ГУАП

КАФЕДРА № 83

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ст. препод. |  |  |  | К.А. Жиданов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2 |
|  |
| по курсу: ЯЗЫКИ ПРОГРАММИРОВАНИЯ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 3145 |  |  |  | А.А. Одинцов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

**Вариант 1**

Двусвязный список (поиск, удаление, вставка)

**Цель работы**

Реализовать АТД (абстрактный тип данных) в виде пользовательского типа данных и набора функций, реализующих заданные операции. Помимо стандартных интерфейсов (чтение/добавление/поиск/удаление), требуется реализовать чтение/выгрузку данных из файла.

**Ход работы**

1. Реализовал на языке Си структуру, выполняющую роль класса для узла:

typedef struct \_Node {

void \*value;

struct \_Node \*next;

struct \_Node \*prev;

} Node;

1. Реализовал на языке Си структуру, выполняющую роль дерева для узлов:

typedef struct \_DblLinkedList {

size\_t size;

Node \*head;

Node \*tail;

} DblLinkedList;

1. Вставка реализована через создание новой структуры:

void pushFront(DblLinkedList \*list, void \*data) {

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

if (tmp == NULL) {

exit(0);

}

tmp->value = data;

tmp->next = list->head;

tmp->prev = NULL;

if (list->head) {

list->head->prev = tmp;

}

list->head = tmp;

if (list->tail == NULL) {

list->tail = tmp;

}

list->size++;

}

void pushBack(DblLinkedList \*list, void \*value) {

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

if (tmp == NULL) {

exit(3);

}

tmp->value = value;

tmp->next = NULL;

tmp->prev = list->tail;

if (list->tail) {

list->tail->next = tmp;

}

list->tail = tmp;

if (list->head == NULL) {

list->head = tmp;

}

list->size++;

}

**4)**Удаление реализовано через отвязывание узла и переноса хвоста или головы списка на следующий элемент:

void\* popFront(DblLinkedList \*list) {

Node \*prev;

void \*tmp;

if (list->head == NULL) {

exit(2);

}

prev = list->head;

list->head = list->head->next;

if (list->head) {

list->head->prev = NULL;

}

if (prev == list->tail) {

list->tail = NULL;

}

tmp = prev->value;

free(prev);

list->size--;

return tmp;

}

void\* popBack(DblLinkedList \*list) {

Node \*next;

void \*tmp;

if (list->tail == NULL) {

exit(4);

}

next = list->tail;

list->tail = list->tail->prev;

if (list->tail) {

list->tail->next = NULL;

}

if (next == list->head) {

list->head = NULL;

}

tmp = next->value;

free(next);

list->size--;

return tmp;

}

**5)**Классический бинарный поиск:

void\* popBack(DblLinkedList \*list) {

Node \*next;

void \*tmp;

if (list->tail == NULL) {

exit(4);

}

next = list->tail;

list->tail = list->tail->prev;

if (list->tail) {

list->tail->next = NULL;

}

if (next == list->head) {

list->head = NULL;

}

tmp = next->value;

free(next);

list->size--;

return tmp;

}

**6)**Полный код программы:

#include <stdio.h>

#include <malloc.h>

#include <assert.h>

#include <conio.h>

typedef struct \_Node {//структура ущла

void \*value;

struct \_Node \*next;

struct \_Node \*prev;

} Node;

typedef struct \_DblLinkedList {//структура листа с началом и концом

size\_t size;

Node \*head;

Node \*tail;

} DblLinkedList;

DblLinkedList\* createDblLinkedList() {

DblLinkedList \*tmp = (DblLinkedList\*) malloc(sizeof(DblLinkedList));

tmp->size = 0;

tmp->head = tmp->tail = NULL;

return tmp;

}

void printInt(void \*value) {//просто функция вывода переменной

printf("%d ", \*((int\*) value));

}

void pushFront(DblLinkedList \*list, void \*data) {

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

if (tmp == NULL) {

exit(0);

}

tmp->value = data;

tmp->next = list->head;

tmp->prev = NULL;

if (list->head) {

list->head->prev = tmp;

}

list->head = tmp;

if (list->tail == NULL) {

list->tail = tmp;

}

list->size++;

}

void\* popFront(DblLinkedList \*list) {

Node \*prev;

void \*tmp;

if (list->head == NULL) {

exit(2);

}

prev = list->head;

list->head = list->head->next;

if (list->head) {

list->head->prev = NULL;

}

if (prev == list->tail) {

list->tail = NULL;

