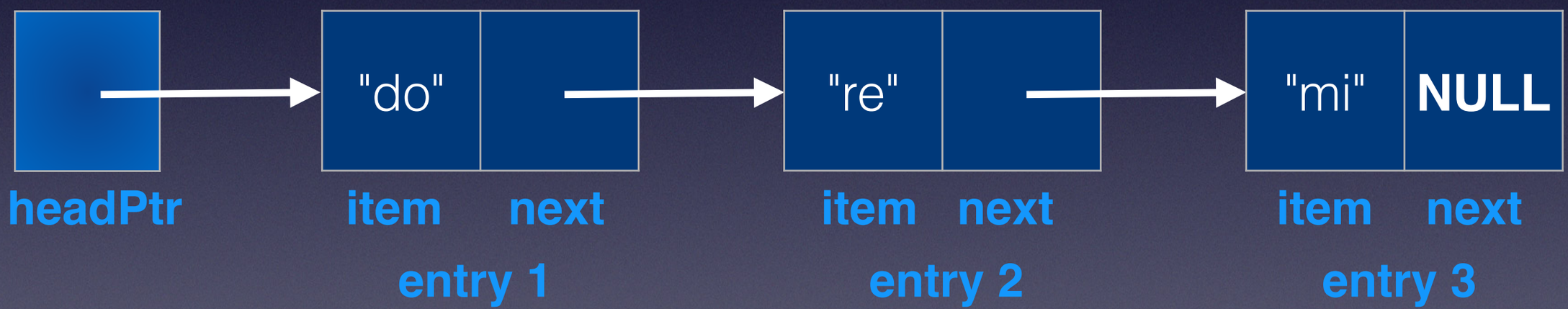
Linked Lists
getNodeAt, deletion

CS110C

Max Luttrell, CCSF

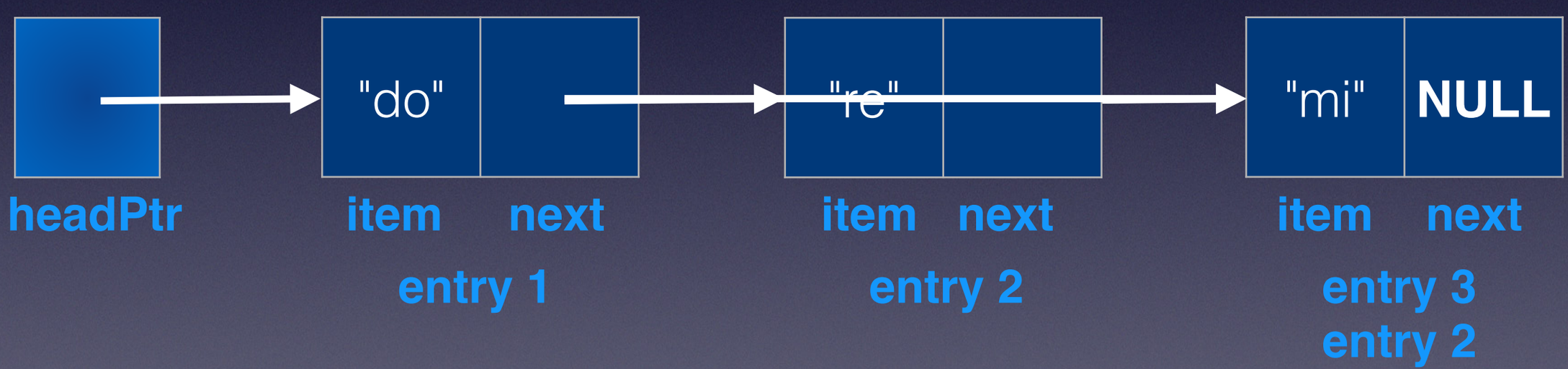
getNodeAt

- How can we get the data at position 2 in our list? at position n?



deletion

- How can we delete the item at position 2? position n?



