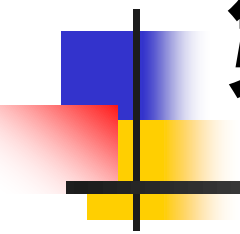
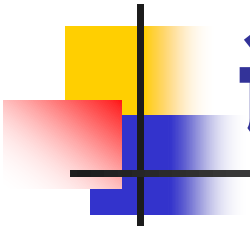


第11讲 链表简单介绍

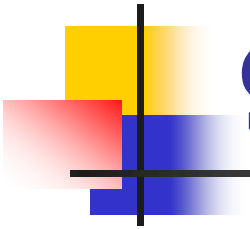


讲授内容

- 自引用结构、链表的概念
- 内存的动态分配和释放
- 单向链表的定义与操作

9.1 链表的基本概念

- 结构数组--必须将数组的大小设定成足够大的值
  - 太浪费
  - 能否需要多少分配多少?
- 链表 = 动态内存分配 + 结构 + 指针
  - 所有结构形成一条链
  - 可以在任何地方插入或删除元素



9.2 单向链表

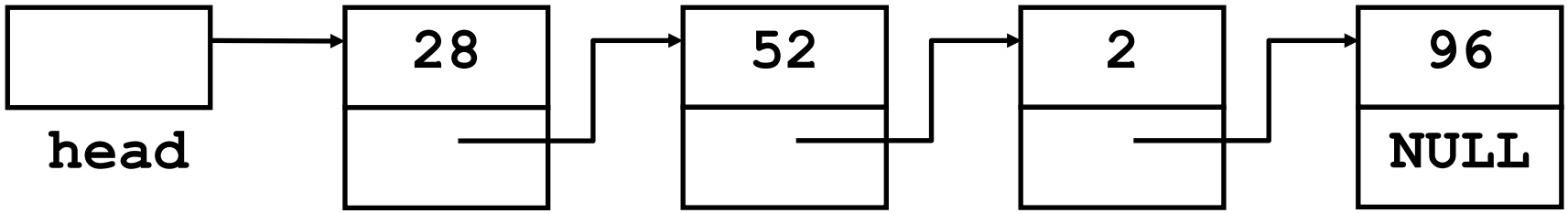
- 自引用结构
  - 结构中包含指向同类型结构的指针
  - 通过指针连接成链表，终点是NULL指针(0)

9.2.1 单向链表定义

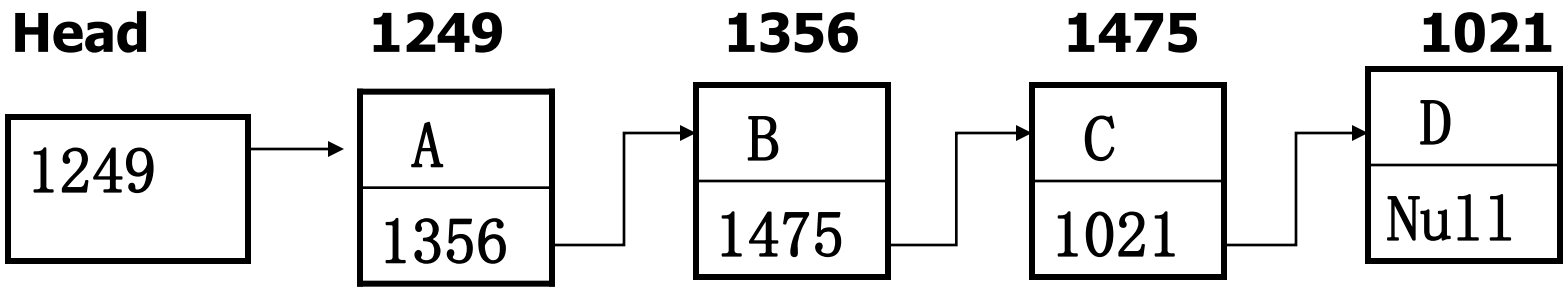
- 例子：

```
class node
 {
    int data;
    node * next;
 };
```

  - next：指向下一个node类型的结构，连接node的纽带



结点里的指针是存放下一个结点的地址



1、链表中的元素称为“结点”，每个结点包括两个域：数据域和指针域；

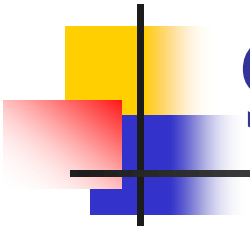
2、单向链表通常由一个头指针（head），用于指向链表头；

3、单向链表有一个尾结点，该结点的指针部分指向一个空结点(NULL）。

```
#include <iostream>
using namespace std;
class student
{  long num;
    float score;
    student  *next;
};
 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;     a.next=&b;     b.next=&c;      c.next=NULL;
    p=head;
  do
 {  cout<<p->num<<"    "<<p->score<<endl;
     p=p->next;
 }while(p!=NULL);
return 0;
}
```

各结点在程序中定义，不是临时开辟的，始终占有内容不放，这种链表称为“静态链表”

