Алгоритмы и структуры данных

ТЕМА #1: ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

КЛАСС: ДВУНАПРАВЛЕННЫЙ СПИСОК

Память под данные выделяется:

- на этапе компиляции (необходимый объем должен быть известен до начала выполнения программы, т.е. задан константой, выделяется одним блоком)
- во время выполнения программы (операторы new, malloc, Но объем должен быть известен до выделения памяти, выделяется одним блоком).
- во время выполнения программы по мере необходимости отдельными блоками, связанными друг с другом с помощью указателей.

Область памяти выделяемая таким образом, называется динамической структурой данных.

Список –

структура данных, в которой объекты расположены в линейном порядке.

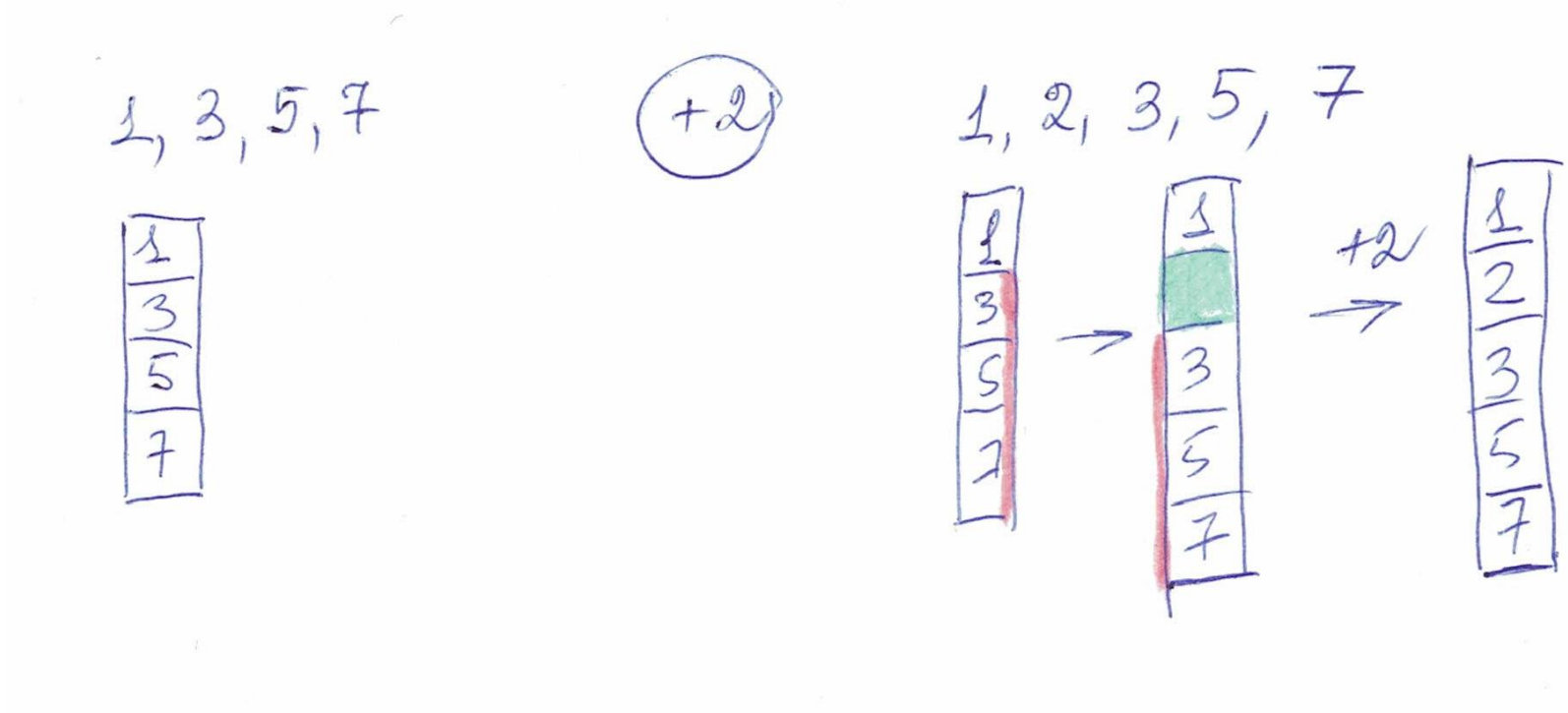
В массиве порядок определяется индексами.

