一、链表的概念

定义：

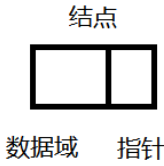
链表是一种**物理存储**上**非连续**，数据元素的**逻辑顺序**通过链表中的**指针**链接次序，实现的一种**线性**存储结构。

特点：

链表由一系列节点（链表中每一个元素称为节点）组成，节点在运行时**动态生成（malloc），**每个节点包括两个部分：

一个是存储数据元素的**数据域**

另一个是存储下一个节点地址的**指针域**



链表的构成：

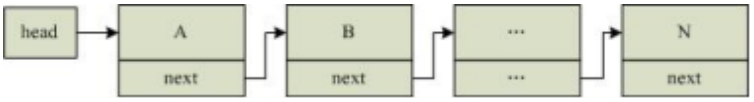
链表由一个个节点构成，每个节点一般采用结构体的形式组织，例如：

```
typedef struct student {
 int num;
 char name[20];
 struct student *next;
}STU;
```

链表节点分为两个域

数据域：存放各种实际的数据，如：num、score等

指针域：存放下一节点的首地址，如：next等.



链表的操作:

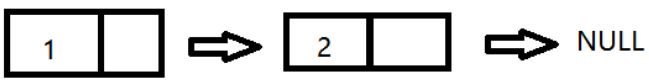
创建

增、删、改、查

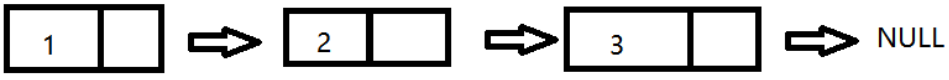
二、链表的创建

第一步：创建一个结点

第二步：创建第二个结点，将其放在第一个结点的后面（第一个结点的指针域保存第二个结点的地址）



第三步：再次创建结点，找到原本链表中的最后一个结点，接着讲最后一个结点的指针域保存新节点的地址，以此类推

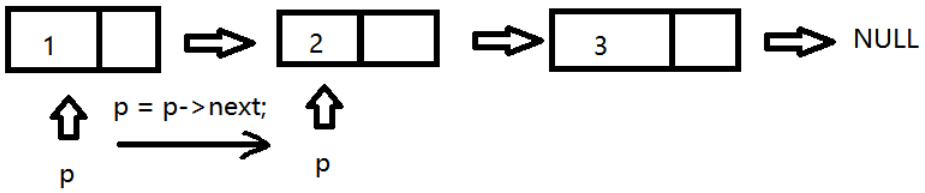


```c
#include <stdio.h>
#include <stdlib.h>
//定义结点结构体
typedef struct student
{
 //数据域
 int num; //学号
 int score; //分数
 char name[20]; //姓名
 //指针域
 struct student *next;
}STU;

void link_creat_head(STU **p_head,STU *p_new)
{
 STU *p_mov = *p_head;
 if(*p_head == NULL) //当第一次加入链表为空时，head执行p_new
 {
 *p_head = p_new;
 p_new->next=NULL;
 }
 else //第二次及以后加入链表
 {
 while(p_mov->next!=NULL)
 {
 p_mov=p_mov->next; //找到原有链表的最后一个节点
 }

 p_mov->next = p_new; //将新申请的节点加入链表
 p_new->next = NULL;
 }
}

int main()
{
 STU *head = NULL,*p_new = NULL;
 int num,i;
 printf("请输入链表初始个数:\n");
 scanf("%d",&num);
```

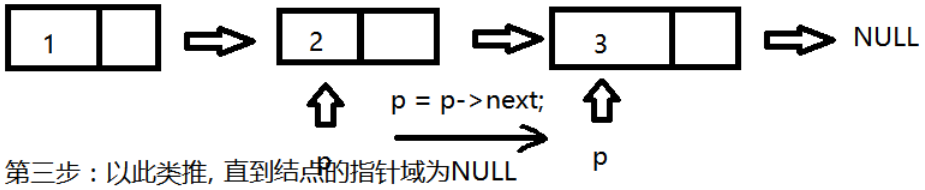
```
 for(i = 0; i < num;i++)
 {
 p_new = (STU*)malloc(sizeof(STU));//申请一个新节点
 printf("请输入学号、分数、名字:\n"); //给新节点赋值
 scanf("%d %d %s",&p_new->num,&p_new->score,p_new->name);

 link_creat_head(&head,p_new); //将新节点加入链表
 }
}
```

