CS302 - Data Structures

using C++

Topic: Linked Lists – Implementation of the Bag ADT

Kostas Alexis

AUTONOMOUS
ROBOTS
LAB

Linked Data

- Consider we need to store data, we need to store data of certain type and we need to have a specific way for how they are linked.

Linked Data

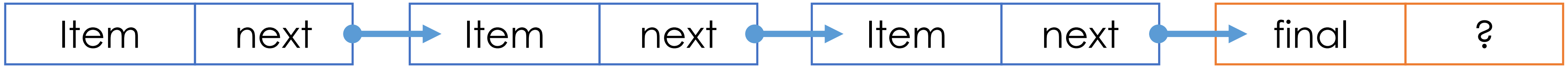
- Consider we need to store data, we need to store data of certain type and we need to have a specific way for how they are linked.
- Consider the example of a train with multiple cars:

Linked Data

- Consider we need to store data, we need to store data of certain type and we need to have a specific way for how they are linked.
- Consider the example of a train with multiple cars:
  - Each car has certain cargo
  - Each car connects to the next car only
  - The locomotive is a special entity – does not store cargo and is always ahead

Linked Data

- Consider we need to store data, we need to store data of certain type and we need to have a specific way for how they are linked.
- A visualization…



Linked Data

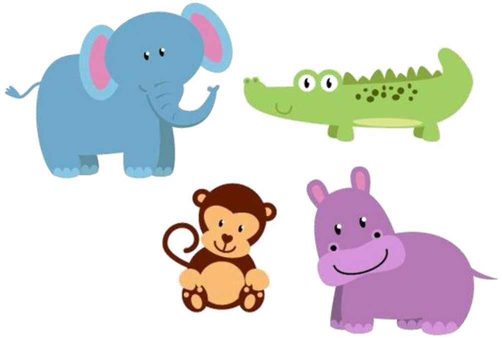
- Consider we need to store data, we need to store data of certain type and we need to have a specific way for how they are linked.
- A toy-example: we want to store "animals" on a list.

Linked Data

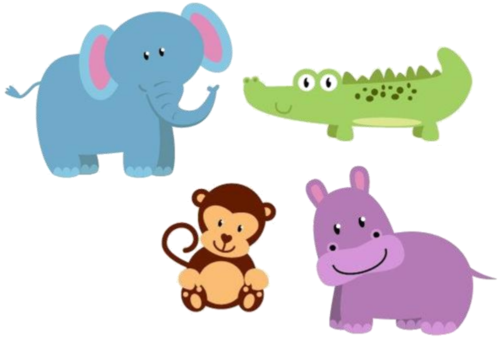
- Consider we need to store data, we need to store data of certain type and we need to have a specific way for how they are linked.
- A toy-example: we want to store "animals" on a list.
- We care to store them such that one after the other we can search which of them we have available. The order of storage has no particular role.

Linked Data

- Consider we need to store data, we need to store data of certain type and we need to have a specific way for how they are linked.
- A toy-example: we want to store “animals” on a list.
- We care to store them such that one after the other we can search which of them we have available. The order of storage has no particular role.
- Let’s define a simple solution…



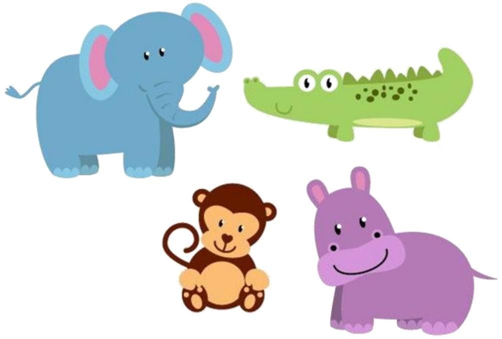
Linked Data



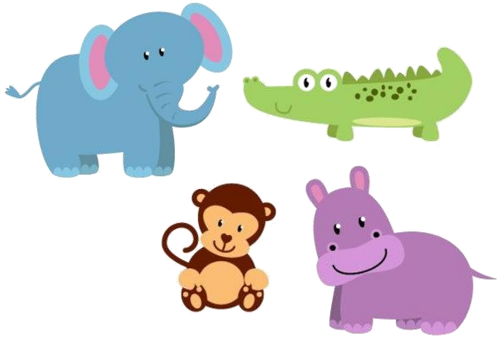
- Node
  - Object used for linking together data
  - Two data fields

