Data Structures

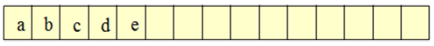
Linked Lists

Teacher : Wang Wei

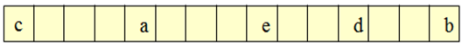
1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. ,
3. http://inside.mines.edu/~dmehta/
4. , ,

Memory Layout

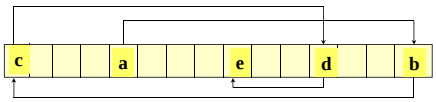
Layout of L = (a,b,c,d,e) using an array representation



A linked representation uses an arbitrary layout



Linked Representation



- ✓ **pointer (or link) in e is NULL**
- ✓ **use a variable first to get to the first element a**

- **In linked list, insertion / deletion of arbitrary elements is much easier :**

The Template Class Chain

```
template<class T>
class Chain
{
   public:
      Chain() {first = 0;}
         // constructor, empty chain
      ~Chain(); // destructor
      bool IsEmpty() const {return first == 0;}
      // other methods defined here
   private:
      ChainNode<T>* first;
};
```


Destructor

```
template<class T>
chain<T>::~chain()
{   // Chain destructor. Delete all nodes in chain.
   while (first != NULL)
   {   // delete first
      ChainNode<T>* next = first->link;
      delete first;
      first = next;
   }
}
```


The Method IndexOf

```
template<class T>
int Chain<T>::IndexOf(const T& theElement) const
{   // search the chain for theElement
   ChainNode<T>* currentNode = first;
   int index = 0;  // index of currentNode
   while (currentNode != NULL &&
          currentNode->data != theElement)
   {      // move to next node
      currentNode = currentNode->next;
      index++;
   }
  // make sure we found matching element
   if (currentNode == NULL)
      return -1;
   else
      return index;
}
```

Insert An Element

```
Template   <class   T>
void  Chain   <T>::Insert(      int    theIndex,
                                const    T& theElement)
{
    if  (theIndex<0)       throw    “Bad insert index”;

    if  (theIndex == 0)                    // insert at front
       first =        new  chainNode<T>(theElement, first);
```



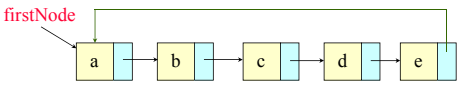
```
   else
   {     // find predecessor of new element
      ChainNode<T>* p = first;
      for    ( int    i = 0; i < theIndex -  1; i++)
      {  if    (p == 0) throw “Bad insert index”;
         p = p->next;
      }
      // insert after p
      p->link =       new  ChainNode<T>(theElement, p->link);
      }
  }
```


Remove An Element

```
  template   <class    T>
  void    Chain<T>::Delete(int  theIndex)
  {
       if    (first == 0)
             throw “Cannot delete from empty chain”;
       ChainNode<T>* deleteNode;
       if    (theIndex  == 0)
       {     // remove first node from chain
             deleteNode  = first;
             first = first->link;
       }
```

```
  else
  {  // use p to get to beforeNode
    ChainNode<T>* p = first;
    for (int i = 0; i < theIndex - 1; i++)
    { if (p == 0) throw "Delete element does not exist";
      p = p->next;
    }
      deleteNode = p->link;
      p->link = p->link->link;
  }
  delete deleteNode;
}
```


a singly-linked Circular List

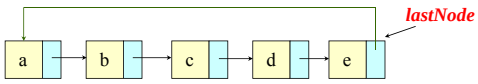


- To check whether a pointer current points to the last node of a circular list

  current→link == first

- Functions for **insertion** into and **deletion** from must ensure
  - –The ***link*** field of the last node points **to** the first node of the list upon completion

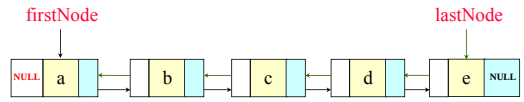
a singly-linked Circular List



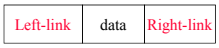
- Suppose want to insert a new node at the front of the list
  - **Move** down the **entire length** of the list **until find** the **last node**
- It is more convenient
  - ***Access pointer*** of the list **points to** the **last node**

```
template <class T>
void CircularList<T>::InsertFront(const T& e)
{   // insert the element e at the "front" of the circular list *this
    // where last points to the last node in the list
  ChainNode<T>* newNode = new ChainNode<T>(e);
  if (last) {                          // nonempty list
     newNode→link = last→link;
     last→link = newNode;
   }
  else {
      last = newNode;
      newNode→link = newNode;
   }
}
```

