به نام خدا

ساختمان داده ها

جلسه دهم

دانشگاه صنعتی همدان

گروه مهندسی کامپیوتر

نیم سال دوم 98-1397

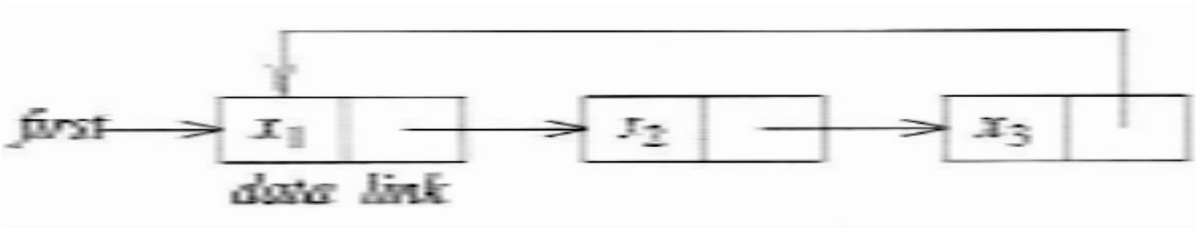
فصل چهارم

لیستهای پیوندی و کاربرد آنها

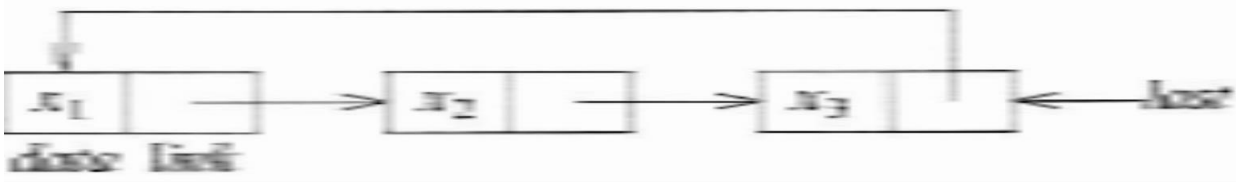
Linked Lists

لیستهای حلقوی Circular linked list

- لیست حلقوی لیستی است که گره انتهایی به گره ابتدا اشاره می کند.
- در لیستهای تک پیوندی با قرار دادن آدرس گره اول در اشاره گر گره آخر لیست حلقوی بوجود می آید.



- در عمل سعی می کنیم آدرس گره آخر برای دسترسی به لیست نگهداری کنیم.



اعمال لیستهای حلقوی

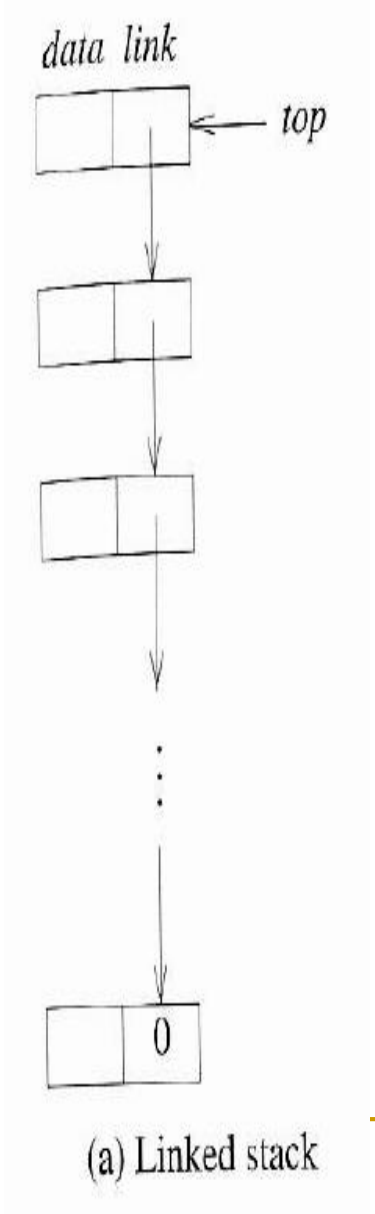
```
class CircList {
friend class CircListIterator<Type>;
public:
    CircList() {first = new ListNode<Type>; first->link = first;};
    void Insert(Type);
    void Delete(Type);
private:
    ListNode<Type> *first;
};
```

```
 void CircList<Type>::Insert(Type k)
 {
 ListNode<Type>
  *newnode = new ListNode<Type>(k);
 newnode->link = first->link;
 first->link = newnode;
 }
```

```
void CircList<Type>::Delete(Type k)
{
    ListNode<Type> *previous = first;
    for (ListNode<Type> *current = first->link;
            (current != first) && (current->data != k) ;
            previous = current, current = current->link);
  if (current != first)
  {  previous->link = current->link;    delete current;   }
}
```

