Зв'язані списки

ПЛАН

1. Поняття та класифікація зв'язаних списків
2. Однозв'язні лінійні списки
3. Двозв'язні лінійні списки

1. ПОНЯТТЯ ТА КЛАСИФІКАЦІЯ ЗВ’ЯЗАНИХ СПИСКІВ

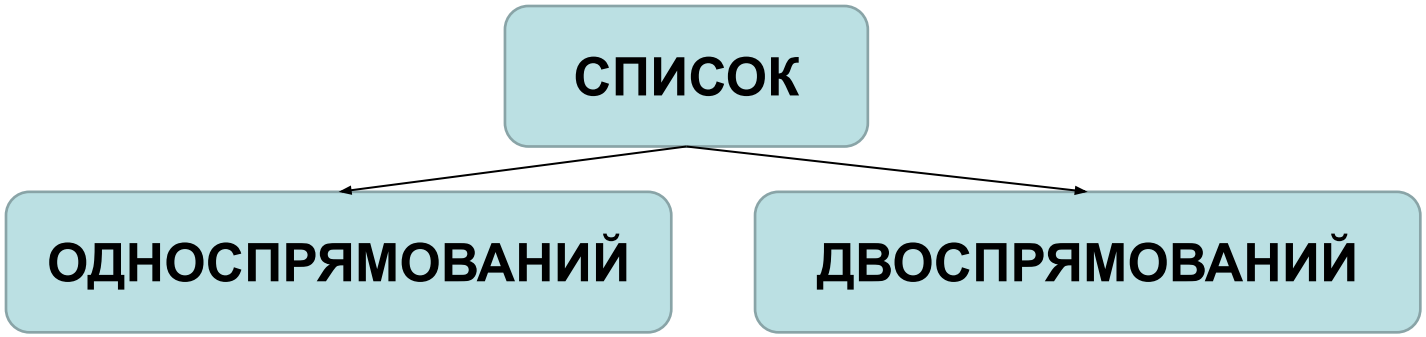
Зв'язний список є найпростішим типом даних динамічної структури, що складається з елементів (вузлів). Кожен вузол включає в себе в класичному варіанті два поля:

- дані (в якості даних може виступати змінна, об'єкт класу або структури і т. д.)
- покажчик на наступний вузол в списку.

Доступ до списку здійснюється через покажчик, який містить адресу першого елемента списку, званий коренем списку.

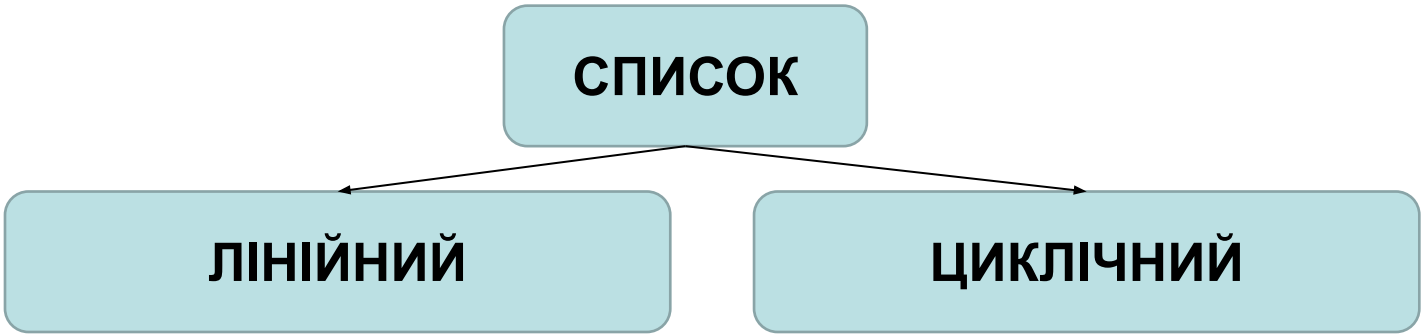
Класифікація списків

За **кількістю полів** покажчиків розрізняють односпрямований (однозв'язний) і двонаправлений (двузв’язаний) списки.



Зв'язний список, що містить тільки один покажчик на наступний елемент, називається одинзв'язний.
Зв'язний список, що містить два поля покажчика - на наступний елемент і на попередній, називається двузв’язний.