三、链表的遍历

第一步：输出第一个结点的数据域，输出完毕后，让指针保存后一个结点的地址



第二步：输出移动后地址对应的结点的数据域，输出完毕后，指针继续后移



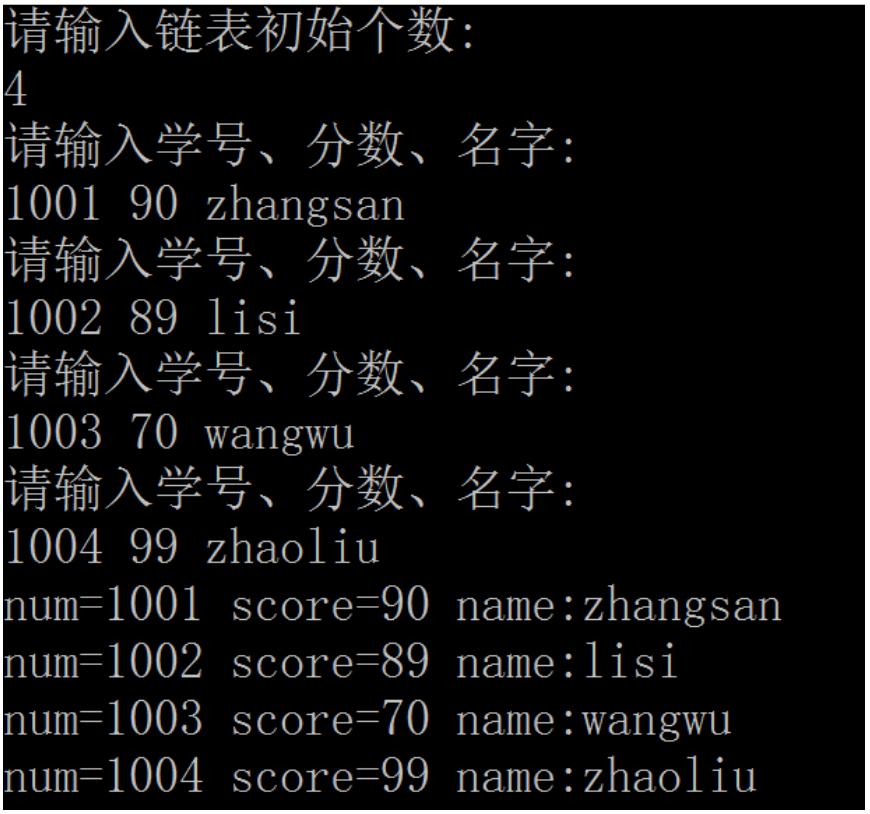
第三步：以此类推，直到结点的指针域为NULL

```
#include <stdio.h>
#include <stdlib.h>
//定义结点结构体
typedef struct student
{
 //数据域
 int num; //学号
 int score; //分数
 char name[20]; //姓名
 //指针域
 struct student *next;
}STU;

void link_creat_head(STU **p_head,STU *p_new)
{
```

```
 STU *p_mov = *p_head;
 if(*p_head == NULL) //当第一次加入链表为空时，head执行p_new
 {
 *p_head = p_new;
 p_new->next=NULL;
 }
 else //第二次及以后加入链表
 {
 while(p_mov->next!=NULL)
 {
 p_mov=p_mov->next; //找到原有链表的最后一个节点
 }

 p_mov->next = p_new; //将新申请的节点加入链表
 p_new->next = NULL;
 }
}

//链表的遍历
void link_print(STU *head)
{
 STU *p_mov;
 //定义新的指针保存链表的首地址，防止使用head改变原本链表
 p_mov = head;
 //当指针保存最后一个结点的指针域为NULL时，循环结束
 while(p_mov!=NULL)
 {
 //先打印当前指针保存结点的指针域
 printf("num=%d score=%d name:%s\n",p_mov->num,\
 p_mov->score,p_mov->name);

 //指针后移，保存下一个结点的地址
 p_mov = p_mov->next;
 }
}

int main()
{
 STU *head = NULL,*p_new = NULL;
 int num,i;
```

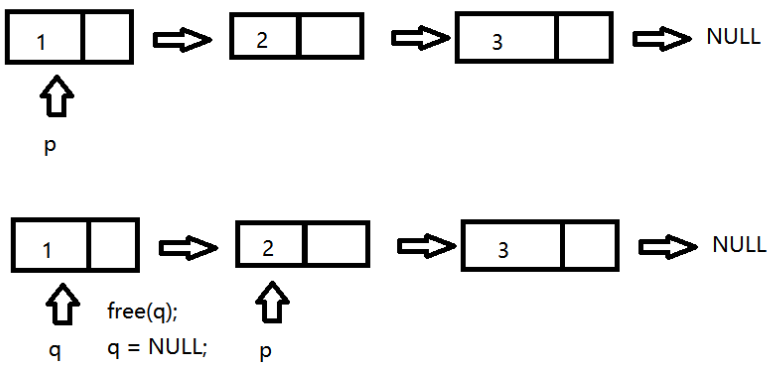
```
56   printf("请输入链表初始个数:\n");
57   scanf("%d",&num);
58   for(i = 0; i < num;i++)
59   {
60   p_new = (STU*)malloc(sizeof(STU));//申请一个新节点
61   printf("请输入学号、分数、名字:\n"); //给新节点赋值
62   scanf("%d %d %s",&p_new->num,&p_new->score,p_new->name);
63
64   link_creat_head(&head,p_new); //将新节点加入链表
65   }
66
67   link_print(head);
68 }
```

执行结果



四、链表的释放

重新定义一个指针q，保存p指向结点的地址，然后p后移保存下一个结点的地址，然后释放q对应的结点，一次类推，直到p为NULL为止




```c
//链表的释放
void link_free(STU **p_head)
{
 //定义一个指针变量保存头结点的地址
 STU *pb=*p_head;

 while(*p_head!=NULL)
 {
 //先保存p_head指向的结点的地址
  pb=*p_head;
  //p_head保存下一个结点地址
  *p_head=(*p_head)->next;
  //释放结点并防止野指针
  free(pb);
  pb = NULL;
  }
}
```

五、链表节点的查找

先对比第一个结点的数据域是否是想要的数据，如果是就直接返回，如果不是则继续查找下一个结点，如果到达最后一个结点的时候都没有匹配的数据，说明要查找数据不存在

```c
#include <stdio.h>
#include <stdlib.h>
//定义结点结构体
typedef struct student
{
 //数据域
 int num; //学号
```

