ГУАП
КАФЕДРА №14

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

Должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

тема: динамические структуры данных

по курсу: ОСНОВЫ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ
СТУДЕНТ ГР. 1441

подпись, дата	Лубинец М.И
	инициалы, фамилия

Санкт-Петербург
2015

1. Постановка задачи

Разработать программу, выполняющую операции с деком.
Алгоритмы выполнения каждой операции оформить в виде функции.
Необходимые к реализации операции:

1) Создание структуры из n элементов
2) Проверку структуры на отсутствие в ней элементов
3) Вывод значений элементов структуры на экран.
4) Добавление элемента слева
5) Удаление элемента справа
6) Подсчет количества заказов данного заказчика.

Функция main () должна выполнять тестирование операций с динамической структурой.

При реализации динамических структур использовать связанные списки, которые состоят из элементов, имеющих поля данных и указатель (указатели) на другой элемент списка.

2. Листинг

3. Тестовые примеры

deque.h:

```
#ifndef DEQUE_H
#define DEQUE_H
#include <stdlib.h>


struct deque_s {
    long order_num;        /* 8 bytes */
    long customer_number;  /* 8 bytes */


    struct deque_s *prev; /* 8 bytes */
    struct deque_s *next; /* 8 bytes */
    /* 32 bytes */


    char customer_name[95];
    char not_empty;


    /* 128 bytes == 1/32 mempage size */
};


typedef struct deque_s deque;


/*
 * Makes new deque contains n nodes
 * returns pointer to the front node of new deque
 */
extern deque* interactive_input(int n);


/* makes new node */
extern deque* init(int order_num, int customer_number, char* customer_name);


/* Common double ended list functions */
extern void push_back(deque *node, deque *new_node);
extern void push_front(deque *node, deque *new_node);
extern deque* pop_back(deque *node);
extern deque* pop_front(deque *node);
extern int  is_empty(deque *node);
extern int  size(deque *node);


/* Prints contents of all nodes */
extern void print(deque *node);


/* print contents of one node */
extern void print_node(deque *node);


extern int orders_per_name(deque *node, char* name);


/* Returns pointers to approptiate nodes */
extern deque* front(deque *node);
extern deque* back(deque *node);


extern void clean(deque *node);


#endif // DEQUE_H
```

deque.c:

```
#include "deque.h"
#include <stdio.h>
#include <string.h>


deque* interactive_input(int n){
    deque* root;


    int order_num;
    int customer_number;
    char customer_name[95];


    deque* prev_node = NULL;


    int i = 0;


    do {
        printf("Enter order number:\t");
        scanf("%i", &order_num);


        printf("Enter customer number:\t");
        scanf("%i", &customer_number);


        printf("Enter customer name:\t");


        do {
            fgets(customer_name, 95, stdin);
        } while(customer_name[0] == '\n');


        int len = strlen(customer_name);
        customer_name[len-1] = '\0';


        root = init(order_num, customer_number, customer_name);
        root->prev = prev_node;


        if(prev_node != NULL) prev_node->next = root;


        prev_node = root;
        root = root->next;


        i++;
    } while(i < n);


    return front(prev_node);
}


deque* init(int order_num, int customer_number, char *customer_name){
    deque* node = malloc(sizeof (deque));
    node->order_num = order_num;
    node->customer_number = customer_number;
    strcpy(node->customer_name, customer_name);


    if(strlen(node->customer_name) >= 1)
        node->not_empty = 1;
    else
        node->not_empty = 0;


    return node;
}


void push_back(deque *node, deque *new_node){
    deque* back_node = back(node);
    new_node->prev = back_node;
    back_node->next = new_node;
}


void push_front(deque *node, deque *new_node){
    deque* front_node = front(node);
    new_node->next = front_node;
    front_node->prev = new_node;
}
```

```
deque* pop_back(deque *node){
    if(size(node) <= 1) return NULL;
    deque* back_node = back(node);
    deque* not_very_back_node = back_node->prev;


    if(node == back_node) node = not_very_back_node;


    back_node->prev = NULL;
    not_very_back_node->next = NULL;


    return back_node;
}


deque* pop_front(deque *node){
    if(size(node) <= 1) return NULL;
    deque* front_node = front(node);
    deque* not_very_front_node = front_node->next;


    if(node == front_node) node = not_very_front_node;


    front_node->next = NULL;
    not_very_front_node->prev = NULL;


    return front_node;
}


int is_empty(deque *node){
    if(node == NULL) return 1;


    node = front(node);
    do {
        if(node->not_empty) return 0;
        node = node->next;
    } while(node != NULL);


    return 1;
}


int size(deque *node){
    int size = 0;


    node = front(node);
    do {
        if(node->not_empty){
            size++;
        }
        node = node->next;
    } while(node != NULL);
    return size;
}


void print(deque *node){
    node = front(node);
    do {
        if(node->not_empty){
            print_node(node);
            printf("\n");
        }
        node = node->next;
    } while(node != NULL);
}


void print_node(deque *node){
    printf("Order number:\t%li\n", node->order_num);
    printf("Customer name:\t%s\n", node->customer_name);
    printf("Customer number: %li\n", node->customer_number);
}


deque* front(deque *node) {
    while(node->prev != NULL){
        node = node->prev;
    }
    return node;
}
```

```
deque* back(deque *node){
    while(node->next != NULL){
        node = node->next;
    }
    return node;
}


void clean(deque *node) {
    deque* n = front(node);
    while(n->next != NULL){
        n->not_empty = 0;
        n = n->next;
        n->prev->next = NULL;
        free(n->prev);
        n->prev = NULL;
    }


    n->prev = NULL;
    n->not_empty = 0;
    free(n);


    node = NULL;
}


int orders_per_name(deque *node, char *name){
    int count = 0;


    node = front(node);
    do {
        if(node->not_empty && !strcmp(node->customer_name, name)){
            count++;
        }
        node = node->next;
    } while(node != NULL);
    return count;
}
```

