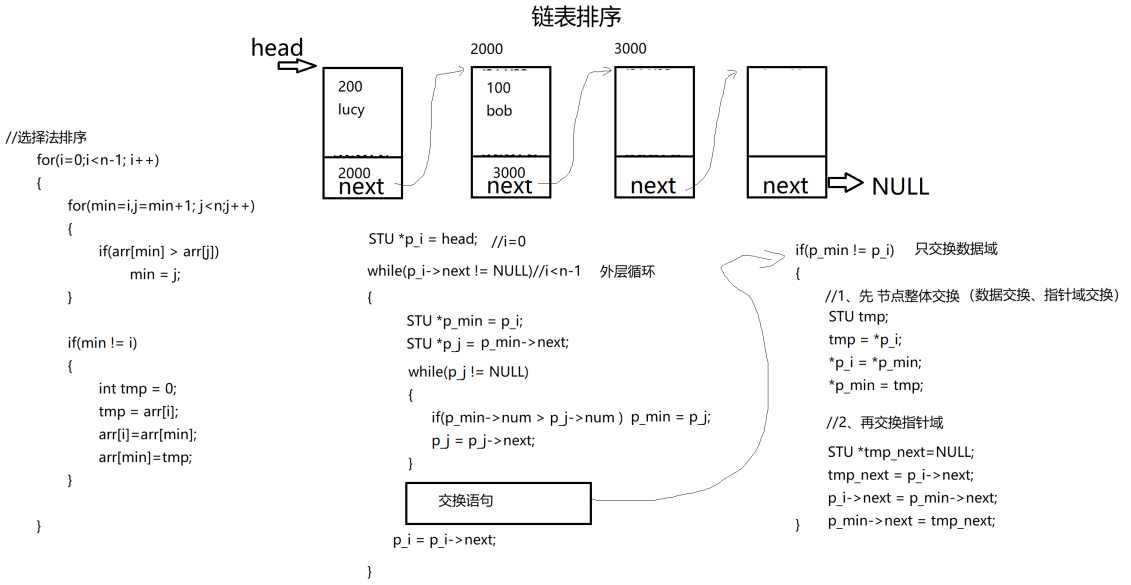
知识点1【链表的排序（选择法）】0-1（了解）




```c
void sort_link(STU *head)
{
 //1、判断链表是否存在
 if(NULL == head)
 {
 printf("link not found\n");
 return;
 }
 else
 {
 STU *p_i = head;//i=0
 while(p_i->next != NULL)//i<n-1 外层循环
 {
 STU *p_min = p_i;//min = i;
 STU *p_j = p_min->next;//j = min+1
 while(p_j != NULL)//j<n 内层循环
 {

 //寻找成员num最小值的 节点
 if(p_min->num > p_j->num)//if(arr[min] > arr[j])
 p_min = p_j;//min = j
 p_j = p_j->next;//j++

 }

```

```
 if(p_min != p_i)//min != i
 {
 //只交换数据域（1、节点内容整体交换 2、只交换指针域）
 //1、节点内容整体交换(数据域交换第1次 指针域交换第1次)
 STU tmp;
 tmp = *p_i;
 *p_i = *p_min;
 *p_min = tmp;

 //2、只交换指针域(指针域交换第2次)
 tmp.next = p_i->next;
 p_i->next = p_min->next;
 p_min->next = tmp.next;
 }


 p_i = p_i->next;//i++
 }

 }
}
```

知识点2【双向循环链表】（了解）0-2

main.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
//定义一个双向循环链表的节点类型
typedef struct stu
{
 //数据域
 int num;
 char name[32];
 int age;

 //指针域
 struct stu *next;//指向下一个节点
 struct stu *pre;//指向前一个节点
```

```c
}STU;

extern STU* insert_link(STU *head, STU tmp);
extern void print_link(STU *head);
extern STU* search_link(STU *head, char *name);
extern STU* delete_link(STU *head, int num);
extern STU* free_link(STU *head);

int main(int argc,char *argv[])
{
 char name[32]="";
 int num = 0;
 int n =0;//保存学生的个数
 int i=0;
 STU *head = NULL;
 STU *ret = NULL;

 printf("请输入学员的个数:");
 scanf("%d", &n);

 for(i=0;i<n;i++)
 {
 STU tmp;
 printf("请输入第%d个学员的信息:\n",i+1);
 scanf("%d %s %d", &tmp.num, tmp.name, &tmp.age);