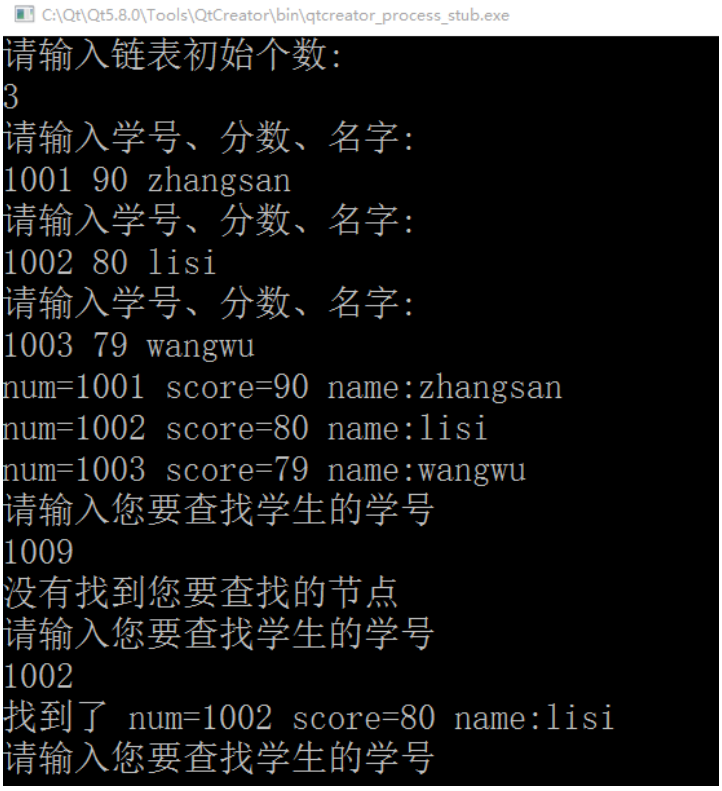
```
 int score; //分数
 char name[20]; //姓名
  //指针域
  struct student *next;
}STU;

void link_creat_head(STU **p_head,STU *p_new)
{
  STU *p_mov = *p_head;
  if(*p_head == NULL) //当第一次加入链表为空时，head执行p_new
  {
  *p_head = p_new;
  p_new->next=NULL;
  }
  else //第二次及以后加入链表
  {
  while(p_mov->next!=NULL)
  {
  p_mov=p_mov->next; //找到原有链表的最后一个节点
  }

  p_mov->next = p_new; //将新申请的节点加入链表
  p_new->next = NULL;
  }
}

//链表的遍历
void link_print(STU *head)
{
  STU *p_mov;
  //定义新的指针保存链表的首地址，防止使用head改变原本链表
  p_mov = head;
  //当指针保存最后一个结点的指针域为NULL时，循环结束
  while(p_mov!=NULL)
  {
  //先打印当前指针保存结点的指针域
  printf("num=%d score=%d name:%s\n",p_mov->num,\
  p_mov->score,p_mov->name);

  //指针后移，保存下一个结点的地址
```

```
 p_mov = p_mov->next;
 }
}

//链表的释放
void link_free(STU **p_head)
{
 //定义一个指针变量保存头结点的地址
 STU *pb=*p_head;

 while(*p_head!=NULL)
 {
 //先保存p_head指向的结点的地址
 pb=*p_head;
 //p_head保存下一个结点地址
 *p_head=(*p_head)->next;
 //释放结点并防止野指针
 free(pb);
 pb = NULL;
 }
}

//链表的查找
//按照学号查找
STU * link_search_num(STU *head,int num)
{
 STU *p_mov;
 //定义的指针变量保存第一个结点的地址
 p_mov=head;
 //当没有到达最后一个结点的指针域时循环继续
 while(p_mov!=NULL)
 {
 //如果找到是当前结点的数据，则返回当前结点的地址
 if(p_mov->num == num)//找到了
 {
 return p_mov;
 }
 //如果没有找到，则继续对比下一个结点的指针域
 p_mov=p_mov->next;
 }
```

```
88
89   //当循环结束的时候还没有找到，说明要查找的数据不存在，返回NULL进行标识
90   return NULL;//没有找到
91  }
92
93  //按照姓名查找
94  STU * link_search_name(STU *head,char *name)
95  {
96   STU *p_mov;
97   p_mov=head;
98   while(p_mov!=NULL)
99   {
100   if(strcmp(p_mov->name,name)==0)//找到了
101   {
102   return p_mov;
103   }
104   p_mov=p_mov->next;
105   }
106   return NULL;//没有找到
107  }
108
109  int main()
110  {
111   STU *head = NULL,*p_new = NULL;
112   int num,i;
113   printf("请输入链表初始个数:\n");
114   scanf("%d",&num);
115   for(i = 0; i < num;i++)
116   {
117   p_new = (STU*)malloc(sizeof(STU));//申请一个新节点
118   printf("请输入学号、分数、名字:\n"); //给新节点赋值
119   scanf("%d %d %s",&p_new->num,&p_new->score,p_new->name);
120
121   link_creat_head(&head,p_new); //将新节点加入链表
122   }
123
124   link_print(head);
125
126   //查学号
127  #if 1
```

```
 STU *pb;
 while(1)
 {
 printf("请输入您要查找学生的学号\n");
 scanf("%d",&num);
 pb=link_search_num(head,num);
 if(pb!=NULL)//找到了
 {
 printf("找到了 num=%d score=%d name:%s\n",pb->num,pb->score,pb->name);
 }
 else
 {
 printf("没有找到您要查找的节点\n");
 }
 }
#endif

 //查姓名
#if 0
 char name[32] = "";
 while(1)
 {
 printf("请输入您要查找学生的姓名\n");
 scanf("%s",name);
 pb=link_search_name(head,name);
 if(pb!=NULL)//找到了
 {
 printf("找到了 num=%d score=%d name:%s\n",pb->num,pb->score,pb->name);
 }
 else
 {
 printf("没有找到您要查找的节点\n");
 }
 }
#endif
 link_free(&head);
}

```

执行结果



六、链表节点的删除

如果链表为空，不需要删除

如果删除的是第一个结点，则需要将保存链表首地址的指针保存第一个结点的下一个结点的地址

如果删除的是中间结点，则找到中间结点的前一个结点，让前一个结点的指针域保存这个结点的后一个结点的地址即可

```
#include <stdio.h>
#include <stdlib.h>
//定义结点结构体
typedef struct student
{
 //数据域
 int num; //学号
 int score; //分数
 char name[20]; //姓名
  //指针域
  struct student *next;
}STU;

void link_creat_head(STU **p_head,STU *p_new)
{
 STU *p_mov = *p_head;
```

```
 if(*p_head == NULL) //当第一次加入链表为空时，head执行p_new
 {
 *p_head = p_new;
 p_new->next=NULL;
 }
 else //第二次及以后加入链表
 {
 while(p_mov->next!=NULL)
 {
 p_mov=p_mov->next; //找到原有链表的最后一个节点
 }

 p_mov->next = p_new; //将新申请的节点加入链表
 p_new->next = NULL;
 }
}

//链表的遍历
void link_print(STU *head)
{
 STU *p_mov;
 //定义新的指针保存链表的首地址，防止使用head改变原本链表
 p_mov = head;
 //当指针保存最后一个结点的指针域为NULL时，循环结束
 while(p_mov!=NULL)
 {
 //先打印当前指针保存结点的指针域
 printf("num=%d score=%d name:%s\n",p_mov->num,\
 p_mov->score,p_mov->name);

 //指针后移，保存下一个结点的地址
 p_mov = p_mov->next;
 }
}

//链表的释放
void link_free(STU **p_head)
{
 //定义一个指针变量保存头结点的地址
 STU *pb=*p_head;
```

```
57
58  while(*p_head!=NULL)
59  {
60  //先保存p_head指向的结点的地址
61  pb=*p_head;
62  //p_head保存下一个结点地址
63  *p_head=(*p_head)->next;
64  //释放结点并防止野指针
65  free(pb);
66  pb = NULL;
67  }
68 }
69
70 //链表结点的删除
71 void link_delete_num(STU **p_head,int num)
72 {
73  STU *pb,*pf;
74  pb=pf=*p_head;
75  if(*p_head == NULL)//链表为空，不用删
76  {
77  printf("链表为空，没有您要删的节点");\
78  return ;
79  }
80  while(pb->num != num && pb->next !=NULL)//循环找，要删除的节点
81  {
82  pf=pb;
83  pb=pb->next;
84  }
85  if(pb->num == num)//找到了一个节点的num和num相同
86  {
87  if(pb == *p_head)//要删除的节点是头节点
88  {
89  //让保存头结点的指针保存后一个结点的地址
90  *p_head = pb->next;
91  }
92  else
93  {
94  //前一个结点的指针域保存要删除的后一个结点的地址
95  pf->next = pb->next;
96  }
```

