<*p<<endl;
    delete p;
    cout<<*p<<endl;
    system("pause");
    return 0;
}
```

```
10
-17891602
请按任意键继续. . .
```

指针的动态性

```
int main()
{
    int *p = new int(5);
    cout<<*p<<endl;
    delete p;
    cout<<*p<<endl;
    system("pause");
    return 0;
}
```

```
5
-17891602
请按任意键继续. . .
```

# new & delete 用于申请或释放链表节点

```
#include<iostream>
using namespace std;
struct Node
{
    int n;
    Node * next;
};
int main()
{
    Node *p = new Node;
    cout<<p->n<<endl;
    cout<<p->next<<endl;
    delete p;
    cout<<p->n<<endl;
    cout<<p->next<<endl;
    system("pause");
    return 0;
}
```

new 直接按数据类型申请相应大小的空间

```
-842150451
CDCDCDCD
-17891602
FEEEFEEE
请按任意键继续. . .
```

# C中的内存申请与释放

- 在malloc.h 库中
  - **void** *malloc(unsigned int size)
    - 申请大小为 size的空间
    - **struct student *p=(struct student*) malloc(sizeof(struct student))**
    - 如果申请成功，返回首地址(注意:类型转换)；
    - 如果没有足够的空间，返回空（NULL）
  - **void** free(void *p)
