# 程序设计语言2022-作业1

## ADT源代码

/\* list.h -- header file for a simple list type \*/

#ifndef LIST\_H\_

#define LIST\_H\_

#include <stdbool.h>

typedef struct node

{

void\* item;

struct node\* next;

} Node;

typedef struct list {

Node\* head;

size\_t size;

}List;

void InitializeList(List\* plist, size\_t size);

bool ListIsEmpty(const List\* plist);

bool ListIsFull(const List\* plist);

unsigned int ListItemCount(const List\* plist);

bool AddItem(void\* item, List\* plist);

void Traverse(const List\* plist, void (\*pfun)(void\* item));

void EmptyTheList(List\* plist);

#endif

/\* list.c -- functions supporting list operations \*/

#include <stdio.h>

#include <stdlib.h>

#include <memory.h>

#include "mylist.h"

/\* local function prototype \*/

static void CopyToNode(void\* item, Node\* pnode, size\_t size);

/\* interface functions \*/

/\* set the list to empty \*/

void InitializeList(List\* plist, size\_t element\_size)

{

plist->head = NULL;

plist->size = element\_size;

}

/\* returns true if list is empty \*/

bool ListIsEmpty(const List\* plist)

{

if (plist->head == NULL)

return true;

else

return false;

}

/\* returns true if list is full \*/

bool ListIsFull(const List\* plist)

{

Node\* pt;

bool full;

void\* item;

pt = (Node\*)malloc(sizeof(Node));

item = malloc(sizeof(plist->size));

if (pt == NULL || item == NULL)

full = true;

else

full = false;

free(pt);

free(item);

return full;

}

/\* returns number of nodes \*/

unsigned int ListItemCount(const List\* plist)

{

unsigned int count = 0;

Node\* pnode = plist->head; /\* set to start of list \*/

while (pnode != NULL)

{

++count;

pnode = pnode->next; /\* set to next node \*/

}

return count;

}

/\* creates node to hold item and adds it to the end of \*/

/\* the list pointed to by plist (slow implementation) \*/

bool AddItem(void\* item, List\* plist)

{

if (ListIsFull(plist))

return false;

Node\* pnew;

Node\* scan = plist->head;

pnew = (Node\*)malloc(sizeof(Node));

if (pnew == NULL)

return false; /\* quit function on failure \*/

CopyToNode(item, pnew, plist->size);

pnew->next = NULL;

if (scan == NULL) /\* empty list, so place \*/

plist->head = pnew; /\* pnew at head of list \*/

else

{

while (scan->next != NULL)

scan = scan->next; /\* find end of list \*/

scan->next = pnew; /\* add pnew to end \*/

}

return true;

}

/\* visit each node and execute function pointed to by pfun \*/

void Traverse(const List\* plist, void (\*pfun)(void\* item))

{

Node\* pnode = plist->head; /\* set to start of list \*/

while (pnode != NULL)

{

(\*pfun)(pnode->item); /\* apply function to item \*/

pnode = pnode->next; /\* advance to next item \*/

}

}

/\* free memory allocated by malloc() \*/

/\* set list pointer to NULL \*/

void EmptyTheList(List\* plist)

{

Node\* psave;

while (plist->head != NULL)

{

psave = (plist->head)->next; /\* save address of next node \*/

free(plist->head); /\* free current node \*/

plist->head = psave; /\* advance to next node \*/

}

}

/\* local function definition \*/

/\* copies an item into a node \*/

static void CopyToNode(void\* item, Node\* pnode, size\_t size)

{

pnode->item = malloc(size);

memcpy(pnode->item, item, size);

}

## 简要功能性描述

List是基于C语言给出的一个线性单链表的实现。List存储的数据占据相同大小的内存空间。List能够支持存储C的所有基本类型数据以及用户自定义的数据结构。List支持用户进行：初始化、向链表尾部插入、判断链表空/满、获取链表长度、顺序遍历框架、清空链表这些操作。在执行前面所述的操作时，List能够自动进行内存的申请和归还。

## 静态结构定义部分

typedef struct node

{

void\* item;

struct node\* next;

} Node;

typedef struct list {

Node\* head;

size\_t size;

}List;

## 动态行为定义部分

static void CopyToNode(void\* item, Node\* pnode, size\_t size);

void InitializeList(List\* plist, size\_t size);

bool ListIsEmpty(const List\* plist);

bool ListIsFull(const List\* plist);

unsigned int ListItemCount(const List\* plist);

bool AddItem(void\* item, List\* plist);

void Traverse(const List\* plist, void (\*pfun)(void\* item));

void EmptyTheList(List\* plist);

## 外部接口

1. 初始化链表

void InitializeList(List\* plist, size\_t size);

1. 判断链表是否为空

bool ListIsEmpty(const List\* plist);

1. 判断链表是否已满

bool ListIsFull(const List\* plist);

1. 获取链表长度

unsigned int ListItemCount(const List\* plist);

1. 向表尾添加元素

bool AddItem(void\* item, List\* plist);

1. 遍历链表框架

void Traverse(const List\* plist, void (\*pfun)(void\* item));

1. 清空链表

void EmptyTheList(List\* plist);

## 如何基于该ADT进行开发

1. 创建并初始化一个存储数据类型为Type的List:

List list;

InitializeList(&list, sizeof(Type));

Type 必须是C基本数据类型或已经给出定义的数据结构.

1. 向完成初始化后的、存储类型数据类型为Type的(List)list添加Type类型的数据t：

AddItem(&t, &list);

1. 判断(List)list 空

ListIsEmpty(&list);

1. 判断(List)list 满

ListIsFull(&list);

1. 遍历List(list), 对每个元素执行操作f

void f(void\* item) {

//f的操作

...

}

Traverse(&list, f);

1. 清空(List)list

EmptyTheList(&list);

## 七、示例代码

#include<stdio.h>

#include<stdlib.h>

#include"mylist.h"

void printNode(void\* item) {

printf("%d\n", \*(int\*)item);

}

int main() {

/\* 创建一个顺序存放数字1-10的链表并打印 \*/

List list;

InitializeList(&list, sizeof(int)); //将list初始化为存放int类型数据的list

printf("%d,%d\n", ListIsEmpty(&list), ListIsFull(&list));

for (int i = 0; i < 10; i++) //将数字1-10顺序加入list

AddItem(&i, &list);

Traverse(&list, printNode); //遍历list打印每个结点的值

printf("%d,%d\n", ListIsEmpty(&list), ListIsFull(&list));

EmptyTheList(&list); //清空list

printf("%d,%d\n", ListIsEmpty(&list), ListIsFull(&list));

return 0;

}