 //插入一个节点
 head = insert_link(head, tmp);
 }

 //遍历整个链表
 print_link(head);

 //查找指定节点
 printf("请输入要查找的用户名:");
 scanf("%s",name);

 //查找中...
#if 1
 ret = search_link(head, name);
 if(ret == NULL)
```

```
 {
 printf("没有找到与%s相关的节点信息\n", name);
 }
 else
 {
 printf("查找到的信息为:%d %s %d\n", ret->num, ret->name,ret->age);
 }
#endif
 //删除指定节点 num
 printf("请输入要删除的学号:");
 scanf("%d", &num);
 head = delete_link(head, num);

 //遍历整个链表
 print_link(head);

 //释放整个链表
 head = free_link(head);

 //遍历整个链表
 print_link(head);

 return 0;
}

//头部之前插入
STU* insert_link(STU *head, STU tmp)
{
 //1、为插入的节点pi申请空间
 STU *pi = (STU *)calloc(1,sizeof(STU));
 //2、将tmp的值 赋值给 *pi
 *pi = tmp;

 //3、判断链表是否存在
 if(head == NULL)//链表不存在
 {
 head = pi;
 pi->next = head;
 pi->pre = head;
 }
 else//链表存在(头部之前插入)
```

```
97  {
98  pi->next = head;
99  pi->pre = head->pre;
100  head->pre->next = pi;
101  head->pre = pi;
102  head = pi;
103  }
104
105  return head;
106 }
107
108 #if 1
109 void print_link(STU *head)
110 {
111  //判断链表是否存在
112  if(head == NULL)
113  {
114  printf("link not found\n");
115  return;
116  }
117  else//链表存在
118  {
119  STU *pb = head;
120  do
121  {
122  //访问节点内容
123  printf("num=%d, name=%s, age=%d\n", pb->num,pb->name,pb->age);
124  //pb指向下一个节点
125  pb = pb->next;
126  }while(pb != head);
127  }
128  return;
129 }
130 #endif
131 #if 0
132 void print_link(STU *head)
133 {
134  //判断链表是否存在
135  if(head == NULL)
136  {
```

```
 printf("link not found\n");
 }
 else//链表存在
 {
 STU *pb = head;//pb指向头结点
 STU *pf = head->pre;//pf指向了尾节点

 do
 {
 if(pb == pf)//相遇 只需要打印pf或pb中任何一个信息就够了
 {
 printf("num=%d,name=%s,age=%d\n", pb->num,pb->name,pb->age);
 break;
 }
 printf("num=%d,name=%s,age=%d\n", pb->num,pb->name,pb->age);
 printf("num=%d,name=%s,age=%d\n", pf->num,pf->name,pf->age);

 pb = pb->next;//next方向的移动
 pf = pf->pre;//pre方向的移动
 }while( pb->pre != pf );//pf和pb不能 擦肩而过

 }
}

#endif

STU* search_link(STU *head, char *name)
{
 //判断链表是否存在
 if(head == NULL)
 {
 return NULL;
 }
 else//链表存在
 {
 STU *pb = head;//指向头结点
 STU *pf = head->pre;//指向的是尾节点
 printf("pb = %p\n", pb);
 printf("pf = %p\n", pf);
 printf("pb->pre = %p\n", pb->pre);
```