    - 释放 p 所指的空间
    - free(p)

**malloc 按大小申请，再按类型转换**

# 创建链表

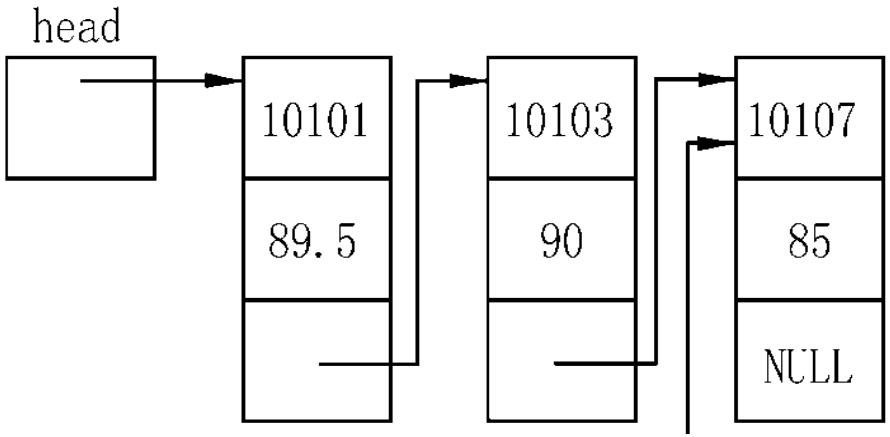
已知

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

先定义头节点，并申请空间

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

构建如下链表

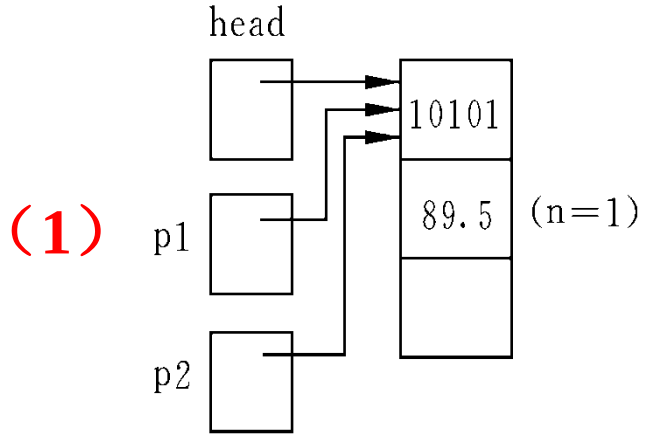


# 建立动态链表

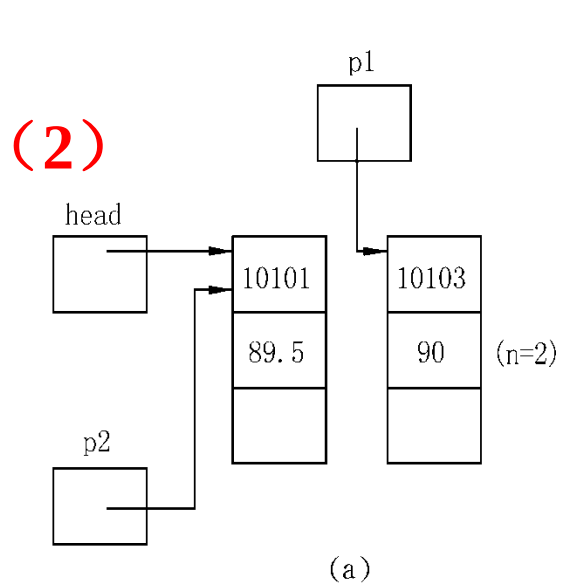
- 所谓建立动态链表是指在程序执行过程中从无到有地构建起一个链表，即一个节点一个节点地创建并输入各节点数据，并建立节点的前后链接关系
- 例　写一函数建立学生数据的单向动态链表。

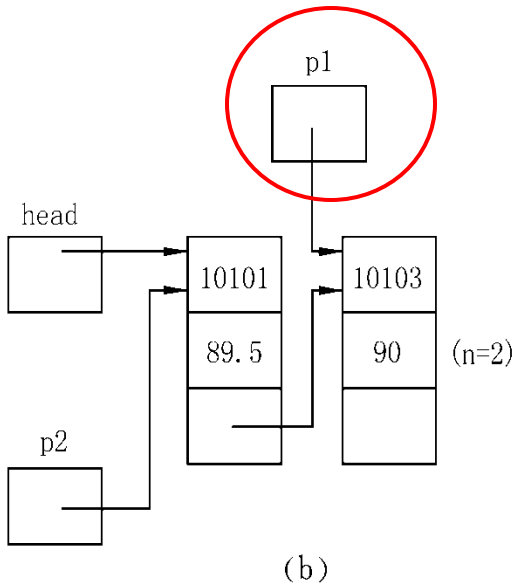
（约定学号不会为零，如果输入的学号为 0 ，则表示输入结束，链表的建立也结束）

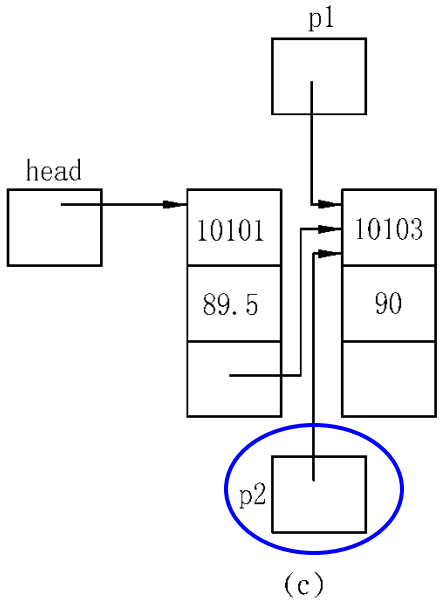
# 建链示意图

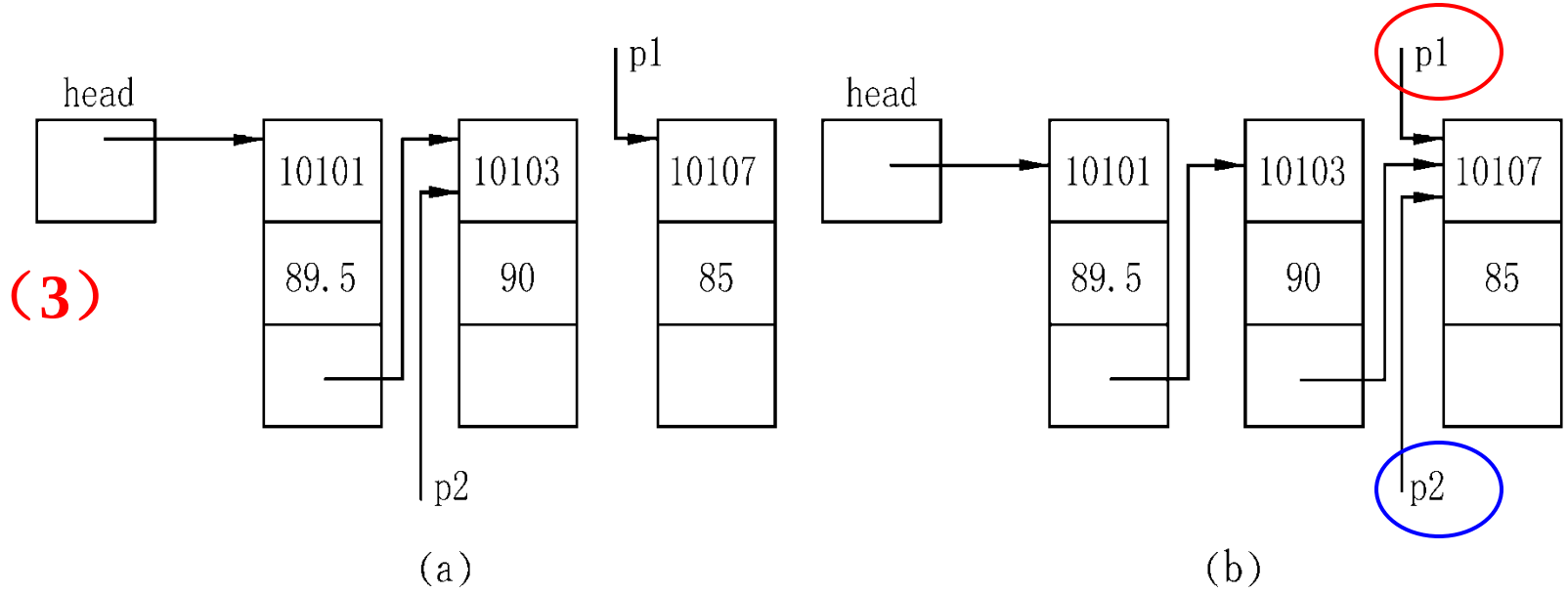



**理解head, p1和p2的作用**

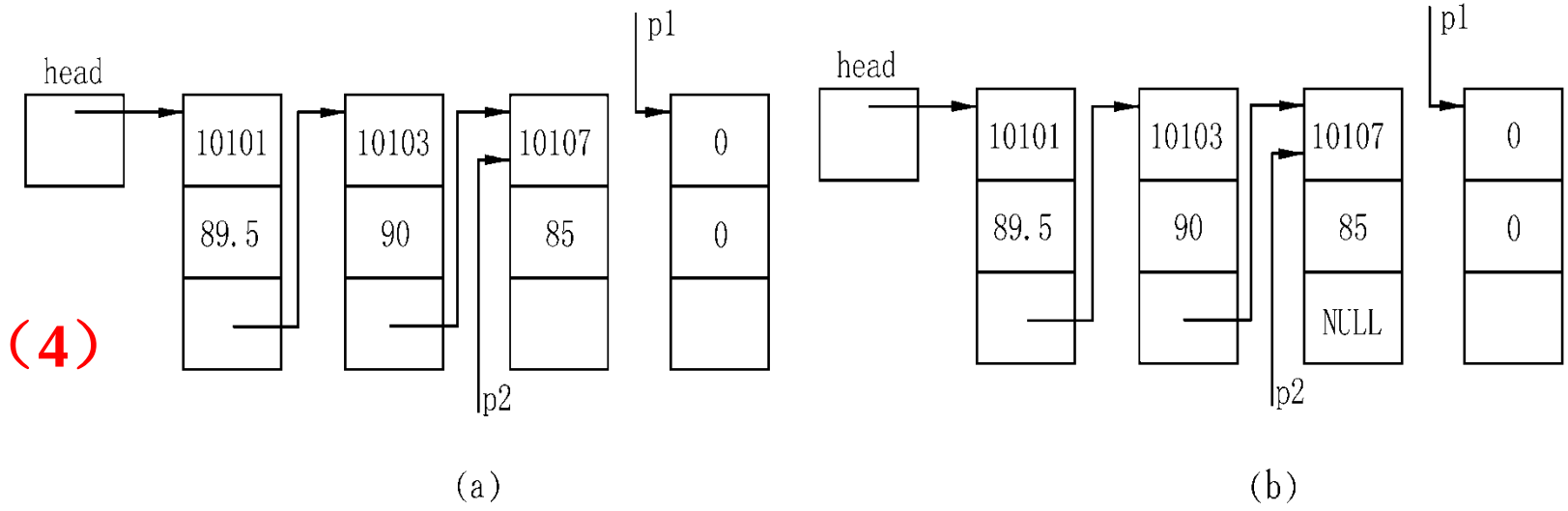



head
10101
89.5
10103
90
p1
10107
85
p2
(a)
head
10101
89.5
10103
90
p1
10107
85
p2
(b)

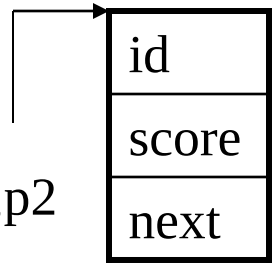
(3)


head
10101
89.5
10103
90
10107
85
p1
0
0
p2
(a)
head
10101
89.5
10103
90
10107
85
NULL
p1
0
0
p2
(b)

(4)

# 创建链表

