知识点 1【链表的概述】

1、数组和链表的区别（背）

数组遍历、查询数据方便、但是插入、删除数据需要移动大量数据。

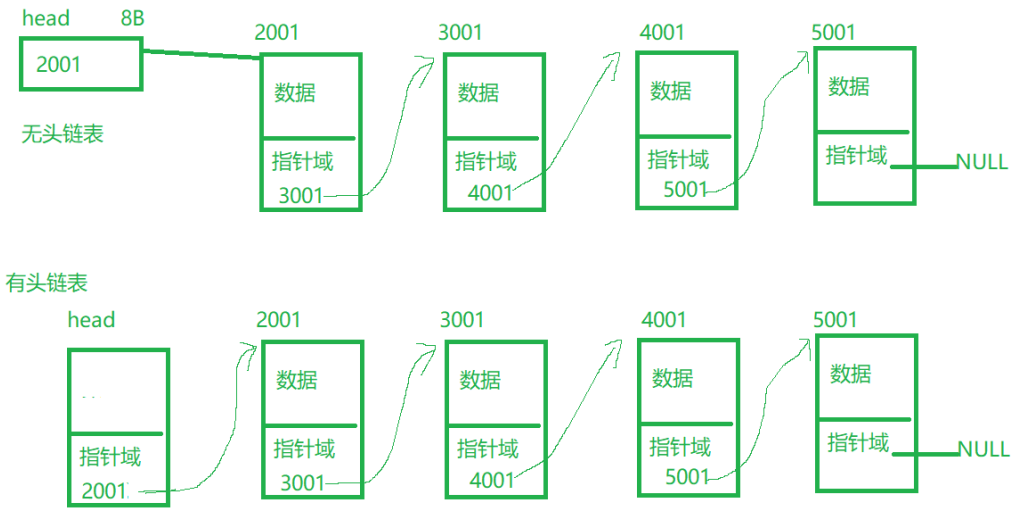
链表遍历、查询不方便，但是插入、删除数据不需要移动数据。

2、链表节点的概念

链表的节点 目的是用来保存数据信息，分为两部分：数据域、指针域

数据域：存放核心数据

指针域：存放下一个节点的地址。



3、链表节点类型定义

```
typedef struct stu

{

    //数据域

    int num;

    char name[32];

    float score;
```

```
    //指针域
    struct stu *next;
}STU;
```

知识点 2【静态链表--单向链表】

```
#include <stdio.h>
typedef struct stu
{
    // 数据域
    int num;
    char name[32];
    float score;

    // 指针域
    struct stu *next;
} STU;

int main(int argc, char const *argv[])
{
    // 局部节点
    STU node1 = {101, "lucy", 99.9f, NULL};
    STU node2 = {102, "bob", 99.9f, NULL};
```

```
    STU node3 = {103, "tom", 99.9f, NULL};
    STU node4 = {104, "德玛", 99.9f, NULL};
    STU node5 = {105, "小炮", 99.9f, NULL};

    // 定义一个链表头
    STU *head = &node1;
    node1.next = &node2;
    node2.next = &node3;
    node3.next = &node4;
    node4.next = &node5;

    // 遍历一个链表
    STU *pb = head;
    while (pb != NULL)
    {
        printf("%d %s %f\n", pb->num, pb->name, pb->score);
        pb = pb->next; // 指向下一个节点
    }
    return 0;
}
```

```
edu@edu:~/work/c/day09$ gcc 00_code.c
edu@edu:~/work/c/day09$ ./a.out
101 lucy 99.900002
102 bob 99.900002
103 tom 99.900002
104 德玛 99.900002
105 小炮 99.900002
edu@edu:~/work/c/day09$
```

知识点 3【以学生管理系统 练习动态链表】

1、主函数代码

```
#include <stdio.h>

#include <string.h>

#include "link.h"

void help(void)

{

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

    printf("*insert:插入链表节点              *\n");

    printf("*print:遍历链表                   *\n");

    printf("*search:查询链表的某个节点        *\n");

    printf("*delete:删除链表的某个节点        *\n");

    printf("*free:释放整个链表                *\n");

    printf("*quit:退出整个进程                *\n");

    printf("*reverse:链表逆序                 *\n");

    printf("*sort:链表排序                    *\n");
```

```
    printf("************************************\n");
    return;
}
int main(int argc, char const *argv[])
{
    //定义一个链表头
    STU *head=NULL;

    help();

    while (1)
    {
        printf("请输入操作指令:");
        char cmd[32] = "";
        scanf("%s", cmd);

        if (strcmp(cmd, "insert") == 0)
        {
            printf("--------insert-------------\n");
        }
        else if (strcmp(cmd, "print") == 0)
        {
```

```
    printf("--------print-------------\n");
}
else if (strcmp(cmd, "search") == 0)
{
    printf("--------search-------------\n");
}
else if (strcmp(cmd, "delete") == 0)
{
    printf("--------delete-------------\n");
}
else if (strcmp(cmd, "free") == 0)
{
    printf("--------free-------------\n");
}
else if (strcmp(cmd, "quit") == 0)
{
    return 0;
}
else if (strcmp(cmd, "reverse") == 0)
{
    printf("--------reverse-------------\n");
}
```

```
    else if (strcmp(cmd, "sort") == 0)
    {
        printf("--------sort------------\n");
    }
  }

  return 0;
}
```

