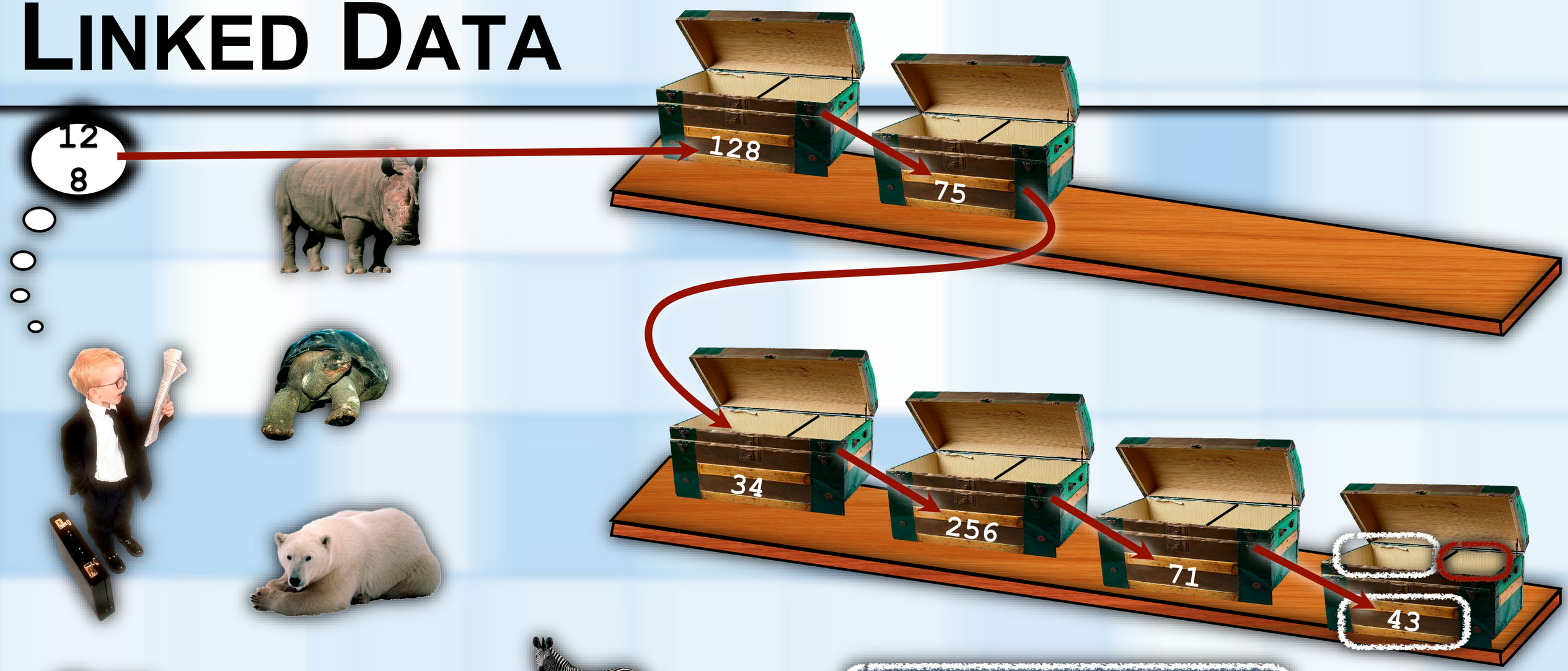
Linked Chain Concepts

Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Linked Data



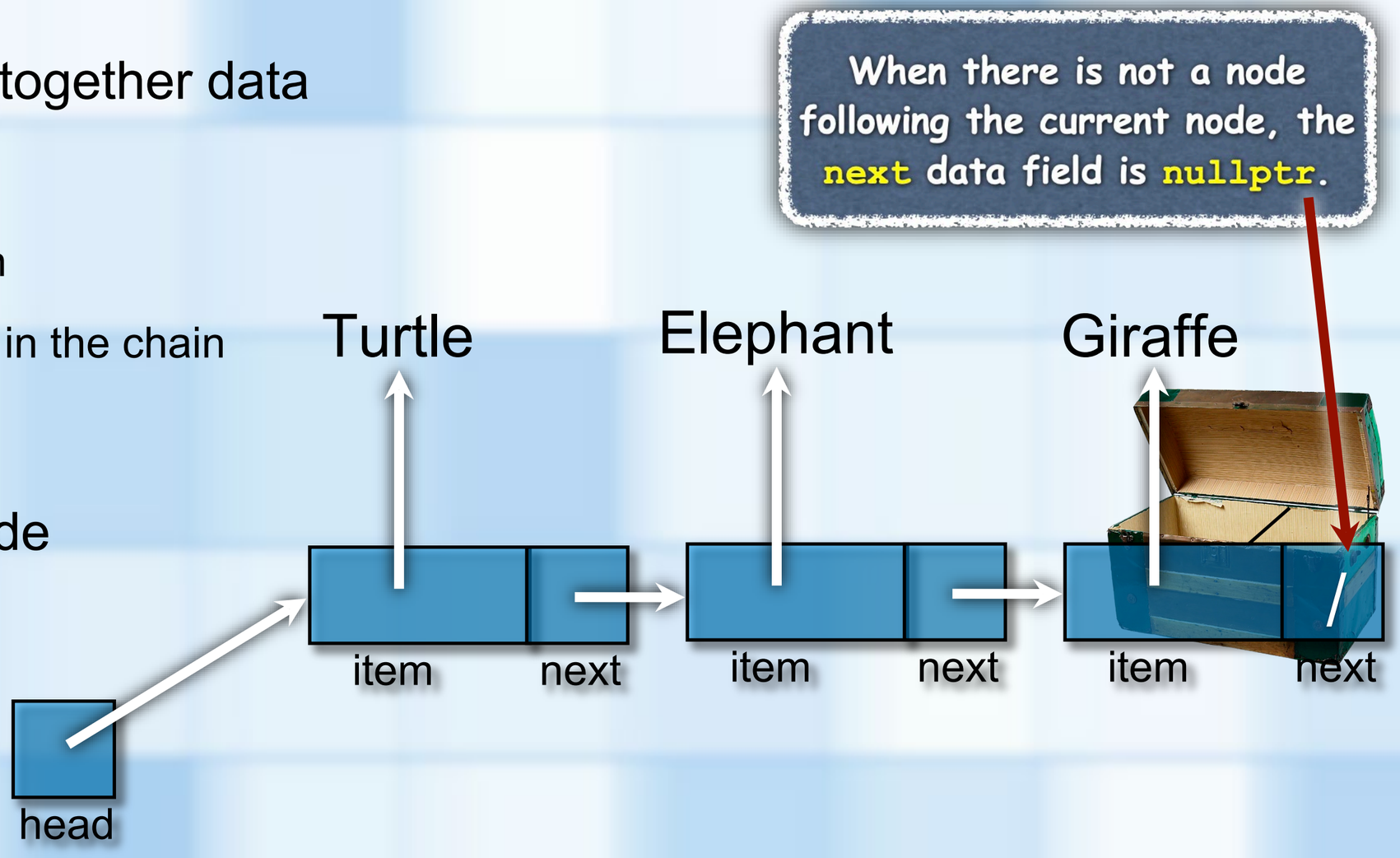
Each trunk has two compartments:
- Item to store
- Number of another trunk

Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Linked Data

- **Node**
  - Object used for linking together data
  - Two data fields
    - Data item in the collection
    - Address of the next node in the chain
- **Head**
  - References the first node in the chain