В связном списке порядок определяется указателями на каждый объект.

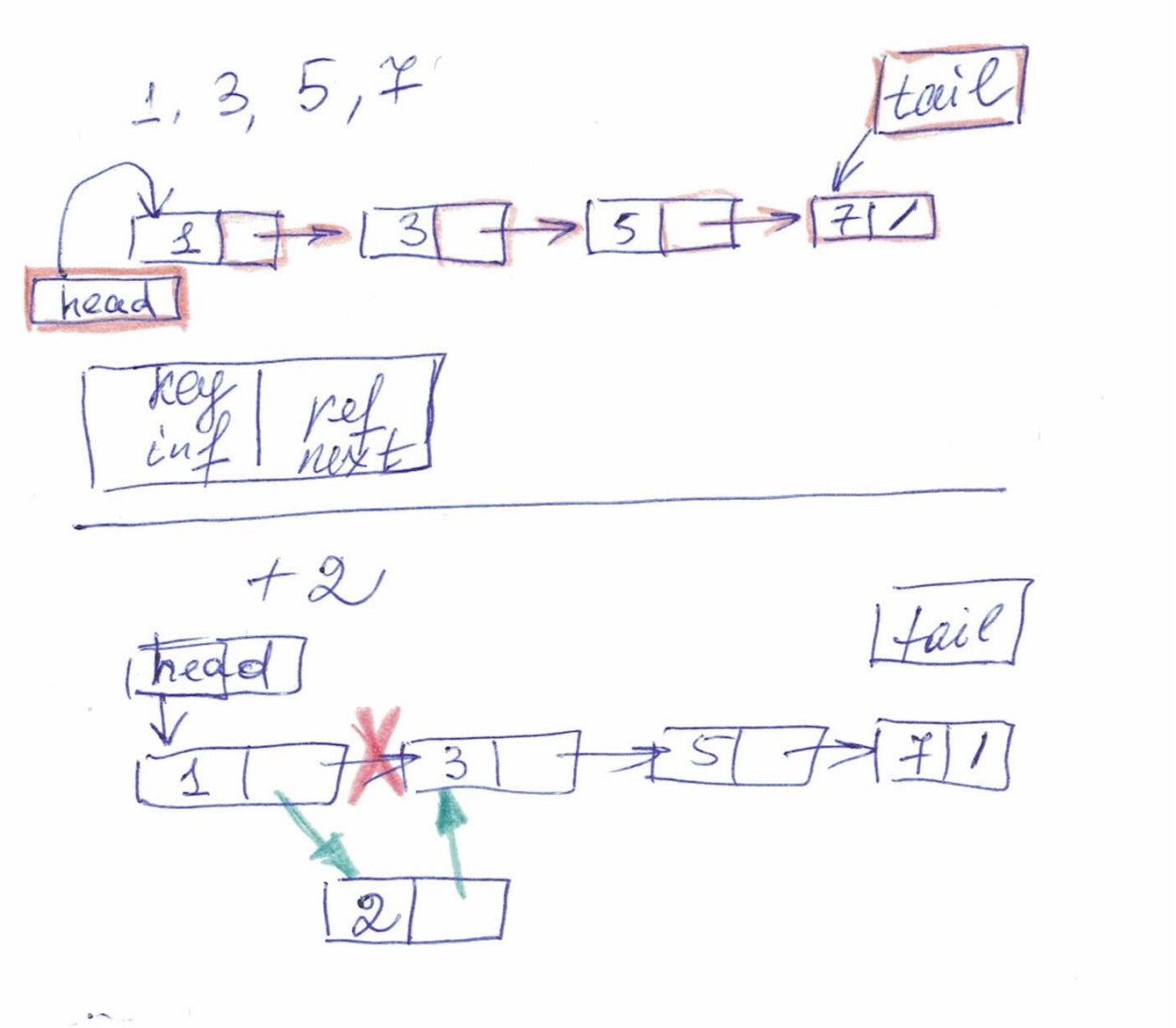
Связные списки используются

для представления динамических множеств и поддерживают для них все операции.