За **способом зв'язку** елементів розрізняють лінійні і циклічні списки.

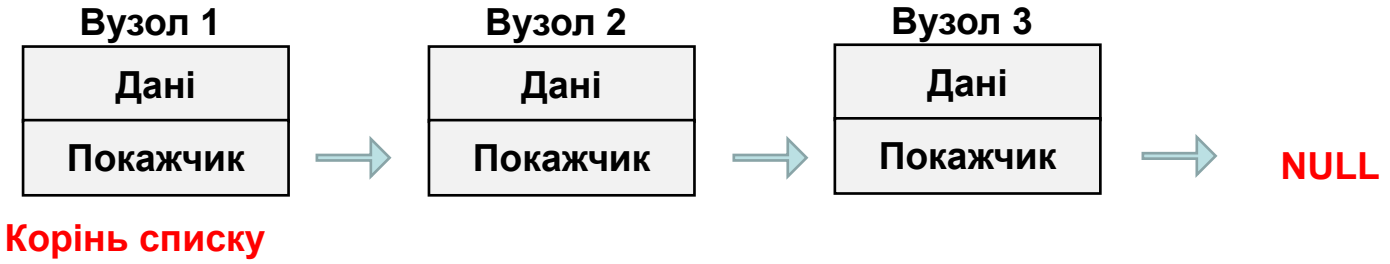


Зв'язний список, в якому, останній елемент вказує на NULL, називається лінійним.

Зв'язний список, в якому останній елемент пов'язаний з першим, називається циклічним.

види списків

Однозв'язний лінійний список (ОЛС).
Кожен вузол ОЛС містить 1 поле покажчика на наступний вузол. Поле покажчика останнього вузла містить нульове значення (вказує на NULL).



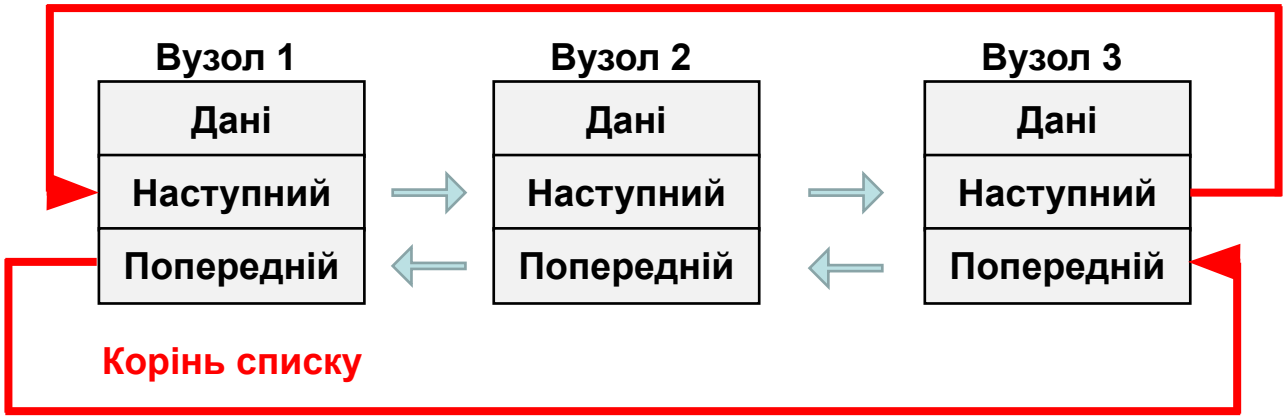
Однозв'язний циклічний список (ОЦС).
Кожен вузол ОЦС містить 1 поле покажчика на наступний вузол. Поле покажчика останнього вузла містить адресу першого вузла (кореня списку).

Двозв’яний лінійний список (ДЛС).

Кожен вузол ДЛС містить два поля покажчиків: на наступний і на попередній вузол. Поле покажчика на наступний вузол останнього вузла містить нульове значення (вказує на NULL). Поле покажчика на попередній вузол першого вузла (кореня списку) також містить нульове значення (вказує на NULL).

Двозв’яний циклічний список (ДЦС).