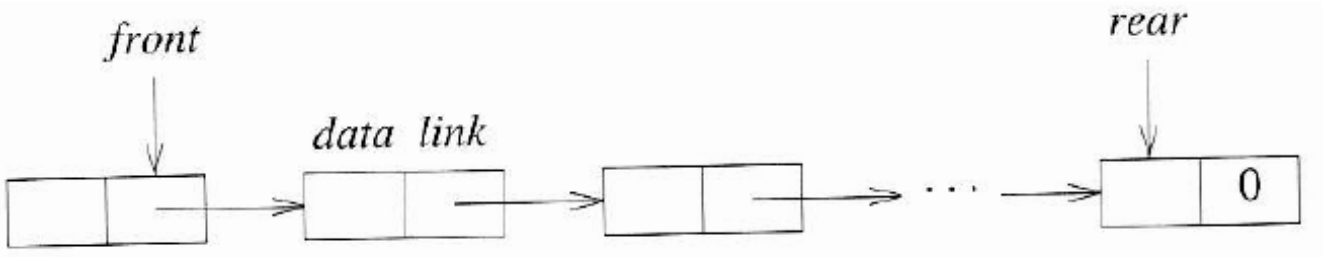
- پشته و صف پیوندی

صف و پشته پیوندی



(a) Linked stack

- برای حل مشکلاتی که در نمایش پشته ها و صف ها با استفاده از آرایه بوجود می آید می توان از پشته و صف پیوندی استفاده کرد.
- با این کار در مصرف حافظه هنگام استفاده از چندین پشته و صف به طور همزمان صرفه جویی می شود
- همچنین محدودیتی در تعداد عناصر صف و پشته وجود نخواهد داشت.



(b) Linked queue

کلاس پشته پیوندی

```
class StackNode {
friend class Stack;
private:
    int data;
    StackNode *link;
    StackNode(int d = 0, StackNode *l = 0) :
    data(d), link(l) {}; // constructor
};

 class Stack {
 private:
     StackNode *top;
     void StackEmpty() {};
 public:
     Stack() { top = 0;};  // constructor
     void Add(const int);
     int* Delete(int&);
 };
```

توابع اضافه و حذف کردن از پشته

```
void Stack::Add(const int y)
{
    top = new StackNode(y, top);
}


int* Stack::Delete(int& retvalue)
//Delete top node from stack and return a pointer to its data
{
    if (top == 0) {StackEmpty(); return 0;}
     StackNode *x = top;
    retvalue = top->data;
    top = x->link;              // remove top node
    delete x;                   // free the node
    return &retvalue;       // return pointer to data
}
```

پشته چند گانه

```
main()
{
  Stack s[50];
  s[15].Add(20);
  s[15].Add(35);
  int retvalue;
  cout << s[15] << endl;
  cout << *s[15].Delete(retvalue);
  cout << s[15];
}
```

کلاس صف پیوندی

```
class QueueNode {
friend class Queue;
private:
    int data;
    QueueNode *link;
    QueueNode(int d = 0, QueueNode *l = 0) :
 data(d), link(l) {};
};
  class Queue {
  friend ostream& operator<<(ostream&, Queue);
  private:
      QueueNode *front, *rear;
      void QueueEmpty() {};
  public:
      Queue() { front = rear = 0;};
      void Add(const int);
      int* Delete(int&);
  };
```

توابع صف پیوندی

```
void Queue::Add(const int y)
{
   if (front == 0)
         front = rear = new QueueNode(y, 0);
   else
         rear = rear->link = new QueueNode(y, 0);
}

   int* Queue::Delete(int& retvalue)
   // delete the first node in queue and return a pointer to its data
   {
      if (front == 0) {QueueEmpty(); return 0;};
      QueueNode *x = front;
      retvalue = front->data;      // get data
      front = x->link;             // delete front node
      delete x;                    // free the node
      return &retvalue;
   }
```

