Data Structures and Algorithms

The List Abstract Data Type - Array-Based List

Dr. Lina Xu
lina.xu@ucd.ie

School of Computer Science,
University College Dublin

November 5, 2018

Learning outcomes

After this lecture and the related practical students should...

- be able to implement an array based list
- understand the resizing strategies

Table of Contents

The List Abstract Data Type

Concept

- The List ADT models a sequence of positions
- Each position store a piece of data
- There is a before/after relation between positions
- This allows for efficient insertion into the middle of a list

The Position Abstract Data Type

Concept

Before we can fully understand the list abstract data type we need to look at the position abstract data type

- The Position abstract data type models the idea of a place within a data structure where a single piece of data is stored
- Positions provide a simple view of different ways of storing data
  - An element in an array
  - A Node in a linked list

The Position Abstract Data Type

Specification

- Operation:
  - element(): This returns the piece of data that is stored in this position

The Position Abstract Data Type

Interface

```
1 public interface Position{
2   public int element();
3 }
```

The List Abstract Data Type

Specification

Operations:

- first(): returns the first position in the list

The List Abstract Data Type

Specification

Operations:

- first(): returns the first position in the list
- last(): return the last position in the list

The List Abstract Data Type

Specification

Operations:

- first(): returns the first position in the list
- last(): return the last position in the list
- before(p): returns the position in the list before p

The List Abstract Data Type

Specification

Operations:

- first(): returns the first position in the list
- last(): return the last position in the list
- before(p): returns the position in the list before p
- after(p) : returns the position in the list after p

The List Abstract Data Type

Specification

Operations:

- first(): returns the first position in the list
- last(): return the last position in the list
- before(p): returns the position in the list before p
- after(p) : returns the position in the list after p
- insertBefore(p, d): inserts the value d into the position in the list before p

The List Abstract Data Type

Specification

Operations:

- first(): returns the first position in the list
- last(): return the last position in the list
- before(p): returns the position in the list before p
- after(p) : returns the position in the list after p
- insertBefore(p, d): inserts the value d into the position in the list before p
- insertAfter(p, d): inserts the value d into the position in the list after p

The List Abstract Data Type

Specification

Operations:

- first(): returns the first position in the list
- last(): return the last position in the list
- before(p): returns the position in the list before p
- after(p) : returns the position in the list after p
- insertBefore(p, d): inserts the value d into the position in the list before p
- insertAfter(p, d): inserts the value d into the position in the list after p
- insertFirst(d): inserts the value d into the first position in the list

The List Abstract Data Type

Specification

Operations:

- first(): returns the first position in the list
- last(): return the last position in the list
- before(p): returns the position in the list before p
- after(p) : returns the position in the list after p
- insertBefore(p, d): inserts the value d into the position in the list before p
- insertAfter(p, d): inserts the value d into the position in the list after p
- insertFirst(d): inserts the value d into the first position in the list
- insertLast(d): inserts the value d into the last position in the list

The List Abstract Data Type

Specification

Operations:

- first(): returns the first position in the list
- last(): return the last position in the list
- before(p): returns the position in the list before p
- after(p) : returns the position in the list after p
- insertBefore(p, d): inserts the value d into the position in the list before p
- insertAfter(p, d): inserts the value d into the position in the list after p
- insertFirst(d): inserts the value d into the first position in the list
- insertLast(d): inserts the value d into the last position in the list
- remove(p): removes the position p from the list

The List Abstract Data Type

Specification

Operations:

- first(): returns the first position in the list
- last(): return the last position in the list
- before(p): returns the position in the list before p
- after(p) : returns the position in the list after p
- insertBefore(p, d): inserts the value d into the position in the list before p
- insertAfter(p, d): inserts the value d into the position in the list after p
- insertFirst(d): inserts the value d into the first position in the list
- insertLast(d): inserts the value d into the last position in the list
- remove(p): removes the position p from the list
- size(): returns the number of elements stored in the list

The List Abstract Data Type

Specification

Operations:

- first(): returns the first position in the list
- last(): return the last position in the list
- before(p): returns the position in the list before p
- after(p) : returns the position in the list after p
- insertBefore(p, d): inserts the value d into the position in the list before p
- insertAfter(p, d): inserts the value d into the position in the list after p
- insertFirst(d): inserts the value d into the first position in the list
- insertLast(d): inserts the value d into the last position in the list
- remove(p): removes the position p from the list
- size(): returns the number of elements stored in the list
- isEmpty(): is the list empty?

The List Abstract Data Type

Interface

```
1  public interface List {
2    public Position first();
3    public Position last();
4    public Position before(Position p);
5    public Position after(Position p);
6    public Position insertBefore(Position p,
      int d);
7    public Position insertAfter(Position p, int
      d);
8    public Position insertFirst(int d);
9    public Position insertLast(int d);
10   public int remove(Position p);
11   public int size();
12   public boolean isEmpty();
13 }
```

The List Abstract Data Type

Implementation Strategies

- Array based implementation

The List Abstract Data Type

Implementation Strategies

- Array based implementation
  - An array of `Position` objects

The List Abstract Data Type

Implementation Strategies

- Array based implementation
  - An array of `Position` objects
  - Search through the array to find the correct position

The List Abstract Data Type

Implementation Strategies

- Array based implementation
  - An array of `Position` objects
  - Search through the array to find the correct position
  - Not very efficient

The List Abstract Data Type

Implementation Strategies

- Array based implementation
  - An array of `Position` objects
  - Search through the array to find the correct position
  - Not very efficient
- Link based implementations

The List Abstract Data Type

Implementation Strategies

- Array based implementation
  - An array of `Position` objects
  - Search through the array to find the correct position
  - Not very efficient
- Link based implementations
- There are two versions

The List Abstract Data Type

Implementation Strategies

- Array based implementation
  - An array of `Position` objects
  - Search through the array to find the correct position
  - Not very efficient
- Link based implementations
- There are two versions
  - Singly-Linked List

The List Abstract Data Type

Implementation Strategies

- Array based implementation
  - An array of `Position` objects
  - Search through the array to find the correct position
  - Not very efficient
- Link based implementations
- There are two versions
  - Singly-Linked List
    - ⋆ Each `Position` object keeps a reference to the next `Position` in the sequence

The List Abstract Data Type

Implementation Strategies

- Array based implementation
  - An array of `Position` objects
  - Search through the array to find the correct position
  - Not very efficient
- Link based implementations
- There are two versions
  - Singly-Linked List
    - ⋆ Each `Position` object keeps a reference to the next `Position` in the sequence
  - Doubly-Linked List

The List Abstract Data Type

Implementation Strategies

- Array based implementation
  - An array of `Position` objects
  - Search through the array to find the correct position
  - Not very efficient
- Link based implementations
- There are two versions
  - Singly-Linked List
    - ⋆ Each `Position` object keeps a reference to the next `Position` in the sequence
  - Doubly-Linked List
    - ⋆ Each `Position` object keeps a reference to the next and previous `Positions` in the sequence

Table of Contents

Position Abstract Data Type

Array-Based Implementation

- We create a `ArrPos` class that implements the `Position` interface

Position Abstract Data Type

Array-Based Implementation

- We create a `ArrPos` class that implements the `Position` interface
- We add functionality to the class to store the index of the ArrPos in the array

Position Abstract Data Type

Array-Based Implementation

- We create a `ArrPos` class that implements the `Position` interface
- We add functionality to the class to store the index of the ArrPos in the array
- We also add some functionality to increment and decrement the index

Position Abstract Data Type

Array-Based Implementation

```
1  public class ArrPos implements Position {
2    private int index;
3    private int data;
4    public ArrPos(int i, int d){
5      index = i;
6      data = d;
7    }
8    public int element() { return data; }
9    public int getIndex(){ return index; }
10   public void incrementPosition() {
11     index++;
12   }
13   public void decrementPosition() {
14     index--;
15   }
16 }
```

Array-Based List Implementation

- We keep an array of `ArrPos` objects to store our data

Array-Based List Implementation

- We keep an array of `ArrPos` objects to store our data
- When inserting or removing data we have to update the `ArrPos` objects so they point to the correct index

Array-Based List Implementation

- We keep an array of `ArrPos` objects to store our data
- When inserting or removing data we have to update the `ArrPos` objects so they point to the correct index
- When we run out of space we must increase the size of the array of objects

Array-Based List Implementation

- We keep an array of `ArrPos` objects to store our data
- When inserting or removing data we have to update the `ArrPos` objects so they point to the correct index
- When we run out of space we must increase the size of the array of objects

Variables:

Array-Based List Implementation

- We keep an array of `ArrPos` objects to store our data
- When inserting or removing data we have to update the `ArrPos` objects so they point to the correct index
- When we run out of space we must increase the size of the array of objects

Variables:

- A reference to the array of `ArrPos` objects

Array-Based List Implementation

- We keep an array of `ArrPos` objects to store our data
- When inserting or removing data we have to update the `ArrPos` objects so they point to the correct index
- When we run out of space we must increase the size of the array of objects

Variables:

- A reference to the array of `ArrPos` objects

```
private ArrPos[] items;
```

Array-Based List Implementation

- We keep an array of `ArrPos` objects to store our data
- When inserting or removing data we have to update the `ArrPos` objects so they point to the correct index
- When we run out of space we must increase the size of the array of objects