```
177
178
179  while( (strcmp(pb->name,name) != 0) && (strcmp(pf->name,name)!=0) && (p
b != pf) )
180  {
181  printf("##pb->name=%s##\n",pb->name);
182  printf("##pf->name=%s##\n",pf->name);
183  pb=pb->next;//next方向移动
184  pf=pf->pre;//pf方向移动
185
186  if(pb->pre == pf)
187  {
188  break;
189  }
190  }
191
192  if(strcmp(pb->name,name) == 0)
193  {
194
195  return pb;
196  }
197  else if(strcmp(pf->name,name)==0)
198  {
199  return pf;
200  }
201  }
202
203  return NULL;
204 }
205
206 STU* delete_link(STU *head, int num)
207 {
208  //判断链表是否存在
209  if(head == NULL)
210  {
211  printf("link not found\n");
212  return head;
213  }
214  else
215  {
216  STU *pb = head;//指向头节点
```

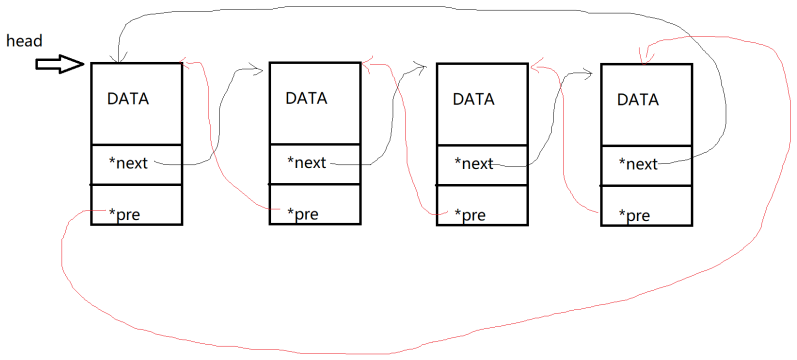
```
217  STU *pf = head->pre;//指向尾节点
218
219  //逐个节点寻找删除点
220  while((pb->num != num) && (pf->num != num) && (pf != pb))
221  {
222  pb = pb->next;//next方向移动
223  pf = pf->pre;//pre方向移动
224  if(pb->pre == pf)
225  break;
226  }
227
228  if(pb->num == num)//删除pb指向的节点
229  {
230
231  if(pb == head)//删除头节点
232  {
233  if(head == head->next)
234  {
235  free(pb);
236  head = NULL;
237  }
238  else
239  {
240  head->next->pre = head->pre;
241  head->pre->next = head->next;
242  head = head->next;
243  free(pb);
244  }
245
246  }
247  else//删除中尾部节点
248  {
249  pb->pre->next = pb->next;
250  pb->next->pre = pb->pre;
251  free(pb);
252  }
253  }
254  else if(pf->num == num)//删除pf指向的节点
255  {
256
```

```
257  if(pf == head)//删除头节点
258  {
259  if(head == head->next)
260  {
261  free(pf);
262  head = NULL;
263  }
264  else
265  {
266  head->next->pre = head->pre;
267  head->pre->next = head->next;
268  head = head->next;
269  free(pf);
270  }
271  }
272  else//删除中尾部节点
273  {
274  pf->pre->next = pf->next;
275  pf->next->pre = pf->pre;
276  free(pf);
277  }
278  }
279  else
280  {
281  printf("未找到%d相关的节点信息",num);
282  }
283  }
284
285  return head;
286 }
287
288 STU* free_link(STU *head)
289 {
290  if(head == NULL)//链表为空
291  {
292  printf("link not found\n");
293  return NULL;
294  }
295  else//链表存在
296  {
```

```
297   STU *pb = head;
298   STU *tmp;
299   do
300   {
301   tmp = pb;
302   pb = pb->next;
303   free(tmp);
304   }while(pb != head);
305   }
306
307   return NULL;
308 }
```

1、认识双向 循环 链表

```
typedef struct stu
{
   int num;
   char name[32];
   float score;

  struct stu *next;
  struct stu *pre;
}STU;
```

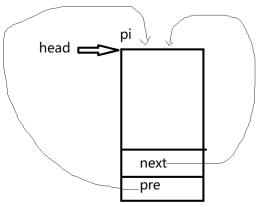


2、双链表 插入

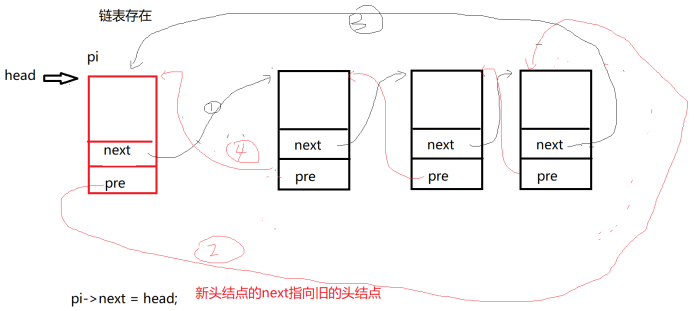
```
typedef struct stu
{
   int num;
   char name[32];
   float score;

