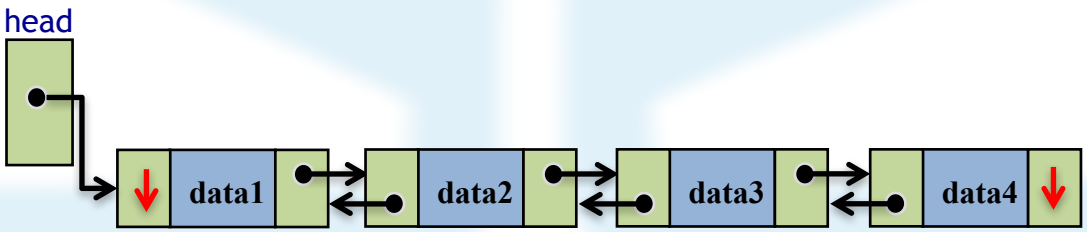
# ساختمان داده‌ها و الگوریتم‌ها

## لیست دو پیوندی

D o u b l y L i n k e d L i s t




**Dr. *Ali Valinejad***

valinejad.ir
valinejad@umz.ac.ir

# پیاده‌سازی لیست دو پیوندی

- **تابع درج یک گره در انتهای لیست (با داشتن داده مربوط به گره):**

```
def Add2EndByValue(self, data):
    new_node= self.DoublyLinkedListNode(None, data, None)
    if self._size == 0:
        self.head = new_node
    else:
        before_node = self.head
        while before_node._next != None:
            before_node =  before_node._next
        before_node._next=new_node
        new_node._prev= before_node
    self._size += 1
    return self.head
```

**گام۱:** تعریف گره

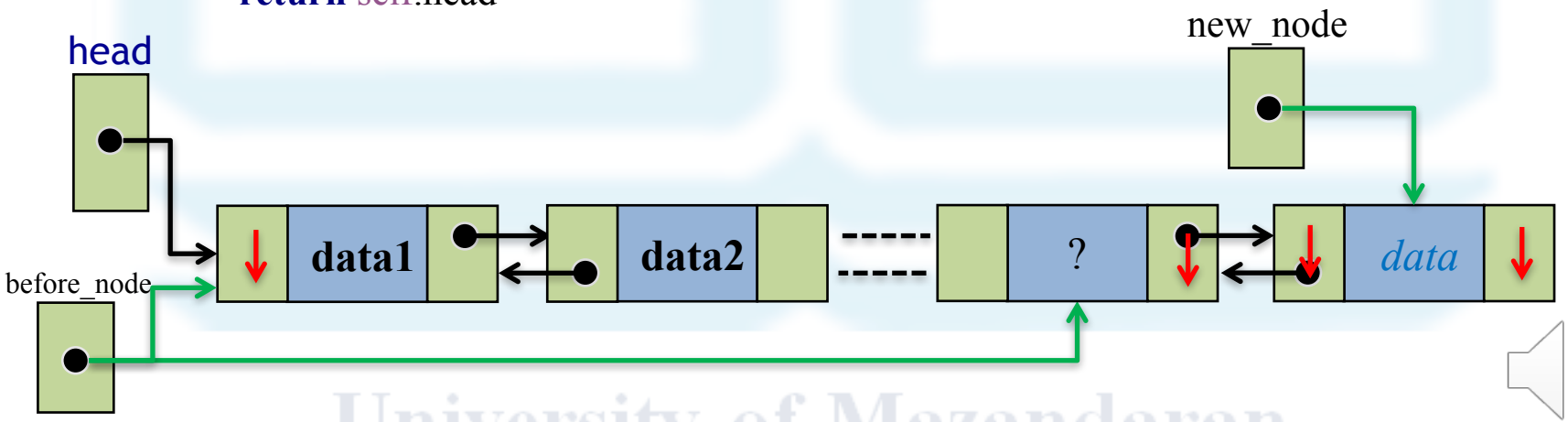
**گام۲:** اضافه کردن گره به لیست

- دستورات لازم در صورتی که لیست خالی باشد:

- دستورات لازم در صورتی که لیست خالی نباشد:

