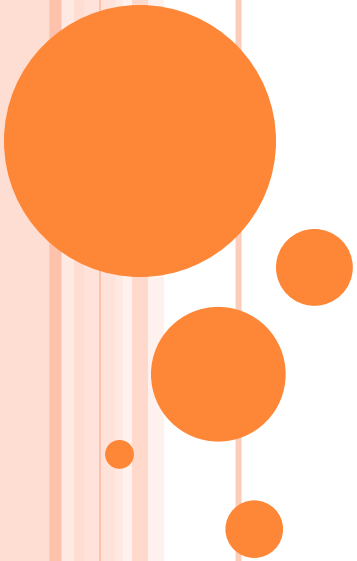


Лекція 22

Списки

Списки

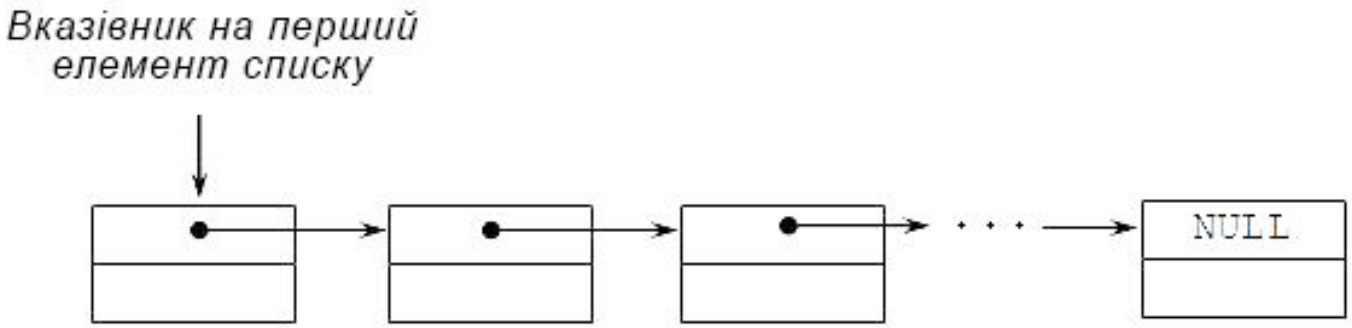
- Списком називається **впорядкована множина, що складається з змінного числа елементів, до яких застосовні операції включення, виключення**.
- Список, що відображає відносини сусідства між елементами, називається лінійним.
- Довжина списку дорівнює числу елементів, що містяться в списку, список нульової довжини називається порожнім списком.
- Списки є спосіб організації структури даних, при якій елементи деякого типу утворюють ланцюжок. Для зв'язування елементів в списку використовують **систему вказівників**. У мінімальному випадку, будь-який елемент лінійного списку має один покажчик, який вказує на наступний елемент у списку або є порожнім покажчиком, що інтерпретується як кінець списку.

Списки

- Структура, елементами якої служать записи з одним і тим же форматом, пов'язані один з одним за допомогою вказівників, що зберігаються в самих елементах, називають **зв'язаним списком.**
- У пов'язаному списку елементи **лінійно впорядковані**, але порядок визначається не номерами, як у масиві, а вказівниками, що входять до складу елементів списку.
- Кожен список має *особливий елемент, званий вказівником початку списку (головою списку)*, який зазвичай за змістом відмінний від інших елементів.
- В поле вказівника останнього елемента списку знаходиться спеціальний ознака NULL, який свідчить про кінець списку.
- **Лінійні зв'язні списки** є найпростішими динамічними структурами даних.
- З усього різноманіття пов'язаних списків можна виділити наступні основні:
  - односпрямовані (одинзв'язні) списки;
  - двонаправлені (двусвязного) списки;
  - циклічні (кільцеві) списки.

Односпрямований (однозв'язний) список

- Односпрямований (однозв'язний) список - це структура даних, що представляє собою послідовність елементів, в кожному з яких зберігається значення і вказівник на наступний елемент списку. В останньому елементі вказівник на наступний елемент дорівнює NULL.



