C封装双向循环链表对象

Aug 29, 2018 | 学习笔记——C数据结构 | 3 阅读 | 1.9k 字 | 11 分钟

文章目录

1. DoubleCircularLinkedList(双向循环链表)
2. DoubleCircularLinkedList.c文件
3. DoubleCircularLinkedList.h文件
4. testDoubleCircularLinkedList.c文件
5. 编译：

DoubleCircularLinkedList(双向循环链表)

github源码
特点：

1.插入一个结点temp

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

2.删除一个结点n，时间复杂度O(1)

```
n->prior->next = n->next;
n->next->prior = n->prior;
free(n);
```

3.在末尾增加一个结点temp

```
temp->elem = *e;
temp->prior = p;
temp->next =  head;
p->next = temp;
head->prior = temp;
```

4.getPriorNode、getNextNode的时间复杂度均为O(1)

```
static Node *getPriorNode(Node *n){
    return n->prior;
}

static Node *getNextNode(Node *n){
    return n->next;
}
```

DoubleCircularLinkedList.c文件

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

static void clear(DoubleCircularLinkedList *This);
static int isEmpty(DoubleCircularLinkedList *This);
static int length(DoubleCircularLinkedList *This);
static void print(DoubleCircularLinkedList *This);
static void circlePrint(DoubleCircularLinkedList *This,int times);
static int indexElem(DoubleCircularLinkedList *This, ElemType* x);
static int indexNode(DoubleCircularLinkedList *This, Node* n);
static int getElem(DoubleCircularLinkedList *This, int index, ElemType *e);
static Node *getNode(DoubleCircularLinkedList *This, int index);
static Node *getPriorNode(Node *n);
static Node *getNextNode(Node *n);
static int modifyElem(DoubleCircularLinkedList *This, int index, ElemType* e);
static int deleteElem(DoubleCircularLinkedList *This, int index, ElemType* e);
static int deleteNode(DoubleCircularLinkedList *This, Node* n);
static int appendElem(DoubleCircularLinkedList *This, ElemType *e);
static int insertElem(DoubleCircularLinkedList *This, int index, ElemType *e);
static int popElem(DoubleCircularLinkedList *This, ElemType* e);

DoubleCircularLinkedList *InitDoubleCircularLinkedList(){
    DoubleCircularLinkedList *L = (DoubleCircularLinkedList *)malloc(sizeof(DoubleCircularLi
    Node *p = (Node *)malloc(sizeof(Node));
    L->This = p;
    p->prior = p;
    p->next = p;
    L->clear = clear;
    L->isEmpty = isEmpty;
    L->length = length;
    L->print = print;
    L->circlePrint = circlePrint;
    L->indexElem = indexElem;
    L->indexNode = indexNode;
    L->getElem = getElem;
    L->getNode = getNode;
    L->getPriorNode = getPriorNode;
    L->getNextNode = getNextNode;
    L->modifyElem = modifyElem;
    L->deleteElem = deleteElem;
    L->deleteNode = deleteNode;
    L->appendElem = appendElem;
    L->insertElem = insertElem;
    L->popElem = popElem;
    return L;
}

void DestroyDoubleCircularLinkedList(DoubleCircularLinkedList *L){
    L->clear(L);
    free(L->This);
    free(L);
    L = NULL;
}

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

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

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

static void print(DoubleCircularLinkedList *This){
    Node *head = This->This;
    Node *p = This->This->next;
    while(p != head){
        printf("%d ", p->elem);
        p = p->next;
    }
    printf("\n");
}

static void circlePrint(DoubleCircularLinkedList *This,int times){
    Node *head = This->This;
    int i = 0;
    Node *p = This->This->next;
    for(i = 0;i<times;){
        if(p == head){
            i++;
        }else{
            printf("%d ", p->elem);
        }
        p = p->next;
    }
    printf("\n");
}

