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）

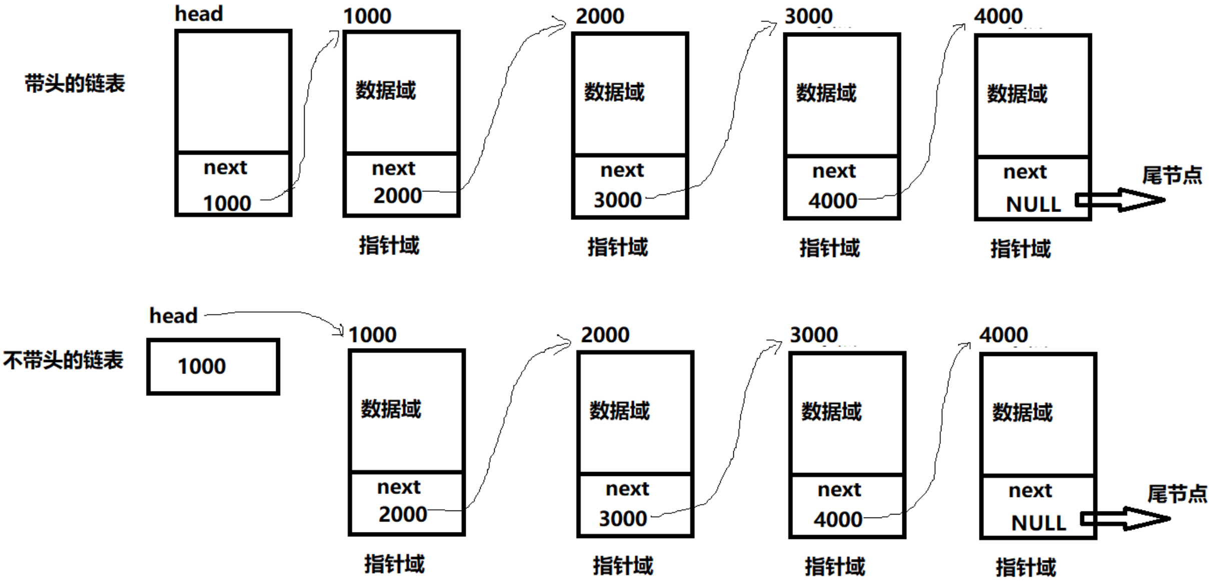
（数组优点：遍历元素效率高）

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。



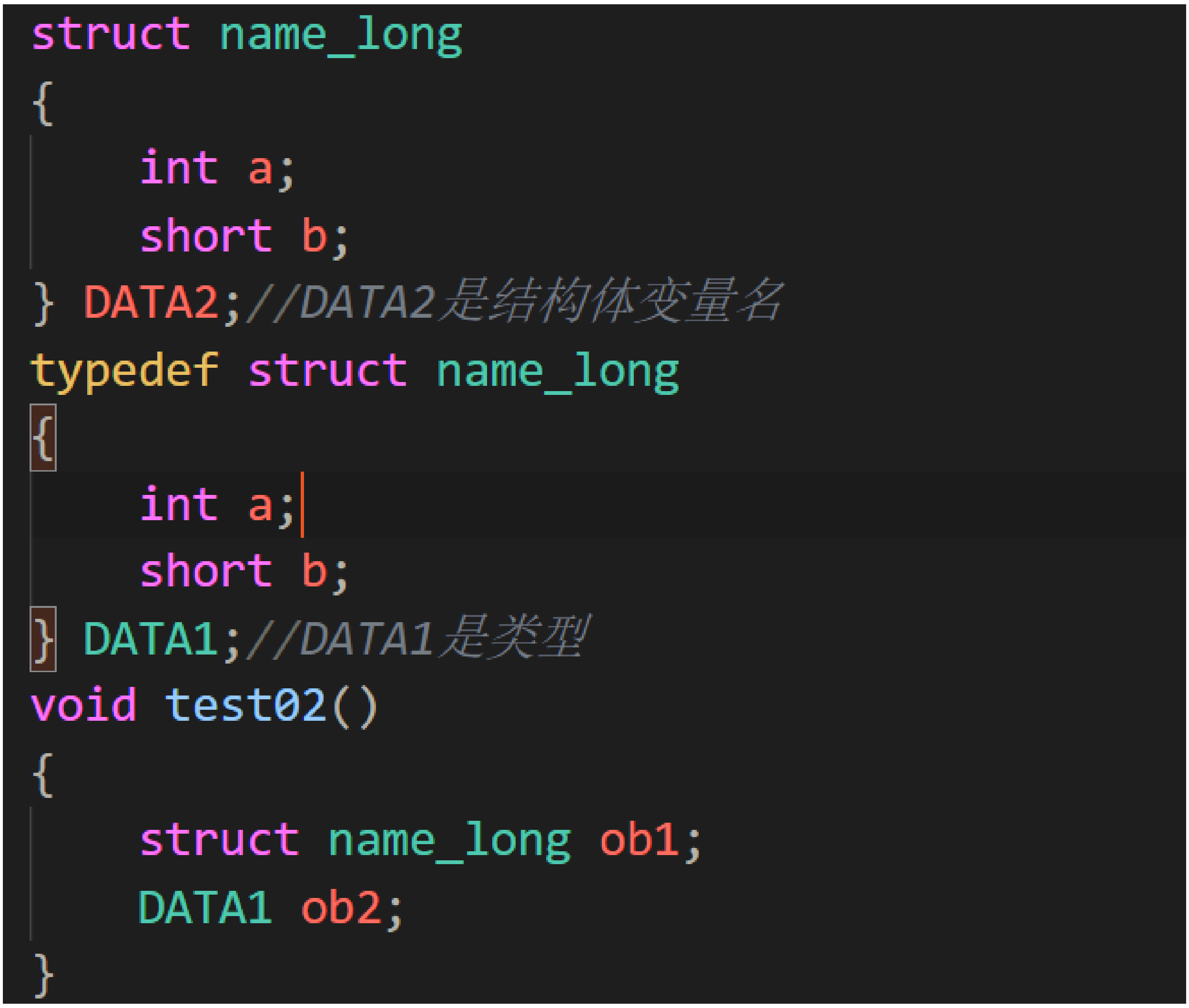
知识点2【静态链表】（了解）

1、设计链表节点

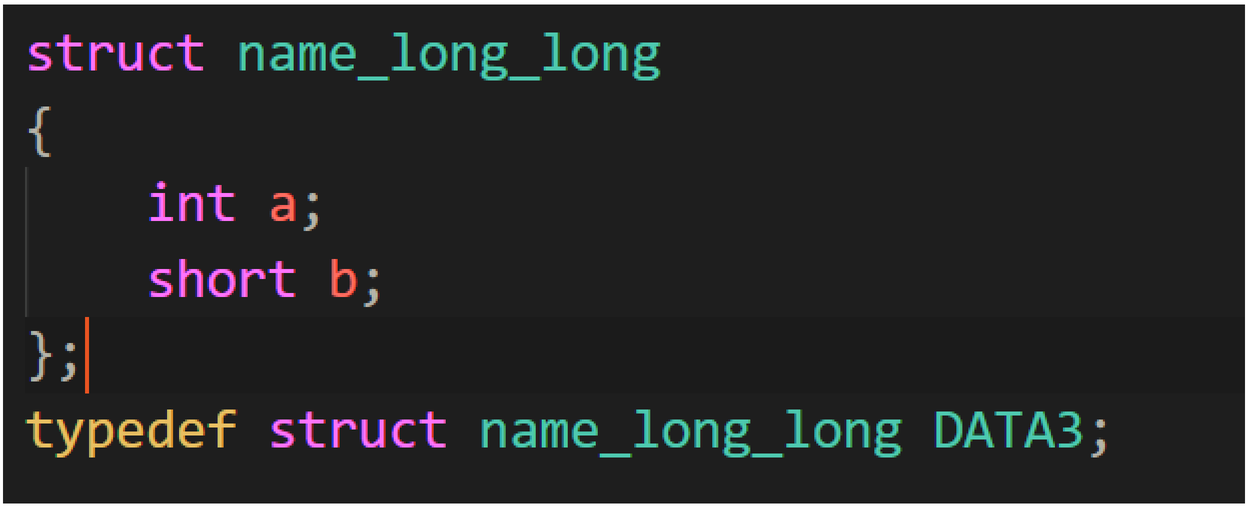
```
#include <stdio.h>

struct stu
{
    //数据域
    int num;
    char name[32];

    //指针域
     struct stu *next;
};
void test01()
{
     struct stu node1 = {100, "lucy", NULL};
     struct stu node2 = {101, "bob", NULL};
     struct stu node3 = {102, "tom", NULL};
     struct stu node4 = {103, "德玛", NULL};
     struct stu node5 = {104, "小法", NULL};

     //定义链表头
     struct stu *head = &node1;
     node1.next = &node2;
     node2.next = &node3;
     node3.next = &node4;
     node4.next = &node5;
     node5.next = NULL;

     //遍历
     struct stu *pb = head;
     while (pb != NULL)
     {
         //访问数据
         printf("%d %s\n", pb->num, pb->name);

         //pb移动到下一个节点位置
         pb = pb->next;
     }
}
int main(int argc, char const *argv[])
{
     test01();
     return 0;
}