  struct stu *next;
  struct stu *pre;
}STU;
```

链表不存在

```
head = pi;
pi->next = head;
pi->pre = head;
```



链表存在




```
pi->next = head;    新头结点的next指向旧的头结点
pi->pre = head->pre;    新头结点的pre指向 尾结点
head->pre->next = pi;    让尾结点的next 指向 新的头结点

head->pre = pi;    旧的头结点pre指向新的头
head = pi;    head指向新的头
```

```
1  //头部之前插入
2  STU* insert_link(STU *head, STU tmp)
3  {
4   //1、为插入的节点pi申请空间
5   STU *pi = (STU *)calloc(1,sizeof(STU));
6   //2、将tmp的值 赋值给 *pi
```

```
 *pi = tmp;

 //3、判断链表是否存在
  if(head == NULL)//链表不存在
  {
  head = pi;
  pi->next = head;
  pi->pre = head;
  }
  else//链表存在(头部之前插入)
  {
  pi->next = head;
  pi->pre = head->pre;
  head->pre->next = pi;
  head->pre = pi;
  head = pi;
  }

  return head;
}
```

3、链表的遍历

单向遍历

```
void print_link(STU *head)
{
  //判断链表是否存在
  if(head == NULL)
  {
  printf("link not found\n");
  return;
  }
  else//链表存在
  {
  STU *pb = head;
  do
  {
  //访问节点内容
  printf("num=%d, name=%s, age=%d\n", pb->num,pb->name,pb->age);
  //pb指向下一个节点
```

```
 pb = pb->next;
 }while(pb != head);
 }
 return;
}
```

双向遍历：1-1

```
void print_link(STU *head)
{
 //判断链表是否存在
 if(head == NULL)
 {
 printf("link not found\n");
 }
 else//链表存在
 {
  STU *pb = head;//pb指向头结点
  STU *pf = head->pre;//pf指向了尾节点

  do
  {
  if(pb == pf)//相遇 只需要打印pf或pb中任何一个信息就够了
  {
  printf("num=%d,name=%s,age=%d\n", pb->num,pb->name,pb->age);
  break;
  }
  printf("num=%d,name=%s,age=%d\n", pb->num,pb->name,pb->age);
  printf("num=%d,name=%s,age=%d\n", pf->num,pf->name,pf->age);

  pb = pb->next;//next方向的移动
  pf = pf->pre;//pre方向的移动
  }while( pb->pre != pf );//pf和pb不能 擦肩而过

  }
}
```

4、双向链表的查找

```
STU* search_link(STU *head, char *name)
{
```

```
//判断链表是否存在
if(head == NULL)
{
return NULL;
}
else//链表存在
{
 STU *pb = head;//指向头结点
 STU *pf = head->pre;//指向的是尾节点
 printf("pb = %p\n", pb);
 printf("pf = %p\n", pf);
 printf("pb->pre = %p\n", pb->pre);


 while( (strcmp(pb->name,name) != 0) && (strcmp(pf->name,name)!=0) && (pb != pf) )
 {
 printf("##pb->name=%s##\n",pb->name);
 printf("##pf->name=%s##\n",pf->name);
 pb=pb->next;//next方向移动
 pf=pf->pre;//pf方向移动