**گام۳:** اضافه کردن یک واحد به _size

**گام۴:** برگرداندن ارجاع به گره اول لیست به عنوان خروجی تابع



# پیاده‌سازی لیست دو پیوندی

- **تابع درج یک گره در انتهای لیست (با داشتن آدرس گره):**

```
def Add2EndByReference(self, node):
    node._next= None
    if self._size == 0:
        self.head = node
    else:
        before_node = self.head
        while before_node._next != None:
            before_node =  before_node._next
        before_node._next=node
        node.prev= before_node
    self._size += 1
    return self.head
```

**گام۱:** تنظیم اشاره گر _next گره

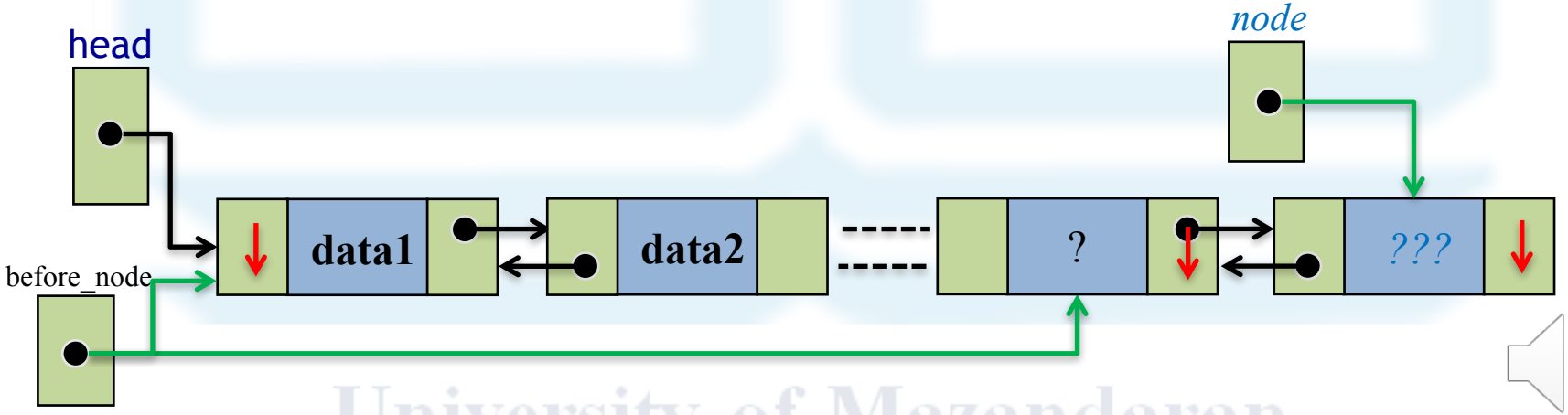
**گام۲:** اضافه کردن گره به لیست

- دستورات لازم در صورتی که لیست خالی باشد:

- دستورات لازم در صورتی که لیست خالی نباشد:

**گام۳:** اضافه کردن یک واحد به _size

**گام۴:** برگرداندن ارجاع به گره اول لیست به عنوان خروجی تابع



# پیاده‌سازی لیست دو پیوندی

قطعه برنامه بهینه

- **تابع درج یک گره در انتهای لیست:**

```
def Add2End (self, inp):
    if type(self.head) != type(inp):
        new_node= self.DoublyLinkedListNode(None, inp, None)
    else:
        new_node= inp
        new_node._next = None
    if self._size == 0:
        self.head = new_node
    else:
        before_node = self.head
        while before_node._next != None:
            before_node =  before_node._next
        before_node._next=new_node
        new_node.prev= before_node
    self._size += 1
    return self.head
```

