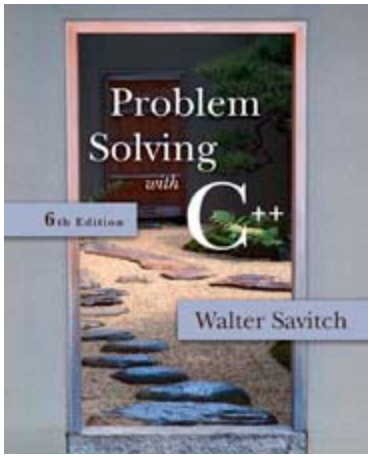
APS105: Lecture 30

Wael Aboelsaadat

wael@cs.toronto.edu

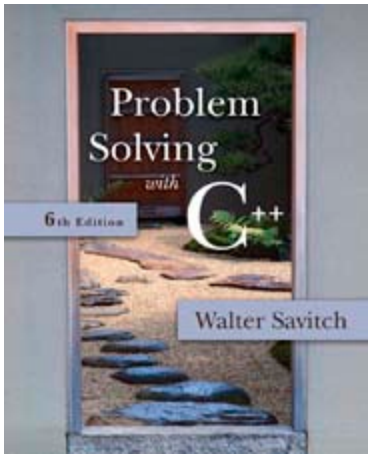
http://ccnet3.utoronto.ca/20079/aps105h1f/

Acknowledgement: These slides are a modified version of the text book slides as supplied by Addison Wesley



Chapter 14

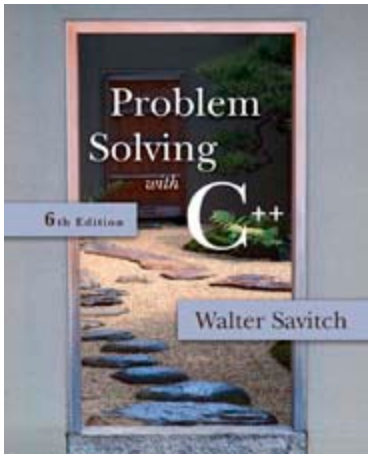
Recursion



```
void MergeSort(int ar[], int left, int right, int pivot)
{
    if(left == right)
        return;
    else
    {
        MergeSort(ar, left, pivot, (left + pivot) / 2);
        MergeSort(ar, pivot + 1, right, (pivot + right + 1) / 2);
    }

    int LeftIndex  = left,
        PivotIndex = pivot + 1;
    while(PivotIndex != right + 1 && LeftIndex != PivotIndex) //continue until either list runs out
    {
        if(ar[PivotIndex] <= ar[LeftIndex])
        {
            int i;
            int iSrc = PivotIndex;
            int iDest= LeftIndex;
            int StoreSrc = ar[iSrc];
            for(i = iSrc; i > iDest; i --)
                ar[i] = ar[i - 1];     // Shifts numbers from iDest to iSrc one step forward
            ar[iDest] = StoreSrc;      // Puts final element in the right place
            PivotIndex++;
            LeftIndex++;
        }
        else
            LeftIndex++; // Skip to the next element
    }
}
```

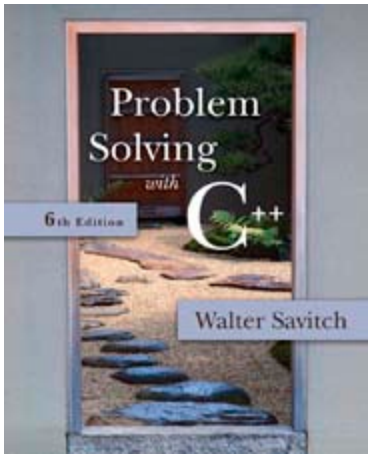
Chapter 13

Pointers and Linked Lists



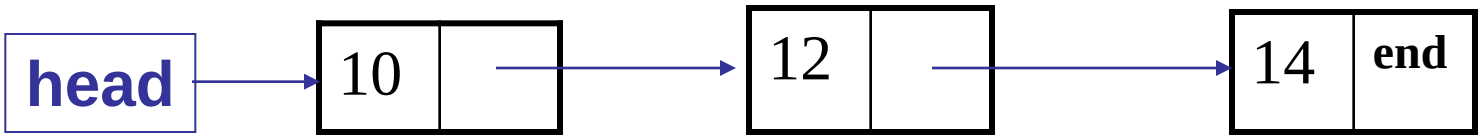
13.1

Nodes and Linked Lists



Nodes and Linked Lists

- A linked list is a list that can grow and shrink while the program is running
- A linked list is constructed using pointers
- A linked list often consists of structs that contain a pointer variable connecting them to other dynamic variables