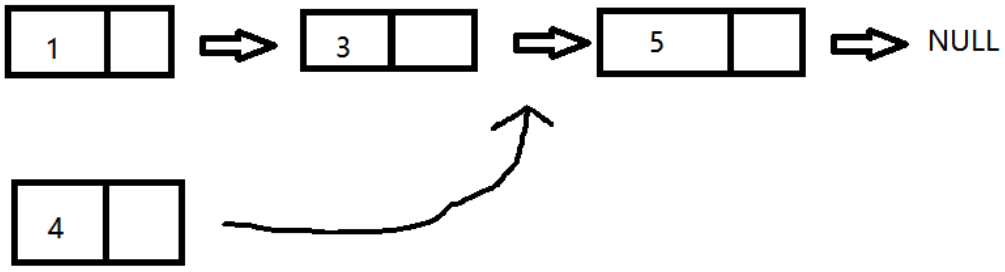
```
97
98  //释放空间
99  free(pb);
100  pb = NULL;
101  }
102  else//没有找到
103  {
104  printf("没有您要删除的节点\n");
105  }
106 }
107
108 int main()
109 {
110  STU *head = NULL,*p_new = NULL;
111  int num,i;
112  printf("请输入链表初始个数:\n");
113  scanf("%d",&num);
114  for(i = 0; i < num;i++)
115  {
116  p_new = (STU*)malloc(sizeof(STU));//申请一个新节点
117  printf("请输入学号、分数、名字:\n"); //给新节点赋值
118  scanf("%d %d %s",&p_new->num,&p_new->score,p_new->name);
119
120  link_creat_head(&head,p_new); //将新节点加入链表
121  }
122
123  link_print(head);
124
125  printf("请输入您要删除的节点的学号\n");
126  scanf("%d",&num);
127  link_delete_num(&head,num);
128
129  link_print(head);
130
131  link_free(&head);
132 }
```

执行结果

```
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
请输入链表初始个数:
3
请输入学号、分数、名字:
1001 90 zhangsan
请输入学号、分数、名字:
1002 99 lisi
请输入学号、分数、名字:
1003 89 wangwu
num=1001 score=90 name:zhangsan
num=1002 score=99 name:lisi
num=1003 score=89 name:wangwu
请输入您要删除的节点的学号
1002
num=1001 score=90 name:zhangsan
num=1003 score=89 name:wangwu
```

七、链表中插入一个节点

链表中插入一个结点，按照原本链表的顺序插入，找到合适的位置



情况（按照从小到大）：

如果链表没有结点，则新插入的就是第一个结点

如果新插入的结点的数值最小，则作为头结点

如果新插入的结点的数值在中间位置，则找到前一个，然后插入到他们中间

如果新插入的结点的数值最大，则插入到最后

```c
#include <stdio.h>
#include <stdlib.h>
//定义结点结构体
typedef struct student
{
 //数据域
 int num; //学号
 int score; //分数
 char name[20]; //姓名
 //指针域
```

```
 struct student *next;
}STU;

void link_creat_head(STU **p_head,STU *p_new)
{
 STU *p_mov = *p_head;
 if(*p_head == NULL) //当第一次加入链表为空时，head执行p_new
 {
 *p_head = p_new;
 p_new->next=NULL;
 }
 else //第二次及以后加入链表
 {
 while(p_mov->next!=NULL)
 {
 p_mov=p_mov->next; //找到原有链表的最后一个节点
 }

 p_mov->next = p_new; //将新申请的节点加入链表
 p_new->next = NULL;
 }
}

//链表的遍历
void link_print(STU *head)
{
 STU *p_mov;
 //定义新的指针保存链表的首地址，防止使用head改变原本链表
 p_mov = head;
 //当指针保存最后一个结点的指针域为NULL时，循环结束
 while(p_mov!=NULL)
 {
 //先打印当前指针保存结点的指针域
 printf("num=%d score=%d name:%s\n",p_mov->num,\
 p_mov->score,p_mov->name);

 //指针后移，保存下一个结点的地址
 p_mov = p_mov->next;
 }
}
```

```
51
52 //链表的释放
53 void link_free(STU **p_head)
54 {
55  //定义一个指针变量保存头结点的地址
56  STU *pb=*p_head;
57
58  while(*p_head!=NULL)
59  {
60  //先保存p_head指向的结点的地址
61  pb=*p_head;
62  //p_head保存下一个结点地址
63  *p_head=(*p_head)->next;
64  //释放结点并防止野指针
65  free(pb);
66  pb = NULL;
67  }
68 }
69
70 //链表的插入：按照学号的顺序插入
71 void link_insert_num(STU **p_head,STU *p_new)
72 {
73  STU *pb,*pf;
74  pb=pf=*p_head;
75  if(*p_head ==NULL)// 链表为空链表
76  {
77  *p_head = p_new;
78  p_new->next=NULL;
79  return ;
80  }
81  while((p_new->num >= pb->num) && (pb->next !=NULL) )
82  {
83  pf=pb;
84  pb=pb->next;
85  }
86
87  if(p_new->num < pb->num)//找到一个节点的num比新来的节点num大，插在pb的前面
88  {
89  if(pb== *p_head)//找到的节点是头节点，插在最前面
90  {
```

```
91  p_new->next= *p_head;
92  *p_head =p_new;
93  }
94  else
95  {
96  pf->next=p_new;
97  p_new->next = pb;
98  }
99  }
100  else//没有找到pb的num比p_new->num大的节点，插在最后
101  {
102  pb->next =p_new;
103  p_new->next =NULL;
104  }
105 }
106
107 int main()
108 {
109  STU *head = NULL,*p_new = NULL;
110  int num,i;
111  printf("请输入链表初始个数:\n");
112  scanf("%d",&num);
113  for(i = 0; i < num;i++)
114  {
115  p_new = (STU*)malloc(sizeof(STU));//申请一个新节点
116  printf("请输入学号、分数、名字:\n"); //给新节点赋值
117  scanf("%d %d %s",&p_new->num,&p_new->score,p_new->name);
118
119  link_creat_head(&head,p_new); //将新节点加入链表
120  }
121
122  link_print(head);
123
124  while(1)
125  {
126  printf("请输入您要插入的节点的 num score name\n");
127  p_new=(STU*)malloc(sizeof(STU));//申请一个新节点
128  scanf("%d %d %s",&p_new->num,&p_new->score,p_new->name);
129  link_insert_num(&head,p_new);
130  link_print(head);
```

```
131   }
132
133   link_free(&head);
134  }
135
```

