Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №2

з дисципліни
«Алгоритми і структури даних»

Виконав:

студент групи ІМ-43
Балалаєв Максим Юрійович
номер варіанту: 3

Перевірив:

Сергієнко А. М.

Київ 2025

Завдання

1. Створити список з ***n*** (***n*** > 0) елементів (***n*** вводиться з клавіатури), якщо інша кількість елементів не вказана у конкретному завданні за варіантом.
2. Тип ключів (інформаційних полів) задано за варіантом.
3. Вид списку (черга, стек, дек, прямий однозв'язний лінійний список, обернений однозв'язний лінійний список, двозв'язний лінійний список, однозв'язний кільцевий список, двозв'язний кільцевий список) вибрати самостійно з метою найбільш доцільного розв'язку поставленої за варіантом задачі.
4. Створити функції (або процедури) для роботи зі списком (для створення, обробки, додавання чи видалення елементів, виводу даних зі списку в консоль, звільнення пам'яті тощо).
5. Значення елементів списку взяти самостійно такими, щоб можна було продемонструвати коректність роботи алгоритму програми. Введення значень елементів списку можна виконати довільним способом (випадкові числа, формування значень за формулою, введення з файлу чи з клавіатури)
6. Виконати над створеним списком дії, вказані за варіантом, та коректне звільнення пам'яті списку.
7. **При виконанні заданих дій, виводі значень елементів та звільненні пам'яті списку вважати, що довжина списку (кількість елементів) *невідома* на момент виконання цих дій.** Тобто, не дозволяється зберігати довжину списку як константу, змінну чи додаткове поле.

Варіант №3

Ключами елементів списку є символи з множини латинських літер та цифр. Перекомпонувати список таким чином, щоб усі цифри стояли на початку списку, не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»).

Текст програми

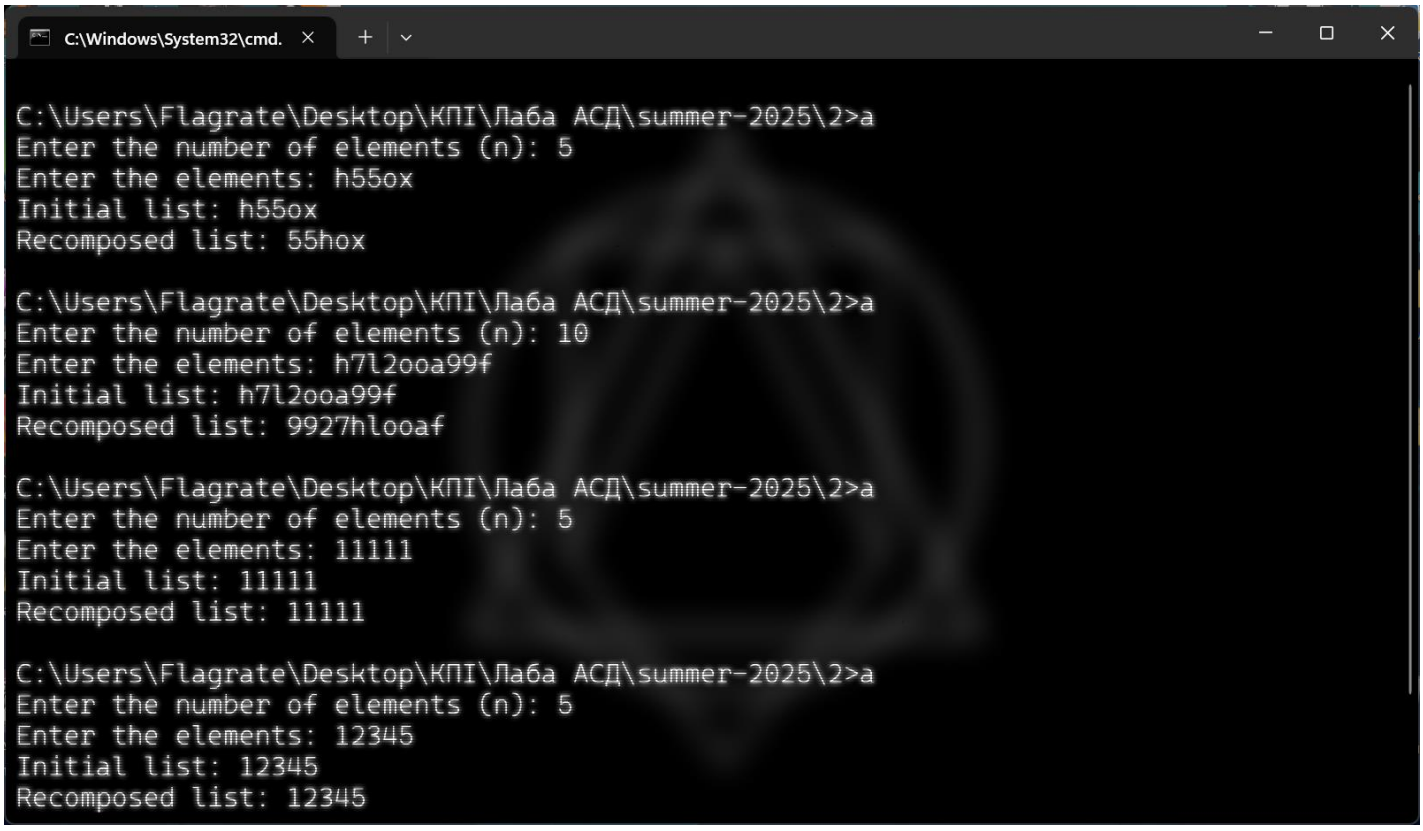
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct linked_list
{
    char data;
    struct linked_list *next;
} Node;