Nodes

- The boxes in the previous drawing represent the nodes of a linked list
  - Nodes contain the data item(s) and a pointer that can point to another node of the same type
    - The pointers point to the entire node, not an individual item that might be in the node
- The arrows in the drawing represent pointers

Display 13.1

Implementing Nodes

- Nodes are implemented in C++ as structs or classes
  - Example: A structure to store two data items and a pointer to another node of the same type, along with a type definition might be:

```
struct ListNode
{
    string item;
    int count;
    ListNode *link;
 };

typedef ListNode* ListNodePtr;
```

This circular definition is allowed in C++

The head of a List

- The box labeled head, in display 13.1, is not a node, but a pointer variable that points to a node
- Pointer variable head is declared as:

```
ListNodePtr head;
```

Accessing Items in a Node

- Using the diagram of 13.1, this is one way to change the number in the first node from 10 to 12:

```
(*head).count = 12;
```

  - head is a pointer variable so *head is the node that head points to
  - The parentheses are necessary because the dot operator . has higher precedence than the dereference operator *

The Arrow Operator

- The arrow operator -> combines the actions of the dereferencing operator * and the dot operator to specify a member of a struct or object pointed to by a pointer
  - (*head).count = 12;
    can be written as
    head->count = 12;
  - The arrow operator is more commonly used

Display 13.2

NULL

- The defined constant NULL is used as…
  - An end marker for a linked list
    - A program can step through a list of nodes by following the pointers, but when it finds a node containing NULL, it knows it has come to the end of the list
  - The value of a pointer that has nothing to point to
- The value of NULL is 0
- Any pointer can be assigned the value NULL:

```
double* there = NULL;
```

To Use NULL

- A definition of NULL is found in several libraries, including <iostream> and <cstddef>
- A using directive is not needed for NULL

Linked Lists

- The diagram in Display 13.2 depicts a linked list
- A linked list is a list of nodes in which each node has a member variable that is a pointer that points to the next node in the list
  - The first node is called the head
  - The pointer variable head, points to the first node
    - The pointer named head is not the head of the list…it points to the head of the list
  - The last node contains a pointer set to NULL

Building a Linked List: The node definition

- Let's begin with a simple node definition:

```
struct Node
 {
    int data;
    Node *link;
 };

 typedef Node* NodePtr;
```

Building a Linked List: Declaring Pointer Variable head

- With the node defined and a type definition to make or code easier to understand, we can declare the pointer variable head:

```
NodePtr head;
```

  - head is a pointer variable that will point to the head node when the node is created

Building a Linked List: Creating the First Node

- To create the first node, the operator new is used to create a new dynamic variable:

```
head = new Node;
```

  - Now head points to the first, and only, node in the list

Building a Linked List: Initializing the Node

- Now that head points to a node, we need to give values to the member variables of the node:

```
head->data = 3;
head->link = NULL;
```

  - Since this node is the last node, the link is set to NULL

Function head_insert

- It would be better to create a function to insert nodes at the head of a list, such as:
  - void head_insert(NodePtr& head, int the_number);
    - The first parameter is a NodePtr parameter that points to the first node in the linked list
    - The second parameter is the number to store in the list
  - head_insert will create a new node for the number
    - The number will be copied to the new node
    - The new node will be inserted in the list as the new head node

Pseudocode for head_insert

- Create a new dynamic variable pointed to by temp_ptr
- Place the data in the new node called *temp_ptr
- Make temp_ptr's link variable point to the head node
- Make the head pointer point to temp_ptr