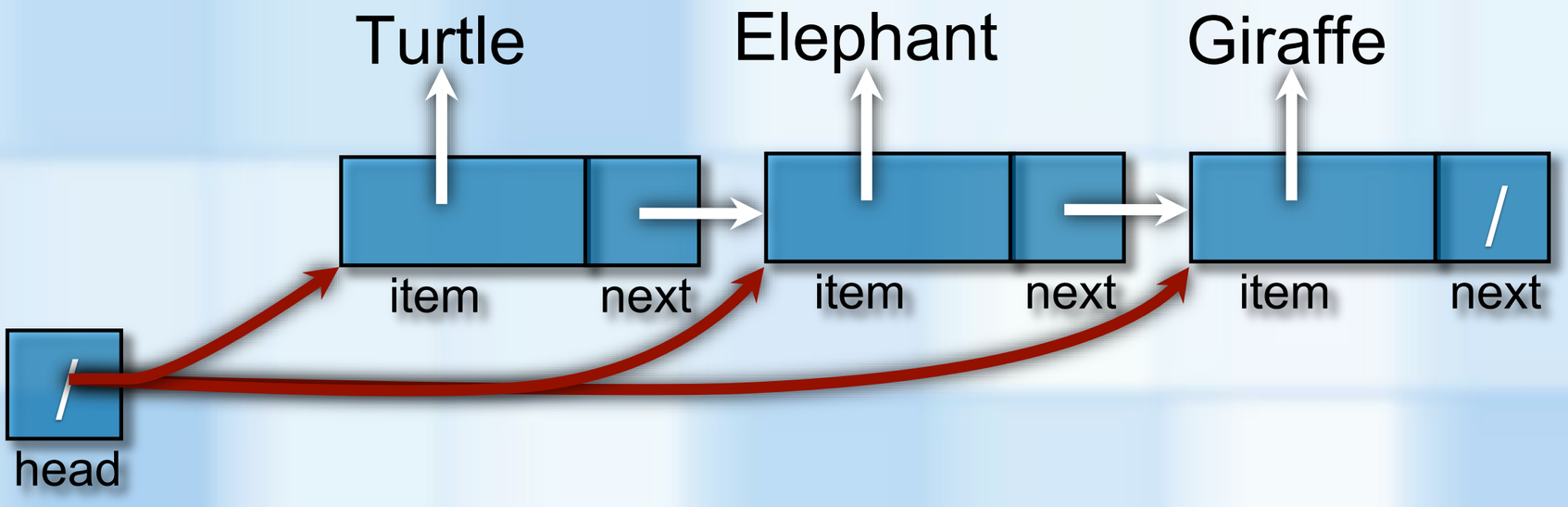


Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Linked Data

- **Forming a Chain**
  - Asked to store item
  - Create a node
  - Store reference to item
  - Store reference to new node in head
- While there are more items
  - Create a node
  - Store reference to item
  - **Copy reference in head to next field in node**
  - Store reference to new node in head



Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Class Node

- **Data fields for**
  - Data stored in the Node
  - Reference to next node in the chain
- **Constructors**
  - With references to data and the next node
  - With reference only to next node
    - Set `next` to `nullptr`
- **Accessor and mutator methods**
  - For getting a reference to the data or next node
  - For setting the next node and the data

Node.h

```
/** @file Node.h */

#ifndef _NODE
#define _NODE

template<class ItemType>
class Node
{
private:
  ItemType       item;
  Node<ItemType>* next;

public:
  Node();
  Node(const ItemType& anItem);
  Node(const ItemType& anItem,
                      Node<ItemType>* nextNode);

  void setItem(const ItemType& anItem);
  void setNext(Node<ItemType>* nextNode);

  ItemType getItem() const;
  Node<ItemType>* getNext() const;

}; // end Node

#include "Node.cpp"
#endif
```

Pearson
Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Class Node

- **Data fields for**
  - Data stored in the Node
  - Reference to next node in the chain
- **Constructors**
  - With reference only to next node
    - Set `next` to `nullptr`
  - With references to data and the next node
- **Accessor and mutator methods**
  - For getting a reference to the data or next node
  - For setting the next node and the data

Node.cpp

```
/** @file Node.cpp */
#include "Node.h"

template<class ItemType>
Node<ItemType>::Node()
                              : next(nulptr)
{ }  // end default constructor

template<class ItemType>
Node<ItemType>::Node(const ItemType& anItem)
                            : item(anItem), next(nullptr)

{ }  // end constructor

template<class ItemType>
Node<ItemType>::Node(const ItemType& anItem,
                                    Node<ItemType>* nextNode)
                            : item(anItem), next(nextNode)
{ }  // end constructor
```

Pearson
Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Class Node

- **Data fields for**
  - Data stored in the Node
  - Reference to next node in the chain
- **Constructors**
  - With reference only to next node
    - Set `next` to `nullptr`
  - With references to data and the next node
- **Accessor and mutator methods**
  - For getting a reference to the data or next node
  - For setting the next node and the data

Node.cpp

```
template<class ItemType>
ItemType Node<ItemType>::getItem() const
{
  return item;
} // end getItem


template<class ItemType>
Node<ItemType>* Node<ItemType>::getNext() const
{
  return next;
} // end getNext


template<class ItemType>
void Node<ItemType>::setItem(const ItemType& anItem)
{
  item = anItem;
} // end setItem


template<class ItemType>
void Node<ItemType>::setNext(Node<ItemType>* nextNode)
{
  next = nextNode;
} // end setNext
```

Pearson
Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Linked Chain Concepts

Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Implementing a LinkedBag

- **Steps to Follow**
  - Decide on Data Fields
  - Implement Constructor
    - Initialize the data fields
  - Implement Core Functions
    - Methods critical to collection functionality
    - Methods to check status of collection
  - Test Your Implementation
  - Implement Additional Methods
    - Test Your Implementation

```
/** @file Node.h */

#ifndef _NODE
#define _NODE

template<class ItemType>
class Node
{
private:
  ItemType       item;
  Node<ItemType>* next;

public:
  Node();
  Node(const ItemType& anItem);
  Node(const ItemType& anItem,
                    Node<ItemType>* nextNode);

  void setItem(const ItemType& anItem);
  void setNext(Node<ItemType>* nextNode);
  ItemType getItem() const;

  Node<ItemType>* getNext() const;
}; // end Node

#include "Node.cpp"
#endif
```

Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Decide on Data Fields

- **Items are stored in a linked chain**
  - Reference to the first node in the chain
  - Number of entires in the chain

LinkedBag.h

```
template<class ItemType>
class LinkedBag : public BagInterface<ItemType>
{
private:
  Node<ItemType>* headPtr;
  int itemCount;

  Node<ItemType>* getPointerTo(const
                               ItemType& anEntry) const;

public:
   LinkedBag();
  LinkedBag(const LinkedBag<ItemType>& aBag);
  ~LinkedBag();
   int getCurrentSize() const;
   bool isEmpty() const;
   bool add(const ItemType& newEntry);
   bool remove(const ItemType& anEntry);
   void clear();
   bool contains(const ItemType& anEntry) const;
   int getFrequencyOf(const ItemType& anEntry) const;
   void toVector(std::vector<ItemType>& bagContents) const;
};  // end LinkedBag
```

Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

IMPLEMENTING CONSTRUCTORS

- **Must happen before other class methods can be called**
- **Ensure all data fields are initialized**
  - No items in bag

LinkedBag.cpp

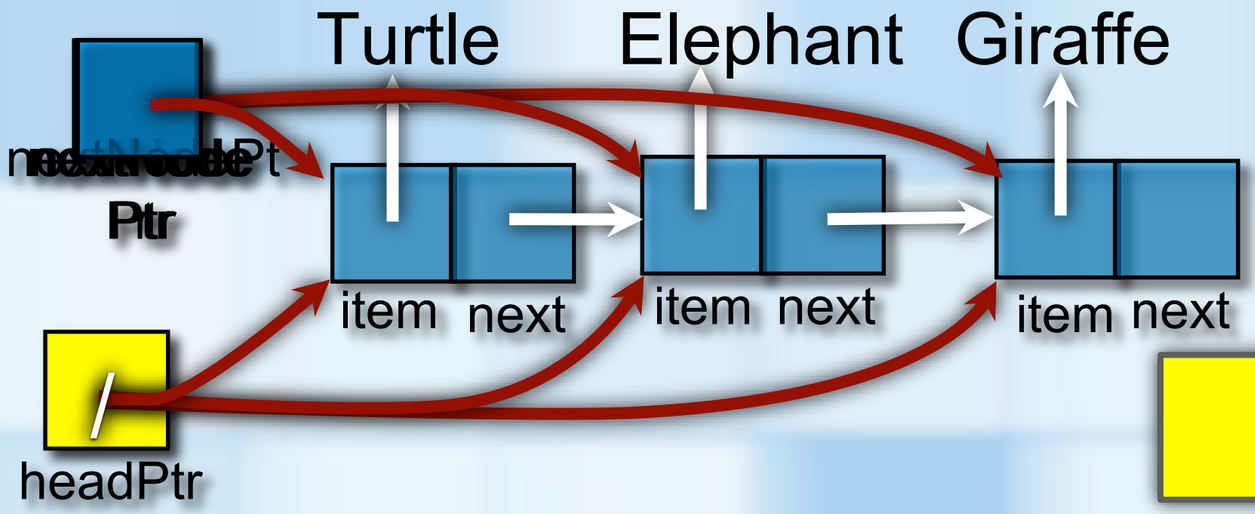
```
template<class ItemType>
LinkedBag<ItemType>::LinkedBag()
                    : headPtr(nullptr), itemCount(0)
{
}  // end default constructor
```

Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Implementing Core Methods

- **Place items into bag**
  - Create a node and store reference to item
  - Copy reference in head (**`headPtr`**) to **`next`** field in node
  - Store reference to new node in head
  - Increment number of entries in bag



`LinkedBag.cpp`

```
template<class ItemType>
bool LinkedBag<ItemType>::add(const
                                    ItemType& newEntry)
{
  // add to beginning of chain:
                    new node references rest of chain;
  // (headPtr is null if chain is empty)

  Node<ItemType>* nextNodePtr =
                new Node<ItemType>(newEntry, headPtr);

  headPtr = nextNodePtr;  // new node is first node
  itemCount++;

  return true;
}  // end add
```