执行结果

八、链表排序

如果链表为空，不需要排序

如果链表只有一个结点，不需要排序

先将第一个结点与后面所有的结点依次对比数据域，只要有比第一个结点数据域小的，则交换位置，

交换之后，拿新的第一个结点的数据域与下一个结点再次对比，如果比他小，再次交换，依次类推

第一个结点确定完毕之后，接下来再将第二个结点与后面所有的结点对比，直到最后一个结点也对比完毕为止

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  //定义结点结构体
4  typedef struct student
5  {
6   //数据域
```

```
7  int num; //学号
8  int score; //分数
9  char name[20]; //姓名
10  //指针域
11  struct student *next;
12 }STU;
13
14 void link_creat_head(STU **p_head,STU *p_new)
15 {
16  STU *p_mov = *p_head;
17  if(*p_head == NULL) //当第一次加入链表为空时，head执行p_new
18  {
19  *p_head = p_new;
20  p_new->next=NULL;
21  }
22  else //第二次及以后加入链表
23  {
24  while(p_mov->next!=NULL)
25  {
26  p_mov=p_mov->next; //找到原有链表的最后一个节点
27  }
28
29  p_mov->next = p_new; //将新申请的节点加入链表
30  p_new->next = NULL;
31  }
32 }
33
34 //链表的遍历
35 void link_print(STU *head)
36 {
37  STU *p_mov;
38  //定义新的指针保存链表的首地址，防止使用head改变原本链表
39  p_mov = head;
40  //当指针保存最后一个结点的指针域为NULL时，循环结束
41  while(p_mov!=NULL)
42  {
43  //先打印当前指针保存结点的指针域
44  printf("num=%d score=%d name:%s\n",p_mov->num,\
45  p_mov->score,p_mov->name);
46
```

```
 //指针后移，保存下一个结点的地址
 p_mov = p_mov->next;
 }
}

//链表的释放
void link_free(STU **p_head)
{
 //定义一个指针变量保存头结点的地址
 STU *pb=*p_head;

 while(*p_head!=NULL)
 {
 //先保存p_head指向的结点的地址
 pb=*p_head;
 //p_head保存下一个结点地址
 *p_head=(*p_head)->next;
 //释放结点并防止野指针
 free(pb);
 pb = NULL;
 }
}

//链表的排序
void link_order(STU *head)
{
 STU *pb,*pf,temp;
 pf=head;

 if(head==NULL)
 {
 printf("链表为空,不用排序\n");
 return ;
 }

 if(head->next ==NULL)
 {
 printf("只有一个节点，不用排序\n");
 return ;
 }
```

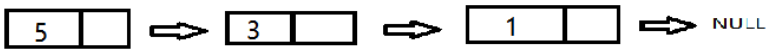
```
87
88  while(pf->next !=NULL)//以pf指向的节点为基准节点，
89  {
90  pb=pf->next;//pb从基准元素的下个元素开始
91  while(pb!=NULL)
92  {
93  if(pf->num > pb->num)
94  {
95  temp=*pb;
96  *pb=*pf;
97  *pf=temp;
98
99  temp.next=pb->next;
100  pb->next=pf->next;
101  pf->next=temp.next;
102  }
103  pb=pb->next;
104  }
105  pf=pf->next;
106  }
107 }
108
109 int main()
110 {
111  STU *head = NULL,*p_new = NULL;
112  int num,i;
113  printf("请输入链表初始个数:\n");
114  scanf("%d",&num);
115  for(i = 0; i < num;i++)
116  {
117  p_new = (STU*)malloc(sizeof(STU));//申请一个新节点
118  printf("请输入学号、分数、名字:\n"); //给新节点赋值
119  scanf("%d %d %s",&p_new->num,&p_new->score,p_new->name);
120
121  link_creat_head(&head,p_new); //将新节点加入链表
122  }
123
124  link_print(head);
125
126  printf("*************************\n");
127
```

```
128   link_order(head);
129
130   link_print(head);
131
132   link_free(&head);
133  }
```

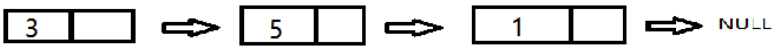
执行结果

```
请输入链表初始个数:
3
请输入学号、分数、名字:
1005 90 zhangsan
请输入学号、分数、名字:
1004 89 lisi
请输入学号、分数、名字:
1001 80 wangwu
num=1005 score=90 name:zhangsan
num=1004 score=89 name:lisi
num=1001 score=80 name:wangwu
**************************
num=1001 score=80 name:wangwu
num=1004 score=89 name:lisi
num=1005 score=90 name:zhangsan
```

九、链表逆序



第一次：将3结点移动到5结点的前面



第二次：将5结点后面的1结点移动到3结点的前面


```
1  #include <stdio.h>
2  #include <stdlib.h>
3  //定义结点结构体
4  typedef struct student
5  {
6   //数据域
7   int num; //学号
```

```
 int score; //分数
 char name[20]; //姓名
  //指针域
  struct student *next;
}STU;

void link_creat_head(STU **p_head,STU *p_new)
{
  STU *p_mov = *p_head;
  if(*p_head == NULL) //当第一次加入链表为空时，head执行p_new
  {
  *p_head = p_new;
  p_new->next=NULL;
  }
  else //第二次及以后加入链表
  {
  while(p_mov->next!=NULL)
  {
  p_mov=p_mov->next; //找到原有链表的最后一个节点
  }

  p_mov->next = p_new; //将新申请的节点加入链表
  p_new->next = NULL;
  }
}

//链表的遍历
void link_print(STU *head)
{
  STU *p_mov;
  //定义新的指针保存链表的首地址，防止使用head改变原本链表
  p_mov = head;
  //当指针保存最后一个结点的指针域为NULL时，循环结束
  while(p_mov!=NULL)
  {
  //先打印当前指针保存结点的指针域
  printf("num=%d score=%d name:%s\n",p_mov->num,\
  p_mov->score,p_mov->name);

