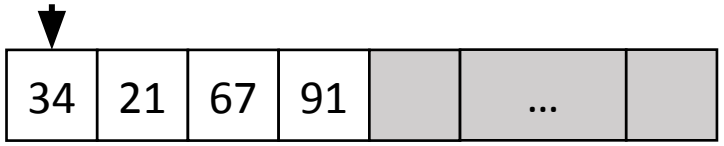
Array-Based Lists

Basic Data Structures

Array Based Lists

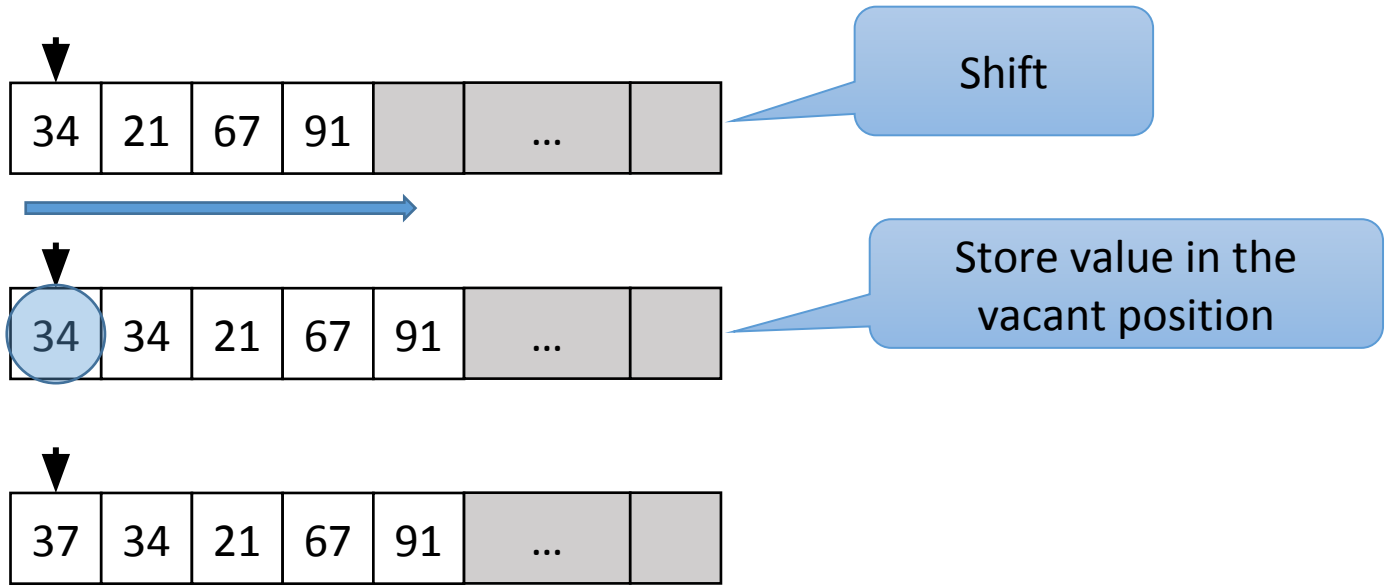
- Use an array to store the list elements



  - Array size is fixed – the list can become full!
  - To insert
    - Move all elements, from current position to the end, one position to the right
    - Store the value at current position
  - To remove:
    - Move all elements after current position one step to the left, overwriting the contents of current position

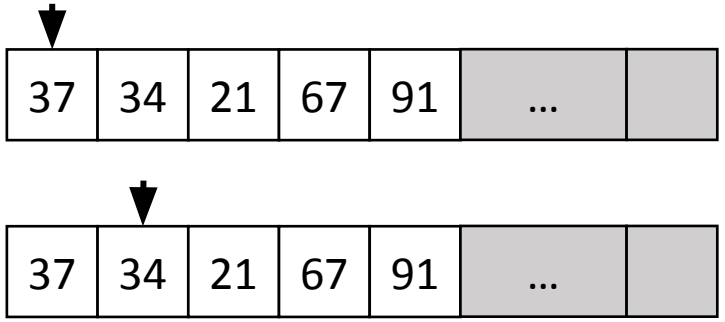
Example: Array Based List Operations

- insert(37)

