# ساختمان‌های داده

## فصل پنجم: لیست

List

# فهرست مطالب

- لیست
- نوع داده انتزاعی لیست
- لیست پیوندی
- لیست پیوندی یک طرفه
- لیست پیوندی دو طرفه
- لیست پیوندی حلقوی
- عملیات روی لیست پیوندی
- پیاده‌سازی چند جمله‌ای با استفاده از لیست پیوندی

# تعاریف

## ❖لیست (List)

- ❑یک توالی از عناصر است

## ❖لیست مرتب شده (Ordered List)

- ❑لیستی است که محل قرار گرفتن اعضای آن مهم است و تقدم و تأخر در لیست تأثیرگذار است.

$$A =< a_1, a_2, \dots, a_n >$$

# نوع داده انتزاعی لیست

# The Abstract Data Type List

**createList( )**
// post: Create an empty list

**isEmpty( )**
// post: Determine if a list is empty

**add(index, item)**
// post: Insert item at position index of a list
// if 1<=index<= size()+1. If index <= size(), items
// at position index onwards are shifted one position
// to the right

**get(index)**
// post: Returns item at position index of
// a list if 1<=index<=size()

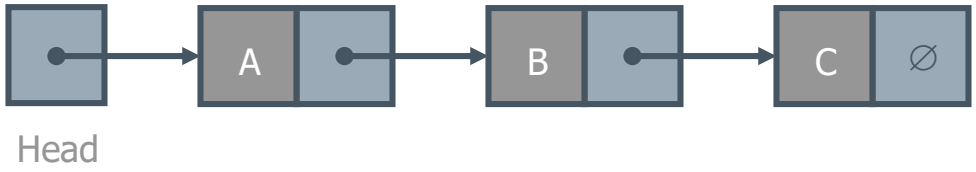
**remove(index)**
// post: Remove item at position index of a list
// if 1<=index<= size(). Items at position
// index+1 onwards are shifted one position to the left

**size( )**
// post: Returns number of items in a list

# لیست پیوندی (Linked List)

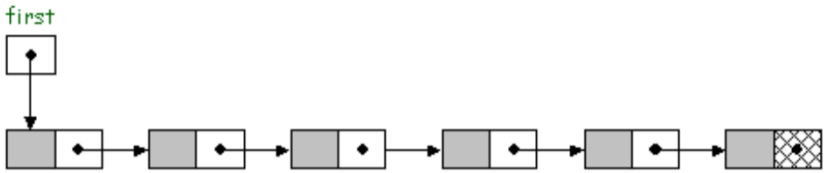
## ❖لیست پیوندی

- ❑دنباله‌ای از گره‌های (node) متصل به هم است.
- ❑هر گره حداقل شامل داده و ارجاعی به گره بعدی در لیست می‌باشد.
- ❑به تعداد عناصر موجود حافظه می‌گیرد، اما برای هر عنصر، یک ارجاع نیز نگه می‌دارد.
- ❑بر خلاف آرایه، امکان دسترسی مستقیم به یک عنصر خاص از لیست وجود ندارد.
- ❑تعداد گره‌های لیست ثابت نیست و طول لیست می‌تواند کم یا زیاد شود.