  //指针后移，保存下一个结点的地址
```

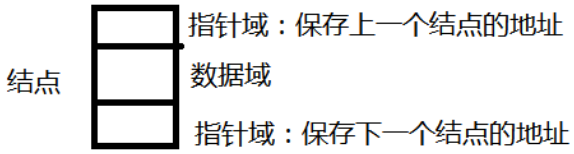
```
 p_mov = p_mov->next;
 }
}

//链表的释放
void link_free(STU **p_head)
{
 //定义一个指针变量保存头结点的地址
 STU *pb=*p_head;

 while(*p_head!=NULL)
 {
 //先保存p_head指向的结点的地址
 pb=*p_head;
 //p_head保存下一个结点地址
 *p_head=(*p_head)->next;
 //释放结点并防止野指针
 free(pb);
 pb = NULL;
 }
}

//链表逆序
STU *link_reverse(STU *head)
{
 STU *pf,*pb,*r;
 pf=head;
 pb=pf->next;

 while(pb!=NULL)
 {
 r=pb->next;
 pb->next=pf;
 pf=pb;
 pb=r;
 }
 head->next=NULL;
 head=pf;
 return head;
}
```

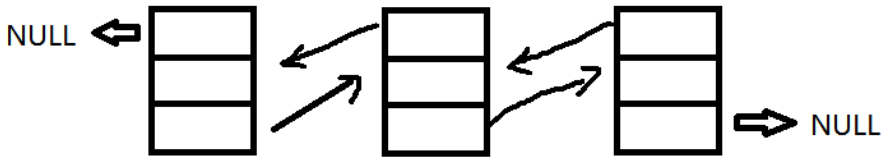
```
88
89 int main()
90 {
91  STU *head = NULL,*p_new = NULL;
92  int num,i;
93  printf("请输入链表初始个数:\n");
94  scanf("%d",&num);
95  for(i = 0; i < num;i++)
96  {
97  p_new = (STU*)malloc(sizeof(STU));//申请一个新节点
98  printf("请输入学号、分数、名字:\n"); //给新节点赋值
99  scanf("%d %d %s",&p_new->num,&p_new->score,p_new->name);
100
101  link_creat_head(&head,p_new); //将新节点加入链表
102  }
103
104  link_print(head);
105
106  printf("**********************\n");
107
108  head = link_reverse(head);
109
110  link_print(head);
111
112  link_free(&head);
113 }
114
```

执行结果

```
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
请输入链表初始个数:
3
请输入学号、分数、名字:
5 5 5
请输入学号、分数、名字:
3 3 3
请输入学号、分数、名字:
1 1 1
num=5 score=5 name:5
num=3 score=3 name:3
num=1 score=1 name:1
**********
num=1 score=1 name:1
num=3 score=3 name:3
num=5 score=5 name:5
```

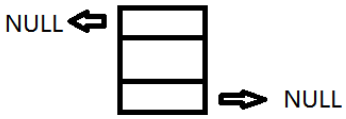
十、双向链表





10.1 双向链表的创建和遍历

第一步：创建一个结点作为头结点，将两个指针域都保存NULL



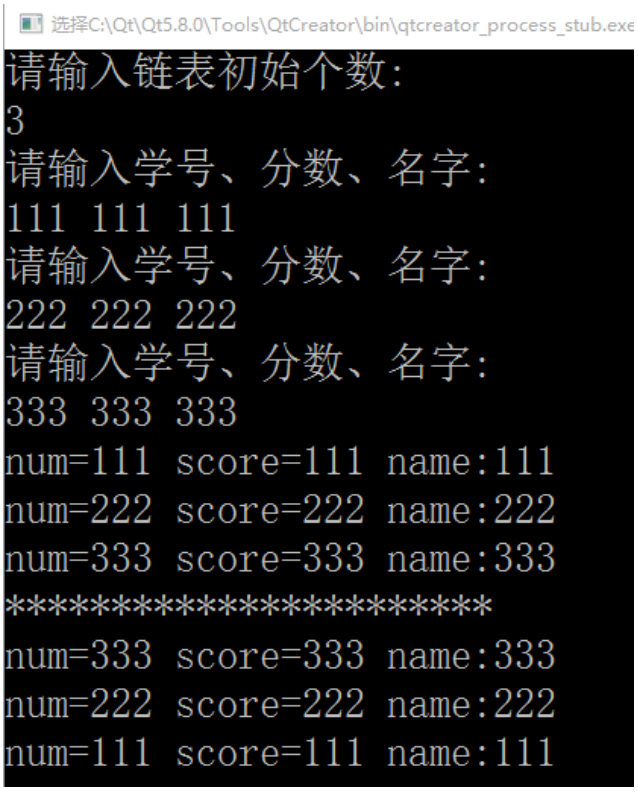
第二步：先找到链表中的最后一个结点，然后让最后一个结点的指针域保存新插入结点的地址，新插入结点的两个指针域，一个保存上一个结点地址，一个保存NULL


```c
#include <stdio.h>
#include <stdlib.h>

```

```c
//定义结点结构体
typedef struct student
{
 //数据域
 int num; //学号
 int score; //分数
  char name[20]; //姓名

 //指针域
 struct student *front; //保存上一个结点的地址
 struct student *next; //保存下一个结点的地址
}STU;

void double_link_creat_head(STU **p_head,STU *p_new)
{
 STU *p_mov=*p_head;
 if(*p_head==NULL) //当第一次加入链表为空时，head执行p_new
 {
 *p_head = p_new;
 p_new->front = NULL;
 p_new->next = NULL;
 }
 else //第二次及以后加入链表
 {
 while(p_mov->next!=NULL)
 {
 p_mov=p_mov->next; //找到原有链表的最后一个节点
 }
 p_mov->next = p_new; //将新申请的节点加入链表
 p_new->front = p_mov;
 p_new->next = NULL;
 }
}


void double_link_print(STU *head)
{
 STU *pb;
 pb=head;
 while(pb->next!=NULL)
```