Node *add_to_start(Node *head, char data)
```

```
{
    Node *new_node = (Node *)malloc(sizeof(Node));
    new_node->data = data;
    new_node->next = head;

    return new_node;
}

Node *remove_node(Node *head, Node *node, Node *prev)
{
    if (head == node)
    {
        head = head->next;
        free(node);
        return head;
    }

    if (prev != NULL)
        prev->next = node->next;

    free(node);
    return head;
}

Node *init_list(const char *data)
{
    Node *head = NULL;

    const unsigned int len = strlen(data);
    for (unsigned int i = len; i > 0; i--)
    {
        head = add_to_start(head, data[i - 1]);
    }

    return head;
}

void print_list(Node *head)
{
    Node *current = head;
    while (current != NULL)
    {
        printf("%c", current->data);
        current = current->next;
    }
    printf("\n");
}

void delete_list(Node *head)
{
    while (head != NULL)
    {
        Node *temp = head;
```

```
        head = head->next;
        free(temp);
    }
}

int is_digit(char c)
{
    return c >= '0' && c <= '9';
}

int is_list_numeric(Node *head)
{
    Node *current = head;
    while (current != NULL)
    {
        if (!is_digit(current->data))
            return 0;
        current = current->next;
    }
    return 1;
}

Node *recompose(Node *head)
{
    Node *current = head;
    Node *prev = NULL;

    if (is_list_numeric(head))
        return head;

    while (current != NULL)
    {
        Node *next = current->next;

        if (current->data >= '0' && current->data <= '9')
        {
            const char digit = current->data;
            head = remove_node(head, current, prev);
            head = add_to_start(head, digit);
        }
        else
        {
            prev = current;
        }

        current = next;
    }

    return head;
}

int main()
{
```

```
    unsigned int n;
    printf("Enter the number of elements (n): ");
    scanf("%u", &n);

    char *data = (char *)malloc(n + 1);
    if (!data)
    {
        printf("Memory allocation failed\n");
        return 1;
    }

    printf("Enter the elements: ");
    getchar();
    fgets(data, n + 1, stdin);

    Node *linked_list = init_list(data);
    printf("Initial list: ");
    print_list(linked_list);

    linked_list = recompose(linked_list);
    printf("Recomposed list: ");
    print_list(linked_list);

    delete_list(linked_list);
    free(data);

    return 0;
}
```

Тестування програми



Висновки

У цій лабораторній роботі було реалізовано динамічну структуру даних – однозв’язний список символів (*char*) мовою програмування С. Також було створено функції для оперування над цією структурою даних: додавання в початок, видалення вузла, створення та виведення списку, звільнення використаної пам’яті та рекомпозиція – переміщення всіх цифр у початок списку.

Однозв’язний лінійний список – класична реалізація динамічної структури даних, в якій інформація зберігається у вузлах – *Node*. Кожен вузол – це користувацький тип даних, який має два поля: власне дані та вказівник на наступний вузол. Через це доступ до списку можливий лише «з голови», і «передавати» список та оперувати ним можна лише за вказівником на перший вузол.

Під час виконання роботи було виявлено межевий випадок, який призводить до поламки програми: коли всі вхідні дані – числа (тобто вхідний рядок складається лише з цифр). Через це програма намагалась пересувати цифри на початок, виявляючи їх знову й знову, через що її робота тривала нескінченно. Для уникнення цієї проблеми було додано функцію-перевірку *is_list_numeric*.

У роботі також було використано такі концепції програмування, як вказівники (*pointers*) та динамічне виділення пам’яті. Це потужні інструменти, які дозволяють працювати з оперативною пам’яттю на більш низькому рівні, пропонуючи високий рівень контролю. Однак неуважність або необережність у використанні цих інструментів можуть призвести до таких проблем, як «висячі» вказівники (вказівники, що вказують на ніщо) або витоки пам’яті (пам’ять під дані, що вже більше не використовуватимуться, не була звільнена).