《计算概论A》课程 程序设计部分

从结构体到链表

李　戈

北京大学 信息科学技术学院 软件研究所

lige@sei.pku.edu.cn

北京大学

值传递 vs. 地址传递

- 如何实现由小到大的排序？

```
#include<iostream>
using namespace std;
void main( )
{
    int a[2] = {12,5};
    rank(&a[0], &a[1]);
    cout<<"min to max:
    "<<a[0]<<", "<<a[1]<<endl;
}
```

```
void rank(int *a, int *b)
{
    int temp = 0;
    if(*a > *b)
    {
        temp = *a;
        *a = *b;
        *b = temp;
    }
}
```

值传递 vs. 地址传递

- 还能实现由小到大的排序？

```
#include<iostream.h>
int main()
{
    int a[2] = {12,5};
    rank(&a[0], &a[1]);
    cout<<"min to max:
    "<<a[0]<<", "<<a[1]<<endl;
    return 0;
}
```

```
void rank(int *a, int *b)
{
    int *temp = NULL;
    if(*a > *b)
    {
        temp = a;
        a = b;
        b = temp;
    }
}
```

指针与结构体

数据与程序

- 什么是程序？
  - 程序是 计算任务的处理对象和处理规则的描述。
    - 处理对象 是指 数据
  - 数据（处理对象）是程序的一个核心部分；
- 关于数据
  - 如何组织数据（处理对象）对程序而已至关重要；
  - 对应现实世界中的组织方式；

再说结构体

- 声明一个名为“学生”的结构体

```
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
 };          \\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“;”
```

北京大学

结构体直接做函数参数

```
#include<iostream>
using namespace std;
struct stru
{  int x;
   char c;
};
void func(stru b)
{  b.x=20; b.c='y';}
int main()
{  stru a = {10,'x'};
   func(a);
   cout<<a.x<<" "<<a.c;
   system("pause");
   return 0;
}
```

结构体直接做函数参数

```
#include<iostream>
using namespace std;
struct stru
{  int x;
   char c;
};
void func(stru b)
{  b.x=20; b.c='y';}
int main()
{  stru a = {10,'x'};
   func(a);
   cout<<a.x<<" "<<a.c;
   system("pause");
   return 0;
}
```

- 结构体做参数时采用值传递的方式；
- 系统会构造一个结构体的副本给函数使用；

```
10 x
Press any key
```

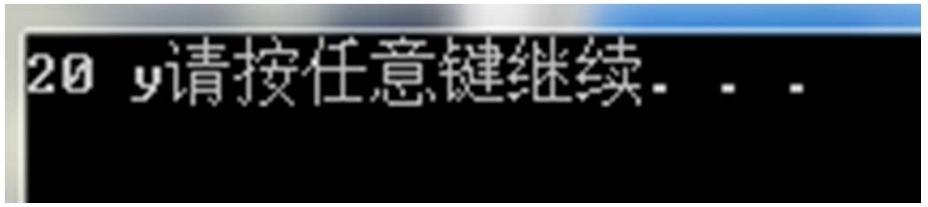
结构体直接做函数返回值

```
#include<iostream>
using namespace std;
struct stru
{  int x;
   char c;
};
stru func(stru b)
{  b.x=20; b.c='y';
   return b;
}
int main()
{  stru a = {10,'x'};
   a = func(a);
   cout<<a.x<<" "<<a.c;
   system("pause");
   return 0;
}
```

结构体直接做函数返回值

```
#include<iostream>
using namespace std;
struct stru
{  int x;
   char c;
};
stru func(stru b)
{  b.x=20; b.c='y';
   return b;
}
int main()
{  stru a = {10,'x'};
   a = func(a);
   cout<<a.x<<" "<<a.c;
   system("pause");
   return 0;
}
```

- 结构体做返回值时也采用值传递的方式；
- 系统会构造一个结构体的副本返回给调用者；

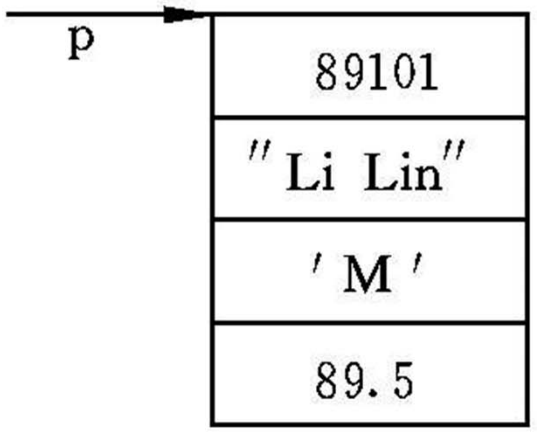


指向结构体类型数据的指针

- 因为
  - 结构体类型与其他数据类型相同；
  - 一个结构体变量在内存中占用一段连续的区域，有一个起始地址；
- 所以
  - 可以设计一个指针变量，用于存放结构体变量的起始地址；
  - 即，指向结构体类型数据的指针；

指向结构体的指针

```
main()
{
   struct student * p;
   p = &stu;
   stu.num=89101;
   strcpy(stu.name, "Li Lin");
   stu.sex='M'; stu.score=89.5;
   cout<<stu.num<<stu.name
                        <<stu.sex<<stu.score);
   cout<<(*p).num<<(*p).name
                        <<(*p).sex<<(*p).score;
}
```



指向结构体的指针

```
#include<iostream>
using namespace std;
int main()
{   student stu,*p;
    int *p1;
    char *p2, *p3;
    float *p4;
    p = &stu;
    p1 = &stu.stuNo;  p2 = stu.name;
    p3 = &stu.sex;  p4 = &stu.score;
    p->stuNo = 89101;
    strcpy(p->name, "Li Lin");
    p->sex = 'M';  p->score = 90;
    cout<<stu.stuNo<<" "<<stu.name
                    <<" "<<stu.sex<<" "<<stu.score<<endl;
    cout<<*p1<<" "<<p2
                    <<" "<<*p3<<" "<<*p4<<endl;
    system("pause");
    return 0;
}
```

```
struct student
{
    int stuNo;
    char name[20];
    char sex;
    float score;
};
```

指向结构体的指针

```
#include<iostream>
using namespace std;
int main()
{   student stu,*p;
    int *p1;
    char *p2, *p3;
    float *p4;
    p = &stu;
    p1 = &stu.stuNo;  p2 = stu.name;
    p3 = &stu.sex;  p4 = &stu.score;
    p->stuNo = 89101;
    strcpy(p->name, "Li Lin");
    p->sex = 'M';  p->score = 90;
    cout<<stu.stuNo<<" "<<stu.name
                    <<" "<<stu.sex<<" "<<stu.score<<endl;
    cout<<*p1<<" "<<p2
                    <<" "<<*p3<<" "<<*p4<<endl;
    system("pause");
    return 0;
}
```

```
89101 Li Lin M 90
89101 Li Lin M 90
请按任意键继续. . .
```

指向结构体的指针做参数

例：
在主函数中输入结构体各成员的值, 在子函数中输出；

```
struct student
{   int No;
    char name[20];
    float score[3];
};
void print(student *p){
    cout<<p->No<<p->name<<p->score[0]
        <<p->score[1]<<p->score[2];}
int main(){
    struct student stu;
    cin >>stu.No>>stu.name>>stu.score[0]
        >>stu.score[1]>>stu.score[2];
    print(&stu);
    return 0;
}
```