```
struct student *creat()
{   struct student *head,  *p1, *p2;
    int   num,  n=0;
    float s;
    head=p1=p2= new student;     //从动态内存区域申请一个节点
    cin>>num>>s;
    while(num!=0)
    {   n++;
        p1->id=num; p1->score=s;
        p2->next=p1;        p2=p1;
        p1=new student; //为下一个节点申请空间
        cin>>num>>s;
    }
    p2->next=NULL;
    if(n==0)  head=NULL;
    delete p1;  //终止输入的节点（应删除）
    return(head);
}
```




假设输入4组数据：

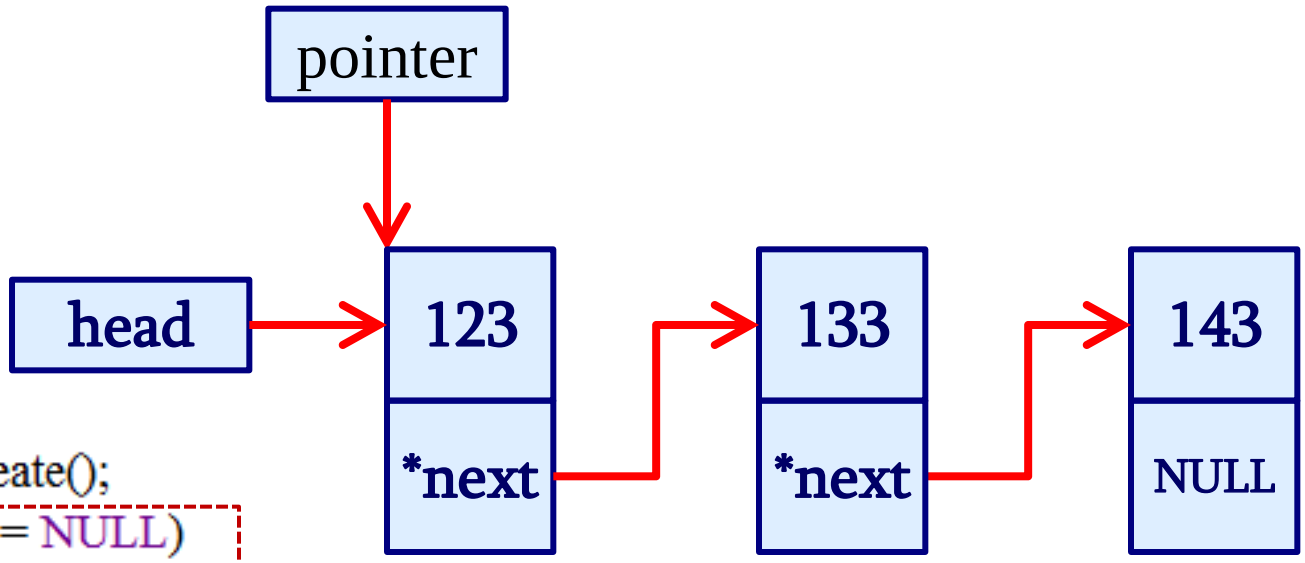
11108  80.5
11105  90.5
11106  87.8
11107  78.5
0  0

请画出链表结构

# 遍历链表

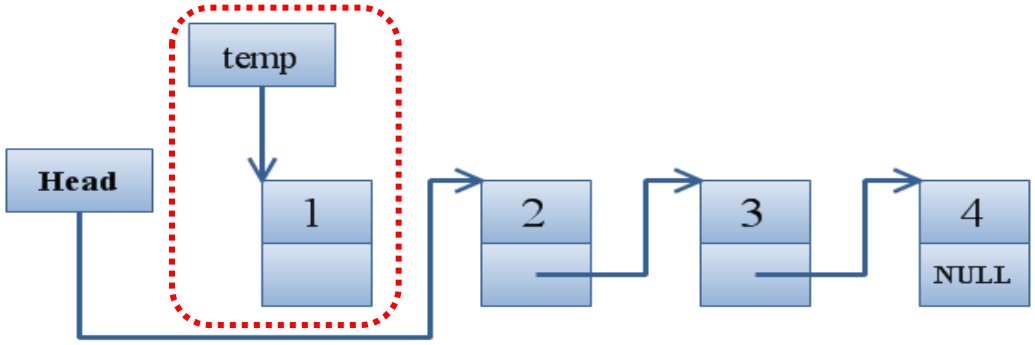
```
#include<iostream>
using namespace std;
struct student
{
    int id;
    student *next;
};
student *create();
int main()
{
    student *pointer = create();
    while (pointer->next != NULL)
    {
        cout << pointer->id <<endl;
