

### 知识点1【链表的概述】（了解）

---

- 1、数组和链表的优缺点
  - 2、链表的概述
- 

### 知识点2【静态链表】（了解）

---

- 1、设计链表节点
- 

### 知识点3【学生管理系统】（了解）

---

- 1、typedef 给结构体类型取别名
  - 2、工程的main函数的设计
  - 3、链表插入节点值----头部之前插入
  - 4、遍历链表
  - 5、链表的尾部插入
  - 6、有序插入
  - 7、查找链表指定节点
  - 8、删除链表指定节点
  - 9、释放链表
  - 10、链表翻转
  - 11、链表排序
- 

### 知识点4【双向循环链表】（了解）

---

- 1、概述
- 2、完整代码

## 知识点1【链表的概述】（了解）

### 1、数组和链表的优缺点

静态数组：int arr[5]; 必须事先确定数组元素的个数，过多浪费 过小容易溢出，删除插入数据效率低（需要移动大量的数据）

动态数组：不需要事先知道元素的个数，在使用中动态申请，删除插入数据效率低（需要移动大量的数据）

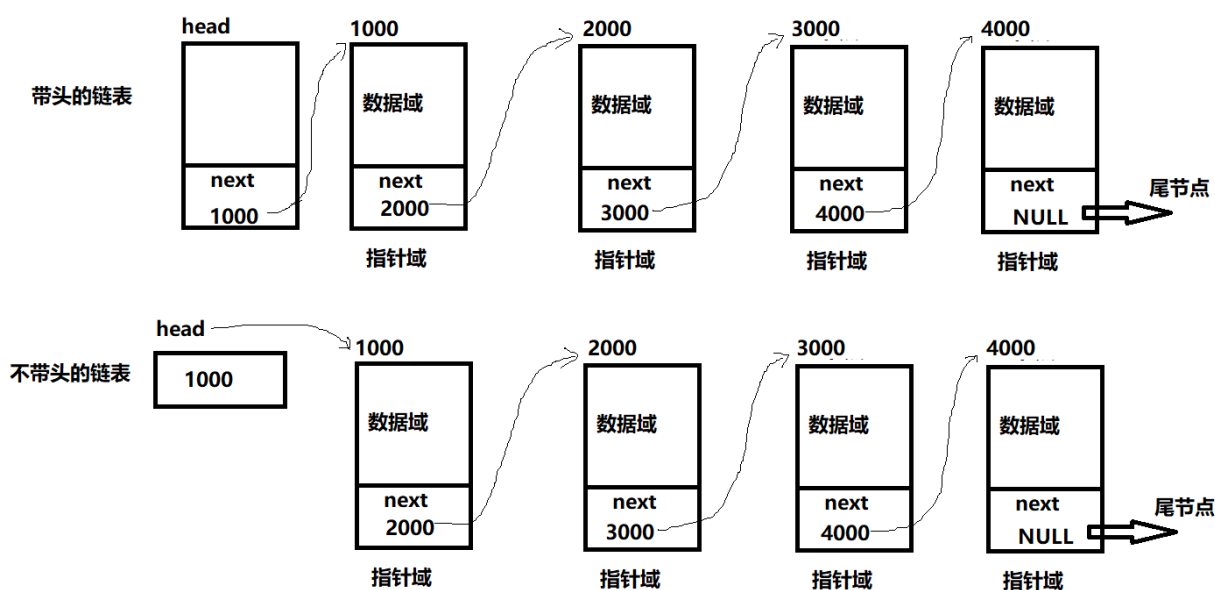
（数组优点：遍历元素效率高）

链表：不需要事先知道数据的个数，在使用中动态申请，插入删除不需要移动数据

（链表缺点：遍历效率低）

## 2、链表的概述

链表是由一个个**节点**组成，节点没有名字，每个节点从堆区动态申请，节点间**物理上**是非连续的，但是每个节点通过**指针域**保存下一个节点的位置 达到**逻辑上**连续。