**گام۱:** تنظیم اولیه:

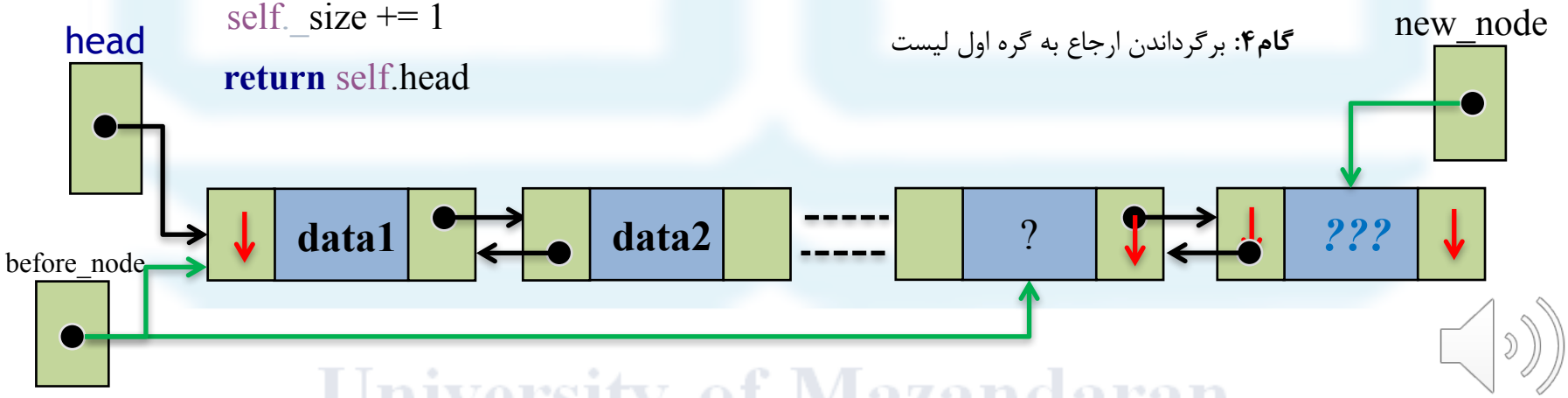
**گام۲:** اضافه کردن گره به لیست

- دستورات لازم در صورتی که لیست خالی باشد:

- دستورات لازم در صورتی که لیست خالی نباشد:

**گام۳:** اضافه کردن یک واحد به _size

**گام۴:** برگرداندن ارجاع به گره اول لیست



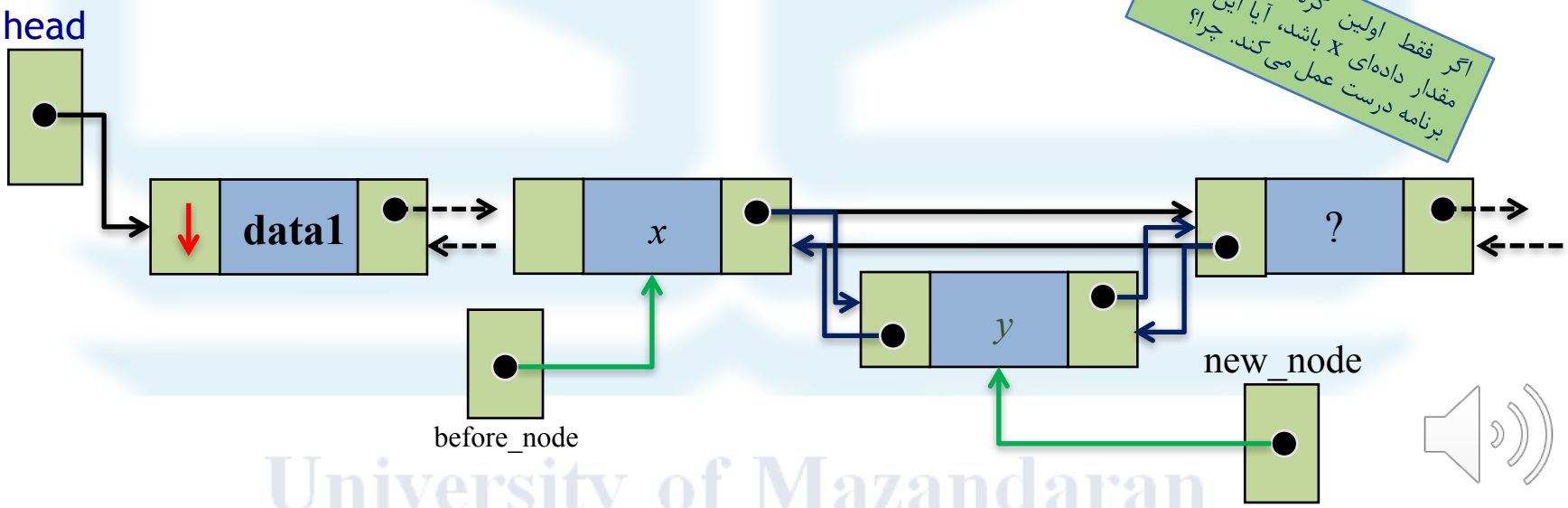
# پیاده‌سازی لیست دو پیوندی

- تابع درج یک گره با مقدار داده‌ای y، بعد از گره‌ای با مقدار داده‌ای x (در صورت وجود):

```
def AddAfterNodeByValue(self, x , y):
    before_node = SearchNodeByValue(self,x):
    if before_node == None :
        return 0
    else:
        new_node= self.DoublyLinkedListNode(before_node , y, before_node._next)
        new_node._prev._next= new_node
        if new_node.next != None :
            new_node._next._prev= new_node
        self._size += 1
    return 1
```

اگر فقط آخرین گره لیست دارای مقدار داده‌ای x باشد، آیا این قطعه برنامه درست عمل می‌کند. چرا؟

اگر فقط اولین گره لیست دارای مقدار داده‌ای x باشد، آیا این قطعه برنامه درست عمل می‌کند. چرا؟



# پیاده‌سازی لیست دو پیوندی

- **تابع حذف یک گره از ابتدای لیست:**

**گام۱:** دستورات حذف یک گره:

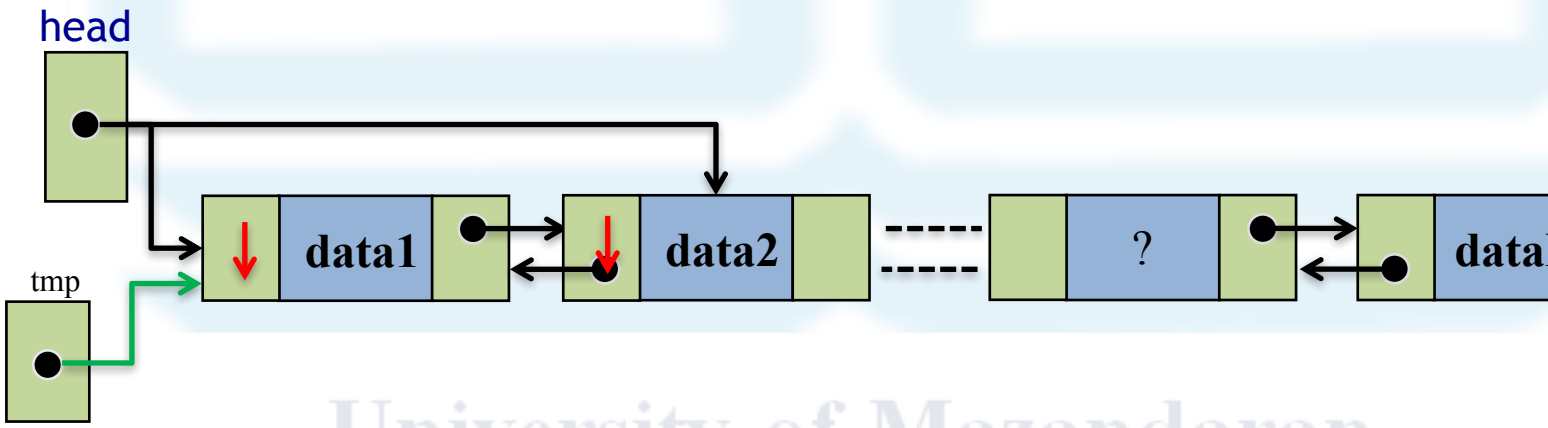
- دستورات لازم در صورتی که لیست خالی باشد:

- دستورات لازم در صورتی که لیست خالی نباشد:

**گام۳:** کم کردن یک واحد از _size

**گام۴:** برگرداندن ارجاع به گره حذف شده به عنوان خروجی تابع

```
def RemoveFirstNode (self):
    if self._size == 0:
        return 0, None
    else:
        tmp = self.head
        self.head = self.head._next
        self.head._prev= None
        self._size -= 1
        return 1, tmp
```