node class

```
template<class ItemType>
class Node
{
private:
   ItemType          item; // A data item
   Node<ItemType>* next; // Pointer 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
```

linked list class

```
template<class ItemType>
class LinkedList : public ListInterface<ItemType>
{
private:
   Node<ItemType>* headPtr; // Pointer to first node in the chain;
                            // (contains the first entry in the list)
   int itemCount;           // Current count of list items

   // Locates a specified node in this linked list.
   Node<ItemType>* getNodeAt(int position) const;

public:
   LinkedList();
   LinkedList(const LinkedList<ItemType>& aList);
   virtual ~LinkedList();

   bool isEmpty() const;
   int getLength() const;
   bool insert(int newPosition, const ItemType& newEntry);
   bool remove(int position);
   void clear();

   ItemType getEntry(int position) const throw(PrecondViolatedExcep);
   void setEntry(int position, const ItemType& newEntry)
                               throw(PrecondViolatedExcep);
}; // end LinkedList
```

getNodeAt()

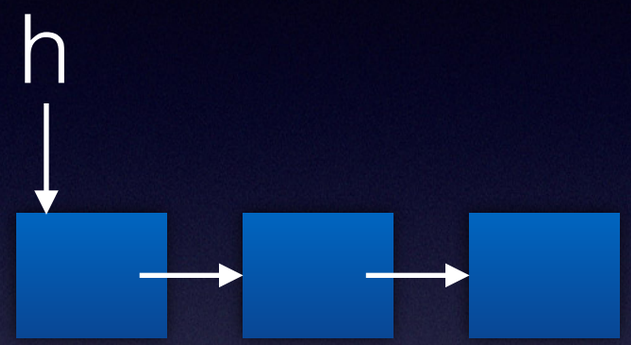
```
template<class ItemType>
Node<ItemType>* LinkedList<ItemType>::getNodeAt(int position) const
{
   // Debugging check of precondition
   assert( (position >= 1) && (position <= itemCount) );

   // Count from the beginning of the chain
   Node<ItemType>* curPtr = headPtr;
   for (int skip = 1; skip < position; skip++)
      curPtr = curPtr->getNext();

   return curPtr;
}  // end getNodeAt
```

remove()

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```

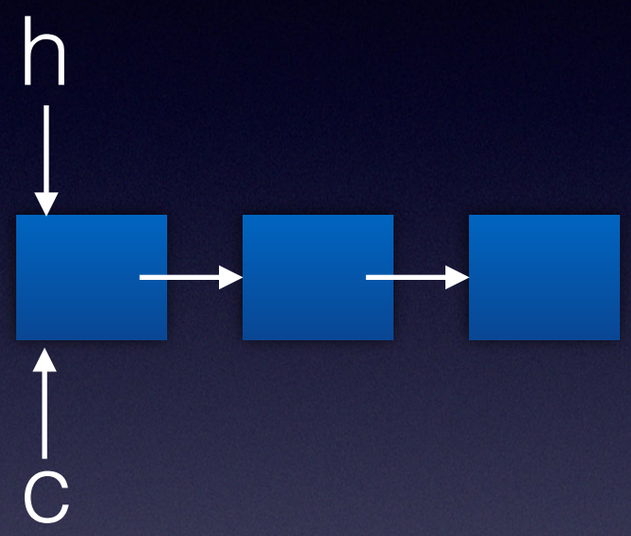
example: remove(1)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