```

知识点3【学生管理系统】（了解）

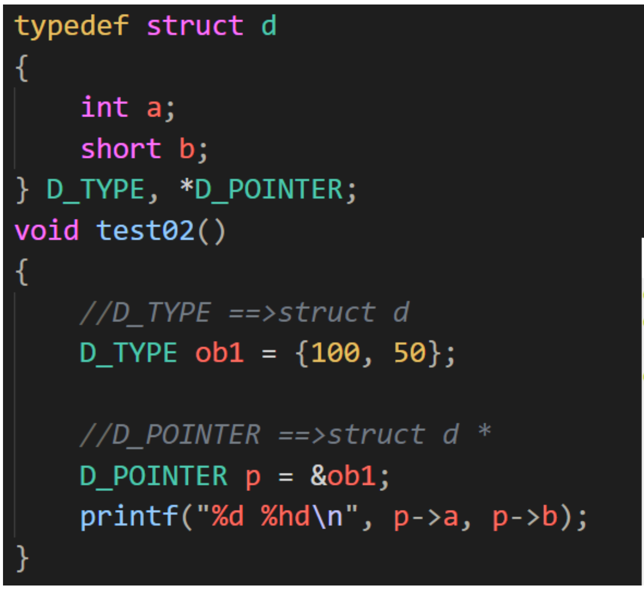
1、typedef 给结构体类型取别名



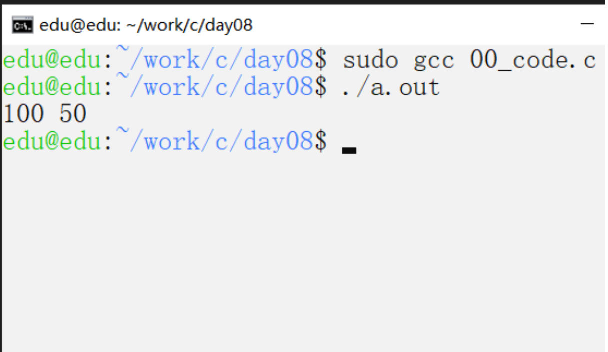
```c
struct name_long
{
    int a;
    short b;
} DATA2;//DATA2是结构体变量名
typedef struct name_long
{
    int a;
    short b;
} DATA1;//DATA1是类型
void test02()
{
    struct name_long ob1;
    DATA1 ob2;
}
```



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



```c
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);
}
```



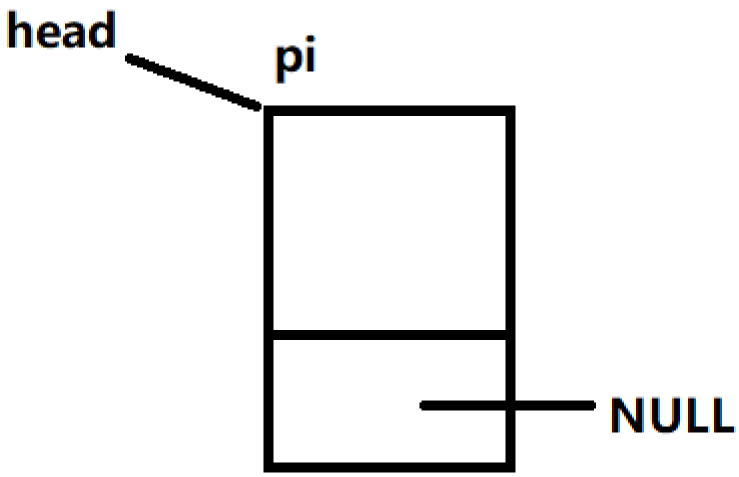
```
edu@edu:~/work/c/day08$ sudo gcc 00_code.c
edu@edu:~/work/c/day08$ ./a.out
100 50
edu@edu:~/work/c/day08$ 
```