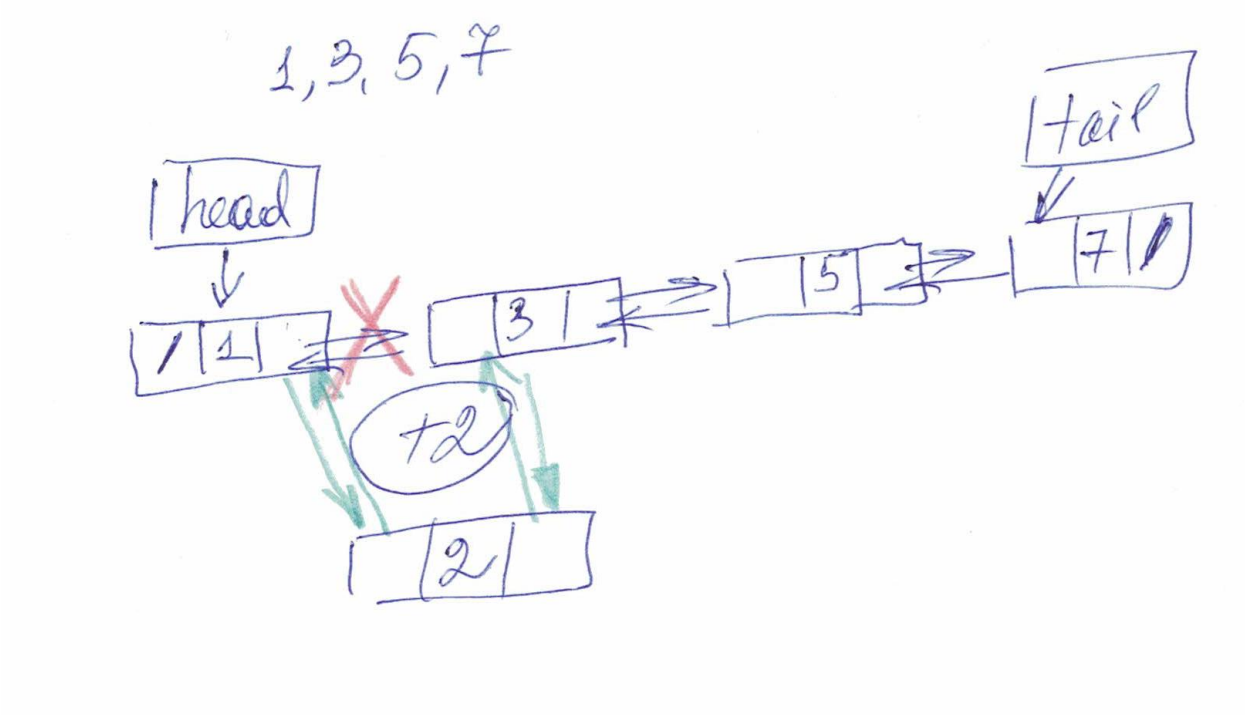
Список – массив.



Односвязный список.