`itemCount`

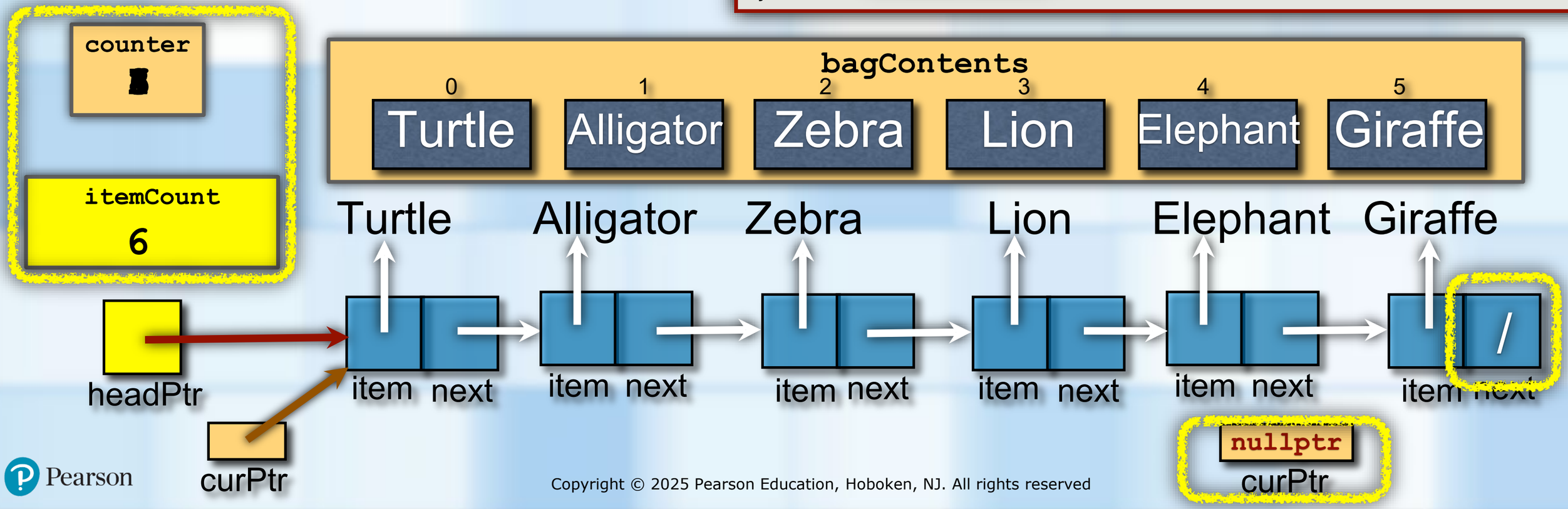
Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Implementing Core Methods

- **Report on items in object**
  - Allows us to determine if the items were added properly.

```
template<class ItemType>
std::vector<ItemType> LinkedBag<ItemType>::toVector() const
{
    std::vector<ItemType> bagContents;
  Node<ItemType>* curPtr = headPtr;
  int counter = 1
    while ((curPtr != nullptr) && (counter <= itemCount))
  {
      bagContents.push_back(curPtr->getItem());
    curPtr = curPtr->getNext();
    counter++;
  }  // end while
}  // end toVector
```



Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

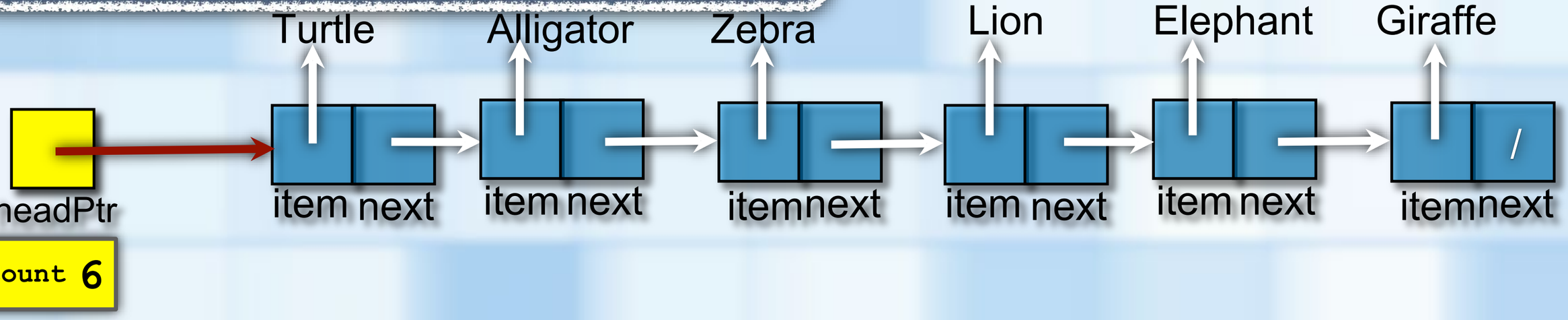
Core LinkedBag Methods

Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Removing a Specific Item

```
remove (anEntry)
  locate node N that contains anEntry
  if ( N exists )
       replace the entry in node N
                with the entry in the first node
       remove the first node
  return true (false) to indicate success (failure)
```