Variables:

- A reference to the array of `ArrPos` objects

```
private ArrPos[] items;
```

- A number to keep track of the size

Array-Based List Implementation

- We keep an array of `ArrPos` objects to store our data
- When inserting or removing data we have to update the `ArrPos` objects so they point to the correct index
- When we run out of space we must increase the size of the array of objects

Variables:

- A reference to the array of `ArrPos` objects

```
private ArrPos[] items;
```

- A number to keep track of the size

```
private int size;
```

Array-Based List Implementation

- We keep an array of `ArrPos` objects to store our data
- When inserting or removing data we have to update the `ArrPos` objects so they point to the correct index
- When we run out of space we must increase the size of the array of objects

Variables:

- A reference to the array of `ArrPos` objects

```
private ArrPos[] items;
```

- A number to keep track of the size

```
private int size;
```

- A number to keep track of the maximum number of elements we can hold

Array-Based List Implementation

- We keep an array of `ArrPos` objects to store our data
- When inserting or removing data we have to update the `ArrPos` objects so they point to the correct index
- When we run out of space we must increase the size of the array of objects

Variables:

- A reference to the array of `ArrPos` objects

```
private ArrPos[] items;
```

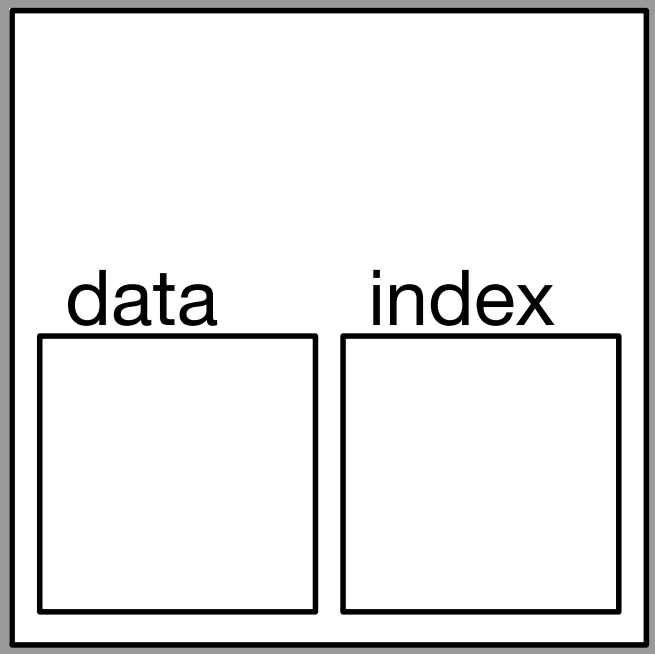
- A number to keep track of the size

```
private int size;
```

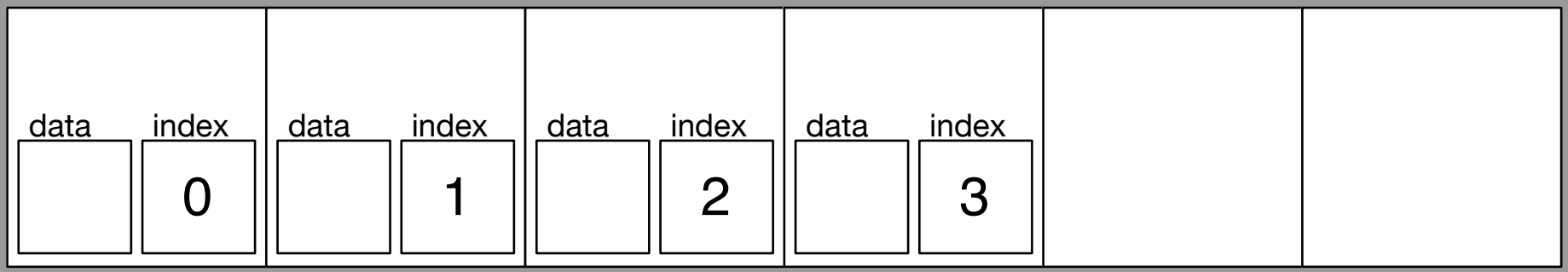
- A number to keep track of the maximum number of elements we can hold

```
private int maxSize;
```

Representation of am ArrPos Object



Representation of a List



Array-Based List Operations

- first()

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()
  - Return the reference that is stored in index `size - 1` of the array `items`

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()
  - Return the reference that is stored in index `size - 1` of the array `items`
- size()

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()
  - Return the reference that is stored in index `size - 1` of the array `items`
- size()
  - Return the value of the `size` variable

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()
  - Return the reference that is stored in index `size - 1` of the array `items`
- size()
  - Return the value of the `size` variable
- isEmpty()

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()
  - Return the reference that is stored in index `size - 1` of the array `items`
- size()
  - Return the value of the `size` variable
- isEmpty()
  - Return the result of the expression `size == 0`

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()
  - Return the reference that is stored in index `size - 1` of the array `items`
- size()
  - Return the value of the `size` variable
- isEmpty()
  - Return the result of the expression `size == 0`
- after(p)

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()
  - Return the reference that is stored in index `size - 1` of the array `items`
- size()
  - Return the value of the `size` variable
- isEmpty()
  - Return the result of the expression `size == 0`
- after(p)
  - Convert `p` to a `ArrPos` object

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()
  - Return the reference that is stored in index `size - 1` of the array `items`
- size()
  - Return the value of the `size` variable
- isEmpty()
  - Return the result of the expression `size == 0`
- after(p)
  - Convert `p` to a `ArrPos` object
  - Access the index stored in `p`

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()
  - Return the reference that is stored in index `size - 1` of the array `items`
- size()
  - Return the value of the `size` variable
- isEmpty()
  - Return the result of the expression `size == 0`
- after(p)
  - Convert `p` to a `ArrPos` object
  - Access the index stored in `p`
  - Return the `ArrPos` object stored in index $+ 1$ in the array `items`

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()
  - Return the reference that is stored in index `size - 1` of the array `items`
- size()
  - Return the value of the `size` variable
- isEmpty()
  - Return the result of the expression `size == 0`
- after(p)
  - Convert `p` to a `ArrPos` object
  - Access the index stored in `p`
  - Return the `ArrPos` object stored in index + 1 in the array `items`
- before(p)

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()
  - Return the reference that is stored in index `size - 1` of the array `items`
- size()
  - Return the value of the `size` variable
- isEmpty()
  - Return the result of the expression `size == 0`
- after(p)
  - Convert `p` to a `ArrPos` object
  - Access the index stored in `p`
  - Return the `ArrPos` object stored in index $+ 1$ in the array `items`
- before(p)
  - Convert `p` to a `ArrPos` object

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()
  - Return the reference that is stored in index `size - 1` of the array `items`
- size()
  - Return the value of the `size` variable
- isEmpty()
  - Return the result of the expression `size == 0`
- after(p)
  - Convert `p` to a `ArrPos` object
  - Access the index stored in `p`
  - Return the `ArrPos` object stored in index + 1 in the array `items`
- before(p)
  - Convert `p` to a `ArrPos` object
  - Access the index stored in `p`

Array-Based List Operations

- first()
  - Return the reference that is stored in index 0 of the array `items`
- last()
  - Return the reference that is stored in index `size - 1` of the array `items`
- size()
  - Return the value of the `size` variable
- isEmpty()
  - Return the result of the expression `size == 0`
- after(p)
  - Convert `p` to a `ArrPos` object
  - Access the index stored in `p`
  - Return the `ArrPos` object stored in index + 1 in the array `items`
- before(p)
  - Convert `p` to a `ArrPos` object
  - Access the index stored in `p`
  - Return the `ArrPos` object stored in index - 1 in the array `items`

Array-Based List

Array Size

- When we implemented this using the links, the list had an infinite capacity

Array-Based List

Array Size

- When we implemented this using the links, the list had an infinite capacity
- However, arrays cannot grow infinitely

Array-Based List

Array Size

- When we implemented this using the links, the list had an infinite capacity
- However, arrays cannot grow infinitely
- Therefore we must replace the array every time that it runs out of room

Array-Based List

Array Size

- When we implemented this using the links, the list had an infinite capacity
- However, arrays cannot grow infinitely
- Therefore we must replace the array every time that it runs out of room
- There are many strategies for resizing the array, we will use this one

Array-Based List

Array Size

- When we implemented this using the links, the list had an infinite capacity
- However, arrays cannot grow infinitely
- Therefore we must replace the array every time that it runs out of room
- There are many strategies for resizing the array, we will use this one
  - Every time we try to insert and there is no space we create a new array which is double the size

Array-Based List

Array Size

- When we implemented this using the links, the list had an infinite capacity
- However, arrays cannot grow infinitely
- Therefore we must replace the array every time that it runs out of room
- There are many strategies for resizing the array, we will use this one
  - Every time we try to insert and there is no space we create a new array which is double the size
  - Every time we remove a value, if we are using less than 25% of the array we we create a new array which is half the size

Array-Based List

Increasing Array Size

- Once an array is created, it is impossible for the size to be changed

Array-Based List

Increasing Array Size

- Once an array is created, it is impossible for the size to be changed
- However, we can create a new array and change the references, so that we point to this instead