 if(pb->pre == pf)
 {
 break;
 }
 }

 if(strcmp(pb->name,name) == 0)
 {

 return pb;
 }
 else if(strcmp(pf->name,name)==0)
 {
 return pf;
 }
 }

 return NULL;
}
```

```
43
```

5、双向链表 删除指定节点1-2

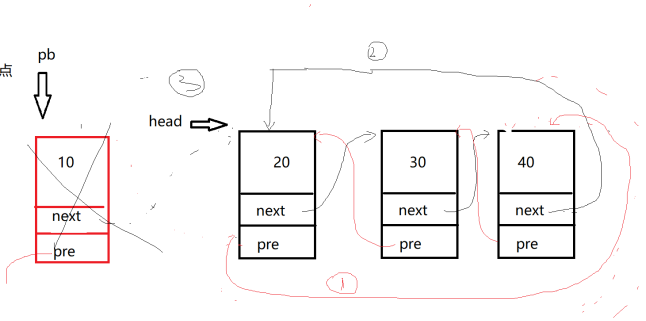
1、删除头结点：

① head->next->pre = head->pre; 新的头节点pre 指向尾节点

② head->pre->next = head->next;
尾节点的next 指向 新的头

③ head = head->next;
head指向 新的头节点

④ free(pb)

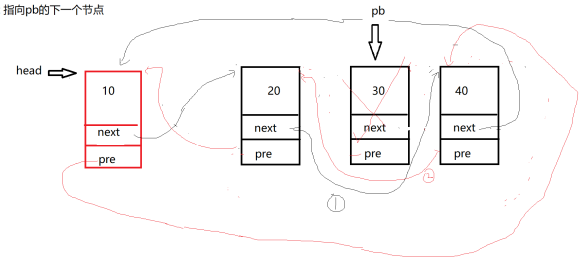


2、删除中部、尾节点

① pb->pre->next = pb->next; 让pb的上一个节点的next 指向pb的下一个节点

② pb->next->pre = pb->pre;
让pb的下一节点的pre 指向 pb的上一个节点

③ free(pb)




```
1  STU* delete_link(STU *head, int num)
2  {
3   //判断链表是否存在
4   if(head == NULL)
5   {
6   printf("link not found\n");
7   return head;
8   }
9   else
10   {
11   STU *pb = head;//指向头节点
12   STU *pf = head->pre;//指向尾节点
13
14   //逐个节点寻找删除点
15   while((pb->num != num) && (pf->num != num) && (pf != pb))
16   {
17   pb = pb->next;//next方向移动
18   pf = pf->pre;//pre方向移动
19   if(pb->pre == pf)
20   break;
21   }
```

```
22
23  if(pb->num == num)//删除pb指向的节点
24  {
25
26  if(pb == head)//删除头节点
27  {
28  if(head == head->next)//链表只有一个节点
29  {
30  free(pb);
31  head = NULL;
32  }
33  else
34  {
35  head->next->pre = head->pre;
36  head->pre->next = head->next;
37  head = head->next;
38  free(pb);
39  }
40
41  }
42  else//删除中尾部节点
43  {
44  pb->pre->next = pb->next;
45  pb->next->pre = pb->pre;
46  free(pb);
47  }
48  }
49  else if(pf->num == num)//删除pf指向的节点
50  {
51
52  if(pf == head)//删除头节点
53  {
54  if(head == head->next)//链表只有一个节点
55  {
56  free(pf);
57  head = NULL;
58  }
59  else
60  {
61  head->next->pre = head->pre;
```

```
 head->pre->next = head->next;
 head = head->next;
 free(pf);
 }
 }
 else//删除中尾部节点
 {
 pf->pre->next = pf->next;
 pf->next->pre = pf->pre;
 free(pf);
 }
 }
 else
 {
 printf("未找到%d相关的节点信息",num);
 }
 }

 return head;
}
```

6、释放这个链表节点（了解）

```
STU* free_link(STU *head)
{
 if(head == NULL)//链表为空
 {
 printf("link not found\n");
 return NULL;
 }
 else//链表存在
 {
 STU *pb = head;
 STU *tmp;
 do
 {
 tmp = pb;
 pb = pb->next;
 free(tmp);
 }while(pb != head);
```

```
 }

 return NULL;
}
```

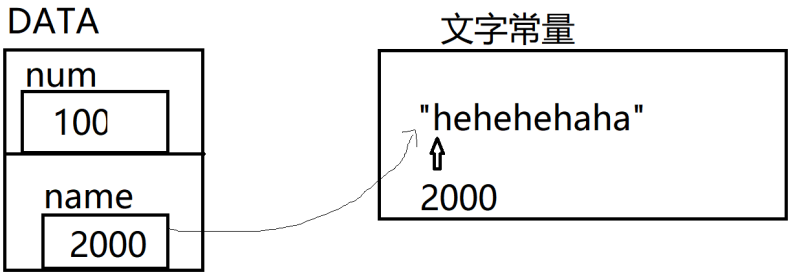
知识点3【结构的浅拷贝 和深拷贝】（了解）

1、知识点的引入（指针变量 作为 结构体的成员）