Односпрямований (однозв'язний) список

- Опис найпростішого елемента такого списку виглядає наступним чином:
-
- struct імя_тіпа
- {
  - інформаційне поле;
  - адресне поле;
- };
-
- де
- **інформаційне поле** - це поле будь-якого, раніше оголошеного або стандартного, типу;
- **адресне поле** - це вказівник на об'єкт того ж типу, що і визначається структура, в нього записується адреса наступного елемента списку.

Односпрямований (однозв'язний) список

```
struct Node
{
        int key; // інформаційне поле
        Node * next; // адресне поле
};
```

Інформаційних полів може бути кілька.

```
struct point
 {
        char * name; // інформаційне поле
        int age; // інформаційне поле
        point * next; // адресне поле
 };
```

Односпрямований (однозв'язний) список

- Основними операціями, здійснюваними з односпрямованим списками, є:
  - створення списку;
  - друк (перегляд) списку;
  - вставка елемента в список;
  - видалення елемента зі списку;
  - пошук елемента в списку
  - перевірка порожнечі списку;
  - видалення списку.

Односпрямований (однозв'язний) список

- Для опису алгоритмів цих основних операцій використовується наступне оголошення:
-
- struct **Single_List**
- { // структура даних
- int Data; // інформаційне поле
- Single_List * Next; // адресне поле
- };
- . . . . . .
- Single_List * **Head**; // вказівник на перший елемент списку
- . . . . . .
- Single_List * **Current**;
- // вказівник на поточний елемент списку (при необхідності)

Створення однозв'язного списку

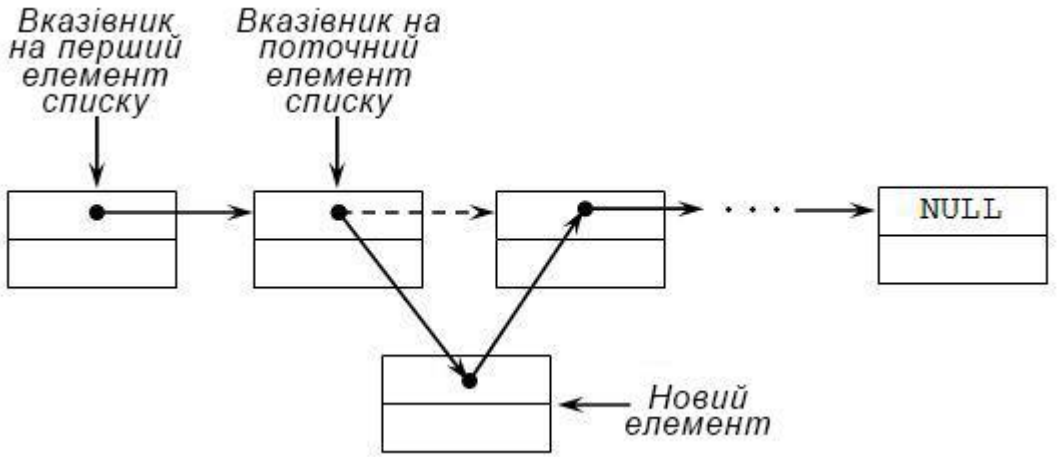
```
// створення односпрямованого списку (додавання в кінець)
void Make_Single_List (int n, Single_List * Head)
{
  if (n> 0) {
    (* Head) = new Single_List ();
    // виділяємо пам'ять під новий елемент
    cout << "Введіть значення";
    cin >> (* Head) -> Data;
    // вводимо значення інформаційного поля
    (* Head) -> Next = NULL; // обнулення адресного поля
     Make_Single_List (n-1, & ((* Head) -> Next));
  }
}
```

Друк (перегляд) однозв'язного списку

```
void Print_Single_List (Single_List * Head)
{
  if (Head! = NULL) {
    cout << Head-> Data << "\ t";
    Print_Single_List (Head-> Next);
    // перехід до наступного елементу
  }
  else cout << "\ n";
}
```