结构体数组做参数

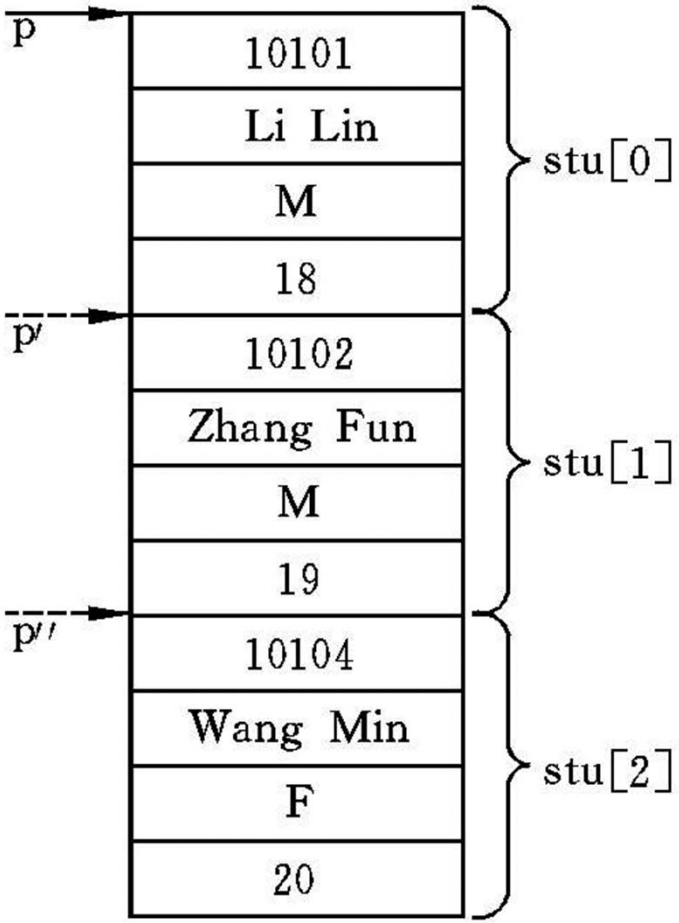
```
void main()
{ student allone[4]=
   {
        {1001,“jone”,60,60,80},
        {1002,“david”,70,70,90},
        {1003,“marit”,80,80,60},
        {1004,“yoke”,90,90,70}
   };
   print(allone, 4);
}
```

```
void print(student *p, int n)
{
    for (int i=0; i<n; i++, p++)
        cout<< p->No
            << p-> name
            << p-> score[0]
            << p-> score[1]
            << p-> score[2]
            << endl;
}
```

北京大学

指向结构体数组的指针

- 解释
  - p用来指向一个struct student型的数据。
  - p加1意味着p所增加的值为结构体数组stu的一个元素所占的字节数；
  - 本例为：4+20+1+4=29字节



指向结构体数组的指针

```
struct test *p,  stu[4];
p = stu;
cout << p++->num << endl;
cout << ++p->num << endl;
cout << p->num++ << endl;
cout << p->num << endl;
cout << (++p)->num++<<endl;
cout << p->num << endl;
```

num	name
10	‘A’
20	‘B’
30	‘C’
40	‘D’

- 注意：->运算的优先级非常高，仅次于（）[ ]

指向结构体数组的指针

```
struct test *p,  stu[4];
p = stu;
cout << p++->num << endl;
cout << ++p->num << endl;
cout << p->num++ << endl;
cout << p->num << endl;
cout << (++p)->num++<<endl;
cout << p->num << endl;
```

```
10
21
21
22
30
31
Press
```

num	name
10	‘A’
20	‘B’
30	‘C’
40	‘D’

- 注意：->运算的优先级非常高，仅次于（）[ ]

结构体应用示例-生日相同问题

- **Description**
  - 在一个有100人的大班级中，存在两个人生日相同的概率非常大，现给出每个学生的学号，出生月日。试找出所有生日相同的学生。
- **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<=m<=12)日(1<=d<=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。

```
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)
                {       cout<<m<<" "<<d<<" ";
                        for(k=1; k<=j;k ++)
                                cout << stu[count[k]].ID << " "<< endl;
                }
        }
}
```

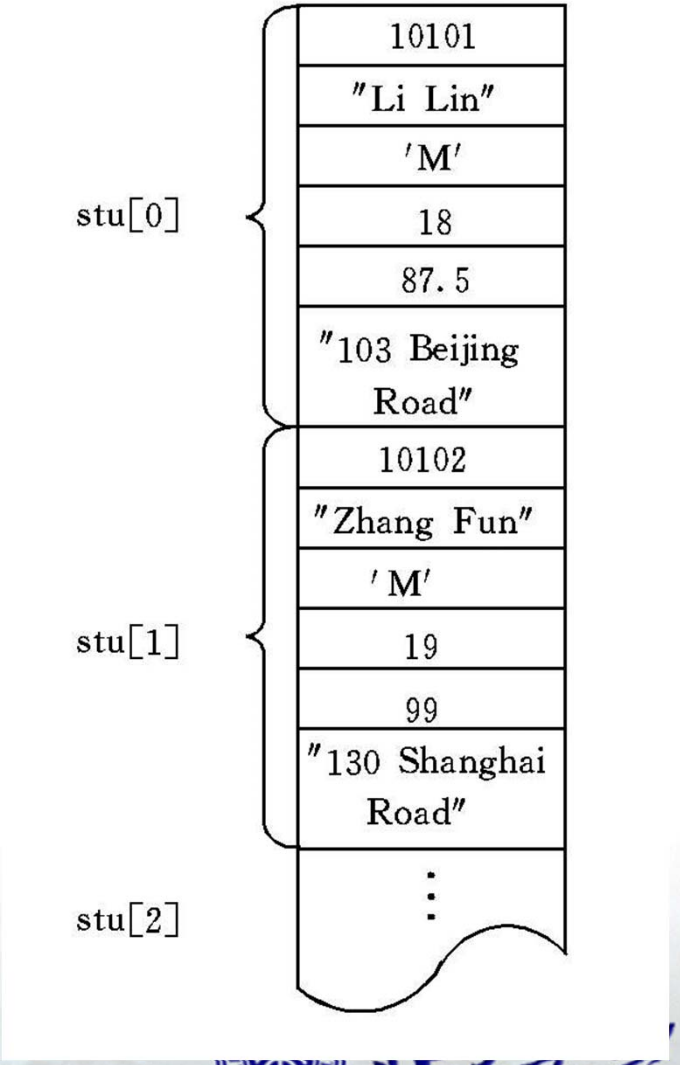
```
struct student
    {
        char ID[10];
        int month;
        int day;
    }stu[100];
```

思考一个问题

- 问题：
  - 已知在教务系统中，需要经常根据学生的学号顺序地读取学生的相关信息；
- 要求：
  - 定义一个结构，按照学生的学号，顺序存放班级学生的信息；
  - 学生信息包括：学号、姓名、性别、年龄、出生日期、地址 等；