```
typedef struct
{
    int num;
    char *name;
}DATA;
```

32位8字节

DATA data={100,"hehehehaha"};



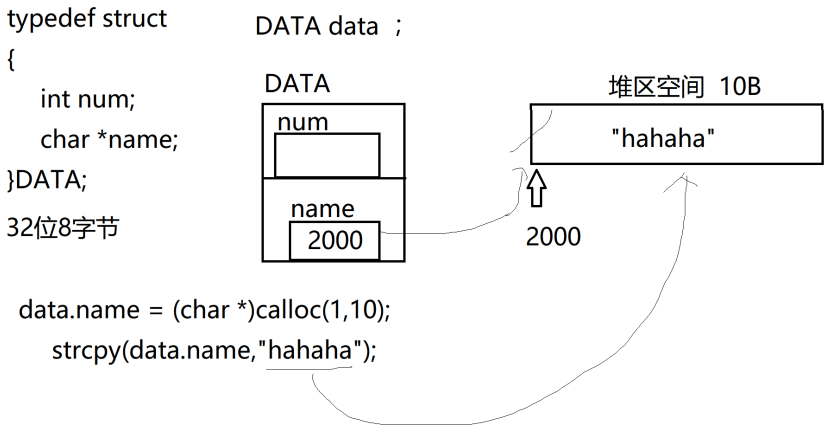

```c
typedef struct
{
 int num;
 char *name;//指针变量作为 结构体的成员
}DATA;
void test01()
{
 DATA data={100,"hehehehaha"};
 printf("%d\n",sizeof(DATA));//8字节

  printf("num = %d\n",data.num);
  //指针变量作为结构体的成员 保存的是空间的地址
  printf("name = %s\n",data.name);
}
```

2、指针变量 作为结构体的成员 操作前 必须有合法的空间

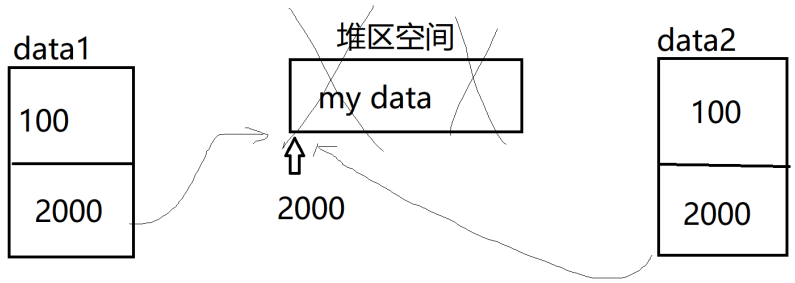
```c
void test02()
{
 DATA data;
 printf("%d\n",sizeof(DATA));

 printf("num = %d\n",data.num);
```

```
7   //指针变量 作为结构体的成员 操作前 必须有合法的空间
8   //data.name = "hehe";
9   //给name 事先申请 一块 堆区空间
10   data.name = (char *)calloc(1,10);
11   strcpy(data.name,"hahaha");
12   printf("name = %s\n",data.name);
13
14   //如果name指向堆区空间 一定要记得释放
15   if(data.name != NULL)
16   {
17   free(data.name);
18   data.name = NULL;
19   }
20  }
```

原理图分析：



3、指针变量 作为结构体的成员 结构体变量间的赋值操作 容易导致“浅拷贝”发生




```
if(data1.name != NULL)
{
    free(data1.name);
    data1.name = NULL;
}
```

data2 = data1;

```
if(data2.name != NULL)
{
    free(data2.name);
    data2.name = NULL;
}
```

```c
void test03()
{
 DATA data1;
 DATA data2;

 data1.num = 100;
 data1.name = (char *)calloc(1,10);
 strcpy(data1.name,"my data");
 printf("data1:num = %d, name = %s\n",data1.num, data1.name);

 //指针变量 作为结构体的成员 结构体变量间的赋值操作 容易导致“浅拷贝”发生
 data2 = data1;//“浅拷贝”
 printf("data2: num = %d, name = %s\n",data2.num, data2.name);

 if(data1.name != NULL)
 {
 free(data1.name);
 data1.name = NULL;
 }

 if(data2.name != NULL)
 {
 free(data2.name);
 data2.name = NULL;
 }
}
```

运行结果 出现段错误