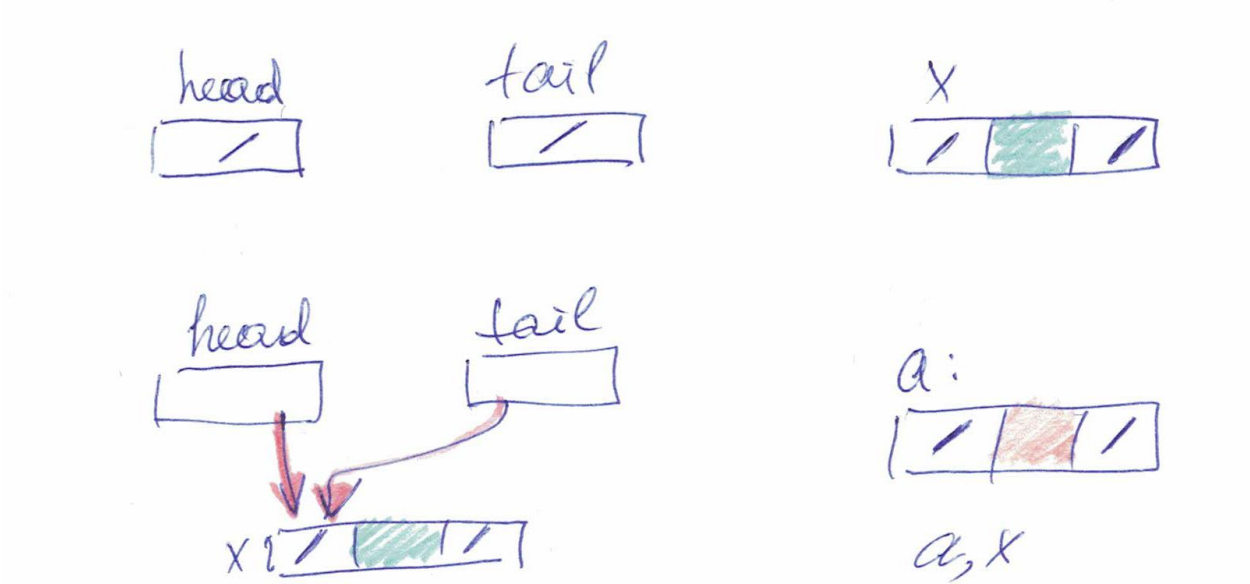


Двусвязный список.

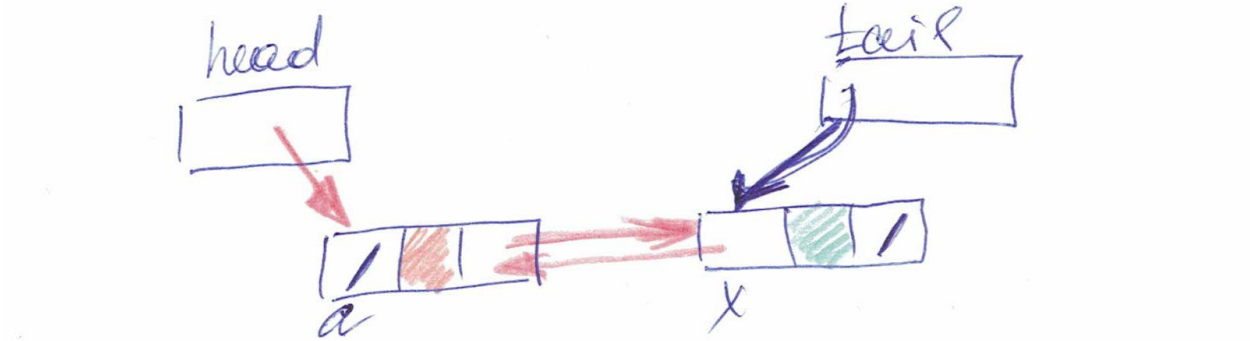


Формирование двусвязного списка

1. Добавление x в пустой список



2. Добавление a в начало списка



Достоинства:

- эффективное (за константное время) добавление и удаление элементов
- размер ограничен только объёмом памяти компьютера и разрядностью указателей
- динамическое добавление и удаление элементов

Недостатки:

Недостатки связных списков вытекают из их главного свойства — последовательного доступа к данным:

- ✓сложность прямого доступа к элементу, а именно определения физического адреса по его индексу (порядковому номеру) в списке
- ✓на поля-указатели (указатели на следующий и предыдущий элемент) расходуется дополнительная память (в массивах, например, указатели не нужны)
- ✓некоторые операции со списками медленнее, чем с массивами, так как к произвольному элементу списка можно обратиться, только пройдя все предшествующие ему элементы
- ✓соседние элементы списка могут быть распределены в памяти не локально, что снизит эффективность кэширования данных в процессоре
- ✓над связными списками, по сравнению с массивами, гораздо труднее (хоть и возможно) производить параллельные векторные операции, такие, как вычисление суммы: накладные расходы на перебор элементов снижают эффективность распараллеливания

Простейшие операции со списками:

- ✓Создать пустой список
- ✓Добавить элемент в начало списка, в конец списка, после (перед) заданным
- ✓Удалить элемент с начала списка, с конца списка, после (перед) заданным
- ✓Возвратить элемент с начала списка, с конца списка, после (перед) заданным
- ✓Определить число элементов списка
- ✓Вывести элементы списка в прямом (обратном порядке)
- ✓. . .

Поддерживающие операции (копирование, уничтожение…)

Рассмотрим типы данных, используемые для работы со списком, и реализацию некоторых операций.

Дома: разобраться с предлагаемым проектом и реализовать несколько методов.

