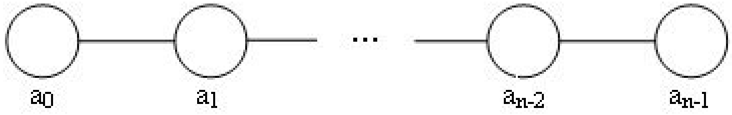
Unit 3 Lists

Array-Based lists

College of Computer Science, CQU



outline

- Definition of ADT
- Array-based List (Sequential List)（顺序表）
- Linked List（链表）
  - Singly Linked List
  - Circular Linked List
  - Doubly Linked List
- Applications

1 Definition

- list
- Length of list
- Empty list
- Order

Definition

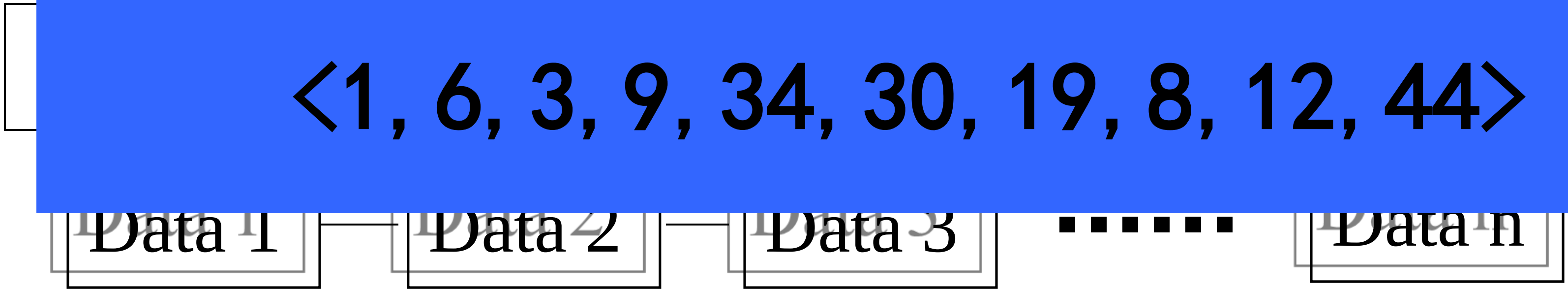
Sorted list

<1, 3, 5, 6, 8, 9, 21, 24, 56, 77>

<98, 65, 43, 23, 11, 10, 9, 6, 5, 4, 2>

Unsorted list

<1, 6, 3, 9, 34, 30, 19, 8, 12, 44>



Data 1 Data 2 Data 3 ...... Data n

Important concept

- a_i is previous（前驱） of element a_{i+1}

  a_i is next（后继） element of a_{i-1}

  a_i and a_{i+1} are adjacent element
- If a element don’t have predecessor ,it is called head element.
- If a element don’t have successor , it is called tail element.
- If a element have both predecessor and successor , it is called intertal element.

Important concept

- Length of list

The number of elements currently stored is called the length of the list.

- Empty list

A list is said to be empty when it contains no elements. the empty list would appear as <>.

- Order（位序）

Order of a element is it’s position in the list.

List Implementation Concepts

Our list implementation will support the concept of a current position.

Operations will act relative to the current position.

- Example: <20, 23 | 12, 15> to indicate the list of four elements, with the current position being to the right of the bar at element 12.

What operations should we implement?

List ADT

```
template <typename E> class List { // List ADT
private:
  void operator =(const List&) {}      // Protect assignment
  List(const List&) {}           // Protect copy constructor
public:
  List() {}          // Default constructor
  virtual ~List() {} // Base destructor

  // Clear contents from the list, to make it empty.
  virtual void clear() = 0;

  // Insert an element at the current location.
  // item: The element to be inserted
  virtual void insert(const E& item) = 0;

```

List ADT

```
// Append an element at the end of the list.
  // item: The element to be appended.
  virtual void append(const E& item) = 0;

  // Remove and return the current element.
  // Return: the element that was removed.
  virtual E remove() = 0;

  // Set the current position to the start of the list
  virtual void moveToStart() = 0;

  // Set the current position to the end of the list
  virtual void moveToEnd() = 0;
```

List ADT

```
// Move the current position one step left. No change  if already at beginning.
  virtual void prev() = 0;
  // Move the current position one step right. No change  if already at end.
  virtual void next() = 0;
  // Return: The number of elements in the list.
  virtual int length() const = 0;

  // Return: The position of the current element.
  virtual int currPos() const = 0;

  // Set current position. pos: The position to make current.
  virtual void moveToPos(int pos) = 0;
  // Return: The current element.
  virtual const E& getValue() const = 0;
};
```