Array-Based List

Increasing Array Size

- Once an array is created, it is impossible for the size to be changed
- However, we can create a new array and change the references, so that we point to this instead
- This means that we only need to copy the data contained in the array and we are finished

Array-Based List

Increasing Array Size

- Once an array is created, it is impossible for the size to be changed
- However, we can create a new array and change the references, so that we point to this instead
- This means that we only need to copy the data contained in the array and we are finished
- We might want to use this in multiple locations, so we will implement it as a private method in the class

Array-Based List

Grow Method

- The steps we must follow are:

Array-Based List

Grow Method

- The steps we must follow are:
  - Create a new array that is twice the previous maximum size

Array-Based List

Grow Method

- The steps we must follow are:
  - Create a new array that is twice the previous maximum size
  - Copy the references of all objects to the new array

Array-Based List

Grow Method

- The steps we must follow are:
  - Create a new array that is twice the previous maximum size
  - Copy the references of all objects to the new array
  - Multiply the value of `maxSize` by 2

Array-Based List

Grow Method

- The steps we must follow are:
  - Create a new array that is twice the previous maximum size
  - Copy the references of all objects to the new array
  - Multiply the value of `maxSize` by 2
  - Copy the reference of the new array into the variable `items`

Array-Based List

Grow Method

- The steps we must follow are:
  - Create a new array that is twice the previous maximum size
  - Copy the references of all objects to the new array
  - Multiply the value of `maxSize` by 2
  - Copy the reference of the new array into the variable `items`

Array-Based List

Grow Method

- The steps we must follow are:
  - Create a new array that is twice the previous maximum size
  - Copy the references of all objects to the new array
  - Multiply the value of `maxSize` by 2
  - Copy the reference of the new array into the variable `items`

```
1 private void grow(){
2   ArrPos[] a = new ArrPos[maxSize*2];
3   for (int i = 0; i < items.length; i++) {
4     a[i] = items[i];
5   }
6   maxSize = maxSize * 2;
7   items = a;
8 }
```

Array-Based List

Shrink Method

- Similar to the grow method we will also write one for shrinking, it has the following steps

Array-Based List

Shrink Method

- Similar to the grow method we will also write one for shrinking, it has the following steps
  - Create a new array that is half the previous maximum size

Array-Based List

Shrink Method

- Similar to the grow method we will also write one for shrinking, it has the following steps
  - Create a new array that is half the previous maximum size
  - Copy the references of all objects to the new array

Array-Based List

Shrink Method

- Similar to the grow method we will also write one for shrinking, it has the following steps
  - Create a new array that is half the previous maximum size
  - Copy the references of all objects to the new array
  - Divide the value of `maxSize` by 2

Array-Based List

Shrink Method

- Similar to the grow method we will also write one for shrinking, it has the following steps
  - Create a new array that is half the previous maximum size
  - Copy the references of all objects to the new array
  - Divide the value of `maxSize` by 2
  - Copy the reference of the new array into the variable `items`

Array-Based List

Shrink Method

- Similar to the grow method we will also write one for shrinking, it has the following steps
  - Create a new array that is half the previous maximum size
  - Copy the references of all objects to the new array
  - Divide the value of `maxSize` by 2
  - Copy the reference of the new array into the variable `items`

Array-Based List

Shrink Method

- Similar to the grow method we will also write one for shrinking, it has the following steps
  - Create a new array that is half the previous maximum size
  - Copy the references of all objects to the new array
  - Divide the value of `maxSize` by 2
  - Copy the reference of the new array into the variable `items`

```
1 private void shrink(){
2   ArrPos[] a = new ArrPos[maxSize/2];
3   for (int i = 0; i < size; i++) {
4     a[i] = items[i];
5   }
6   maxSize = maxSize / 2;
7   items = a;
8 }
```

Array-Based List Operations

insertFirst(d)

- Insert the value into the first position

Array-Based List Operations

insertFirst(d)

- Insert the value into the first position
  - Construct a new `ArrPos` object, called n, containing the value and the index 0

Array-Based List Operations

insertFirst(d)

- Insert the value into the first position
  - Construct a new `ArrPos` object, called n, containing the value and the index 0
  - Copy all ArrPos objects into the position next in the array and increment the index in each

Array-Based List Operations

insertFirst(d)

- Insert the value into the first position
  - Construct a new `ArrPos` object, called n, containing the value and the index 0
  - Copy all ArrPos objects into the position next in the array and increment the index in each
  - Copy the reference of n to index 0 in the array

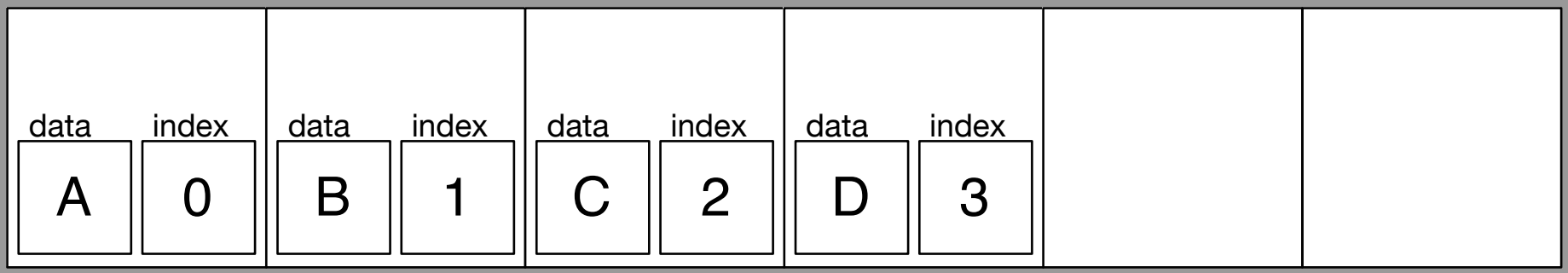
Array-Based List Operations

insertFirst(d)

- Insert the value into the first position
  - Construct a new `ArrPos` object, called n, containing the value and the index 0
  - Copy all ArrPos objects into the position next in the array and increment the index in each
  - Copy the reference of n to index 0 in the array
  - Increment the size

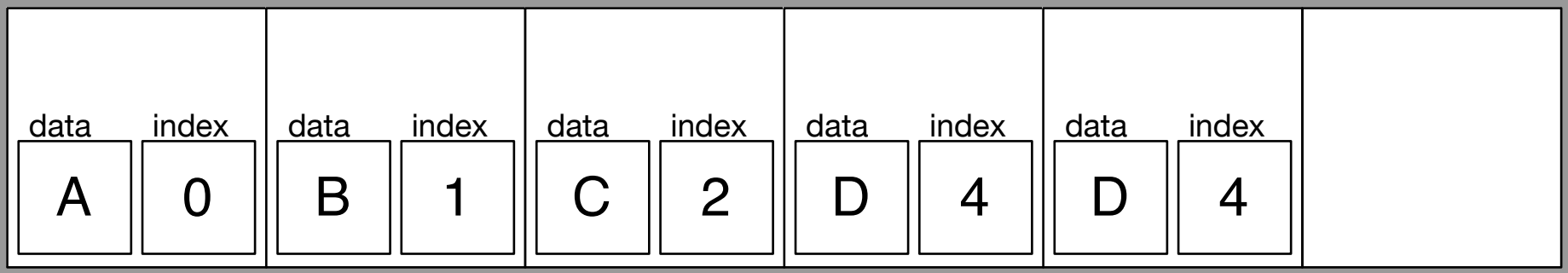
Array-Based List Operations

insertFirst(d)



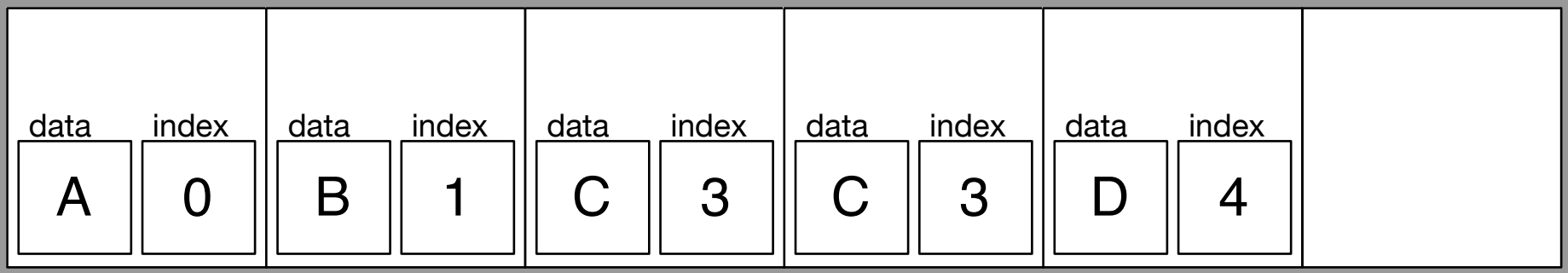
Array-Based List Operations

insertFirst(d)