Вставка елемента в однозв'язний список

- В динамічні структури легко додавати елементи, так як для цього достатньо змінити значення адресних полів. Вставка першого і наступних елементів списку відрізняються один від одного. Тому у функції, що реалізує дану операцію, спочатку здійснюється перевірка, на яке місце вставляється елемент. Далі реалізується відповідний алгоритм додавання

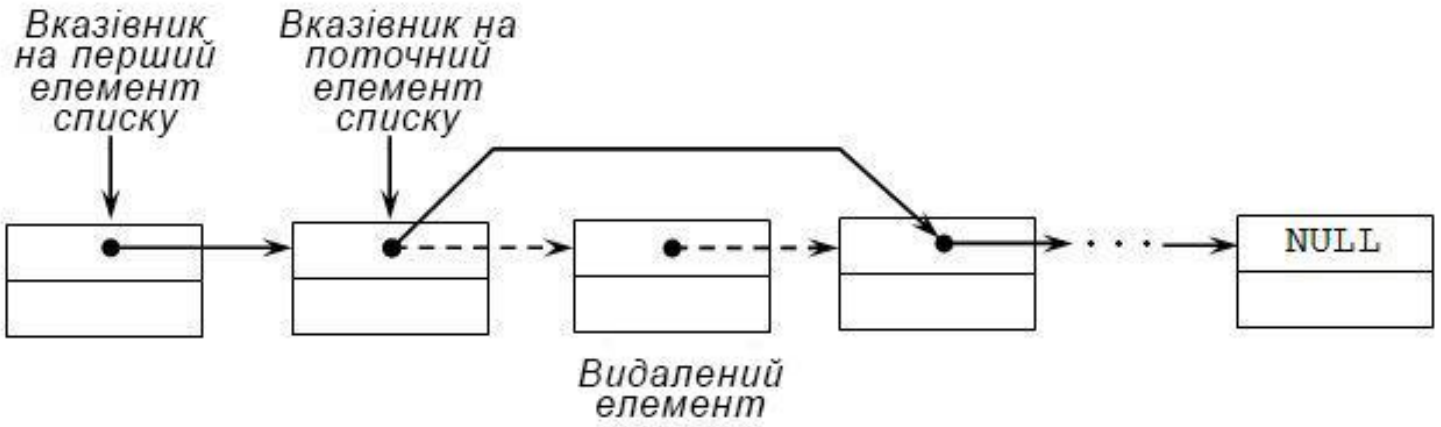


Вставка елемента в однозв'язний список

```
Single_List * Insert_Item_Single_List (Single_List * Head, int Number, int DataItem) {
  Number--;
  Single_List * NewItem = new (Single_List);
  NewItem-> Data = DataItem;
  NewItem-> Next = NULL;
  if (Head == NULL) {// список порожній
    Head = NewItem; // створюємо перший елемент списку
  }
  else {// списку не порожній
    Single_List * Current = Head;
    for (int i = 1; i <Number && Current-> Next! = NULL; i ++)
    Current = Current-> Next;
    if (Number == 0) {
      // вставляємо новий елемент на перше місце
      NewItem-> Next = Head;
      Head = NewItem;
    }
    else {// вставляємо новий елемент на непервих місце
      if (Current-> Next! = NULL)
      NewItem-> Next = Current-> Next;
      Current-> Next = NewItem;
    }
  }
  return Head;
}
```

Видалення елемента з однозв'язного списку

- З динамічних структур можна видаляти елементи, так як для цього достатньо змінити значення адресних полів.
- Алгоритми видалення першого і наступних елементів списку відрізняються один від одного. Тому у функції, що реалізує дану операцію, здійснюється перевірка, який об'єкт був видалений.