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

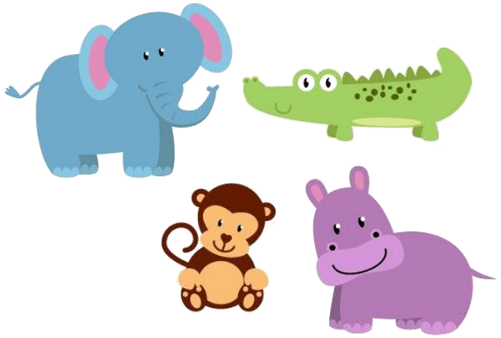


Linked Data

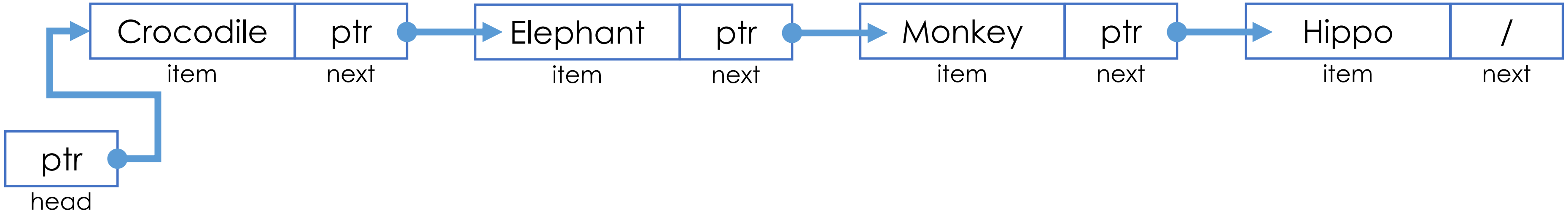


- Node
    - Object used for linking together data
    - Two data fields
        - Data item in the collection
        - Address of the next node in the chain
- Head
    - References the first node in the chain
        - First node

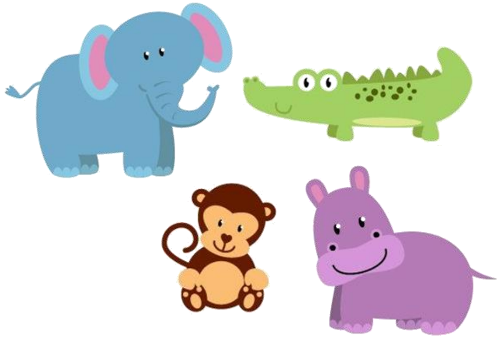
Linked Data



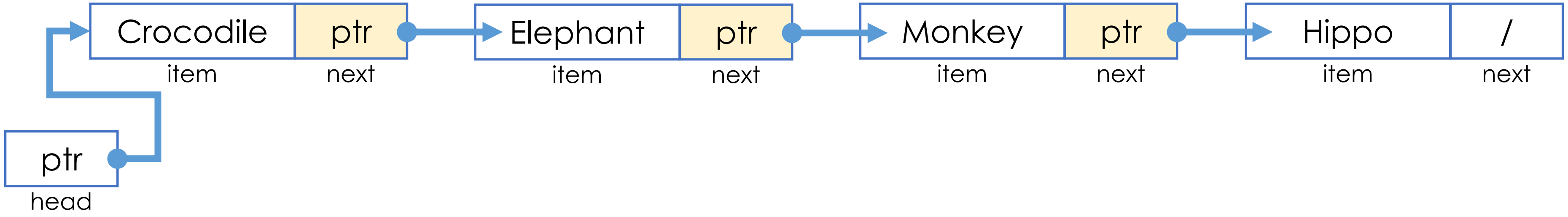
- Node
    - Object used for linking together data
    - Two data fields
        - Data item in the collection
        - Address of the next node in the chain
- Head
    - References the first node in the chain
        - First node