}

tmp = prev->value;

free(prev);

list->size--;

return tmp;

}

void pushBack(DblLinkedList \*list, void \*value) {

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

if (tmp == NULL) {

exit(3);

}

tmp->value = value;

tmp->next = NULL;

tmp->prev = list->tail;

if (list->tail) {

list->tail->next = tmp;

}

list->tail = tmp;

if (list->head == NULL) {

list->head = tmp;

}

list->size++;

}

void\* popBack(DblLinkedList \*list) {

Node \*next;

void \*tmp;

if (list->tail == NULL) {

exit(4);

}

next = list->tail;

list->tail = list->tail->prev;

if (list->tail) {

list->tail->next = NULL;

}

if (next == list->head) {

list->head = NULL;

}

tmp = next->value;

free(next);

list->size--;

return tmp;

}

Node\* search(DblLinkedList \*list, size\_t index) {

Node \*tmp = NULL;

size\_t i;

if (index < list->size/2) {

i = 0;

tmp = list->head;

while (tmp && i < index) {

tmp = tmp->next;

i++;

}

} else {

i = list->size - 1;

tmp = list->tail;

while (tmp && i > index) {

tmp = tmp->prev;

i--;

}

}

return tmp;

}

void insert(DblLinkedList \*list, size\_t index, void \*value) {

Node \*elm = NULL;

Node \*ins = NULL;

elm = search(list, index);

if (elm == NULL) {

exit(5);

}

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

ins->value = value;

ins->prev = elm;

ins->next = elm->next;

if (elm->next) {

elm->next->prev = ins;

}

elm->next = ins;

if (!elm->prev) {

list->head = elm;

}

if (!elm->next) {

list->tail = elm;

}

list->size++;

}

void\* deleteNth(DblLinkedList \*list, size\_t index) {

Node \*elm = NULL;

void \*tmp = NULL;

elm = search(list, index);

if (elm == NULL) {

exit(5);

}

if (elm->prev) {

elm->prev->next = elm->next;

}

if (elm->next) {

elm->next->prev = elm->prev;

}

tmp = elm->value;

if (!elm->prev) {

list->head = elm->next;

}

if (!elm->next) {

list->tail = elm->prev;

}

free(elm);

list->size--;

return tmp;

}

void printDblLinkedList(DblLinkedList \*list, void (\*fun)(void\*)) {

Node \*tmp = list->head;

while (tmp) {

fun(tmp->value);

tmp = tmp->next;

}

printf("\n");

}

int main() {

DblLinkedList \*list = createDblLinkedList();

int a, a2, a3, a4, a5, a6, a7;

a = 10;

a2 = 20;

a3 = 30;

a4 = 40;

a5 = 50;

a6 = 60;

a7 = 70;//записываем значения в лист

pushFront(list, &a);

pushFront(list, &a2);

pushFront(list, &a3);

pushBack(list, &a4);

pushBack(list, &a5);

pushBack(list, &a6);

printDblLinkedList(list, printInt);//выводим лист

printf("length %d\n", list->size);//применяем все функции по заданию

printf("Element 2 %d\n", \*((int\*)(search(list, 2))->value));

printf("Element 5 %d\n", \*((int\*)(search(list, 5))->value));

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

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

printf("head %d\n", \*((int\*)(list->head->value)));

printf("tail %d\n", \*((int\*)(list->tail->value)));

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

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

printf("length %d\n", list->size);

printDblLinkedList(list, printInt);

insert(list, 1, &a7);

printDblLinkedList(list, printInt);

deleteNth(list, 0);

printDblLinkedList(list, printInt);

printf("From file:\n");

int val, n = 0;

FILE \* f = fopen("data.txt","r");//открываем файл и создаём лист из значений в файле

if(!f)

printf("Error open data.txt\n");

else

{

while(!feof(f))

{

fscanf(f,"%d\n",&val);

n++;

}

fclose(f);

}

//int arr[n];

int \*arr = (int\*) malloc(n \* sizeof(int));

f = fopen("data.txt","r");

if(!f)

printf("Error open data.txt\n");

else

{

for(int i = 0; i < n; i++){

if(fscanf(f,"%d\n", &val)){

arr[i] = val;

}

}

}

DblLinkedList \*list2 = createDblLinkedList();

for(int i = 0; i < n; i++){

pushFront(list2, &arr[i]);

}

printDblLinkedList(list2, printInt);

getch();

}

**Вывод**

Изучил методы разработки структур в Си и работу с указателями.