2、工程的main函数的设计

```c
#include <stdio.h>
#include <string.h>
void help(void)
{
    printf("*******************************\n");
    printf("*help:帮助信息                *\n");
    printf("*insert:插入链表节点          *\n");
    printf("*print:遍历链表节点           *\n");
    printf("*search:查询链表某个节点      *\n");
    printf("*delete:删除链表某个节点      *\n");
    printf("*free:释放整个链表            *\n");
    printf("*quit:退出程序                *\n");
    printf("*******************************\n");
}
int main(int argc, char const *argv[])
{
    help();
    while (1)
    {
        char cmd[128] = "";
        printf("请输入操作命令:");
        scanf("%s", cmd);

        if (strcmp(cmd, "help") == 0)
        {
            help();
        }
        else if (strcmp(cmd, "insert") == 0)
        {
            printf("---------链表插入--------\n");
        }
        else if (strcmp(cmd, "print") == 0)
        {
            printf("---------链表遍历--------\n");
        }
        else if (strcmp(cmd, "search") == 0)
        {
            printf("---------链表查询--------\n");
        }
        else if (strcmp(cmd, "delete") == 0)
        {
            printf("---------删除链表指定节点--------\n");
        }
        else if (strcmp(cmd, "free") == 0)
        {
            printf("---------释放链表--------\n");
        }
        else if (strcmp(cmd, "quit") == 0)
        {
            break;
        }
    }
    return 0;
}
```

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：

```c
//头部之前插入
STU *insert_link(STU *head, STU tmp)
{
    //为待插入的数据申请 空间
    STU *pi = (STU *)calloc(1, sizeof(STU));
    if (pi == NULL)
    {
        perror("calloc");
        exit(-1); //结束进程
    }
    //将tmp数据赋值到 *pi
    *pi = tmp;
    pi->next = NULL;

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

    return head;
}
```

4、遍历链表

```
void print_link(STU *head)
{
    //判断链表是否存在
    if(head == NULL)
    {
        printf("link not exits\n");
        return;
    }
    else
     {
         STU *pb = head;
         while(pb != NULL)
         {
             printf("%d %s %f\n",pb->num, pb->name,pb->score);
             pb = pb->next;
         }
     }
     return;
}
```

5、链表的尾部插入

```
//尾部插入
STU *insert_link(STU *head, STU tmp)
{
    //为待插入的数据申请 空间
    STU *pi = (STU *)calloc(1, sizeof(STU));
    if (pi == NULL)
    {
        perror("calloc");
        exit(-1); //结束进程
     }
     //将tmp数据赋值到 *pi
     *pi = tmp;
     pi->next = NULL;

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

        //pb就是尾节点
        pb->next = pi;
    }

    return head;
}
```

6、有序插入

```
//有序插入
STU *insert_link(STU *head, STU tmp)
{
    //为待插入的数据申请 空间
    STU *pi = (STU *)calloc(1, sizeof(STU));
    if (pi == NULL)
    {
        perror("calloc");
        exit(-1); //结束进程
    }
    //将tmp数据赋值到 *pi
    *pi = tmp;
    pi->next = NULL;

    //判断链表是否存在
    if (head == NULL) //不存在
    {
        head = pi;
        //return head;
    }
    else //存在
    {
        //寻找插入点的位置
        STU *pb = head, *pf = head;
        while ((pb->num < pi->num) && (pb->next != NULL))
        {
            pf = pb;
            pb = pb->next;
        }

        if (pb->num >= pi->num) //头部，中部插入
        {
            if (head == pb) //头部之前插入
            {
                pi->next = head;
                head = pi;
            }
            else //中部插入
            {
                pf->next = pi;
                pi->next = pb;
            }
        }
        else //尾部插入
        {
            pb->next = pi;
        }
    }

    return head;
}
```