Display 13.3

Translating head_insert to C++

- The pseudocode for head_insert can be written in C++ using these lines in place of the lines of pseudocode:
  - NodePtr temp_ptr; //create the temporary pointer
    temp_ptr = new Node; // create the new node
  - temp_ptr->data = the_number; //copy the number
  - temp_ptr->link = head; //new node points to first node
    head = temp_ptr; // head points to new
    // first node

Display 13.4

An Empty List

- A list with nothing in it is called an empty list
- An empty linked list has no head node
- The head pointer of an empty list is NULL

head = NULL;

  - Any functions written to manipulate a linked list should check to see if it works on the empty list

Losing Nodes

- You might be tempted to write head_insert using the head pointer to construct the new node:

```
head = new Node;
head->data = the_number;
```

- Now to attach the new node to the list
  - The node that head used to point to is now lost!

Display 13.5

Memory Leaks

- Nodes that are lost by assigning their pointers a new address are not accessible any longer
- The program has no way to refer to the nodes and cannot delete them to return their memory to the freestore
- Programs that lose nodes have a memory leak
  - Significant memory leaks can cause system crashes

Searching a Linked List

- To design a function that will locate a particular node in a linked list:
  - We want the function to return a pointer to the node so we can use the data if we find it, else return NULL
  - The linked list is one argument to the function
  - The data we wish to find is the other argument
  - This declaration will work:

```
NodePtr search(NodePtr head, int target);
```

Function search

- Refining our function
  - We will use a local pointer variable, named here, to move through the list checking for the target
    - The only way to move around a linked list is to follow pointers
  - We will start with here pointing to the first node and move the pointer from node to node following the pointer out of each node

Display 13.6

Pseudocode for search

- Make pointer variable here point to the head node
- while(here does not point to a node containing target
  AND here does not point to the last node)
  {
  make here point to the next node
  }
- If (here points to a node containing the target)
  return here;
  else
  return NULL;

Moving Through the List

- The pseudocode for search requires that pointer here step through the list
  - How does here follow the pointers from node to node?
  - When here points to a node, here->link is the address of the next node
  - To make here point to the next node, make the assignment:

```
here = here->link;
```

A Refinement of search

- The search function can be refined in this way:

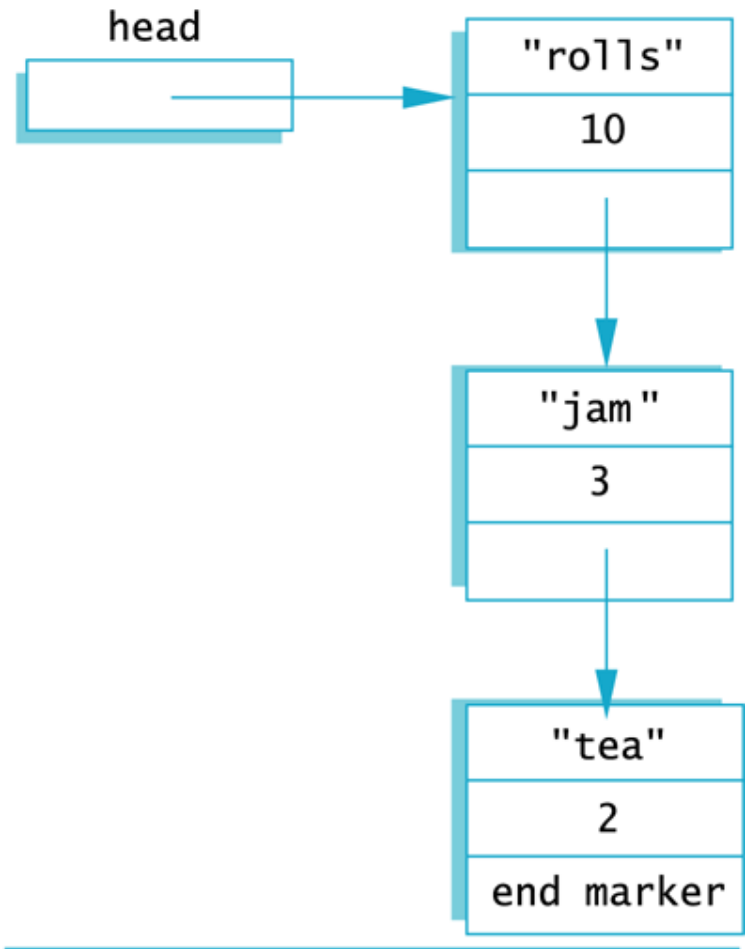
```
here = head;
while(here->data != target && here->link !=
NULL)
  {
       here = here->next;
   }
  if (here->data = = target)
      return here;
  else
      return NULL;
```