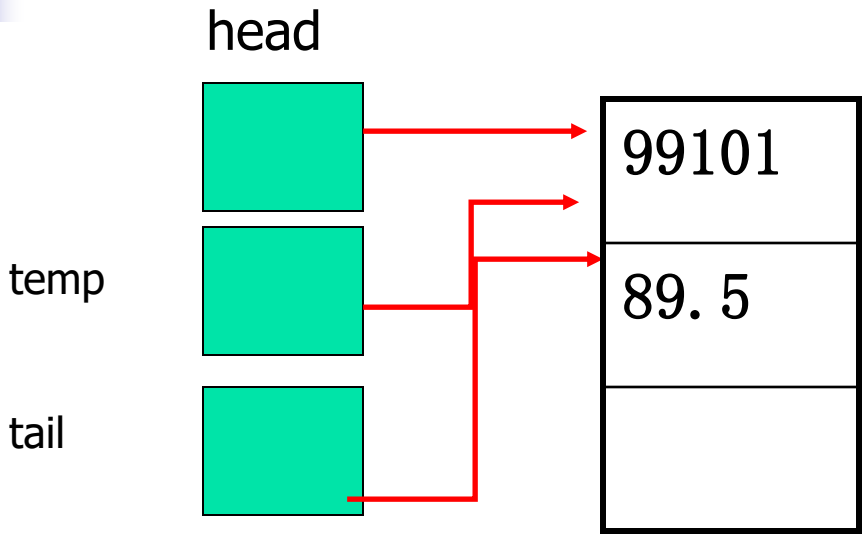
建立和输出一个简单链表



9.2.2 单向链表的操作

- 建立单向链表
  - 声明一个链首指针变量head，并赋初值NULL(包含0个节点的链表)
  - 动态分配一个新节点，将该节点链入链尾
  - 重复上一步

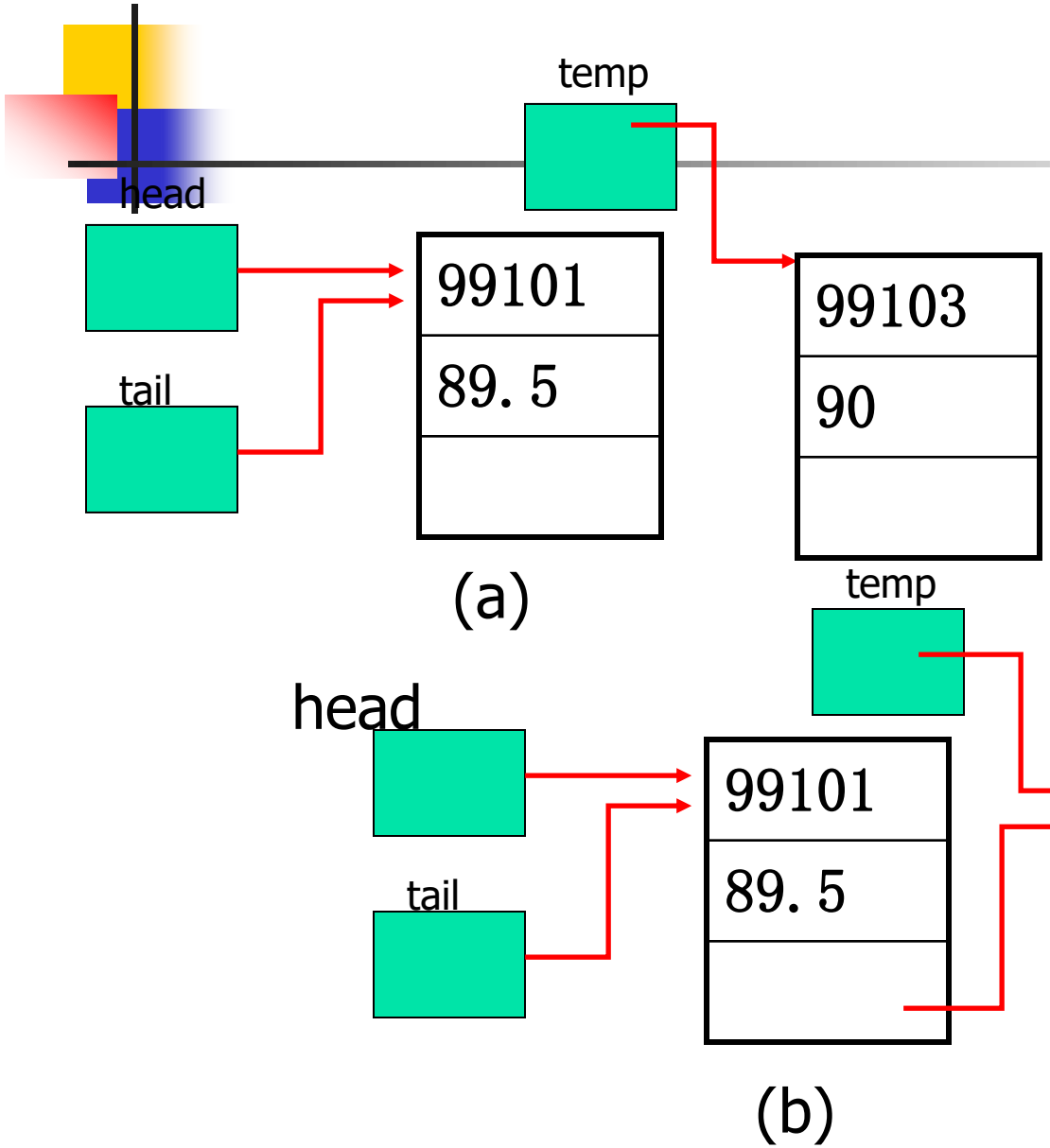
建立动态链表



待插入的结点temp数据部分初始化,该结点被头结点**head**、尾结点**tail**同时指向.

