Московский Авиационный Институт

(национальный исследовательский университет)

Факультет “Прикладной математики и информатики”

Кафедра 806

Курсовой проект по теме: «Линейные списки»

Выполнил: Студент группы М8О-108Б-20

Борисов Я.А

Проверил: Трубченко Н.М

Москва, 2021

**ОТЧЕТ**

**Цель:** составить и отладить программу на языке Си для обработки линейного списка заданной организации с отображением списка на динамические структуры или на массив. Навигацию по списку следует реализовать с применением итераторов.

**Идея:** идея заключается в том, чтобы реализовать линейный однонаправленный список целых чисел, функции для работы с ним (добавление элемента по позиции, добавление элемента в конец, удаление элемента по позиции, вывод списка, поиск элемента в списке). Так же мне попалось нестандартное действие: ”удалить из списка все элементы, предшествующие и последующие заданному значению”. Все функции были реализованы с помощью итераторов.

Код программы:

List.h

#ifndef KP\_8\_9\_FINAL\_LIST\_H

#define KP\_8\_9\_FINAL\_LIST\_H

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct Node

{

int data;

struct Node\* next;

};

struct List

{

struct Node\* head;

int size;

struct Node\* end;

};

struct List createList(){

struct List list;

list.head = NULL;

list.size = 0;

return list;

}

typedef struct Node\* ListIterator;

void set\_element(struct Node\* \_node, int element){

if (\_node != NULL){

\_node->data = element;

}

}

void set\_next\_element(struct Node\* \_node, struct Node\* \_next){

if (\_node != NULL){

\_node->next = \_next;

}

}

bool is\_list\_empty(struct List\* \_list){

return \_list->size == 0;

}

ListIterator end(struct List\* \_list){

return \_list->end;

}

ListIterator begin(struct List\* \_list){

if (\_list != NULL && \_list->head != NULL){

return \_list->head;

}

else

return end(\_list);

}

ListIterator next(ListIterator It){

return It->next;

}

ListIterator plus(struct List\* \_list, ListIterator It, int \_n){

if (is\_list\_empty(\_list)){

return begin(\_list);

}

ListIterator ansIterator = It;

int i = 0;

while (i < \_n && next(ansIterator) != end(\_list)){

ansIterator = next(ansIterator);

++i;

}

return ansIterator;

}

int distance(ListIterator \_begin, ListIterator \_end){

int ans = 0;

ListIterator curIt = \_begin;

while (curIt != \_end){

++ans;

curIt = next(curIt);

}

return ans;

}

void increase(struct List\* \_list, ListIterator \_lIt, int \_n){

int i = 0;

while (i < \_n && \_lIt != end(\_list)){

\_lIt = next(\_lIt);

}

}

int getListLength(struct List\* \_list){

return distance(begin(\_list), end(\_list));

}

void pushBack(struct List\* \_list, int \_newData){

struct Node\* newNode = (struct Node\*) malloc(sizeof(struct Node));

set\_element(newNode, \_newData);

if (is\_list\_empty(\_list)){

set\_next\_element(newNode, end(\_list));

\_list->head = newNode;

++\_list->size;

return;

}

struct Node\* lastNode = begin(\_list);

while (next(lastNode) != end(\_list)){

lastNode = next(lastNode);

}

set\_next\_element(lastNode, newNode);

set\_next\_element(newNode, end(\_list));

++\_list->size;

}

void popBack(struct List\* \_list){

ListIterator prevNode = plus(\_list, begin(\_list), \_list->size - 2);

if (getListLength(\_list) > 1){

ListIterator deletedNode = next(prevNode);

free(deletedNode);

set\_next\_element(prevNode, end(\_list));

--\_list->size;

}

else if (getListLength(\_list) == 1){

free(\_list->head);

\_list->head = end(\_list);

--\_list->size;