## 知识点2【静态链表】（了解）

### 1、设计链表节点

```
1 #include <stdio.h>
2
3 struct stu
4 {
5     //数据域
6     int num;
7     char name[32];
8
9     //指针域
10    struct stu *next;
11 };
12 void test01()
13 {
```

```

14     struct stu node1 = {100, "lucy", NULL};
15     struct stu node2 = {101, "bob", NULL};
16     struct stu node3 = {102, "tom", NULL};
17     struct stu node4 = {103, "德玛", NULL};
18     struct stu node5 = {104, "小法", NULL};
19
20     //定义链表头
21     struct stu *head = &node1;
22     node1.next = &node2;
23     node2.next = &node3;
24     node3.next = &node4;
25     node4.next = &node5;
26     node5.next = NULL;
27
28     //遍历
29     struct stu *pb = head;
30     while (pb != NULL)
31     {
32         //访问数据
33         printf("%d %s\n", pb->num, pb->name);
34
35         //pb移动到下一个节点位置
36         pb = pb->next;
37     }
38 }
39 int main(int argc, char const *argv[])
40 {
41     test01();
42     return 0;
43 }
44

```

## 知识点3 【学生管理系统】（了解）

### 1、typedef 给结构体类型取别名

```
struct name_long
{
    int a;
    short b;
} DATA2; //DATA2是结构体变量名
typedef struct name_long
{
    int a;
    short b;
} DATA1; //DATA1是类型
void test02()
{
    struct name_long ob1;
    DATA1 ob2;
}
```

```
struct name_long_long
{
    int a;
    short b;
};
typedef struct name_long_long DATA3;
```

```

typedef struct d
{
    int a;
    short b;
} D_TYPE, *D_POINTER;

void test02()
{
    //D_TYPE ==> struct d
    D_TYPE ob1 = {100, 50};

    //D_POINTER ==> struct d *
    D_POINTER p = &ob1;
    printf("%d %hd\n", p->a, p->b);
}

```

```

edu@edu: ~/work/c/day08
edu@edu: ~/work/c/day08$ sudo gcc 00_code.c
edu@edu: ~/work/c/day08$ ./a.out
100 50
edu@edu: ~/work/c/day08$ _

```

## 2、工程的main函数的设计

```

1 #include <stdio.h>
2 #include <string.h>
3 void help(void)
4 {
5     printf("*****\n");
6     printf("*help:帮助信息          *\n");
7     printf("*insert:插入链表节点        *\n");
8     printf("*print:遍历链表节点         *\n");
9     printf("*search:查询链表某个节点     *\n");
10    printf("*delete:删除链表某个节点    *\n");
11    printf("*free:释放整个链表          *\n");
12    printf("*quit:退出程序              *\n");
13    printf("*****\n");
14 }
15 int main(int argc, char const *argv[])
16 {
17     help();
18     while (1)
19     {
20         char cmd[128] = "";
21         printf("请输入操作命令:");
22         scanf("%s", cmd);
23
24         if (strcmp(cmd, "help") == 0)
25         {
26             help();
27         }

```

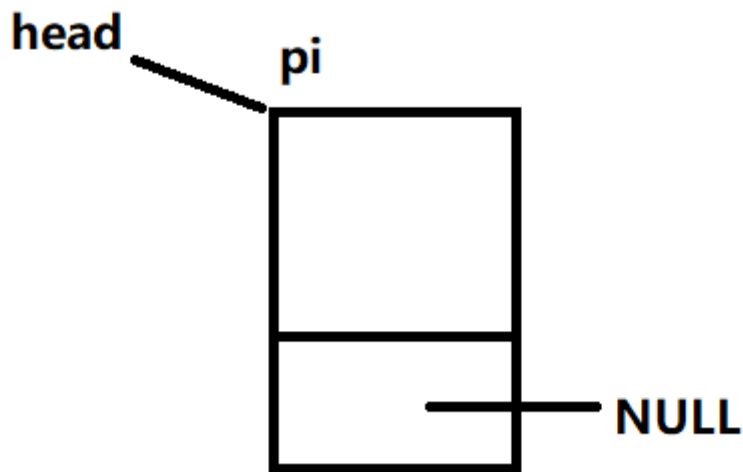
```

28     else if (strcmp(cmd, "insert") == 0)
29     {
30         printf("-----链表插入-----\n");
31     }
32     else if (strcmp(cmd, "print") == 0)
33     {
34         printf("-----链表遍历-----\n");
35     }
36     else if (strcmp(cmd, "search") == 0)
37     {
38         printf("-----链表查询-----\n");
39     }
40     else if (strcmp(cmd, "delete") == 0)
41     {
42         printf("-----删除链表指定节点-----\n");
43     }
44     else if (strcmp(cmd, "free") == 0)
45     {
46         printf("-----释放链表-----\n");
47     }
48     else if (strcmp(cmd, "quit") == 0)
49     {
50         break;
51     }
52 }
53 return 0;
54 }