7、查找链表指定节点

```
STU *search_link(STU *head, char *name)
{
    //判断链表是否存在
    if (NULL == head)
    {
        printf("link not exist\n");
        return NULL;
    }
    else //链表存在
    {
        STU *pb = head;
        while ((strcmp(pb->name, name) != 0) && (pb->next != NULL))
        {
            pb = pb->next;
        }

        //找到
        if (strcmp(pb->name, name) == 0)
        {
            return pb;
        }
    }

    printf("未找到相关数据\n");
    return NULL;
}
```

8、删除链表指定节点

```
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
    {
        //查找删除的点
        STU *pb=head, *pf = head;
        while((pb->num != num)&&(pb->next != NULL))
        {
            pf = pb;
            pb = pb->next;
        }

        if(pb->num == num)//找到
        {
            //判断删除点的位置
            if(pb == head)//删除头节点
            {
                head=head->next;
                //free(pb);
            }
            else//删除中尾部节点
            {
                pf->next = pb->next;
                //free(pb);
            }
            free(pb);
            printf("已成功删除num=%d的节点\n", num);
        }
        else//未找到
        {
            printf("未找到需要删除的节点\n");
        }
    }

    return head;
}
```

9、释放链表

```
STU* free_link(STU* head)
{
    //判断链表是否存在
    if(NULL == head)
    {
        printf("link not exist\n");
    }
    else
    {
        STU *pb = head;
        while(pb!=NULL)
        {
            head = pb->next;
            free(pb);
            pb = head;
        }
    }
    return NULL;
}
```

10、链表翻转

```
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 = head->next;
         STU *pn = NULL;
         head->next = NULL;

         while(pb != NULL)
         {
             //纪录pb下一个节点位置
             pn = pb->next;
             //pb连接上一个节点
             pb->next = head;
             head = pb;
             pb = pn;
         }

         printf("链表翻转成功\n");
     }

     return head;
 }
```

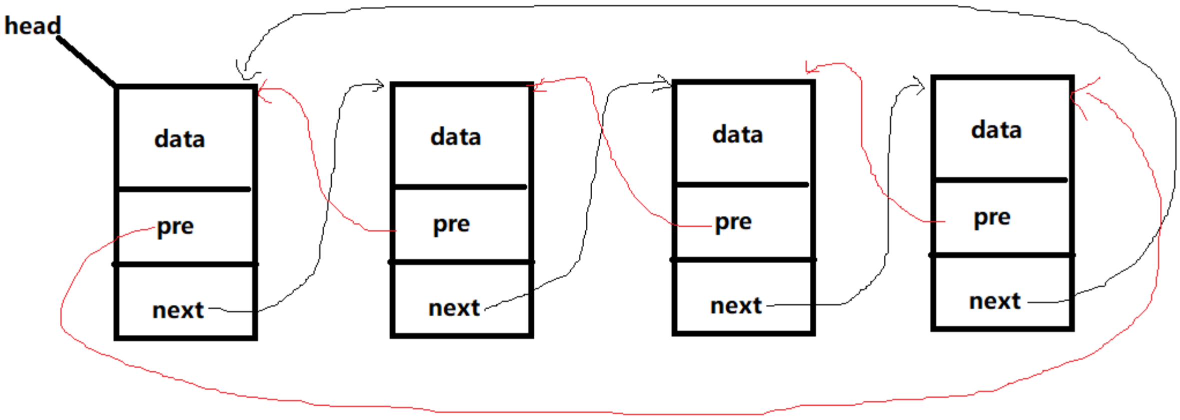
11、链表排序

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

```
   else
    {
        STU *p_i = head, *p_j = head; //int i=0,j=0;
        while (p_i->next != NULL)     //for(i=0;i<n-1;i++)
        {
            STU *p_min = p_i;   //int min = i;
            p_j = p_min->next;  //j=min+1;
            while (p_j != NULL) //for(;j<n;j++)
            {
                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_i != p_min) //if(i != min)
            {
                //交换数据
                STU tmp = *p_i;
                *p_i = *p_min;
                *p_min = tmp;

                tmp.next = p_i->next;
                p_i->next = p_min->next;
                p_min->next = tmp.next;
            }

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

知识点4【双向循环链表】（了解）

1、概述



```
typedef struct stu
{
    //数据域
    int num;
    char name[32];

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

2、完整代码

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

    //指针域
    struct stu *pre;
    struct stu *next;
} STU;
STU *head = NULL;
void insert_link(STU **p_head, STU tmp);
void print_link(STU *head);
STU *search_link(STU *head, int num);
void delete_link(STU **p_head, int num);
```

