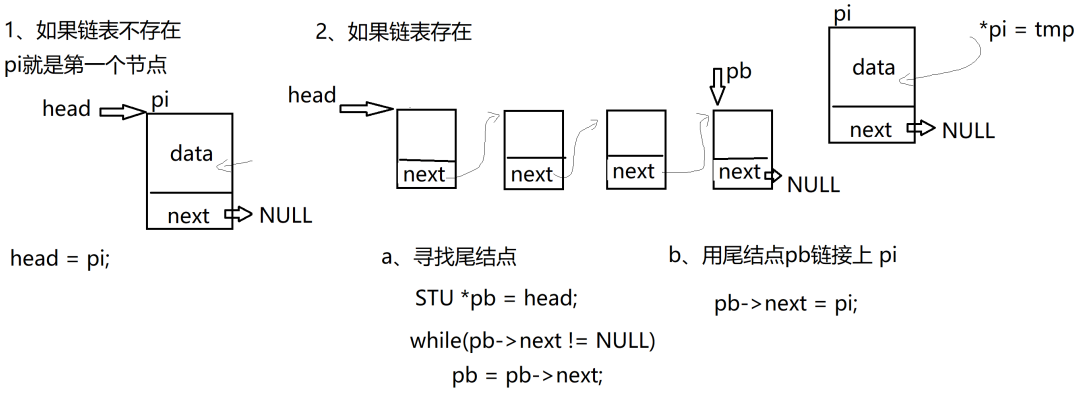
知识点1【链表的插入】0-1

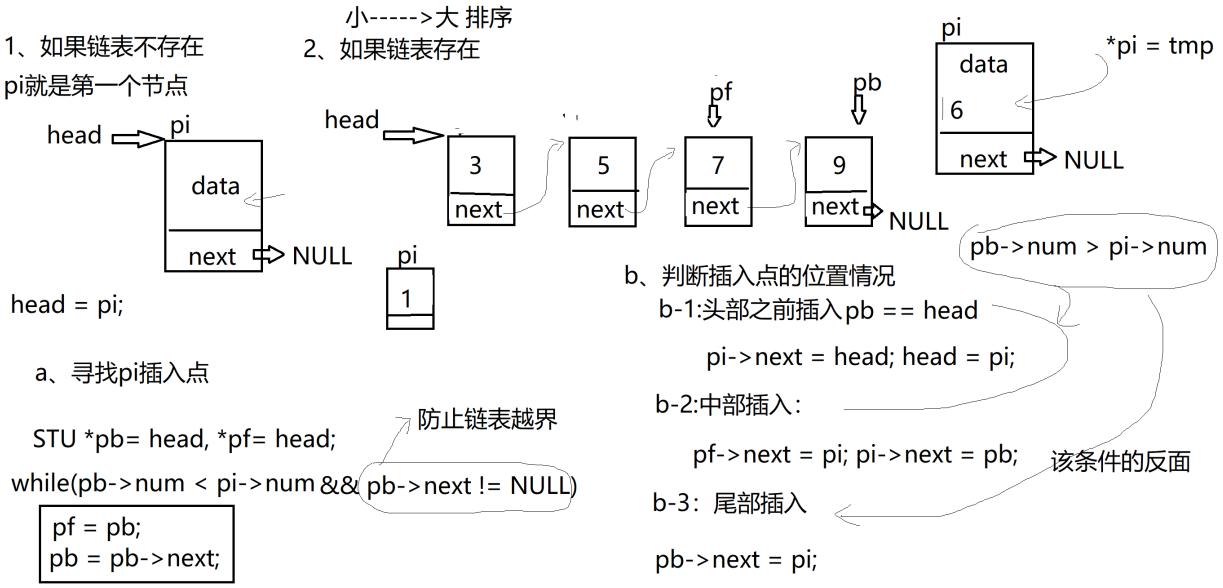
1、在链表的尾部插入




```
//链表的尾部插入
STU* insert_link(STU *head, STU tmp)
{
 //1、申请待插入的节点
```

```
5   STU *pi = (STU *)calloc(1,sizeof(STU));
6   if(pi == NULL)
7   {
8   perror(calloc);
9   return head;
10   }
11
12   //2、将tmp的数据 赋值到 *pi
13   *pi = tmp;
14   pi->next = NULL;
15
16   //3、将节点插入到链表的尾部
17   if(head == NULL)//链表不存在
18   {
19   head = pi;
20   return head;
21   }
22   else//链表存在
23   {
24   //a、寻找链表的尾节点
25   STU *pb = head;
26   while(pb->next != NULL)//如果不是尾节点
27   pb = pb->next;//pb就指向下一个节点
28
29   //b、用尾结点 pb 链接上 插入的节点pi
30   pb->next = pi;
31   return head;
32   }
33
34   return head;
35  }
```