        pointer = pointer->next;
    }
    return 0;
}
```

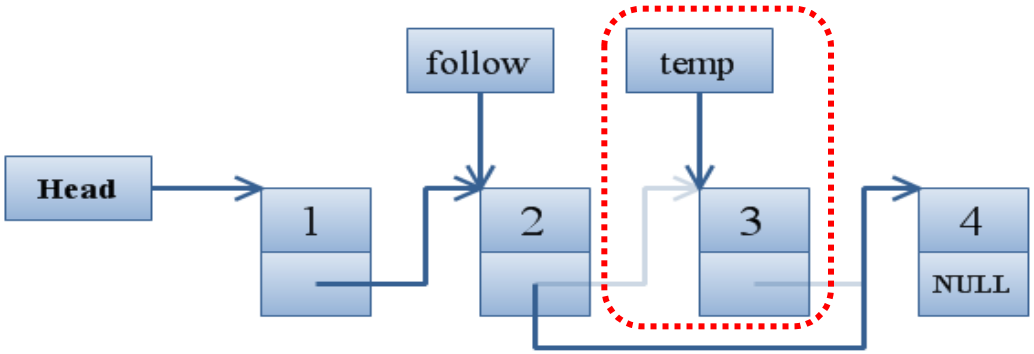
所谓遍历，就是逐个节点访问



# 链表节点删除

在表头：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

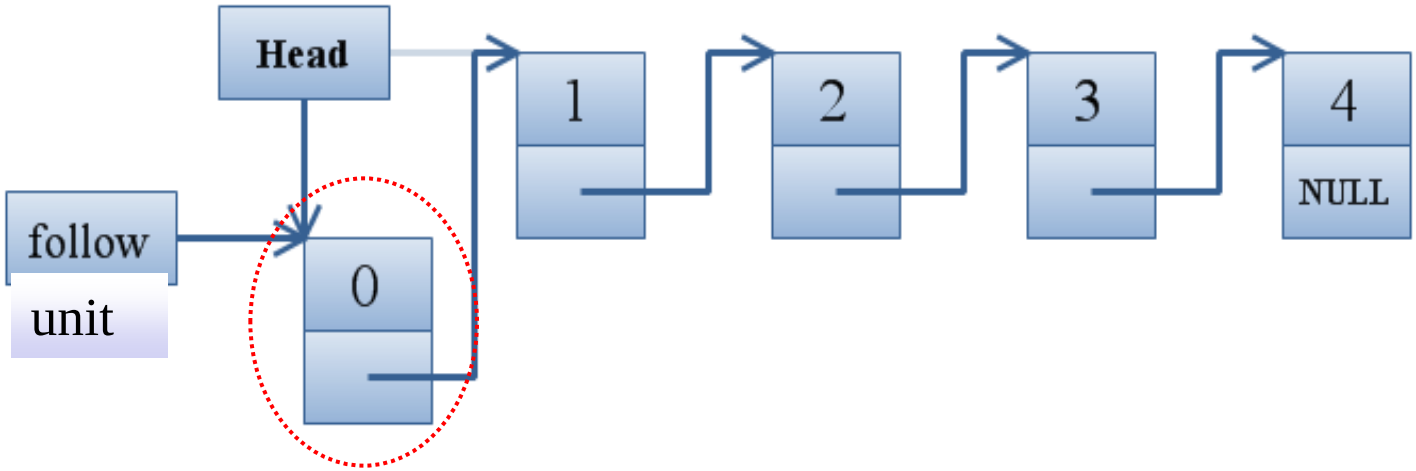
```
linker *dele(student *head; int n)
{    student *temp,* follow;
     temp = head;
     if (head == NULL)   return(head);   //head为空，空表的情况
     if (head->num = = n)        //第一个节点是要删除的目标；
     {        head = head->next;
              delete temp;
              return(head);
     }
     while(temp != NULL && temp->num != n)                    //寻找要删除的目标；
     {        follow= temp;
              temp = temp->next;
     }
     if (temp == NULL) cout<<"not found";          //没寻到要删除的目标；
     else {
              follow->next =temp->next;                    //删除目标节点；
              delete temp;
     }
     return(head);
