به نام خدا

ساختمان داده ها

جلسه نهم

دانشگاه صنعتی همدان

گروه مهندسی کامپیوتر

نیم سال دوم 98-1397

فصل چهارم

لیستهای پیوندی و کاربرد آنها

Linked Lists

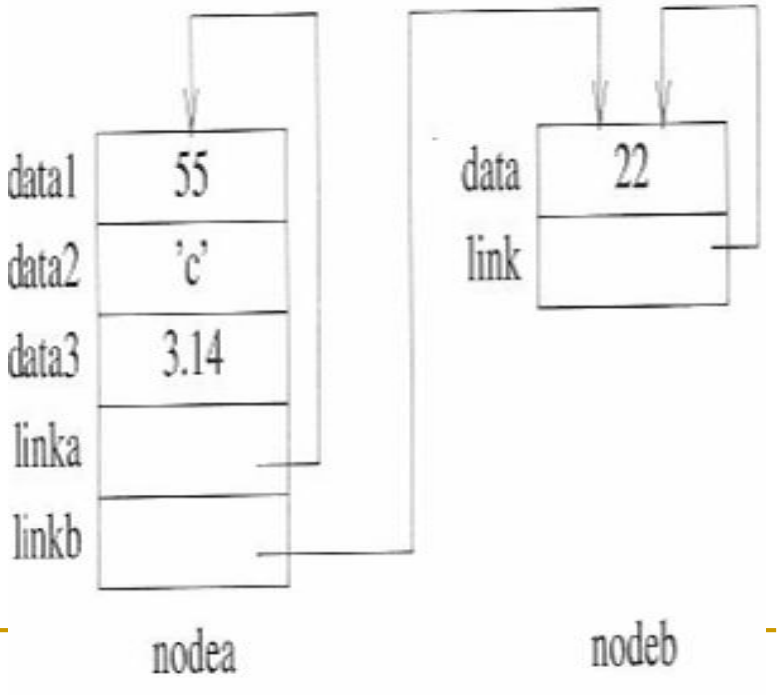
نمایش لیستها در C++

گره یا Node ساختمان داده ای است که از یک فیلد داده ای و یک فیلد اشاره گری که به گره دیگری اشاره میکند تشکیل شده است. مثلا برای مثال قبل تعریف گره می تواند به صورت زیر باشد:

```
class ThreeLetterNode
{
char data[3];
ThreeLetterNode*    link;
};
```

می توان ساختار پیچیده تری از لیستها نیز به صورت زیر تغریف نمود:

```
class nodeb
{
intdata;
nodeb*link;
};
class nodea
{
Int  data1; char  data2;
Float  data3;
nodea*   linka;
nodeb*   linkb;
};
```



طراحی ساختمان داده یک لیست

روش اول : در این روش متغیر سراسری first را به صورت سراسری تعریف می کنیم:

```
ThreeLetterNode*   first;
```

