C封装双向链表对象

Aug 28, 2018 | 学习笔记——C数据结构 | 2 阅读 | 1.4k 字 | 8 分钟

文章目录

1. DoubleLinkedList(双向链表)
2. DoubleLinkedList.c文件
3. DoubleLinkedList.h文件
4. testDoubleLinkedList.c文件
5. 编译：

DoubleLinkedList(双向链表)

github源码

特点：

1.在单链表中，nextElem的执行时间为O(1),而priorElem的执行时间为O(n)。这是因为单链表只有一个指示直接后继的指针域。为克服单链表这种单向性的缺点，可以使用双向链表。

2.在双向链表的结点中有两个指针域，其中一指向直接后继，另一指向直接前驱。

3.在双向链表中，有些操作，如length、getElem、indexElem等仅需涉及一个方向的指针，则他们的算法和单链表的相同，在插入、删除时则有不同之处，需要同时修改两个方向上的指针。

4.在双向链表中插入节点temp：

```
temp->prior = p;
temp->next = p->next;
p->next = temp;
p->next->prior = temp;
```

6.在双向链表中删除节点temp：

```
temp = p->next;
p->next = temp->next;
temp->next->prior = p;
free(temp);
```

DoubleLinkedList.c文件

```
#include <stdio.h>
#include <malloc.h>
#include "DoubleLinkedList.h"

static void clear(DoubleLinkedList *This);
static int isEmpty(DoubleLinkedList *This);
static int length(DoubleLinkedList *This);
static void print(DoubleLinkedList *This);
static int indexElem(DoubleLinkedList *This, ElemType* x);
static int getElem(DoubleLinkedList *This, int index, ElemType *e);
static int modifyElem(DoubleLinkedList *This, int index, ElemType* e);
static int deleteElem(DoubleLinkedList *This, int index, ElemType* e);
static int appendElem(DoubleLinkedList *This, ElemType *e);
static int insertElem(DoubleLinkedList *This, int index, ElemType *e);
static int popElem(DoubleLinkedList *This, ElemType* e);

DoubleLinkedList *InitDoubleLinkedList(){
    DoubleLinkedList *L = (DoubleLinkedList *)malloc(sizeof(DoubleLinkedList));
    Node *p = (Node *)malloc(sizeof(Node));
    L->This = p;
    p->prior = NULL;
    p->next = NULL;
    L->clear = clear;
    L->isEmpty = isEmpty;
    L->length = length;
    L->print = print;
    L->indexElem = indexElem;
    L->getElem = getElem;
    L->modifyElem = modifyElem;
    L->deleteElem = deleteElem;
    L->appendElem = appendElem;
    L->insertElem = insertElem;
```

```
    L->popElem = popElem;
    return L;
}

void DestroyDoubleLinkedList(DoubleLinkedList *L){
    L->clear(L);
    free(L->This);
    free(L);
    L = NULL;
}

static void clear(DoubleLinkedList *This){
    Node *p = This->This->next;
    Node *temp = NULL;
    while(p){
        temp = p;
        p = p->next;
        free(temp);
    }
    p = This->This;
    p->next = NULL;
}

static int isEmpty(DoubleLinkedList *This){
    Node *p = This->This;
    if(p->next){
        return 0;
    }else{
        return 1;
    }
}

static int length(DoubleLinkedList *This){
    int j = 0;
    Node *p = This->This->next;
    while(p){
        j++;
        p = p->next;
    }
    return j;
}

static void print(DoubleLinkedList *This){
```

```
    Node *p = This->This->next;
    while(p){
        printf("%d ", p->elem);
        p = p->next;
    }
    printf("\n");
}

static int indexElem(DoubleLinkedList *This, ElemType* e){
    Node *p = This->This->next;
    int pos = -1;
    int j = 0;
    while(p){
        if(*e == p->elem){
            pos = j;
        }
        p = p->next;
        j++;
    }
    return pos;
}

static int getElem(DoubleLinkedList *This, int index, ElemType *e){
    Node *p = This->This->next;
    int j = 0;
    while(p && j < index){
        p = p->next;
        j++;
    }
    if(!p || j > index) return -1;
    *e = p->elem;
    return 0;
}

static int modifyElem(DoubleLinkedList *This, int index, ElemType* e){
    Node *p = This->This->next;
    int j = 0;
    while(p && j < index){
        p = p->next;
        j++;
    }
    if(!p || j > index) return -1;
    p->elem = *e;
```

```
    return 0;
}

static int insertElem(DoubleLinkedList *This, int index, ElemType *e){
    Node *p = This->This;
    int j = 0;
    Node *temp = (Node *)malloc(sizeof(Node));
    if(!temp) return -1;
    while(p && j < index){
        p = p->next;
        j++;
    }
    if(!p || j > index) return -1;
    temp->elem = *e;
    p->next->prior = temp;
    temp->prior = p;
    temp->next = p->next;
    p->next = temp;
    return 0;
}

static int deleteElem(DoubleLinkedList *This, int index, ElemType* e){
    Node *p = This->This;
    Node *temp = NULL;
    int j = 0;
    while(p->next && j < index){
        p = p->next;
        j++;
    }
    if(!p->next || j > index) return -1;
    temp = p->next;
    p->next = temp->next;
    temp->next->prior = p;
    *e = temp->elem;
    free(temp);
    return 0;
}

static int appendElem(DoubleLinkedList *This, ElemType *e){
    Node *p = This->This;
    Node *temp = (Node *)malloc(sizeof(Node));
    if(!temp) return -1;
    while(p){
```

```
        if(NULL == p->next){
            temp->elem = *e;
            p->next = temp;
            temp->prior = p;
            temp->next =  NULL;
        }
        p = p->next;
    }
    return 0;
}

static int popElem(DoubleLinkedList *This, ElemType* e){
    Node *p = This->This;
    Node *temp = NULL;
    while(p->next->next){
        p = p->next;
    }
    temp = p->next;
    if(!temp) return -1;
    *e = temp->elem;
    free(temp);
    p->next = NULL;
    return 0;
}
```