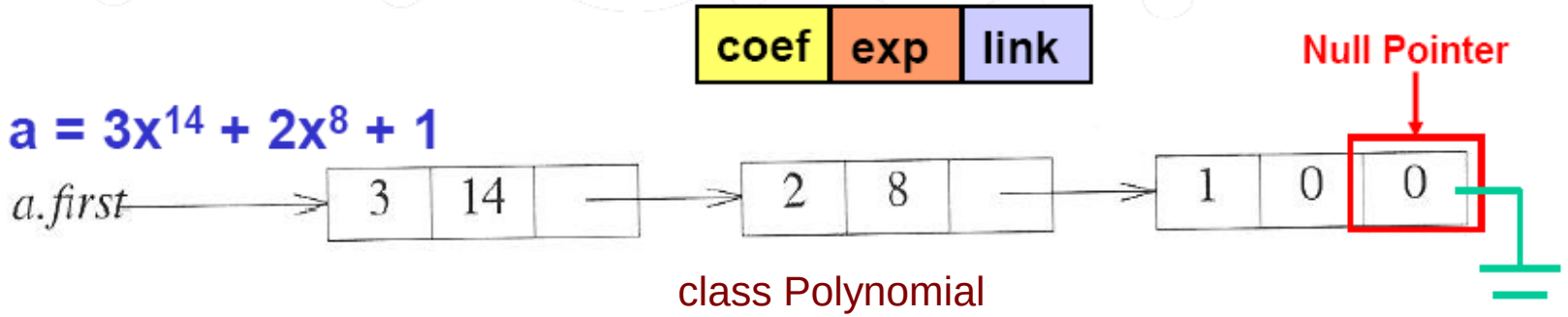
صف پیوندی چند گانه

```
main()
{
  int retvalue;
  int m = 50;
  Queue *s = new Queue[m];
  s[15].Add(20);
  s[15].Add(35);
  s[15].Add(45);
  cout << s[15] << endl;
  cout << *s[15].Delete(retvalue) << endl;
  cout << s[15] << endl;
  cout << *s[15].Delete(retvalue) << endl;
  cout << s[15] << endl;
  cout << *s[15].Delete(retvalue) << endl;
  cout << s[15] << endl;
}
```

ADT چند جمله ایها با لیستهای پیوندی

ADT چند جمله ایها با لیستهای پیوندی

- برای حل مشکل حافظه در نمایش چند جمله ایها با آرایه ها از روش لیستهای پیوندی استفده می کنیم.
- برای این منظور یک کلاس برای نود هر جمله تعریف می کنیم(poly) و سپس یک کلاس دیگر برای چند جمله ای تعریف می کنیم (polynomial)




```
struct Term
{
    int coef; // coefficient
    int exp;  // exponent
    void Init(int c, int e) { coef = c; exp = e;};
};
```

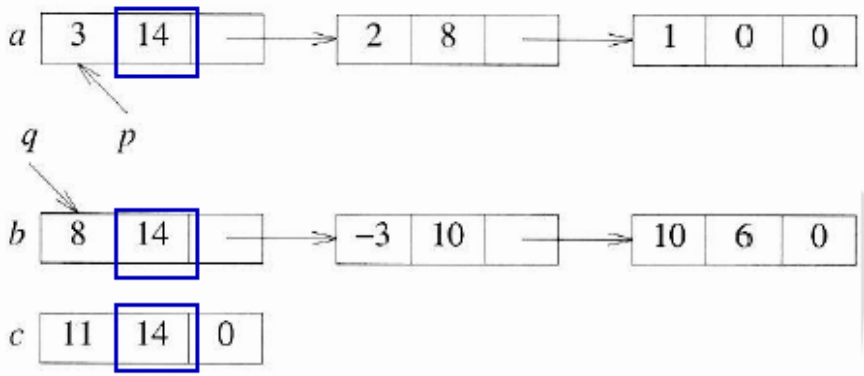
```
class Polynomial
{
friend Polynomial operator+(const Polynomial&,
const Polynomial&);
private:
    List<Term> poly;
public:
    void Add(Term);
};
```

```
void Polynomial::Add(Term e)
{
    poly.Attach(e);
}

template <class Type>
void List<Type>::Attach(Type k)
{
  ListNode<Type> *newnode =  new ListNode<Type>(k);
  if (first == 0) first = last = newnode;
  else {
    last->link = newnode;
    last = newnode;
  }
}
```

