1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 2**

1. «**Определение количества вагонов в кольцевом поезде с использованием двусвязного списка**»
2. по дисциплине «Структуры данных»
3. Выполнил
4. студент гр.5151001/40001 Волошкевич М.А.

<*подпись*>

1. Преподаватель Семьянов П.В.

<*подпись*>

1. Санкт-Петербург
2. 2025 г.
3. **Цель работы**

Разработать алгоритм для определения количества вагонов в кольцевом поезде, используя двусвязный список для моделирования замкнутой структуры вагонов и организации движения по ним.

**2. Постановка задачи**

Требуется:

Требуется:

1. Создать модель кольцевого поезда, где каждый вагон представлен элементом двусвязного списка.
2. Реализовать возможность движения по вагонам в обоих направлениях (вперёд и назад).
3. Разработать алгоритм, который на основе изменения состояния освещения вагонов определяет их общее количество.
4. Обеспечить корректную обработку граничных условий (например, один вагон).

**3. Теоретические исследования**

Для решения задачи были изучены:

* **Двусвязные списки**: структура, где каждый элемент содержит указатели на предыдущий и следующий элементы, что позволяет эффективно перемещаться в обоих направлениях.
* **Кольцевые (циклические) списки**: частный случай списка, где последний элемент связан с первым, образуя замкнутый контур. Идеально подходит для моделирования поезда.

**4. Описание решения**

**4.1. Структуры данных**

* element:
  + next — указатель на следующий вагон.
  + previous — указатель на предыдущий вагон.
  + data — состояние света ('0' — выключен, '1' — включен).
* list:
  + head — указатель на первый вагон.
  + tail — указатель на последний вагон.

**4.2. Ключевые функции**

1. add\_to\_end — добавляет вагон в конец списка, связывая его с предыдущим и головным вагоном для обеспечения кольцевой структуры.
2. move — перемещает текущий указатель на следующий или предыдущий вагон в зависимости от направления (1 — вперёд, 0 — назад).

****4.3 Алгоритм подсчёта вагонов**:**

* + Начинаем с первого вагона, включаем в нём свет ('1').
  + Двигаемся вперёд на count вагонов, выключаем свет ('0').
  + Возвращаемся назад на count вагонов.
  + Увеличиваем count на 1 и повторяем процесс, пока не обнаружим, что свет в начальном вагоне выключен.

Сложность алгоритма можно вычислить, зная что в каждый вагон надо пройти туда и обратно, получается 2 раза. Для каждого последующего нужно на 2 пути больше, соответственно получается для, например 3 вагонов у нас получится последовательность: 2 раза пройти для первого, 4 для второго, 6 для третьего = 2 + 4 + 6. Двойку можно вынести, соответственно получается 2( 1 + 2 + 3 … n), значит сумма будет 2 \* ((1 + n) / 2) \* n, двойки сокращаются и сложность выходит n^2 + n

**4.4. Пример работы**

* **Входные данные**: "0000" (4 вагона).
  1. Включён свет в вагоне 1: "1000".
  2. Двигаемся на 1 вагон вперёд, выключаем свет: "1000" → "1000".
  3. Возвращаемся на 1 вагон назад. Свет в начальном вагоне остаётся включённым.
  4. Увеличиваем count до 2 и повторяем.
  5. Конечный результат: count = 4.

**5. Тестирование и результаты работы программы**

**1) Проверка корректности работы**

* **Входные данные**: "0000000000" (10 вагонов с выключенным светом).
* **Ожидаемый результат**: 10.
* **Фактический результат**: 10.

**2) Проверка на другом наборе данных**

* **Входные данные**: "010" (3 вагона).
* **Ожидаемый результат**: 3.
* **Фактический результат**: 3.

**3) Проверка на минимальном наборе**

* **Входные данные**: "1" (1 вагон).
* **Ожидаемый результат**: 1.
* **Фактический результат**: 1.

**Выводы**

Программа корректно определяет количество вагонов в поезде для различных входных данных. Результаты соответствуют ожиданиям.

**6. Вывод**

Была успешно реализована программа, использующая циклический список для моделирования управления освещением в поезде. Программа прошла тестирование на различных входных данных, подтвердив свою корректность. Для дальнейшего улучшения можно добавить обработку ошибок (например, проверку на некорректные символы во входной строке).

**7. Приложения**

**Приложение1:**

#pragma once

#include <stdio.h>

#include <stdlib.h>

typedef struct element

{

struct element \*next;

struct element \*previous;

char data;

}element;

typedef struct list

{

element \*head;

element \*tail;

}list;

void add\_to\_begin(list \*list, char data)

{

element \*new\_element = (element\*) malloc(sizeof(element));

new\_element->data = data;

if (list->head == NULL)

{

new\_element->next = new\_element;

new\_element->previous = new\_element;

list->head = new\_element;

list->tail = new\_element;

}

else

{

new\_element->next = list->head;

new\_element->previous = list->tail;

list->head = new\_element;

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

}

}

void add\_to\_end(list \*list, char data)

{

if (list->head == NULL)

{

add\_to\_begin(list, data);

return;

}

element \*new\_element = (element\*) malloc(sizeof(element));

new\_element->data = data;

new\_element->next = list->head;

new\_element->previous = list->tail;

list->tail->next = new\_element;

list->tail = new\_element;

}

void list\_init(list \*list, char \*str)

{

list->head = NULL;

list->tail = NULL;

int i = 0;

while (str[i] != '\0')

{

add\_to\_end(list, str[i]);

i++;

}

}

element \*move(int dir, element \*current)

{

return dir == 1 ? current->next : current->previous;

}

void print\_list(list\* list)

{

element \*current = list->head;

printf("%c", current->data);

while(current != list->tail)

{

current = current->next;

printf("%c", current->data);

}

printf("\n");

}

**Приложение2:**

#include "list.h"

#define TRAIN\_SIZE 100

char train\_on = '1';

char train\_off = '0';

int main()

{

list train;

char train\_str[TRAIN\_SIZE] = "";

int index = 0;

int count = 0;

scanf("%s", train\_str);

list\_init(&train, train\_str);

element \*start = train.head;

start->data = train\_on;

count++;

element \*current = start;

while (1)

{

index += count;

for (int i = 0; i < index; i++)

{

current = move(1, current);

}

current->data = train\_off;

for (int i = index; i > index - count; i--)

{

current = move(0, current);

}

index -= count;

if (start->data == train\_off) break;

count++;

}

printf("%d \n", count);

return 0;

}