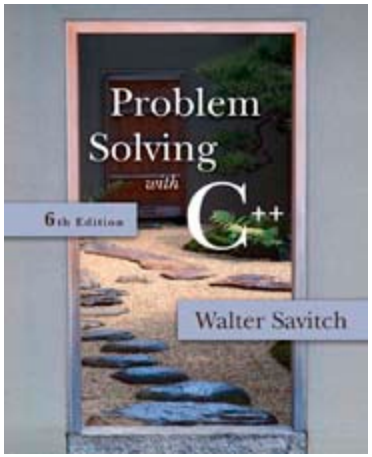
APS105: Lecture 31A

Wael Aboelsaadat

wael@cs.toronto.edu

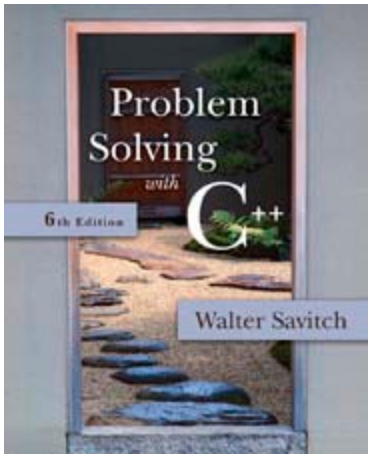
http://ccnet3.utoronto.ca/20079/aps105h1f/

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



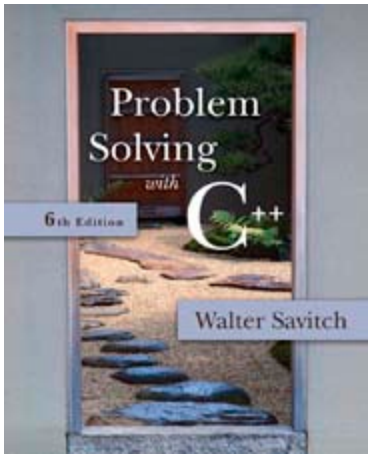
Chapter 13

Pointers and Linked Lists



13.1

Nodes and Linked Lists



Inserting a Node Inside a List

- To insert a node after a specified node in the linked list:
  - Use another function to obtain a pointer to the node after which the new node will be inserted
    - Call the pointer after_me
  - Use function insert, declared here to insert the node:

```
void insert(NodePtr after_me, int the_number);
```

