Programming techniques

Week 8 - Recursion (cont)

04/2023

Agenda

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

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 <u>iteration</u> 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 <u>should</u> 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:

```
void display(Node* pHead) {
  while (pHead) {
    cout << pHead->data->title << endl;
    pHead = pHead->pNext;
  }
}
```

- 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:

```
void display(Node* pHead) {
  if (pHead) {
    cout << pHead->data->title << endl;
    display(pHead->pNext);
  }
}
```

- 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 <u>should</u> 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? <u>**allocate memory and save the data**</u>
  - what should be done otherwise? call the function recursively <u>**with the next ptr**</u>

Example #2

- If we were to do this iteratively:

```
void append(Node* &pHead, const Video& d) {
  if (!pHead) {
    pHead = new Node;
    pHead->data = ••• //save the data
    pHead->pNext = nullptr;
  } else {
    Node* cur = pHead;
    while (cur->pNext)
      cur = cur->pNext;
    cur->pNext = new Node;
    cur = cur->pNext;
    cur->data = ••• //save the data
    cur->pNext = nullptr;
  }
}
```

Example #2

- If we were to do this recursively:

```
void append(Node* &pHead, const Video& d) {
  if (!pHead) {
    pHead = new Node;
    pHead->data = •••; //save the data
    pHead->pNext = nullptr;
  } else
    append(pHead->pNext, 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):

```
Node* append(Node* pHead, const Video& d) {
   if (!pHead) {
      pHead = new Node;
      pHead->data = •••; //save the data
      pHead->pNext = nullptr;
   } else
      pHead->pNext = append(pHead->pNext, d);
   return pHead;
}
```

- 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 <u>should</u> be done iteratively!
  - but, as an exercise determine what the stopping condition should be first:
    - when the pHead pointer is nullptr
    - when a match (the item to be removed) is found
  - what should be done when this condition is reached? **<u>deallocate memory</u>**
  - what should be done otherwise? call the function recursively **<u>with the next ptr</u>**

Example #3

- If we were to do this recursively:

```
int remove(Node* &pHead, const Video& d) {
  if (!pHead) return 0;  //match not found!
  if (strcmp(pHead->data->title, d->title)==0) {
    delete [] pHead->data->title;
    delete pHead->data;
    delete pHead;
    pHead = nullptr;
    return 1;
  } return remove(pHead->pNext,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)`

```
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:

```
cout << watch(-7)
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