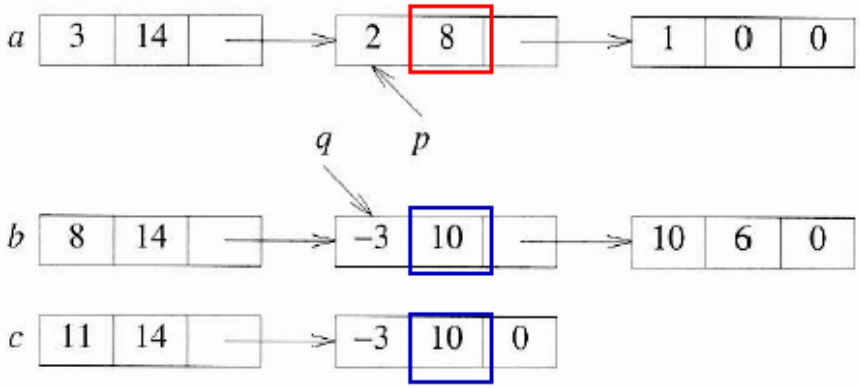
```
main()
{
Polynomial p;
Polynomial *q = new Polynomial;
Term e;
cout << "start" << endl;
e.Init(3, 14); p.Add(e);
e.Init(2,8); p.Add(e);
e.Init(1,0); p.Add(e);
cout << "p" << endl << p;

e.Init(-3,14) ; q->Add(e);
e.Init(-2,8); q->Add(e);
e.Init(-1,0) ; q->Add(e);

cout << "q" << endl << *q;
```

جمع
دو چند جمله ای

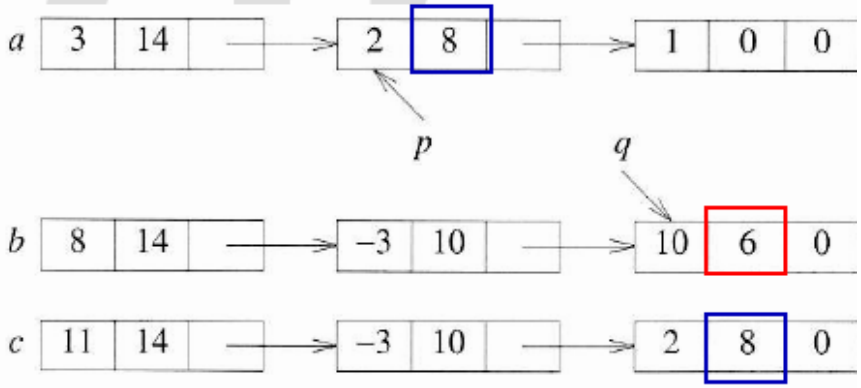
```
Polynomial operator+(const Polynomial& a, const Polynomial& b) {
  Term *p, *q, temp; Polynomial c;
  ListIterator<Term> Aiter(a.poly);   ListIterator<Term> Biter(b.poly);
  p = Aiter.First();   q = Biter.First();
  while (Aiter.NotNull() && Biter.NotNull()) {
          switch (compare(p->exp,q->exp)) {
             case '=':  int x = p->coef + q->coef;
                        temp.Init(x, q->exp);
                        if (x) c.poly.Attach(temp);
                        p = Aiter.Next();
                        q = Biter.Next(); // advance to next term
                        break;
             case '<':  temp.Init(q->coef, q->exp);
                        c.poly.Attach(temp);     q = Biter.Next(); // next term of b
                        break;
             case '>':  temp.Init(p->coef, p->exp);
                        c.poly.Attach(temp);     p = Aiter.Next(); // next term of a
                    }                                          }
  while (Aiter.NotNull()) { temp.Init(p->coef, p->exp);
                            c.poly.Attach(temp);
                            p = Aiter.Next(); }
  while (Biter.NotNull()) { temp.Init(q->coef, q->exp);
                           c.poly.Attach(temp);
                            q = Biter.Next();}
  return c;  }
```