Видалення елемента з однозв'язного списку

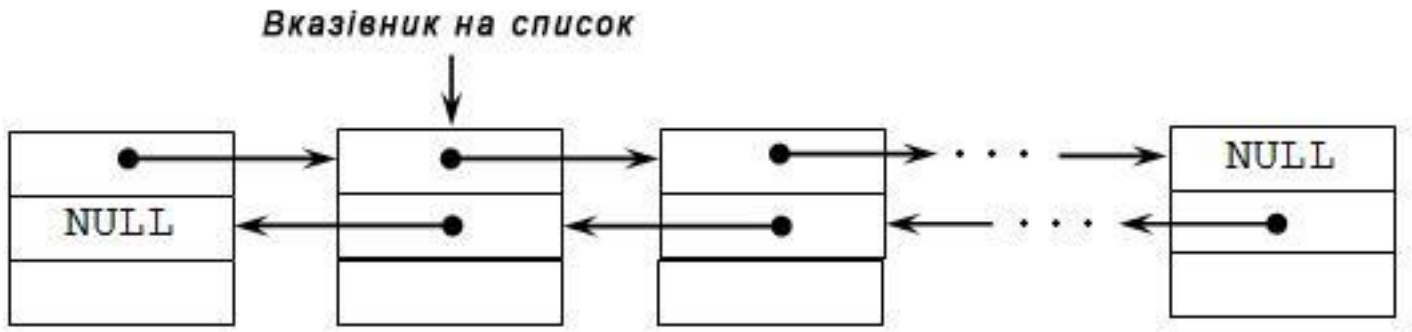
```
Single_List * Delete_Item_Single_List (Single_List * Head, int Number) {
  Single_List * ptr;    Single_List * Current = Head;
  for (int i = 1; i <Number && Current! = NULL; i ++)
    Current = Current-> Next;
  if (Current! = NULL) {// перевірка на коректність
    if (Current == Head) {// видаляємо перший елемент
      Head = Head-> Next;
      delete (Current);
      Current = Head;
    }
    else {// видаляємо неперший елемент
      ptr = Head;
      while (ptr-> Next! = Current)
        ptr = ptr-> Next;
      ptr-> Next = Current-> Next;
      delete (Current);
      Current = ptr;    }
  }
  return Head;}
```

Пошук елемента в однозв'язному списку

```
bool Find_Item_Single_List (Single_List * Head, int DataItem) {
  Single_List * ptr; // допоміжним вказівник
  ptr = Head;
  while (ptr! = NULL) {// поки не кінець списку
    if (DataItem == ptr-> Data) return true;
    else ptr = ptr-> Next;
  }
  return false;
}
```

Двоспрямований (двузв'язний) список

Двунаправлений список (двузв'язний) список - це структура даних, що складається з послідовності елементів, кожен з яких містить інформаційну частину і два вказівника на сусідні елементи. При цьому два сусідні елементи повинні містити взаємні посилання один на одного.



Двоспрямований (двузв'язний) список

- Опис найпростішого елемента такого списку виглядає наступним чином:
- struct імя_тіпа {
- інформаційне поле;
- адресне поле 1;
- адресне поле 2;
- };
- де інформаційне поле - це поле будь-якого, раніше оголошеного або стандартного, типу;
- адресне поле 1 - це вказівник на об'єкт того ж типу, що і визначається структура, в нього записується адреса наступного елемента списку;
- адресне поле 2 - це вказівник на об'єкт того ж типу, що і визначається структура, в нього записується адреса попереднього елемента списку.

Двоспрямований (двузв'язний) список

- struct list {
- type elem;
- list * next, * pred;
- }
- list * headlist;
-
- де type - тип інформаційного поля елемента списку;
- * Next, * pred - вказівники на наступний і попередній елементи цієї структури відповідно.

Двоспрямований (двузв'язний) список

- Основні операції, здійснювані з двонаправленими списками, такі як:
  - створення списку;
  - друк (перегляд) списку;
  - вставка елемента в список;
  - видалення елемента зі списку;
  - пошук елемента в списку;
  - перевірка порожнечі списку;
  - видалення списку.

