数据结构_非常方便的链表

链表是线性表的一种，是一种物理存储结构上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。

和顺序表差不多，用结构体定义一个节点，包括存储的数据和下一个节点的地址。

这里我们写几个接口：尾插，头插，尾删，头删，查找元素，指定位置插入，指定位置删除，销毁。

SList.h

```
#define _CRT_SECURE_NO_WARNINGS 1

#include<stdio.h>
#include<stdlib.h>
#include<assert.h>

typedef int SList_Datatype;

typedef struct SList_Node
{
    SList_Datatype Data;
    struct SList_Node* Next;
}STL_Node;

void SLTPrint(STL_Node* phead);

//尾插
void SLTPushBack(STL_Node** pphead, SList_Datatype x);
//头插
void SLTPushFront(STL_Node** pphead, SList_Datatype x);
//尾删
void SLTPopBack(STL_Node** pphead);
//头删
void SLTPopFront(STL_Node** pphead);

//查找
STL_Node* SLTFind(STL_Node* phead, SList_Datatype x);

//在指定位置之前插入数据
void SLTInsert(STL_Node** pphead, STL_Node* pos, SList_Datatype x);
//在指定位置之后插入数据
void SLTInsertAfter(STL_Node* pos, SList_Datatype x);

//删除pos节点
void SLTErase(STL_Node** pphead, STL_Node* pos);
//删除pos之后的节点
void SLTEraseAfter(STL_Node* pos);
```

```
//销毁链表
void SListDesTroy(STL_Node** pphead);
```

SList.c

打印：

（不改变实参）传一级指针，新建立一个头节点pur，用while循环每次循环打印pur里面的数据，并将pur->next赋给pur。以pur != NULL为循环终止条件。

新建节点：

创建一个newnode指针指向一个新开辟的节点，将数据x放进data，将next置空。

尾插：

（要改变实参）传一个二级指针，用**新建节点函数**新建一个newnode。这里分情况：

（1）如果*pphead为空，即这个链表没有节点就直接将newnode赋给pphead。

（2）这里要先找尾，定义一个ptail指针首先指向*pphead，之后用whlie循环进行找尾。将ptail->next指向newnode。

头插：

头插比较简单（因为不需要找尾），**传一个二级指针**，新建一个节点将newnode的next指向*pphead，将*pphead指向newnode。就完成了头节点的替换。

尾删：

传二级指针，尾删和尾插差不多的，只是在找尾的时候需要两个指针，最后分别指向尾节点和尾节点的前一个节点。这里就不多叙述了。

头删：

传二级指针，新建一个指针指向第二个节点，将第一个节点释放掉后，将*pphead指向先前备份好的第二个节点。

查找元素：

传一级指针，新建一个指针指向phead循环遍历链表里面的数据，得到结果返回相应节点的指针，否则返回空指针。

在指定位置前/后插入数据：

因为前插比较复杂，这里只讲前插。

传二级指针，先新建一个节点将数据放进去，分两种情况

（1）pos为头指针：这里直接调用头插的函数。

（2）pos不为头指针：新建指针prev，这里还是用while循环找pos前一个节点的指针，循环结束条件为prev->next != pos 、之后将prev的next指向新节点，新节点的next指向pos；

删除pos节点/删除pos之后的节点（第二个我觉得没必要）：

这里讲第一个，**传二级指针，**分两种情况

（1）pos == *pphead：直接调用头插。

（2）pos ！ = *pphead：新建指针prev用while循环找尾节点的前一个节点（不多讲）将prev的next指向pos的next，此时pos就不在链表里面了，只需要将它的空间释放掉，将指针置空就ok了。

销毁链表：

传二级指针，用while在循环里面将当前*pphead指向的空间释放，再将*pphead指向，下一个节点，结束条件为*pphead != NULL，循环结束后记得将*pphead置空（**这是个好习惯**）