List ADT Examples

List: <12 | 32, 15>

```
L.insert(99);
```

Result: <12 | 99, 32, 15>

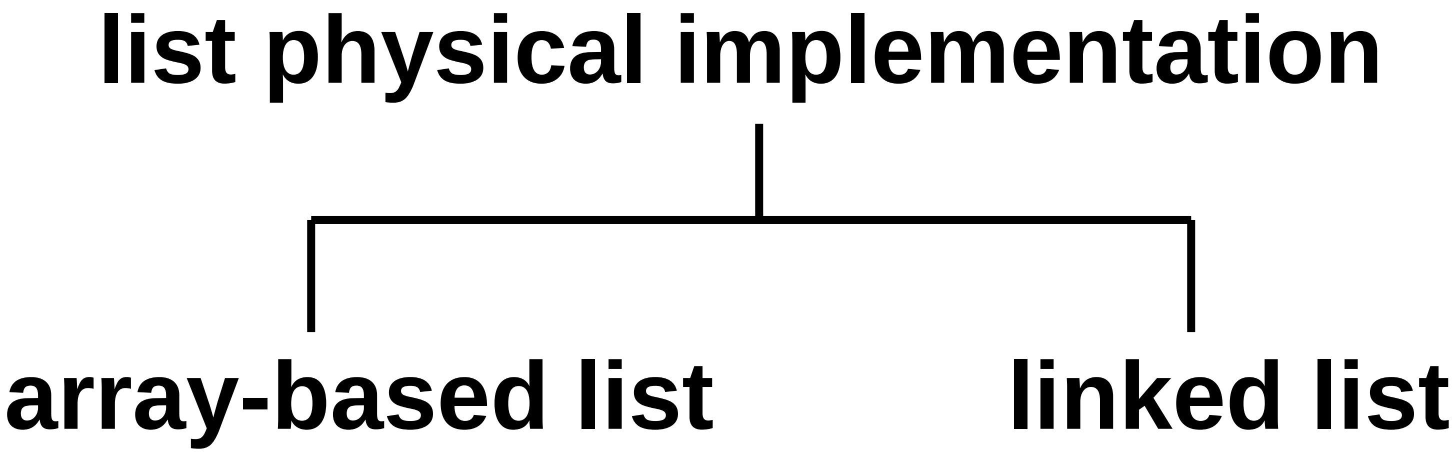
Iterate through the whole list:

```
for (L.moveToStart(); L.currPos()<L.length(); L.next())
{  it = L.getValue();
   doSomething(it);
}
```

List Find Function

```
//return True if k is in list L,
//false otherwise
bool find(List<int>& L, int k) {
    int it;
    for (L.moveToStart();
       L.currPos()<L.length(); L.next())
    {it=L.getValue();
     if (k == it) return true;}
    return false;           // k not found
}
```

