//Michael Weyman

/\*\*This program is used to create a HashTable of values using an array of linked lists\*/

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

//struct for the lists in the arrays

struct list{

struct node \*front;

int size;

};

//struct for the items in the list

struct node{

int value;

//next node in the list

struct node \*next;

};

//function for inserting nodes

struct node \*orderedInsert(struct node \*, int);

//function for destroying lists

void destroy(struct node \*);

//function for printing lists

void print(struct node \*);

//function finds the largest in the hashtable

int findLargest(struct list \*);

//dunction finds the smallest in the hashtable

int findSmallest(struct list \*);

void main(){

int smallest = 0, largest = 0;

double sAverage = 0, lAverage = 0;

int s1 = 0,l1 = 0,smallSum = 0, largeSum = 0;

int q;

//set random number

srand(time(NULL));

//i and j are used in for loops

//small is the smallest in the hashtable

//large is the largest in the hashtable

int i, j, small = 0, large = 0;

//count used for inserting 100 values

//x is used for the random number

int x, count;

//hash value where x is inserted

int hashVal;

for(q = 0; q < 1000; q++){

//create an array of structs

struct list hash[11];

count = 0;

//initializing the structs

for(i = 0; i < 11; i++){

hash[i].front = NULL;

hash[i].size = NULL; }

//creatinng random numbers and inserting

while(count < 101){

x = rand()%1000000;

if(s1 == 0 && l1 == 0){

s1 = x;

l1 = x;

}

//finding largest number

if(x > l1)

l1 = x;

//finding smallest number

if(x < s1)

s1 = x;

hashVal = x % 11;

hash[hashVal].front = orderedInsert(hash[hashVal].front,x);

hash[hashVal].size += 1;

count += 1;

}

//sum of the small numbers

smallSum += s1;

//sum of the large numbers

largeSum += l1;

//finding the smallest value

small = findSmallest(hash);

if (smallest == 0)

smallest = small;

if (small < smallest)

smallest = small;

//finding the largest value

large = findLargest(hash);

if (largest == 0)

largest = large;

if (large > largest)

largest = large;

//destroying the hash table

for(j = 0; j < 11; j++){

destroy(hash[j].front);

}

}

sAverage = (double) smallSum / 1000;

lAverage = (double) largeSum / 1000;

printf("The smallest average: %.2f\nThe largest average: %.2f\n", sAverage, lAverage);

printf("The smallest list: %d\nThe largest list: %d\n", smallest, largest);

}

struct node \*orderedInsert(struct node \*front, int num){

//temp if for new node, temp 2 is for the next node

//tempFront holds the pointer at the fron of the list

struct node \*temp, \*temp2, \*tempFront = front;

//creating a new node

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

//setting the node value

temp->value=num;

//setting the next node to NULL

temp->next=NULL;

//if list is empty set new node to front

if(front==NULL) return temp;

//if new value is lower then front

else if(num < front->value){

//set front node to temp

temp2 = front;

//make the new node the front

front = temp;

//set original front to the next node

front->next = temp2;

}

else{

//if value is higher than front

while(num > front->value && front->next != NULL)

//get location where value fits

front = front->next;

temp2 = front->next;

front->next = temp;

temp->next = temp2;

//return front

return tempFront;

}

}

void destroy(struct node \*front){

//temp node to delete the list

struct node \*temp;

//while going through the list

while(front!=NULL) {

//set the node to delete

temp = front;

//move to next node

front=front->next;

//delete original node

free(temp);

}

}

void print(struct node \*front){

//prints the hashtable

while(front!=NULL) {

printf("%d ", front->value);

front=front->next;

}

}

int findLargest(struct list \*hash){

//finds the largest list

int i, size;

for(i = 0; i < 11; i++){

if(i == 0)

size = hash[i].size;

else if (size < hash[i].size)

size = hash[i].size;

}

return size;

}

int findSmallest(struct list \*hash){

//finds the smallest list

int i, size;

for(i = 0; i < 11; i++){

if(i == 0)

size = hash[i].size;

else if (size > hash[i].size)

size = hash[i].size;

}

return size;

}

/\*\*Over 1000 runs, my smallest average was around 1. My largest average was around 33,000.

my smallest list was either 1 or 0 and my largest was 24 or 25.\*/