Двунаправленный список (double linked list) –

объект с полем ключа **key**

двумя полями указателями **next** (следующий)и **prev** (предыдущий)

Для заданного элемента списка ***x:***

Указатель ***next*** – указывает на следующий элемент списка

Указатель ***prev*** – указывает на предыдущий элемент списка

Если ***prev = nullptr,*** у элемента ***x нет предшественника (головной).***

Если ***next = nullptr,*** у элемента ***x нет последующего (хвостовой).***

head – указывает на ***первый элемент списка***

tail – указывает на ***последний элемент списка***

head = nullptr, tail = nullptr, – список ***пуст***

Виды списков:

Однократно связанный (однонаправленный, отсутствует prev)

Дважды связанный (двунаправленный)

Отсортированный (линейный порядок соответствует линейному порядку его ключей)

Неотсортированный (элементы располагаются в произвольном порядке)

Кольцевой (указатель prev головного элемента указывает на хвост, указатель next хвостового - на головной)

Не кольцевой

Класс DoubleLinkedList :

двунаправленный связный список из целых чисел (не отсортированный)

1. Элемента списка (узел) - тип Node, в котором значение элемента списка item_ (key_), а для связи между элементами используются поля next__ и prev__.

2. Тип Node может использоваться только в классе DoubleLinkedList

```
// Тип  Node используется для описания элемента списка, связанного со
// следующим с помощью поля next_ и предшествующим с помощью поле prev_

  struct Node    // может использоваться только в классе DoubleLinkedList
  {
    int   item_;      // значение элемента списка (key_)
    Node *next_;      // указатель на следующий элемент списка
    Node *prev_;      // указатель на предшествующий элемент списка
 };
```

```
 private:
// Тип  Node используется для описания элемента списка, связанного со
// следующим с помощью поля next_ и предшествующим с помощью поле prev_
struct Node   // может использоваться только в классе DoubleLinkedList
{
    int   item_;      // значение элемента списка
    Node *next_;      // указатель на следующий элемент списка
    Node *prev_;      // указатель на предшествующий элемент списка

 // Конструктор для создания нового элемента списка.
 Node (int item, Node *next = nullptr, Node *prev = nullptr ):
              item_(item) , next_(next),  prev_(prev)
 { }
};
```

Класс DoubleLinkedList

```
class DoubleLinkedList
{
private:
struct Node            // только в классе DoubleLinkedList
{. . .
};

  Node* head_;         // первый элемент списка
  Node* tail_;         // последний элемент списка (! не обязательно)

  int count_;          // счетчик числа элементов (! не обязательно)
```

```
 public:

 // Конструктор "по умолчанию" - создание пустого списка
 DoubleLinkedList():
    count_( 0 ),
    head_( nullptr ),
    tail_( nullptr )
 {  }

 DoubleLinkedList testList1;  // создание пустого списка
```

```
// Вставить сформированный узел в начало списка (голову)
void DoubleLinkedList ::insertHead (Node* x)
{
     // x->item = …, x->prev_ == nullptr,  x->next_ == nullptr
     x ->next_ = head_;
     if ( head_ != nullptr )  {
         head_->prev_ =  x ;  // список был НЕ пуст – новый элемент будет первым
     }
      else {
         tail_ =  x ; // список был пуст – новый элемент будет и первым, и последним
      }
      head_ =  x ;
      count_ ++ ;      // число элементов списка увеличилось
}
```

```
// Вставить сформированный узел в начало списка (голову)
void DoubleLinkedList ::insertHead (Node* x)
{
  // x->item = …, x->prev_ == nullptr,  x->next_ == nullptr
  // . .
}
```

```
// Вставить элемент (новое число) в начало (голову) списка
void DoubleLinkedList :: insertHead ( int item)
{
  // создаем новый элемент списка и добавляем в начало списка
  insertHead ( new Node ( item ) );
}
```

```
// Вставить сформированный узел в начало списка (голову)
void DoubleLinkedList ::insertHead (Node* x);

 // Вставить элемент (новое число) в начало (голову) списка
void DoubleLinkedList :: insertHead ( int item);
```

```
// main
// создание пустого списка
DoubleLinkedList testList1;

// создаем новый элемент списка и
// добавляем в начало списка   insertHead ( new Node ( item ) );
testList1.insertHead (10);
```

Два набора методов:

1. private для работы с узлами – для разработчика класса
2. public работы со значениями (ключами) – для пользователя