Array-Based List



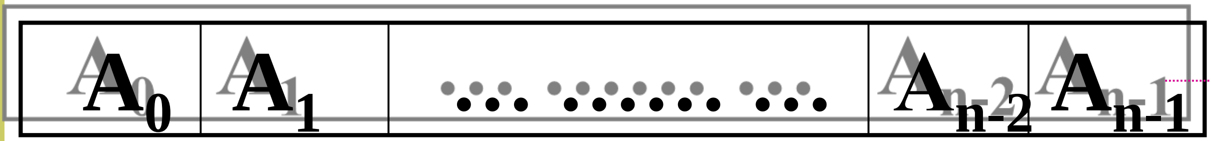
list physical implementation

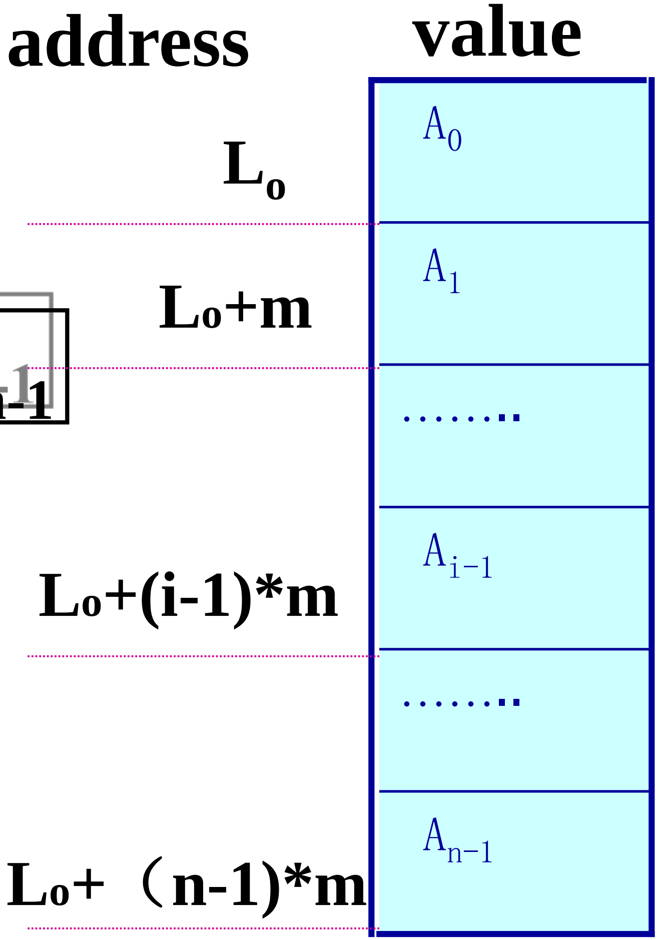
array-based list

linked list

Array-Based List (cont)

element



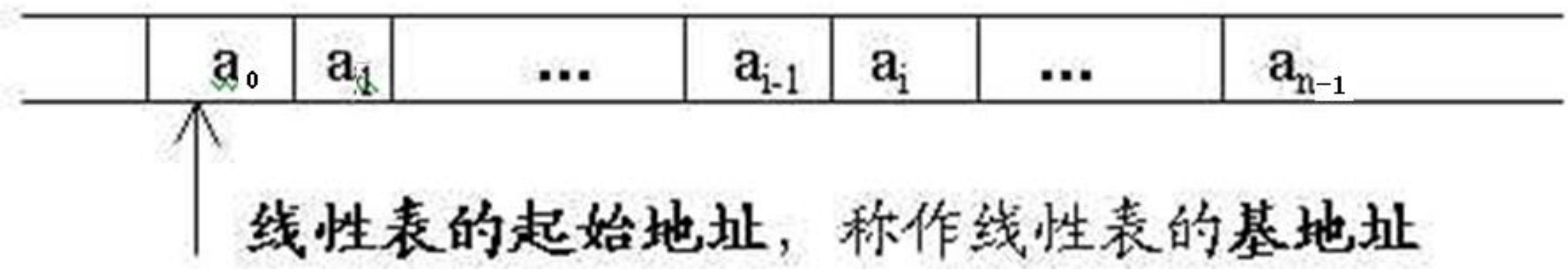


$Loc(a)=L_o+i*m$

Given some position in the list

Array-Based List Implementation

Array_Based List : The elements are stored in a consecutive storage area one by one



a_0 | a_1 | … | a_{i-1} | a_i | … | a_{n-1}

线性表的起始地址，称作线性表的基地址

Notes：

- With ordered pair <a_{i-1}， a_i> to express “Storage is adjacent to”， loc（a_i）=loc（a_{i-1}）+m
- Unnecessary to store logic relationship
- First data component location can decide all data elements locations

Array-Based List Class (1)

```
#include "list.h‟
template <typename E> // Array-based list implementation
class AList : public List<E> {
private:
   int maxSize;        // Maximum size of list
   int listSize;       // Number of list items now
   int curr;           // Position of current element
   E* listArray;       // Array holding list elements
```

Array-Based List Class (2)

```
  public:
    AList(int size=defaultSize)
{ // Constructor
    maxSize = size;
    listSize = curr = 0;
    listArray = new E[maxSize];
 }

~AList() { delete [] listArray; } // Destructor
```

Array-Based List Class (3)

- void clear() {   // Reinitialize the list
-     delete [] listArray;   // Remove the array
-     listSize = curr = 0;   // Reset the size
-     listArray = new E[maxSize]; // Recreate array
-   }
- void moveToStart() { curr = 0; }
- void moveToEnd() { curr = listSize; }
- void prev() { if (curr != 0) curr--; }
- void next()   { if (curr < listSize) curr++; }

Array-Based List Class (4)