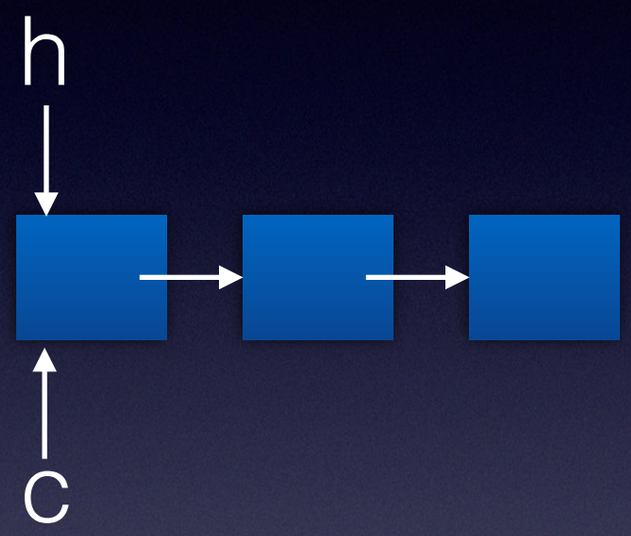
example: remove(1)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



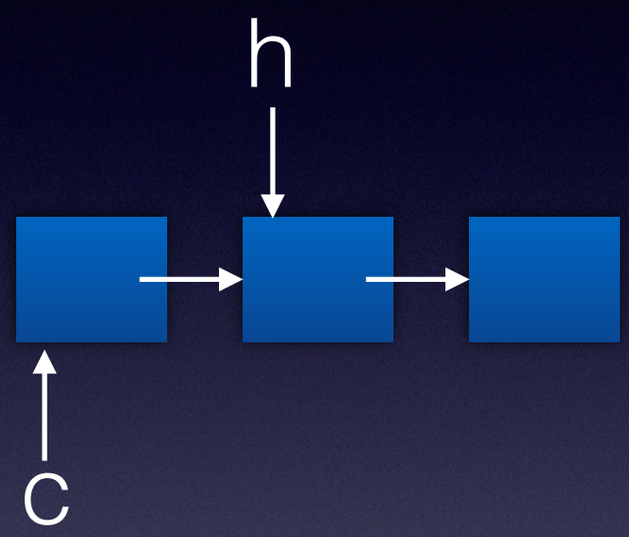
example: remove(1)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



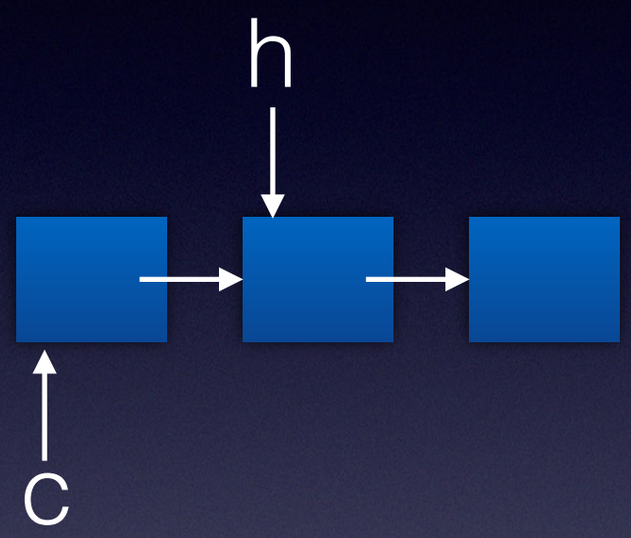
example: remove(1)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



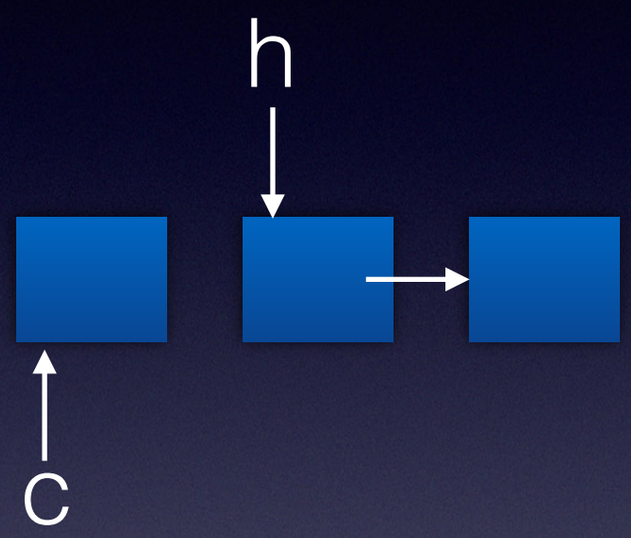
example: remove(1)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