# پیاده‌سازی لیست دو پیوندی

- **تابع حذف یک گره از انتهای لیست:**

```
def RemoveLastNode (self):
    if self._size == 0:
        return 0, None
    elif self._size ==1:
        flag, delete_node = self.RemoveFirstNode ():
    else:
        before_node = self.head
        while before_node._next._next != None:
            before_node =  before_node._next
        delete_node = before_node._next
        before_node._next = None
        flag = 1
        self._size -= 1
    return flag, delete_node
```

**گام۱:** دستورات حذف یک گره:

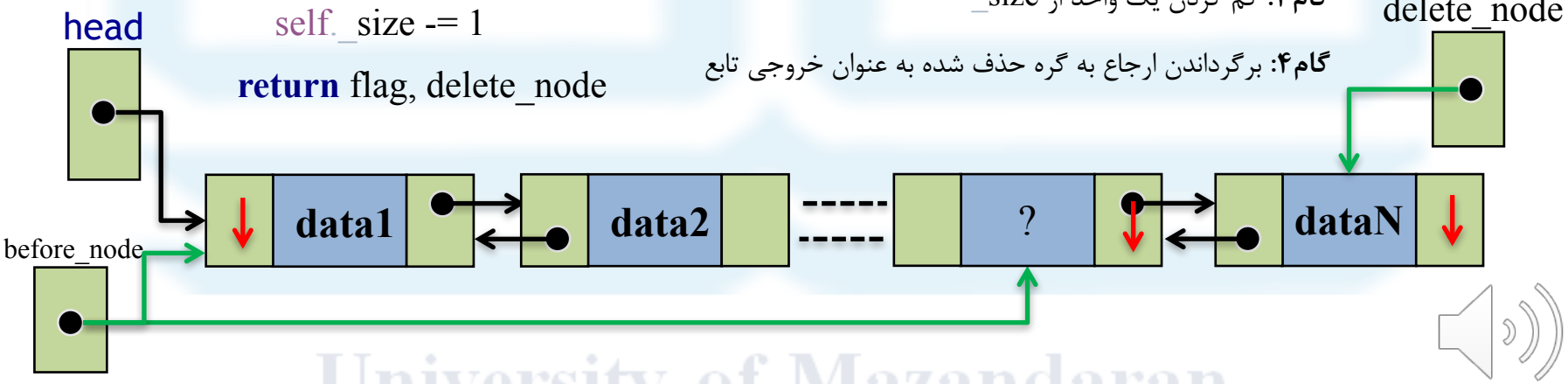
- دستورات لازم در صورتی که لیست خالی باشد:

- دستورات لازم در صورتی که لیست فقط شامل یک گره باشد:

- دستورات لازم در صورتی که لیست شامل بیش از یک گره باشد:

**گام۳:** کم کردن یک واحد از _size

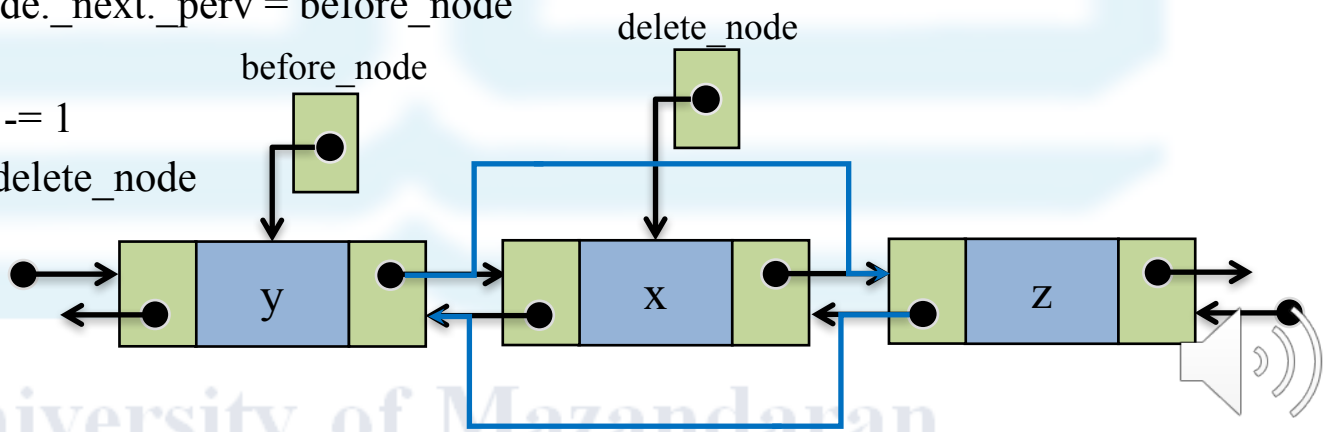
**گام۴:** برگرداندن ارجاع به گره حذف شده به عنوان خروجی تابع