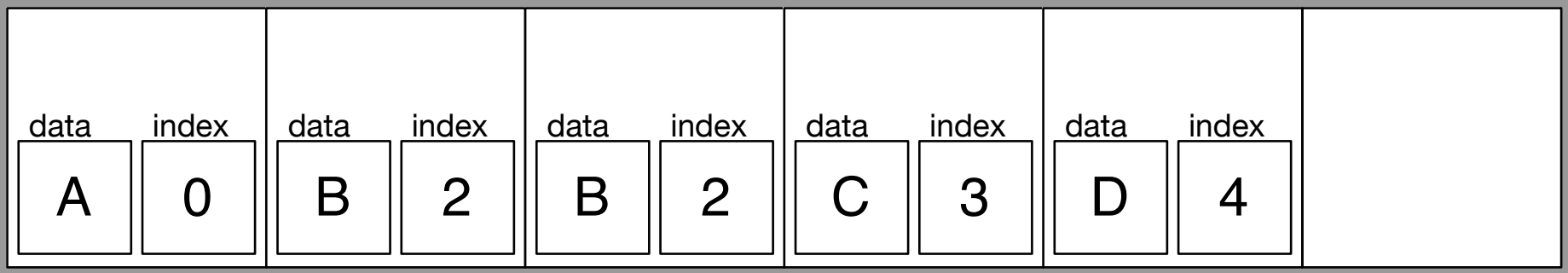
Array-Based List Operations

insertFirst(d)



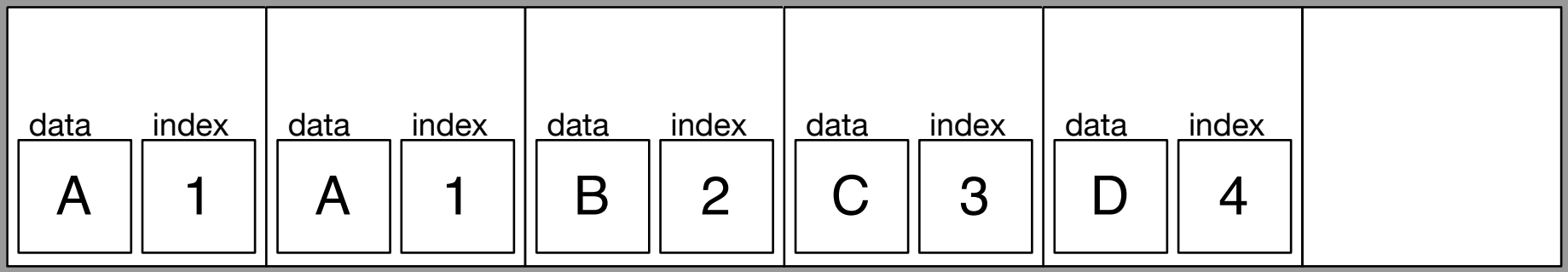
Array-Based List Operations

insertFirst(d)



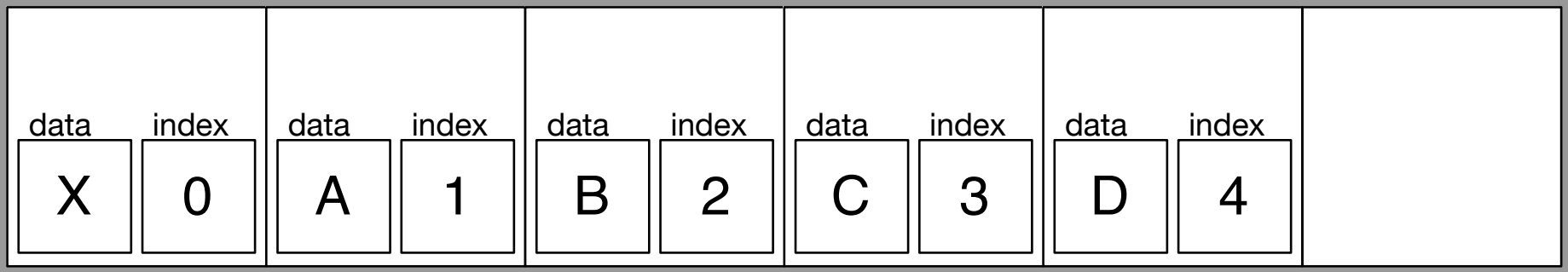
Array-Based List Operations

insertFirst(d)



Array-Based List Operations

insertFirst(d)



Array-Based List Operations

insertFirst(d)

```
1  Algorithm insertFirst (v):
2    Input: The value to be inserted
3    Output: The position it was inserted in
4
5  if size = maxSize then
6    grow()
7  Create ArrPos n containing v with index 0
8  for every integer value i in the range size
       to 1 (inclusive) do
9      items[i] ← items[i - 1]
10     items[i].incrementIndex()
11 items[0] ← n
12 size ← size + 1
13 return n
```

Array-Based List Operations

insertLast(d)

- Insert the value into the last position

Array-Based List Operations

insertLast(d)

- Insert the value into the last position
  - Construct a new `ArrPos` object, called `n`, containing the value and the index `size`

Array-Based List Operations

insertLast(d)

- Insert the value into the last position
  - Construct a new `ArrPos` object, called n, containing the value and the index `size`
  - Copy the reference of n to index size in the array

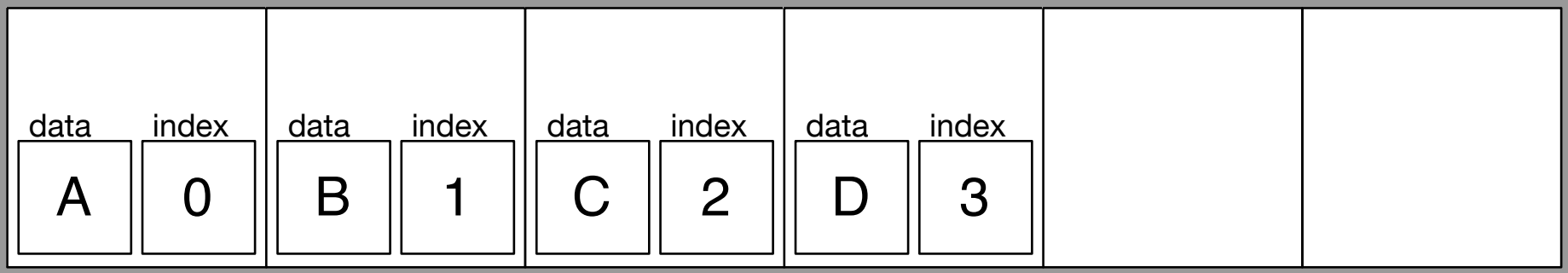
Array-Based List Operations

insertLast(d)

- Insert the value into the last position
  - Construct a new `ArrPos` object, called n, containing the value and the index `size`
  - Copy the reference of n to index size in the array
  - Increment the size

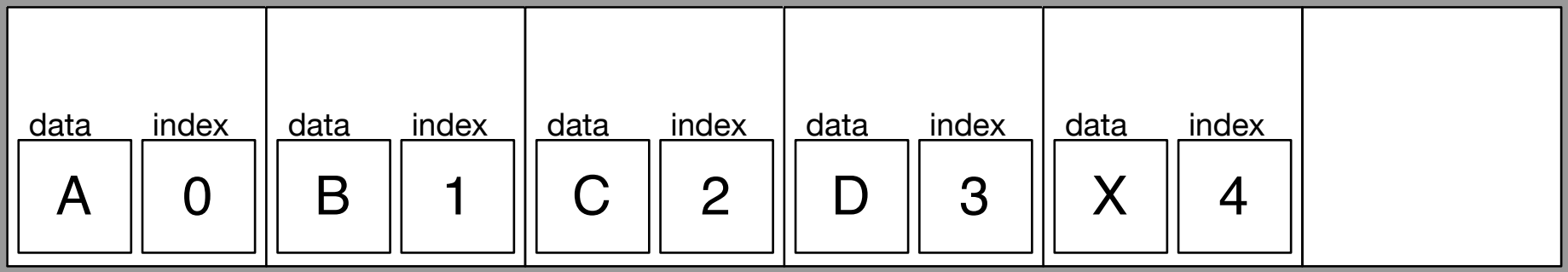
Array-Based List Operations

insertLast(d)



Array-Based List Operations

insertLast(d)



Array-Based List Operations

insertLast(d)

```
1  Algorithm insertLast (v):
2    Input: The value to be inserted
3    Output: The position it was inserted in
4
5  if size = maxSize then
6    grow()
7  Create ArrPos n containing v with index size
8  items[size] ← n
9  size ← size + 1
10 return n
```

Array-Based List Operations

insertAfter(p, d)

- Insert the value into the position after p

Array-Based List Operations

insertAfter(p, d)

- Insert the value into the position after p
  - Convert p to an `ArrPos` object

Array-Based List Operations

insertAfter(p, d)

- Insert the value into the position after p
  - Convert p to an `ArrPos` object
  - Copy the index from `p` to a variable called `n`

Array-Based List Operations

insertAfter(p, d)

- Insert the value into the position after p
  - Convert p to an `ArrPos` object
  - Copy the index from `p` to a variable called `n`
  - Copy all `ArrPos` objects in the array after p one index down and increment their index

Array-Based List Operations

insertAfter(p, d)