Двоспрямований (двузв'язний) список

- Для опису алгоритмів цих основних операцій використовується наступне оголошення:
- struct Double_List {// структура даних
- int Data; // інформаційне поле
- Double_List * Next, // адресне поле
- * Prior; // адресне поле
- };
- . . . . . .
- Double_List * Head; // вказівник на перший елемент списку
- . . . . . .
- Double_List * Current;
- // вказівник на поточний елемент списку (при необхідності)

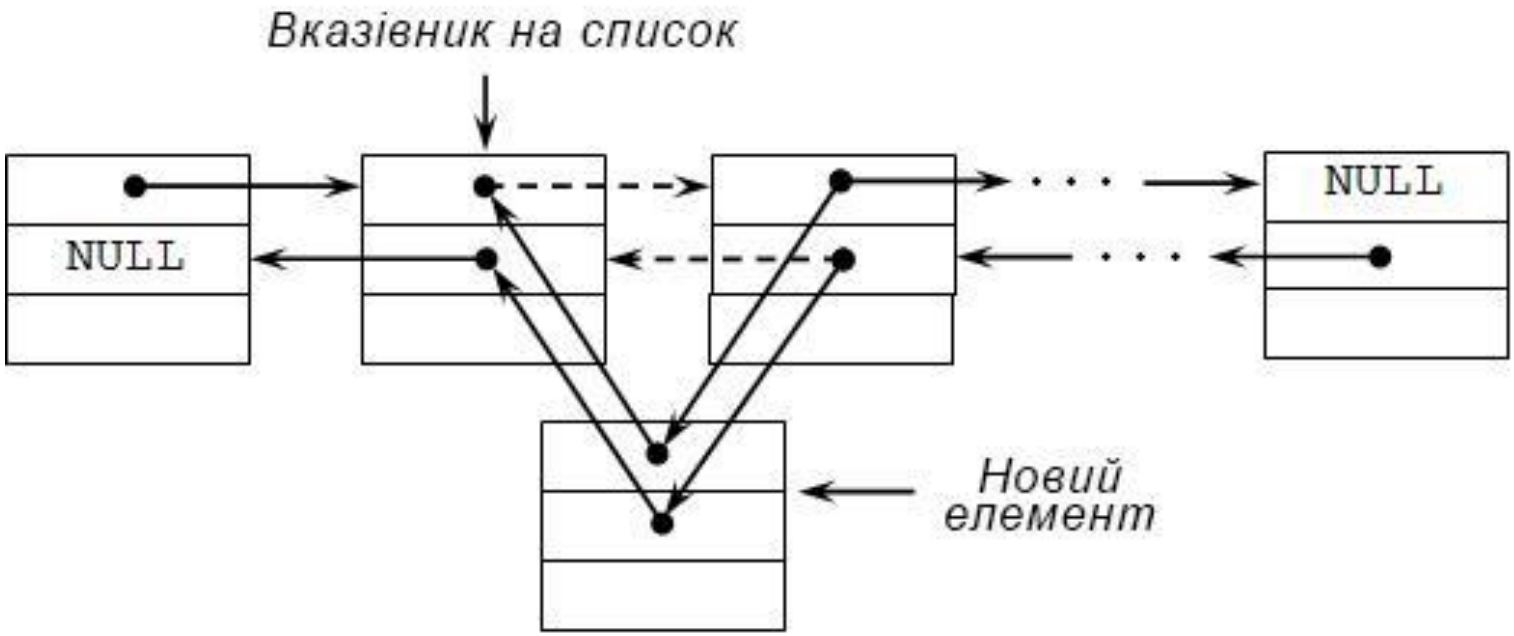
Створення двузв'язного списку

```
void Make_Double_List (int n, Double_List ** Head,
        Double_List * Prior) {
  if (n> 0) {
    (* Head) = new Double_List (),
     // виділяємо пам'ять під новий елемент
    cout << "Введіть значення";
    cin >> (* Head) -> Data;
    // вводимо значення інформаційного поля
    (* Head) -> Prior = Prior;
    (* Head) -> Next = NULL; // обнулення адресного поля
    Make_Double_List (n-1, & ((* Head) -> Next), (* Head));
  }
  else (* Head) = NULL;
}
```