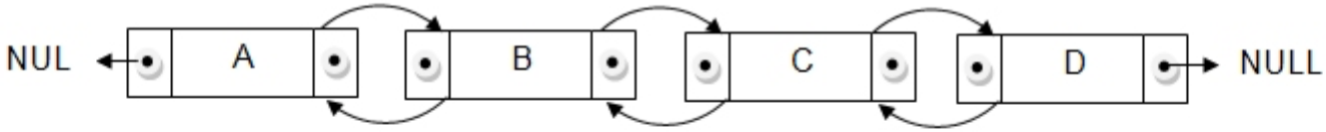


# انواع لیست پیوندی

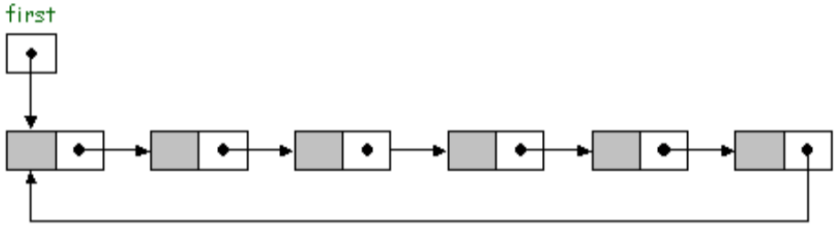
❖لیست پیوندی یک طرفه (Single linked list)




❖لیست پیوندی دو طرفه (Doubly linked list)




❖لیست پیوندی حلقوی (Circular linked list)



# پیاده‌سازی لیست پیوندی

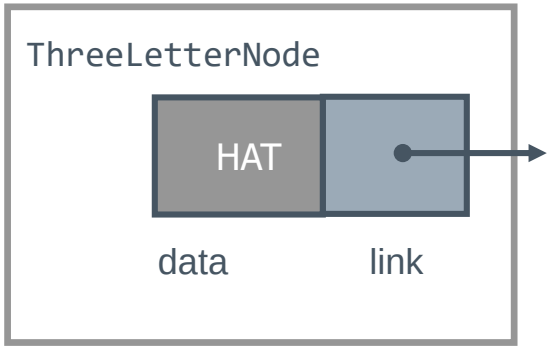
- تعریف گره

```
class Node {
   double data;                    //data
   Node *next;                     //reference to next
};
```

# پیاده‌سازی لیست پیوندی

❖تعریف گره

```
class ThreeLetterNode{
   char data[3];              //data
   ThreeLetterNode *link;     //reference to next
};
```



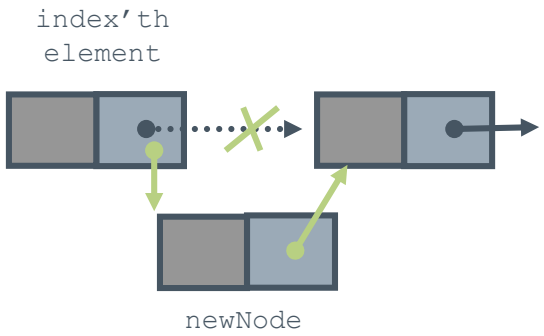
# پیاده‌سازی لیست پیوندی

```
class List {
        public List() { head = null; }       // constructor

        public bool IsEmpty() { return head == null; }
        public Node InsertNode(int index, double x);
        public int  FindNode(double x);
        public int  DeleteNode(double x);
        public void DisplayList();

        private Node head;
};
```

# درج گره جدید

- `void InsertNode(Node *a, double x)`
- void InsertNode(int index, double x)
  - گره‌ای با مقدار داده `x` را بعد از عنصر `index` ام لیست وارد می‌کند. اگر `index` برابر با صفر باشد، گره جدید در ابتدای لیست وارد می شود.

- مراحل:
  1. تعیین محل عنصر `index` ام
  2. تخصیص حافظه برای گره جدید
  3. اشاره دادن گره جدید به گره بعد از خود
  4. اشاره دادن گره قبلی به گره جدید



# درج گره جدید بعد از گره با آدرس a

```
void InsertNode(Node *a, double x) {
        Node *t=new Node(x);
        if(IsEmpty()){
                first=t;
                return;
        }
        t->link=a->link;
        a->link=t;
}
```