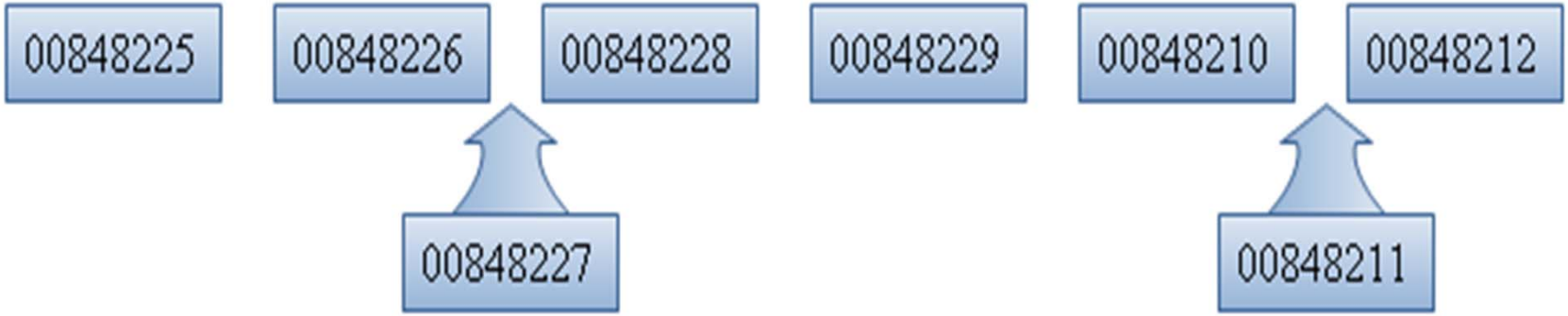
结构体数组

```
struct   date
{ int   month;
  int   day;
  int   year; };
struct   student
{ int   num;
  char   name[20];
  char   sex;
  int   age;
  struct   date birthday;
  char   addr[30];
}students[1000];
```



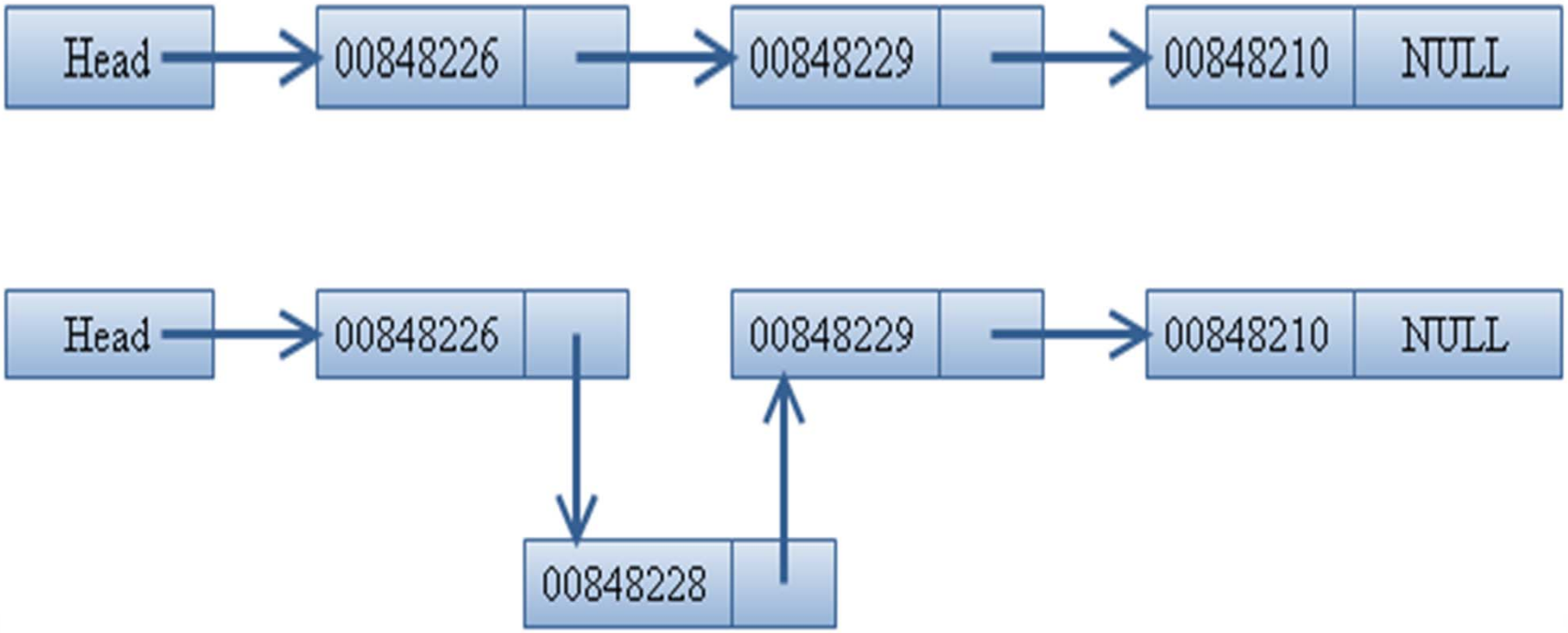
使用数组的缺点

- 插入元素
  - 每次插入元素时，需要将后面元素统统后移！



可以采取的办法

- 用指针把结构体链起来！



链 表

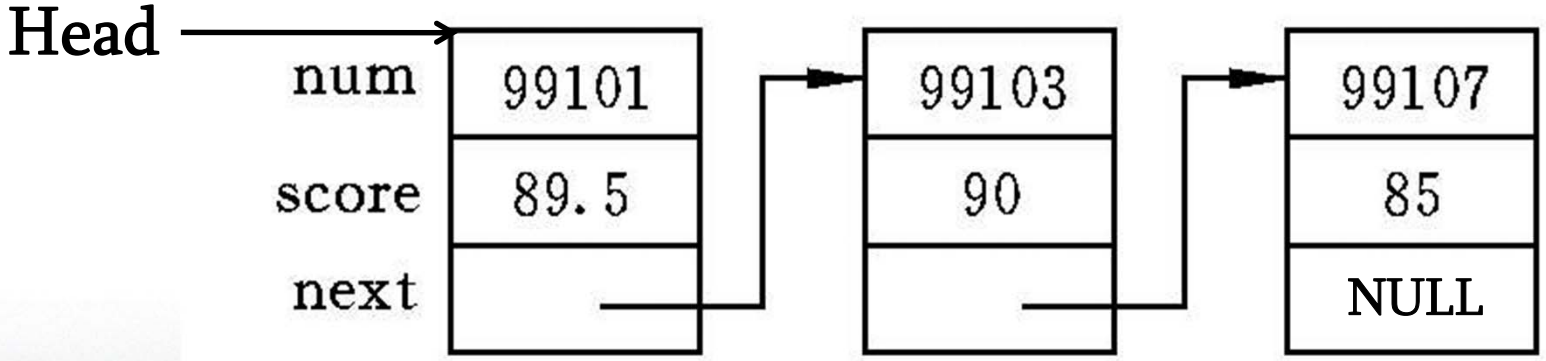
链表

- 链表是一种常用的重要的数据结构
  - 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。

链表的定义

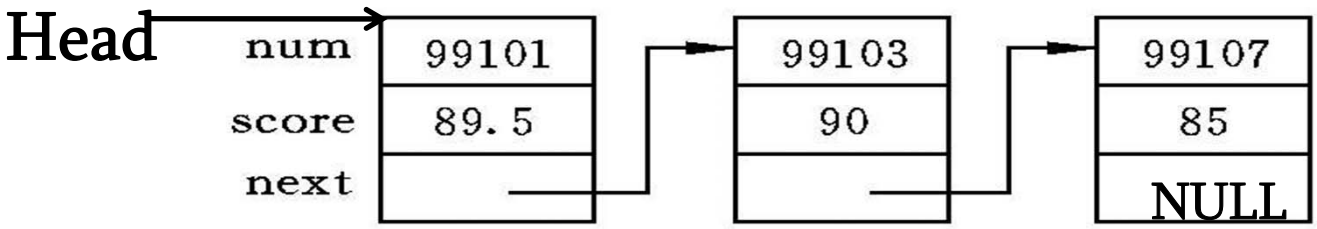
```
struct student
{
    long num;
    float score;
    struct student *next;
};
```

student a,b,c,*head,*p;