2、链表头部之前插入节点

主函数添加的代码

```
while (1)
{
    printf("请输入操作指令:");
    char cmd[32] = "";
    scanf("%s", cmd);

    if (strcmp(cmd, "insert") == 0)
    {
        printf("请输入num name score:");
        STU tmp;//暂时获取学生信息
        scanf("%d %s %f", &tmp.num, tmp.name, &tmp.score);

        //调用链表插入函数
        head = insert_link(head,tmp);
    }
    else if (strcmp(cmd, "print") == 0)
```

link.c 中插入链表函数：

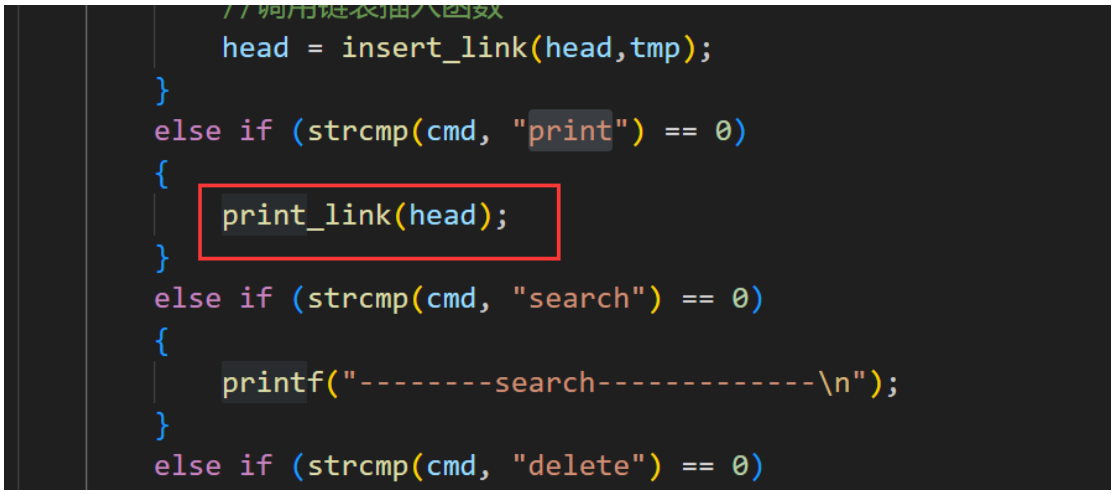
```
// 插入链表之 头部之前插入

STU *insert_link(STU *head, STU tmp)

{

  // 1、为节点申请空间
```

```
STU *pi = (STU *)malloc(sizeof(STU));
if (NULL == pi)
{
    printf("malloc\n");
    return head;
}

// 2、将键盘输入的 tmp 数据 赋值给 链表的 pi 节点
*pi = tmp;
pi->next = NULL;

// 3、判断链表是否存在
if (NULL == head)
{
    // 如果链表不存在 那么 pi 就是链表的第一个节点 直接 head 保存 pi 的值即可
    head = pi;
    // return head;
}
else
{
    pi->next = head; // pi 将会是新的头节点  pi->next 应该保存旧头节点的地址
    head = pi;       // 让 head 指向新的头节点
```

```
    // return head;
  }


  return head;
}
```

3、遍历链表

主函数添加代码：



```
        head = insert_link(head,tmp);
    }
    else if (strcmp(cmd, "print") == 0)
    {
        print_link(head);
    }
    else if (strcmp(cmd, "search") == 0)
    {
        printf("--------search------------\n");
    }
    else if (strcmp(cmd, "delete") == 0)
```

link.c 实现函数：

```
void print_link(STU *head)
{
  // 1、判断链表是否存在
  if (NULL == head)
  {
    printf("link not found\n");
    return;
  }
```

```
    else
    {
        STU *pb = head;    // 定义一个 pb 指针变量 指向头节点
        while (pb != NULL) // 逐个节点访问
        {
            // 遍历 pb 指向的节点
            printf("%d %s %f\n", pb->num, pb->name, pb->score);
            // pb 移动到下一个节点
            pb = pb->next;
        }
    }
    return;
}
```