(i) $p \rightarrow exp == q \rightarrow exp$

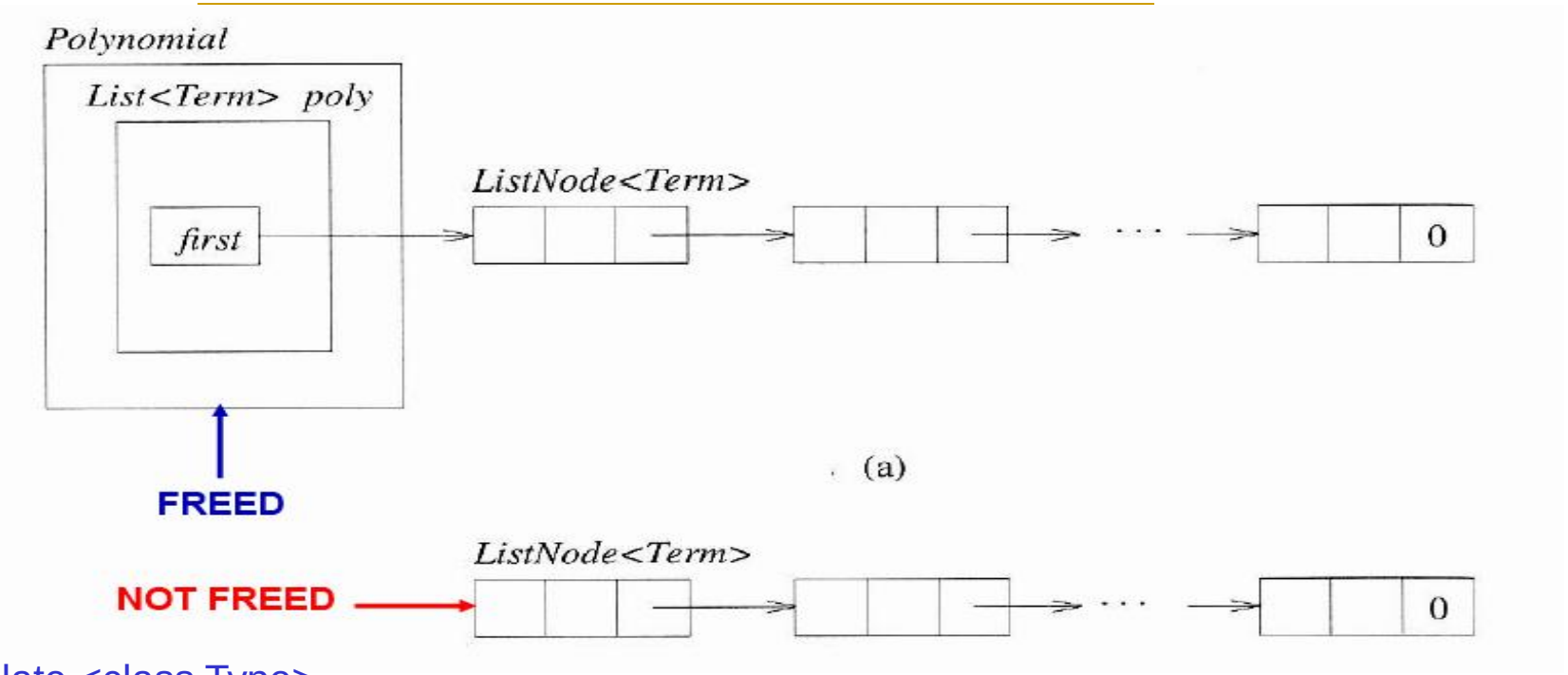


(ii) $p \rightarrow exp < q \rightarrow exp$



(iii) $p \rightarrow exp > q \rightarrow exp$

تخریب کننده چند جمله ایها




```
template <class Type>
List<Type>::~List()
// Free all nodes in the chain
{
   ListNode<Type>* next;
   for (;first; first = next) {
      next = first->link;
      delete first;
   }}
```