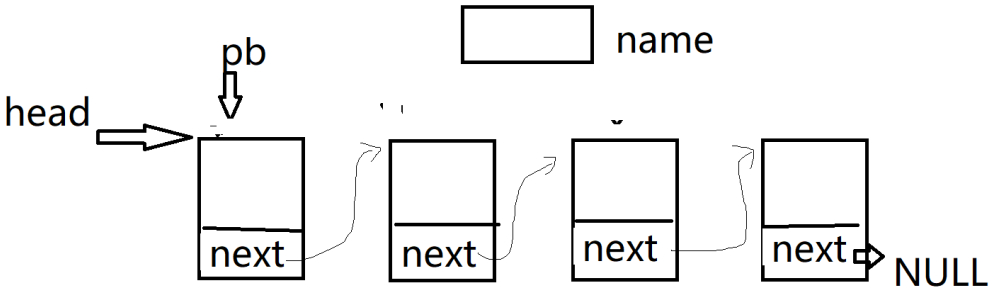
2、链表的有序插入（难度）0-2




```c
//链表的有序插入 以num的顺序为准（小--->大）
STU* insert_link(STU *head, STU tmp)
{
 //1、给待插入的节点pi 申请 堆区空间
 STU *pi = (STU *)calloc(1,sizeof(STU));
 if(pi == NULL)
 {
 perror("calloc");
 return head;
 }

 //2、将tmp的内容 赋值给 *pi
 *pi = tmp;
 pi->next = NULL;

 //3、链表节点pi的插入
 if(head == NULL)//链表不存在
 {
 head = pi;
 return head;
 }
 else//存在
 {
 //a、寻找插入点
 STU *pb = head, *pf = head;
 while(pb->num < pi->num && pb->next != NULL)
```

```
27  {
28  pf = pb;
29  pb = pb->next;
30  }
31
32  //b、插入点的判断
33  if(pb->num >= pi->num)//头部 中部插入
34  {
35  if(pb == head)//头部之前插入
36  {
37  pi->next = head;
38  head = pi;
39  return head;
40  }
41  else//中部插入
42  {
43  pf->next = pi;
44  pi->next = pb;
45  return head;
46  }
47  }
48  else//尾部插入
49  {
50  pb->next = pi;
51  return head;
52  }
53  }
54  return head;
55 }
```

知识点2【链表查询某个节点】 按姓名 查找



逐个节点查找：

```
STU *pb = head;
while(strcmp(pb->name, name) != 0 && pb->next != NULL)
        pb = pb->next;
if(strcmp(pb->name,name) == 0)//找到
    return pb;
else//没找
    return NULL;
```