Searching an Empty List

- Our search algorithm has a problem
  - If the list is empty, here equals NULL before the while loop so…
    - here->data is undefined
    - here->link is undefined
  - The empty list requires a special case in our search function
  - A refined search function that handles an empty list is shown in **Display 13.7**

Display 13.1

Nodes and Pointers

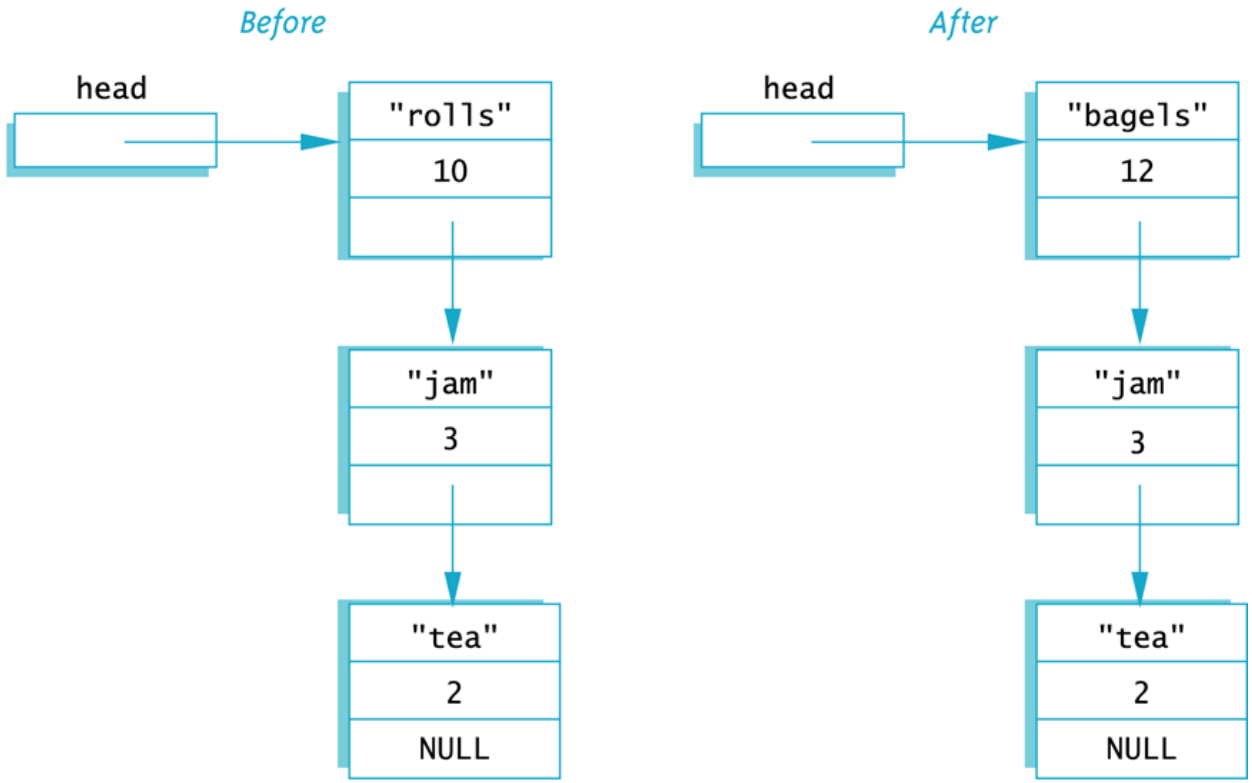


Display 13.2

Back Next

Accessing Node Data

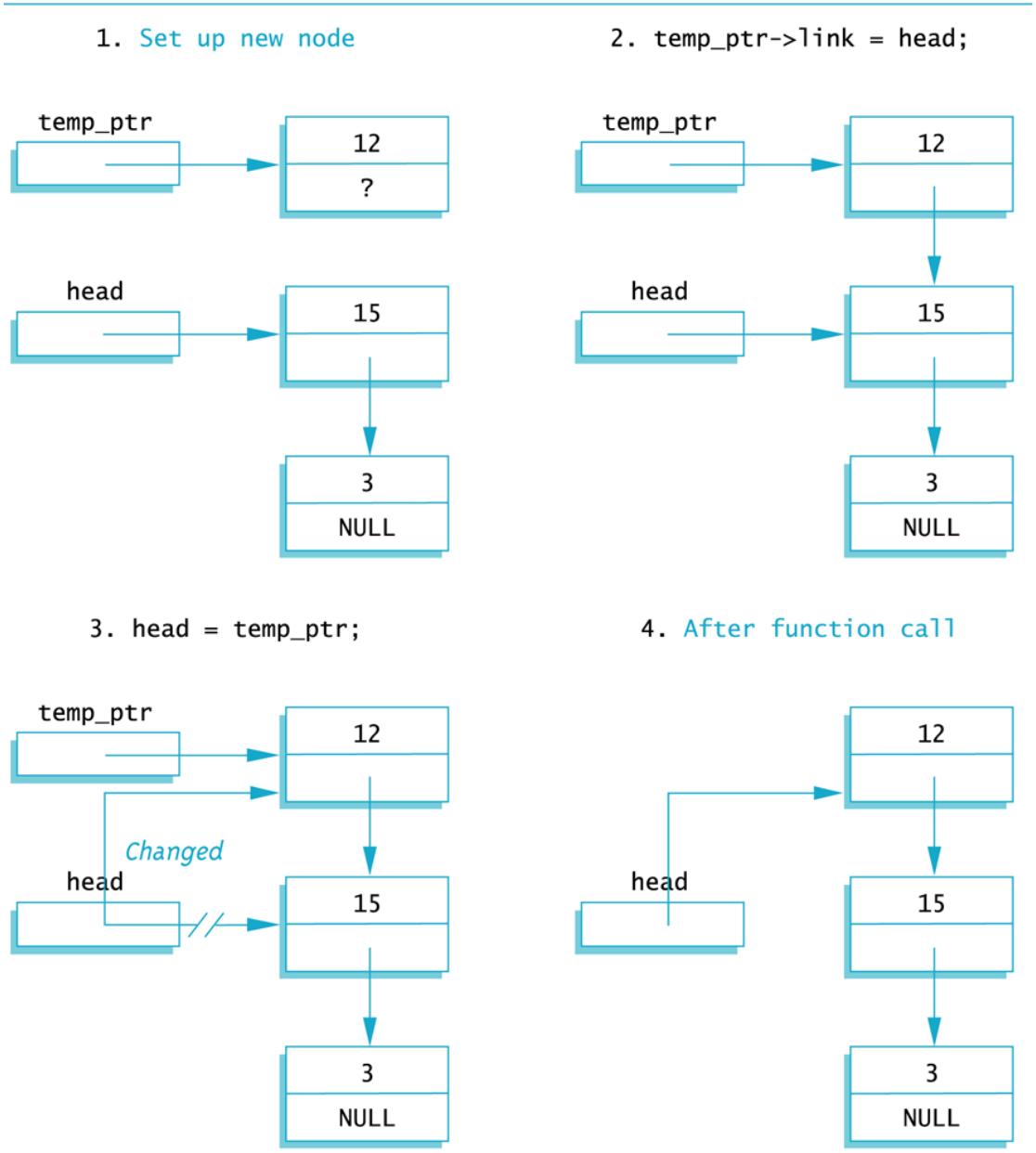
```
head->count = 12;
head->item = "bagels";
```



Display 13.3

Back Next

Adding a Node to a Linked List



Display 13.4

Function to Add a Node at the Head of a Linked List

Function Declaration

```
struct Node
{
    int data;
    Node *link;
};

typedef Node* NodePtr;

void head_insert(NodePtr& head, int the_number);
//Precondition: The pointer variable head points to
//the head of a linked list.
//Postcondition: A new node containing the_number
//has been added at the head of the linked list.
```