1.任务是开辟结点和输入数据

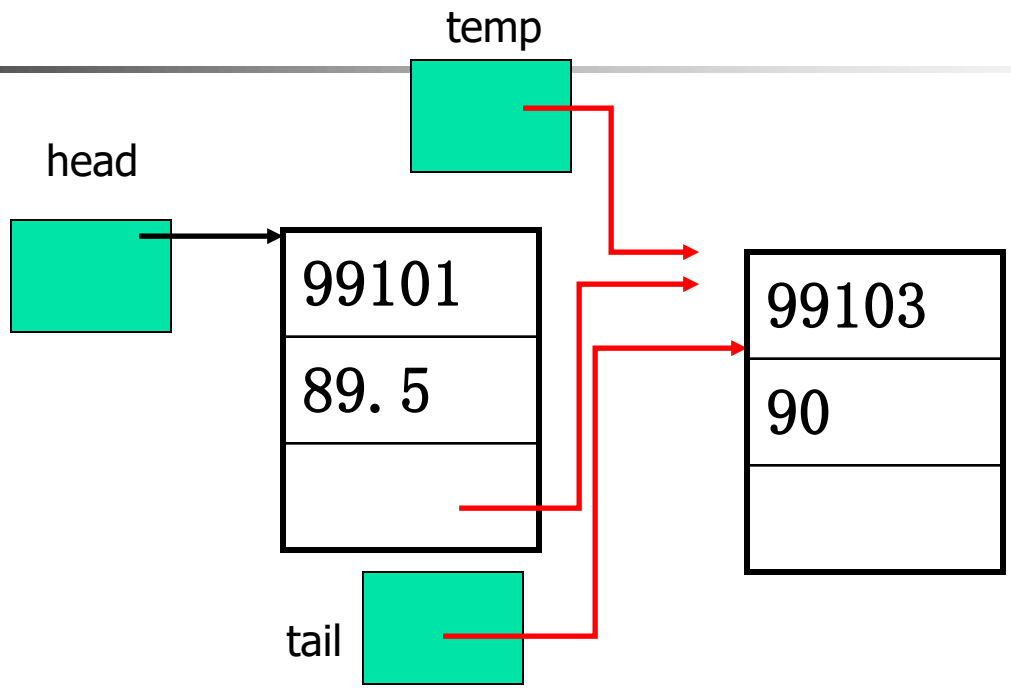
2.并建立前后相链的关系



temp重复申请待插入结点空间，对该结点的数据部分赋值（或输入值）

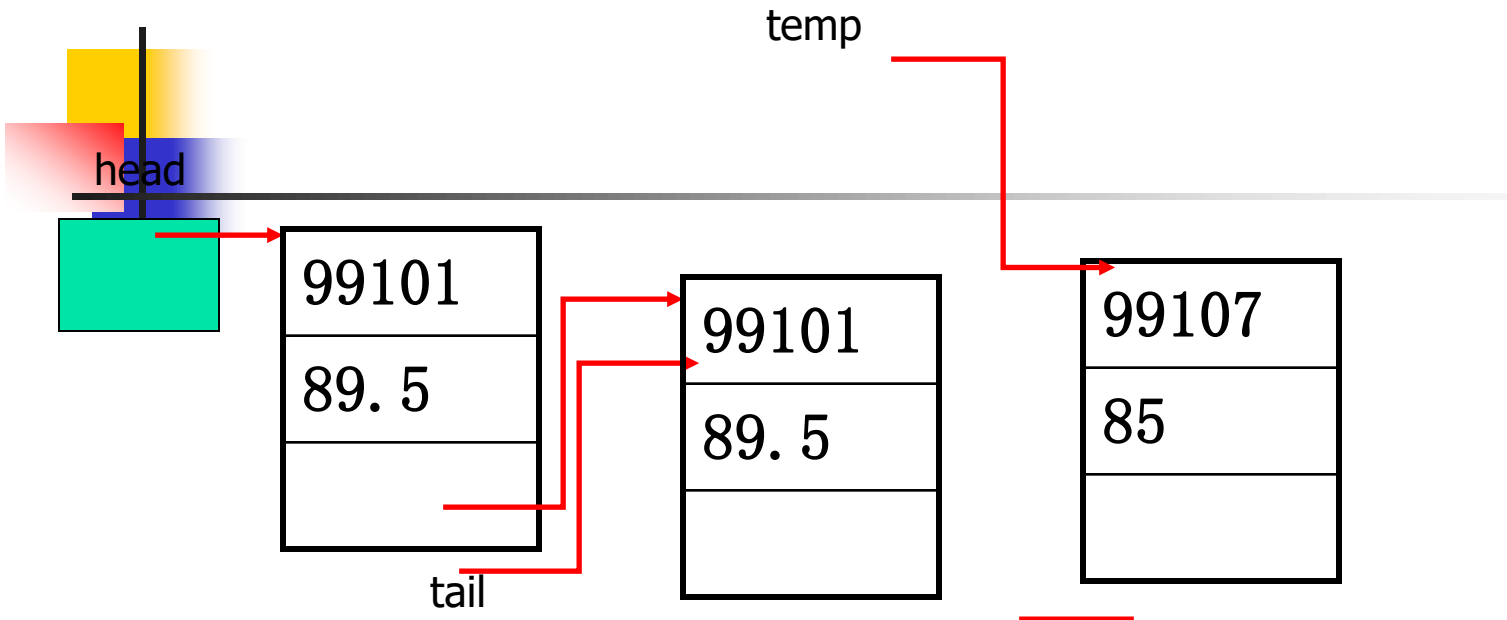
tail->next 指向temp新开辟的结点。

图11.14

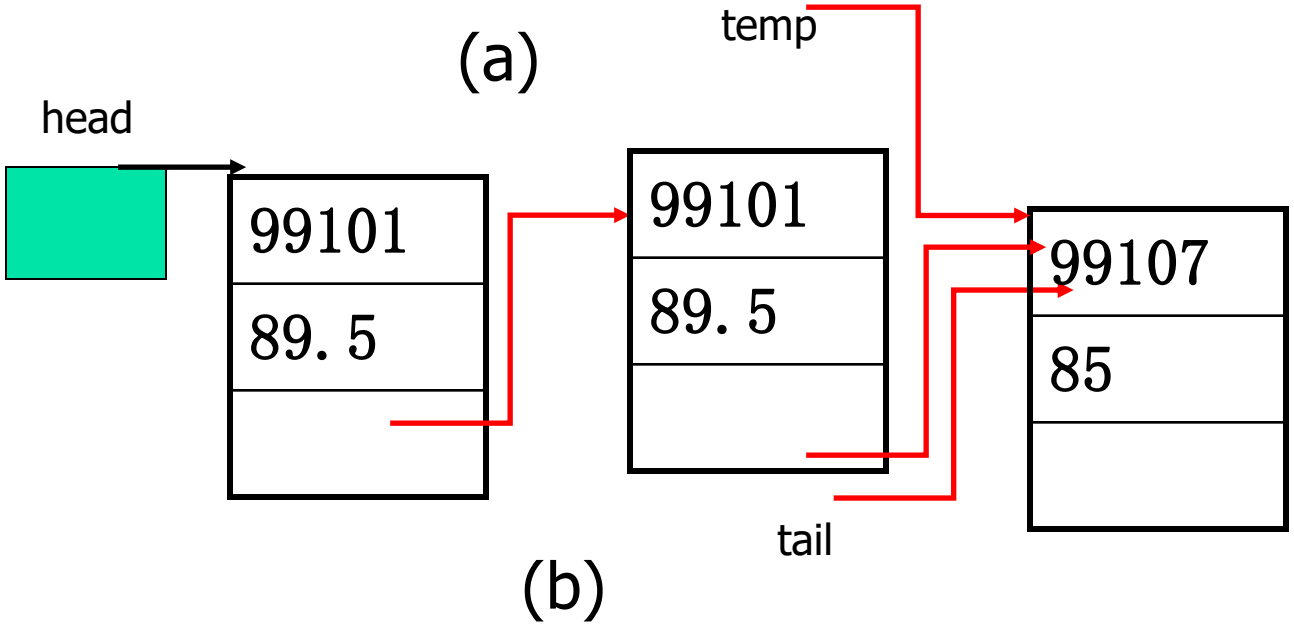


(c)