# پیاده‌سازی لیست دو پیوندی

- تابع حذف گره‌ای با مقدار داده‌ای x از لیست(در صورت وجود):

```
def RemoveNodeByValue (self, x):
    delete_node = SearchNodeByValue(self, x):
    if delete_node == None:
        return 0, None
    else:
        if delete_node._prev== None :
            flag, delete_node = self.RemoveFirstNode ():
        elif delete_node._next == None:
            flag, delete_node = self.RemoveLastNode ():
        else:
            before_node = self.head
            while before_node._next != delete_node :
                before_node =  before_node._next
            before_node._next = delete_node._next
            delete_node._next._perv = before_node
            flag = 1
            self._size -= 1
        return flag, delete_node
```



# پیاده‌سازی لیست دو پیوندی

## تمرین

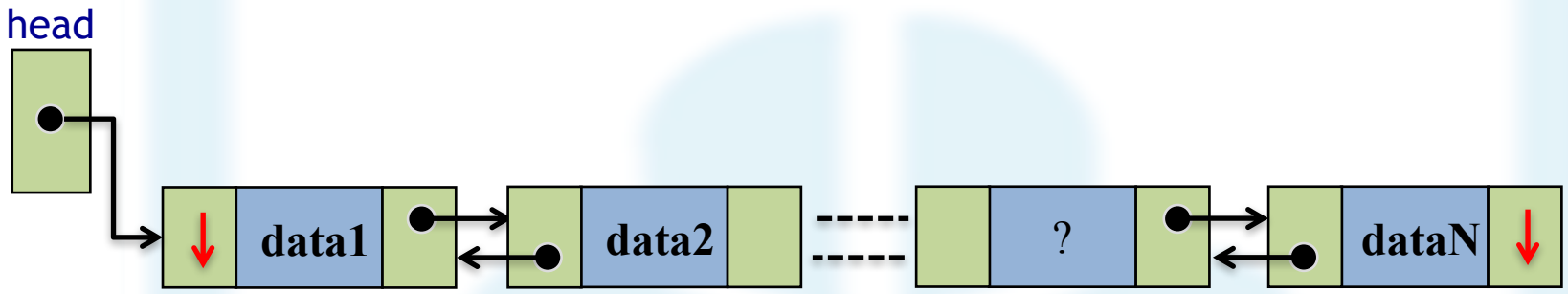
۱- تابعی بنویسید که گره‌ای با آدرس معلوم را قبل از گره‌ای با مقدار داده‌ای x (در صورت وجود) درج نماید.

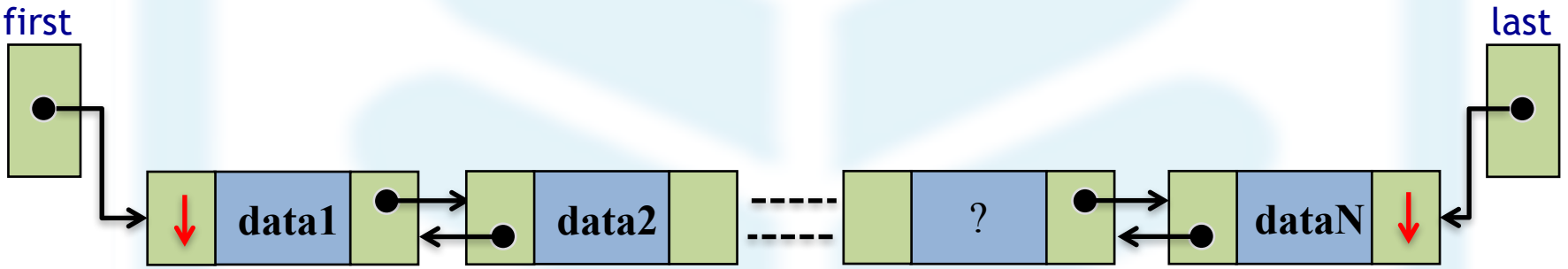
۲- تابعی بنویسید که گره‌ای با آدرس معلوم را بعد از گره‌ای با مقدار داده‌ای x (در صورت وجود) درج نماید.