}
```

# 链表中插入节点

- 将结点unit插入链表的头部：

**插在最前面**的情况

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

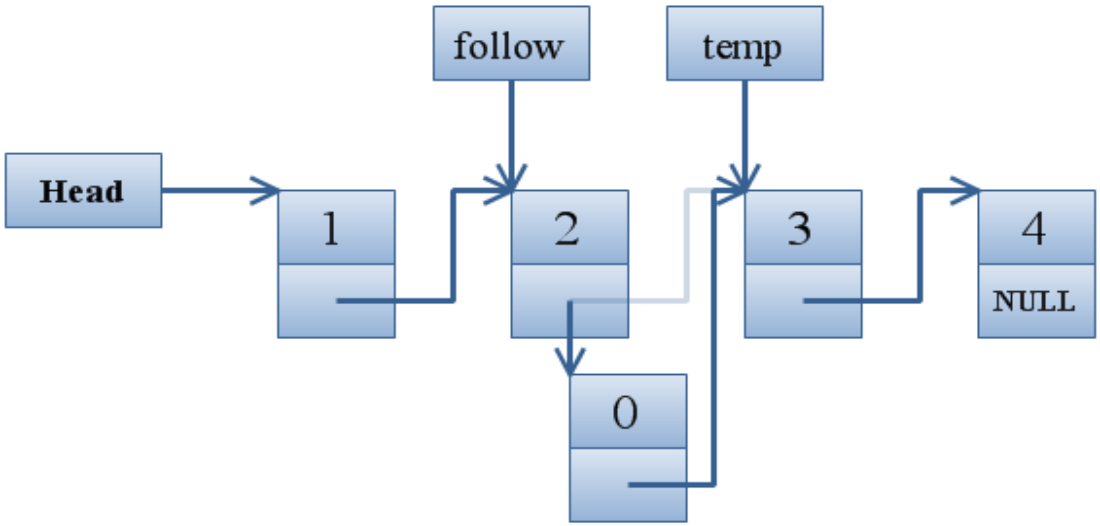


# 链表中插入节点

- 将结点unit插入链表的中间：

**插在中间**的情况

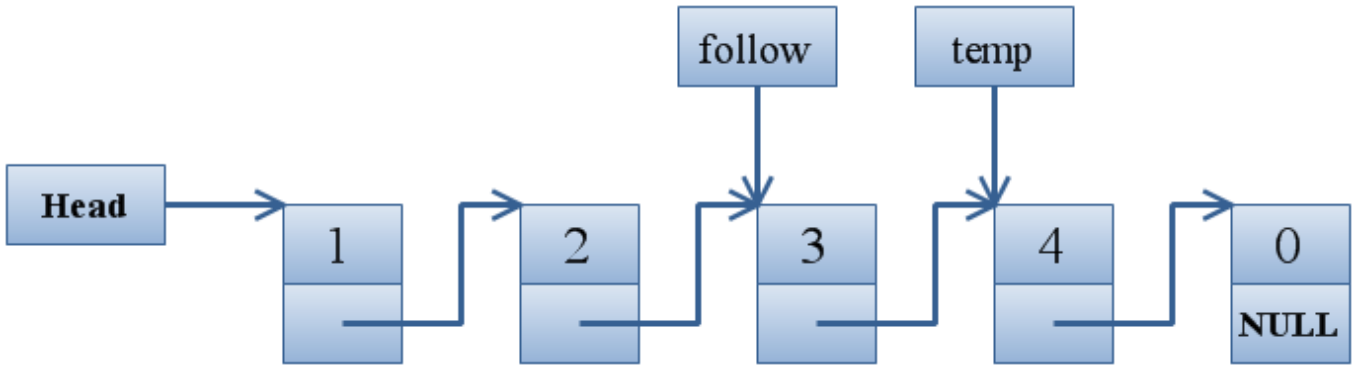
```
unit->next = temp;
follow->next = unit;
```



# 链表中插入节点

- 将结点unit插入链表的末尾：

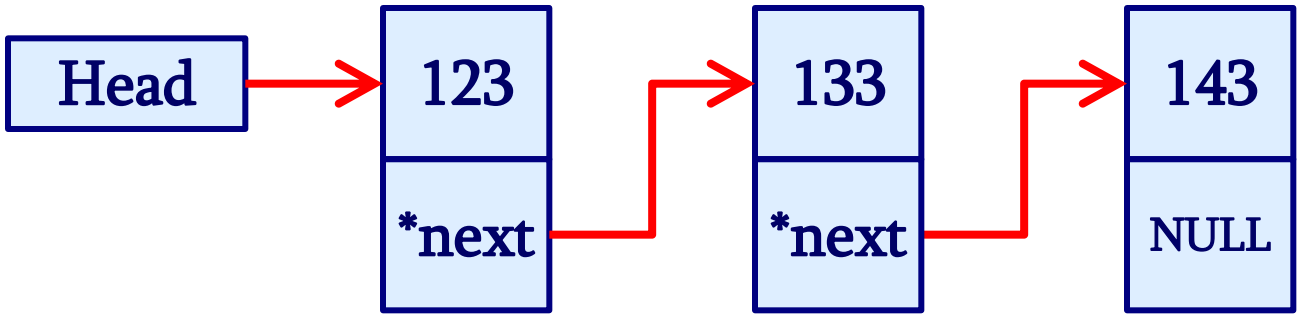
**插在最后面**的情况
temp->next = unit;



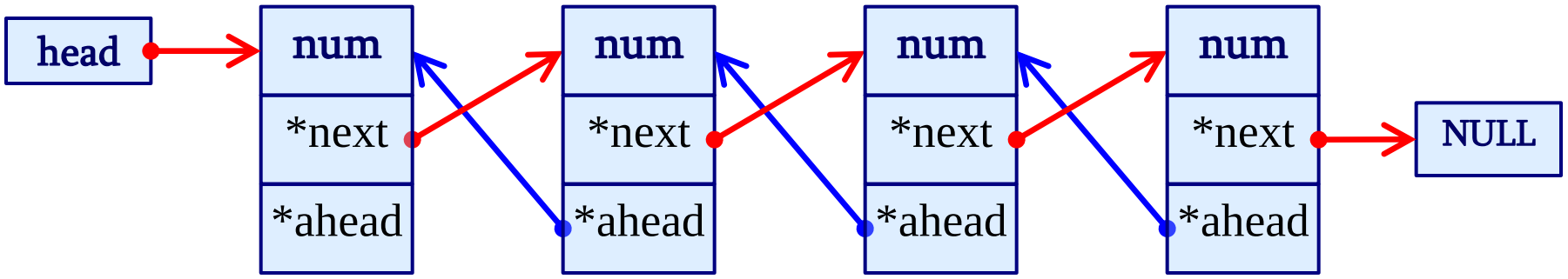
```
Student *insert(student *head; int n) {
    student *temp,*unit,*follow;                    //插入结点值为n的结点
    temp = head;   unit = new student;
    unit->num = n;    unit->next = NULL;
    if (head==NULL)                                 //如果链表为空，直接插入
    {       head=unit;
            return(head);
    }                                   //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