```
 {
 printf("num=%d score=%d name:%s\n",pb->num,pb->score,pb->name);
 pb=pb->next;
 }
 printf("num=%d score=%d name:%s\n",pb->num,pb->score,pb->name);

 printf("**********************\n");

 while(pb!=NULL)
 {
 printf("num=%d score=%d name:%s\n",pb->num,pb->score,pb->name);
 pb=pb->front;
 }
}

int main()
{
 STU *head=NULL,*p_new=NULL;
 int num,i;
 printf("请输入链表初始个数:\n");
 scanf("%d",&num);
 for(i=0;i<num;i++)
 {
 p_new=(STU*)malloc(sizeof(STU));//申请一个新节点
 printf("请输入学号、分数、名字:\n"); //给新节点赋值
 scanf("%d %d %s",&p_new->num,&p_new->score,p_new->name);
 double_link_creat_head(&head,p_new); //将新节点加入链表
 }

 double_link_print(head);
}
```

执行结果



10.2 双向链表节点的删除

如果链表为空，则不需要删除

如果删除第一个结点，则保存链表首地址的指针保存后一个结点的地址，

　　并且让这个结点的front保存NULL

如果删除最后一个结点，只需要让最后一个结点的前一个结点的next保存NULL即可

如果删除中间结点，则让中间结点的前后两个结点的指针域分别保存对方的地址即可

```c
#include <stdio.h>
#include <stdlib.h>

//定义结点结构体
typedef struct student
{
 //数据域
 int num; //学号
 int score; //分数
  char name[20]; //姓名

 //指针域
 struct student *front; //保存上一个结点的地址
 struct student *next; //保存下一个结点的地址
}STU;

```

```c
void double_link_creat_head(STU **p_head,STU *p_new)
{
 STU *p_mov=*p_head;
 if(*p_head==NULL) //当第一次加入链表为空时，head执行p_new
 {
 *p_head = p_new;
 p_new->front = NULL;
 p_new->next = NULL;
 }
 else //第二次及以后加入链表
 {
 while(p_mov->next!=NULL)
 {
 p_mov=p_mov->next; //找到原有链表的最后一个节点
 }
 p_mov->next = p_new; //将新申请的节点加入链表
 p_new->front = p_mov;
 p_new->next = NULL;
 }
}


void double_link_print(STU *head)
{
 STU *pb;
 pb=head;
 while(pb->next!=NULL)
 {
 printf("num=%d score=%d name:%s\n",pb->num,pb->score,pb->name);
 pb=pb->next;
 }
 printf("num=%d score=%d name:%s\n",pb->num,pb->score,pb->name);

 printf("**********************\n");

 while(pb!=NULL)
 {
 printf("num=%d score=%d name:%s\n",pb->num,pb->score,pb->name);
 pb=pb->front;
 }
```

```
}

//双向链表的删除
void double_link_delete_num(STU **p_head,int num)
{
 STU *pb,*pf;
 pb=*p_head;
 if(*p_head==NULL)//链表为空，不需要删除
 {
 printf("链表为空，没有您要删除的节点\n");
 return ;
 }
 while((pb->num != num) && (pb->next != NULL) )
 {
 pb=pb->next;
 }
 if(pb->num == num)//找到了一个节点的num和num相同，删除pb指向的节点
 {
 if(pb == *p_head)//找到的节点是头节点
 {
 if((*p_head)->next==NULL)//只有一个节点的情况
 {
 *p_head=pb->next;
 }
 else//有多个节点的情况
 {
 *p_head = pb->next;//main函数中的head指向下个节点
 (*p_head)->front=NULL;
 }
 }
 else//要删的节点是其他节点
 {
 if(pb->next!=NULL)//删除中间节点
 {
 pf=pb->front;//让pf指向找到的节点的前一个节点
 pf->next=pb->next; //前一个结点的next保存后一个结点的地址
 (pb->next)->front=pf; //后一个结点的front保存前一个结点的地址
 }
 else//删除尾节点
 {
```

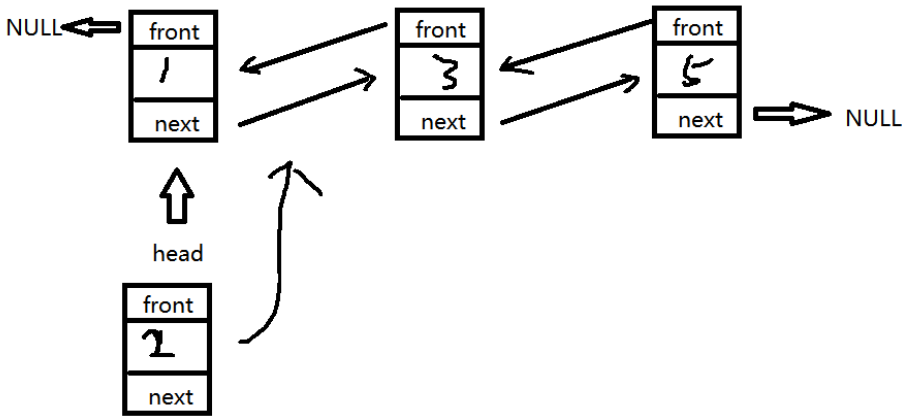
```
97  pf=pb->front;
98  pf->next=NULL;
99  }
100  }
101
102  free(pb);//释放找到的节点
103
104  }
105  else//没找到
106  {
107  printf("没有您要删除的节点\n");
108  }
109 }
110
111 int main()
112 {
113  STU *head=NULL,*p_new=NULL;
114  int num,i;
115  printf("请输入链表初始个数:\n");
116  scanf("%d",&num);
117  for(i=0;i<num;i++)
118  {
119  p_new=(STU*)malloc(sizeof(STU));//申请一个新节点
120  printf("请输入学号、分数、名字:\n"); //给新节点赋值
121  scanf("%d %d %s",&p_new->num,&p_new->score,p_new->name);
122  double_link_creat_head(&head,p_new); //将新节点加入链表
123  }
124
125  double_link_print(head);
126
127  printf("请输入您要删除的节点的num\n");
128  scanf("%d",&num);
129  double_link_delete_num(&head,num);
130  double_link_print(head);
131
132 }
```

执行结果

