第七讲 标准模版库

一、链表

一、 链表

链表是一种物理存储单元上不连续的存储结构，数据元素之间是通过链表中的指针进行链接。链表是由一系列的节点（链表中每一个元素称为节点）组成，节点可以在运行时动态生成。

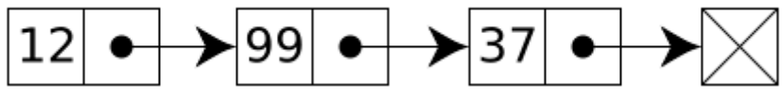
每一个节点都包含两个部分：一个是存储数据的数据域，另一个是存储下一个节点地址的指针域。

一般链表在一些需要快速插入/删除，而不太关心或者不需要随机访问的情况下使用。

链表相对于数组：链表允许在任意位置插入或删除节点，但是链表不支持随机访问节点，只能从头节点逐个访问（遍历）每个节点。

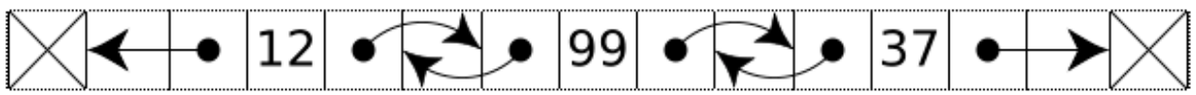
链表分三种：单向链表、双向链表以及循环链表，分别如下图所示：

1、单向链表



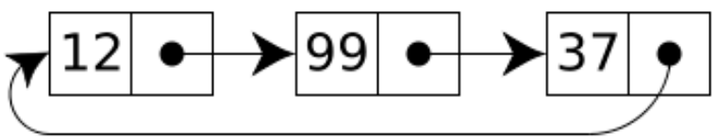
单向链表是链表中最简单的链表，它包含两个域：一个数据域和一个指针域，它只可以向一个方向遍历。

2、双向链表



双向链表中每个节点中有数据域和两个指针域，前面的指针指向前一个节点，后一个指针指向下一个节点。

3、循环链表



循环链表中，首节点和末节点被连接在一起，它也可以被视为“无头无尾”。这种链表比较利于数据存储缓存。

【示例1-1】创建链表

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define  N 10

//构建一个结构体(节点)
typedef struct Node {
    char name[20];
    int age;
    struct Node* link;
}student;
```

```
//创建链表(有n个节点, 函数返回链表头节点)
student* createList(int n) {
    student* head = NULL;//头节点
    student* pNode = NULL;//动态节点
    student* sNode = NULL;//每次创建新的节点指针
    if((head=(student*)malloc(sizeof(student)))==NULL) {
        printf("Memory allocation fails...");
        return NULL;
    }
```

```
    //对头节点初始化
    strcpy(head->name, "headNode");
    head->age = 25;
    head->link = NULL;

    //动态节点指向头节点
    pNode = head;
    for(int i=0;i<n-1;++i) {
        if((sNode=(student*)malloc(sizeof(student)))==NULL) {
            printf("Memory allocation fails...");
            return NULL;
        }
        //节点指针指向下一个节点
        pNode->link = sNode;
        printf("please input %d person name:\n",i+1);
        scanf("%s",sNode->name);
        printf("please input %d person age:\n",i+1);
        scanf("%d",&(sNode->age));
        sNode->link = NULL;

        //动态节点指向刚创建的新节点
        pNode = sNode;
    }
    return head;
}
```

【示例1-2】：打印链表

```
//输出链表所有节点(head:链表头节点)
void showNode(student* head) {
    student* pNode;//动态节点
    pNode = head;
    while (pNode!=NULL) {//判断节点是否为空
        printf("**************************\n");
        printf("name = %s\n",pNode->name);
        printf("age = %d\n",pNode->age);

        pNode = pNode->link;
    }
}
```

```
int main(int argc, const char * argv[]) {
    //调用函数，创建链表共4个节点
    student* head = createList(4);