建立一个简单的链表

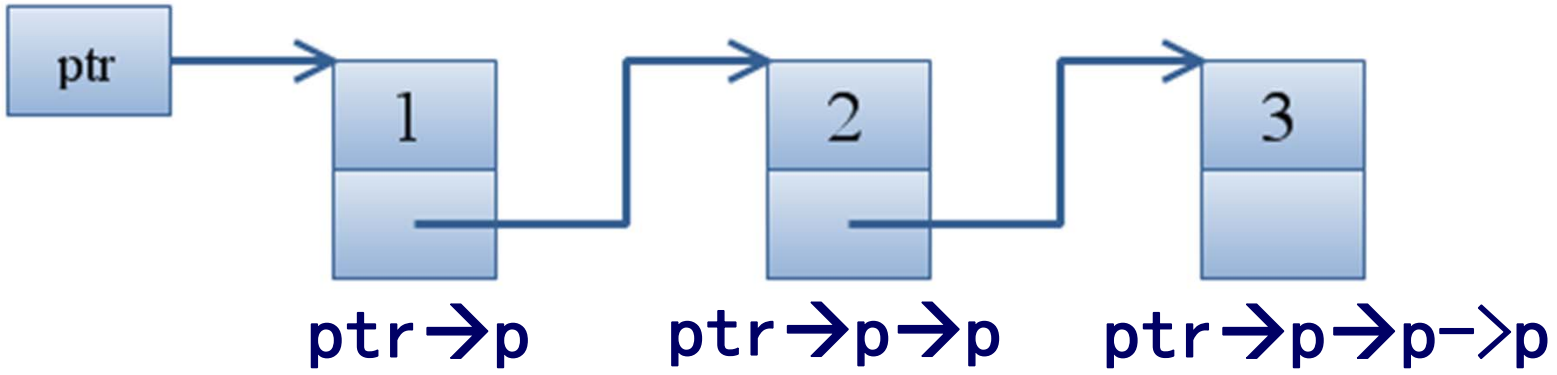
```
#include<iostream>
using namespace std;
int main() {
    student a,b,c,*head,*p;
    a.num = 99101;
    a.score = 89.5;
    b.num = 99103;
    b.score = 90;
    c.num = 99107;
    c.score = 85;
    head = &a;          /*将结点a的起始地址赋给头指针head*/
    a.next = &b;        /*将结点b的起始地址赋给a结点的next成员*/
    b.next = &c;        /*将结点c的起始地址赋给b结点的next成员*/
    c.next = NULL;      /*c结点的next成员不存放其他结点地址*/
}
```



访问链表中的元素

```
ptr = new linker ;
ptr->p = new linker ;
ptr->p->p = new linker ;
```

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



- 要想访问第4个元素，必须表示成：

  head->p->p->p->num

链表可以 动态地 创建

- 动态地 建立数组
  - **int *p = new int[20];**
- 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

- 定义一个临时指针指向链表的最后一个节点：

  student *temp = head;

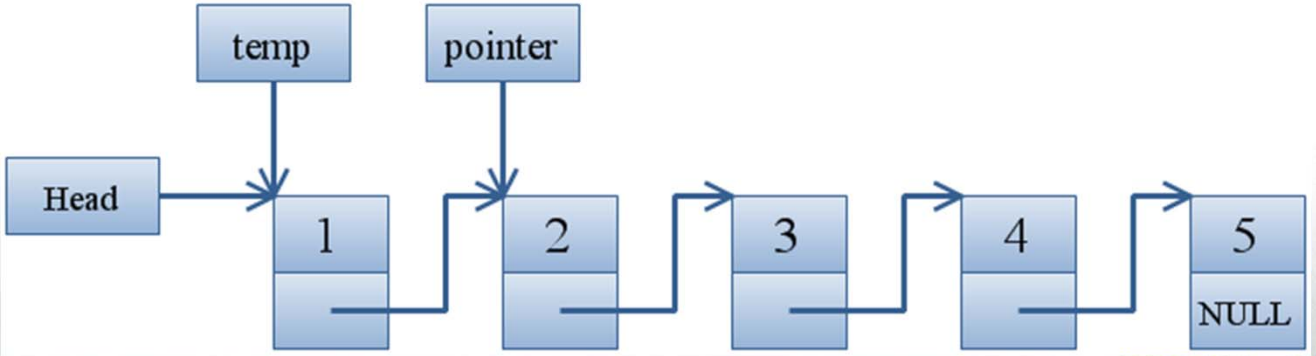
- 申请一个新的链表节点：

  pointer = new student

  temp->next = pointer;

- 指针后移一个链表节点：

  temp = temp->next



链表结构——逐步建立链表

步骤：

① 申请第一个单元：**head = new student;**

② 建立临时指针：**temp = head;**

③ 输入数据，如果不为结束标志，存入链表;