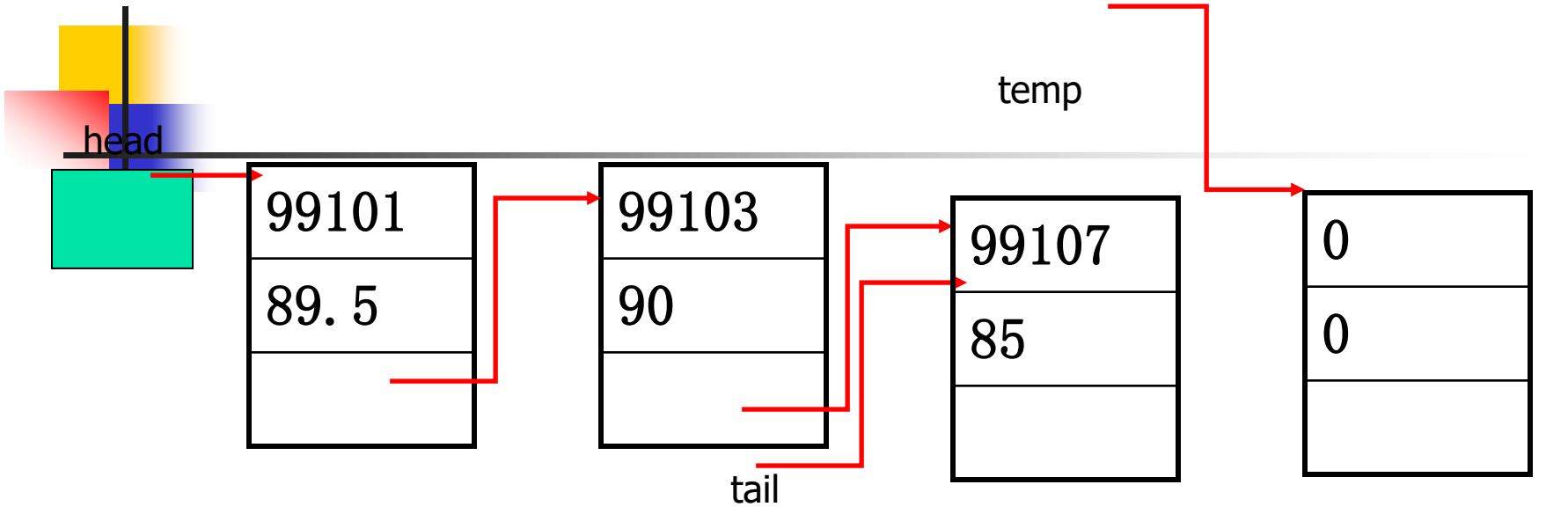
tail指向新结点tail=temp

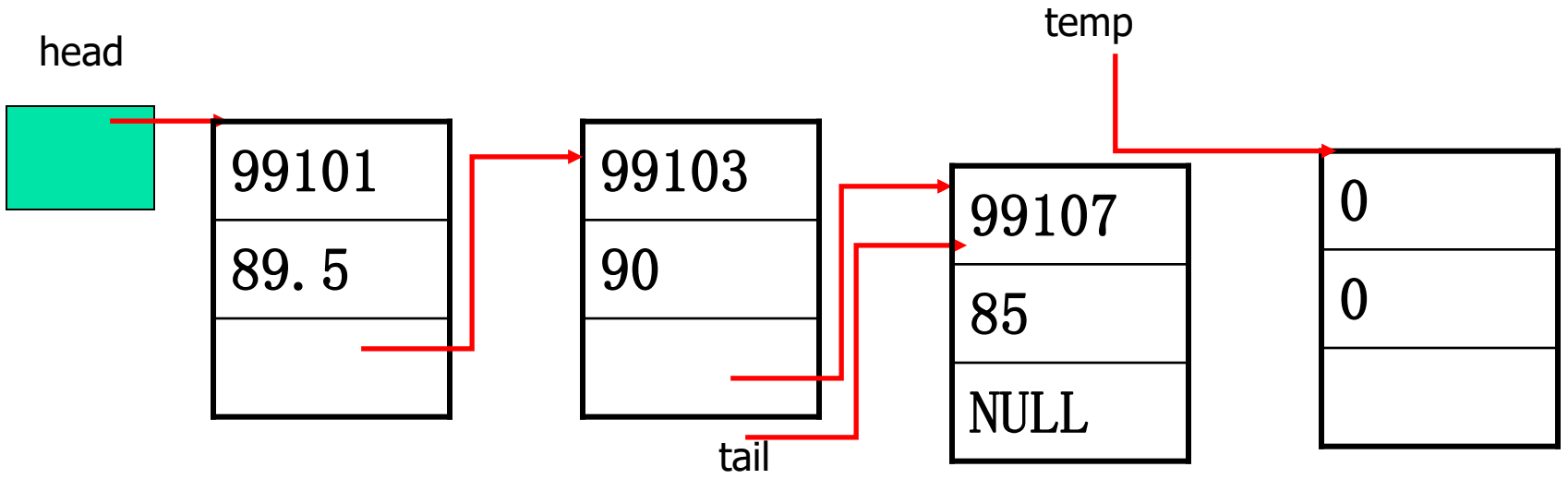

temp
head
99101
89. 5
99101
89. 5
99107
85
tail

(a)


temp
head
99101
89. 5
99101
89. 5
99107
85
tail

(b)


temp
head
99101
89.5
99103
90
99107
85
0
0
tail


head
temp
99101
89.5
99103
90
99107
85
NULL
0
0
tail

例子1：建立链表，读入n个整数，每个整数作为一个新结点插入到链尾

```
#include <iostream.h>
class node  {
  int data;
  node * next;
};
node * createList(int n);
int main() {
   int n;
   node * listHead = NULL;
   cout << "Please enter the number of nodes:";
   cin >> n;
   if (n > 0)
        listHead = createList(n);
   return 0;
}
```

例子1：建立链表，读入n个整数，每个整数作为一个新结点插入到链尾

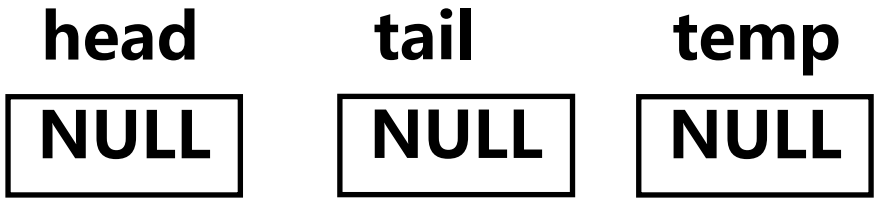
```