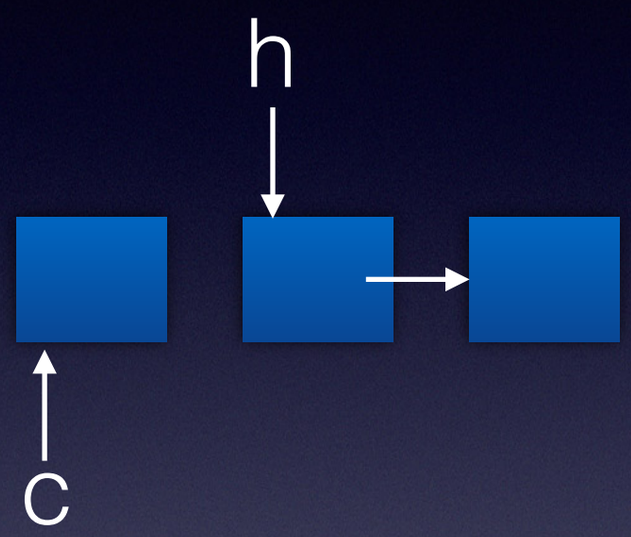
example: remove(1)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



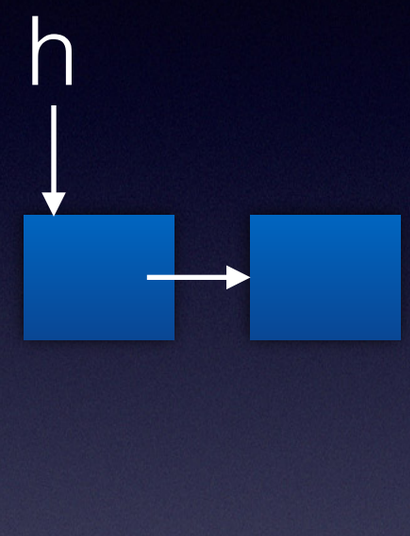
example: remove(1)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



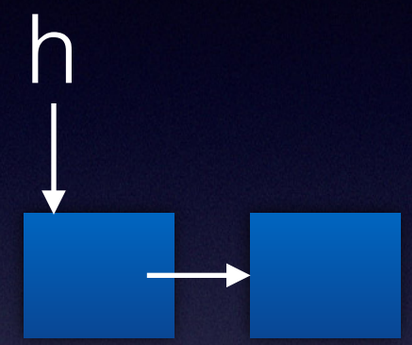
example: remove(1)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



example: remove(1)