Друк (перегляд) двузв'язного списку

```
void Print_Double_List (Double_List * Head) {
   if (Head! = NULL) {
     cout << Head-> Data << "\ t";
     Print_Double_List (Head-> Next);
     // перехід до наступного елементу
   }
   else cout << "\ n";
}
```

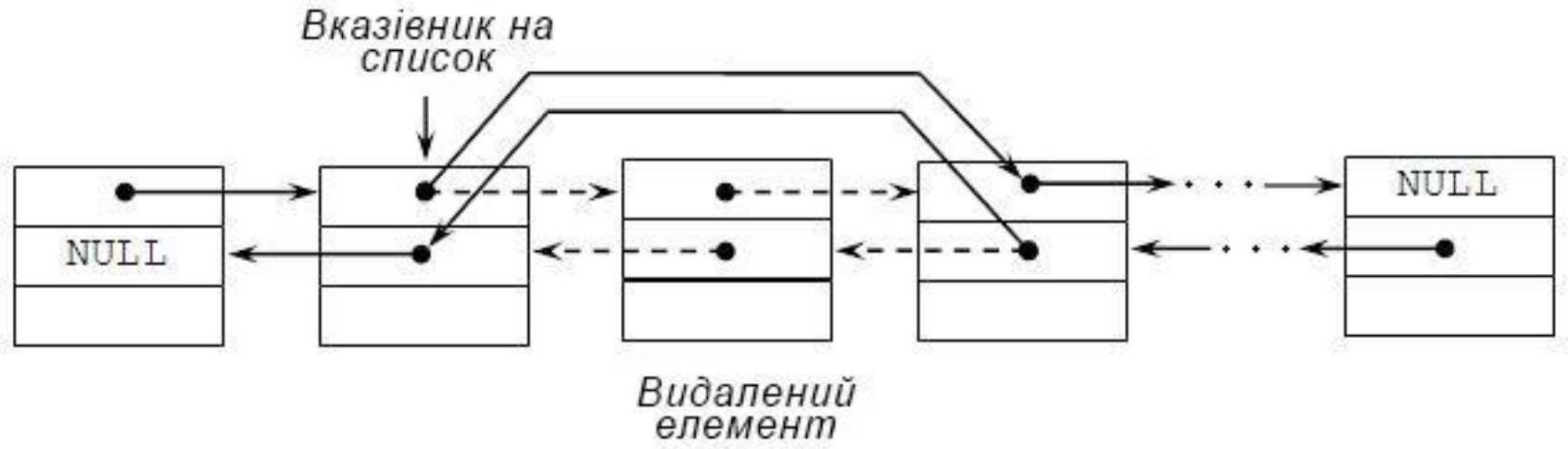
Вставка елемента в двузв'язний список



Вставка елемента в двузв'язний список

```
Double_List *
Insert_Item_Double_List
(Double_List * Head,
      int Number, int DataItem) {
  Number--;
  Double_List * NewItem = new
(Double_List);
  NewItem-> Data = DataItem;
  NewItem-> Prior = NULL;
  NewItem-> Next = NULL;
  if (Head == NULL) {// список
порожній
    Head = NewItem;
  }
  else {// списку не порожній
    Double_List * Current = Head;
    for (int i = 1; i <Number &&
Current-> Next! = NULL; i ++)
    Current = Current-> Next;
  if (Number == 0) {
    // вставляємо новий елемент на перше
місце
      NewItem-> Next = Head;
      Head-> Prior = NewItem;
      Head = NewItem;
    }
    else {// вставляємо новий елемент на
непервих місце
      if (Current-> Next! = NULL)
Current-> Next-> Prior = NewItem;
      NewItem-> Next = Current-> Next;
      Current-> Next = NewItem;
      NewItem-> Prior = Current;
      Current = NewItem;
    }
  }
  return Head;
}
```