④ 申请新的单元：**temp->next = new student ;**

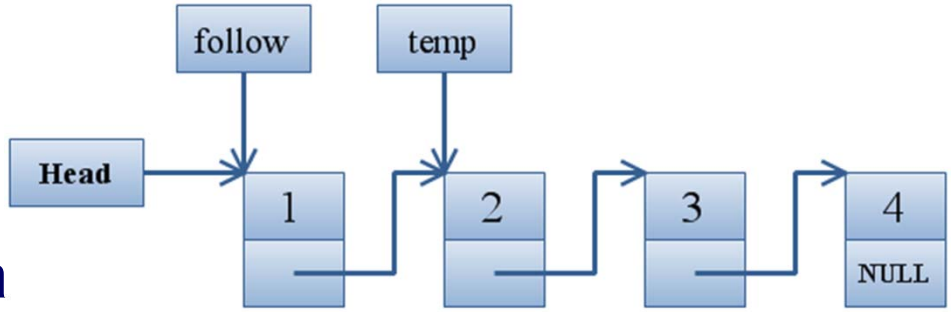
⑤ 移动临时指针：**temp = temp->next;**

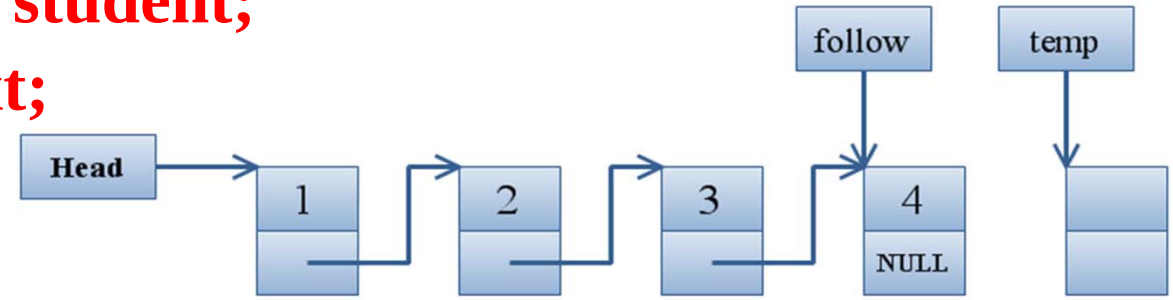
⑥ 回到第③步，直到输入结束标志转到⑦ ；

⑦ 不再申请单元，将当前单元的指针赋为**null;**

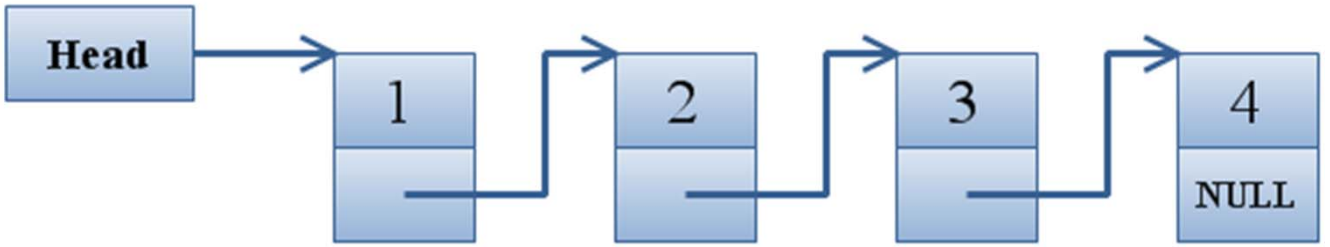
北京大学

```
student *create()
{   student *follow,*temp,*head; int num, n;
    head = new student;
    temp = head;       n=0;
    cin >> num;
    while(num != -1)
    {      n = n + 1;
           temp->num = num
           follow = temp;
           temp->next = new student;
           temp = temp->next;
           cin>>num;
    }
    if (n==0) head=NULL; else follow->next = NULL;
    delete temp;  return(head);
}
```





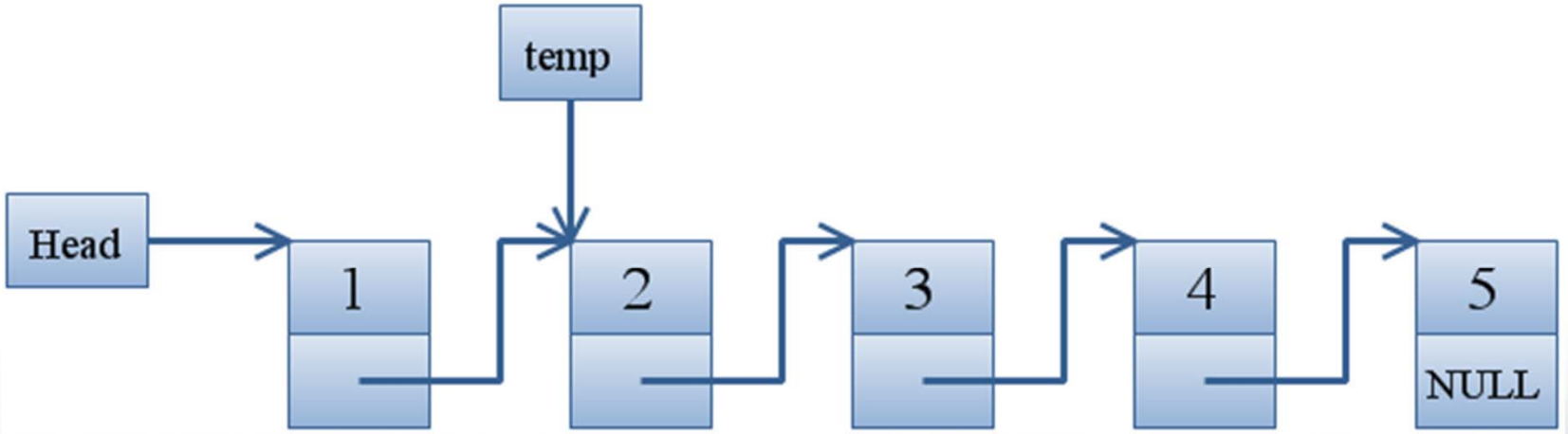
```
struct student
{
   int num;
   struct student *next;
};
student *create();      //返回指针的函数
void showList(student*);
void main( )
{
    student *head;
   head = create();      showList(head);
}
```



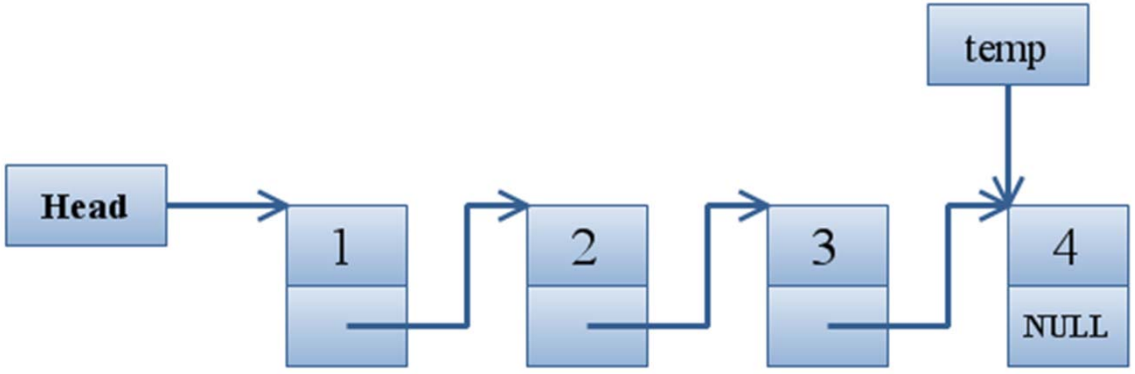
链表元素的遍历

- 利用temp指针遍历链表

```
temp = head;
do {        cout<<temp→num;
            temp = temp→p;
   } while (temp != null);
```



```
void showList(student *head)
{   student *temp;
   if (head==NULL) cout<<“This is a null linker\n”;
   else
   {   temp=head;
       do{ cout<<temp->num;
            temp=temp->next;          //指针temp向后走，
          } while(temp!=NULL);    //直到遇到结束标志
   }
   cout<<endl;
}
```