# درج گره جدید در محل index

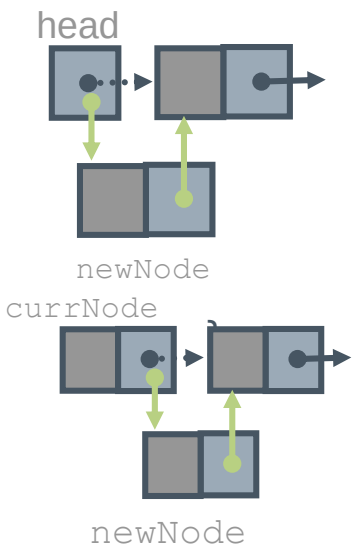
```
void InsertNode(int index, double x) {

    int currIndex = 1;
    Node *currNode = head;
    while (currNode!=null && currIndex  < index ) {
            currNode  = currNode->next;
            currIndex++;
    }

    Node *newNode = new Node();
    newNode->data = x;
    if (index == 0) {
            newNode->next = head;
            head = newNode;
    }
    else {
            newNode->next = currNode->next;
            currNode->next = newNode;
    }
```

تعیین محل عنصر ام index

ایجاد گره جدید

درج گره به عنوان اولین عنصر

درج گره بعد از گره currNode

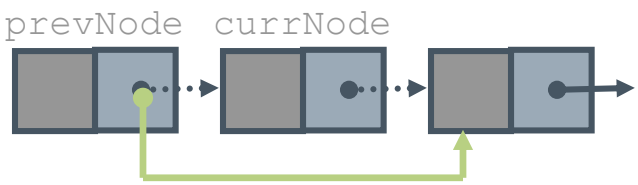


# حذف یک گره

- int DeleteNode(double x)
- Void DeleteNode(Node *a)

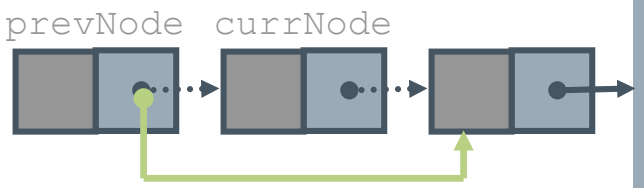
- گره با مقدار x را از لیست حذف می‌کند.
- اگر چنین گره‌ای موجود بود، اندیس آن و در غیر این صورت، مقدار صفر را برمی‌گرداند.

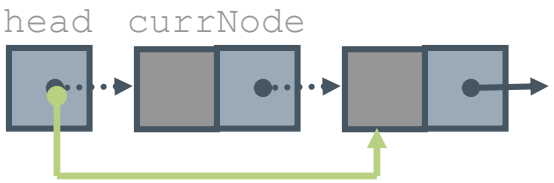
- مراحل
  - پیدا کردن گره مورد نظر
  - آزاد کردن حافظه گره یافته شده
  - اشاره دادن گره قبلی گره یافت شده به گره بعدی آن



# پیدا کردن و حذف یک گره با مقدار x

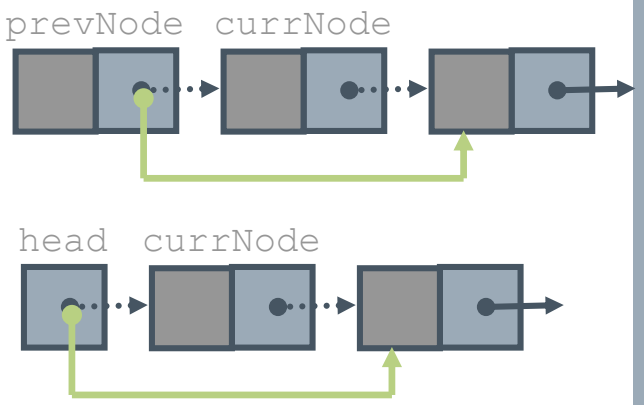
```
int DeleteNode(double x) {
  Node *prevNode = null;
  Node *currNode = head;
  int currIndex = 1;
  while (currNode!=null && currNode->data != x) {
    prevNode = currNode;
    currNode = currNode->next;
    currIndex++;
  }
  if (currNode != null ) {
     if (prevNode != null) {
     prevNode->next = currNode->next;
     currNode = null;
     }
     else {
     head = currNode->next;
     currNode = null;
     }
     return currIndex;
  }
  return 0;
}
```