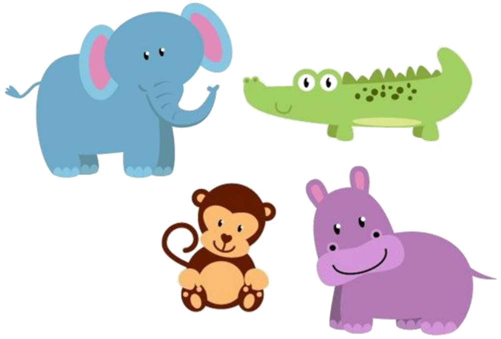
Linked Data



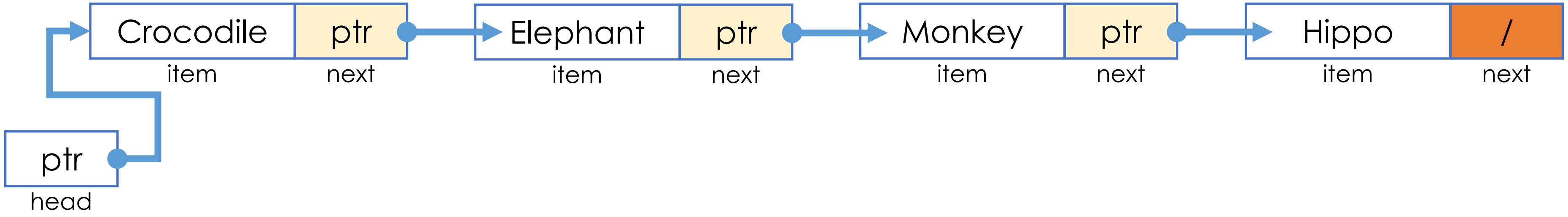
- Node
  - Object used for linking together data
  - Two data fields
    - Data item in the collection
    - Address of the next node in the chain
- Head
  - References the first node in the chain
    - First node



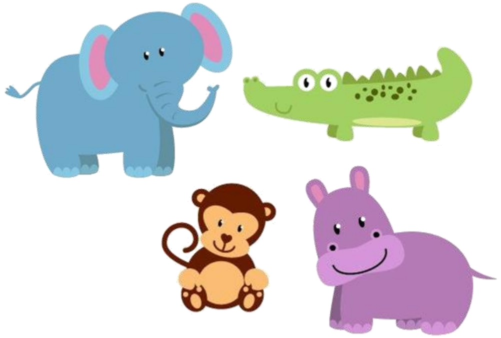
Linked Data



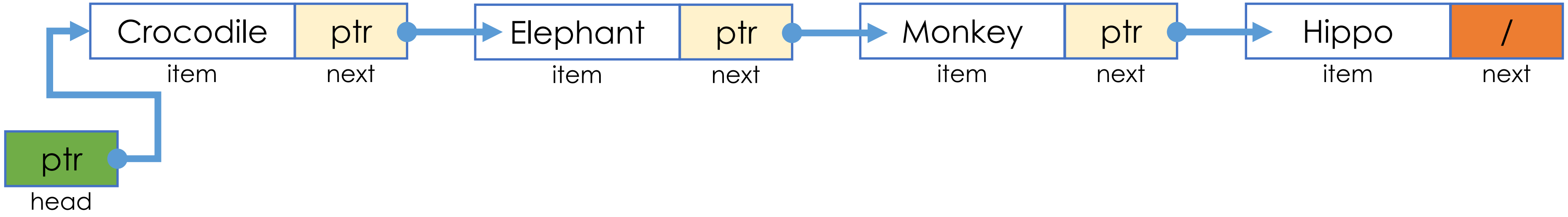
- Node
  - Object used for linking together data
  - Two data fields
    - Data item in the collection
    - Address of the next node in the chain
- Head
  - References the first node in the chain
    - First node