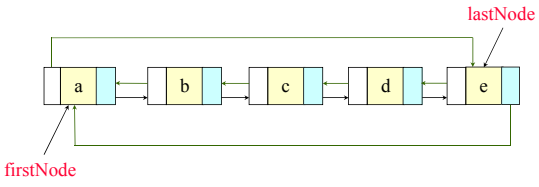

Doubly Linked List



- A node in doubly linked list has at least 3 field

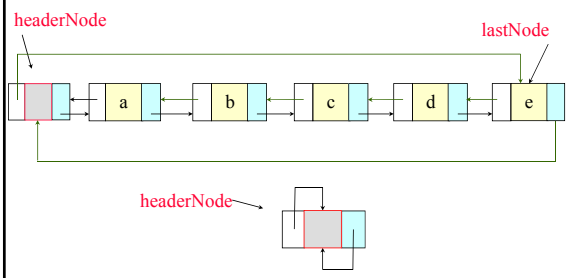


Doubly Linked Circular List



p == p→llink→rlink == p→rlink→llink

Doubly Linked Circular List With Header Node



Empty Doubly Linked Circular List With Header Node

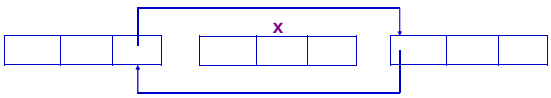

```
class DblList;

class DblListNode {
friend class DblList;
private:
   int data;
   DblListNode *left, *right;
};

class DblList {
public:
   // List manipulation operations
   // …
private:
   DblListNode *first;    // points to head node
};
```


Delete

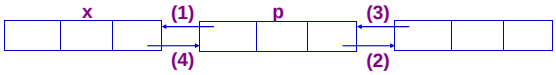
```
void DblList::Delete(DblListNode *x )
{
   if(x == first) throw "Deletion of head node not permitted";
   else  {
      x→left→right = x→right;
      x→right→left = x→left;
      delete x;
   }
}
```



Insert

```
void DblList::Insert(DblListNode *p, DblListNode *x )
{ // insert node p to the right of node x

      p→left = x;                 // (1)
      p→right = x→right;          // (2)
      x→right→left = p;           // (3)
      x→right = p;                // (4)
}
```



Data Structures

Generalized Lists

Teacher : Wang Wei

1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. ,
3. http://inside.mines.edu/~dmehta/
4. ,

Generalized Lists (GL :)

- Definition
  - Is a finite sequence of n ů 0 elements
  - Let : $A = (a_0, a_1, a_3, \ldots, a_{n-1})$
    - A is the mane of the generalized list
    - a_i is either an atom or a sublist , $0 \le i \le n-1$
      - Atoms are represented by lowercase letters
      - All sumlist names are represented by capital letters
    - n is its length
  - If n ů 1
    - a_0 is the *head* of A
    - $(a_1, a_3, \ldots, a_{n-1})$ is *tail* of A

- A=( )
  - the null or empty list; its length is zero;
- B=(a,(b, c))
  - length is two
  - first element is atom *a*, second element is linear list (b,c)
- C(B,B, ( ))
  - Length is three
  - First two elements are list B, third element is null list
- D=(a, D)
  - Recursive list; length is two
  - D corresponds to (a,(a,(a,…) ← infinite list

$P(x, y, z) = C(x, y)z^2 + D(x, y)z$

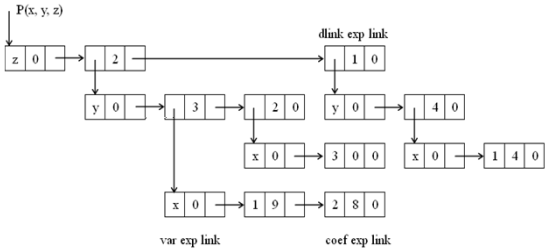
$C(x, y) = E(x)y^3 + F(x)y^2$

$D(x, y) = G(x)y^4$

$E(x) = x^9 + 2x^8$

$F(x) = 3$

$G(x) = x^4$



GenListNode

```
Template <class T> class GenList;    // forward declaration
enum Boolean {FALSE, TRUE};
Template <class T>
class GenListNode
{                                    tag = FALSE/TRUE   data/dlink   link
 friend class GenList<T>;
 private:
    Boolean tag;
    union { char data; GenListNode *dlink; };
    GenListNode<T> *link;
};
```

GenList class

```
Template <class T>
class GenList
 {
   public:
      // operations (                )
   private:
      GenListNode<T> *first;
};
```



```
template <class T>                //
void GenList<T>::Copy(const GenList<T> & P)
   first = Copy(P.first);         //
};
template <class T>
GenListNode<T>  *GenList<T> ::Copy(GenListNode<T> * p)
{   GenListNode<T>  *q = 0;
    if (p) {                                  // p
        q = new GenListNode;                  //
        q→tag = p→tag;
        if (!p→tag)   q→data = p→data;        //
       else  q→dlink = Copy(p→dlink);         //
       q→link = Copy(p→link);                 //
   }
   return q;
}
```


GenList Depth

$$\text{Depth(LS)} = \begin{cases} 1, & \textit{empty} \\ 0, & \textit{atom} \\ 1 + \max\limits_{0 \le i \le n-1} \{\text{Depth}(\alpha_i)\}, & \textit{other}, \ n \ge 1 \end{cases}$$

Example

E (B (a, b), D (B (a, b), C (u, (x, y, z)), A ( )))