```c
void free_link(STU **p_head);
int main(int argc, char const *argv[])
{
    int n = 0;
    printf("请输入学生的个数:");
    scanf("%d", &n);

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

        insert_link(&head, tmp);
    }

    //遍历链表
    print_link(head);

    //查询
    printf("请输入你要查询的学号:");
    int num = 0;
    scanf("%d", &num);
    STU *ret = search_link(head, num);
    if (ret != NULL)
    {
        printf("查询的结果:%d %s\n", ret->num, ret->name);
    }

    //删除指定节点
    printf("请输入你要删除的学号:");
    scanf("%d", &num);

    delete_link(&head, num);

    //遍历链表
    print_link(head);

    //释放这个链表
    free_link(&head);

    //遍历链表
    print_link(head);

    return 0;
}

//尾插法
void insert_link(STU **p_head, STU tmp)
{
    STU *head = *p_head;

    //为插入的节点申请空间
    STU *pi = (STU *)calloc(1, sizeof(STU));
    *pi = tmp;
    pi->next = NULL;
    pi->pre = NULL;

    //判断链表是否为空
    if (NULL == head)
    {
        head = pi;
        pi->next = pi;
        pi->pre = pi;
    }
    else
    {
        head->pre->next = pi;
        pi->next = head;
        pi->pre = head->pre;
        head->pre = pi;
    }

    //更新外部的head
    *p_head = head;
}

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

        while (1)
        {
            if (pn == pr) //链表节点为奇数个
            {
                printf("%d %s\n", pn->num, pn->name);
                break;
            }
            else if (pn->next == pr) ////链表节点为偶数个
            {
                printf("%d %s\n", pn->num, pn->name);
                printf("%d %s\n", pr->num, pr->name);
                break;
            }
            else
            {
                printf("%d %s\n", pn->num, pn->name);
                printf("%d %s\n", pr->num, pr->name);
                pn = pn->next;
                pr = pr->pre;
            }
        }
    }
    return;
}

STU *search_link(STU *head, int num)
{
    //判断链表是否存在
    if (NULL == head)
    {
        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("没有找到相关节点\n");
        }
    }

    return NULL;
}

#if 0
void delete_link(STU **p_head, int num)
{
    STU *head = *p_head;

    if (NULL == head)
    {
        printf("link not exist\n");
        return;
    }
    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) //头部、中部节点
        {
            if (pn == head) //删除头节点
            {
                head->next->pre = head->pre;
                head->pre->next = head->next;
                head = head->next;
                //free(pn);
            }
            else //删除中部节点
            {
                pn->pre->next = pn->next;
                pn->next->pre = pn->pre;
                //free(pn);
            }
            printf("成功删除节点:%d %s\n", pn->num, pn->name);
            free(pn);
        }
        else if (pr->num == num) //尾部、中部
        {
            pr->pre->next = pr->next;
            pr->next->pre = pr->pre;
            printf("成功删除节点:%d %s\n", pr->num, pr->name);
            free(pr);
        }
        else
        {
            printf("没有找到相关节点\n");
        }
    }

    *p_head = head;
}
#endif

#if 1
void delete_link(STU **p_head, int num)
{
    STU *head = *p_head;

    if (NULL == head)
    {
        printf("link not exist\n");
        return;
    }
    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) //头部、中部节点
        {
            pn->next->pre = pn->pre;
            pn->pre->next = pn->next;
            if (pn == head) //删除头节点
            {
                head = head->next;
            }

            printf("成功删除节点:%d %s\n", pn->num, pn->name);
            free(pn);
        }
        else if (pr->num == num) //尾部、中部
        {
            pr->pre->next = pr->next;
            pr->next->pre = pr->pre;
            printf("成功删除节点:%d %s\n", pr->num, pr->name);
            free(pr);
        }
        else
        {
            printf("没有找到相关节点\n");
        }
    }

    *p_head = head;
}
#endif

void free_link(STU **p_head)
{
    STU *head = *p_head;

    if (NULL == head)
    {
        printf("link not exist\n");
        return;
    }
    else
    {
        STU *pn = head;
        do
        {
            head = head->next;
            free(pn);
            pn = head;
        } while (pn != (*p_head));
    }

    *p_head = NULL;
    return;
```

```
}
```