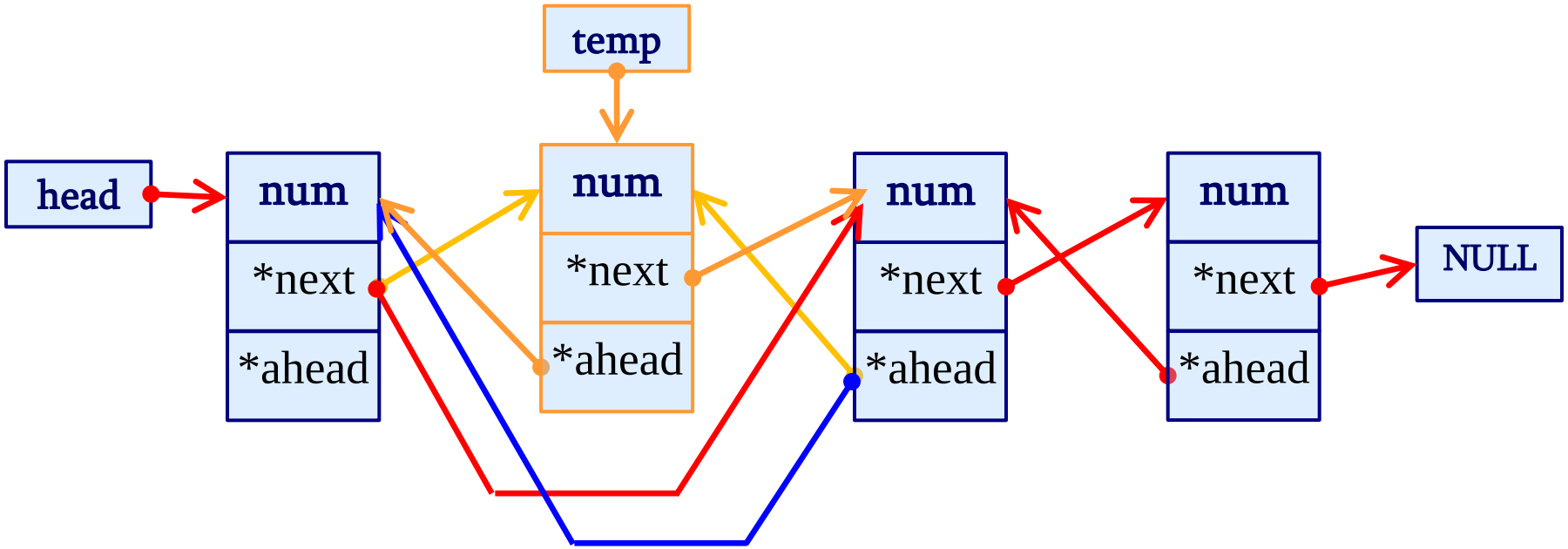
# 双向链表




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

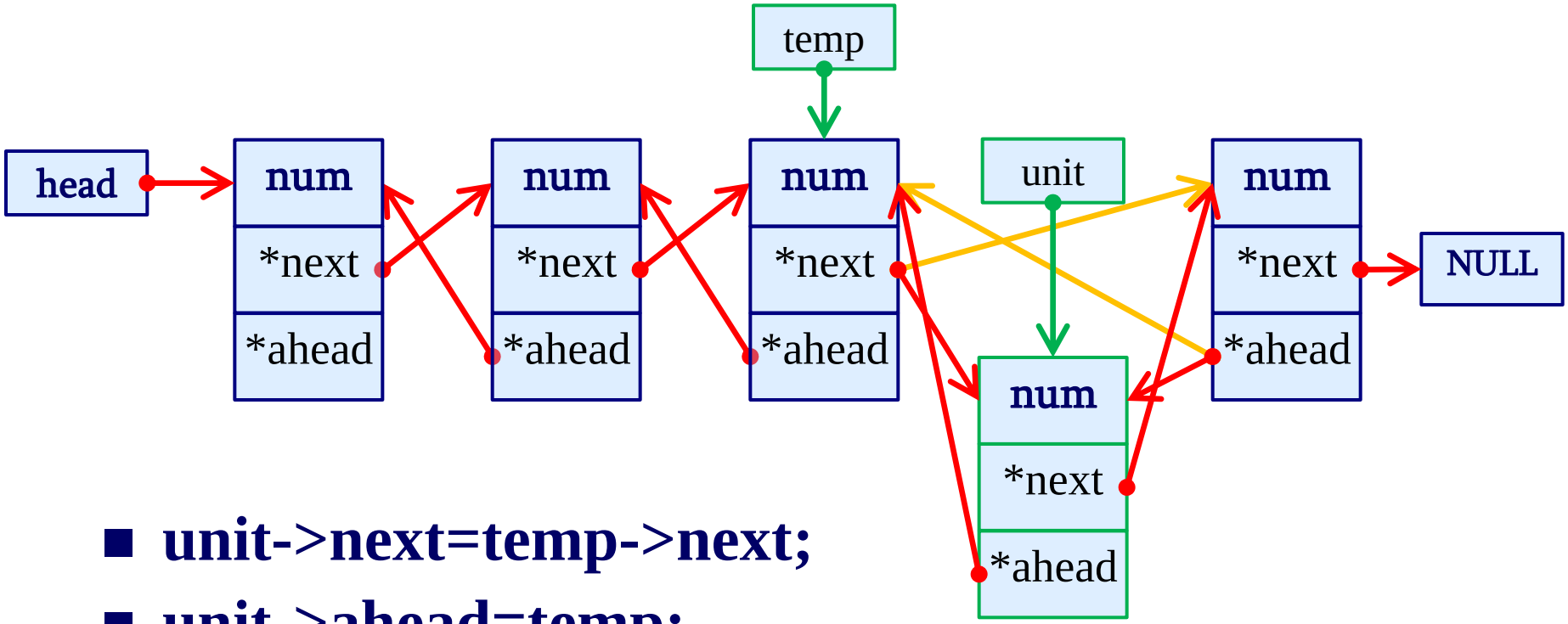
方便向左/右查找节点
如何删除/插入节点？

# 双向链表中的节点删除




- **temp->ahead->next=temp->next;**
- **temp->next->ahead=temp->ahead;**
- **delete temp**