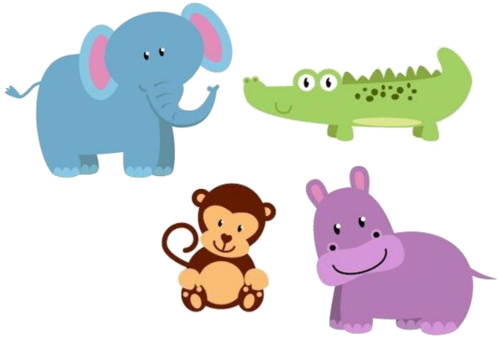
Linked Data



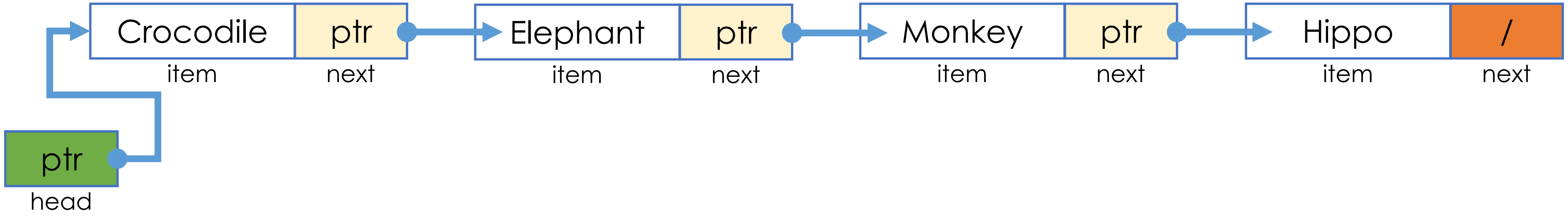
- Node
  - Object used for linking together data
  - Two data fields
    - Data item in the collection
    - Address of the next node in the chain
- Head
  - References the first node in the chain
    - First node



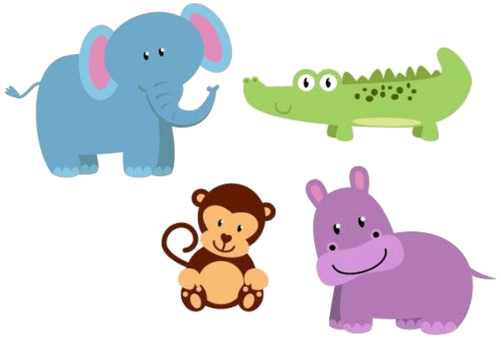
Linked Data



- Forming a chain
  - Start with a variable that holds a reference to the first node in the chain: reasonable choice to start with nullptr