در این حالت برای دسترسی به عناصر این گره داریم

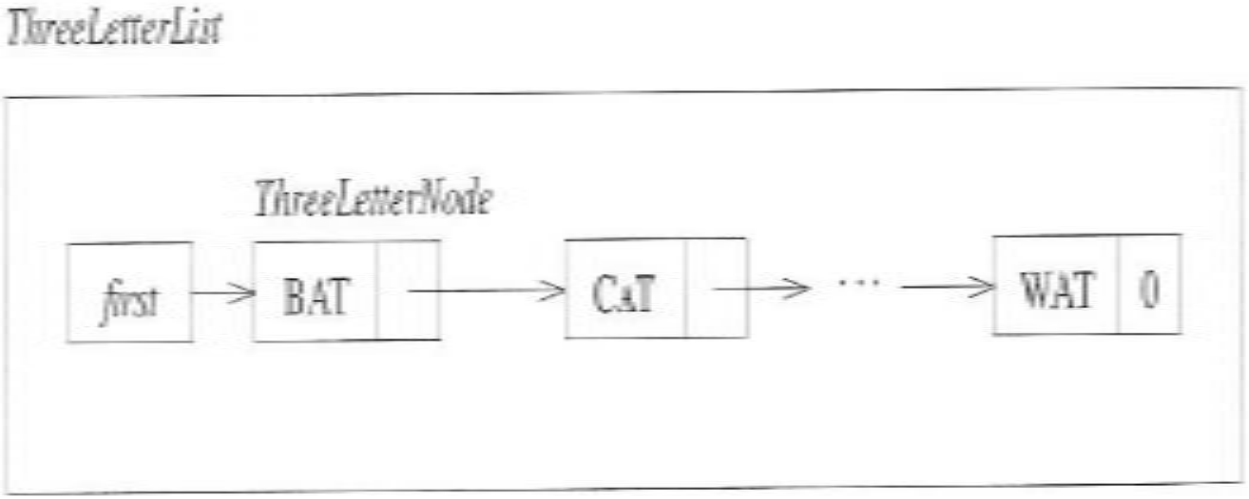
```
First->data;  ,  first->link;
```

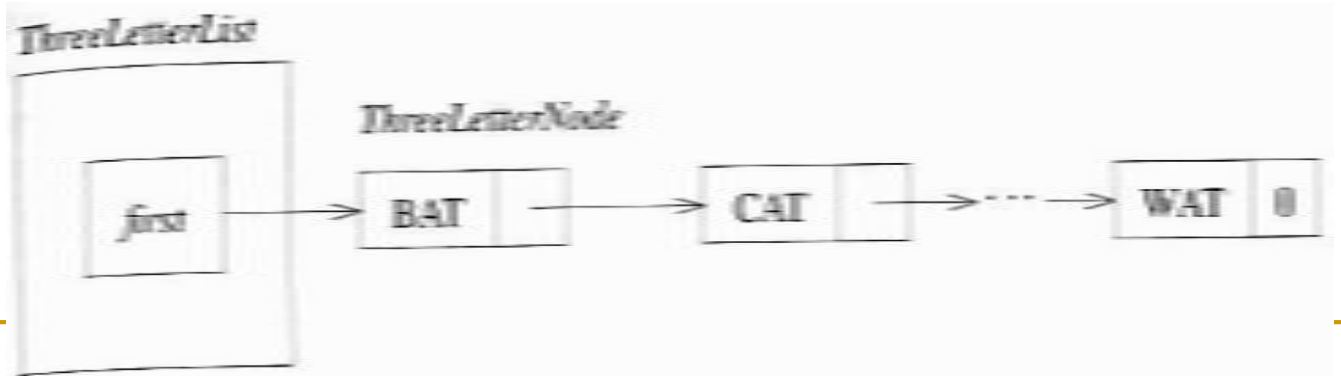
مشکلی که این روش دارد دسترسی به متغیرهای خصوصی است که باعث به وجود آمدن خطا خواهد شد.

روش دوم: برای رفع این مشکل می توان داده ها را به صورت عمومی تعریف کرد یا توابعی برای دسترسی به آنها تعریف نمود که اصل پنهان سازی را با اشکال مواجه می کند.

طراحی ساختمان داده یک لیست

- روش سوم: در این روش یک کلاس برای کل لیست پیوندی تعریف می کنیم که توابعی را که برای دستکاری لیست لازم است پشتیبانی می کند. در واقع این کلاس اشاره گری به اولین گره لیست خواهد بود.





تعریف کلاس لیست به صورت ترکیبی

```
class ThreeLetterList; // forward declaration

class ThreeLetterNode
{
Friend  class ThreeLetterList;
private:
char data[3];
ThreeLetterNode*    link;
}


class ThreeLetterList
{
private:
ThreeLetterNode*   first;

public:       // list manipulation functions, discussed later
};
```