4、尾部插入节点

```
//链表插入节点值尾部插入
STU *insert_link(STU *head, STU tmp)
{
    //1、为插入的数据申请堆区空间
    STU *pi = (STU *)malloc(sizeof(STU));
    if(NULL == pi)
    {
        printf("malloc\n");
```

```
    return head;
}

//2、将 tmp 数据 赋值给堆区空间 *pi
*pi=tmp;
pi->next =NULL;

//3、判断链表是否存在
if(NULL == head)//不存在
{
    head=pi;
    return head;
}
else//存在
{
    //4、寻找链表的尾部
    STU *pb= head;
    while(pb->next!=NULL)
        pb=pb->next;

    //5、在尾部 插入 pi 节点
    pb->next = pi;
```

```
    }


    return head;
}
```

5、有序插入

```
STU *insert_link(STU *head, STU tmp)
{
    //1、为插入的数据申请堆区空间
    STU *pi = (STU *)malloc(sizeof(STU));
    if(NULL == pi)
    {
        printf("malloc\n");
        return head;
    }


    //2、将 tmp 数据 赋值给堆区空间 *pi
    *pi=tmp;
    pi->next =NULL;


    //3、判断链表是否存在
    if(NULL == head)//不存在
    {
```

```
    head=pi;
    return head;
}
else//存在
{
    //4、寻找插入点
    STU *pf=head,*pb=head;
    while((pb->num<pi->num) &&(pb->next!=NULL))
    {
        pf=pb;
        pb=pb->next;
    }

    //5、判断插入点是头、中、尾
    if(pb->num>=pi->num)//头部 中部插入
    {
        if(pb==head)//头部
        {
            pi->next=head;
            head=pi;
        }
        else//中部
```

```
        {
            pf->next  = pi;
            pi->next = pb;
        }
    }
    else//尾部插入
    {
        pb->next = pi;
    }
  }

  return head;
}
```

6、查询指定节点

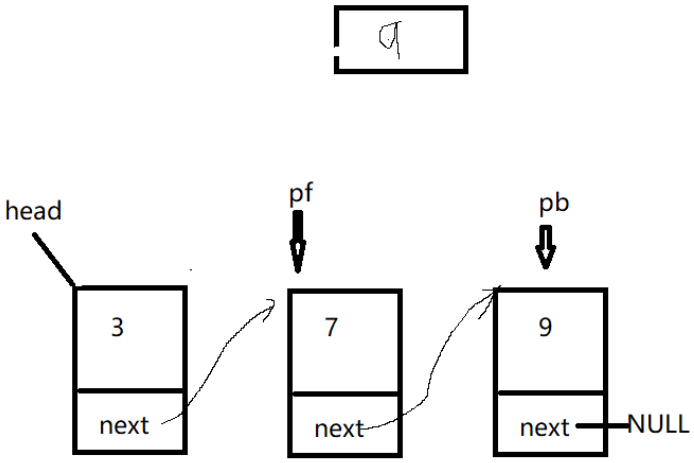
main 函数中：

link.c

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

{

    // 1、判断链表是否存在

    if (NULL == head)

    {

        printf("link not exits\n");

        return NULL;

    }

    else

    {

        // 逐个节点查询

        STU *pb = head;

        while ((strcmp(pb->name, name) != 0) && (pb->next != NULL))

            pb = pb->next;



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

            return pb;

        else

            printf("not found\n");

    }
```

```
    return NULL;

}
```

7、删除指定节点



头节点删除：

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

中部删除： 尾部删除：

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

main 中

```
    }
    else if (strcmp(cmd, "delete") == 0)
    {
        printf("请输入需要删除的序号:");
        int num = 0;
        scanf("%d", &num);

        head = delete_link(head,num);
    }
    else if (strcmp(cmd, "free") == 0)
    {
        printf("--------free-------------\n");
```

link.c

```
STU * delete_link(STU *head,int num)