    //调用函数，输出节点
    showNode(head);
    return 0;
}
```

程序运行结果如下：

```
**********
name = headNode
age = 25
**********
name = aa
age = 11
**********
name = bb
age = 22
**********
name = cc
age = 33
```

插入节点就是让一个特定的节点的指针链接我们新创建的节点，并且让新创建的节点的指针链接原特定节点的下一个节点。

【示例1-3】插入节点

```
//插入节点(pNode:在pNode节点后添加节点)
void insertNode(student* pNode) {
    char name[20];
    int age;
    student* sNode = NULL;//新节点
    if((sNode = (student*)malloc(sizeof(student)))==NULL) {
        printf("Memory allocation fails!\n");
        return;
    }
    printf("please input name\n");
    scanf("%s",name);
    strcpy(sNode->name, name);

    printf("please input age\n");
    scanf("%d",&age);
    sNode->age = age;
```

```
    //新节点链接特定节点下一个节点
    sNode->link = pNode->link;
    //特定节点链接新节点
    pNode->link = sNode;
}

int main(int argc, const char * argv[]) {
    student* head = NULL;
    head = (student*)malloc(sizeof(student));
    strcpy(head->name, "aaa");
    head->age = 20;
    head->link = NULL;

    insertNode(head);

    showNode(head);

    return 0;
}
```

程序运行结果如下：

```
**********
name = aaa
age = 20
**********
name = bbb
age = 30
```

删除节点就是让删除节点的上一个节点指向删除节点的下一个节点，再把当前节点空间释放掉。

【示例1-4】删除节点

```
//删除节点(head:链表头节点,name:删除节点的名字)
student* deleteNode(student* head,char* name) {
    if(head==NULL){
        return NULL;
    }
    student* p1Node = NULL;//动态节点
    student* p2Node = NULL;//动态节点

    p1Node = head;//指向头节点
    p2Node = head->link;//指向头节点下一个节点

    //判断是否删除头节点
    if(strcmp(p1Node->name, name)==0) {
        head = head->link;
        free(p1Node);
        return head;
    }
```

```
    //判断删除后面节点
    while(p2Node!=NULL) {
        if(strcmp(p2Node->name, name)==0) {
            p1Node->link = p2Node->link;
            free(p2Node);
            return head;
        }
        //p1Node p2Node分别向后移动
        p1Node = p2Node;
        p2Node = p2Node->link;
    }
    return head;
}
```

```c
int main(int argc, const char * argv[]) {

    //创建一个链表
    student* head = createList(4);

    //删除的节点名字
    char name[20] = "bbb";
    head = deleteNode(head, name);

    //删除后链表
    showNode(head);

    return 0;
}
```

程序运行结果如下:

```
**********
name = headNode
age = 25
**********
name = aaa
age = 11
**********
name = ccc
age = 33
```

查找节点就是遍历链表找到对应的几点，不需要改变任何节点和节点指针。

【示例1-5】查找节点

```
//查找节点(head:链表头节点,name:查找节点名字)
student* searchNode(student* head,char* name) {

    student* pNode = head;//创建动态节点
    while (pNode != NULL) {
        if(strcmp(pNode->name, name)==0){
            return pNode;
        }
        pNode = pNode->link;
    }

    return pNode;
}
```

```c
int main(int argc, const char * argv[]) {

    //创建链表
    student* head = createList(4);

    char name[20] = "c";//要查找的几点名字

    //调用查找函数
    student* sNode = searchNode(head,name);
    if(sNode==NULL){
        printf("no have %s\n",name);
    }else{
        printf("yes\n");
        printf("name = %s\n",sNode->name);
        printf("age = %d\n",sNode->age);
    }

    return 0;
}
```

程序运行结果如下：

yes

name = c

age = 3

课后作业：

(1) 编写一个函数，用来删除整个链表。

(2) 编写一个函数，用来逆序输出链表。