```
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
3
请输入学号、分数、名字:
111 111 111
请输入学号、分数、名字:
222 222 222
请输入学号、分数、名字:
333 333 333
num=111 score=111 name:111
num=222 score=222 name:222
num=333 score=333 name:333
**********
num=333 score=333 name:333
num=222 score=222 name:222
num=111 score=111 name:111
请输入您要删除的节点的num
222
num=111 score=111 name:111
num=333 score=333 name:333
**********
```

10.3 双向链表插入节点

按照顺序插入结点



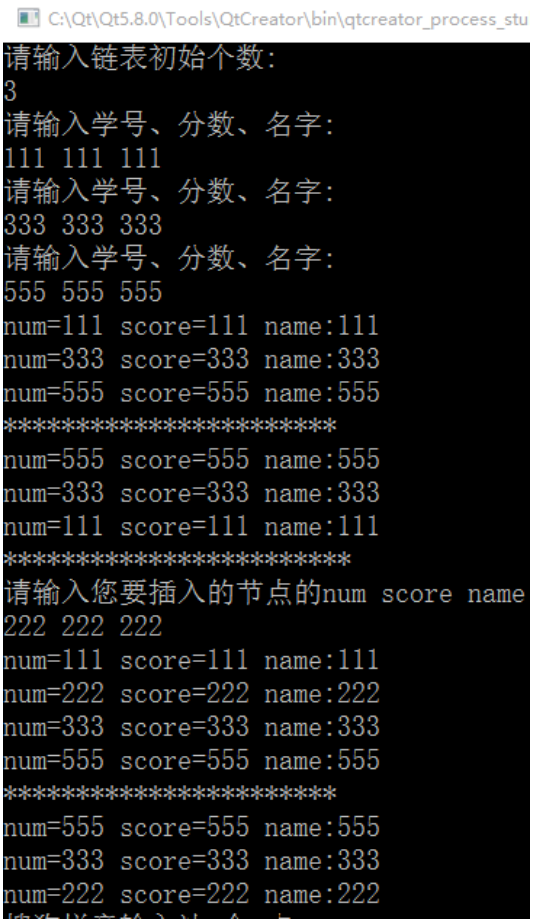

```c
#include <stdio.h>
#include <stdlib.h>

//定义结点结构体
typedef struct student
{
 //数据域
 int num; //学号
 int score; //分数
  char name[20]; //姓名

```

```
 //指针域
 struct student *front; //保存上一个结点的地址
 struct student *next; //保存下一个结点的地址
}STU;

void double_link_creat_head(STU **p_head,STU *p_new)
{
 STU *p_mov=*p_head;
 if(*p_head==NULL) //当第一次加入链表为空时，head执行p_new
 {
 *p_head = p_new;
 p_new->front = NULL;
 p_new->next = NULL;
 }
 else //第二次及以后加入链表
 {
 while(p_mov->next!=NULL)
 {
 p_mov=p_mov->next; //找到原有链表的最后一个节点
 }
 p_mov->next = p_new; //将新申请的节点加入链表
 p_new->front = p_mov;
 p_new->next = NULL;
 }
}


void double_link_print(STU *head)
{
 STU *pb;
 pb=head;
 while(pb->next!=NULL)
 {
 printf("num=%d score=%d name:%s\n",pb->num,pb->score,pb->name);
 pb=pb->next;
 }
 printf("num=%d score=%d name:%s\n",pb->num,pb->score,pb->name);

 printf("**********************\n");

```

```
 while(pb!=NULL)
 {
 printf("num=%d score=%d name:%s\n",pb->num,pb->score,pb->name);
 pb=pb->front;
 }
}

//双向链表的插入
void double_link_insert_num(STU **p_head,STU *p_new)
{
 STU *pb,*pf;
 pb=*p_head;
 if(*p_head == NULL)//链表为空，新来的节点就是头节点
 {
 *p_head=p_new;
 p_new->front=NULL;
 p_new->next=NULL;
 return ;
 }
 while((p_new->num >= pb->num) && (pb->next!=NULL) )
 {
 pb=pb->next ;
 }
 if(p_new->num < pb->num)//找到了一个pb的num比新来的节点的num大，插在pb前边
 {
 if(pb==*p_head)//找到的节点是头节点，插在头节点的前边
 {
 p_new->next=*p_head; //新插入的结点的next保存之前头结点的地址
 (*p_head)->front=p_new; //之前头结点的front保存新插入的结点的地址
 p_new->front=NULL; //新插入的结点的front保存NULL
 *p_head=p_new; //让原本保存链表首地址的指针保存新插入结点的地址
 }
 else
 {
 pf=pb->front;//pf指向 找到节点的前一个节点

 p_new->next=pb;
 p_new->front=pf;
 pf->next=p_new;
 pb->front=p_new;
```

```
92  }
93  }
94  else//所有pb指向节点的num都比p_new指向的节点的num小，插在最后
95  {
96  pb->next=p_new;
97  p_new->front=pb;
98  p_new->next=NULL;
99  }
100 }
101
102 int main()
103 {
104  STU *head=NULL,*p_new=NULL;
105  int num,i;
106  printf("请输入链表初始个数:\n");
107  scanf("%d",&num);
108  for(i=0;i<num;i++)
109  {
110  p_new=(STU*)malloc(sizeof(STU));//申请一个新节点
111  printf("请输入学号、分数、名字:\n"); //给新节点赋值
112  scanf("%d %d %s",&p_new->num,&p_new->score,p_new->name);
113  double_link_creat_head(&head,p_new); //将新节点加入链表
114  }
115
116  double_link_print(head);
117
118  printf("**********************\n");
119
120  while(1)
121  {
122  p_new=(STU*)malloc(sizeof(STU));//申请一个新节点
123  printf("请输入您要插入的节点的num score name\n");
124  scanf("%d %d %s",&p_new->num,&p_new->score,p_new->name);
125  double_link_insert_num(&head,p_new);
126  double_link_print(head);
127
128  }
129 }
130
```

执行结果


C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stu
请输入链表初始个数:
3
请输入学号、分数、名字:
111 111 111
请输入学号、分数、名字:
333 333 333
请输入学号、分数、名字:
555 555 555
num=111 score=111 name:111
num=333 score=333 name:333
num=555 score=555 name:555
****
num=555 score=555 name:555
num=333 score=333 name:333
num=111 score=111 name:111
****
请输入您要插入的节点的num score name
222 222 222
num=111 score=111 name:111
num=222 score=222 name:222
num=333 score=333 name:333
num=555 score=555 name:555
****
num=555 score=555 name:555
num=333 score=333 name:333
num=222 score=222 name:222