{

    //判断链表是否存在
```

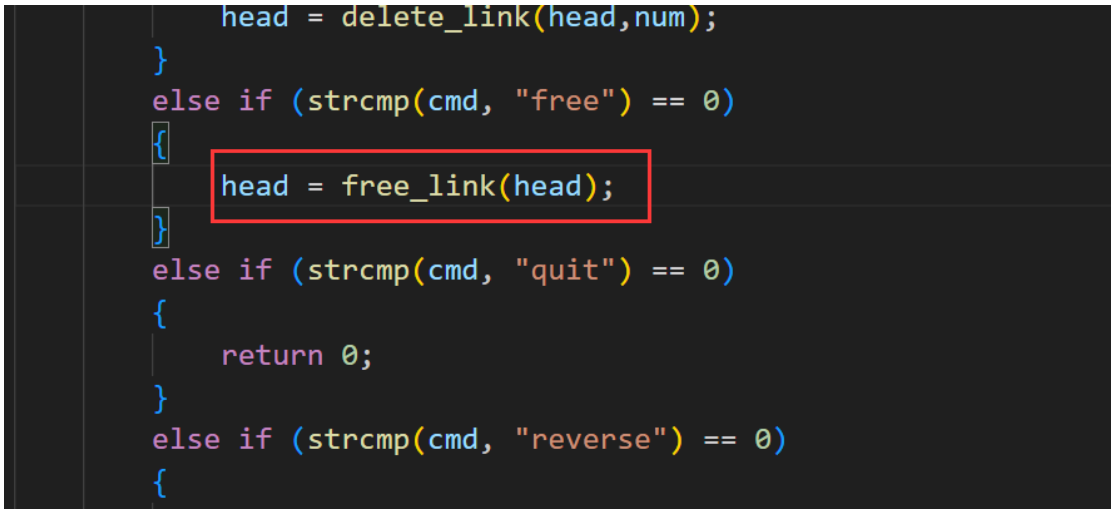
```
if(NULL == head)
{
    printf("link not exist\n");
    return head;
}
else
{
    //逐个节点寻找
    STU *pf=head,*pb=head;
    while((pb->num != num)&&(pb->next!=NULL))
    {
        pf=pb;
        pb=pb->next;
    }

    //判断删除点的位置
    if(pb->num == num)//找到
    {
        if(pb==head)//删除头部节点
        {
            head=head->next;
        }
```

```
            else//删除中尾部节点
            {
                pf->next=pb->next;
            }
            //释放 pb
            free(pb);
            printf("相应节点成功删除\n");
        }
        else
        {
            printf("未找到 num=%d 的相关节点\n",num);
        }
    }
    return head;
}
```

8、释放链表的所有节点

千万不要直接 free(head),而是逐个节点释放。释放完整个链表后一定要将 head 置 NULL

main 函数中：



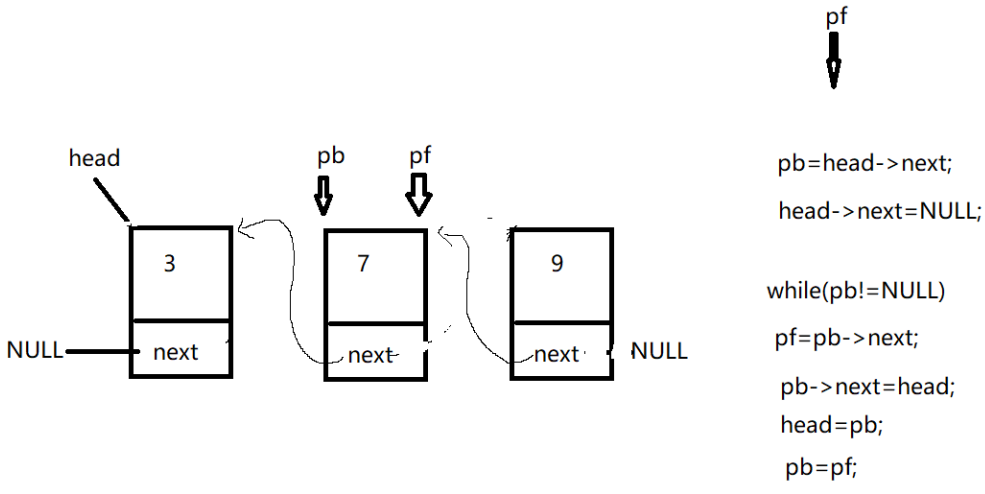
```
            head = delete_link(head,num);
        }
        else if (strcmp(cmd, "free") == 0)
        {
            head = free_link(head);
        }
        else if (strcmp(cmd, "quit") == 0)
        {
            return 0;
        }
        else if (strcmp(cmd, "reverse") == 0)
        {
```

link.c

```
STU *free_link(STU *head)

{

   // 判断链表是否存在

   if (NULL == head)

   {

      printf("link not exist\n");

      return head;

   }

   else

   {

      // 逐个节点释放

      STU *pb = head;

      while (pb != NULL)

      {

         // 纪录下一个节点位置
```

```
            head = head->next;
            // 释放 pb
            free(pb);
            // 让 pb 指向 head
            pb = head;
        }
        printf("整个链表释放完毕\n");
    }

    return NULL;
}
```

9、链表逆序



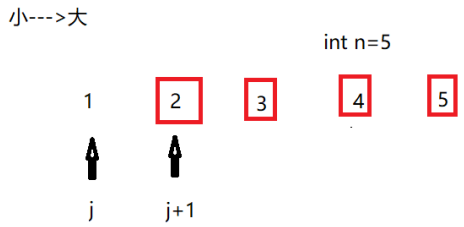

```
STU *reverse_link(STU *head)
{
```

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

    // 需要将 head->next 置 NULL 所以需要 pb 纪录 head->next
    pb = head->next;
    // 将 head->next 置 NULL
    head->next = NULL;

    // 逐个节点反向指向
    while (pb != NULL)
    {
        // 需要将 pb->next 指向前一个节点 需要 pf 纪录下一个节点的位置
        pf = pb->next;
        // pb 的 next 指向前一个
        pb->next = head;
```

```
            // 移动 head 到 pb
            head = pb;
            // pb 移动到 pf 处
            pb = pf;
        }
        printf("链表逆序成功\n");
    }

    return head;
}
```