Кожен вузол ДЦС містить два поля покажчиків: на наступний і на попередній вузол. Поле покажчика на наступний вузол останнього вузла містить адресу першого вузла (кореня списку). Поле покажчика на попередній вузол першого вузла (кореня списку) містить адресу останнього вузла



МАСИВ	СПИСОК
Виділення пам'яті здійснюється одноразово під весь масив до початку його використання	Виділення пам'яті здійснюється в міру введення нових елементів
При видаленні / додаванні елемента потрібно копіювання всіх наступних елементів для здійснення їх зсуву	Видалення / додавання елемента здійснюється перевстановлення покажчиків, при цьому самі дані не копіюються
Для зберігання елемента потрібно обсяг пам'яті, необхідний тільки для зберігання даних цього елемента	Для зберігання елемента потрібно обсяг пам'яті, достатній для зберігання даних цього елемента і покажчиків (1 або 2) на інші елементи списку
Доступ до елементів може здійснюватися в довільному порядку	Можливий тільки послідовний доступ до елементів

2. Однозв'язні лінійні списки

Вузол ОЛС можна представити у вигляді структури

```
  struct list
{
   int field; // Поле даних
   struct list * ptr; // Покажчик на наступний
елемент
};
```

Основні дії, вироблені над елементами ОЛС:

- Ініціалізація списку
- Додавання вузла в список
- Видалення вузла зі списку
- Видалення кореня списку
- Висновок елементів списку
- Взаємообмін двох вузлів списку

Ініціалізація ОЛС

Ініціалізація списку призначена для створення кореневого вузла списку, у якого поле покажчика на наступний елемент містить нульове значення.

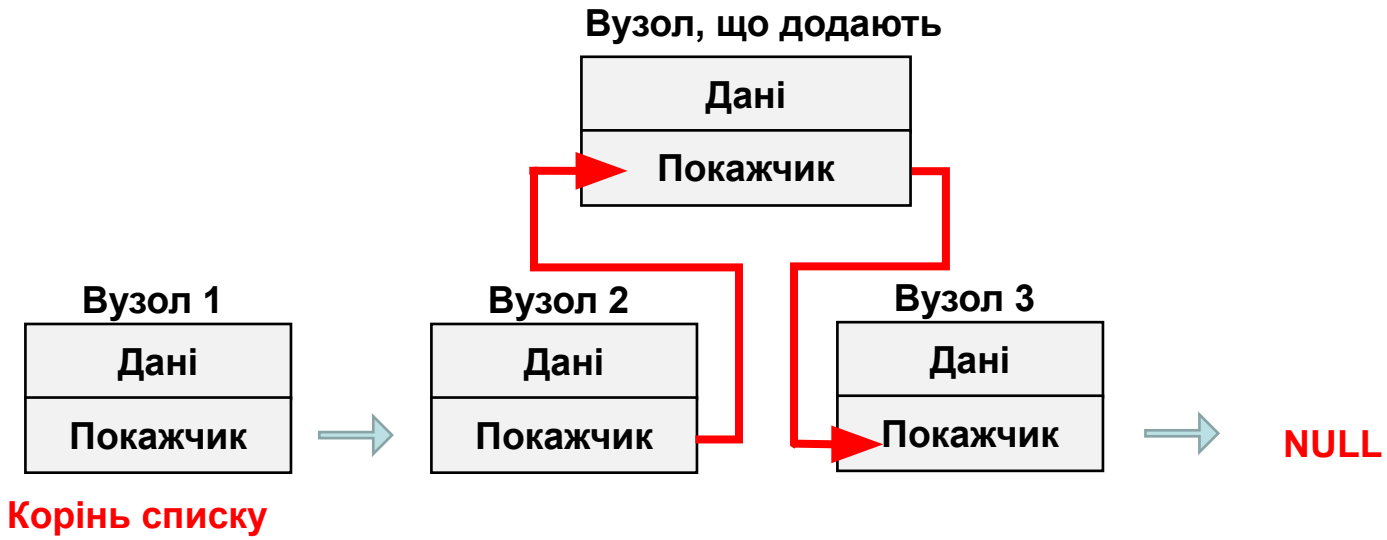

```
struct list * init (int a) // a- значення першого
вузла
{
   struct list * lst;
   // Виділення пам'яті під корінь списку
   lst = (struct list *) malloc (sizeof (struct
list));
   lst-> field = a;
   lst-> ptr = NULL; // Це останній вузол списку
   return (lst);
}
```

Додавання вузла в ОЛС

Функція додавання вузла в список приймає два аргументи:

- Покажчик на вузол, після якого відбувається додавання
- Дані у доданому вузла.