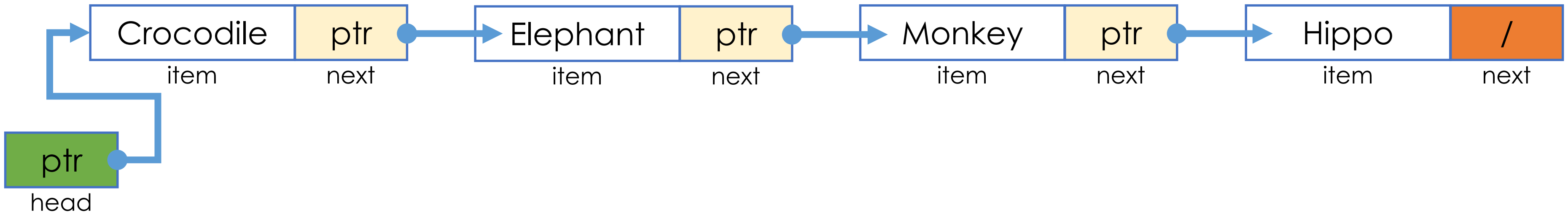


Linked Data

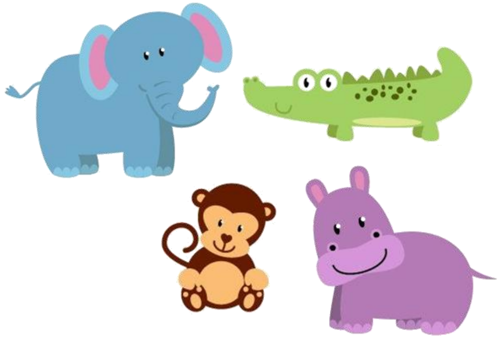


- Forming a chain
  - Asked to store item
  - Create a node
  - Store reference to item
  - Store reference to new node in head

- 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

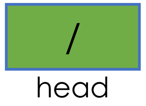


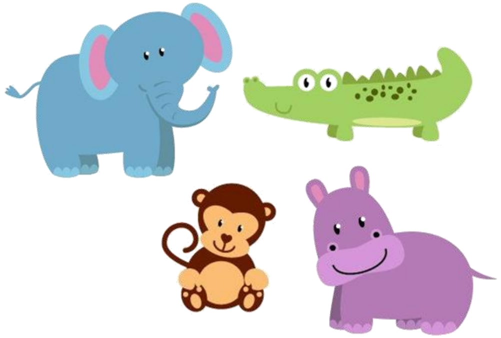
Linked Data



- 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

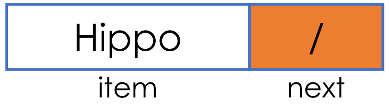


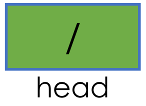


Linked Data

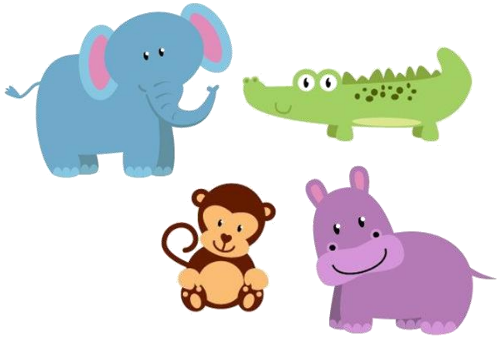
- Forming a chain
  - Asked to store item
  - Create a node
  - Store reference to item
  - Store reference to new node in head