```

### 3、链表插入节点值----头部之前插入

如果链表不存在



如果链表存在：

```
1 //头部之前插入
2 STU *insert_link(STU *head, STU tmp)
3 {
4     //为待插入的数据申请 空间
5     STU *pi = (STU *)calloc(1, sizeof(STU));
6     if (pi == NULL)
7     {
8         perror("calloc");
9         exit(-1); //结束进程
10    }
11    //将tmp数据赋值到 *pi
12    *pi = tmp;
13    pi->next = NULL;
14
15    //判断链表是否存在
16    if (head == NULL) //不存在
17    {
18        head = pi;
19        //return head;
20    }
21    else //存在
22    {
23        pi->next = head;
24        head = pi;
25        //return head;
26    }
27
28    return head;
```

## 4、遍历链表

```
1 void print_link(STU *head)
2 {
3     //判断链表是否存在
4     if(head == NULL)
5     {
6         printf("link not exists\n");
7         return;
8     }
9     else
10    {
11        STU *pb = head;
12        while(pb != NULL)
13        {
14            printf("%d %s %f\n", pb->num, pb->name, pb->score);
15            pb = pb->next;
16        }
17    }
18    return;
19 }
```

## 5、链表的尾部插入

```
1 //尾部插入
2 STU *insert_link(STU *head, STU tmp)
3 {
4     //为待插入的数据申请 空间
5     STU *pi = (STU *)calloc(1, sizeof(STU));
6     if (pi == NULL)
7     {
8         perror("calloc");
9         exit(-1); //结束进程
10    }
11    //将tmp数据赋值到 *pi
12    *pi = tmp;
13    pi->next = NULL;
14
15    //判断链表是否存在
16    if (head == NULL) //不存在
17    {
```



```

18     head = pi;
19     //return head;
20 }
21 else //存在
22 {
23     //寻找链表的尾节点
24     STU *pb = head;
25     while(pb->next != NULL)
26     {
27         pb = pb->next;
28     }
29
30     //pb就是尾节点
31     pb->next = pi;
32 }
33
34 return head;
35 }

```

## 6、有序插入

```

1 //有序插入
2 STU *insert_link(STU *head, STU tmp)
3 {
4     //为待插入的数据申请 空间
5     STU *pi = (STU *)calloc(1, sizeof(STU));
6     if (pi == NULL)
7     {
8         perror("calloc");
9         exit(-1); //结束进程
10    }
11    //将tmp数据赋值到 *pi
12    *pi = tmp;
13    pi->next = NULL;
14
15    //判断链表是否存在
16    if (head == NULL) //不存在
17    {
18        head = pi;
19        //return head;
20    }
21    else //存在

```

```

22     {
23         //寻找插入点的位置
24         STU *pb = head, *pf = head;
25         while ((pb->num < pi->num) && (pb->next != NULL))
26         {
27             pf = pb;
28             pb = pb->next;
29         }
30
31         if (pb->num >= pi->num) //头部，中部插入
32         {
33             if (head == pb) //头部之前插入
34             {
35                 pi->next = head;
36                 head = pi;
37             }
38             else //中部插入
39             {
40                 pf->next = pi;
41                 pi->next = pb;
42             }
43         }
44         else //尾部插入
45         {
46             pb->next = pi;
47         }
48     }
49
50     return head;
51 }

```

## 7、查找链表指定节点

```

1  STU *search_link(STU *head, char *name)
2  {
3      //判断链表是否存在
4      if (NULL == head)
5      {
6          printf("link not exist\n");
7          return NULL;
8      }
9      else //链表存在

