Project: LightBulbs

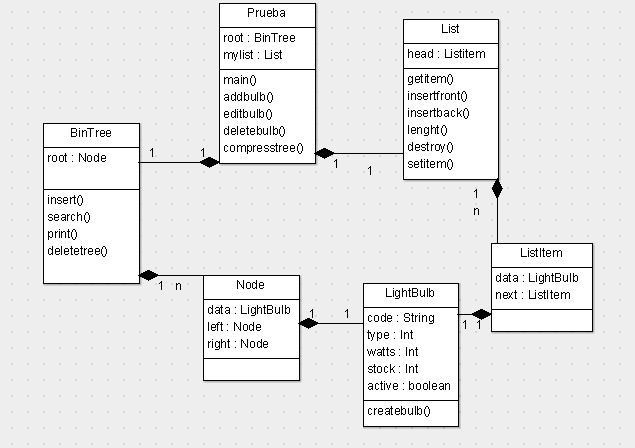
Author: Tomás Lucas Vilaboa

Hours: 9

Code Lines: 285

Files: Main.c, tree.h,list.h,lightbulb.h

***UML***



***Main.c***

#include<stdlib.h>

#include<stdio.h>

#include "lightbulb.h"

#include "List.h"

#include "tree.h"

//loads a list ,to load the list u want,code must be modified to create o import that list here

void main()

{

node \*root;

List mylist;

initlist(&mylist);

root = NULL;

int n=length(mylist);

int i=0;

for(i;i<n;i++){

insert(&root,getitem(mylist,i));

}

int menu;

do{printf("\n\n1- Add lightbulb \n2- Delete lightbulb \n3- Edit lightbulb \n4- Print tree\n5- Compress tree\n\n");

scanf("%d",&menu);

switch(menu){

case 1: addbulb(&root); break;

case 2: deletebulb(root); break;

case 3: editbulb(root);break;

case 4: printtree(root);break;

case 5: compresstree(root);break;

}}

while(menu<6);

}

void addbulb(node \*\* root){

struct lightbulb bulb;

createBulb(&bulb);

insert(root, bulb);

}

void deletebulb(node \* root){

struct lightbulb find;

createBulb(&find);

node \*tmp;

tmp = search(&root, find);

if (tmp)

{

tmp->data.active=0;

printf("Data succesfully deleted.\n");

}

else

{

printf("Data Not found in tree.\n");

}

}

void editbulb(node \* root){

struct lightbulb find;

createBulb(&find);

node \*tmp;

tmp = search(&root, find);

if (tmp)

{

printf("\nInsert new values\n");

createBulb(&root->data);

printf("Data succesfully edited.\n");

}

else

{

printf("Data Not found in tree.\n");

}

}

void printtree(node \*root){

printf("Pre Order Display\n");

print\_preorder(root);

printf("In Order Display\n");

print\_inorder(root);

printf("Post Order Display\n");

print\_postorder(root);

}

//compress tree,deletion is implemented in logical way, so tree must be compressed in order to really free the memory

void compresstree(node \*root){

node \*newroot;

newroot=NULL;

compress(root,newroot);

root=newroot;

}

void compress(node \*tree,node \*newroot){

if (tree)

{

insert(&newroot,tree->data);

compress(tree->left,newroot);

compress(tree->right,newroot);

}

}

***Tree.h***

struct bin\_tree {

struct lightbulb data;

struct bin\_tree \* right, \* left;

};

typedef struct bin\_tree node;

void insert(node \*\* tree, struct lightbulb val)

{

node \*temp = NULL;

if(!(\*tree))

{

temp = (node \*)malloc(sizeof(node));

temp->left = temp->right = NULL;

temp->data = val;

\*tree = temp;

return;

}

if(val.type < (\*tree)->data.type)

{

insert(&(\*tree)->left, val);

}

else if(val.type > (\*tree)->data.type)

{

insert(&(\*tree)->right, val);

}

}

void print\_preorder(node \* tree)

{

if (tree)

{

printlightBulb(tree);

print\_preorder(tree->left);

print\_preorder(tree->right);

}

}

void print\_inorder(node \* tree)

{

if (tree)

{

print\_inorder(tree->left);

printlightBulb(tree);

print\_inorder(tree->right);

}

}

void print\_postorder(node \* tree)

{

if (tree)