- 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



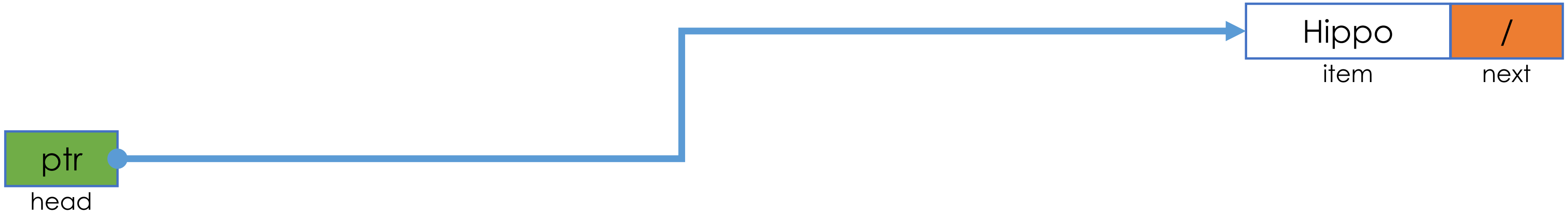


Linked Data

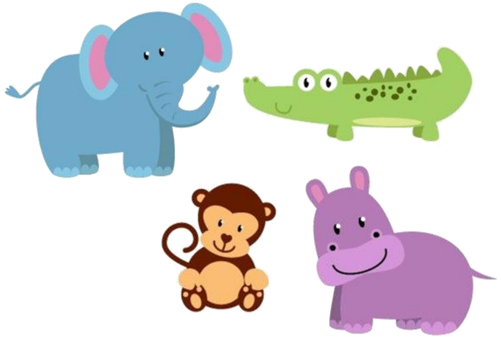


- Forming a chain
  - Asked to store item
  - Create a node
  - Store reference to item
  - Store reference to new node in head

- 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

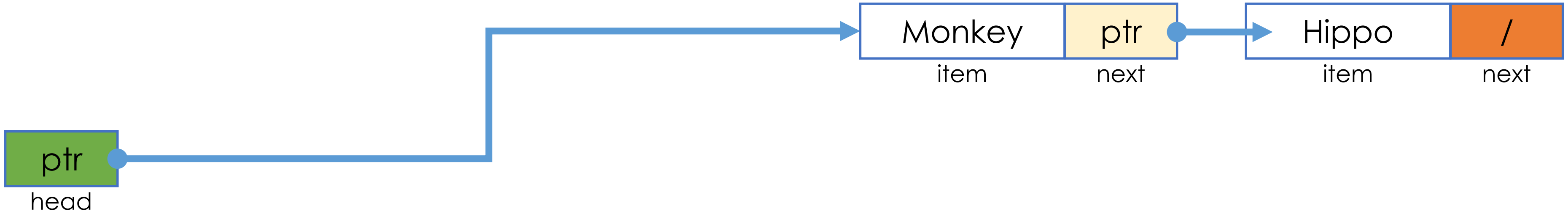


Linked Data

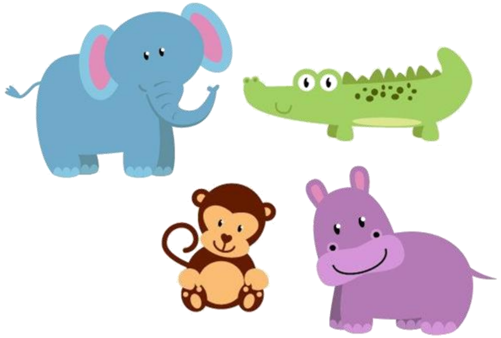


- Forming a chain
  - Asked to store item
  - Create a node
  - Store reference to item
  - Store reference to new node in head

- 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

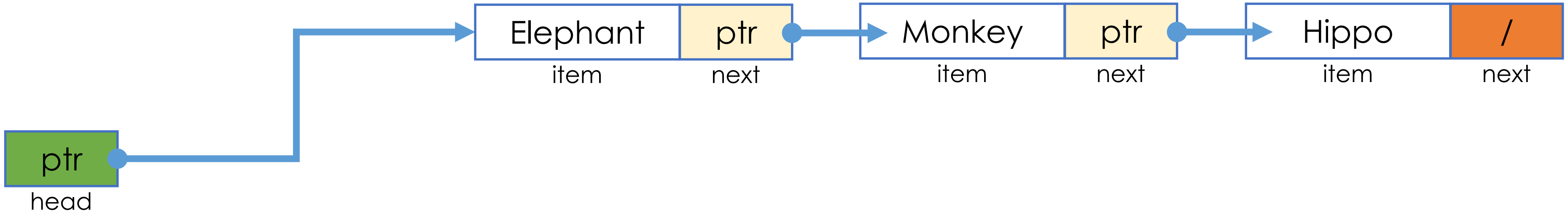


Linked Data

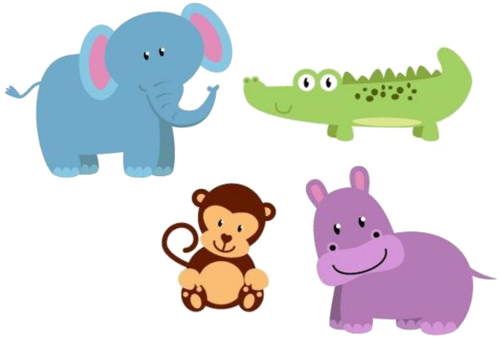


- Forming a chain
  - Asked to store item
  - Create a node
  - Store reference to item
  - Store reference to new node in head