# حذف گره مدنظر

```
void DeleteNode(Node *a) {
  Node *prevNode = null;
  Node *currNode = head;
  while (prevNode->next != node) {
     prevNode = currNode;
     currNode = currNode->next;
  }
  if (prevNode == null ) {
     head= currNode->next;
    }
    else {
     prevNode->next = currNode->next;
     currNode = null;
    }
}
```



# پیدا کردن یک گره

- `int FindNode(double x)`
  - جستجو به دنبال گره‌ای با مقدار x در لیست و برگرداندن اندیس آن
  - در صورت عدم وجود، مقدار صفر برگردانده می‌شود.

```
int FindNode(double x) {
        Node currNode  = head;
        int currIndex  = 1;
        while (currNode!=null && currNode->data != x) {
                currNode = currNode->next;
                currIndex++;
        }
        if (currNode!= null) return currIndex;
        return 0;
}
```

جستجوی موفق

جستجوی ناموفق

# چاپ تمام عناصر لیست پیوندی

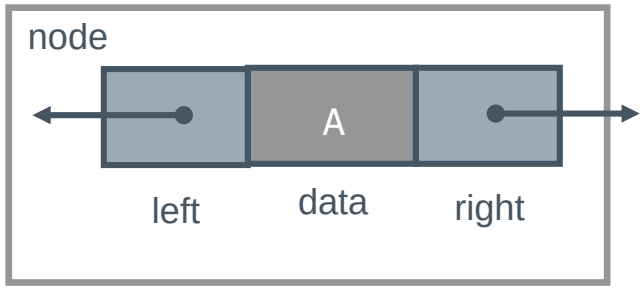
- void DisplayList()
  - چاپ تمام عناصر لیست و تعداد آن‌ها

```
void DisplayList()
{
   int num = 0;
   Node *currNode = head;
   while (currNode != null){
        cout<<currNode->data<<“\n”;
        currNode = currNode->next;
        num++;
   }
   cout<<num;
}
```

# پیاده‌سازی لیست پیوندی دو طرفه

❖تعریف گره

```
class Node {
   double data;                  //data
   Node *right;                  //reference to right node
   Node *left;                   //reference to left node
};
```



# پیاده‌سازی لیست پیوندی دو طرفه

```
class RecList {
        public RecList() { head = null; }    // constructor

        public bool IsEmpty() { return first == null; }
        public Node InsertNode(int index, double x);
        public int  FindNode(double x);
        public int  DeleteNode(double x);
        public void DisplayList();

        private Node first;
};
```

# درج گره جدید بعد از گره با آدرس a

```
void InsertNode(Node *a, double x) {
        Node *t=new Node(x);
        if(IsEmpty()){
                first=t;
                return;
        }
        t->right=a->right;
        a->right->left=t;
        a->right=t;
        t->left=a;
        }
```

# درج گره جدید در محل index

```
void InsertNode(int index, double x) {

        int currIndex = 1;
        Node *currNode = first;
        while (currNode!=null && currIndex  < index ) {
                currNode  = currNode->right;
                currIndex++;
        }

        Node *newNode = new Node();
        newNode->data = x;
        if (index == 0) {
                newNode->right = head;
                head->left=newNode;
                newNode->left=null;
                head = newNode;
        }
        else {
                newNode->right = currNode->right;
                currNode->right->left=newNode;
                currNode->right=newNode;
                newNode->left=currNode;
        }
}
```