Display 13.8

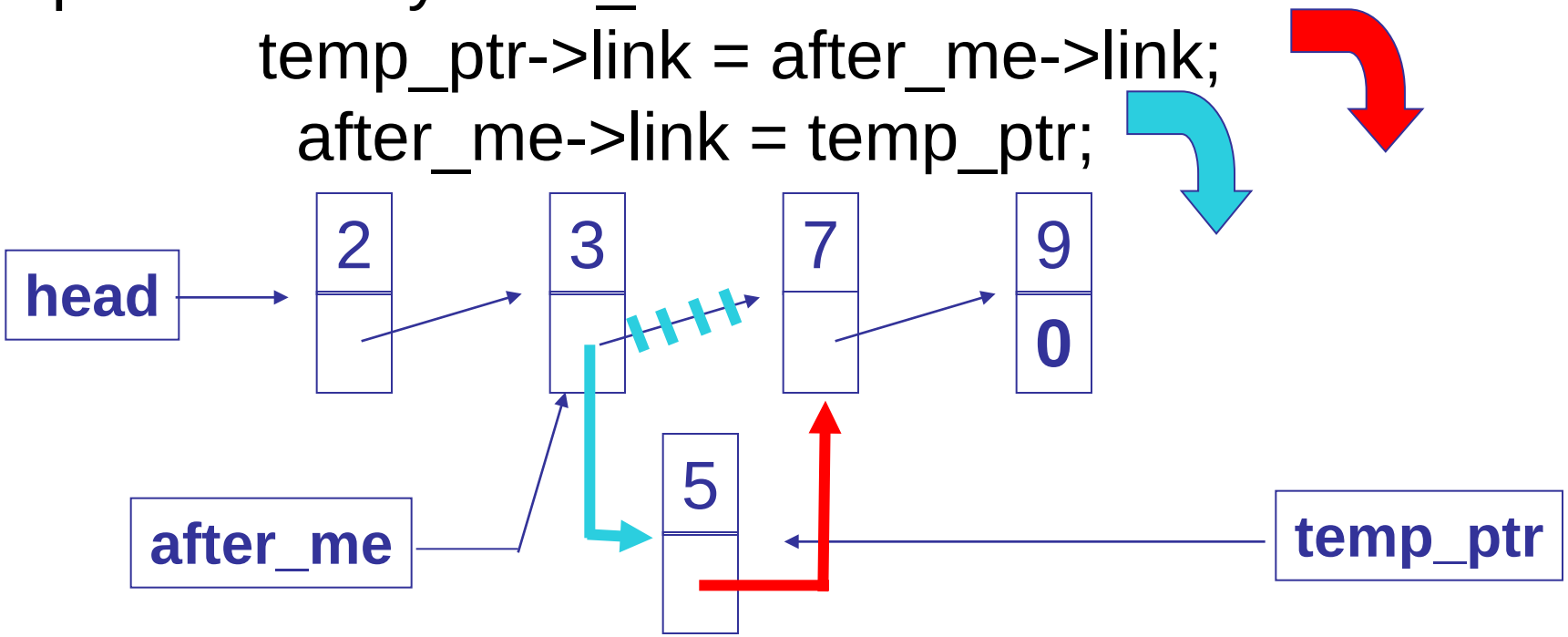
Inserting the New Node

- Function insert creates the new node just as head_insert did
- We do not want our new node at the head of the list however, so…
  - We use the pointer after_me to insert the new node

Inserting the New Node

- This code will accomplish the insertion of the new node, pointed to by temp_ptr, after the node pointed to by after_me:

```
temp_ptr->link = after_me->link;
after_me->link = temp_ptr;
```



Caution!

- The order of pointer assignments is critical
  - If we changed after_me->link to point to temp_ptr first, we would loose the rest of the list!

- The complete insert function is shown in **Display 13.9**

Function insert Again

- Notice that inserting into a linked list requires that you only change two pointers
  - This is true regardless of the length of the list
  - Using an array for the list would involve copying as many as all of the array elements to new locations to make room for the new item
- Inserting into a linked list is often more efficient than inserting into an array

Removing a Node

- To remove a node from a linked list
  - Position a pointer, before, to point at the node prior to the node to remove
  - Position a pointer, discard, to point at the node to remove
  - Perform: before->link = discard->link;
    - The node is removed from the list, but is still in memory
  - Return *discard to the freestore: delete discard;

Display 13.10

AssignmentWith Pointers

- If head1 and head2 are pointer variables and head1 points to the head node of a list:

  head2 = head1;

  causes head2 and head1 to point to the same list
  - There is only one list!
- If you want head2 to point to a separate copy, you must copy the list node by node or overload the assignment operator appropriately

Pointers as Iterators

- An iterator is a construct that allows you to cycle through the data items in a data structure to perform an action on each item
  - An iterator can be an array index, or simply a pointer
- A general outline using a pointer as an iterator:

```
Node_Type *iter;
  for (iter = Head; iter != NULL; iter = iter->Link)
  //perform the action on the node iter points to
```

  - Head is a pointer to the head node of the list

Iterator Example

- Using the previous outline of an iterator we can display the contents of a linked list in this way:

```
NodePtr iter;
    for (iter = Head; iter != NULL; iter = iter->Link)
        cout << (iter->data);
```