知识点 4【数值数组的排序之冒泡排序】

冒泡排序是相邻两两比较。



第0轮：j=0;循环条件 j+1<n; 步进语句j++

第1轮：j=0;循环条件 j+1<n-1; 步进语句j++

第2轮：j=0;循环条件 j+1<n-2l 步进语句j++

第3轮 j=0;循环条件 j+1<n-3; 步进语句j++

外层循环：i=0;i<n-1;i++

内层循环：j=0;j<n-1-i;j++

交换条件：if(arr[j] > arr[j+1])

int tmp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=tmp;

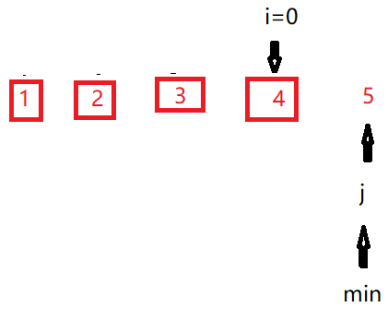
```
#include <stdio.h>

void sort_int_array01(int *arr, int n)
```

```
{
    int i = 0;
    for (i = 0; i < n - 1; i++)
    {
        int j = 0;
        for (j = 0; j < n - 1 - i; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                int tmp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = tmp;
            }
        }
    }
    return;
}
int main(int argc, char const *argv[])
{
    int arr[10] = {0};
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    printf("请输入%d 个 int 数值:", n);
    int i = 0;
    for (i = 0; i < n; i++)
    {
        scanf("%d", arr + i);
    }

    // 排序
    sort_int_array01(arr, n);

    for (i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

知识点 4【数值数组的排序之选择排序】

小--->大　　int n=5



第0轮：min=i,j=min+1;　j<n;j++

第1轮：min=i,j=min+1;　j<n;j++

第2轮：min=i,j=min+1;　j<n;j++

第3轮：min=i,j=min+1;　j<n;j++

外层循环：i=0;i<n-1;i++

内层循环：min=i,j=min+1;j<n;j++

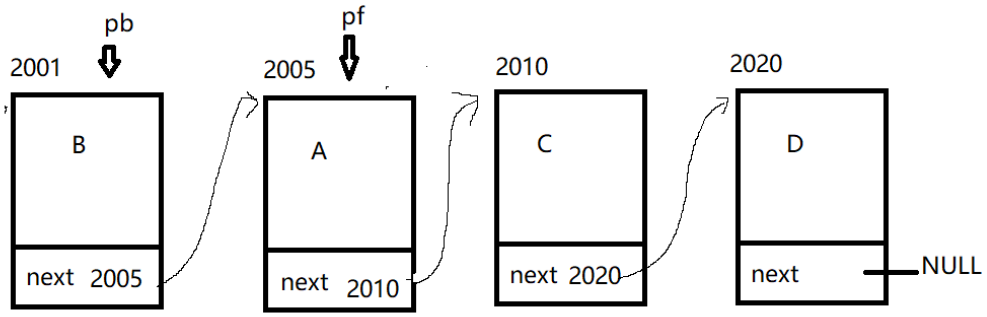
```
if(arr[min] > arr[j])
        min=j;
```

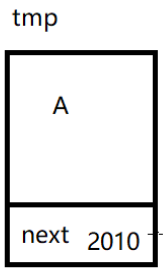
```
if(min != i)
{
    int tmp=arr[i];
    arr[i] = arr[min];
    arr[min] = tmp;
}
```