- 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

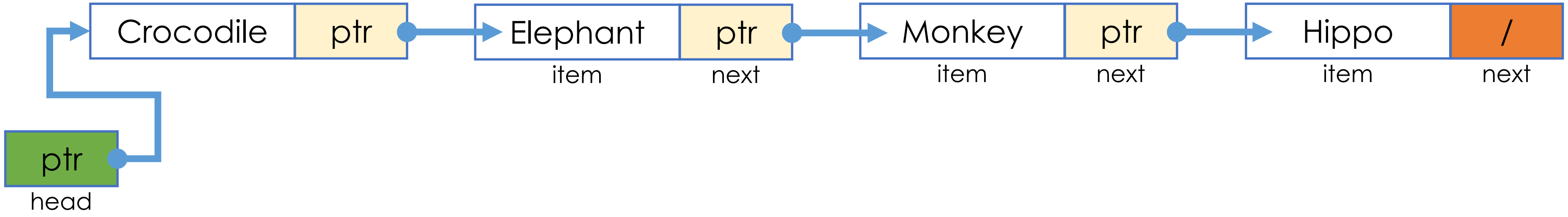


Linked Data



- 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



Class Node

- Data fields for
    - Data stored in node
    - Reference to next node in the chain

Class Node

- Data fields for
  - Data stored in 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`**

Class Node

- Data fields for
  - Data stored in 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**

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

#ifndef NODE_
#define NODE_

template<class ItemType>
class Node
{
private:
      ItemType         item; // A data item
      Node<ItemType>* next; // Point to next node
public:
      Node();
      Node(const ItemType& anItem);
      Node(const ItemType& anItem, Node<ItemType>* nextNodePtr);
      void setItem(const ItemType& anItem)
      void setNext(Node<ItemType>* nextNodePtr);
      ItemType getItem() const;
      Node<ItemType>* getNext() const;
};    // end Node
#include “Node.cpp”
#endif
```

Class Node

- Data fields for
  - Data stored in 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`**

```
/** @file Node.cpp */
#include "Node.h"
template<class ItemType>
Node<ItemType>:: Node() : next(nullptr)
{
}     // 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>* nextNodePtr) :
      item(anItem), next(nextNodePtr)
{
}     // end constructor
template<class ItemType>
void Node<ItemType>::setItem(const ItemType& anItem)
{
      item = anItem;
}     // end setItem
template<class ItemType>
void Node<ItemType>::setNext(Node<ItemType>* nextNodePtr)
{
      next = nextNodePtr;
}     // end setNext
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
```

Class Node

- Data fields for
  - Data stored in 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 and the data

```cpp
/** @file Node.cpp */
#include "Node.h"
template<class ItemType>
Node<ItemType>:: Node() : next(nullptr)
{
}     // 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>* nextNodePtr) :
      item(anItem), next(nextNodePtr)
{
}     // end constructor
template<class ItemType>
void Node<ItemType>::setItem(const ItemType& anItem)
{
      item = anItem;
}     // end setItem
template<class ItemType>
void Node<ItemType>::setNext(Node<ItemType>* nextNodePtr)
{
      next = nextNodePtr;
}     // end setNext
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
```

Implementing a LinkedBag

- Steps to follow
  - Decide on Data Fields
  - Implement a Constructor
    - Initialize the data fields
- Implement Core Functions
  - With references to data and the next node
  - With reference only to next node
    - Set **`next`** to **`nullptr`**
- Test your implementation
- Implement additional methods
  - Test your implementation