node *createList(int n) {
    node *temp, *tail = NULL, *head = NULL ;
    int num;
    cin >> num;
    head = new node ;  // 为新节点动态分配内存
    if (head == NULL)  {
        cout << "No memory available!";
        return NULL;
      }
      else  {
        head->data = num;
        head->next = NULL;
        tail = head;
    }
```

例子1：建立链表，读入n个整数，每个整数作为一个新结点插入到链尾

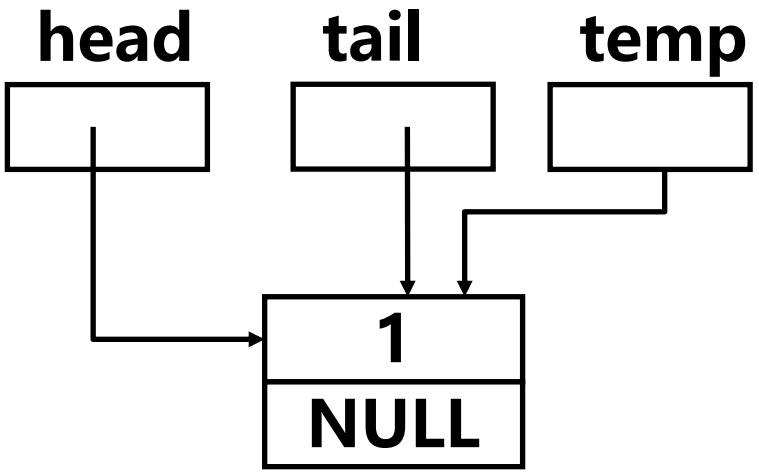
```
  for ( int i = 0; i < n - 1; i ++)  {
    cin >> num;
    temp = new node ;  // 为新节点动态分配内存
    if (temp == NULL) {
      cout << "No memory available!";
      return head;
    }
    else  {
      temp->data = num;
      temp->next = NULL;
      tail->next = temp;
      tail = temp;
    }
  }
  return head ;
}
```