链表结构——结点检索

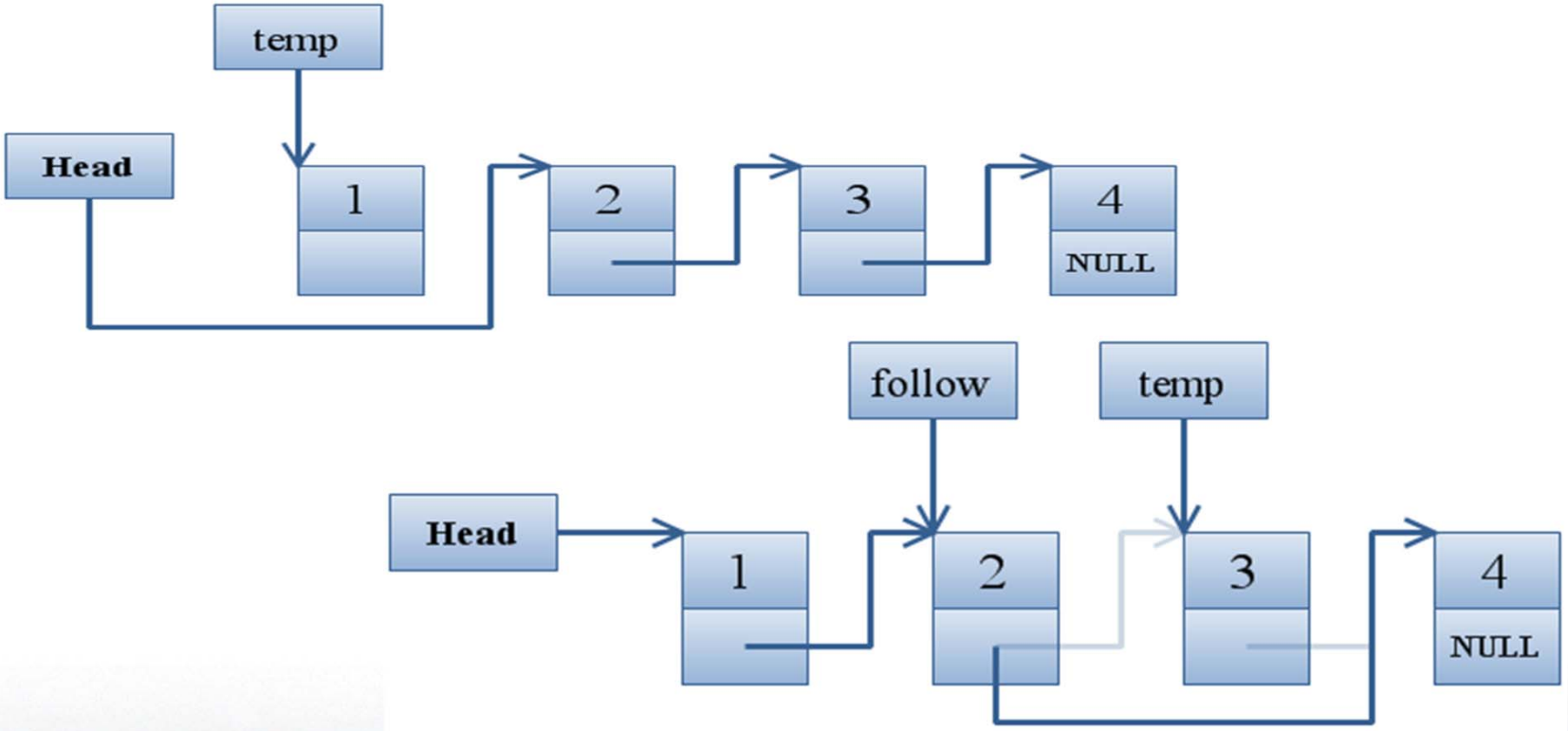
■ 在链表中找出n的位置，并输出。

```
cin >> n;
temp = head;
while (temp != null && temp->num != n)
{  temp = temp->next  };
                        //只要没有和n相同的数  并且
                        //还没有找到链表的末尾，则继续找
if (temp!=null) cout<<temp->num;
else cout<<“not found”;
```

链表结构——删除结点

在链表中将值为n的元素删掉

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



follow→next = temp→next; delete temp;

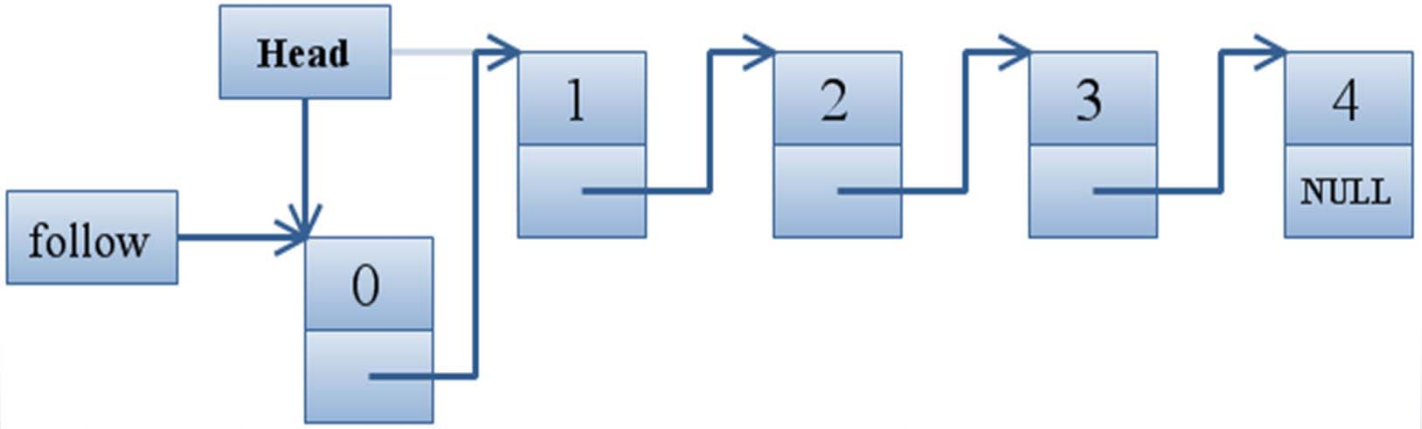
```
linker *dele(student *head; int n)
{   student *temp,* follow;
    temp = head;
    if (head == NULL) {              //head为空，空表的情况
        return(head);
    }
    if (head->num = = n) {           //第一个节点是要删除的目标；
        head = head->next;
        delete temp;
        return(head);
    }
    while(temp != 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;
```