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

static int indexNode(DoubleCircularLinkedList *This, Node* n){
    Node *head = This->This;
    Node *p = This->This->next;
    int pos = -1;
    int j = 0;
    while(p != head){
        if(n == p){
            pos = j;
        }
        p = p->next;
        j++;
    }
    return pos;
}
```

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

static Node *getNode(DoubleCircularLinkedList *This, int index){
    Node *head = This->This;
    Node *p = This->This->next;
    int j = 0;
    while(p != head && j < index){
        p = p->next;
        j++;
    }
    if(p == head || j > index) return NULL;
    return p;
}

static Node *getPriorNode(Node *n){
    return n->prior;
}

static Node *getNextNode(Node *n){
    return n->next;
}

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

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

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

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

DoubleCircularLinkedList.h文件

```c
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __DOUBLECIRCULARLINKEDLIST_H
#define __DOUBLECIRCULARLINKEDLIST_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 DoubleCircularLinkedList{
    Node *This;
    void (*clear)(struct DoubleCircularLinkedList *This);
    int (*isEmpty)(struct DoubleCircularLinkedList *This);
    int (*length)(struct DoubleCircularLinkedList *This);
    void (*print)(struct DoubleCircularLinkedList *This);
    void (*circlePrint)(struct DoubleCircularLinkedList *This,int times);
    int (*indexElem)(struct DoubleCircularLinkedList *This, ElemType* x);
    int (*indexNode)(struct DoubleCircularLinkedList *This, Node* n);
    int (*getElem)(struct DoubleCircularLinkedList *This, int index, ElemType *e);
    Node *(*getNode)(struct DoubleCircularLinkedList *This, int index);
    Node *(*getPriorNode)(Node *n);
    Node *(*getNextNode)(Node *n);
    int (*modifyElem)(struct DoubleCircularLinkedList *This, int index, ElemType* e);
    int (*deleteElem)(struct DoubleCircularLinkedList *This, int index, ElemType* e);
    int (*deleteNode)(struct DoubleCircularLinkedList *This, Node* n);
    int (*appendElem)(struct DoubleCircularLinkedList *This, ElemType *e);
    int (*insertElem)(struct DoubleCircularLinkedList *This, int index, ElemType *e);
    int (*popElem)(struct DoubleCircularLinkedList *This, ElemType* e);
}DoubleCircularLinkedList;

/* Exported macro ------------------------------------------------------------*/
DoubleCircularLinkedList *InitDoubleCircularLinkedList();
void DestroyDoubleCircularLinkedList(DoubleCircularLinkedList *L);

#endif
```

testDoubleCircularLinkedList.c文件

```c
#include <stdio.h>
#include <malloc.h>
#include "DoubleCircularLinkedList.h"

int main(void){
    int i;
    ElemType elem,elem1;
    Node *tempn;
    Node *tempm;
    DoubleCircularLinkedList *list = InitDoubleCircularLinkedList();
    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);
    printf("circle print 3 times:\n");
    list->circlePrint(list,3);
    tempn = list->getNode(list,5);
    printf("get node of index 5: node elem = %d\n",tempn->elem);
    printf("the index of node: %d\n",list->indexNode(list, tempn));
    tempm = list->getPriorNode(tempn);
    printf("get Prior node of index 5: Prior node elem = %d\n",tempm->elem);
    tempm = list->getNextNode(tempn);
    printf("get Next node of index 5: Next node elem = %d\n",tempm->elem);
    list->deleteNode(list,tempn);
    printf("delete node of index 5\n");
    list->print(list);
    DestroyDoubleCircularLinkedList(list);
    return 0;
}
```

编译：

```
gcc DoubleCircularLinkedList.c DoubleCircularLinkedList.h testDoubleCircularLinkedList.c -o te
```

运行testDoubleCircularLinkedList

输出：

```
list is empty:0
0 1 2 3 4 5 6 7 8 9
list is empty:1
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
circle print 3 times:
10 11 12 31 14 25 15 17 18 10 11 12 31 14 25 15 17 18 10 11 12 31 14 25 15 17 18
get node of index 5: node elem = 25
the index of node: 5
get Prior node of index 5: Prior node elem = 14
get Next node of index 5: Next node elem = 15
delete node of index 5
10 11 12 31 14 15 17 18
```

Donate

C

用线性表表示的顺序栈

用单链表表示一元多项式及其简单计算

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