ЛАБОРАТОРНА РОБОТА №14

Тема: **Зв’язані списки**
Мета: **Ознайомитись з основами роботи зі зв’язаними списками**
Час: 2 год.

Виконання роботи

- Надати викладачу, виконане завдання для самопідготовки в п. 4.2.
- Вивчити теоретичні відомості.
- Відповісти тестові завдання.
- Виконати самостійну роботу.

Завдання для самопідготовки

В процесі підготовки до заняття студент у обов'язковому порядку повинен виконати наступні завдання:

а) За допомогою конспекту лекцій і рекомендованої літератури розглянути суть таких питань:
1) Поняття та способи створення зв’язаних списків
2) Обробка зв’язаних списків
3) Видалення та додавання елементів зв’язаного списку

б) Занести в звіт такі дані:
1) номер практичної роботи;
2) тему і мету роботи;
3) короткий конспект основних теоретичних відомостей.

Теоретичні відомості

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 (як даних може виступати змінна, об'єкт класу або структури і т. Д.)
• покажчик на наступний вузол в списку.

Елементи пов'язаного списку можна поміщати і виключати довільним чином.

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 (двусвязний) списки.

Зв'язний список, що містить тільки один покажчик на наступний елемент, називається одинзв'язного.

Зв'язний список, що містить два поля покажчика - на наступний елемент і на попередній, називається двусвязного.

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

Таким чином, розрізняють 4 основних види списків.

• одинзв'язного лінійний список (ОЛС).

Кожен вузол ОЛС містить 1 поле покажчика на наступний вузол. Поле покажчика останнього вузла містить нульове значення (вказує на NULL).

• одинзв'язного циклічний список (ОЦС).

Кожен вузол ОЦС містить 1 поле покажчика на наступний вузол. Поле покажчика останнього вузла містить адресу першого вузла (кореня списку).

• двузв’язного 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).

• двузв’язного 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..

Однозв’язні лінійні списки

Кожен вузол односпрямованого (одинзв'язного) лінійного списку (ОЛС) містить одне поле покажчика на наступний вузол. Поле покажчика останнього вузла містить нульове значення (вказує на NULL).

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 для додається вузла.

Додавання вузла в ОЛС включає в себе наступні етапи:

• створення додається вузла і заповнення його поля даних;

• перевстановлення покажчика вузла, що передує додає, на що додається вузол;

• 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);
}
```

Значенням функції є адреса доданого вузла.

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 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 на екран 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);
}
```

Двохзв'язані списки

Кожен вузол двонаправленого (двохзв'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.
Вузол ДЛС можна представити у вигляді структури:

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

Основні дії, вироблені над вузлами ДЛС:
• Ініціалізація списку
• Додавання вузла в список
• Видалення вузла зі списку
• Видалення кореня списку
• Висновок елементів списку
• Висновок елементів списку в зворотному порядку
• Взаємообмін двох вузлів списку

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 для додається вузла.

Додавання вузла в ДЛС включає в себе наступні етапи:

• створення вузла додається елемента і заповнення його поля даних;

• перевстановлення покажчика "наступний" вузла, що передує додає, на що додається вузол;

• перевстановлення покажчика "попередній" вузла, наступного за додається, на що додається вузол;

• установка покажчика "наступний" додається вузла на наступний вузол (той, на який вказував попередній вузол);

• установка покажчика "попередній" додається вузла на вузол, що передує додає (вузол, переданий в функцію).

Таким чином, функція додавання вузла в ДЛС має вигляд:

```
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);
}
```

Значенням функції є адреса доданого вузла.

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 ДЛС включає в себе наступні етапи:
• установка покажчика "наступний" попереднього вузла на вузол, наступний за видаляється;
• установка покажчика "попередній" наступного вузла на вузол, що передує удаляемому;
• 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
}
```

Для ДЛС також можна викликати функцію виведення елементів списку в зворотному порядку.

Виведення елементів ДЛС в зворотному порядку

Функція виведення елементів ДЛС в зворотному порядку приймає в якості аргументу покажчик на корінь списку. Функція переміщує покажчик на останній вузол списку і здійснює послідовний обхід усіх вузлів з виведенням їх значень в зворотному порядку.

```
void listprintr (list * lst)
{
  struct list * p;
  p = lst;
  while (p-> next! = NULL)
    p = p-> next; // Перехід до кінця списку
```

Самостійна робота

1. Створити зв’язаний список чисел розміром, сказаним користувачем. Заповнити випадковими числами. Повернути на екран. Створити функцію додавання та видалення вузлу, вказаного користувачем.
2. Створити двонаправлений зв’язаний список довільною довжини, заповнений користувачем. Скопіювати у зворотному напрямку список з вузла, вказаного користувачем.
3. Створити зв’язаний список, що буде імітувати роботу складу. Передбачити наступні поля: найменування, кількість, ціна, сума, артикул, місце розташування. Акцентувати увагу на необхідності перерахування суми у разі зміни кількості або ціни. Передбачити можливість введення, зміну та видалення товарів. Виведення відомості, про залишки на складі.
4. Завдання носить ім'я Йосипа Флавія - відомого історика I століття. Існує легенда, що Йосип вижив і став відомим завдяки своєму математичному дару. В ході іудейської війни він, будучи в складі загону з 41 іудейського війна була загнана римлянами в печеру. Загін, в якому перебував Йосип, володів неймовірно сильним бойовим духом і не бажаючи здаватися в полон, війни вважали за краще самогубство. Вони встали в коло і послідовно почали вбивати кожного третього з живих до тих пір, поки не залишиться жодної людини. А Йосип не поділяв сподівань товаришів по службі. Щоб не бути вбитим, Йосип обчислив "рятівні місця" на яке поставив себе і свого товариша.

   Створити зв’язаний список з *n*-елементі (де *n*-випадкове число). Видаляти по кругу кожен третій елемент, доки не залишиться 1. Кожен крок повертати на екран.

Вимоги до оформлення звіту

Звіт повинен містити:

- Короткий конспект теоретичних відомостей;
- Результати виконаних дій.