- Insert the value into the position after p
  - Convert p to an `ArrPos` object
  - Copy the index from `p` to a variable called `n`
  - Copy all `ArrPos` objects in the array after p one index down and increment their index
  - Create a new `ArrPos` object containing the data and the index `n + 1`

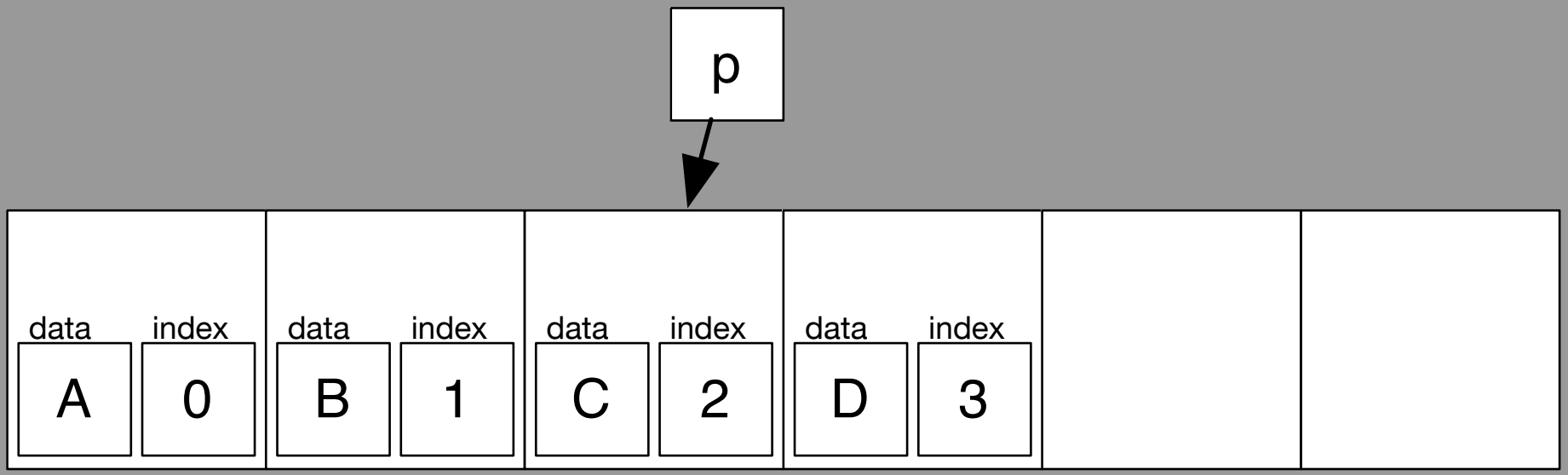
Array-Based List Operations

insertAfter(p, d)

- Insert the value into the position after p
  - Convert p to an `ArrPos` object
  - Copy the index from `p` to a variable called `n`
  - Copy all `ArrPos` objects in the array after p one index down and increment their index
  - Create a new `ArrPos` object containing the data and the index `n + 1`
  - Increment the size

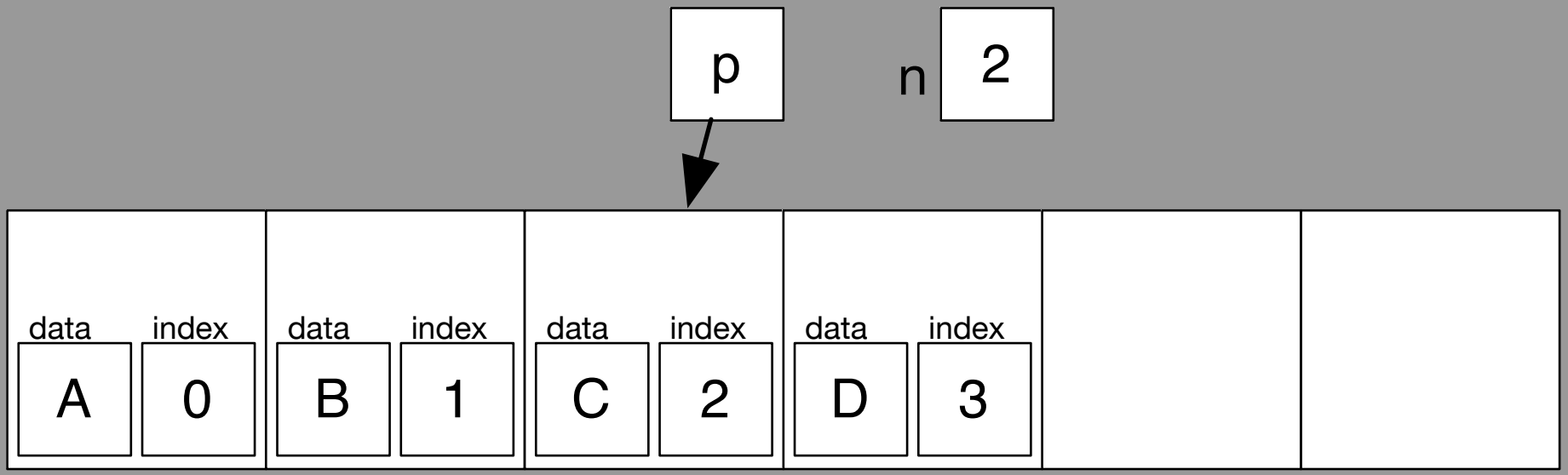
Array-Based List Operations

insertAfter(p, d)



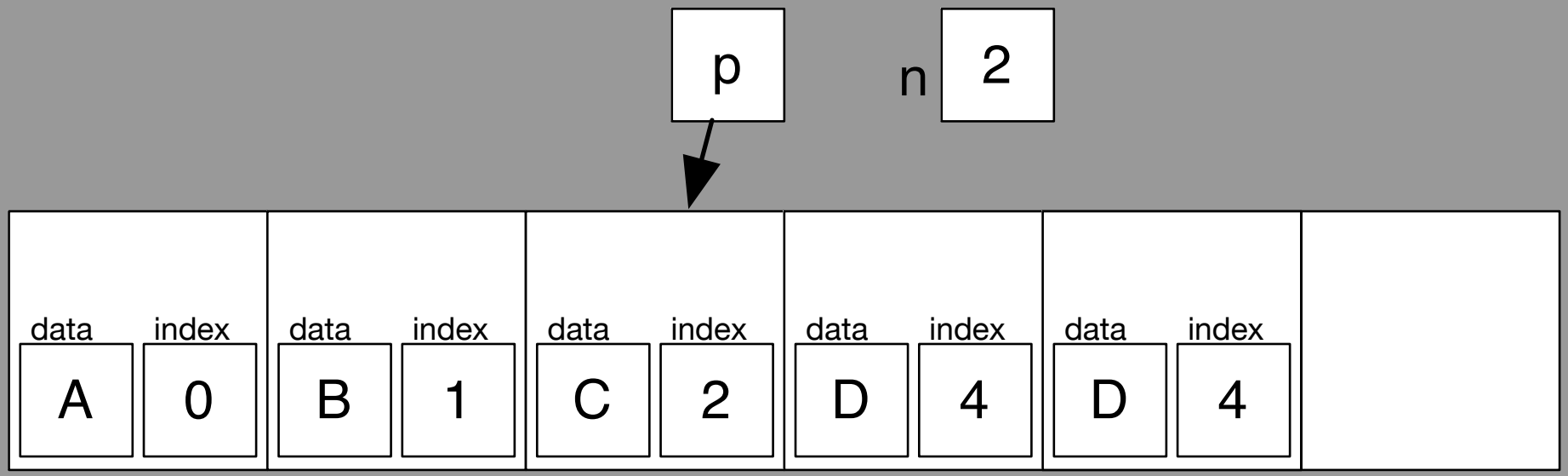
Array-Based List Operations

insertAfter(p, d)



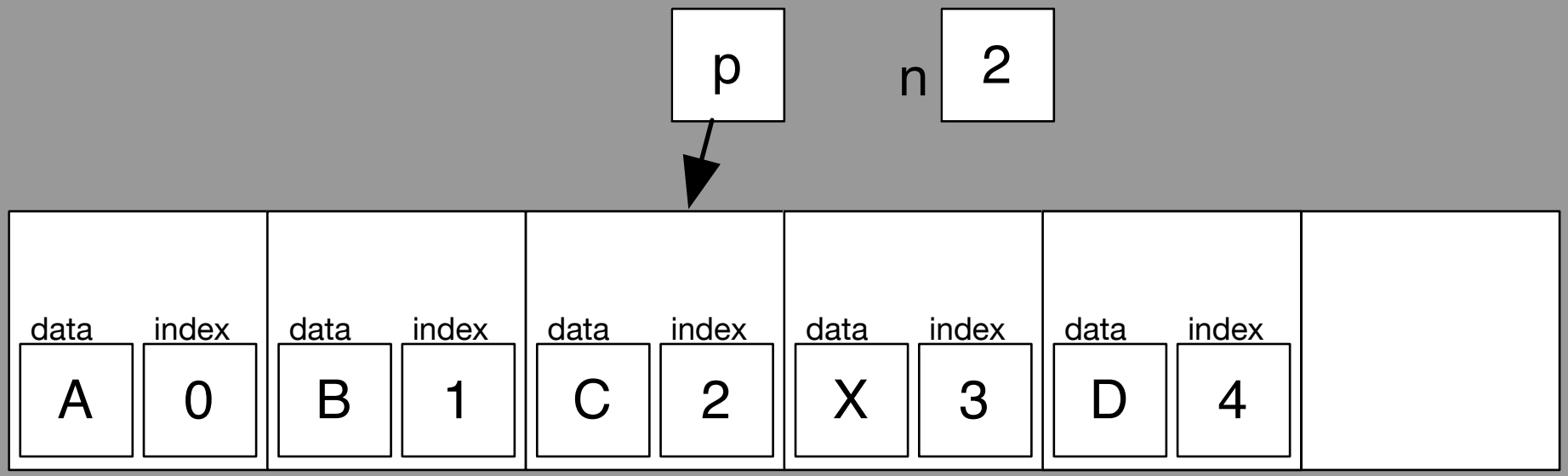
Array-Based List Operations

insertAfter(p, d)