Видалення елемента з двузв'язного списку



Видалення елемента з двузв'язного списку

```
Double_List * Delete_Item_Double_List (Double_List * Head,
      int Number) {
  Double_List * ptr; // допоміжний вказівник
  Double_List * Current = Head;
  for (int i = 1; i <Number && Current! = NULL; i ++)
    Current = Current-> Next;
  if (Current! = NULL) {// перевірка на коректність
    if (Current-> Prior == NULL) {// видаляємо перший елемент
      Head = Head-> Next;
      delete (Current);
      Head-> Prior = NULL;
      Current = Head;
    }
    else {// видаляємо непервих елемент
      if (Current-> Next == NULL) {
      // видаляємо останній елемент
        Current-> Prior-> Next = NULL;
        delete (Current);
        Current = Head;
      }
      else {// видаляємо непервих і не останню елемент
        ptr = Current-> Next;
        Current-> Prior-> Next = Current-> Next;
        Current-> Next-> Prior = Current-> Prior;
        delete (Current);
        Current = ptr;
      }
    }
  }
  return Head;
}
```

Пошук елемента в двузв'язному списку

- Операція пошуку елемента в двунаправленном списку реалізується абсолютно аналогічно відповідної функції для односпрямованого списку. Пошук елемента в двунаправленном списку можна вести:
- а) переглядаючи елементи від початку до кінця списку;
- б) переглядаючи елементи від кінця списку до початку;
- в) переглядаючи список в обох напрямках одночасно: від початку до середини списку і від кінця до середини (враховуючи, що елементів в списку може бути парне або непарна кількість).

Пошук елемента в двузв'язному списку

```
bool Find_Item_Double_List (Double_List * Head, int DataItem) {
  Double_List * ptr; // допоміжний вказівник
  ptr = Head;
  while (ptr! = NULL) {// поки не кінець списку
    if (DataItem == ptr-> Data) return true;
    else ptr = ptr-> Next;
    }
  return false;
}
```

Перевірка порожності двузв'язного списку

```
// перевірка порожнечі двоспрямованістю списку
bool Empty_Double_List (Double_List * Head)
{
   if (Head! = NULL) return false;
   else return true;
}
```

Видалення двузв'язного списку

```
void Delete_Double_List (Double_List * Head) {
    if (Head! = NULL) {
      Delete_Double_List (Head-> Next);
      delete Head;
    }
}
```

Приклад 1

- N-натуральний чисел є елементами двонаправленого списку L, обчислити: X1 * Xn + X2 * Xn-1 + … + Xn * X1.
- Вивести на екран кожне множення і підсумкову суму.
- **Алгоритм:**

  1. Створюємо структуру.
  2. Формуємо список цілих чисел.
  3. Просуваємося за списком: від початку до кінця і від кінця до початку в одному циклі, перемножуємо дані, що містяться у відповідних елементах списку.
  4. Підсумовуємо отримані результати.
  5. Виводимо на друк

Приклад 1

```
// пошук останнього елемента списку
Double_List * Find_End_Item_Double_List
(Double_List * Head)
{
Double_List * ptr; // додатковий вказівник
ptr = Head;
while (ptr-> Next! = NULL)
{
 ptr = ptr-> Next;
}
return ptr;
}
```

Приклад 1

```
// підсумкова сума множень
void Total_Sum (Double_List * Head)
{
Double_List * lel = Head;
Double_List * mel = Find_End_Item_Double_List (Head);
 int mltp, sum = 0;
while (lel! = NULL)
 {
mltp = (lel-> Data) * (mel-> Data); // множення елементів
printf ( "\ n \ n% d *% d =% d", lel-> Data, mel-> Data, mltp);
sum = sum + mltp; // підсумовування творів
lel = lel-> Next;    // йдемо за списком з першого елемента в останній
mel = mel-> Prior;    // йдемо за списком з останнього елемента в перший
}
printf ( "\ n \ n Підсумкова сума дорівнює% d", sum);
}
```

Дякую за увагу!