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)

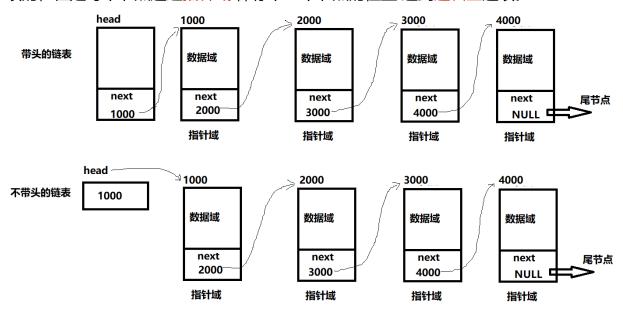
(数组优点:遍历元素效率高)

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

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

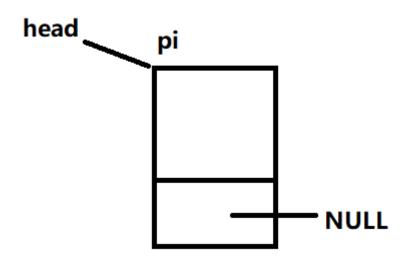
#### 2、工程的main函数的设计

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

```
else if (strcmp(cmd, "insert") == 0)
28
         {
29
            printf("------链表插入-----\n");
30
31
         else if (strcmp(cmd, "print") == 0)
32
34
            printf("-------链表遍历-----\n");
         else if (strcmp(cmd, "search") == 0)
36
37
            printf("-------链表查询-----\n");
38
39
         else if (strcmp(cmd, "delete") == 0)
40
41
         {
            42
43
         else if (strcmp(cmd, "free") == 0)
45
            46
47
         else if (strcmp(cmd, "quit") == 0)
48
49
50
            break;
51
52
      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 {
      //为待插入的数据申请 空间
      STU *pi = (STU *)calloc(1, sizeof(STU));
      if (pi == NULL)
6
      {
          perror("calloc");
8
          exit(-1); //结束进程
9
10
      //将tmp数据赋值到 *pi
11
      *pi = tmp;
12
       pi->next = NULL;
13
14
       //判断链表是否存在
15
       if (head == NULL) //不存在
16
17
          head = pi;
18
          //return head;
19
       }
20
       else //存在
21
22
           pi->next = head;
23
          head = pi;
24
          //return head;
25
26
       }
27
28
       return head;
```

29 }

### 4、遍历链表

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

### 5、链表的尾部插入

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

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

#### 6、有序插入

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

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

#### 7、查找链表指定节点

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

#### 8、删除链表指定节点

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

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

#### 9、释放链表

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

### 10、链表翻转

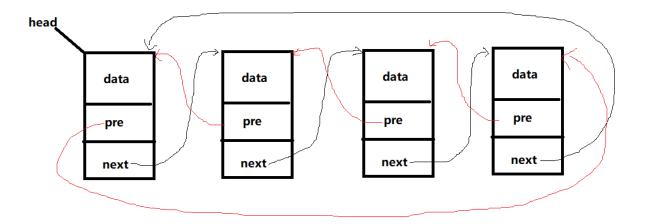
```
1 STU* reverse_link(STU *head)
2 {
      //判断链表是否存在
3
      if(NULL == head)
4
          printf("link not exist\n");
6
          //return head;
      }
8
      else
      {
10
           STU *pb = head->next;
11
           STU *pn = NULL;
12
           head->next = NULL;
13
14
15
          while(pb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

#### 11、链表排序

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

# 知识点4【双向循环链表】(了解)

## 1、概述



#### 2、完整代码

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 typedef struct stu
5 {
     //数据域
6
     int num;
     char name[32];
8
9
     //指针域
10
      struct stu *pre;
11
       struct stu *next;
12
13 } STU;
14 STU *head = NULL;
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 {
       int n = 0;
22
       printf("请输入学生的个数:");
23
       scanf("%d", &n);
24
25
       int i = 0;
26
       for (i = 0; i < n; i++)
27
28
           printf("请输入第%d个学员的信息:", i + 1);
29
           STU tmp;
30
           scanf("%d %s", &tmp.num, tmp.name);
           insert_link(&head, tmp);
34
       }
       //遍历链表
36
       print_link(head);
38
       //查询
39
       printf("请输入你要查询的学号:");
40
       int num = 0;
41
       scanf("%d", &num);
42
       STU *ret = search_link(head, num);
43
       if (ret != NULL)
44
45
           printf("查询的结果:%d %s\n", ret->num, ret->name);
46
47
       }
48
       //删除指定节点
49
       printf("请输入你要删除的学号:");
50
51
       scanf("%d", &num);
       delete_link(&head, num);
54
       //遍历链表
       print_link(head);
56
57
       //释放这个链表
58
       free_link(&head);
59
```

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

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

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

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

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

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