建立链表过程

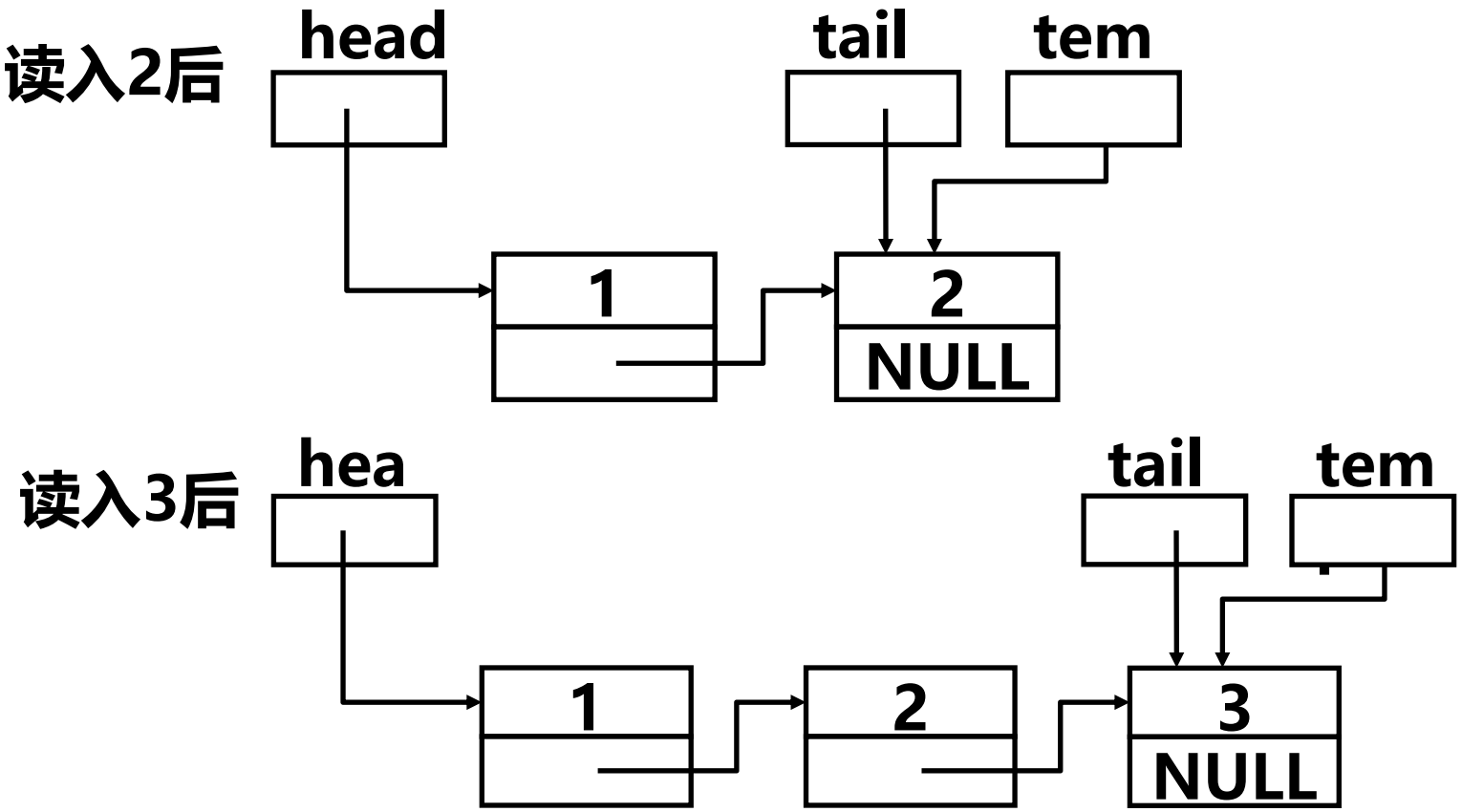
初始状态



读入1后



建立链表过程



9.2.2 单向链表的操作

- 遍历链表
  - 依次访问链表中的每个节点的信息

```
head->data = 15;
head->next->data = 15;
```

  - 一般遍历方法

```
node * curNode = head;
while (curNode )
 curNode = curNode->next;
```

例子2：编写一个函数，输出例1链表中各节点的data成员的值

```
void outputList(node * head)
{
    cout << "List: ";
    node *curNode = head;
    while ( curNode )   {
      cout << curNode->data;
      if (curNode ->next)
            cout << " -> ";
      curNode = curNode ->next;
    }
    cout << endl;
    return;
}
```

例子3：编写一个函数，在例1的链表中查找包含指定整数的节点

```
node * findData(int n, node * head)
{
  node *curNode = head;
  while ( curNode )  {
     if ( curNode->data == n)  {
       cout<<"Find "<<n<<" in the list."<<endl;
       return curNode;
     }
     curNode = curNode->next;
  }
  cout<<"Can't find "<<n<<" in the list."<<endl;
  return NULL;
}
```

9.2.2 单向链表的操作

- 在链表中节点a之后插入节点c

  (1) 指针cptr指向节点c,aptr指向节点a;

  (2) 把a后继节点的地址赋给节点c的后继指针

  ```
  cptr->next = aptr->next;
  ```

  (3) 把c的地址赋给节点a的后继指针

  ```
  aptr->next=cptr;
  ```

例子4：编写一个函数，将输入的整数从小到大插入链表

```
node * insertData(int n, node * head)
{
  node *curNode = head;// 指向插入点的后节点
  node *preNode = NULL;// 指向插入点的前节点
  node *newNode = NULL;// 指向新建节点
  while ((curNode!=NULL)&&(curNode->data<n)) {
    preNode = curNode; // 后节点变为前节点
    curNode = curNode->next;
  }
  newNode = new node ;
  if (newNode == NULL) {
    cout << "No memory available!";
    return head;
  }
```

例子4：编写一个函数，将输入的整数从小到大插入链表

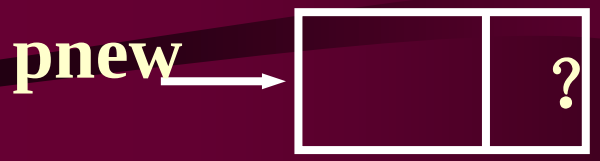
```
  newNode->data = n;
  if (preNode == NULL) //插入到链表头
  {
    newNode->next = curNode;
    return newNode;
  }
  else  {
    preNode->next = newNode;
    newNode->next = curNode;
    return head;
  }
}
```