تعریف کلاس لیست به صورت تو در تو

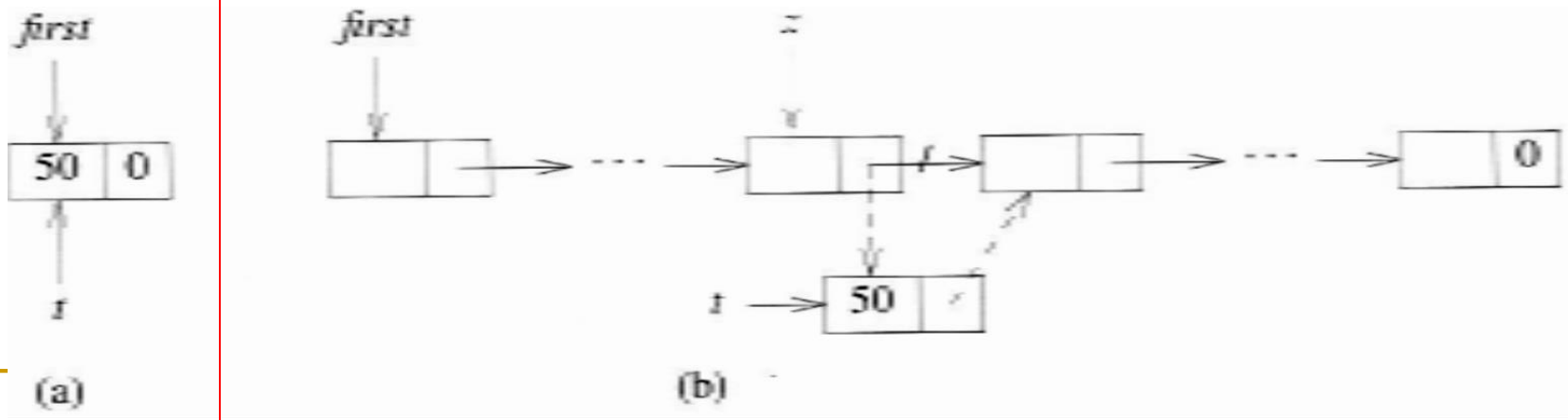
```
class ThreeLetterList
{
private:
class ThreeLetterNode
{
public:
char data[3];
ThreeLetterNode*    link;
}
ThreeLetterNode*   first;

public:       // list manipulation functions, discussed later
};
```

اعمال دستکاری لیستها

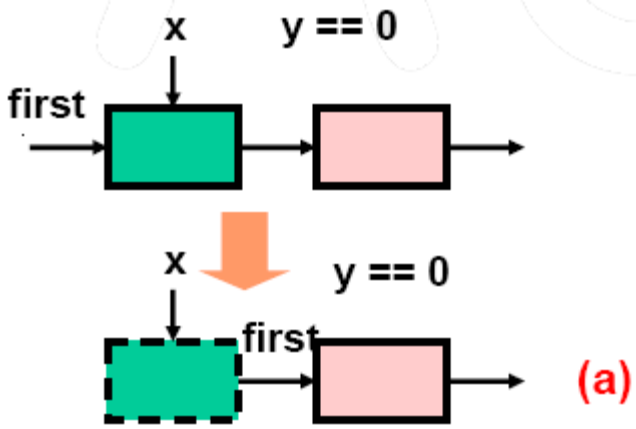
```
class List; // forward declaration
class ListNode
{
friend class List;
int    data;
ListNode*  link;
public:
ListNode(int  value = 0);
}

class List
{
ListNode*  first;
public:
void Create2();
void Insert50(ListNode *x);
void Delete(ListNode*x, ListNode*y);
};
```

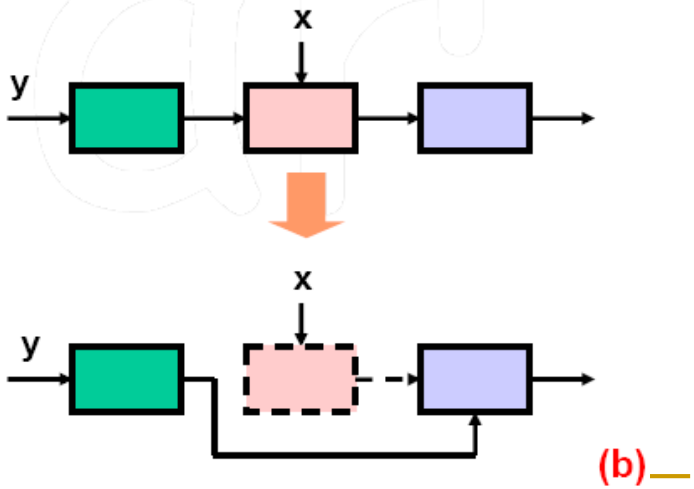
```
ListNode::ListNode(int value):data(value), link(0)
{}

void List::Create2()
{

first = new ListNode(10);
first->link= new ListNode(20);

}
```

```
void List::Insert50(ListNode *x)
 {
ListNode* t = new ListNode(50);
If   (first== 0)
{
first = t;
return;
}
t->link = x->link;
x->link = t;
}
```



```
void List::Delete(ListNode*x, ListNode*y)
{
If  (! y)
first = first->link;
else
y->link = x->link;
delete x;
}
```





ADT لیستهای تک پیوندی توسط الگوها

```
template <class Type>
class ListNode {
friend class List<Type>;
friend class ListIterator<Type>;
private:
    Type data;
    ListNode *link;
    ListNode(Type);
};
 template <class Type>
 ListNode<Type>::ListNode(Type element)
 {
     data = element;
     link = 0;
 }
```

```
template <class Type>
class List {
friend class ListIterator<Type>;
public:
    List() {first = 0;};
    void Insert(Type);
    void Delete(Type);
    void Invert();
    void Concatenate(List<Type>);
private:
    ListNode<Type> *first;
};
```