{

print\_postorder(tree->left);

print\_postorder(tree->right);

printlightBulb(tree);

}

}

void printlightBulb(node \* tree){

if(tree->data.active==1){

printf("\nCode:%s\n",tree->data.code);

printf("Watts:%d\n",tree->data.watts);

printf("Type:%d\n",tree->data.type);

printf("Stock:%d\n\n",tree->data.cant);

}

}

void deltree(node \* tree)

{

if (tree)

{

deltree(tree->left);

deltree(tree->right);

free(tree);

}

}

//search and return a lightbulb

node\* search(node \*\* tree, struct lightbulb val)

{

if(!(\*tree))

{

return NULL;

}

if(val.type < (\*tree)->data.type)

{

search(&((\*tree)->left), val);

}

else if(val.type > (\*tree)->data.type)

{

search(&((\*tree)->right), val);

}

else if(val.type == (\*tree)->data.type)

{

return \*tree;

}

}

***List.h***

#ifndef \_LIST\_H

#define \_LIST\_H

struct listitem {

struct lightbulb data;

struct listitem \*next;

};

typedef struct listitem Listitem;

struct list {

Listitem \*head;

};

typedef struct list List;

void initlist (List \*); /\* initialize an empty list \*/

void insertfront(List \* , struct lightbulb val); /\* insert val at front \*/

void insertback(List \*, struct lightbulb val); /\* insert val at back \*/

int length(List); /\* returns list length \*/

void destroy(List \*); /\* deletes list \*/

void setitem(List \*, int n, struct lightbulb val);/\* modifies item at n to val\*/

struct lightbulb getitem(List, int n); /\* returns value at n\*/

#endif /\* \_LIST\_H \*/

void initlist(List \*ilist) {

ilist->head = 0;

}

void insertfront(List \*ilist, struct lightbulb val) {

Listitem \*newitem;

newitem = (Listitem \*)malloc(sizeof(Listitem));

newitem->next = ilist->head;

newitem->data = val;

ilist->head = newitem;

}

void insertback(List \*ilist, struct lightbulb val) {

Listitem \*ptr;

Listitem \*newitem;

newitem = (Listitem \*)malloc(sizeof(Listitem));

newitem->data = val;

newitem->next = 0;

if (!ilist->head) {

ilist->head = newitem;

return;

}

ptr = ilist->head;

while (ptr->next)

{

ptr = ptr->next;

}

ptr->next = newitem;

}

int length(List ilist){ /\* returns list length \*/

Listitem \*ptr;

int count = 1;

if (!ilist.head) return 0;

ptr = ilist.head;

while (ptr->next) {

ptr = ptr->next;

count++;

}

return count;

}

void destroy(List \*ilist) { /\* deletes list \*/

Listitem \*ptr1,\*ptr2;

if (!ilist->head) return; /\* nothing to destroy \*/

ptr1 = ilist->head; /\* destroy one by one \*/

while (ptr1) {

ptr2 = ptr1;

ptr1 = ptr1->next;

free(ptr2);

}

ilist->head = 0;

}

void setitem(List \*ilist, int n, struct lightbulb val){

/\* modifies a value\*/

/\* assume length is at least n long \*/

Listitem \*ptr;

int count = 0;

if (!ilist->head) return;

ptr = ilist->head;

for (count = 0;count < n;count ++)

{

if (ptr) ptr = ptr->next;

else return;

}

if (ptr)

ptr->data = val;

}

struct lightbulb getitem(List ilist, int n) {

/\* returns a list value,

\* assume length is at least n long \*/

Listitem \*ptr;

int count = 0;

ptr = ilist.head;

if (n==0) return ptr->data;

while (ptr->next) {

ptr = ptr->next;

count++;

if (n == count)

return (ptr->data);

}

}

***LightBulb.h***

struct lightbulb{

char code[5];

int watts;

int type;

int cant;

int active; //deletion is implemented in logical way

} ;

void createBulb(struct lightbulb \*bulb){

printf("Enter lightbulb code (max 5 chars). \n");

scanf("%s",bulb->code);

printf("Enter lightbulb watts.\n");

scanf("%d",&bulb->watts);

printf("Enter lightbulb type. \n");

scanf("%d",&bulb->type);

printf("Enter lightbulb stock. \n");

scanf("%d",&bulb->cant);

bulb->active=1;

}