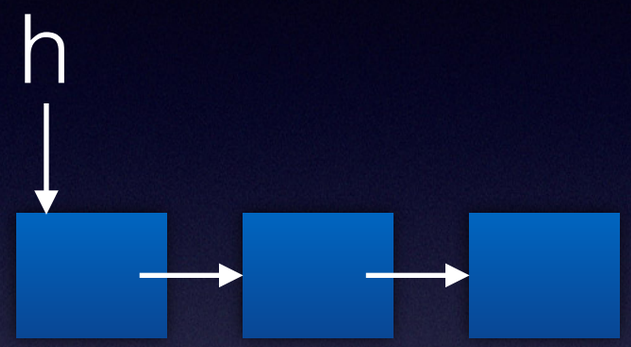
```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



C

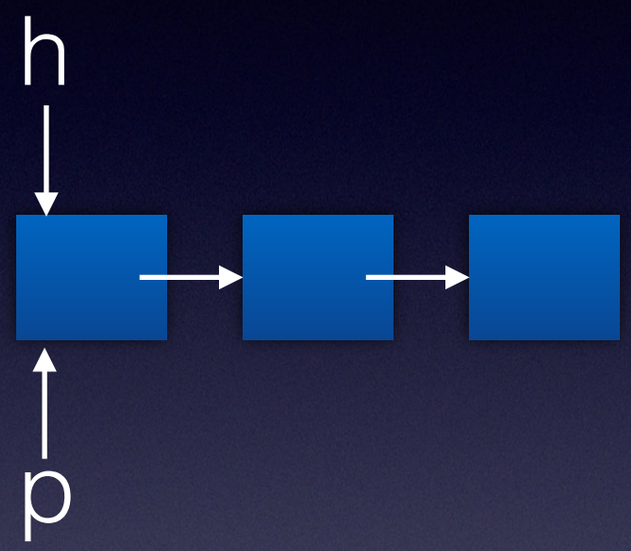
example: remove(2)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



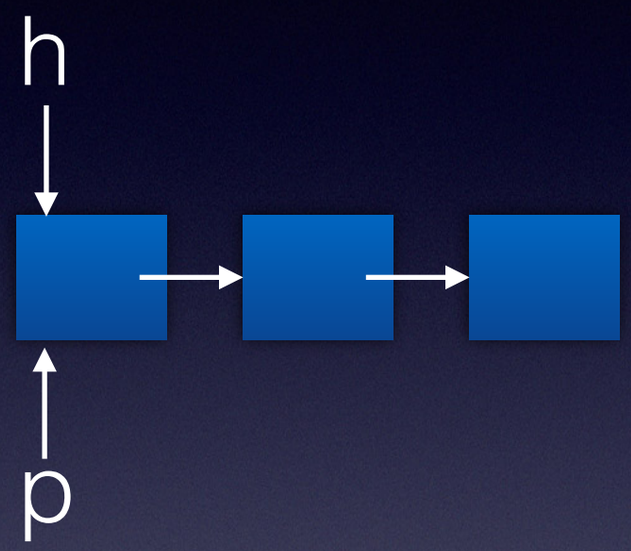
example: remove(2)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



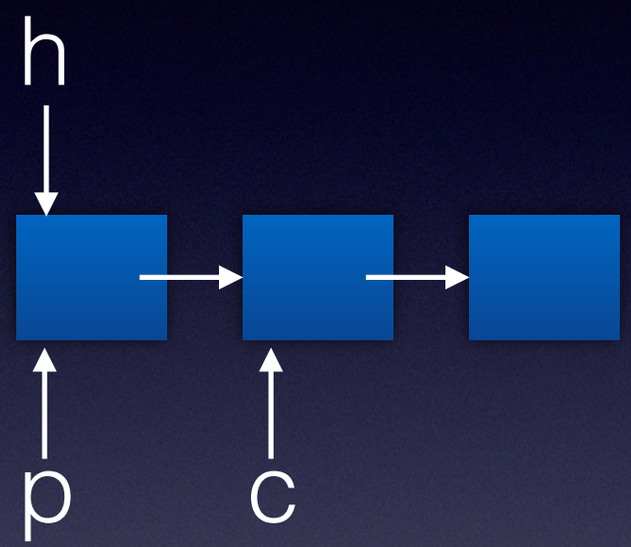
example: remove(2)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



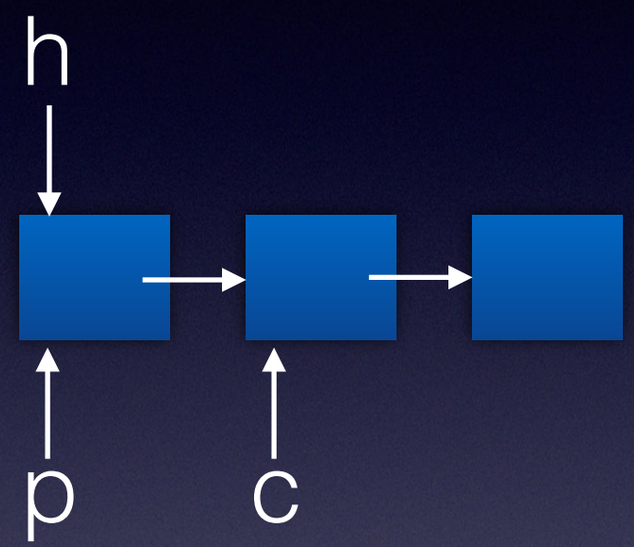
example: remove(2)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



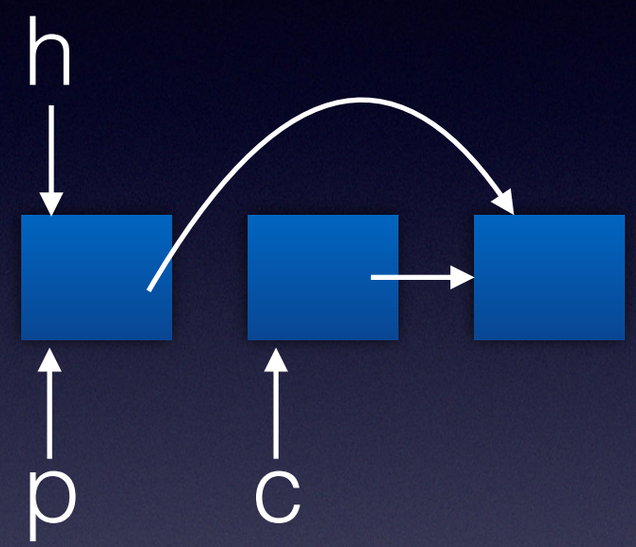
example: remove(2)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



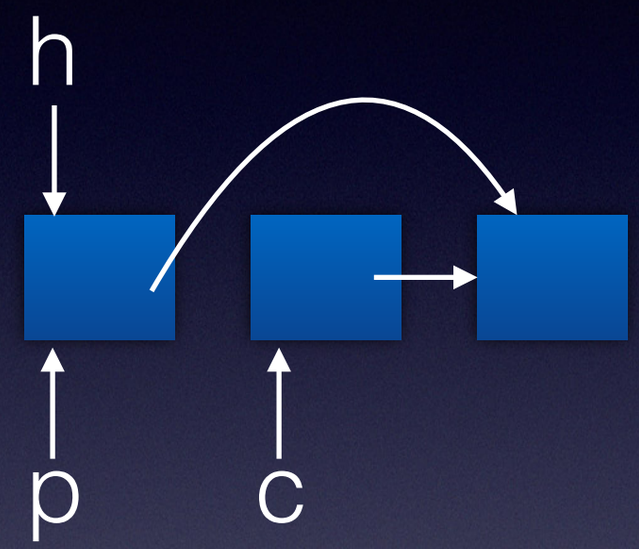
example: remove(2)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



example: remove(2)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```



example: remove(2)

```
bool LinkedList<ItemType>::remove(int position)
{
   bool ableToRemove = (position >= 1) && (position <= itemCount);
   if (ableToRemove)
   {
      Node<ItemType>* curPtr = nullptr;
      if (position == 1)
      {
         // Remove the first node in the chain
         curPtr = headPtr; // Save pointer to node
         headPtr = headPtr->getNext();
      }
      else
      {
         // Find node that is before the one to delete
         Node<ItemType>* prevPtr = getNodeAt(position - 1);

         // Point to node to delete
         curPtr = prevPtr->getNext();

         // Disconnect indicated node from chain by connecting the
         // prior node with the one after
         prevPtr->setNext(curPtr->getNext());
      }  // end if

      // Return node to system
      curPtr->setNext(nullptr);
      delete curPtr;
      curPtr = nullptr;

      itemCount--;  // Decrease count of entries
   }  // end if

   return ableToRemove;
}  // end remove
```