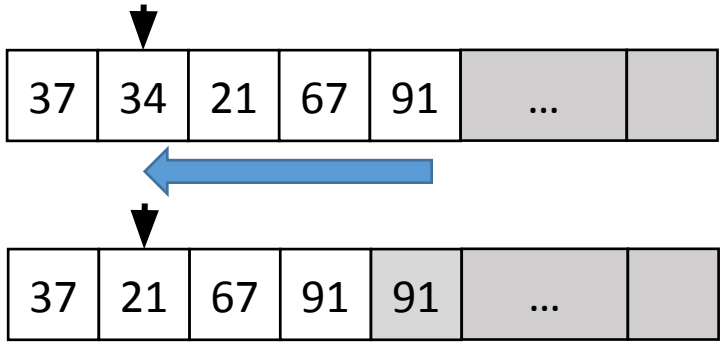


Example: Array-Based List Operations

- next()



- remove()



Java implementation of Array-Based List

- Use the List interface (ADT)
  - Define a class that implements the List interface
- See the provided Java code

```java
public class ArrayList<E> implements List<E> {
    private static int defaultMaxSize = 20;
    private int arraySize; // size of the array
    private int listSize; // number of elements in the list
    private int curr; // current position
    private E[] listArray; // array holding list elements

    public ArrayList(){ // constructor using defualt array size
        listArray = (E[]) new Object[defaultMaxSize];
        arraySize = defaultMaxSize;
        listSize = curr = 0;
    }

    public ArrayList(int maxListSize){ // constructor with array size
        listArray = (E[]) new Object[maxListSize];
        arraySize = maxListSize;
        listSize = curr = 0;
    }
```

```java
@Override
public void clear() {
    // curr must be set to 0 for an empty list because insertion is always done at
    // poition curr
    listSize = curr = 0;
}

@Override
public void insert(E item) {
    assert listSize < arraySize : "List is full";
    assert curr >= 0 && curr < listSize : "invalid list position";

    // Shift all elements 1 step to the right
    for (int i = listSize; i > curr; i--)
        listArray[i] = listArray[i - 1];

    listArray[curr] = item;
    listSize++; // update the size of the list
}
```

```java
@Override
public void append(E item) {
    assert listSize < arraySize : "List is full";
    // Store at 1 past the last list index and increment the list size
    listArray[listSize++] = item;
}
```

```java
@Override
public E remove() {
    assert listSize > 0 : "List is empty";
    assert curr >= 0 && curr < listSize : "Invalid list position";
    E removedItem = listArray[curr];

    // move all elements after curr 1 step to the left to overwite the deleted
    // element
    for (int i = curr; i < listSize - 1; i++)
        listArray[i] = listArray[i + 1];

    listSize--; // decrement list size, since 1 item has been removed
    return removedItem;
}

@Override
public void setFirst() {
    curr = 0;
}
```

```java
@Override
public void next() {
    curr++;
}

@Override
public void prev() {
    curr--;
}

@Override
public int length() {
    return listSize;
}

@Override
public void setPos(int pos) {
    curr = pos;
}
```

```java
@Override
public void setValue(E val) {
    assert curr >= 0 && curr < listSize : "Invalid list position";
    listArray[curr] = val;
}

@Override
public E currValue() {
    assert curr >= 0 && curr < listSize : "Invalid list position";
    return listArray[curr];
}

@Override
public boolean isEmpty() {
    return listSize == 0;
}
```

```java
@Override
public boolean isInList() {
    return curr >= 0 && curr < listSize;
}

@Override
public void print() {
    if (isEmpty()) {
        System.out.println("()");
    } else {
        System.out.print("(");

        for (int i = 0; i < listSize - 1; i++) {
            System.out.print(listArray[i] + ", ");
        }

        System.out.println(listArray[listSize -1] + ")");
    }
}
```