کلاس تکرار کننده لیستها

```
template <class Type>
class ListIterator {
public:
    ListIterator(const List<Type>& l): list(l), current (l.first) {};
    Boolean NotNull();
    Boolean NextNotNull();
    Type* First();
    Type* Next();
private:
    const List<Type> &list;  // refers to an existing list
    ListNode<Type>* current; // points to a node in list
};
```

توابع مربوط به تکرار کننده لیست

```
Boolean ListIterator<Type>::NotNull()
{ if (current) return TRUE;
    else            return FALSE;
}
```

تابع مشخص کننده نول نبودن گره جاری

```
Boolean ListIterator<Type>::NextNotNull()
{ if (current && current->link) return TRUE;
  else return FALSE;
}
```

تابع مشخص کننده نول نبودن گره بعدی

```
  Type* ListIterator<Type>::First()
  {   if (list.first) return &list.first->data;
        else return 0;
  }
```

تابع برگرداننده آدرس اولین گره

```
      Type* ListIterator<Type>::Next()
      {    if (current) { current = current->link;
                          return &current->data;  }
           else return 0;
      }
```

تابع برگرداننده ادرس گره بعدی

استفاده از کلاس تکرار کننده برای محاسبه
مجموع عناصر لیست

```
int sum(const List<int>& l)
{
  ListIterator<int> li(l);  //li is associated with list l
  if (!li.NotNull()) return 0; // empty list, return 0
  int retvalue = *li.First(); // get first element
  while (li.NextNotNull())  // make sure that next element exists
     retvalue += *li.Next(); // get it, add it to current total
  return retvalue;
}
```

```
ostream& operator<<(ostream& os, List<char>& l)
{
    ListIterator<char> li(l);
    if (!li.NotNull()) return os;
    os << *li.First() << endl;
    while  (li.NextNotNull())
            os << *li.Next() << endl;
    return os;
}
```

اعمال لیستها

اضافه کردن یک گره به لیست

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

حذف یک گره از لیست

```
void List<Type>::Delete(Type k)
{
    ListNode<Type> *previous = 0;
    for (ListNode<Type> *current = first; current && current->data != k;
                previous = current, current = current->link);
   if (current)
   {   if (previous) previous->link = current->link;
           else first = first->link;
       delete current;
   }
}
```

اعمال لیستها

معکوس کردن لیست

```
void List<Type>::Invert()
{  ListNode<Type> *p = first, *q = 0; // q trails p
    while (p) {
       ListNode<Type> *r = q; q = p; // r trails q
       p = p->link; // p moves to next node
       q->link = r; // link q to preceding node
    }
    first = q;
 }
```

الحاق دو لیست

```
void List<Type>::Concatenate(List<Type> b)
{
   if (! first) {first = b.first; return;}
   if (b.first) {
            for (ListNode<Type> *p = first; p->link; p = p->link); // no body
            p->link = b.first;
   }
}
```

مثالی از عملکرد لیستها

```
main()
{
   List<int> intlist;
    intlist.Insert(5);
    intlist.Insert(15);
    intlist.Insert(25);
    intlist.Insert(35);
    cout << endl;
    cout << sum(intlist) << endl;
    intlist.Delete(20);
    intlist.Delete(15);
    intlist.Delete(35);
    cout << sum(intlist) << endl;
```

```
List<char> charlist;
   cout << "shd be empty: " << charlist << endl;
   charlist.Invert();
   cout << "shd be empty: " << charlist << endl;
   charlist.Insert('d');
   charlist.Invert();
   cout << "shd have a d : " << charlist << endl;
   charlist.Insert('c');
   charlist.Insert('b');
   charlist.Insert('a');
   cout << "shd have abcd: ";
   cout << charlist << endl;
   charlist.Invert();
   cout << "shd invert prev list";
   cout << charlist << endl;
```

```
List<char> char2list;
 charlist.Concatenate(char2list);
 cout << charlist << endl;
 char2list.Insert('e');
 char2list.Insert('f');
 char2list.Concatenate(charlist);
charlist.Delete('e');
 charlist.Delete('c');
 cout << char2list << endl;
 char2list.Invert();
 cout << char2list << endl;
```