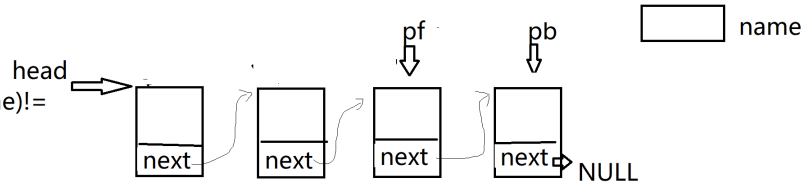
```c
STU* search_link(STU *head, char *name)
{
 //1、判断链表是否存在
 if(head == NULL)//不存在
 {
 printf("link not found\n");
 return NULL;
 }
 else//链表存在
  {
  STU *pb = head;

  //逐个将节点中的name 和 name比较 如果不相等 pb=pb->next
  while(strcmp(pb->name,name)!=0 && pb->next != NULL)
  pb = pb->next;

  //判断是否找到
  if(strcmp(pb->name,name)==0)//找到
  return pb;
  else//没找到
  return NULL;
  }
```

```
23
24   return NULL;
25  }
```

知识点3【删除链表指定节点】1-1

1、寻找删除点

```
while(strcmp(pb->name,name)!=
0 && pb->next != NULL)
{
    pf = pb;
    pb = pb->next;
}
```



2、找到删除点

```
strcmp(pb->name,name) == 0;
```

3、判断删除点的位置

3-1：删除头结点

条件：pb == head

```
head = pb->next;\
free(pb);
```

3-2:删除中节点

```
pf->next = pb->next;
free(pb);
```

3-3:删除尾部节点

```
pf->next = pb->next;
free(pb);
```

```
1  STU* detele_link(STU *head,char *name)
2  {
3   //1、判断链表是否存在
4   if(head == NULL)//不存在
5   {
6   printf("link not found\n");
7   return head;
8   }
9   else//存在
10   {
11   //2、寻找删除点
12   STU *pf=head, *pb = head;
13   while(strcmp(pb->name,name)!=0 && pb->next != NULL)
14   {
15   pf = pb;
16   pb = pb->next;
17   }
18
19   //3、找到删除点
20   if(strcmp(pb->name,name)==0)//找到删除点
21   {
22   //4、判断删除的位置
```

```
 if(pb == head)//删除头结点
 {
 head = pb->next;
 free(pb);

 }
 else//中部 或 尾部节点
 {
 pf->next = pb->next;
 free(pb);
 }
 printf("已成功删除%s的相关节点\n",name);
 return head;
 }
 else//没找到删除点
 {
 printf("链表中没有%s相关的数据节点信息\n",name);
 }
 }
 return head;
}
```

知识点4【链表的释放】

```
STU* free_link(STU *head)
{
 //判断链表是否存在
 if(head == NULL)
 {
 printf("link not found\n");
 return head;
 }
 else//链表存在
 {
 STU *pb = head;

 //逐个节点释放
 while(pb != NULL)
 {
 //head保存下一个节点的位置
```

```
17    head = pb->next;
18    //释放pb指向的节点
19    free(pb);
20    //pb 指向 head
21    pb = head;
22    }
23
24    printf("链表已经释放完毕\n");
25    return head;
26    }
27
28    return head;
29  }
```