Процедуру додавання вузла можна відобразити наступною схемою:



Додавання вузла в ОЛС включає в себе наступні етапи:

1. створення додається вузла і заповнення його поля даних;
2. перевстановлення покажчика вузла, що передує додає, на що додається вузол;
3. установка покажчика додається вузла на наступний вузол (той, на який вказував попередній вузол).

Таким чином, функція додавання вузла в ОЛС має вигляд:

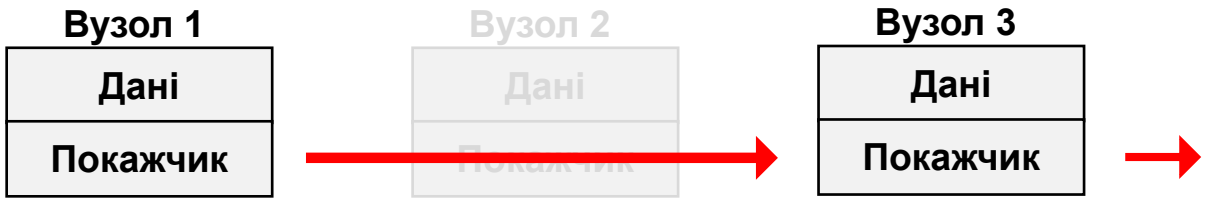
```
struct list * addelem (list * lst, int number)
{
   struct list * temp, * p;
   temp = (struct list *) malloc (sizeof (list));
   p = lst-> ptr; // Збереження покажчика на
наступний вузол
   lst-> ptr = temp; // Попередній вузол вказує на
створюваний
   temp-> field = number; // Збереження поля даних
додається вузла
   temp-> ptr = p; // Створений вузол вказує на
наступний елемент
   return (temp);
}
```

Видалення вузла ОЛС

Як аргументи функції видалення елемента ОЛС передаються покажчик на видаляється вузол, а також покажчик на корінь списку.

Функція повертає покажчик на вузол, наступний за видаляється.

Видалення вузла може бути представлено наступною схемою:



Видалення вузла ОЛС включає в себе наступні етапи:

- установка покажчика попереднього вузла на вузол, наступний за видаляється;
- звільнення пам'яті видаляється вузла.

```
struct list * deletelem (list * lst, list * root)
{
   struct list * temp;
   temp = root;
   while (temp-> ptr! = lst) // переглядаємо список
починаючи з кореня
   {// Поки не знайдемо вузол, що передує lst
     temp = temp-> ptr;
   }
   temp-> ptr = lst-> ptr; // Переставляємо покажчик
   free (lst); // Звільняємо пам'ять видаляється
вузла
   return (temp);
}
```

Видалення кореня списку

Функція видалення кореня списку в якості аргументу отримує покажчик на поточний корінь списку. Значенням буде новий корінь списку - той вузол, на який вказує видаляється корінь.

```
struct list * deletehead (list * root)
{
   struct list * temp;
   temp = root-> ptr;
   free (root); // Звільнення пам'яті поточного
кореня
   return (temp); // Новий корінь списку
}
```

Виведення елементів списку

Як аргумент на функцію виведення елементів передається покажчик на корінь списку.

Функція здійснює послідовний обхід усіх вузлів з виведенням їх значень

```
void listprint (list * lst)
{
   struct list * p;
   p = lst;
   do {
     printf ("% d", p-> field); // Вивід значення
елемента p
     p = p-> ptr; // Перехід до наступного вузла
   } While (p! = NULL);
}
```

3. Двозв'язні лінійні списки

Кожен вузол двонаправленого (двозв’язного) лінійного списку (ДЛС) містить два поля покажчиків - на наступний і на попередній вузли. Покажчик на попередній вузол кореня списку містить нульове значення. Покажчик на наступний вузол останнього вузла також містить нульове значення.

```
struct list
{
    int field; // Поле даних
    struct list * next; // Покажчик на наступний
елемент
    struct list * prev; // Покажчик на попередній
елемент
};
```

Ініціалізація ДЛС

Ініціалізація списку призначена для створення кореневого вузла списку, у якого поля покажчиків на наступний і попередній вузли містять нульове значення.


```
struct list * init (int a) // a- значення першого
вузла
{
   struct list * lst;
   // Виділення пам'яті під корінь списку
   lst = (struct list *) malloc (sizeof (struct
list));
   lst-> field = a;
   lst-> next = NULL; // Покажчик на наступний вузол
   lst-> prev = NULL; // Покажчик на попередній вузол
   return (lst);
}
```