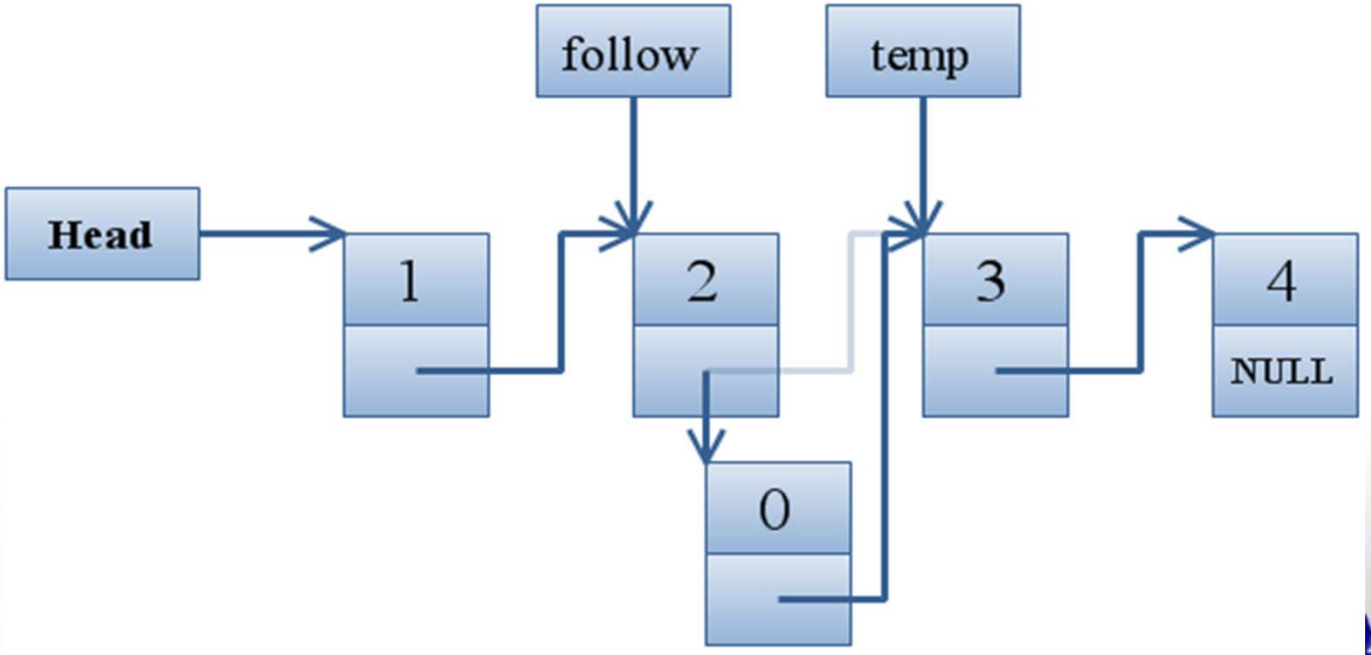


链表结构——插入结点

- 将结点**unit**插入链表：

插在中间的情况

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



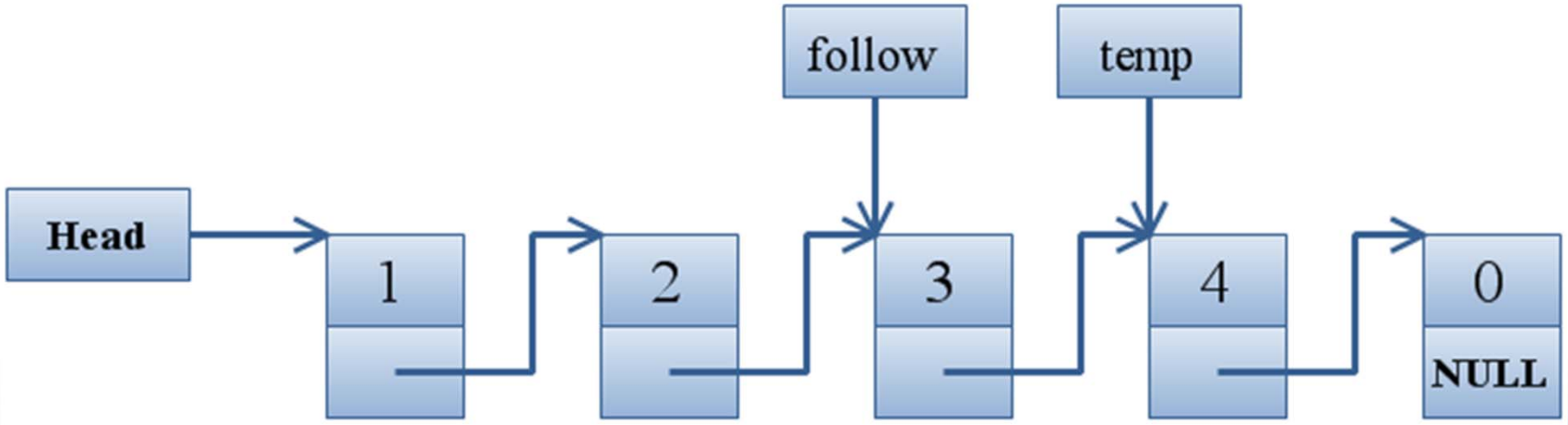
链表结构——插入结点

- 将结点**unit**插入链表：

插在最后面的情况

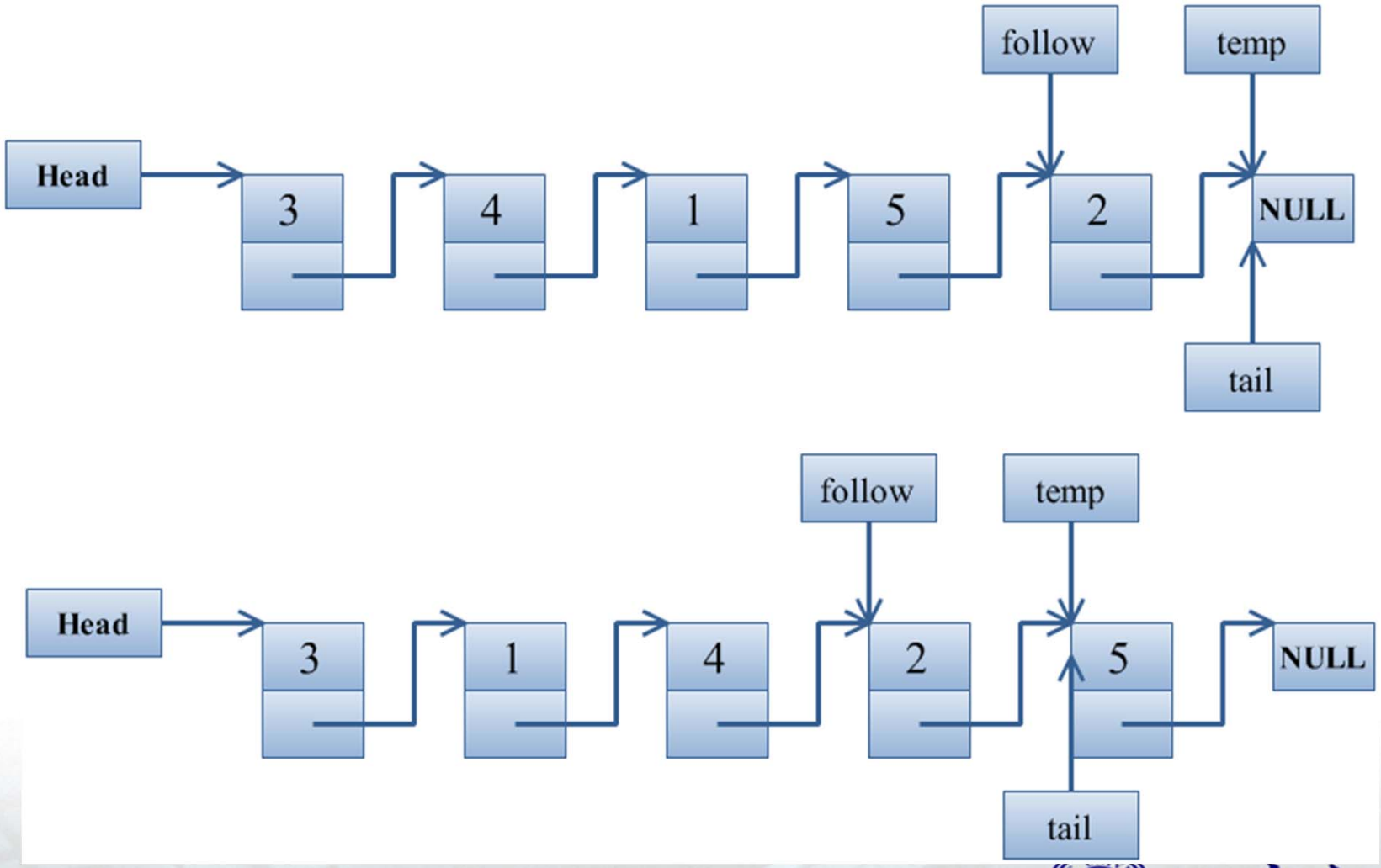
temp->next = unit;