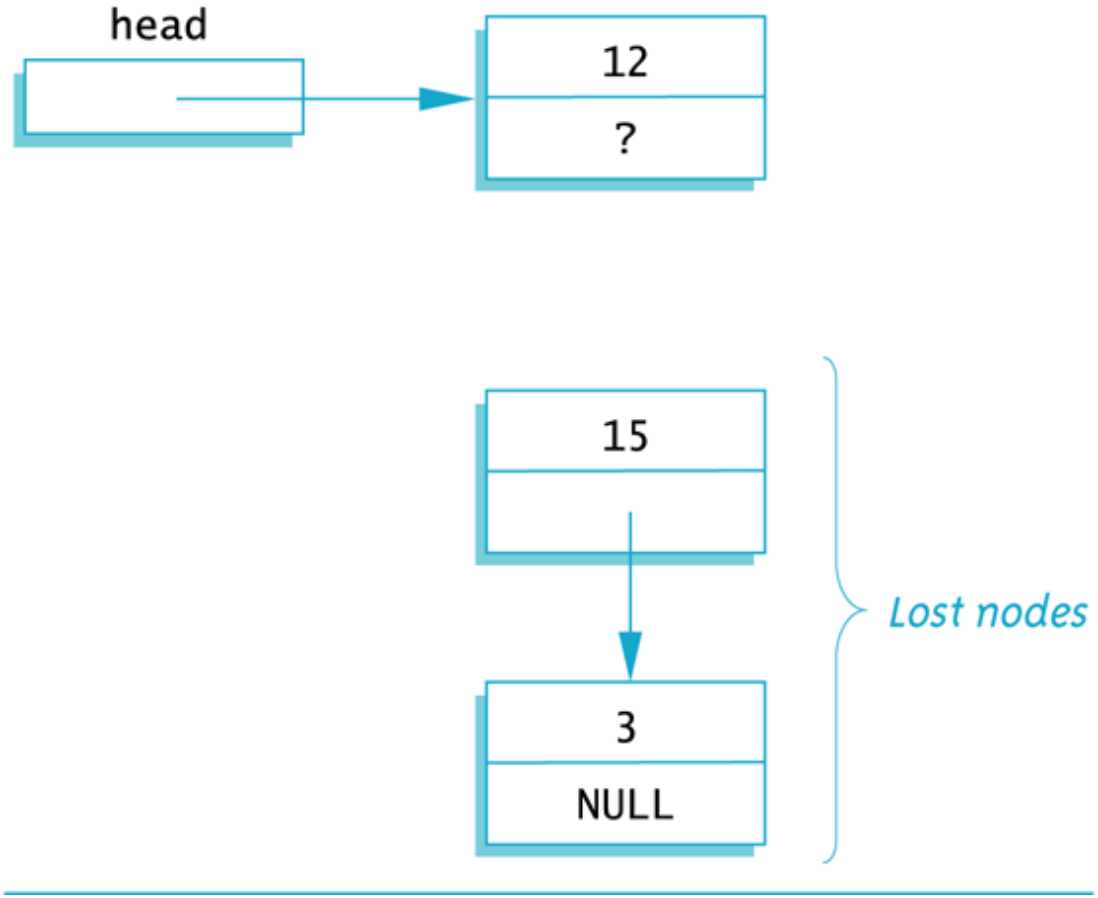
Function Definition

```
void head_insert(NodePtr& head, int the_number)
{
    NodePtr temp_ptr;
    temp_ptr = new Node;

    temp_ptr->data = the_number;

    temp_ptr->link = head;
    head = temp_ptr;

}
```