在链表中插入节点

- **问题描述：**

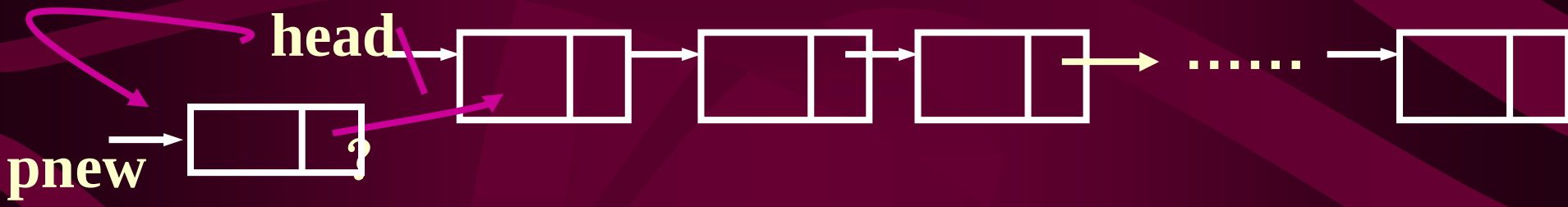
  有一个存放学生信息的有序链表，各节点按学生信息score的值从低到高排列，现在要求在该链表中插入一个新的学生信息，插入完成后，链表还是维持原来的升序排列。

(1) 建立新节点pnew




```
pnew=new student;
cin>>pnew->name>>pnew->score;
```

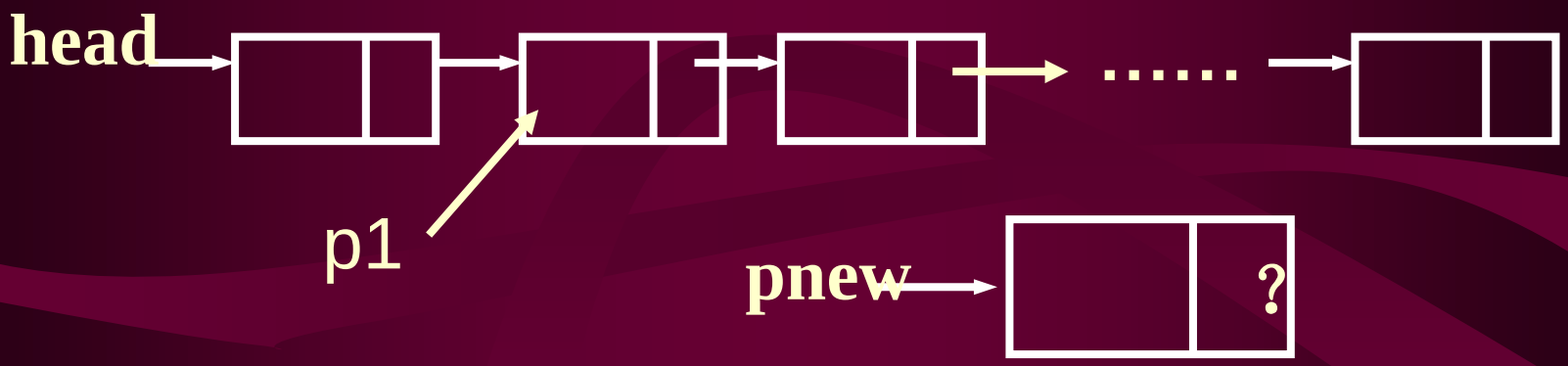
(2) 和头节点比较，如果需要插入头节点前面，则作相应处理




```
if (pnew->score<head->score){
        pnew->next=head; head=pnew;}
```

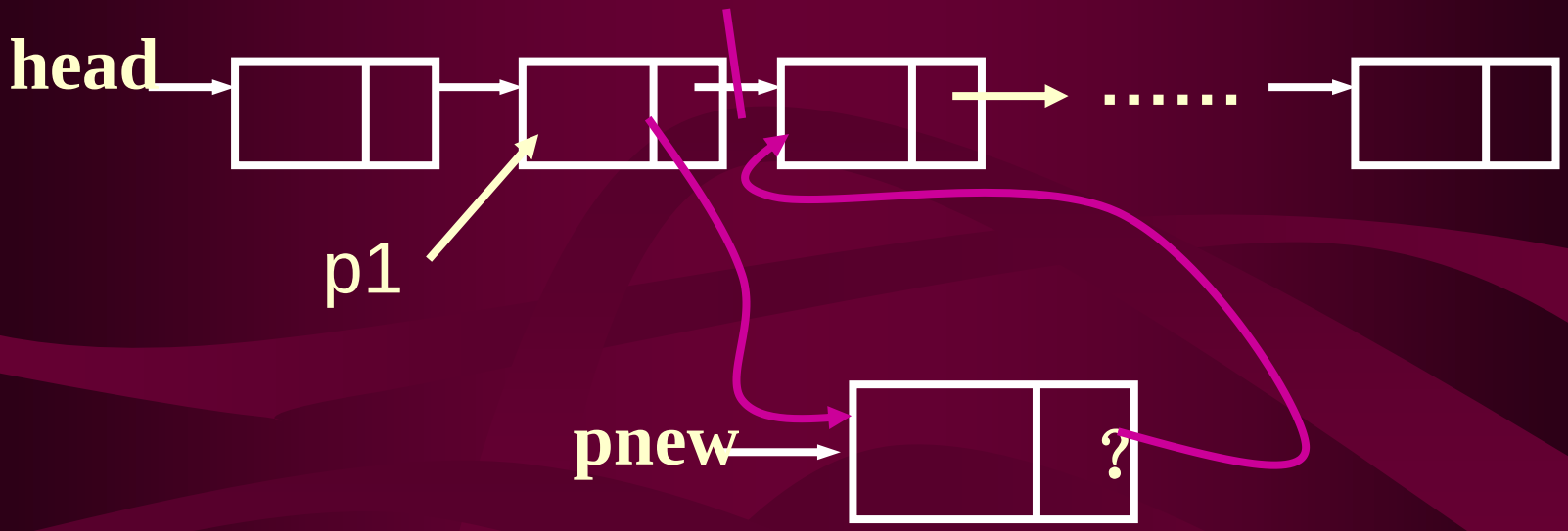
(2) 使p1指向头节点，并通过遍历，移动到插入位置

p1=head;




```
while(p1->next!=NULL)
    if(pnew->score>p1->next->score) p1=p1->next;
    else  break;
```