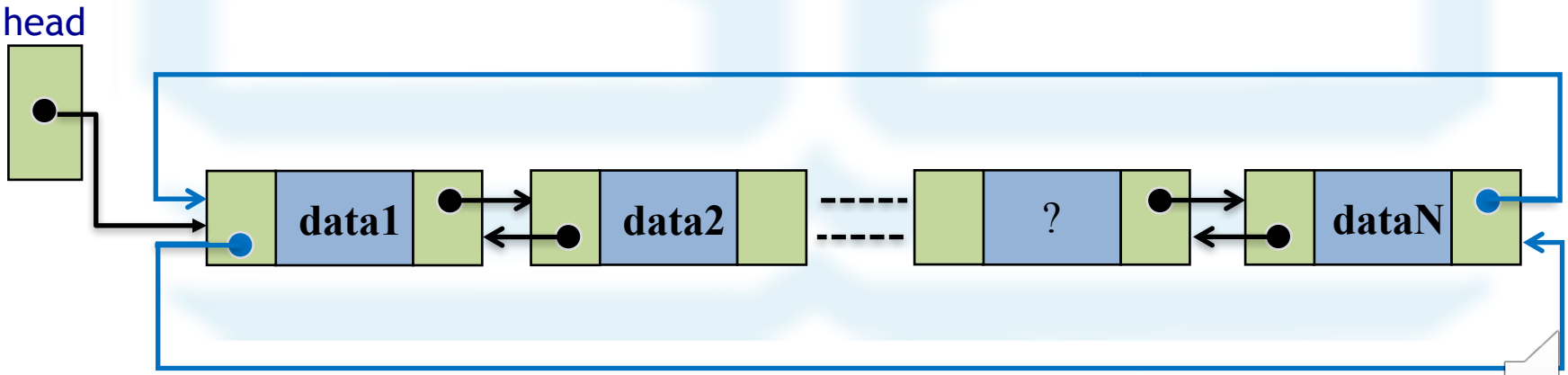
۳- تابعی بنویسید که گره‌ای با مقدار داده‌ای y را قبل از گره‌ای با مقدار داده‌ای x (در صورت وجود) درج نماید.

۴- تابعی بنویسید که گره‌ای با آدرسی خاص را در صورت وجود از لیست حذف کند.

