C封装单向循环链表对象

Aug 28, 2018 | 学习笔记——C数据结构 | 3 阅读 | 1.5k 字 | 8 分钟

文章目录

SingleCircularLinkedList(单向循环链表)

github源码

特点：

1.表中最后一个结点的指针指向头结点，整个链表形成一个环。由表中任一结点出发均可找到表中其他结点。

2.循环链表的操作和线性链表基本一致，差别仅在于算法中的循环条件不是p或p->next是否为空，而是他们是否等于头指针。

SingleCircularLinkedList.c文件

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

static void clear(SingleCircularLinkedList *This);
static int isEmpty(SingleCircularLinkedList *This);
static int length(SingleCircularLinkedList *This);
static void print(SingleCircularLinkedList *This);
static void circlePrint(SingleCircularLinkedList *This,int times);
static int indexElem(SingleCircularLinkedList *This, ElemType* x);
static int getElem(SingleCircularLinkedList *This, int index, ElemType *e);
```

```
static int modifyElem(SingleCircularLinkedList *This, int index, ElemType* e);
static int deleteElem(SingleCircularLinkedList *This, int index, ElemType* e);
static int appendElem(SingleCircularLinkedList *This, ElemType *e);
static int insertElem(SingleCircularLinkedList *This, int index, ElemType *e);
static int popElem(SingleCircularLinkedList *This, ElemType* e);

SingleCircularLinkedList *InitSingleCircularLinkedList(){
    SingleCircularLinkedList *L = (SingleCircularLinkedList *)malloc(sizeof(SingleCircularLi
    Node *p = (Node *)malloc(sizeof(Node));
    L->This = p;
    p->next = p;
    L->clear = clear;
    L->isEmpty = isEmpty;
    L->length = length;
    L->print = print;
    L->circlePrint = circlePrint;
    L->indexElem = indexElem;
    L->getElem = getElem;
    L->modifyElem = modifyElem;
    L->deleteElem = deleteElem;
    L->appendElem = appendElem;
    L->insertElem = insertElem;
    L->popElem = popElem;
    return L;
}

void DestroySingleCircularLinkedList(SingleCircularLinkedList *L){
    L->clear(L);
    free(L->This);
    free(L);
    L = NULL;
}

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

```
}

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

static int length(SingleCircularLinkedList *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(SingleCircularLinkedList *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(SingleCircularLinkedList *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(SingleCircularLinkedList *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 getElem(SingleCircularLinkedList *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 int modifyElem(SingleCircularLinkedList *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(SingleCircularLinkedList *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;
    temp->next = p->next;
    p->next = temp;
    return 0;
}

static int deleteElem(SingleCircularLinkedList *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;
    *e = temp->elem;
    free(temp);
    return 0;
}

static int appendElem(SingleCircularLinkedList *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;
    p->next = temp;
    temp->next = head;
    return 0;
}

static int popElem(SingleCircularLinkedList *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;
    return 0;
}
```

SingleCircularLinkedList.h文件

```
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __SINGLECIRCULARLINKEDLIST_H
#define __SINGLECIRCULARLINKEDLIST_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 *next;
}Node;

typedef struct SingleCircularLinkedList{
    Node *This;
    void (*clear)(struct SingleCircularLinkedList *This);
```

```
    int (*isEmpty)(struct SingleCircularLinkedList *This);
    int (*length)(struct SingleCircularLinkedList *This);
    void (*print)(struct SingleCircularLinkedList *This);
    void (*circlePrint)(struct SingleCircularLinkedList *This,int times);
    int (*indexElem)(struct SingleCircularLinkedList *This, ElemType* x);
    int (*getElem)(struct SingleCircularLinkedList *This, int index, ElemType *e);
    int (*modifyElem)(struct SingleCircularLinkedList *This, int index, ElemType* e);
    int (*deleteElem)(struct SingleCircularLinkedList *This, int index, ElemType* e);
    int (*appendElem)(struct SingleCircularLinkedList *This, ElemType *e);
    int (*insertElem)(struct SingleCircularLinkedList *This, int index, ElemType *e);
    int (*popElem)(struct SingleCircularLinkedList *This, ElemType* e);
}SingleCircularLinkedList;

/* Exported macro ------------------------------------------------------------*/
SingleCircularLinkedList *InitSingleCircularLinkedList();
void DestroySingleCircularLinkedList(SingleCircularLinkedList *L);

#endif
```

testSingleCircularLinkedList.c文件

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

int main(void){
    int i;
    ElemType elem,elem1;
    SingleCircularLinkedList *list = InitSingleCircularLinkedList();
    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);
    DestroySingleCircularLinkedList(list);
    return 0;
}
```

编译：

```
gcc SingleCircularLinkedList.c SingleCircularLinkedList.h testSingleCircularLinkedList.c -o te
```

运行testSingleCircularLinkedList

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
```


C

C封装双向链表对象 C封装单链表对象

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