```
void sort_int_array02(int *arr, int n)
{
    int i = 0;
    for (i = 0; i < n - 1; i++)
    {
        int min, j;
        // 寻找最小值的下标
        for (min = i, j = min + 1; j < n; j++)
        {
            if (arr[min] > arr[j])
                min = j;
        }
```

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

知识点 5【链表节点的排序】





tmp=*pb;

*pb=*pf;

*pf= tmp;

节点整体交换：

数据域交换

指针域交换

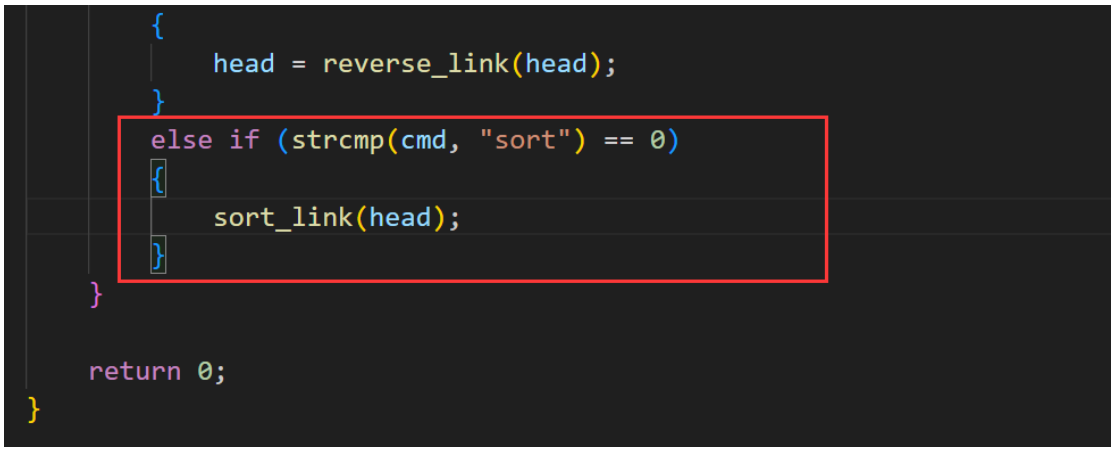
tmp.next = pb->next;

pb->next = pf->next;

pf->next = tmp.next;

指针域交换

在 main 函数中：



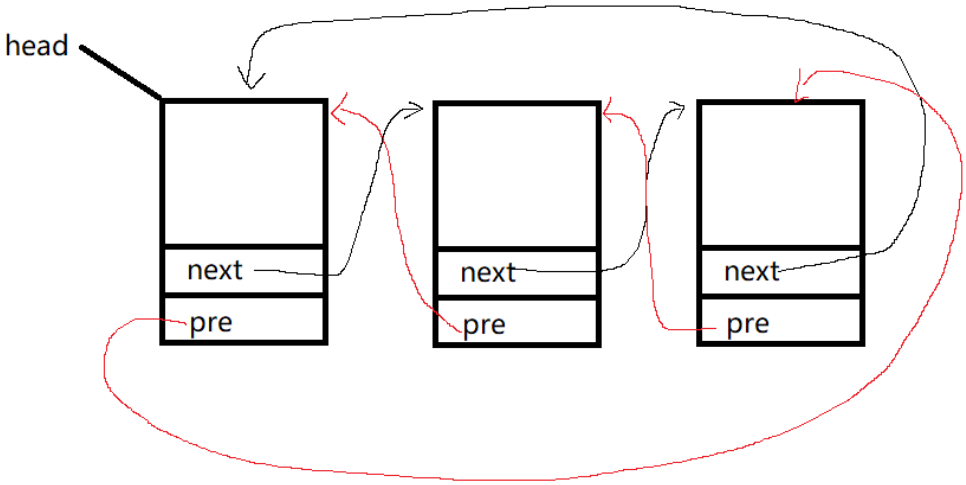

```
void sort_link(STU *head)

{

   // 判断链表是否存在

   if (NULL == head)

   {

      printf("link not exist\n");

      return;

   }

   else

   {

      // int i = 0;

      STU *p_i = NULL;



      // for ( i = 0; i < n-1; i++)

      for (p_i = head; p_i->next != NULL; p_i = p_i->next)

      {
```

```
// int min, j;
STU *p_min, *p_j;
// 寻找最小值的下标
// for (min = i, j = min + 1; j < n; j++)
for (p_min = p_i, p_j = p_min->next; p_j != NULL; p_j = p_j->next)
{
    // if (arr[min] > arr[j])
    if (p_min->num > p_j->num)
    {
        // min = j;
        p_min = p_j;
    }
}

// if (min != i)
if (p_min != p_i)
{
    // int tmp = arr[i];
    // arr[i] = arr[min];
    // arr[min] = tmp;
    STU tmp = *p_min;
    *p_min = *p_i;
```

```
            *p_i = tmp;



            tmp.next = p_min->next;

            p_min->next = p_i->next;

            p_i->next = tmp.next;

        }

    }

    printf("链表排序完成\n");