```

```

10     {
11         STU *pb = head;
12         while ((strcmp(pb->name, name) != 0) && (pb->next != NULL))
13         {
14             pb = pb->next;
15         }
16
17         //找到
18         if (strcmp(pb->name, name) == 0)
19         {
20             return pb;
21         }
22     }
23
24     printf("未找到相关数据\n");
25     return NULL;
26 }

```

## 8、删除链表指定节点

```

1  STU* delete_link(STU *head, int num)
2  {
3      //判断链表是否存在
4      if(NULL == head)
5      {
6          printf("link not exist\n");
7          return head;
8      }
9      else
10     {
11         //查找删除的点
12         STU *pb=head, *pf = head;
13         while((pb->num != num)&&(pb->next != NULL))
14         {
15             pf = pb;
16             pb = pb->next;
17         }
18
19         if(pb->num == num)//找到
20         {
21             //判断删除点的位置
22             if(pb == head)//删除头节点

```

```

23         {
24             head=head->next;
25             //free(pb);
26         }
27         else//删除中尾部节点
28         {
29             pf->next = pb->next;
30             //free(pb);
31         }
32         free(pb);
33         printf("已成功删除num=%d的节点\n", num);
34     }
35     else//未找到
36     {
37         printf("未找到需要删除的节点\n");
38     }
39 }
40
41 return head;
42 }

```

## 9、释放链表

```

1  STU* free_link(STU* head)
2  {
3      //判断链表是否存在
4      if(NULL == head)
5      {
6          printf("link not exist\n");
7      }
8      else
9      {
10         STU *pb = head;
11         while(pb!=NULL)
12         {
13             head = pb->next;
14             free(pb);
15             pb = head;
16         }
17     }
18     return NULL;
19 }

```

## 10、链表翻转

```
1  STU* reverse_link(STU *head)
2  {
3      //判断链表是否存在
4      if(NULL == head)
5      {
6          printf("link not exist\n");
7          //return head;
8      }
9      else
10     {
11         STU *pb = head->next;
12         STU *pn = NULL;
13         head->next = NULL;
14
15         while(pb != NULL)
16         {
17             //纪录pb下一个节点位置
18             pn = pb->next;
19             //pb连接上一个节点
20             pb->next = head;
21             head = pb;
22             pb = pn;
23         }
24
25         printf("链表翻转成功\n");
26     }
27
28     return head;
29 }
```

## 11、链表排序

```
1  void sort_link(STU *head)
2  {
3      //判断链表是否存在
4      if (NULL == head)
5      {
6          printf("link not exist\n");
7          return;
8      }
```

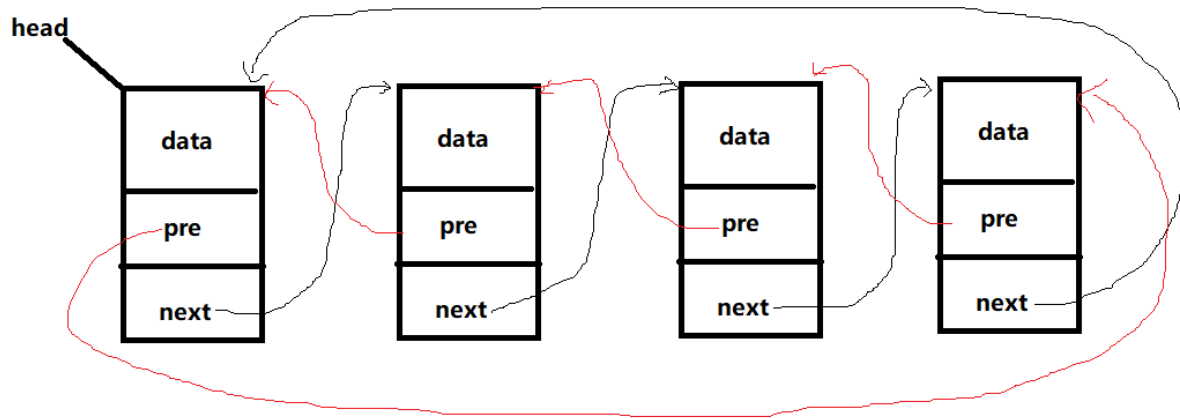
```

9      else
10     {
11         STU *p_i = head, *p_j = head; //int i=0,j=0;
12         while (p_i->next != NULL)      //for(i=0;i<n-1;i++)
13         {
14             STU *p_min = p_i;    //int min = i;
15             p_j = p_min->next;    //j=min+1;
16             while (p_j != NULL) //for(;j<n;j++)
17             {
18                 if (p_min->num > p_j->num) //if(arr[min] > arr[j])
19                     p_min = p_j;          //min = j;
20
21                 p_j = p_j->next; //j++
22             }
23             if (p_i != p_min) //if(i != min)
24             {
25                 //交换数据
26                 STU tmp = *p_i;
27                 *p_i = *p_min;
28                 *p_min = tmp;
29
30                 tmp.next = p_i->next;
31                 p_i->next = p_min->next;
32                 p_min->next = tmp.next;
33             }
34
35             p_i = p_i->next; //i++
36         }
37     }
38     return;
39 }