Display 13.5

Back Next

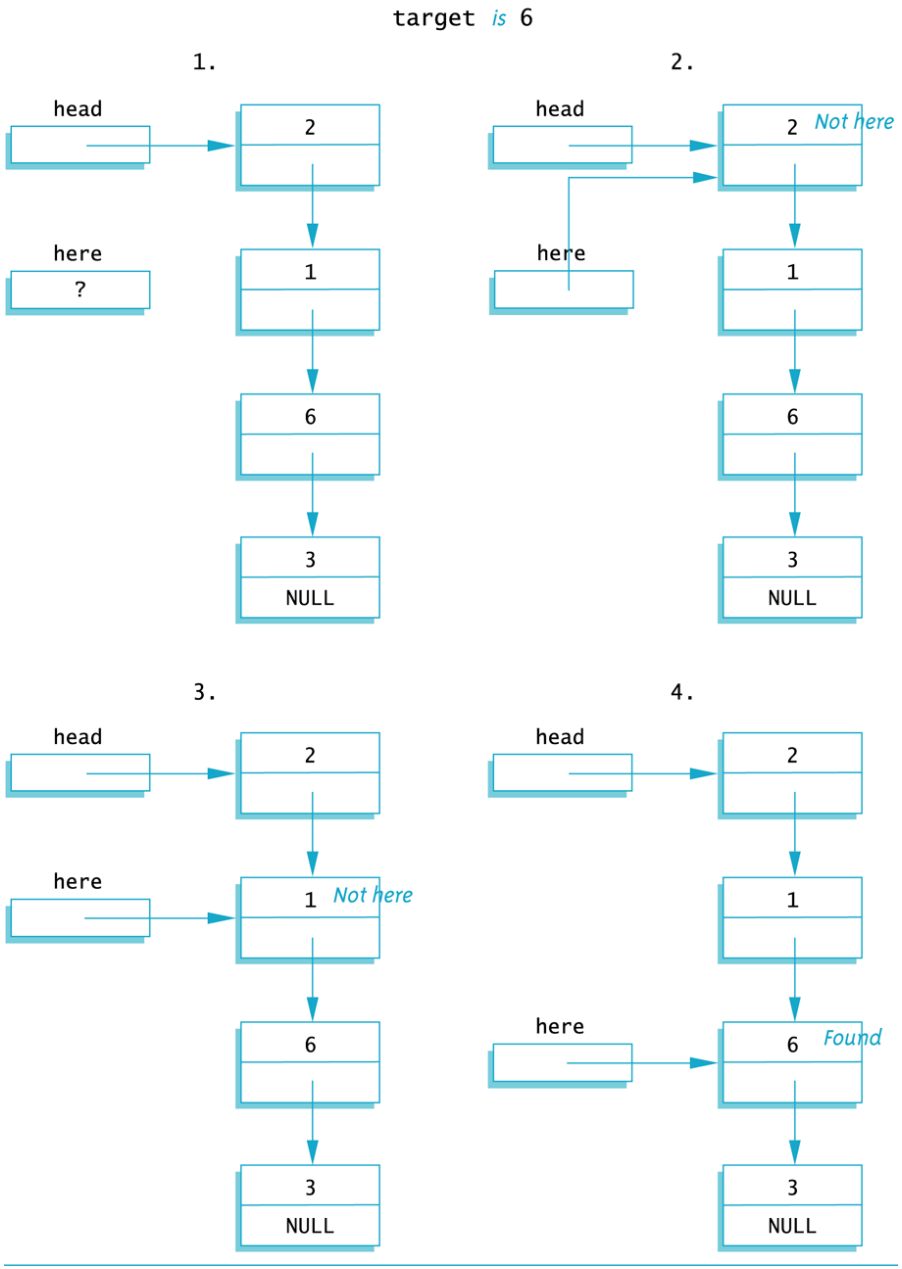
Lost Nodes



Display 13.6

Back Next

Searching a Linked List



Display 13.7

Back Next

Function to Locate a Node in a Linked List

Function Declaration

```
struct Node
{
    int data;
    Node *link;
};

typedef Node* NodePtr;

NodePtr search(NodePtr head, int target);
//Precondition: The pointer head points to the head of
//a linked list. The pointer variable in the last node
//is NULL. If the list is empty, then head is NULL.
//Returns a pointer that points to the first node that
//contains the target. If no node contains the target,
//the function returns NULL.
```

Function Definition

```
//Uses cstddef:
NodePtr search(NodePtr head, int target)
{
    NodePtr here = head;

    if (here == NULL)
    {
        return NULL;
    }
    else
    {
        while (here->data != target &&
                                    here->link != NULL)
            here = here->link;

        if (here->data == target)
            return here;
        else
            return NULL;
    }
}
```

Empty list case