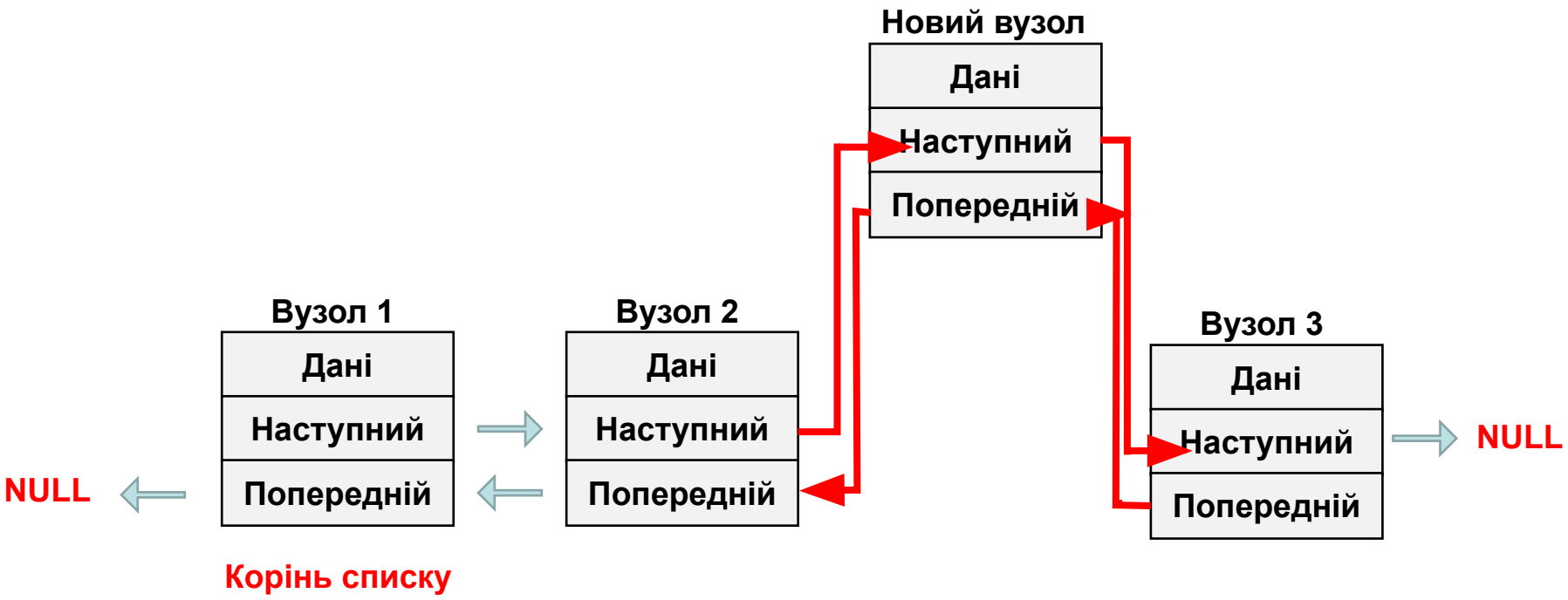
Додавання вузла в ДЛС

Функція додавання вузла в список приймає два аргументи:

Покажчик на вузол, після якого відбувається додавання

Дані у доданому вузла.

Процедуру додавання вузла можна відобразити наступною схемою:



Додавання вузла в ДЛС включає в себе наступні етапи:

1. створення вузла додається елемента і заповнення його поля даних;
2. перевстановлення покажчика "наступний" вузла, що передує додає, на що додається вузол;
3. перевстановлення покажчика "попередній" вузла, наступного за додається, на що додається вузол;
4. установка покажчика "наступний" додається вузла на наступний вузол (той, на який вказував попередній вузол);
5. установка покажчика "попередній" додається вузла на вузол, що передує додає (вузол, переданий в функцію).