```

## 知识点4【双向循环链表】（了解）

### 1、概述



```
1 typedef struct stu
2 {
3     //数据域
4     int num;
5     char name[32];
6
7     //指针域
8     struct stu *pre;
9     struct stu *next;
10 }STU;
```

## 2、完整代码

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 typedef struct stu
5 {
6     //数据域
7     int num;
8     char name[32];
9
10    //指针域
11    struct stu *pre;
12    struct stu *next;
13 } STU;
14 STU *head = NULL;
15 void insert_link(STU **p_head, STU tmp);
16 void print_link(STU *head);
17 STU *search_link(STU *head, int num);
18 void delete_link(STU **p_head, int num);
```

```
19 void free_link(STU **p_head);
20 int main(int argc, char const *argv[])
21 {
22     int n = 0;
23     printf("请输入学生的个数:");
24     scanf("%d", &n);
25
26     int i = 0;
27     for (i = 0; i < n; i++)
28     {
29         printf("请输入第%d个学员的信息:", i + 1);
30         STU tmp;
31         scanf("%d %s", &tmp.num, tmp.name);
32
33         insert_link(&head, tmp);
34     }
35
36     //遍历链表
37     print_link(head);
38
39     //查询
40     printf("请输入你要查询的学号:");
41     int num = 0;
42     scanf("%d", &num);
43     STU *ret = search_link(head, num);
44     if (ret != NULL)
45     {
46         printf("查询的结果:%d %s\n", ret->num, ret->name);
47     }
48
49     //删除指定节点
50     printf("请输入你要删除的学号:");
51     scanf("%d", &num);
52
53     delete_link(&head, num);
54
55     //遍历链表
56     print_link(head);
57
58     //释放这个链表
59     free_link(&head);
```



```

60
61     //遍历链表
62     print_link(head);
63
64     return 0;
65 }
66
67 //尾插法
68 void insert_link(STU **p_head, STU tmp)
69 {
70     STU *head = *p_head;
71
72     //为插入的节点申请空间
73     STU *pi = (STU *)calloc(1, sizeof(STU));
74     *pi = tmp;
75     pi->next = NULL;
76     pi->pre = NULL;
77
78     //判断链表是否为空
79     if (NULL == head)
80     {
81         head = pi;
82         pi->next = pi;
83         pi->pre = pi;
84     }
85     else
86     {
87         head->pre->next = pi;
88         pi->next = head;
89         pi->pre = head->pre;
90         head->pre = pi;
91     }
92
93     //更新外部的head
94     *p_head = head;
95 }
96
97 void print_link(STU *head)
98 {
99     //判断链表是否存在

```