تعیین محل عنصر ام index

ایجاد گره جدید

درج گره به عنوان اولین عنصر

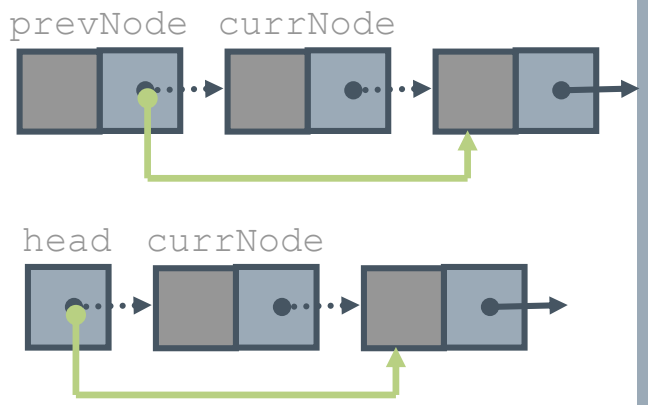
درج گره بعد از گره currNode

# پیدا کردن و حذف یک گره با مقدار x

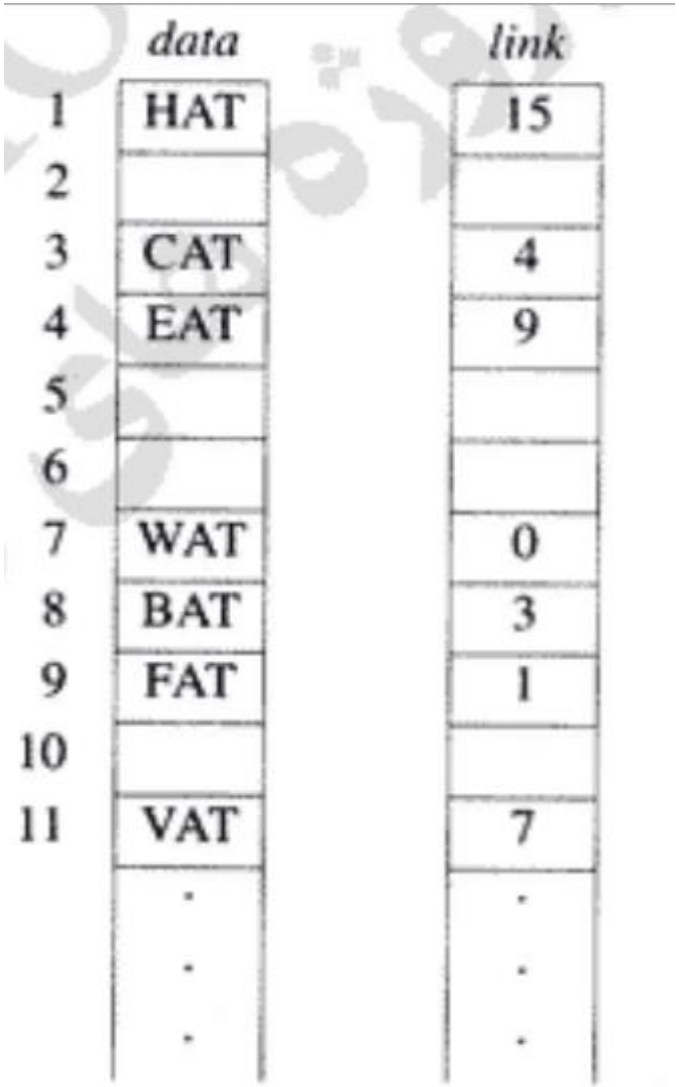
```
int DeleteNode(double x) {
  Node *currNode = first;
  int currIndex = 1;
  while (currNode!=null && currNode->data != x) {
        currNode = currNode->right;
        currIndex++;
  }
  if (currNode != null ) {
     if (currNode->left != null) {
     currNode->left->right = currNode->right;
     curreNode->right->left = currNode->left;
     currNode = null;
    }
    else {
     first = currNode->right;
     first->left=null;
     currNode = null;
    }
    return currIndex;
  }
  return 0;
}
```

# حذف گره مدنظر

```
void DeleteNode(Node *node) {
    Node *currNode = first;
    if (node->left == null ) {
        node->right->left=null;
        first=node->right;
    }
    else {
        node->right->left=node->left;
        node->left->right=node->right;
}
}
```