# 双向链表中的节点插入




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

# 内容

- 链表
- **例题**

# 例子：约瑟夫序列

**描述**

n 个小孩围坐成一圈，并按顺时针编号为1,2,…,n，从编号为 p 的小孩顺时针依次报数，由1报到m ，当报到 m 时，该小孩从圈中出去，然后下一个再从1报数，当报到 m 时再出去。如此反复，直至所有的小孩都从圈中出去。请按出去的先后顺序输出小孩的编号(假设小孩的个数不多于300个)。

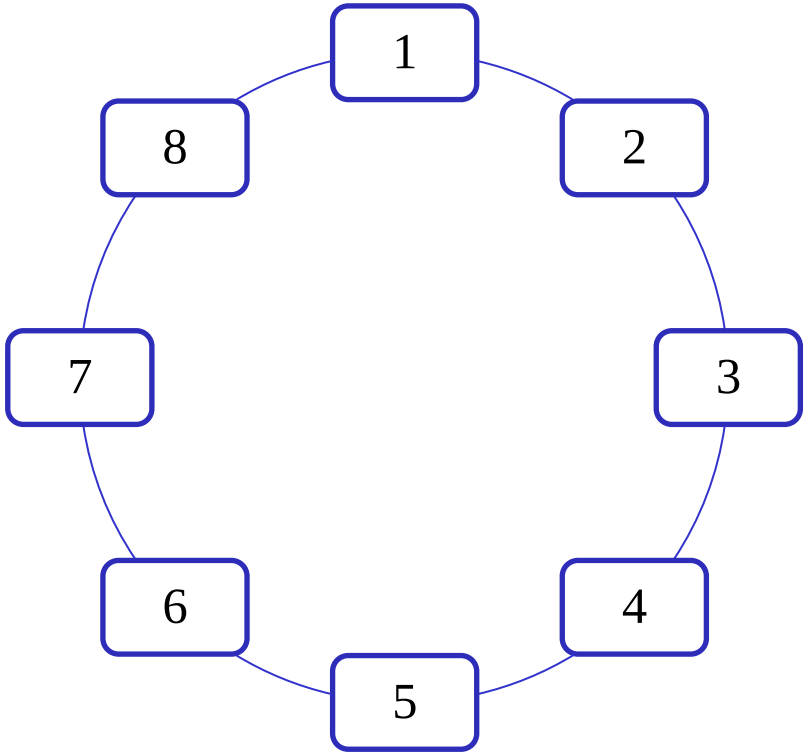
**关于输入**

n,p,m 的值在1行内输入，以空格间隔。

**关于输出**

按出圈的顺序输出编号，编号之间以逗号间隔。

# 双向链表求解约瑟夫序列




```
struct Node
{
    int num;
    Node * ahead;
    Node * next;
};
```

**例子输入**

```
8 3 4
```

n p m

**例子输出**

```
6, 2, 7, 4, 3, 5, 1, 8
```

# 求解约瑟夫序列

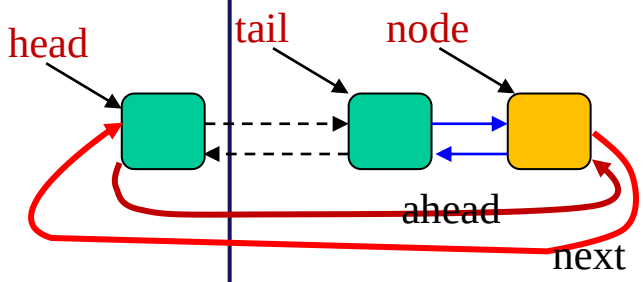
```
struct Node
{
    int num;
    Node * ahead;
    Node * next;
};
```

双向链表（向前，向后）

```
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;
}
```

# 求解约瑟夫序列

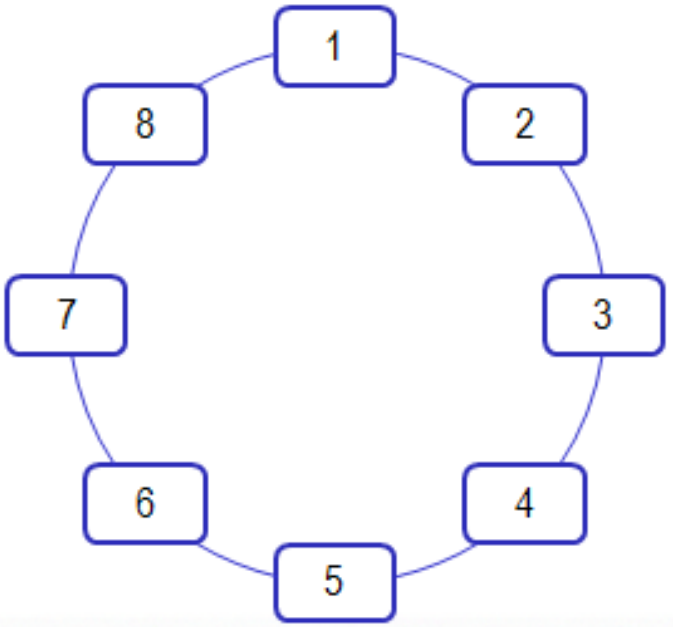
```
Node *Create(int N)  //创建包含N个节点的双向循环链表
{
    int n = 1;
    Node * node = new Node;
    node->num = n;
    Node * head = node;          //指向第一个节点
    Node * tail = head;          //指向最后一个节点
    while( n++ < N)
    {
        node = new Node;         //建新节点
        node->num = n;           //赋值
  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



# 求解约瑟夫序列