Методы могут быть перегруженными, т.е. можно использовать одно и то же имя для private и public методов.

```
 private:
 // Доступ к головному узлу списка
 Node* head () const { return head_; }


 // Доступ к хвостовому узлу списка
 Node* tail () const { return tail_; }

 // Вставить сформированный узел в хвост списка
 void insertTail ( Node * x );
 // Вставить сформированный узел в начало списка
 void insertHead ( Node * x ); //
 // Удаление заданного узла
 void deleteNode  ( Node * x );
 // Поиск узла (адрес) с заданным значением
 Node* searchNode  ( int item );
 // Замена информации узла на новое
 Node* replaceNode ( Node * x, int item );
```

```
 public:
  // Доступ к информации головного узла списка
  int    headItem ( ) const;
  int&   headItem ( ) ;


   // Доступ к информации хвостового узла списка
   int    tailItem ( ) const ;
   int&   tailItem ( ) ;
  // Вставить число в хвост списка
  void insertTail ( int item);
  // Вставить число в начало списка
  void insertHead (int item );
  // Удаление узла с заданным значением
  void deleteNode  (int item);
  // Поиск узла (адрес) с заданным значением
  bool searchNode  ( int item );
  // Замена информации узла на новое
  bool replaceNode ( int itemOld, int itemNew );
```

Тестирование методов работы с двунаправленным списком DoubleLinkedLis

```
 DoubleLinkedList list1;          // Создание пустого списка
    list.insertHead (2);          // Добавление элементов
    list.insertHead (3);
    list.insertHead (1);
    list1.outAll();                // Печать элементов
    cout << ( (list1.searchItem(1)) ?  "1 find " :  "1 not find ")<< endl;
    cout << ( (list1.searchItem(8)) ? " 8 find " : " 8 not find ")<< endl;
```

Тестирование методов работы с двунаправленным списком DoubleLinkedList

```
DoubleLinkedList  list2 ( list1 );    // Копирование списка
list2.insertHead (4);                 // Добавление элемента
list2.insertHead (5);                 // Добавление элемента
list2.deleteHead ( );                 // Удаление головного
std::cout << list2;                   // Вывод всех элементов списка
list2.insertHead (6);
list2.deleteHead ( );
std::cout << list2;
```

Простейшие операции со списками:

- ✓Создать пустой список
- ✓Добавить элемент в начало списка, в конец списка, после (перед) заданным
- ✓Удалить элемент с начала списка, с конца списка, после (перед) заданным
- ✓Возвратить элемент с начала списка, с конца списка, после (перед) заданным
- ✓Определить число элементов списка
- ✓Вывести элементы списка в прямом (обратном порядке)

Поддерживающие операции (копирование, уничтожение…)

Класс DoubleLinkedList

```
#ifndef __DOUBLE_LINKED_LIST
#define __DOUBLE_LINKED_LIST

class DoubleLinkedList
{
. . . .
};
#endif
```

Домашнее задание: Класс `DoubleLinkedList`

Отчет по работе должен содержать

- ✓описание и реализацию класса `DoubleLinkedList`,
- ✓описание и реализацию дружественных функций,
- ✓функции для тестирования

Домашнее задание. Методы класса `DoubleLinkedList`

```
// Вставить число в хвост списка
void insertTail ( int item);
// Вставить сформированный узел в хвост списка
void DoubleLinkedList::insertTail(Node* x)

// Удалить элемент из хвоста списка
bool DoubleLinkedList::deleteTail()
// Удаление узла с заданным значением
bool DoubleLinkedList::deleteItem(const int item)
// Удаление узла с заданным адресом
void DoubleLinkedList::deleteNode(Node * x)
```

// Заменить информацию узла на новое

DoubleLinkedList::Node* DoubleLinkedList::replaceNode(DoubleLinkedList::Node* x, int item)

// Заменить информацию узла на новое

bool DoubleLinkedList::replaceItem(int itemOld, int itemNew)

// Операция “<<” для вывода элементов «от головы до хвоста» (дружественная функция)

// Операция “==” для сравнения списков (метод класса).

// Списки равны, если равны значения и порядок информационных частей.

// Добавить в хвост исходного списка элементы списка, заданного параметром метода.

// Результат: модифицированный исходный список, пустой список (параметр метода).