```

100     if (NULL == head)
101     {
102         printf("link not exist\n");
103         return;
104     }
105     else
106     {
107         STU *pn = head;
108         STU *pr = head->pre;
109
110         while (1)
111         {
112             if (pn == pr) //链表节点为奇数个
113             {
114                 printf("%d %s\n", pn->num, pn->name);
115                 break;
116             }
117             else if (pn->next == pr) ////链表节点为偶数个
118             {
119                 printf("%d %s\n", pn->num, pn->name);
120                 printf("%d %s\n", pr->num, pr->name);
121                 break;
122             }
123             else
124             {
125                 printf("%d %s\n", pn->num, pn->name);
126                 printf("%d %s\n", pr->num, pr->name);
127                 pn = pn->next;
128                 pr = pr->pre;
129             }
130         }
131     }
132     return;
133 }
134
135 STU *search_link(STU *head, int num)
136 {
137     //判断链表是否存在
138     if (NULL == head)
139     {

```

```

140         printf("link not exist\n");
141         return NULL;
142     }
143     else
144     {
145         STU *pn = head;
146         STU *pr = head->pre;
147
148         while ((pn->num != num) && (pr-
>num != num) && (pn != pr) && (pn->next != pr))
149         {
150             pn = pn->next;
151             pr = pr->pre;
152         }
153
154         if (pn->num == num)
155         {
156             return pn;
157         }
158         else if (pr->num == num)
159         {
160             return pr;
161         }
162         else
163         {
164             printf("没有找到相关节点\n");
165         }
166     }
167
168     return NULL;
169 }
170
171 #if 0
172 void delete_link(STU **p_head, int num)
173 {
174     STU *head = *p_head;
175
176     if (NULL == head)
177     {
178         printf("link not exist\n");
179         return;

```

```
180     }
181     else
182     {
183         STU *pn = head;
184         STU *pr = head->pre;
185
186         while ((pn->num != num) && (pr->num != num) && (pn != pr) && (pn->next != pr))
187         {
188             pn = pn->next;
189             pr = pr->pre;
190         }
191
192         if (pn->num == num) //头部、中部节点
193         {
194             if (pn == head) //删除头节点
195             {
196                 head->next->pre = head->pre;
197                 head->pre->next = head->next;
198                 head = head->next;
199                 //free(pn);
200             }
201             else //删除中部节点
202             {
203                 pn->pre->next = pn->next;
204                 pn->next->pre = pn->pre;
205                 //free(pn);
206             }
207             printf("成功删除节点:%d %s\n", pn->num, pn->name);
208             free(pn);
209         }
210         else if (pr->num == num) //尾部、中部
211         {
212             pr->pre->next = pr->next;
213             pr->next->pre = pr->pre;
214             printf("成功删除节点:%d %s\n", pr->num, pr->name);
215             free(pr);
216         }
217         else
218         {
```

```

219         printf("没有找到相关节点\n");
220     }
221 }
222
223     *p_head = head;
224 }
225 #endif
226
227 #if 1
228 void delete_link(STU **p_head, int num)
229 {
230     STU *head = *p_head;
231
232     if (NULL == head)
233     {
234         printf("link not exist\n");
235         return;
236     }
237     else
238     {
239         STU *pn = head;
240         STU *pr = head->pre;
241
242         while ((pn->num != num) && (pr->num != num) && (pn != pr) && (pn->next != pr))
243         {
244             pn = pn->next;
245             pr = pr->pre;
246         }
247
248         if (pn->num == num) //头部、中部节点
249         {
250             pn->next->pre = pn->pre;
251             pn->pre->next = pn->next;
252             if (pn == head) //删除头节点
253             {
254                 head = head->next;
255             }
256
257             printf("成功删除节点:%d %s\n", pn->num, pn->name);
258             free(pn);

```

```

259     }
260     else if (pr->num == num) //尾部、中部
261     {
262         pr->pre->next = pr->next;
263         pr->next->pre = pr->pre;
264         printf("成功删除节点:%d %s\n", pr->num, pr->name);
265         free(pr);
266     }
267     else
268     {
269         printf("没有找到相关节点\n");
270     }
271 }
272
273 *p_head = head;
274 }
275 #endif
276
277 void free_link(STU **p_head)
278 {
279     STU *head = *p_head;
280
281     if (NULL == head)
282     {
283         printf("link not exist\n");
284         return;
285     }
286     else
287     {
288         STU *pn = head;
289         do
290         {
291             head = head->next;
292             free(pn);
293             pn = head;
294         } while (pn != (*p_head));
295     }
296
297     *p_head = NULL;
298     return;

```