Array-Based List Operations

insertAfter(p, d)



Array-Based List Operations

insertAfter(p, d)

```
1  Algorithm insertAfter (p, v):
2    Input: The value to be inserted and the
       position it should be inserted after
3    Output: The position it was inserted in
4  if size = maxSize then
5    grow()
6  n ← p.getIndex()
7  for every integer value i in the range size
       to n + 1 do
8      items[i] ← items[i - 1]
9      items[i].incrementIndex()
10 Create new ArrPos object, called a,
       containing data and index n+1
11 items[n+1] ← a
12 size ← size + 1
13 return a
```

Array-Based List Operations

insertBefore(p, d)

- Insert the value into the position before p

Array-Based List Operations

insertBefore(p, d)

- Insert the value into the position before p
  - Convert p to an `ArrPos` object

Array-Based List Operations

insertBefore(p, d)

- Insert the value into the position before p
  - Convert p to an `ArrPos` object
  - Copy the index from `p` to a variable called `n`

Array-Based List Operations

insertBefore(p, d)

- Insert the value into the position before p
  - Convert p to an `ArrPos` object
  - Copy the index from `p` to a variable called `n`
  - Copy all `ArrPos` objects in the array after and including p one index down and increment their index

Array-Based List Operations

insertBefore(p, d)

- Insert the value into the position before p
  - Convert p to an `ArrPos` object
  - Copy the index from `p` to a variable called `n`
  - Copy all `ArrPos` objects in the array after and including p one index down and increment their index
  - Create a new `ArrPos` object containing the data and the index `n`

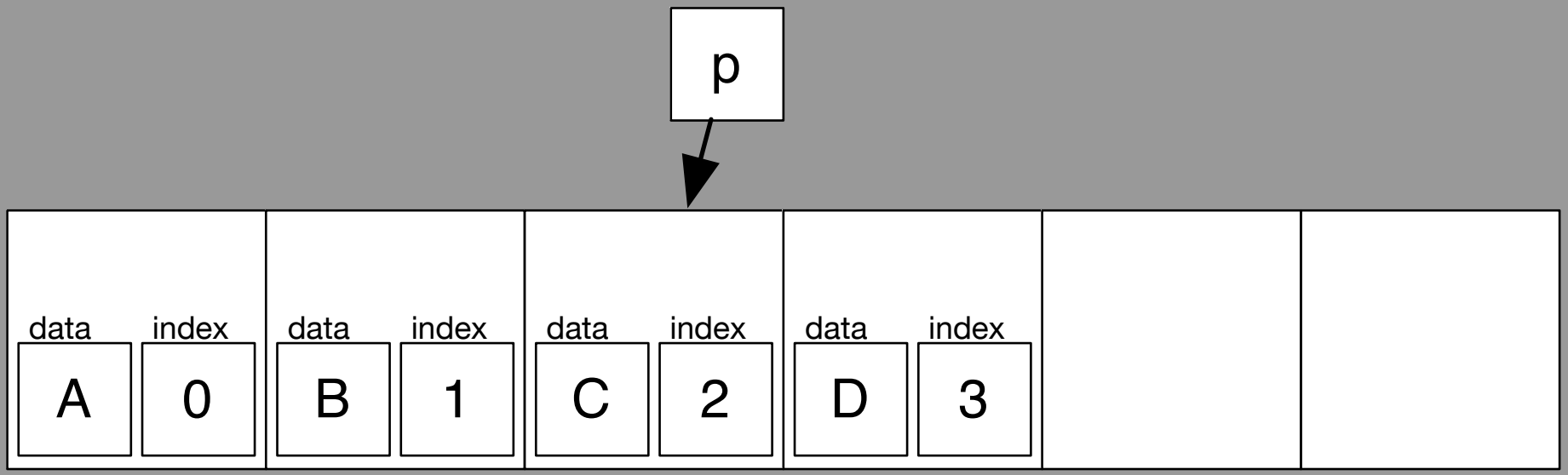
Array-Based List Operations

insertBefore(p, d)

- Insert the value into the position before p
  - Convert p to an `ArrPos` object
  - Copy the index from `p` to a variable called `n`
  - Copy all `ArrPos` objects in the array after and including p one index down and increment their index
  - Create a new `ArrPos` object containing the data and the index `n`
  - Increment the size

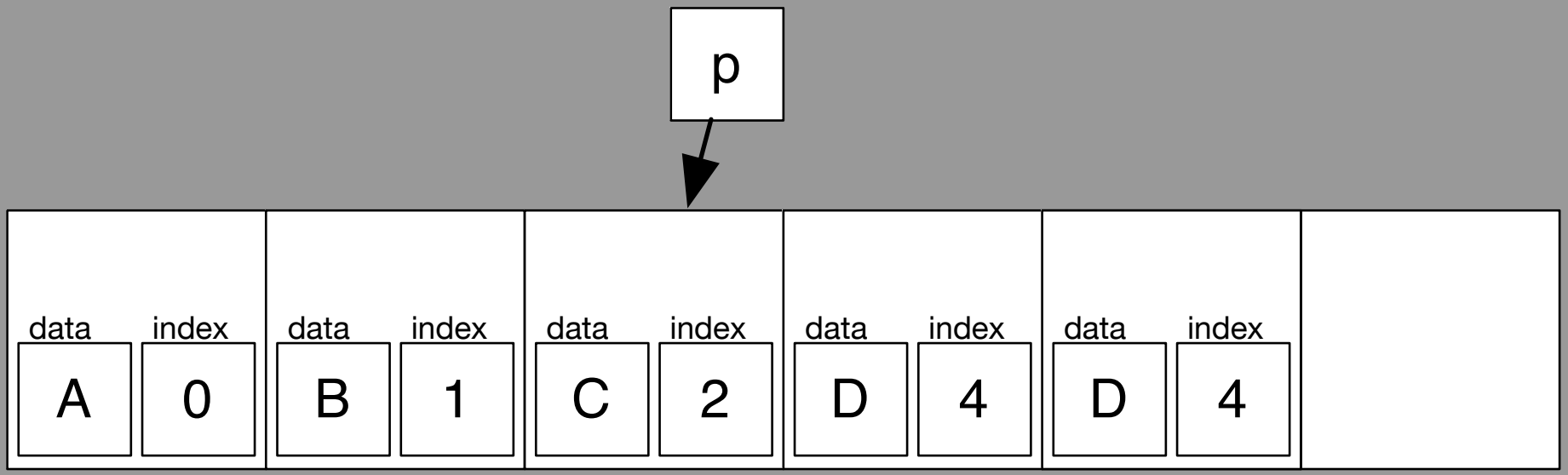
Array-Based List Operations

insertBefore(p, d)



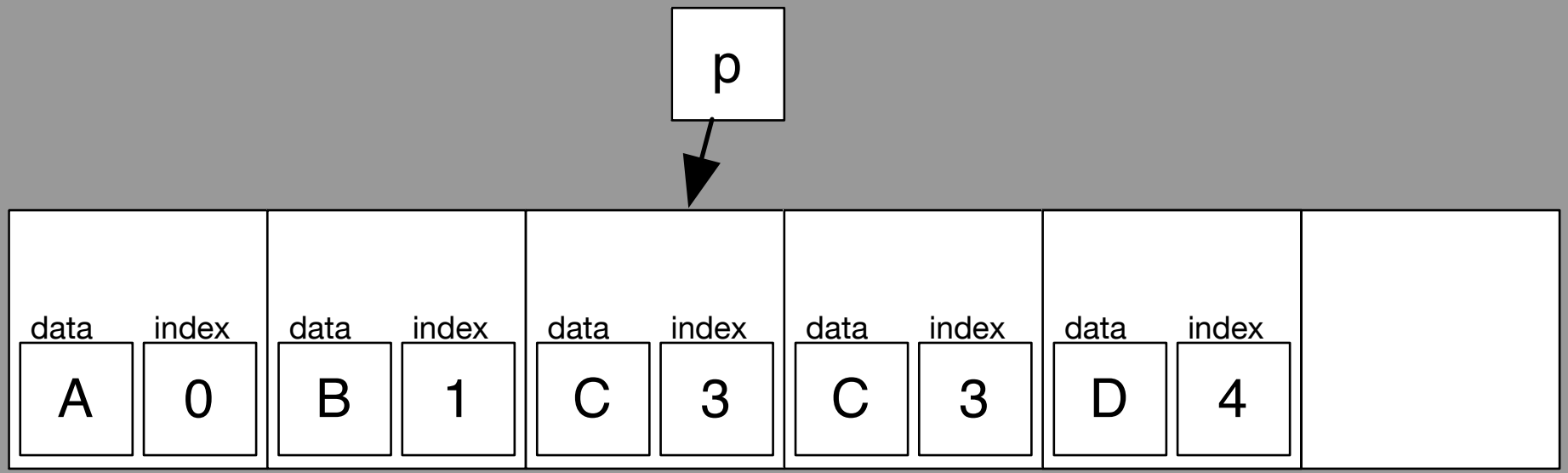
Array-Based List Operations

insertBefore(p, d)