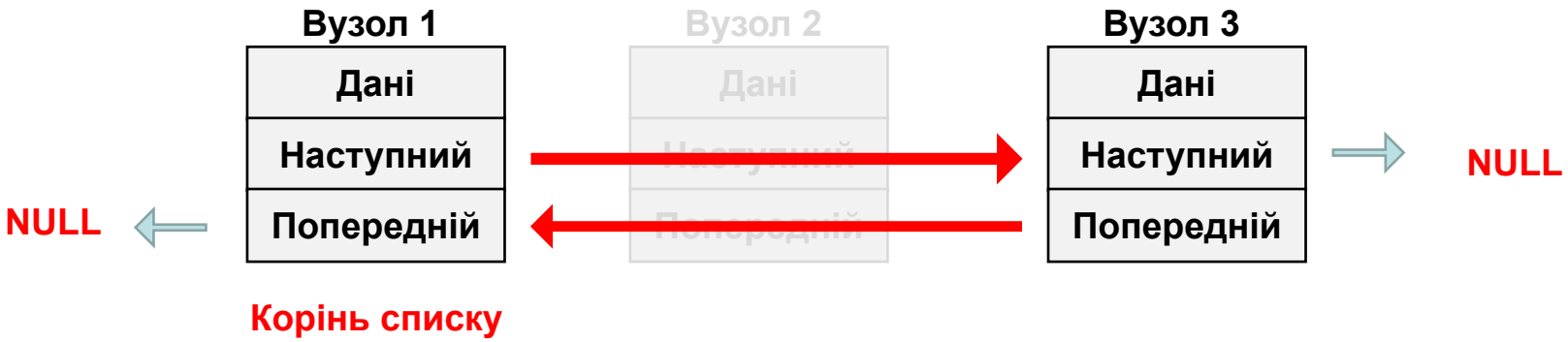
```
struct list * addelem (list * lst, int number)
{
   struct list * temp, * p;
   temp = (struct list *) malloc (sizeof (list));
   p = lst-> next; // Збереження покажчика на
наступний вузол
   lst-> next = temp; // Попередній вузол вказує на
створюваний
   temp-> field = number; // Збереження поля даних
додається вузла
   temp-> next = p; // Створений вузол вказує на
наступний вузол
   temp-> prev = lst; // Створений вузол вказує на
попередній вузол
   if (p! = NULL)
     p-> prev = temp;
   return (temp);
}
```

Видалення вузла ДЛС

Як аргументи функції видалення вузла ДЛС передається покажчик на видаляється вузол. Оскільки вузол списку має поле покажчика на попередній вузол, немає необхідності передавати покажчик на корінь списку.

Функція повертає покажчик на вузол, наступний за видаляється.

Видалення вузла може бути представлено наступною схемою:



Видалення вузла ДЛС включає в себе наступні етапи:

1. установка покажчика "наступний" попереднього вузла на вузол, наступний за видаляється;
2. установка покажчика "попередній" наступного вузла на вузол, що передує удаляемому;
3. звільнення пам'яті видаляється вузла.

```
struct list * deletelem (list * lst)
{
   struct list * prev, * next;
   prev = lst-> prev; // Вузол, що передує lst
   next = lst-> next; // Вузол, наступний за lst
   if (prev! = NULL)
     prev-> next = lst-> next; // Переставляємо
покажчик
   if (next! = NULL)
     next-> prev = lst-> prev; // Переставляємо
покажчик
   free (lst); // Звільняємо пам'ять видаляється
елемента
   return (prev);
}
```

Видалення кореня ДЛС

Функція видалення кореня ДЛС як аргумент отримує покажчик на поточний корінь списку. Значенням буде новий корінь списку - той вузол, на який вказує видаляється корінь.

```
struct list * deletehead (list * root)
{
   struct list * temp;
   temp = root-> next;
   temp-> prev = NULL;
   free (root); // Звільнення пам'яті поточного
кореня
   return (temp); // Новий корінь списку
}
```

Виведення елементів ДЛС

Функція виведення елементів ДЛС аналогічна функції для ОЛС.

Як аргумент на функцію виведення елементів передається покажчик на корінь списку.

Функція здійснює послідовний обхід усіх вузлів з виведенням їх значень.

```
void listprint (list * lst)
{
   struct list * p;
   p = lst;
   do {
     printf ("% d", p-> field); // Вивід значення
елемента p
     p = p-> next; // Перехід до наступного вузла
   } While (p! = NULL); // Умова закінчення обходу
}
```