  }

}
```

知识点 6【双向循环链表】

1、双向循环链表

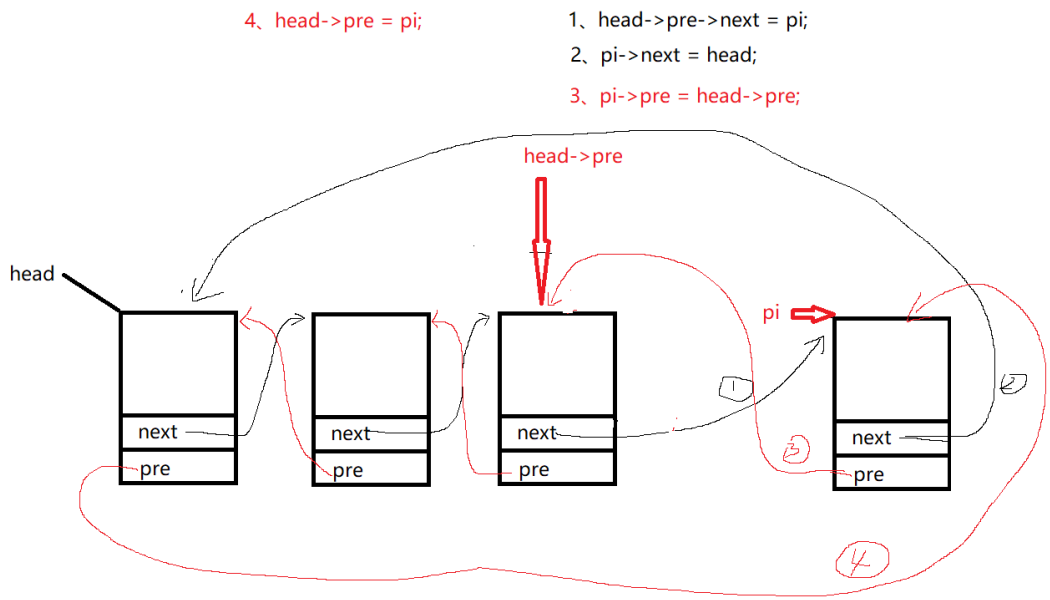


2、双向循环链表节点定义

```
struct stu

{
```

```
    //数据域
    int num;
    char name[32];
    float score;

    //指针域
    struct stu *next;
    struct stu *pre;
};
```

3、双向链表插入节点




```
STU *insert_double_link(STU *head, STU tmp)
{
    // 1、为插入的节点申请堆区空间
```

```
STU *pi = (STU *)malloc(sizeof(STU));
if (NULL == pi)
{
    printf("malloc error\n");
    return head;
}

// 2、将 tmp 的值 赋值给*pi
*pi = tmp;

// 判断链表是否存在
if (NULL == head)
{
    head = pi;
    head->next = head;
    head->pre = head;
    return head;
}
else
{
    head->pre->next = pi;
    pi->next = head;
```

```
        pi->pre = head->pre;
        head->pre = pi;
    }
    return head;
}
```

4、遍历双向循环链表

```
void print_double_link(STU *head)
{
    // 判断链表是否存在
    if (head == NULL)
    {
        printf("link not exist\n");
        return;
    }
    else
    {
        STU *pn = head;      // 在头节点
        STU *pr = head->pre; // 在尾节点
        while (1)
        {
            if (pn == pr) // 奇数个节点
            {
```

```
            printf("%d %s %f\n", pn->num, pn->name, pn->score);
            break;
        }
        else if (pn->next == pr) // 偶数个节点
        {
            printf("%d %s %f\n", pn->num, pn->name, pn->score);
            printf("%d %s %f\n", pr->num, pr->name, pr->score);
            break;
        }

        printf("%d %s %f\n", pn->num, pn->name, pn->score);
        printf("%d %s %f\n", pr->num, pr->name, pr->score);

        pn = pn->next;
        pr = pr->pre;
    }
  }
}
```

5、查找节点

```
STU *search_double_link(STU *head, int num)
{
  if (head == NULL)
```

```
    {
        printf("link not exist\n");
        return NULL;
    }
    else
    {
        STU *pn = head;
        STU *pr = head->pre;

        while ((pn->num != num) && (pr->num != num) && (pn != pr) && (pn->next != pr))
        {
            pn = pn->next;
            pr = pr->pre;
        }

        if (pn->num == num)
        {
            return pn;
        }
        else if (pr->num == num)
        {
```

```
            return pr;
        }
        else
        {
            printf("未找到相关节点\n");
        }
    }
    return NULL;
}
```

6、删除指定节点

```
STU *delete_double_link(STU *head, int num)
{
    if (head == NULL)
    {
        printf("link not exist\n");
        return NULL;
    }
    else
    {
        STU *pn = head;
        STU *pr = head->pre;
```

```
    while ((pn->num != num) && (pr->num != num) && (pn != pr) && (pn->next != pr))
    {
        pn = pn->next;
        pr = pr->pre;
    }

    if (pn->num == num) // 头、中删除
    {
        if (pn == head) // 头
        {
            head = head->next;
            pn->pre->next = head;
            head->pre = pn->pre;
            free(pn);
        }
        else // 中
        {
            pn->pre->next = pn->next;
            pn->next->pre = pn->pre;
            free(pn);
        }
```