```
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;
}
```

从编号为p的小朋友开始

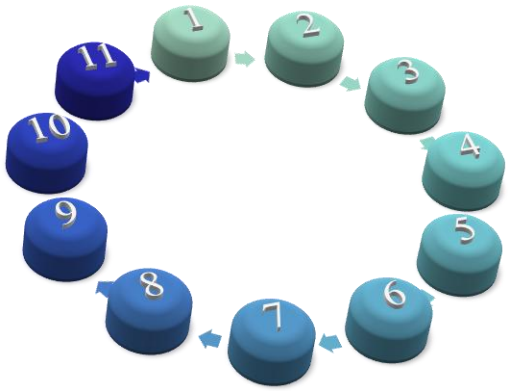


# 求解约瑟夫序列

```
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->ahead->next = temp->next;      //移除第M个节点
    temp->next->ahead = temp->ahead;     //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;
}
```

# 约瑟夫问题

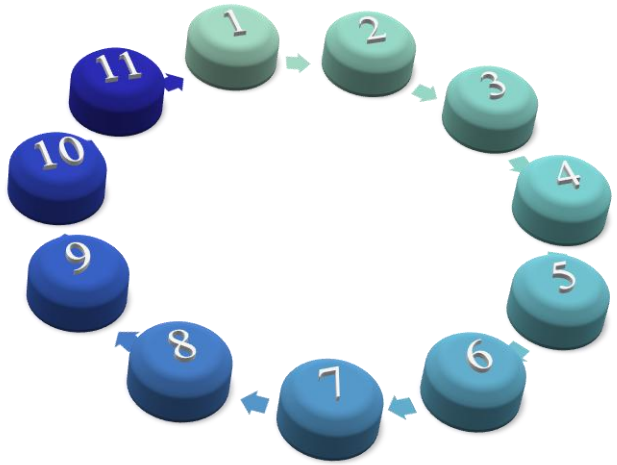
- 问题描述
  - 编号为1 ~ N的N个人围坐在一起形成一个圆圈，从第P个人开始，依次按照顺时针的方向报数，数到第M的人出列，直到最后剩下一个人。
  - 请写一个程序，对于给定的 N, P, M，计算并打印出依次出列的人的编号。



# 约瑟夫问题

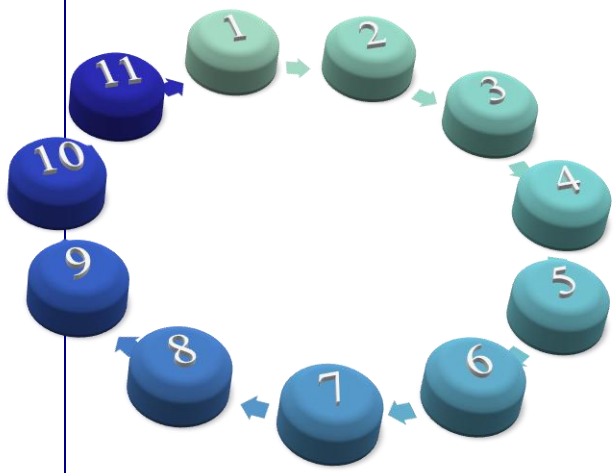
```
#include<iostream>
using namespace std;
struct Node
{
    int num;
    Node * ahead;
    Node * next;
};
Node *Create(int N);
Node *Search(Node *head, int P);
Node *Release(Node *head, int M);
int main()
{
    int N, P, M = 0;              //N-起始节点数，P-开始节点
    cin >> N >> P >> M;           //每次释放第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
```



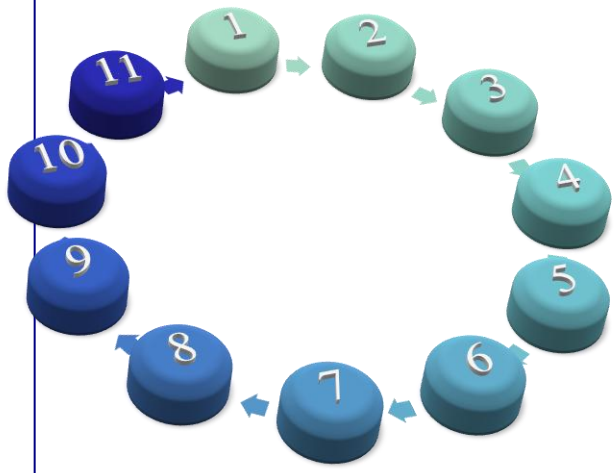
# 约瑟夫问题

```
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->ahead->next = temp->next;     //移除第M个节点
   temp->next->ahead = temp->ahead;    //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;
}
```



# 课堂思考之一

- 输入 n 名学生的成绩，请按按单项链表的形式从大到小方式排序
  - **每名学生含：学号（不超过8位），姓名（不超过10个字符）以及成绩（为整数）。**

# 课堂思考之一

- 按照 边输入，边排序的方式进行

# 课堂思考之二

- 年级大排名
- 题目：已知某个年级有 n 个班，每个班均按从高到低的成绩已经排名，现要对 n 个班按年级统一排名，请按年级排名
  - 每名学生含：学号（不超过8位），姓名（不超过10个字符）以及成绩（为整数）。

# 课堂思考之二

- 设计一个函数，将两个链表归并