```
/** #file BagInterface.h */
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
            virtual int getCurrentSize() const = 0;
            virtual bool isEmpty() const = 0;
            virtual bool add(const ItemType& newEntry) = 0;
            virtual bool remove(const ItemType& anEntry) = 0;
            virtual void clear() = 0;
            virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
            virtual bool contains(const ItemType& anEntry) const = 0;
            virtual std::vector<ItemType> toVector() const = 0;
            virtual ~BagInterface() { }
}; // end BagInterface
#endif
```

Implementing a LinkedBag

- Steps to follow
  - Decide on Data Fields
  - Implement a Constructor
    - Initialize the data fields
- Implement Core Functions
  - With references to data and the next node
  - With reference only to next node
    - Set **`next`** to **`nullptr`**
- Test your implementation
- Implement additional methods
  - Test your implementation

```
/** #file BagInterface.h */
#ifndef _BagInterface_h
#define _BagInterface_h

#include <vector>

template<class ItemType>
class BagInterface
{
public:
            virtual int getCurrentSize() const = 0;
            virtual bool isEmpty() const = 0;
            virtual bool add(const ItemType& newEntry) = 0;
            virtual bool remove(const ItemType& anEntry) = 0;
            virtual void clear() = 0;
            virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
            virtual bool contains(const ItemType& anEntry) const = 0;
            virtual std::vector<ItemType> toVector() const = 0;
            virtual ~BagInterface() { }
}; // end BagInterface
#endif
```

Deciding on Data Fields

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

```
/** ADT bag: Link-based implementation
@file LinkedBag.h */
#ifndef LINKED_BAG_
#define LINKED_BAG_

#include “BagInterface.h”
#include “Node.h
template<class ItemType>
class LinkedBag : public BagInterface<ItemType>
{
private:
      Node<ItemType>*   headPtr; // Pointer to first node
      int itemCount; // Current count of bag items
      Node<ItemType>* getPointerTo(const ItemType& target) const;
public:
      LinkedBag();      // Default constructor
      LinkedBag(const LinkedBag<ItemType>& aBag); // Copy constructor
      virtual ~LinkedBag(); // Destructor is virtual
      int getCurrentSize() const;
      bool isEmpty() const;
      bool add(const ItemType& newEntry);
      bool add(const ItemType& anEntry);
      void clear();
      bool contains(const ItemType& anEntry) const;
      int getFrequencyOf(const ItemType& anEntry) const;
      vector<ItemType> toVector() const;
};    // end LinkedBag
#include “LinkedBag.cpp”
#endif
```

Deciding on Data Fields

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

```
/** ADT bag: Link-based implementation
@file LinkedBag.h */
#ifndef LINKED_BAG_
#define LINKED_BAG_

#include “BagInterface.h”
#include “Node.h
template<class ItemType>
class LinkedBag : public BagInterface<ItemType>
{
private:
      Node<ItemType>*   headPtr; // Pointer to first node
      int itemCount; // Current count of bag items
      Node<ItemType>* getPointerTo(const ItemType& target) const;
public:
      LinkedBag();      // Default constructor
      LinkedBag(const LinkedBag<ItemType>& aBag); // Copy constructor
      virtual ~LinkedBag(); // Destructor is virtual
      int getCurrentSize() const;
      bool isEmpty() const;
      bool add(const ItemType& newEntry);
      bool add(const ItemType& anEntry);
      void clear();
      bool contains(const ItemType& anEntry) const;
      int getFrequencyOf(const ItemType& anEntry) const;
      vector<ItemType> toVector() const;
};    // end LinkedBag
#include “LinkedBag.cpp”
#endif
```

Implementing Constructors

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

```
/** ADT bag: Link-based implementation
@file LinkedBag.cpp */

// Default Constructor

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

Thank you