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

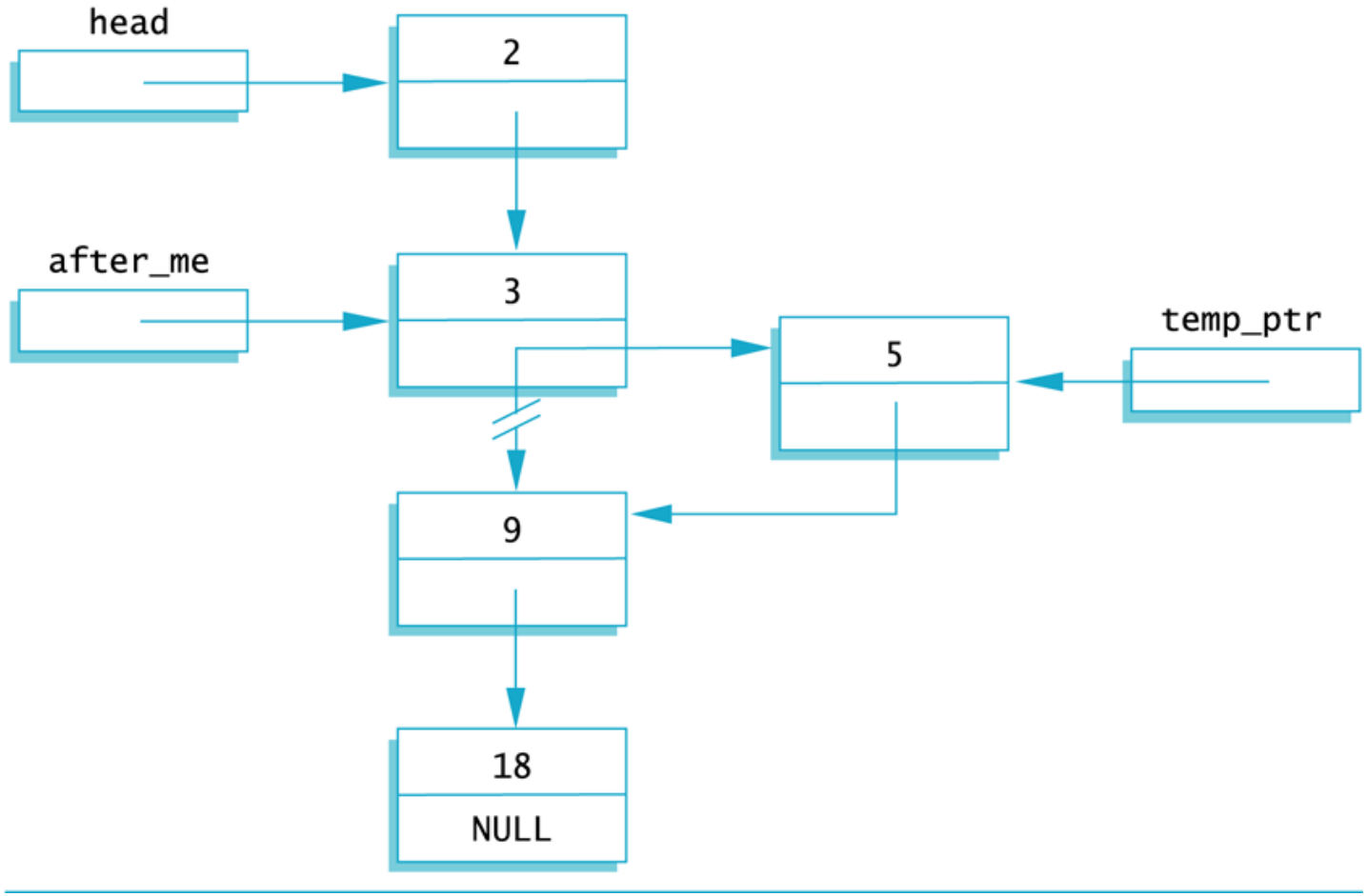
```
//Uses cstddef:
NodePtr search(NodePtr head, int target)
{
    NodePtr here = head;

    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

Display 13.8

Back Next

Inserting in the Middle of a Linked List



Display 13.9

Function to Add a Node in the Middle of a Linked List

Function Declaration

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

typedef Node* NodePtr;

void insert(NodePtr after_me, int the_number);
//Precondition: after_me points to a node in a linked
//list.
//Postcondition: A new node containing the_number
//has been added after the node pointed to by after_me.
```

Function Definition

```
void insert(NodePtr after_me, int the_number)
{
    NodePtr temp_ptr;
    temp_ptr = new Node;

    temp_ptr->data = the_number;

    temp_ptr->link = after_me->link;
    after_me->link = temp_ptr;
}
```

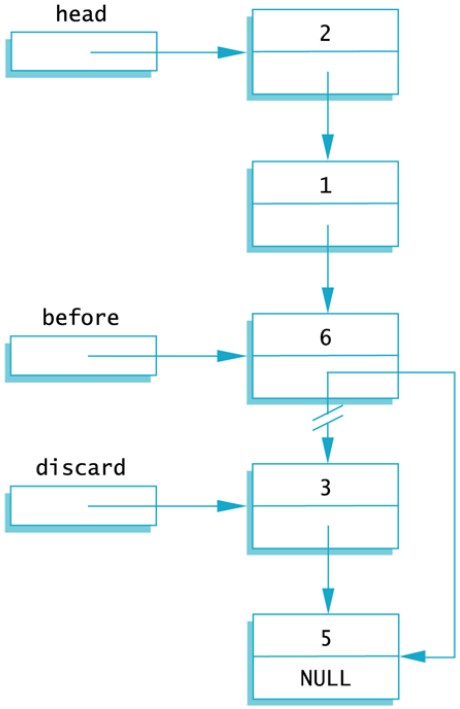
Display 13.10

Back Next

Removing a Node

1. Position the pointer `discard` so that it points to the node to be deleted, and position the pointer `before` so that it points to the node before the one to be deleted.

2. `before->link = discard->link;`



3. *delete* `discard;`