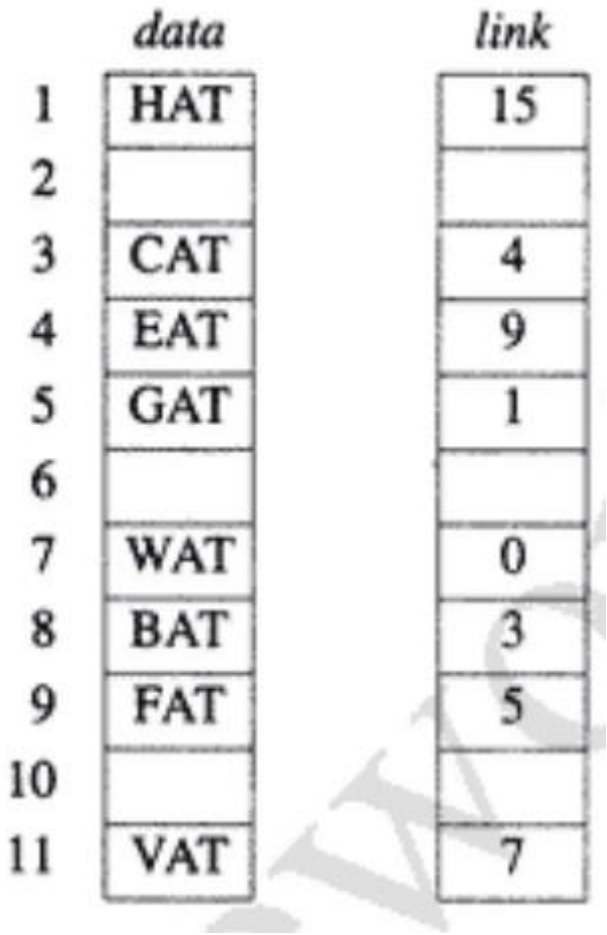


# نمایش لیست مرتب با آرایه



| | data | link |
|---|---|---|
| 1 | HAT | 15 |
| 2 | | |
| 3 | CAT | 4 |
| 4 | EAT | 9 |
| 5 | | |
| 6 | | |
| 7 | WAT | 0 |
| 8 | BAT | 3 |
| 9 | FAT | 1 |
| 10 | | |
| 11 | VAT | 7 |
| | . | . |
| | . | . |
| | . | . |