}

}

void insert(struct List\* \_list, int \_newData, int \_pos){

if (\_pos > 0){

if (is\_list\_empty(\_list) || \_pos > getListLength(\_list)){

pushBack(\_list, \_newData);

return;

}

struct Node\* newNode = (struct Node\*) malloc(sizeof(struct Node));

set\_element(newNode, \_newData);

if (\_pos == 1){

set\_next\_element(newNode, \_list->head);

\_list->head = newNode;

++\_list->size;

return;

}

struct Node\* prevNode = plus(\_list, begin(\_list), \_pos - 2);

set\_next\_element(newNode, next(prevNode));

set\_next\_element(prevNode, newNode);

++\_list->size;

}

}

void printList(struct List\* \_list){

printf("list: ");

struct Node\* curNode = begin(\_list);

while (curNode != end(\_list)){

printf("%d ", curNode->data);

curNode = next(curNode);

}

printf("**\n**");

}

ListIterator findPrevNode(struct List\* \_list, int \_data){

ListIterator ans = begin(\_list);

if (ans->data == \_data){

return ans;

}

while (ans != end(\_list)){

if (next(ans)->data == \_data){

return ans;

}

else {

ans = next(ans);

}

}

return ans;

}

bool deleteByValue(struct List\* \_list, int \_data){

ListIterator prevNode = findPrevNode(\_list, \_data);

if (prevNode == begin(\_list)){

if (prevNode == end(\_list)){

return **false**;

}

if (prevNode->data == \_data){

struct Node\* deletedNode = prevNode;

\_list->head = next(deletedNode);

free(deletedNode);

--\_list->size;

return **true**;

}

else {

struct Node\* deletedNode = next(prevNode);

set\_next\_element(prevNode, next(deletedNode));

free(deletedNode);

--\_list->size;

return **true**;

}

}

else if (prevNode != end(\_list)){

struct Node\* deletedNode = next(prevNode);

set\_next\_element(prevNode, next(deletedNode));

free(deletedNode);

--\_list->size;

return **true**;

}

return **false**;

}

void deletePrevAndNext(struct List\* \_list, int \_data){

while (deleteByValue(\_list, \_data - 1));

while (deleteByValue(\_list, \_data + 1));

}

void destroyList(struct List\* \_list){

struct Node\* deletedNode = begin(\_list);

if (deletedNode == end(\_list)){

return;

}

struct Node\* nextNode = next(deletedNode);

while (nextNode != end(\_list)){

printf("%d %d**\n**", deletedNode->data, nextNode->data);

free(deletedNode);

deletedNode = nextNode;

nextNode = next(nextNode);

}

free(deletedNode);

}

int deleteByPosition(struct List\* \_list, int position){

ListIterator Data = begin(\_list);

for(int i = 1; i < position; i++){

Data = next(Data);

}

return Data->data;

}

#endif //KP\_8\_9\_FINAL\_LIST\_H

Main.c

#include "list.h"

int main(){

struct List list = createList();

int symb;

while ((symb = getchar()) != EOF) {

switch (symb) {

case '+': {

getchar();

int newElem;

scanf("%d", &newElem);

getchar();

pushBack(&list, newElem);

**break**;

}

case 'p': {

printList(&list);

**break**;

}

case '-': {

popBack(&list);

**break**;

}

case 'f': {

getchar();

int elem;

scanf("%d", &elem);

deletePrevAndNext(&list, elem);

**break**;

}

case 'i': {

getchar();

int newElem;

scanf("%d", &newElem);

getchar();

int position;

scanf("%d", &position);

insert(&list, newElem, position);

**break**;

}

case 'd':{

getchar();

int position;

scanf("%d", &position);

deleteByValue(&list, deleteByPosition(&list, position));

**break**;

}

}

}

destroyList(&list);

return 0;

}

**Вывод:** составил и отладил программу на языке Си для обработки линейного списка заданной организации с отображением списка на динамические структуры или на массив. Навигацию по списку реализовал с применением итераторов.