Подготовка к КР

1. **Parse IP**

   - Объявите структуру IP_address, описывающую IP-адрес. IP-адрес обычно записывается как 4 десятичных числа, разделённых точками. Диапазон изменения каждого из чисел: от 0 до 255. Пример: 192.168.5.1.
   - Предположим, что информация об IP-адресе записана в строке: `char str[50] = ‘‘192.168.5.1’’`. Создать экземпляр структуры IP и считать информацию из строки в эту структуру с помощью sscanf().
   - Напишите функцию `IP_address parse_point(char* str)`, которая считывает информацию из строки `str` в структуру и возвращает эту структуру. Нужно использовать функцию `sscanf()`.

2. **Malloc + Reverse** На вход программе подаётся целое число n и n вещественных чисел типа int. Нужно эти числа напечатать в обратном порядке. Всю необходимую память выделить динамически.

3. **Malloc 2D** На вход программе подаются 2 целых числа n и m. Затем на вход подаются $n \times m$ вещественных чисел типа int. Нужно напечатать матрицу, отзеркаленную по вертикали Всю необходимую память выделить динамически.

Связный список

```c
struct node {
    int val;
    struct node* next;
};
typedef struct node Node;

// Вам нужно написать соответствующие функции
// ...
int main()
{
    Node* head = list_create();
    list_add_first(&head, 14);
    for (int i = 0; i < 5; i++)
    {
        list_add_first(&head, i);
        list_add_last(&head, 10 + i);
    }
    printf("%d\n", list_remove_last(&head));
    list_print(head);
    list_reverse(&head);
    list_print(head);
}
```

```c
void list_add_last(Node** p_head, int x)
{
    // Выделяем память на новый элемент
    Node* p_new_node = malloc(sizeof(Node));
    p_new_node->val = x;
    p_new_node->next = NULL;

    // Создаём указатель на первый элемент
    Node* ptr = *p_head;
    if (ptr == NULL)
    {
        *p_head = p_new_node;
    }
    else
    {
        // Идём до последнего элемента
        while (ptr->next != NULL)
            ptr = ptr->next;
        ptr->next = p_new_node;
    }
}
```

Задачи на связный список:

1. Написать функцию `Node* list_create()`, которая инициализирует список (просто возвращает NULL).

2. Написать функцию `void list_add_first(Node** p_head, int x)`, которая добавляет элемент x в начало списка. Чтобы добавить элемент, нужно для начала выделить необходимое количество памяти под этот элемент, затем задать поля нового элемента таким образом, чтобы он указывал на начало списка. В конце нужно поменять значение указателя на начало списка. Обратите внимание, что так как нужно изменить значение указателя, то в эту функцию нужно передавать указатель на указатель.

3. Написать функцию `void list_add_last(Node** p_head, int x)`, которая добавляет элемент x в конец списка.

4. Написать функцию `int list_remove_first(Node** p_head)`, которая удаляет элемент из начала списка и возвращает его значение.
5. Написать функцию `int list_remove_last(Node** p_head)`, которая удаляет элемент из конца списка и возвращает его значение.
6. Написать функцию `void list_print(Node* head)`, которая распечатывает все элементы списка.
7. Написать функцию `int list_size(Node* head)`, которая возвращает количество элементов списка.
8. Написать функцию `int list_is_empty(Node* head)`, которая возвращает 0 если список пуст и 1 если не пуст.
9. Написать функцию `int list_destroy(Node* head)`, которая освобождает всю память, выделенную под список. Так как память выделялась под каждый элемент отдельно, то освобождать нужно также каждый элемент по отдельности.
10. Написать функцию `void list_reverse(Node** p_head)`, которая переворачивает связный список. Первый элемент становится последним, а последний первым.
11. Создать структуру Stack которая будет содержать 1 элемент: указатель на Node. Написать функции `void stack_push(Stack* s, int x)`, `int stack_pop(Stack* s)` и `void stack_print(Stack* s)`, используя функции работы со списком.
12. Проверить работу программы с помощью valgrind. Для этого выполните в терминале `valgrind ./<имя программы>`
13. Написать функцию `int list_concatenate(Node** p_head1, Node** p_head2)`, которая добавляет второй связный список в конец первого.
14. Написать функцию `int list_is_loop(Node* p_head)`, которая проверяет, если в связном списке цикл.