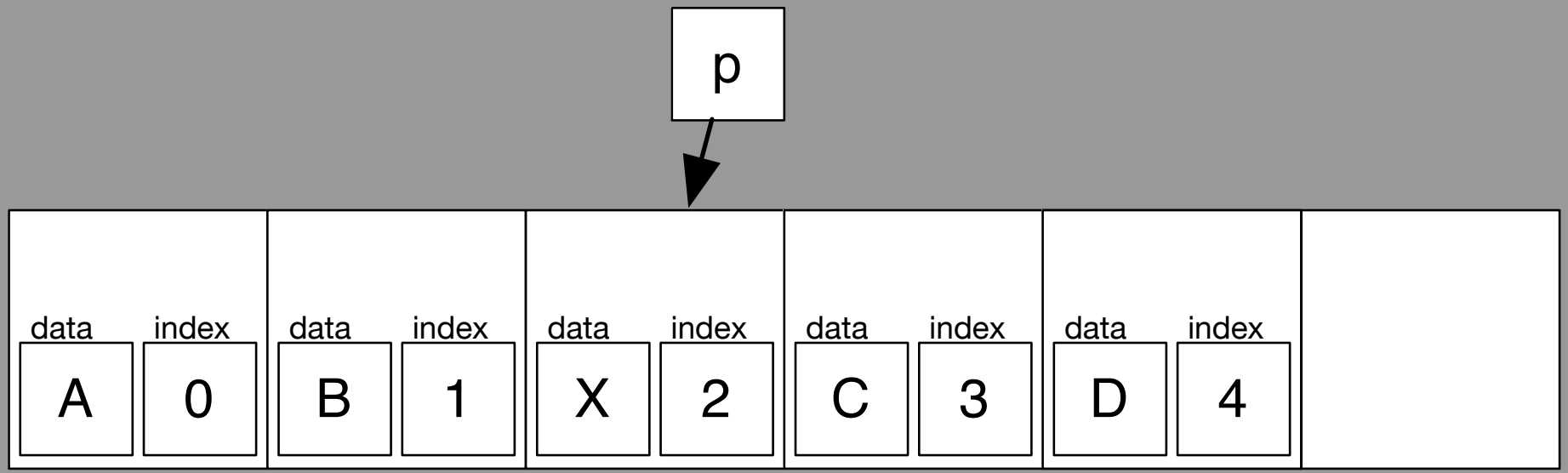
Array-Based List Operations

insertBefore(p, d)



Array-Based List Operations

insertBefore(p, d)



Array-Based List Operations

insertBefore(p, d)

```
1  Algorithm insertBefore (p, v):
2    Input: The value to be inserted and the
       position it should be inserted before
3    Output: The position it was inserted in
4  if size = maxSize then
5    grow()
6  n ← p.getIndex()
7  for every integer value i in the range size
       to n do
8      items[i] ← items[i - 1]
9      items[i].incrementIndex()
10 Create new ArrPos object, called a,
       containing data and index n
11 items[n] ← a
12 size ← size + 1
13 return a
```

Array-Based List Operations

remove(p)

- Remove the value from the list

Array-Based List Operations

remove(p)

- Remove the value from the list
  - Convert p to an `ArrPos` object

Array-Based List Operations

remove(p)

- Remove the value from the list
  - Convert p to an `ArrPos` object
  - Copy value from p

Array-Based List Operations

remove(p)

- Remove the value from the list
  - Convert p to an `ArrPos` object
  - Copy value from p
  - Copy the index from `p` to a variable called `n`

Array-Based List Operations

remove(p)

- Remove the value from the list
  - Convert p to an `ArrPos` object
  - Copy value from p
  - Copy the index from `p` to a variable called `n`
  - Copy all `ArrPos` objects in the array after p one index lower and decrement their index

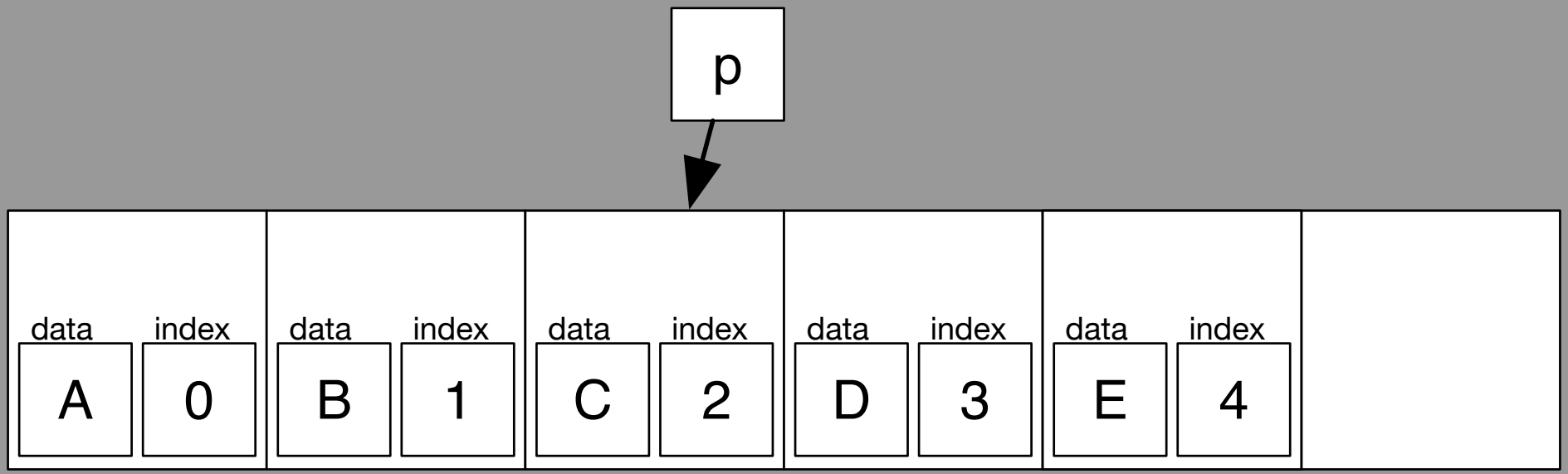
Array-Based List Operations

remove(p)

- Remove the value from the list
  - Convert p to an `ArrPos` object
  - Copy value from p
  - Copy the index from `p` to a variable called `n`
  - Copy all `ArrPos` objects in the array after p one index lower and decrement their index
  - Decrement the size

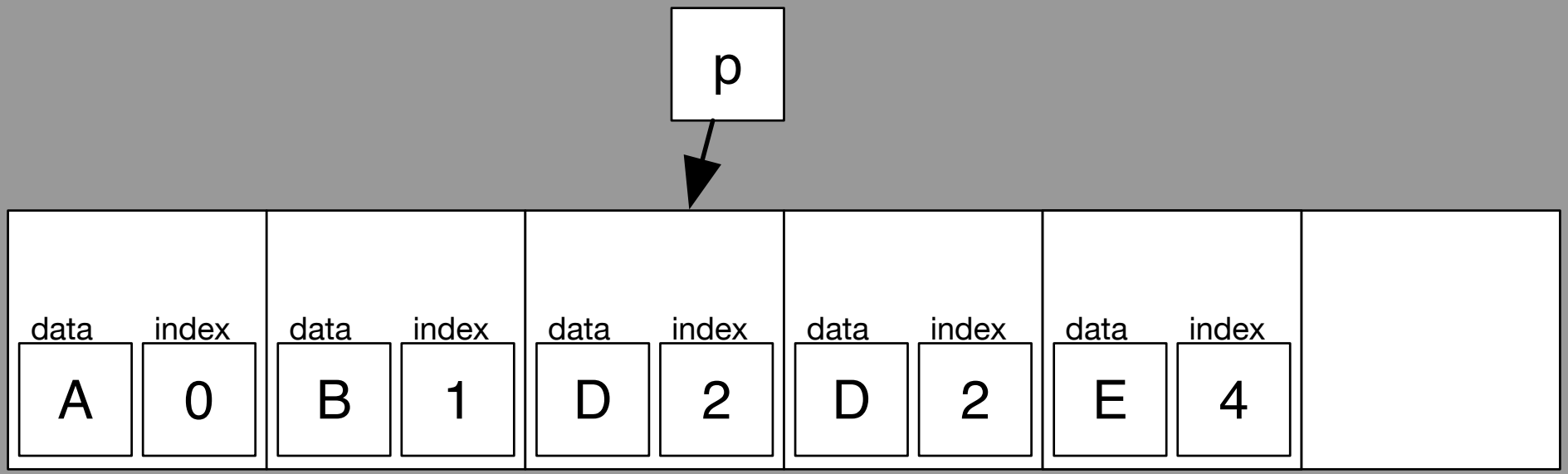
Array-Based List Operations

remove(p)