itemCount 6

Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

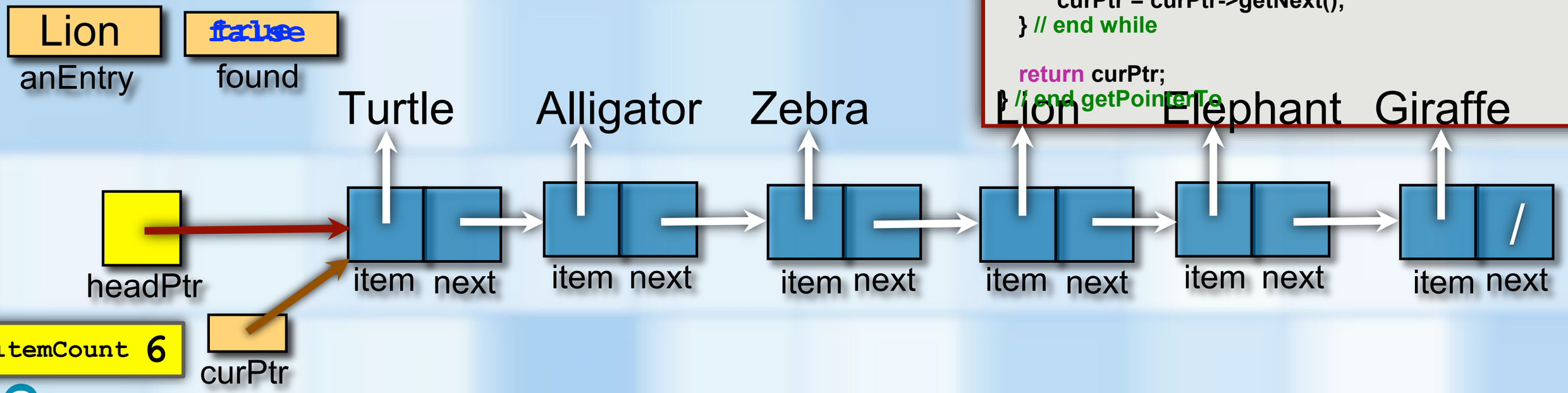
REMOVING A SPECIFIC ITEM

- **The method `getPointerTo`**
  - returns a reference to the node containing **`anEntry`** or **`nullptr`** if **`anEntry`** is not in the bag.
  - while the entry has not been found and there are more nodes in the chain
    - compare the entry in the current node to the target entry

```
template<class ItemType>
Node<ItemType>*
LinkedBag<ItemType>::getPointerTo(
                const ItemType& anEntry) const
{
  bool found = false;
  Node<ItemType>* curPtr = headPtr;

  while (!found && (curPtr != nullptr))
  {
    if (anEntry == curPtr->getItem())
      found = true;
    else
      curPtr = curPtr->getNext();
  } // end while

  return curPtr;
} // end getPointerTo
```



Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

REMOVING AN ITEM

```
remove (anEntry)
  locate node N that contains anEntry
  if ( N exists )
      replace the entry in node N
              with the entry in the first node
      remove the first node
  return true (false) to indicate success (failure)
```

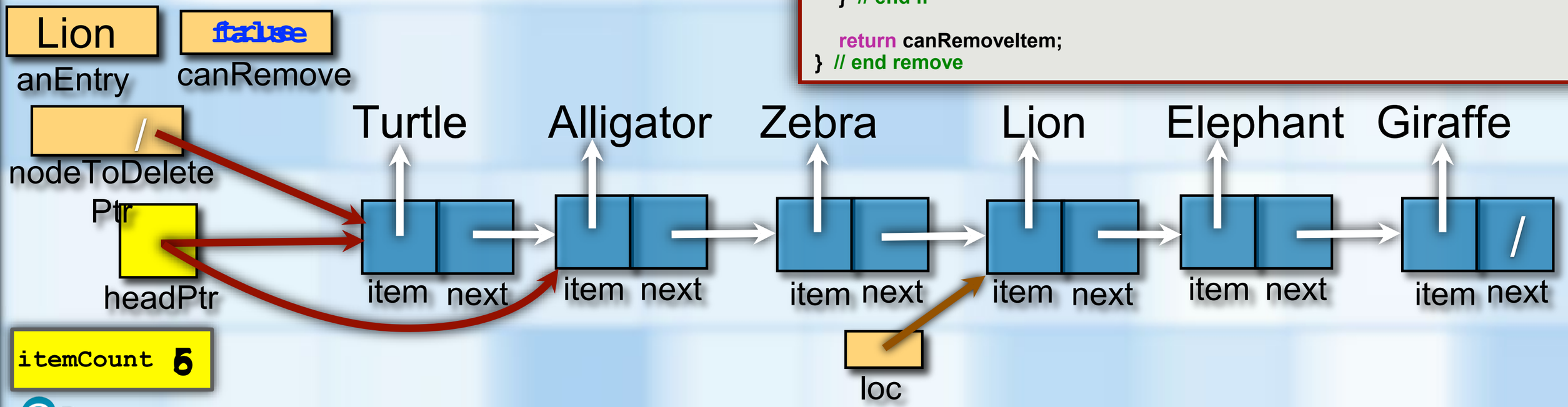
```
template<class ItemType>
bool LinkedBag<ItemType>::remove(const ItemType& anEntry)
{
  Node<ItemType>* loc = getPointerTo(anEntry);
   bool canRemoveItem = !isEmpty() && (loc != nullptr);
   if (canRemoveItem)
   {
    // copy data from first node to located node
    loc->setItem(headPtr->getItem());

    // delete first node
    Node<ItemType>* nodeToDeletePtr = headPtr;
    headPtr = headPtr->getNext();

    // return node to the system
    delete nodeToDeletePtr;
    nodeToDeletePtr = nullptr;

     itemCount--;
   } // end if

   return canRemoveItem;
} // end remove
```



Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

The Method contains

- **Use the private helper method `getPointerTo`**
  - if the helper returns **`nullptr`**
    - the item is not in the bag
  - if the helper returns a reference to a node
    - the item is in the bag

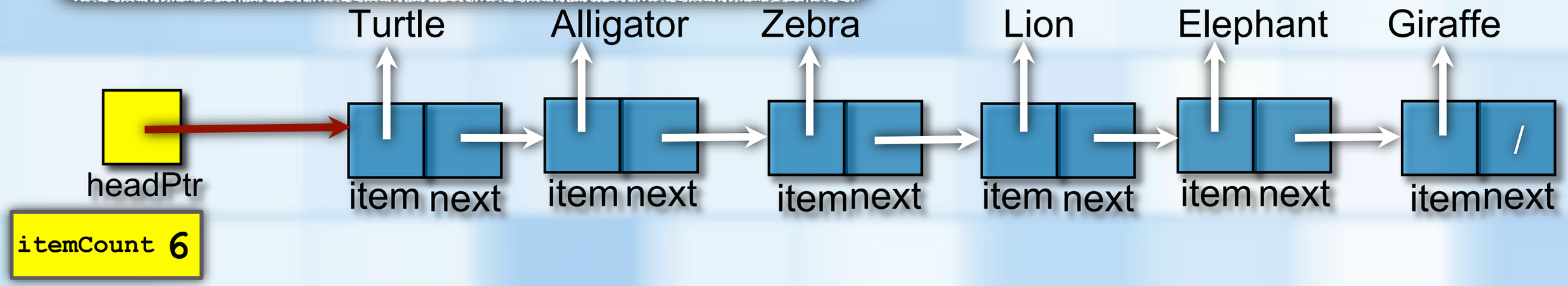
```
template<class ItemType>
bool LinkedBag<ItemType>::contains(
                        const ItemType& anEntry) const
{
   return (getPointerTo(anEntry) != nullptr);
}  // end contains
```

Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Removing a Specific Item

```
remove (anEntry)
  locate node N that contains anEntry
  if ( N exists )
      replace the entry in node N
              with the entry in the first node
      remove the first node
  return true (false) to indicate success (failure)
```



Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

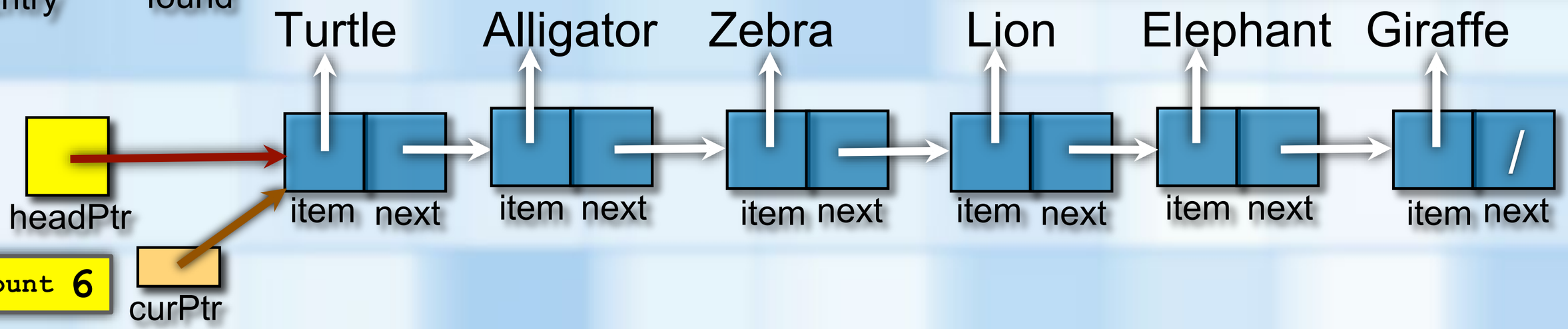
REMOVING A SPECIFIC ITEM

- **The method `getPointerTo`**
  - returns a reference to the node containing **`anEntry`** or **`nullptr`** if **`anEntry`** is not in the bag.
  - while the entry has not been found and there are more nodes in the chain
    - Lion
    - anEntry
    - false
    - found
    - ...try in the current node to the target entry

```
template<class ItemType>
Node<ItemType>* LinkedBag<ItemType>::getPointerTo(
                    const ItemType& anEntry) const
{
  bool found = false;
  Node<ItemType>* curPtr = headPtr;

  while (!found && (curPtr != nullptr))
  {
    if (anEntry == curPtr->getItem())
      found = true;
    else
      curPtr = curPtr->getNext();
  } // end while

  return curPtr;
} // end getPointerTo
```



Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

Removing an Item

```
remove (anEntry)
  locate node N that contains anEntry
  if ( N exists )
      replace the entry in node N
              with the entry in the first node
      remove the first node
  return true (false) to indicate success (failure)
```

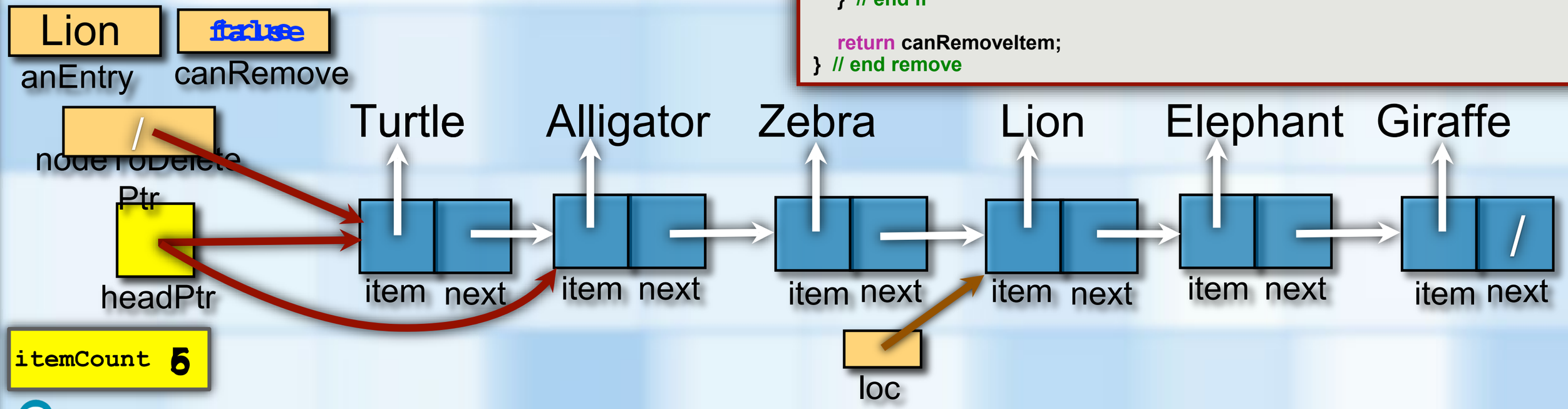
```
template<class ItemType>
bool LinkedBag<ItemType>::remove(const ItemType& anEntry)
{
  Node<ItemType>* loc = getPointerTo(anEntry);
   bool canRemoveItem = !isEmpty() && (loc != nullptr);
   if (canRemoveItem)
   {
     // copy data from first node to located node
     loc->setItem(headPtr->getItem());

     // delete first node
     Node<ItemType>* nodeToDeletePtr = headPtr;
     headPtr = headPtr->getNext();

     // return node to the system
     delete nodeToDeletePtr;
     nodeToDeletePtr = nullptr;

      itemCount--;
   }  // end if

   return canRemoveItem;
}  // end remove
```



Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved

The Method contains

- **Use the private helper method `getPointerTo`**
  - if the helper returns **`nullptr`**
    - the item is not in the bag
  - if the helper returns a reference to a node
    - the item is in the bag

```
template<class ItemType>
bool LinkedBag<ItemType>::contains(
                        const ItemType& anEntry) const
{
   return (getPointerTo(anEntry) != nullptr);
}  // end contains
```

Pearson

Copyright © 2025 Pearson Education, Hoboken, NJ. All rights reserved