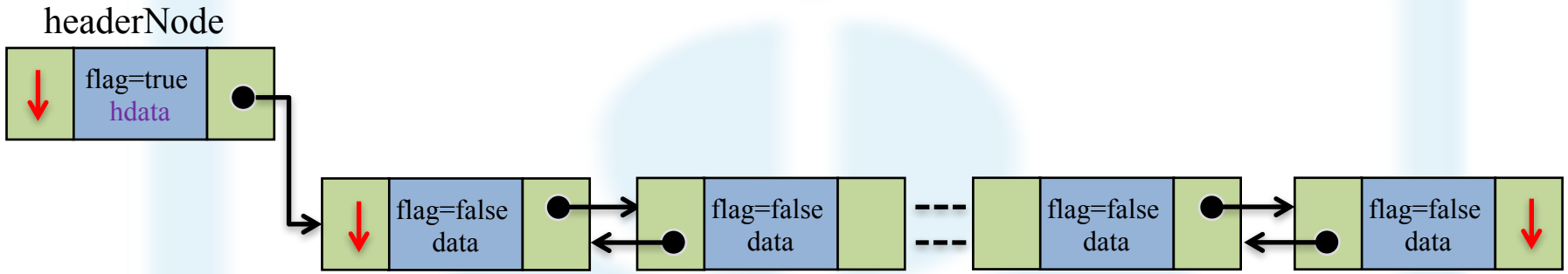
# انواع لیست دو پیوندی

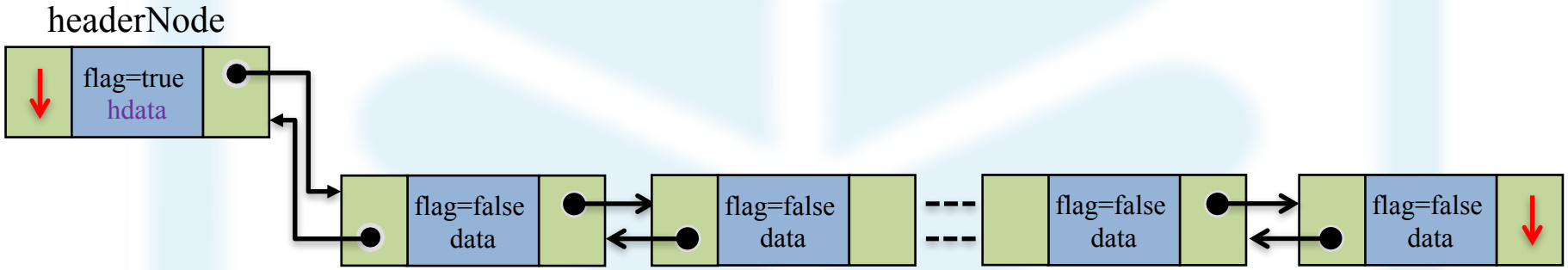


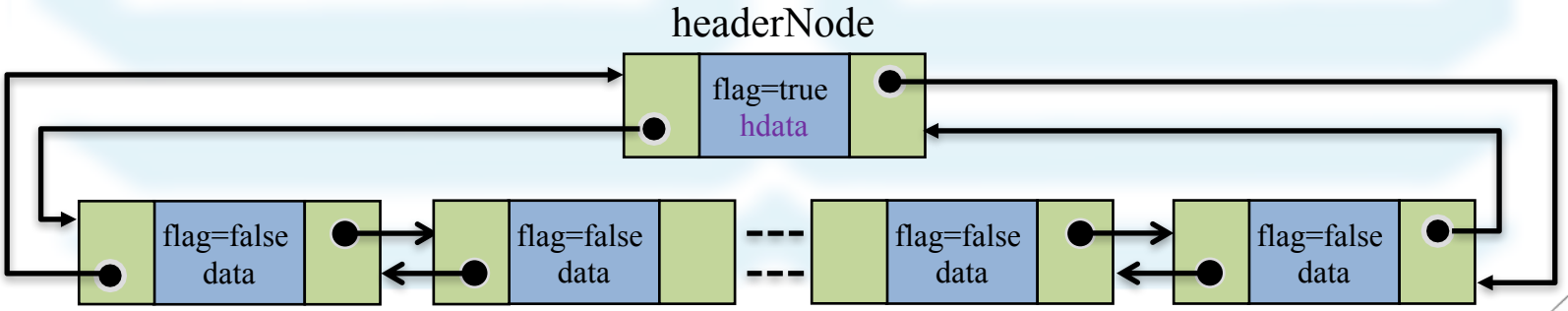



# انواع لیست دو پیوندی



# پروژه

## کاربرد لیست دو پیوندی:

**پروژه دفترتلفن:** برنامه کاملی بنویسید که از لیست دو پیوندی برای ذخیره‌سازی اطلاعات ***نام***، ***نام خانوادگی*** و ***شماره‌ی تلفن*** دوستان شما استفاده می‌کند (اطلاعات باید بر اساس نام خانوادگی به صورت مرتب شده ذخیره شوند).

**بعضی** از متغیرها و توابع مورد نیاز برنامه عبارتند از:

۱- متغیری برای ذخیره‌سازی تعداد اعضای موجود در لیست،
۲- تابعی برای اضافه کردن اطلاعات یک عضو جدید،
۳- تابعی برای حذف کردن اطلاعات یک عضو موجود در لیست،
۴- تابعی برای جستجوی اطلاعات یک عضو بر اساس شماره تلفن،
۵- تابعی برای ویرایش اطلاعات یک عضو،
۶- تابعی برای چاپ اطلاعات همه اعضای موجود در لیست.

** **از ساختار لیست دوپیوندی زیر برای نوشتن برنامه استفاده شود:**