DoubleLinkedList.h文件

```
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __DOUBLELINKEDLIST_H
#define __DOUBLELINKEDLIST_H
/* Includes ------------------------------------------------------------------*/
/* Exported types ------------------------------------------------------------*/
typedef int ElemType;      //数据元素的类型，假设是int型的

typedef struct Node{
    ElemType elem;  //存储空间
    struct Node *prior;
    struct Node *next;
}Node;
```

```
typedef struct DoubleLinkedList{
    Node *This;
    void (*clear)(struct DoubleLinkedList *This);
    int (*isEmpty)(struct DoubleLinkedList *This);
    int (*length)(struct DoubleLinkedList *This);
    void (*print)(struct DoubleLinkedList *This);
    int (*indexElem)(struct DoubleLinkedList *This, ElemType* x);
    int (*getElem)(struct DoubleLinkedList *This, int index, ElemType *e);
    int (*modifyElem)(struct DoubleLinkedList *This, int index, ElemType* e);
    int (*deleteElem)(struct DoubleLinkedList *This, int index, ElemType* e);
    int (*appendElem)(struct DoubleLinkedList *This, ElemType *e);
    int (*insertElem)(struct DoubleLinkedList *This, int index, ElemType *e);
    int (*popElem)(struct DoubleLinkedList *This, ElemType* e);
}DoubleLinkedList;

/* Exported macro ------------------------------------------------------------*/
DoubleLinkedList *InitDoubleLinkedList();
void DestroyDoubleLinkedList(DoubleLinkedList *L);

#endif
```

testDoubleLinkedList.c文件

```
#include <stdio.h>
#include <malloc.h>
#include "DoubleLinkedList.h"

int main(void){
    int i;
    ElemType elem,elem1;
    DoubleLinkedList *list = InitDoubleLinkedList();
    printf("list is empty:%d\n",list->isEmpty(list));
    for(i=0;i<10;i++){
        list->appendElem(list,&i);
    }
    list->print(list);
    printf("list is empty:%d\n",list->isEmpty(list));
    printf("list length:%d\n",list->length(list));
    list->clear(list);
    for (i = 10; i < 20; i++){
```

```
        list->appendElem(list,&i);
    }
    list->print(list);
    list->getElem(list,3,&elem1);
    printf("the elem of index 3 is %d\n",elem1);
    elem = 31;
    list->modifyElem(list,3,&elem);
    list->getElem(list,3,&elem1);
    printf("modify the elem of index 3 to %d\n",elem1);
    list->print(list);
    elem = 25;
    list->insertElem(list,5,&elem);
    printf("insert elem %d to index 5\n",elem);
    list->print(list);
    list->deleteElem(list,7,&elem);
    printf("delete elem %d of index 7\n",elem);
    list->print(list);
    elem = 14;
    printf("the index of 14 is %d\n",list->indexElem(list,&elem));
    list->popElem(list,&elem);
    printf("pop elem %d\n",elem);
    list->print(list);
    DestroyDoubleLinkedList(list);
    return 0;
}
```

编译：

```
gcc DoubleLinkedList.c DoubleLinkedList.h testDoubleLinkedList.c -o testDoubleLinkedList
```

运行testDoubleLinkedList

输出：

```
list is empty:1
0 1 2 3 4 5 6 7 8 9
list is empty:0
list length:10
```

```
10 11 12 13 14 15 16 17 18 19
the elem of index 3 is 13
modify the elem of index 3 to 31
10 11 12 31 14 15 16 17 18 19
insert elem 25 to index 5
10 11 12 31 14 25 15 16 17 18 19
delete elem 16 of index 7
10 11 12 31 14 25 15 17 18 19
the index of 14 is 4
pop elem 19
10 11 12 31 14 25 15 17 18
```


C

用单链表表示一元多项式及其简单计算

C封装单向循环链表对象

Search

分类

学习笔记——C 算法
学习笔记——C数据结构
学习笔记——Python
学习笔记——android
学习笔记——expert c programming
学习笔记——linux
学习笔记——opencv
学习笔记——嵌入式开发
学习笔记——机器学习
学习笔记——网络协议

标签

android C 网络协议 linux 嵌入式开发 Python opencv 机器学习

最近文章

linux解压缩命令
linux查找命令
Little Kernel 04
Little Kernel 03
Little Kernel 02
Little Kernel 01

消息摘要算法

C按位操作实现CRC计算算法

CRC循环冗余校验算法

链表的反转

友情链接

人生的小站