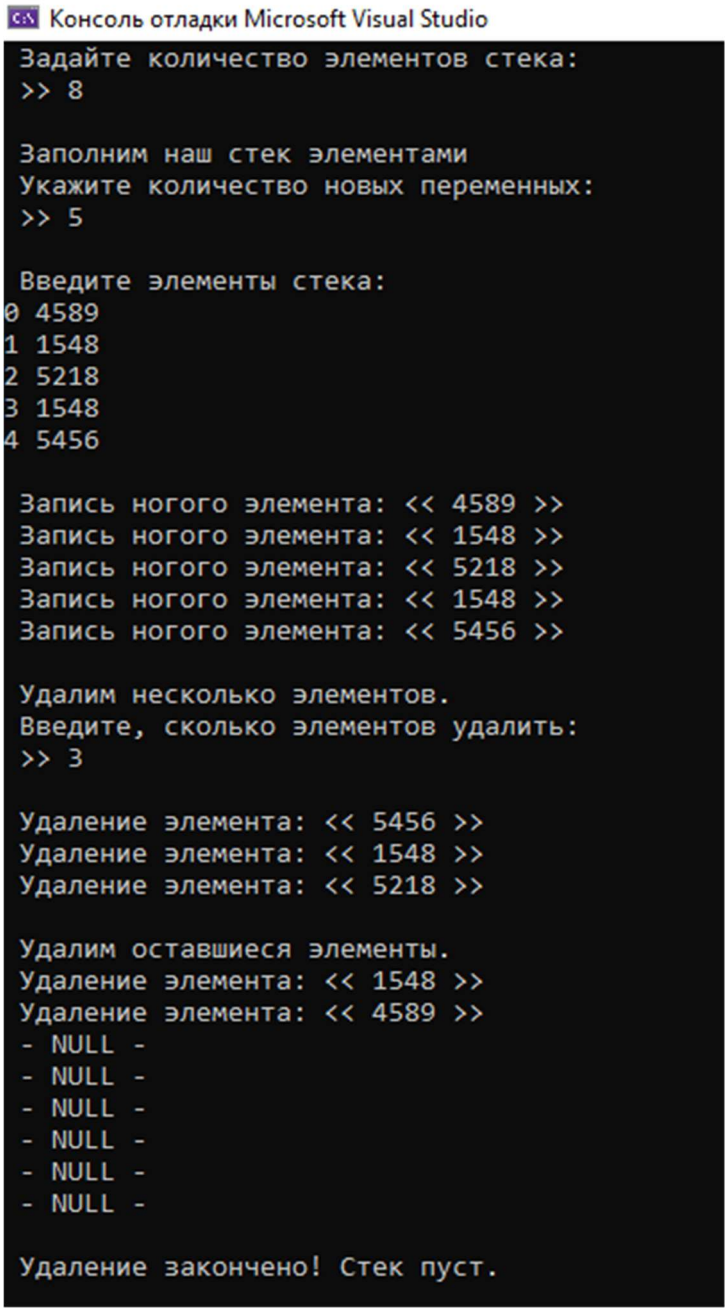
Вариант 2 – FIFO

Задание:

Список должен реализовать принцип FIFO. Список должен быть ограничен.
Функции (методы) – запись данных в очередь, выборка данных из очереди, проверка на пустоту, проверка на заполнение (переполнение).

Ответ (пример работы кода):



Мой код:

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

struct Node
{
    int data;
    Node* next;
};

int nodesCount;

/* Вспомогательная функция для добавления элемента в начало стека */
void push(struct Node** top, const int* mas, int p, int i, int& nodesCount)
{
    if (nodesCount < p)
    {
        Node* node = NULL;
        node = (struct Node*)malloc(sizeof(struct Node));

        cout << " Запись нового элемента: << " << mas[i] << " >>" << endl;

        node->data = mas[i]; // Вносим даные в узел
        node->next = *top;   // Сдвигаемся в структуре стека
        *top = node;         // Создаем новую голову
        nodesCount += 1;     // Увеличиваем размер стека на 1
    }
    else
    {
        cout << "\nОшибка! Стек переполнен\n" << endl;
    }
}

/* Вспомогательная функция для проверки заполненности стека */
int CheckStack(struct Node* top)
{
    return top == NULL;
}

/* Вспомогательная функция для возврата верхнего элемента stack */
int peek(struct Node* top)
{
    if (!CheckStack(top))
    {
        return top->data;
    }
    else
    {
        cout << "\nОшибка! Стек переполнен\n" << endl;
    }
}

/* Вспомогательная функция для извлечения верхнего элемента из stack */
int pop(struct Node** top)
{
    struct Node* node;

    if (*top == NULL)
    {
        printf(" - NULL - \n");
        return 0;
```

```
    }

    int x = peek(*top);
    cout << " Удаление элемента: << " << x << " >>" << endl;

    node = *top;
    *top = (*top)->next;
    nodesCount -= 1;

    free(node);

    return x;
}

/* Вспомогательная функция для возврата числа узлов стека */
int size()
{
    return nodesCount;
}

void main()
{
    setlocale(LC_ALL, "Russian");
    struct Node* top = NULL;
    int sizeStek, b, deleteDataCount, newDataCoutn;

    /* Создали стек */
    cout << " Задайте количество элементов стека: " << endl << " >> "; cin >>
sizeStek;
    int* mas = new int[sizeStek];//создание одномерного динамического массива

    /* Заполнили стек */
    cout << "\n Заполним наш стек элементами" << endl;

    if (nodesCount == sizeStek)
    {
        cout << "Стек переполнен!" << endl;
    }

    cout << " Укажите количество новых переменных:" << endl << " >> "; cin >>
newDataCoutn;

    if (newDataCoutn > sizeStek - nodesCount)
    {
        cout << "Стек будет переполнен! Слишком много элементов." << endl;
    }

    cout << "\n Введите элементы стека:" << endl;

    for (int i = 0; i < newDataCoutn; i++)
    {
        cout << i << " ";  cin >> mas[i];
    }
    cout << endl;

    for (int i = 0; i < newDataCoutn; i++)
    {
        push(&top, mas, newDataCoutn, i, nodesCount);
    }

    /* Удалим несколько элементов */
    cout << "\n Удалим несколько элементов." << endl;
    if (CheckStack(top))
    {
        cout << "Ошибка! Стек пуст." << endl;
```

```
    }

    cout << " Введите, сколько элементов удалить:" << endl << " >> "; cin >>
deleteDataCount; cout << endl;
    for (int i = 0; i < deleteDataCount; i++)
    {
        pop(&top);
    }

    /* Очистим стек (удалим все элементы) */
    cout << "\n Удалим оставшиеся элементы." << endl;
    if (CheckStack(top))
    {
        cout << "Ошибка! Стек пуст." << endl;
    }

    for (int i = 0; i < sizeStek; i++)
    {
        pop(&top);
    }

    /* Проверим что стек действительно пуст */
    if (CheckStack(top))
    {
        cout << "\n Удаление закончено! Стек пуст." << endl;
    }
}
```