```
            printf("删除指定节点\n");
        }
        else if (pr->num == num) // 中、尾删除
        {
            pr->pre->next = pr->next;
            pr->next->pre = pr->pre;
            free(pr);
            printf("删除指定节点\n");
        }
    }
    return head;
}
```

6、完整代码

```
#include <stdio.h>
#include <stdlib.h>
typedef struct stu
{
    // 数据域
    int num;
    char name[32];
    float score;
```

```
    // 指针域
    struct stu *next;
    struct stu *pre;
} STU;
STU *insert_double_link(STU *head, STU tmp);
void print_double_link(STU *head);
STU *search_double_link(STU *head, int num);
STU *delete_double_link(STU *head, int num);

STU *head = NULL;
int main(int argc, char const *argv[])
{
    int n = 0;
    printf("请输入学员的个数:");
    scanf("%d", &n);

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

```
    head = insert_double_link(head, tmp);
}

// 遍历链表
print_double_link(head);

// 查找指定节点
printf("请输入需要查找的学号:");
int num = 0;
scanf("%d", &num);

STU *ret = search_double_link(head, num);
if (ret != NULL)
{
    printf("查询结果:%d %s %f\n", ret->num, ret->name, ret->score);
}

// 删除指定节点
printf("请输入需要删除的学号:");
scanf("%d", &num);
head = delete_double_link(head, num);
```

```
    // 遍历链表
    print_double_link(head);


    return 0;
}
STU *insert_double_link(STU *head, STU tmp)
{
    // 1、为插入的节点申请堆区空间
    STU *pi = (STU *)malloc(sizeof(STU));
    if (NULL == pi)
    {
        printf("malloc error\n");
        return head;
    }


    // 2、将 tmp 的值 赋值给*pi
    *pi = tmp;


    // 判断链表是否存在
    if (NULL == head)
    {
        head = pi;
```

```
        head->next = head;
        head->pre = head;
        return head;
    }
    else
    {
        head->pre->next = pi;
        pi->next = head;
        pi->pre = head->pre;
        head->pre = pi;
    }
    return head;
}
void print_double_link(STU *head)
{
    // 判断链表是否存在
    if (head == NULL)
    {
        printf("link not exist\n");
        return;
    }
    else
```

```
{
    STU *pn = head;      // 在头节点
    STU *pr = head->pre; // 在尾节点
    while (1)
    {
        if (pn == pr) // 奇数个节点
        {
            printf("%d %s %f\n", pn->num, pn->name, pn->score);
            break;
        }
        else if (pn->next == pr) // 偶数个节点
        {
            printf("%d %s %f\n", pn->num, pn->name, pn->score);
            printf("%d %s %f\n", pr->num, pr->name, pr->score);
            break;
        }

        printf("%d %s %f\n", pn->num, pn->name, pn->score);
        printf("%d %s %f\n", pr->num, pr->name, pr->score);

        pn = pn->next;
        pr = pr->pre;
```

```
        }
    }
}

STU *search_double_link(STU *head, int num)
{
    if (head == NULL)
    {
        printf("link not exist\n");
        return NULL;
    }
    else
    {
        STU *pn = head;
        STU *pr = head->pre;

        while ((pn->num != num) && (pr->num != num) && (pn != pr) && (pn-
>next != pr))
        {
            pn = pn->next;
            pr = pr->pre;
        }
```

```
        if (pn->num == num)
        {
            return pn;
        }
        else if (pr->num == num)
        {
            return pr;
        }
        else
        {
            printf("未找到相关节点\n");
        }
    }
    return NULL;
}
STU *delete_double_link(STU *head, int num)
{
    if (head == NULL)
    {
        printf("link not exist\n");
        return NULL;
```

```
    }
    else
    {
        STU *pn = head;
        STU *pr = head->pre;

        while ((pn->num != num) && (pr->num != num) && (pn != pr) && (pn->next != pr))
        {
            pn = pn->next;
            pr = pr->pre;
        }

        if (pn->num == num) // 头、中删除
        {
            if (pn == head) // 头
            {
                head = head->next;
                pn->pre->next = head;
                head->pre = pn->pre;
                free(pn);
            }
```

```
        else // 中
        {
            pn->pre->next = pn->next;
            pn->next->pre = pn->pre;
            free(pn);
        }
        printf("删除指定节点\n");
    }
    else if (pr->num == num) // 中、尾删除
    {
        pr->pre->next = pr->next;
        pr->next->pre = pr->pre;
        free(pr);
        printf("删除指定节点\n");
    }
  }
  return head;
}-
```