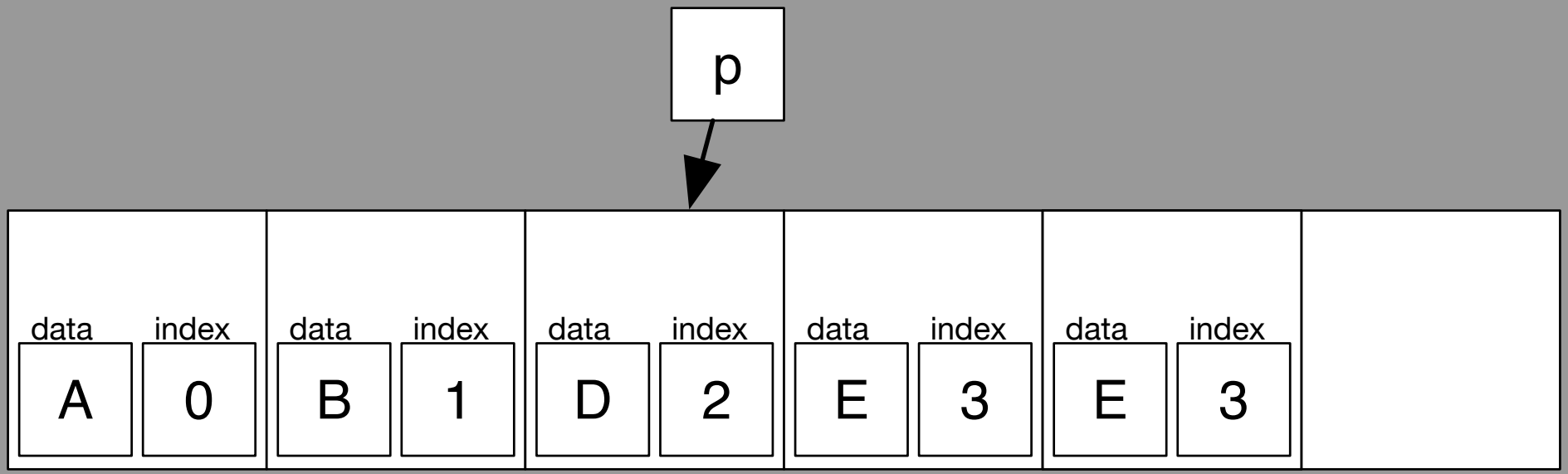
Array-Based List Operations

remove(p)



Array-Based List Operations

remove(p)



Array-Based List Operations

remove(p)

```
1  Algorithm remove (p):
2    Input: The position to be removed from the
       list
3    Output: The value that was removed
4
5  n ← p.getIndex()
6  a ← p.element()
7  for every integer value i in the range n to
       size do
8      items[i] ← items[i + 1]
9      items[i].decrementIndex()
10 size ← size - 1
11 if size < maxSize/4 then
12     shrink()
13 return a
```

Table of Contents

Algorithmic Complexity

- first()

Algorithmic Complexity

- first()
  - O(1)

Algorithmic Complexity

- first()
  - O(1)
- last()

Algorithmic Complexity

- first()
  - O(1)
- last()
  - O(1)

Algorithmic Complexity

- first()
  - O(1)
- last()
  - O(1)
- size()

Algorithmic Complexity

- first()
  - O(1)
- last()
  - O(1)
- size()
  - O(1)

Algorithmic Complexity

- first()
  - O(1)
- last()
  - O(1)
- size()
  - O(1)
- isEmpty()

Algorithmic Complexity

- first()
  - O(1)
- last()
  - O(1)
- size()
  - O(1)
- isEmpty()
  - O(1)

Algorithmic Complexity

- first()
  - O(1)
- last()
  - O(1)
- size()
  - O(1)
- isEmpty()
  - O(1)
- after(p)

Algorithmic Complexity

- first()
  - O(1)
- last()
  - O(1)
- size()
  - O(1)
- isEmpty()
  - O(1)
- after(p)
  - O(1)

Algorithmic Complexity

- first()
  - O(1)
- last()
  - O(1)
- size()
  - O(1)
- isEmpty()
  - O(1)
- after(p)
  - O(1)
- before(p)

Algorithmic Complexity

- first()
  - O(1)
- last()
  - O(1)
- size()
  - O(1)
- isEmpty()
  - O(1)
- after(p)
  - O(1)
- before(p)
  - O(1)

Algorithmic Complexity

insertFirst(d)

```
1  Algorithm insertFirst (v):
2    Input: The value to be inserted
3    Output: The position it was inserted in
4
5  if size = maxSize then
6    grow()
7  Create ArrPos n containing v with index 0
8  for every integer value i in the range size
       to 1 (inclusive) do
9      items[i] ← items[i - 1]
10     items[i].incrementIndex()
11 items[0] ← n
12 size ← size + 1
13 return n
```

Algorithmic Complexity

insertFirst(d)

```
1  Algorithm insertFirst (v):
2    Input: The value to be inserted
3    Output: The position it was inserted in
4
5  if size = maxSize then
6    grow()
7  Create ArrPos n containing v with index 0
8  for every integer value i in the range size
     to 1 (inclusive) do
9      items[i] ← items[i - 1]
10     items[i].incrementIndex()
11 items[0] ← n
12 size ← size + 1
13 return n
```

- Complexity is O(n)

Algorithmic Complexity

insertLast(d)

```
1  Algorithm insertLast (v):
2    Input: The value to be inserted
3    Output: The position it was inserted in
4
5  if size = maxSize then
6    grow()
7  Create ArrPos n containing v with index size
8  items[size] ← n
9  size ← size + 1
10 return n
```

Algorithmic Complexity

insertLast(d)

```
1  Algorithm insertLast (v):
2    Input: The value to be inserted
3    Output: The position it was inserted in
4
5  if size = maxSize then
6    grow()
7  Create ArrPos n containing v with index size
8  items[size] ← n
9  size ← size + 1
10 return n
```

- Complexity is O(1)

Algorithmic Complexity

insertAfter(p, d)

```
1  Algorithm insertAfter (p, v):
2    Input: The value to be inserted and the position it
       should be inserted after
3    Output: The position it was inserted in
4  if size = maxSize then
5    grow()
6  n ← p.getIndex()
7  for every integer value i in the range size to n + 1 do
8      items[i] ← items[i − 1]
9      items[i].incrementIndex()
10 Create new ArrPos object, called a, containing data and
     index n+1
11 items[n+1] ← a
12 size ← size + 1
13 return a
```

Algorithmic Complexity

insertAfter(p, d)

```
1  Algorithm insertAfter (p, v):
2    Input: The value to be inserted and the position it
       should be inserted after
3    Output: The position it was inserted in
4  if size = maxSize then
5    grow()
6  n ← p.getIndex()
7  for every integer value i in the range size to n + 1 do
8      items[i] ← items[i − 1]
9      items[i].incrementIndex()
10 Create new ArrPos object, called a, containing data and
     index n+1
11 items[n+1] ← a
12 size ← size + 1
13 return a
```

- Complexity is O(n)

Algorithmic Complexity

insertBefore(p, d)

```
1  Algorithm insertBefore (p, v):
2    Input: The value to be inserted and the position it
       should be inserted before
3    Output: The position it was inserted in
4  if size = maxSize then
5    grow()
6  n ← p.getIndex()
7  for every integer value i in the range size to n do
8      items[i] ← items[i − 1]
9      items[i].incrementIndex()
10 Create new ArrPos object, called a, containing data and
     index n
11 items[n] ← a
12 size ← size + 1
13 return a
```

Algorithmic Complexity

insertBefore(p, d)

```
1  Algorithm insertBefore (p, v):
2    Input: The value to be inserted and the position it
       should be inserted before
3    Output: The position it was inserted in
4  if size = maxSize then
5    grow()
6  n ← p.getIndex()
7  for every integer value i in the range size to n do
8      items[i] ← items[i − 1]
9      items[i].incrementIndex()
10 Create new ArrPos object, called a, containing data and
     index n
11 items[n] ← a
12 size ← size + 1
13 return a
```

- Complexity is O(n)

Algorithmic Complexity

remove(p)

```
1  Algorithm remove (p):
2    Input: The position to be removed from the list
3    Output: The value that was removed
4
5  n ← p.getIndex()
6  a ← p.element()
7  for every integer value i in the range n to size do
8      items[i] ← items[i + 1]
9      items[i].decrementIndex()
10 size ← size − 1
11 if size < maxSize/4 then
12     shrink()
13 return a
```

Algorithmic Complexity

remove(p)

```
1  Algorithm remove (p):
2    Input: The position to be removed from the list
3    Output: The value that was removed
4
5  n ← p.getIndex()
6  a ← p.element()
7  for every integer value i in the range n to size do
8      items[i] ← items[i + 1]
9      items[i].decrementIndex()
10 size ← size − 1
11 if size < maxSize/4 then
12     shrink()
13 return a
```

- Complexity is O(n)