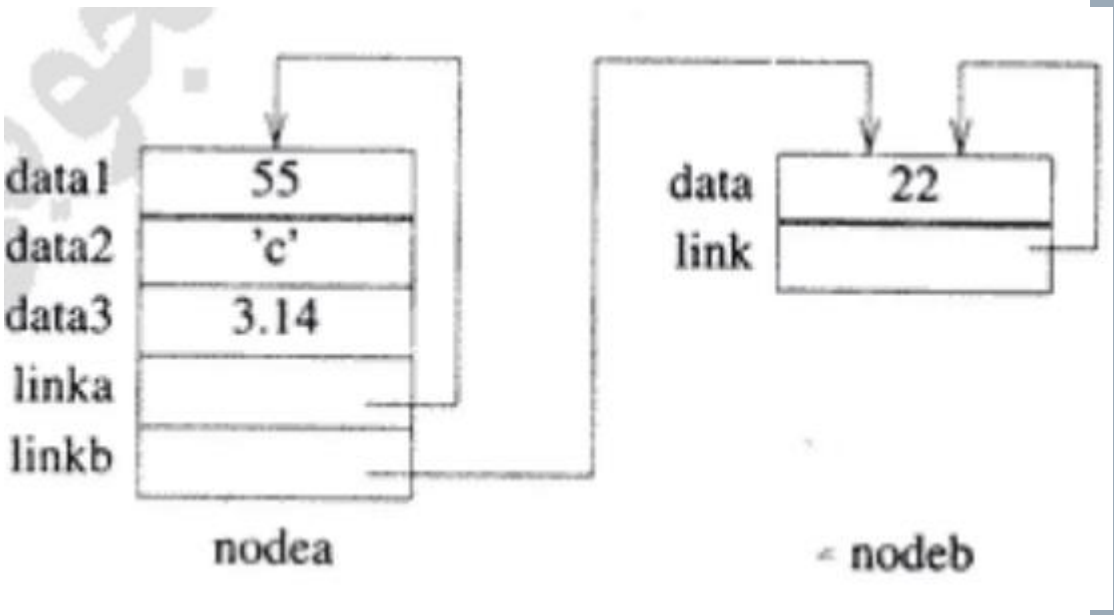
# GAT درج



| | data | link |
|---|---|---|
| 1 | HAT | 15 |
| 2 | | |
| 3 | CAT | 4 |
| 4 | EAT | 9 |
| 5 | GAT | 1 |
| 6 | | |
| 7 | WAT | 0 |
| 8 | BAT | 3 |
| 9 | FAT | 5 |
| 10 | | |
| 11 | VAT | 7 |

# ساختارهای پیچیده تر

```
class nodea {
private:
        int data1 ;
        char data2 ;
        float data3 ;
        nodea *linka ;
        nodeb *linkb ;
};
class nodeb {
private:
        int data ;
        nodeb *link ;
};
```



# نمایش چند جمله ای با لیست پیوندی

```
public class Node {
        int coef;
        int exp;
        Node next;
        Node(int coef, int exp) {
            this.coef = coef;
            this.exp  = exp;
        }
}
```

# نمایش چند جمله ای با لیست پیوندی (ادامه)

ایجاد چند جمله ای صفر

ایجاد چند جمله ای با یک جمله

جمع دو چند جمله ای

```
public class LinkedPolynomial {
    private Node first = new Node(0, 0);
    private Node last  = first;
    // 0
    private LinkedPolynomial() { }
    // a * x^b
    public LinkedPolynomial(int coef, int exp) {
        last.next = new Node(coef, exp);
        last = last.next;
    }
    // return c = a + b
    public LinkedPolynomial plus(LinkedPolynomial b) {
        LinkedPolynomial a = this;
        LinkedPolynomial c = new LinkedPolynomial();
        Node x = a.first.next;
        Node y = b.first.next;
        while (x != null || y != null) {
            Node t = null;
            if      (x == null)     { t = new Node(y.coef, y.exp);  y = y.next; }
            else if (y == null)     { t = new Node(x.coef, x.exp);  x = x.next; }
            else if (x.exp > y.exp) { t = new Node(x.coef, x.exp);  x = x.next; }
            else if (x.exp < y.exp) { t = new Node(y.coef, y.exp);  y = y.next; }
            else {
                int coef = x.coef + y.coef;
                int exp  = x.exp;
                x = x.next;
                y = y.next;
                if (coef == 0) continue;
                t = new Node(coef, exp);
            }
            c.last.next = t;
            c.last = c.last.next;
        }
        return c;
    }
```

# نمایش چند جمله ای با لیست پیوندی (ادامه)

```
public class Main {
    static int a;
    public static void main(String[] args){

        LinkedPolynomial zero = new LinkedPolynomial(0, 0);

        LinkedPolynomial p1   = new LinkedPolynomial(4, 3);
        LinkedPolynomial p2   = new LinkedPolynomial(3, 2);
        LinkedPolynomial p3   = new LinkedPolynomial(1, 0);
        LinkedPolynomial p4   = new LinkedPolynomial(2, 1);
        LinkedPolynomial p    = p1.plus(p2).plus(p3).plus(p4);

        LinkedPolynomial q1   = new LinkedPolynomial(3, 2);
        LinkedPolynomial q2   = new LinkedPolynomial(5, 0);
        LinkedPolynomial q    = q1.plus(q2);

        LinkedPolynomial r    = p.plus(q);
        LinkedPolynomial s    = p.times(q);
        System.out.println("zero(x) =     " + zero);
        System.out.println("p(x) =        " + p);
        System.out.println("q(x) =        " + q);
        System.out.println("p(x) + q(x) = " + r);
        System.out.println("p(x) * q(x) = " + s);
    }
}
```