main.c:

```
#include <stdio.h>
#include "deque.h"


int main(void) {
    /* make_deque test */
    printf("Make empty deque with 10 nodes\n");
    deque *orders = interactive_input(2);


    printf("\nCurrent state:\n");
    print(orders);


    printf("This deque is %s", (is_empty(orders)) ? "empty\n" : "not empty\n");


    printf("Customer %s ordered %i items\n", orders->customer_name, orders_per_name(orders, orders-
>customer_name));


    printf("\n");


    /* push_back test */
    printf("Pushing node to back\n");
    push_back(orders, init(3, 2097149, "Mike Lubinets"));


    printf("\nCurrent state:\n");
    print(orders);


    /* push_front test */
    printf("Pushing node to front\n");
    push_front(orders, init(1, 9125270, "Marina Shamis"));


    printf("\nCurrent state:\n");
    print(orders);


    deque* poped;
    /* pop_back test */
    printf("Poping node from back\n");
    poped = pop_back(orders);


    printf("Poped one:\n");
    print_node(poped);


    printf("\nCurrent state:\n");
    print(orders);


    /* pop_front test */
    printf("Poping node from front\n");
    poped = pop_front(orders);
    printf("Poped one:\n");
    print_node(poped);


    printf("\nCurrent state:\n");
    print(orders);


    /* clean test */
    clean(orders);
    printf("Clean deque\n");


    printf("\nCurrent state:\n");
    print(orders);
    printf("This deque is %s", (is_empty(orders)) ? "empty\n" : "not empty\n");


    return 0;
}
```

1) Входные данные:

```
1
5556828
Sergey

2
5556828
Sergey
```

Вывод:

```
Current state:
Order number:    1
Customer name:   Sergey
Customer number: 5556828

Order number:    2
Customer name:   Sergey
Customer number: 5556828

This deque is not empty
Customer Sergey ordered 2 items

Pushing node to back

Current state:
Order number:    1
Customer name:   Sergey
Customer number: 5556828

Order number:    2
Customer name:   Sergey
Customer number: 5556828

Order number:    3
Customer name:   Mike Lubinets
Customer number: 2097149

Pushing node to front

Current state:
Order number:    1
Customer name:   Marina Shamis
Customer number: 9125270

Order number:    1
Customer name:   Sergey
Customer number: 5556828

Order number:    2
Customer name:   Sergey
Customer number: 5556828

Order number:    3
Customer name:   Mike Lubinets
Customer number: 2097149

Poping node from back
Poped one:
Order number:    3
Customer name:   Mike Lubinets
Customer number: 2097149

Current state:
Order number:    1
Customer name:   Marina Shamis
Customer number: 9125270

Order number:    1
Customer name:   Sergey
Customer number: 5556828

Order number:    2
Customer name:   Sergey
Customer number: 5556828

Poping node from front
Poped one:
Order number:    1
Customer name:   Marina Shamis
Customer number: 9125270

Current state:
Order number:    1
Customer name:   Sergey
Customer number: 5556828

Order number:    2
Customer name:   Sergey
Customer number: 5556828

Clean deque

Current state:
This deque is empty
```