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);
}
```

链表结构——结点排序


follow
temp
Head
3
4
1
5
2
NULL
tail
follow
temp
Head
3
1
4
2
5
NULL
tail

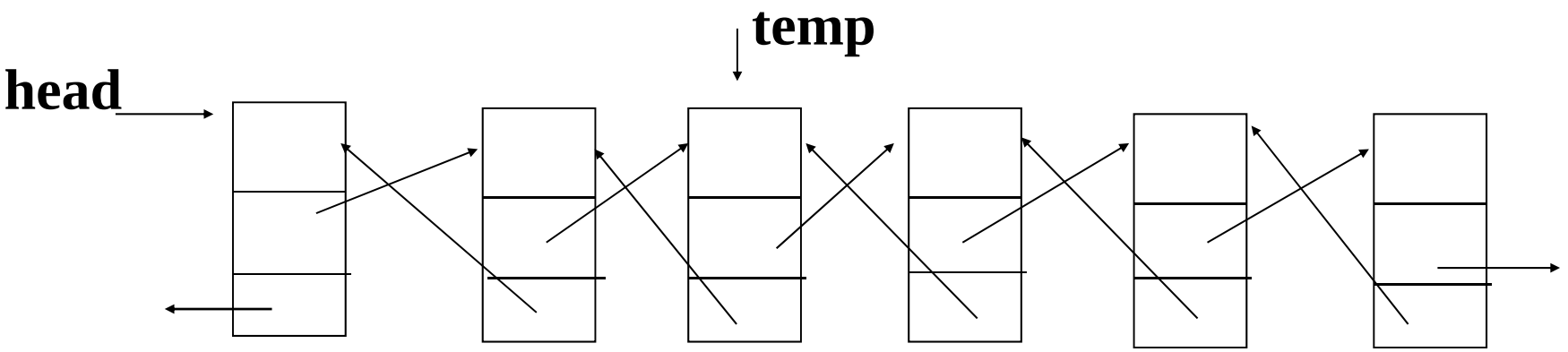
```
void sort (student *head) {  //将链表排序
   student *temp, *follow, *tail = NULL;    int t;
   while(head->next != tail) {
        follow= head;  temp = follow→next;
        while(temp!= tail) {
            if (follow→num > temp→num) {
                t = follow→num;
                follow→num = temp→num;
                temp→num=t;
            };
            follow = temp;
            temp = temp->next;
        }
        tail = follow;
   }
}
```

双向链表

- 单向链表的缺陷
  - 不能通过当前位置找到前面的元素；
- 双向链表
  - 有两个链，一个指向前面的元素，一个指向后面的元素；
  - 双向链表结点的定义

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

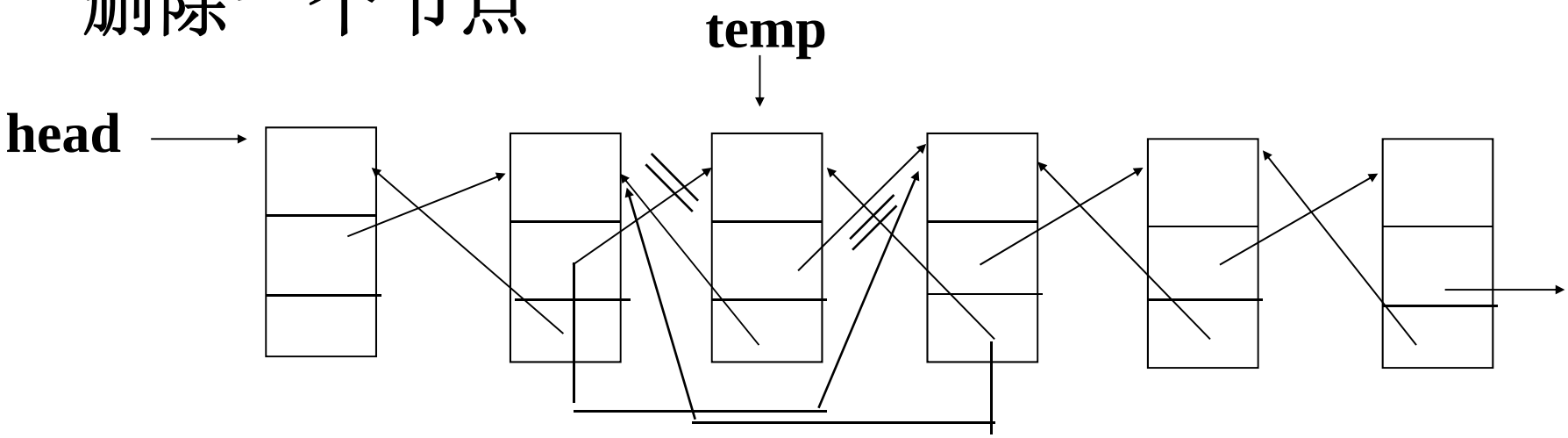
双向链表



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

双向链表——删除结点

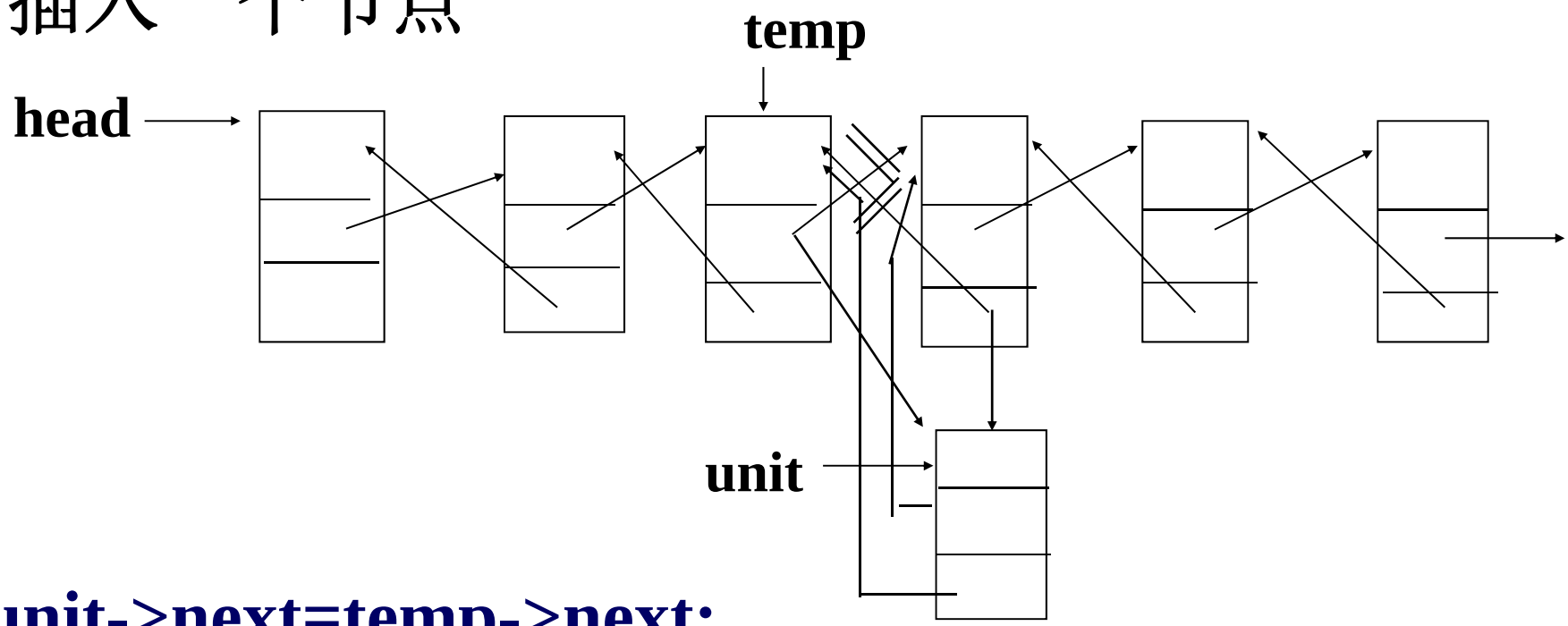
删除一个节点



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

双向链表——插入结点

插入一个节点



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

好好想想，有没有问题?

谢　谢　！

北京大学