下面参考代码附上：

```
#define _CRT_SECURE_NO_WARNINGS 1
#include"SList.h"

void SLTPrint(STL_Node* phead)
{
    STL_Node* pur = phead;
    while (pur)
    {
        printf("%d->", pur->Data);
        pur = pur->Next;
    }
    printf("NULL\n");
}

STL_Node* SLTBuyNode(SList_Datatype x)
{
    STL_Node* newnode = (STL_Node*)malloc(sizeof(STL_Node));
    if (newnode == NULL)
    {
        perror("malloc is fail!");
        exit(1);
    }
    newnode->Data = x;
    newnode->Next = NULL;
    return newnode;
}

void SLTPushBack(STL_Node** pphead, SList_Datatype x)
{
```

```
    assert(pphead);
    STL_Node* newnode = SLTBuyNode(x);
    if (*pphead == NULL)
    {
        *pphead = newnode;
    }
    else
    {
        STL_Node* ptail = *pphead;
        while (ptail->Next)
        {
            ptail = ptail->Next;
        }
        ptail->Next = newnode;
    }
}

void SLTPushFront(STL_Node** pphead, SList_Datatype x)
{
    assert(pphead);
    STL_Node* newnode = SLTBuyNode(x);
    newnode->Next = *pphead;
    *pphead = newnode;
}

void SLTPopBack(STL_Node** pphead)
{
    assert(pphead&&*pphead);
    if ((*pphead)->Next == NULL)
    {
        free(pphead);
        pphead = NULL;
    }
    STL_Node* prve = *pphead;
    STL_Node* ptail = *pphead;
    while (ptail->Next)
    {
        prve = ptail;
        ptail = ptail->Next;
    }
    free(ptail);
    ptail = NULL;
    prve->Next = NULL;
}

void SLTPopFront(STL_Node** pphead)
{
    assert(pphead&&*pphead);
    STL_Node* tmp = (*pphead)->Next;
    free(*pphead);
    *pphead = tmp;
}

STL_Node* SLTFind(STL_Node* phead, SList_Datatype x)
```

```
{
    STL_Node* pur = phead;
    while (pur)
    {
        if (pur->Data == x)
        {
            return pur;
        }
        pur = pur->Next;
    }
    return NULL;
}
//在指定位置之前插入数据
void SLTInsert(STL_Node** pphead, STL_Node* pos, SList_Datatype x)
{
    assert(pphead&&*pphead);
    assert(pos);
    STL_Node* newnode = SLTBuyNode(x);
    if (*pphead == pos)
    {
        SLTPushFront(pphead,x);
        free(newnode);
        newnode = NULL;
    }
    else
    {
        STL_Node* prve = *pphead;
        while (prve->Next != pos)
        {
            prve = prve->Next;
        }
        prve->Next = newnode;
        newnode->Next = pos;

    }
}
//在指定位置之后插入数据
void SLTInsertAfter(STL_Node* pos, SList_Datatype x)
{
    assert(pos);
    STL_Node* newnode = SLTBuyNode(x);
    newnode->Next = pos->Next;
    pos->Next = newnode;
}
//删除pos节点
void SLTErase(STL_Node** pphead, STL_Node* pos)
{
    assert(pphead&&*pphead);
    assert(pos);
    if (pos == *pphead)
        SLTPopFront(pphead);
    else
    {
        STL_Node* prev = *pphead;
```

```
        while (prev->Next != pos)
        {
            prev = prev->Next;
        }
        prev->Next = pos->Next;
        free(pos);
        pos = NULL;
    }
}
//删除pos之后的节点
void SLTEraseAfter(STL_Node* pos)
{
    assert(pos&&pos->Next);
    STL_Node* del = pos->Next;
    pos->Next = del->Next;
    free(del);
    del = NULL;
}
//销毁链表
void SListDesTroy(STL_Node** pphead)
{
    assert(pphead && *pphead);
    while (*pphead)
    {
        STL_Node* next = (*pphead)->Next;
        free(*pphead);
        *pphead = next;
    }
    *pphead = NULL;
}
```

本期博客到这里就结束了，如果有什么错误，欢迎指出，如果对你有帮助，请点个赞，谢谢！