回顾一下：

链表的插入：头部之前插入　尾部插入　有序插入

1、为插入的节点pi申请空间

2、将tmp的值赋值给*pi　　*pi = tmp

3、判断链表是否 存在

　3-1：不存在　　head = pi

　3-2:　存在　（尾部插入　有序插入）寻找插入点　安装具体的位置 插入节点

链表的遍历：

1、判断链表是否存在

　　1-1：不存在　不执行任何操作

　　1-2：存在　　逐个节点遍历　注意 别越界。

链表的查询：

1、判断链表是否存在

　　1-1：不存在　不执行任何操作

　　1-2：存在　　逐个节点比较　比较成功返回位置　　注意 别越界。

链表节点的删除：

1、判断链表是否存在

　　1-1：不存在　不执行任何操作

　　1-2：存在　　逐个节点比较　删除指定节点　　注意 别越界。

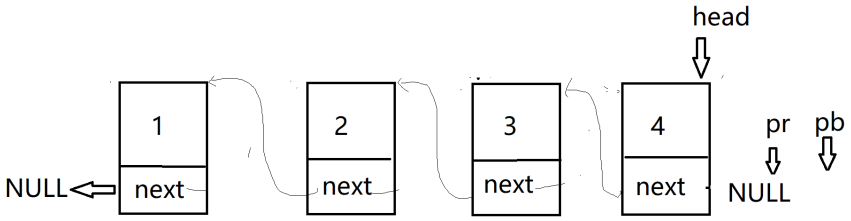
释放链表：

1、判断链表是否存在

1-1：不存在 不执行任何操作

1-2：存在　　逐个节点节点释放　　注意 别越界

知识点5【链表的逆序】1-2



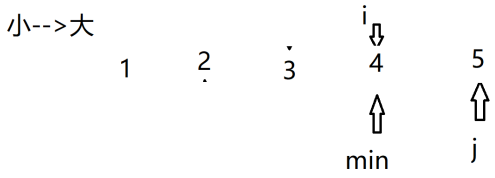

```
pb = head->next;
head->next = NULL;
while(pb != NULL)
{
   //记录pb->next     pr=pb->next;
   pb->next = head;
   head = pb;
   pb = pr;
```

```
1  STU* reverse_link(STU *head)
2  {
3   //判断链表是否存在
4   if(head == NULL)
5   {
6   printf("link not founf\n");
7   return head;
8   }
9   else//链表存在
10   {
11   //int *p,num;//p为int * ,  num为int
12   STU *pb,*pr;//pb为STU * , pr为STU *
13
14   //pb保存head->next（原因head->next会置NULL）
15   pb = head->next;
16   //将head->next置NULL （原因：头节点变尾节点）
17   head->next = NULL;
18
19   while(pb != NULL)
20   {
```

```
  //pr保存pb->next (原因:pb->next会指向head)
  pr = pb->next;

  //pb->next 指向 head （原因：逆转方向）
  pb->next = head;

  //保存 逆转方向的代码 可以重复 执行
  head = pb;
  pb = pr;
  }

  return head;
  }
  return head;
}
```

知识点6【链表的排序】

选择法排序：（以数组实现）



n=5

第0论:初始条件 min=0,j=min+1,；j<n;j++

第1论：初始条件min=1,j=min+1;j<n;j++

第2论：初始条件min=2，j=min+1;j<n;j++

第3论：初始条件min=3,j=min+1;j<n;j++

```
if(arr[min] > arr[j])
{
   min =j;
}
```

如果用i表示轮数:i 0~3　　i<n-1

i=0;i<n-1;i++ 外层循环条件。

第i轮：

min=i,j=min+1;j<n;j++ 内层循环条件

```
if(i != min)
{
  int tmp; tmp = arr[i];
arr[i]=arr[min];arr[min]=tmp;
```

```
#include<stdio.h>
int main()
{
 int arr[10]={0};
 int n = sizeof(arr)/sizeof(arr[0]);
 int i=0,j=0,min=0;

 printf("请输入%d个int数据\n",n);
 for(i=0;i<n;i++)
```

```
10  {
11  scanf("%d",arr+i);
12  }
13
14  //选择法排序
15  for(i=0;i<n-1; i++)
16  {
17  for(min=i,j=min+1; j<n;j++)
18  {
19  if(arr[min] > arr[j])
20  min = j;
21  }
22
23  if(min != i)
24  {
25  int tmp = 0;
26  tmp = arr[i];
27  arr[i]=arr[min];
28  arr[min]=tmp;
29  }
30
31  }
32
33  for(i=0;i<n;i++)
34  {
35  printf("%d ",arr[i]);
36  }
37  printf("\n");
38
39  return 0;
40 }
```

运行结果：

```
请输入10个int数据
9 7 8 4 5 6 2 1 3 0
0 1 2 3 4 5 6 7 8 9
Press any key to continue
```

选择法排序：（以链表实现）