(3) 插入新节点




```
pnew->next=p1->next;
p1->next=pnew;
```

```
student *insert(student *head)
{   student *pnew,*p1;
     pnew=new student;
     cin>>pnew->name>>pnew->score;
    if(pnew->score<head->score) {
       pnew->next=head; head=pnew; return head;
     }
    p1=head;
     while(p1->next!=NULL)
         if(pnew->score>p1->next->score) p1=p1->next;
        else  break;
     pnew->next=p1->next;  p1->next=pnew;
  return head;
}
```

9.2.2 单向链表的操作

- 从链表中删除一个节点c

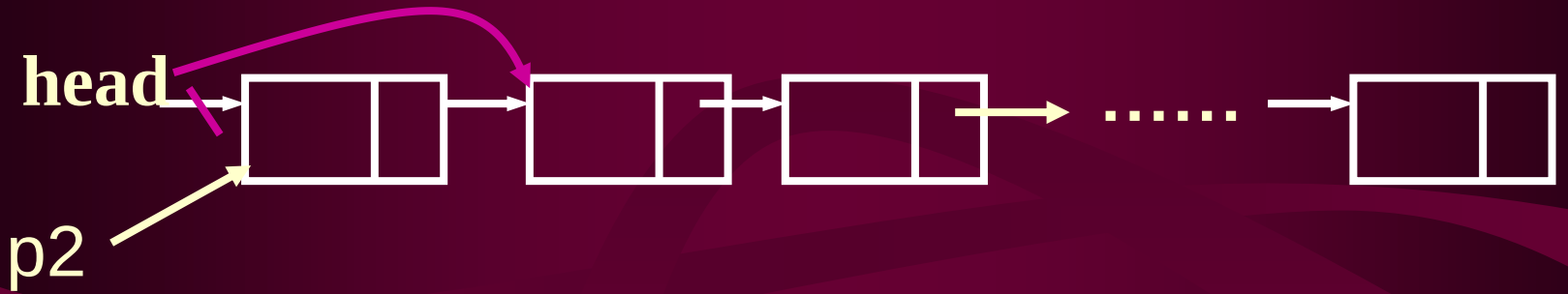
  (1)在链表中查找要删除的节点c，用指针cptr指向节点c;

  (2)如果c有前驱节点(设为d,用指针dptr指向d)，则将d的后继指针指向c的后继节点：dptr->next=cptr->next

  (3)释放c占用的空间

删除链表中的节点

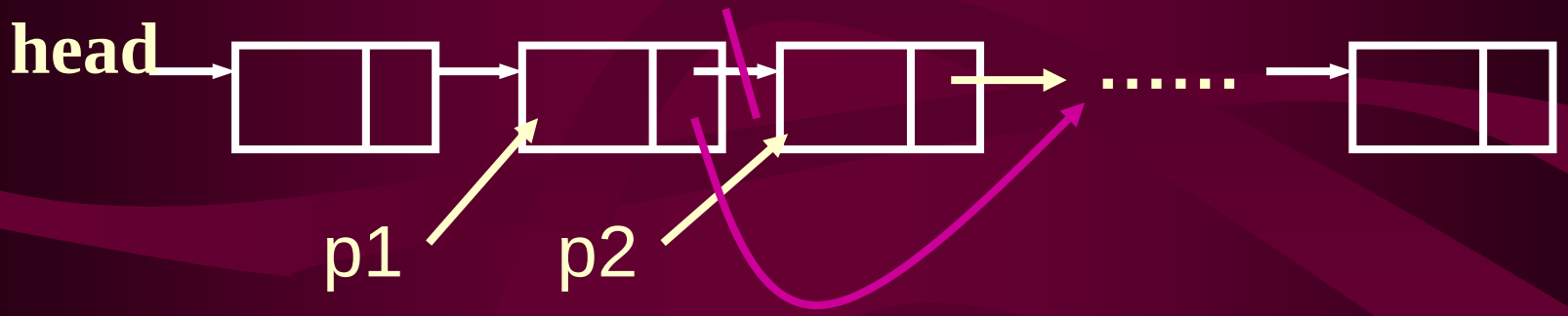
(1) 删除头节点



p2=head;

head=head->next; delete p2;

（2）删除其他节点




```
p1->next=p2->next;
delete p2;
```

例子5：编写一个函数，删除链表中包含指定整数的节点

```
node * deleteData(int n, node * head) {
  node *curNode = head;// 指向当前节点
  node *preNode = NULL;// 指向当前节点的前驱节点
  while (curNode && curNode->data != n) {
    preNode = curNode; // 当前节点变为前驱节点
    curNode = curNode->next;
  }
  if (curNode == NULL) {
    cout<<"Can't find "<<n<<" in the list"<<endl;
    return head;
  }
  if (preNode == NULL) //删除链首节点
    head = head->next;
  else
    preNode->next = curNode->next;
  delete curNode;
  return head; // 返回链首指针
}
```

课堂练习

- 节点类Node，包含整形数data；链表类LinkList，包含头指针Head，完成构造函数和析构函数，实现功能包括：向链表头插入一个整形数push，遍历链表并显示display，查找特定的整形数findData，按照从小到大的顺序插入一个整形数insertData，删除整形数deleteData.