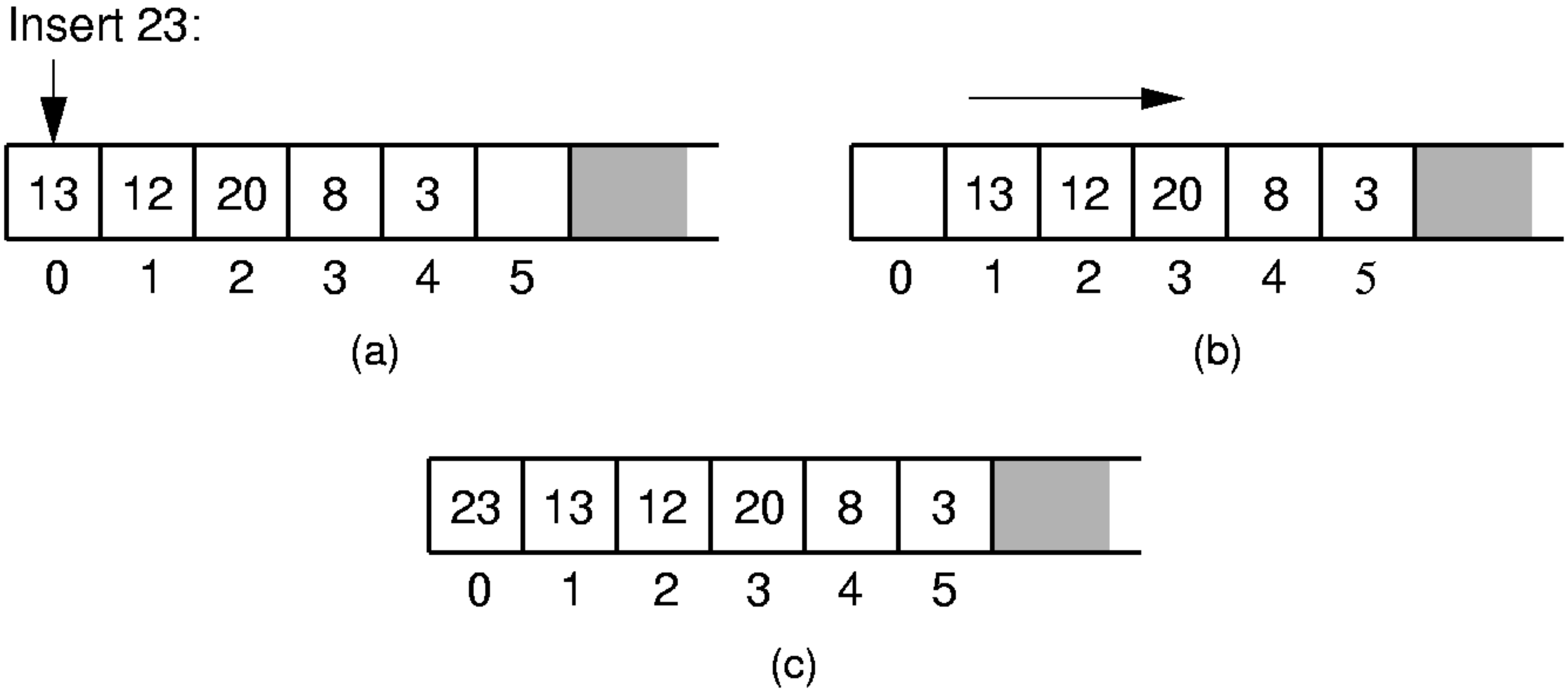
- // Return list size
- int length() const { return listSize; }
- // Return current position
- int currPos() const { return curr; }
- // Set current list position to "pos"
- void moveToPos(int pos) {
- Assert ((pos>=0)&&(pos<=listSize), "Pos out of range");
- curr = pos; }

Array-Based List Class (5)

- // Return current element

- const E& getValue() const
- { Assert((curr>=0)&&(curr<listSize),"No current element");
-     return listArray[curr];
- }

Array-Based List Insert


Insert 23:

(a)

(b)

(c)

Insert

```
// Insert "it" at current position
 void insert(const E& it) {
   Assert(listSize < maxSize, "List capacity exceeded");
   for(int i=listSize; i>curr; i--)  // Shift elements up
     listArray[i] = listArray[i-1];  //   to make room
   listArray[curr] = it;
   listSize++;                      // Increment list size
 }
```

