# Лабораторна робота № 3

# Організація лінійних однозв’язних списків.

**Завдання**.

Розробити програму, в якій реалізувати обробку лінійного однозв'язного списку, елементами якого є цілі числа.

**Обов’язкові вимоги**:

* після завантаження програма пропонує користувачеві меню для вибору дії з такими пунктами:
  + Додати елемент до списку.
    - *для парних варіантів – на початок списку,*
    - *для непарних варіантів -* *в кінець списку.*

Значення елементу, що додається, вводити з клавіатури.

* + Видалити елемент із списку.
    - *для парних варіантів – за номером елемента в списку.* Номер вводити з клавіатури.
    - *для непарних варіантів - за значенням*. Значення елементу вводити з клавіатури. Видалити перший елемент з заданим значенням, який зустрівся.
  + Вивести елементи списку на екран.
  + Очистити список (видалити всі елементи).
  + *Пункт щодо індивідуального завдання згідно з варіантом*;
  + Завершення роботи.
* Організувати роботу програми таким чином, щоб перелічені вище дії можна було б виконувати неодноразово за один сеанс роботи програми. Як організувати меню – див. лабораторні роботи № 1, 2.
* При видаленні елементів передбачити обробку помилкових ситуацій, коли будуть виконуватися спроби видалення елемента із порожньої структури.

**Індивідуальні завдання за варіантами.**

При виконанні завдань передбачити вивід на екран списку до та після перетворень за завданнями.

|  |  |
| --- | --- |
| № вар. | Завдання |
| 1 | Реалізувати алгоритм, який видаляє кожний другий елемент списку. |
| 2 | Реалізувати алгоритм, який знаходить мінімальний (перший, якщо їх декілька) та максимальний (останній, якщо їх декілька) елементи списку, залишає в списку тільки елементи між знайденими мінімальним та максимальним елементами (включаючи їх), решту вилучає. |
| 3 | Реалізувати алгоритм, який дублює в списку всі непарні елементи, тобто за кожним непарним елементом має йти ще один, рівний йому. |
| 4 | Реалізувати алгоритм, який знаходить мінімальний (перший, якщо їх декілька) та максимальний (останній, якщо їх декілька) елементи списку. Мінімальний елемент переставляє на початок списку, максимальний – в кінець списку. Кількість елементів у списку не має змінюватися. |
| 5 | Реалізувати алгоритм, який перевертає список, тобто змінює посилання таким чином, щоб елементи виявилися розташованими в зворотному порядку. Змінювати тільки вказівники, елементи не переписувати. |
| 6 | Реалізувати алгоритм, який розраховує середнє арифметичне всіх елементів списку та додає його на початок списку. |
| 7 | Реалізувати алгоритм, який видаляє всі від’ємні елементи списку. |
| 8 | Реалізувати алгоритм, який видаляє кожний третій елемент списку. |
| 9 | Реалізувати алгоритм, який видаляє елементи списку, значення яких лежить в заданому діапазоні. Границі діапазону вводити з клавіатури. |
| 10 | Реалізувати алгоритм, який знаходить суму останнього та передостаннього елементів списку та додає її на початок списку. |
| 11 | Реалізувати алгоритм, який додає після кожного від’ємного елемента його модуль. |
| 12 | Реалізувати алгоритм, який переставляє (міняє місцями) перший та останній елементи списку. Змінювати тільки вказівники, елементи не переписувати. |
| 13 | Реалізувати алгоритм, який знаходить мінімальний (перший, якщо їх декілька) та максимальний (останній, якщо їх декілька) елементи списку та видаляє всі елементи, що розташовані між знайденими мінімальним та максимальним. |
| 14 | Реалізувати алгоритм, який знаходить мінімальний елемент *x*min списку та перетворити кожний елемент списку *x* на (*x*- *x*min). |
| 15 | Реалізувати алгоритм, який після кожного парного елемента вставляє елемент із значенням 0. |

# Методичні рекомендації.

Завдання елемента списку може бути таким:

struct element

{int data; // поле даних, в даному випадку – цілого типу

element \*next; // вказівник на наступний елемент списку

}

element \*head; // заголовок - вказівник на перший елемент

// списку, первісно head=NULL

Нижче наведені фрагменти програм для виконання деяких дій з елементами однозв'язного списку.

1. Приклад створення нового елемента:

element \*newnode=new element; // виділяємо пам'ять під новий елемент

newnode->data=*x*; // записуємо дані елемента

newnode->next=NULL;

2. Якщо елемент вставляється на початок списку, то

newnode->next=head;

head=newnode;

3. Якщо елемент вставляється в кінець списку, то

element \*node=head;

if (head==NULL) head=newnode;

else

{

while (node->next!=NULL)

{

node=node->next;

}

node->next=newnode;

}

4. Для видалення елемента зі значенням *х* із списку:

element \*node=head;

element \*prev\_node=NULL;

if (node==NULL) // немає елементів в списку

{ cout<<" Немає елементів в списку " << endl; }

else

{

while (node->next!=NULL)

{

if (node->data==*x*) break;

prev\_node=node;

node=node->next;

}

if (node->data==*x*) // елемент для видалення знайдений

{

if (prev\_node==NULL) head=node->next;

// если элемент 1-й

else prev\_node->next=node->next;

delete node;

}

else cout <<"Елемент *x*="<< *x* << " в списку не знайдено." <<endl;

}

Контрольні питання

1. Що представляє собою «список» як логічна структура даних?
2. Які основні функції для роботи зі списками?
3. Переваги і недоліки організації списків на основі масивів.
4. Дати характеристику однозв’язного списку. Структура елемента.
5. Дати характеристику двозв’язного списку. Структура елемента.
6. Види двозв’язних списків.
7. Переваги і недоліки організації списків за допомогою вказівників.
8. Як організуються зв’язані списки в пам’яті комп’ютера?
9. Порівняння реалізації списку на базі масиву та на базі однозв’язного списку.
10. Ефективність операцій вставки, видалення та пошуку елементів в зв’язаних списках.
11. Ефективність операцій вставки, видалення та пошуку елементів в списках на базі масиву.
12. Наведіть порівняльний аналіз вивчених способів організації списків.