Programming techniques

Week 7 - Recursion (cont)
Agenda

- Problem solving with recursion
- Work through examples to get used to the recursive process

CuuDuongThanCong.com
https://fb.com/tailieudientucntt
Using Recursion

- Today we will walk through examples solving problems with recursion

- To get used to this process
  - we will select simple problems that in reality should be solved using iteration and not recursion
  - but, it should give you an understanding of how to design using recursion
  - which we will need to understand for CS163
Example #1

- First, let’s display the contents of a linear linked list, recursively
  - obviously this is **should** be done iteratively!
  - but, as an exercise determine what the stopping condition should be first:
    - when the head pointer is NULL
    - what should be done when this condition is reached? return
    - what should be done otherwise? display and call the function recursively
Example #1

☐ If we were to do this iteratively:

```c++
void display(node * head) {
    while (head) {
        cout << head->data->title << endl;
        head = head->next;
    }
}
```

☐ Why is it ok in this case to change head?

☐ Look at the stopping condition

- with recursion we will replace the while with an if....and replace the traversal with a function call
Example #1

☐ If we were to do this recursively:

```cpp
void display(node * head) {

    if (head) {
        cout << head->data->title << endl;
        display(head->next);
    }
}

☐ Now, change this to display the list backwards (recursively)

☐ Discuss the code you’d need to do THAT recursively....
Example #2

- Next, let’s insert at the end of a linear linked list, recursively
  - again this is **should** be done iteratively!
  - but, as an exercise determine what the stopping condition should be first:
    - when the head pointer is NULL
    - what should be done when this condition is reached? **allocate memory and save the data**
    - what should be done otherwise? call the function recursively **with the next ptr**
Example #2

- If we were to do this iteratively:

```cpp
void append(node * & head, const video & d) {
    if (!head) {
        head = new node;
        head->data = ••• //save the data
        head->next = NULL;
    } else {
        node * current = head;
        while (current->next) {
            current = current->next;
        }
        current->next = new node;
        current = current->next;
        current->data = ••• //save the data
        current->next = NULL;
    }
}
```
Example #2

If we were to do this recursively:

```c
void append(node * & head, const video & d) {
    if (!head) {
        head = new node;
        head->data = ... //save the data
        head->next = NULL;
    } else
        append(head->next, d);
}
```
Example #2

- Notice this is much shorter (but less efficient)
- Notice the stopping condition (!head)
- Examine how the pass by reference can be used to implicitly connect up the nodes
- Walk thru an example of invoking this function
Example #2

- This can also be done recursively by using the returned value (rather than call by reference):

```c
node * append(node * head, const video & d) {
    if (!head) {
        head = new node;
        head->data = ••• //save the data
        head->next = NULL;
    } else
        head->next = append(head->next, d);
    return head;
}
```

- Notice the function call must use the returned value
- Here, we are explicitly connecting up the nodes
- Walk thru an example of invoking this function
Example #3

- Next, let’s remove an item from a linear linked list, recursively
  - again this is **should** be done iteratively!
  - but, as an exercise determine what the stopping condition should be first:
    - when the head pointer is NULL
    - when a match (the item to be removed) is found
  - what should be done when this condition is reached? **deallocate memory**
  - what should be done otherwise? call the function recursively **with the next ptr**
Example #3

- If we were to do this recursively:

  ```c
  int remove(node * & head, const video & d) {
    if (!head) return 0;  //match not found!
    if (strcmp(head->data->title, d->title)==0) {
      delete [] head->data->title;
      delete head->data;
      delete head;
      head = NULL;
      return 1;
    } return remove(head->next,d);
  }
  ```

- Does this reconnect the nodes?
- How does it handle the special cases of a) empty list, b) deleting the first item, c) deleting elsewhere
More Examples

- Now in class, let’s design and implement the following **recursively**
  - count the number of items in a linear linked list
  - delete all nodes in a linear linked list

- Why would recursion **not** be the proper solution for push, pop, enqueue, dequeue?
More Examples

What is the output for the following program fragment? called: $f(5)$

```c
int f(int n) {
    cout << n << endl;
    if (n == 0) return 4;
    else if (n == 1) return 2;
    else if (n == 2) return 3;
    n = f(n - 2) * f(n - 4);
    cout << n << endl;
    return n;
}
```
More Examples

What is the output of the following program or write INFINITE if there are indefinite recursive calls? called:

```cpp
int watch(int n) {
    if (n > 0)
        return n;
    cout << n << endl;
    return watch(n + 2) * 2;
}
```
For Next Time

☐ Practice Recursion

☐ Do the following:

- Make a copy of a linear linked list, recursively
- Merge two sorted linear linked lists, keeping the result sorted, recursively