Append

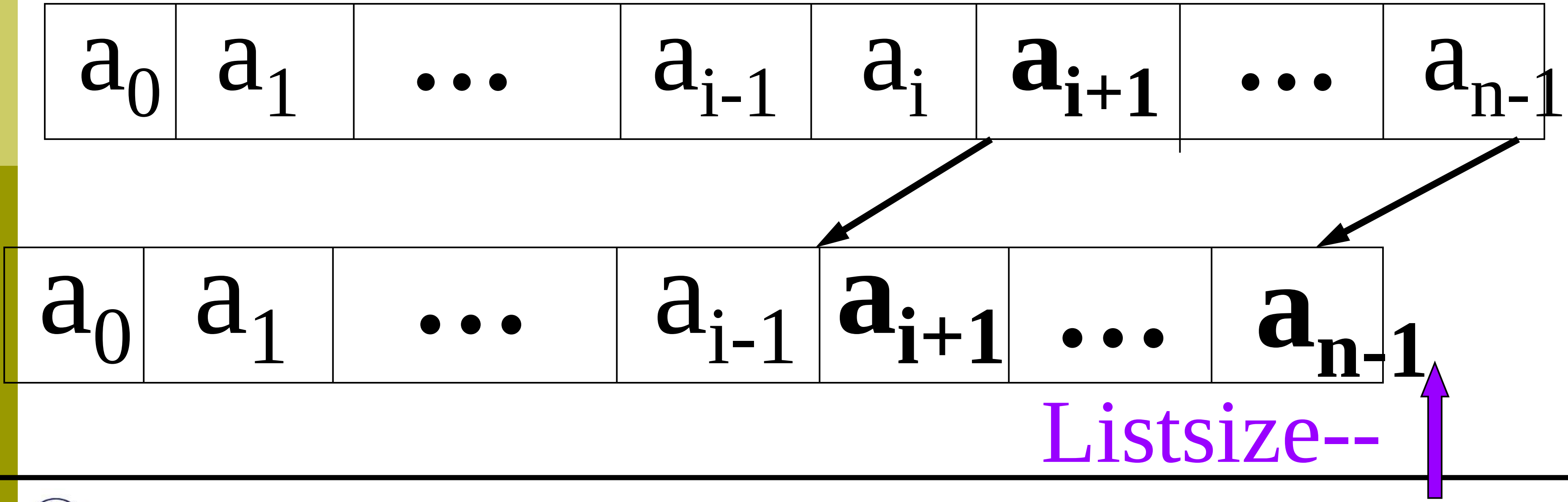
```
 void append(const E& it) {      // Append "it"
    Assert(listSize < maxSize, "List capacity exceeded");
    listArray[listSize++] = it;
 }
```

★Remove

$\langle a_0, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_{n-1}\rangle$ change to

$\langle a_0, \ldots, a_{i-1}, a_{i+1}, \ldots, a_{n-1}\rangle$

$\langle a_{i-1}, a_i\rangle, \langle a_i, a_{i+1}\rangle \Longrightarrow \langle a_{i-1}, a_{i+1}\rangle$



a_0	a_1	…	a_{i-1}	a_i	a_{i+1}	…	a_{n-1}

a_0	a_1	…	a_{i-1}	a_{i+1}	…	a_{n-1}

Listsize--

Remove

```
// Remove and return the current element.
 E remove() {
   Assert((curr>=0) && (curr < listSize), "No element");
   E it = listArray[curr];           // Copy the element
   for(int i=curr; i<listSize-1; i++)  // Shift them down
     listArray[i] = listArray[i+1];
   listSize--;                        // Decrement size
   return it;
 }
```

Summing Up

- Advantages　：
  - Stores a collection of items contiguously.
    - Stores no relations
    - Access randomly
- Disadvantages　：
  - Need to shift many elements in the array whenever there is an insertion or deletion.
  - Need to allocate a fix amount of memory in advance.

Exercise ：

- 1.To give an example to illustrate data structure idea and describe it in abstract data type form .
- 2.Analyses the time complexity of the following algorithms 。

```
1. i=1;                2. i=n;                3. x=y=1;
   while (s<n)           do {                    while(x++ * y++<n);
   { i++;s+=i;}            i++;
                          } while (i<n)
```

- 3.Design an Improve LocateElem's algorithm to look for all the elements matching the relationship 。
- 4.Design an algorithm to reverse an sequential list ($a_1a_2..a_n$)->($a_na_{n-1}...a_1$)

Reference

- **P95----P103**

Preview

- **Chapter 4, pp. 103--P112**

End

Thank you for listening!