Prelab 14: Algorithm Design & Recursion

Example 1

Using recursion, it's required to implement the function “calSum” which takes a list as an input parameter and returns the sum of all its elements. Check out the following example.

```python
def calSum(A):
    #TODO
    #TODO
    #TODO
def main():
    myList = [3, -1, 5, 7, 2, 4]
    print(calSum(myList))  # should print 20
main()
```

By recursion we mean to design a function that calls itself. You can think of recursion is dividing any problem into smaller versions of itself and trying to combine these pieces together. Consider the following implementation for “calSum” recursively. First let's assume we have N elements in the list, the idea is, what if someone told you not to worry about the sum of all elements from the second element (index 1) to the end of the list, he will somehow calculate it for you! then all you have to do to get the total sum is to add the first element (with index 0) to the sum you will be given i.e. total sum = first element + sum from second element to the end of the list. More concretely: total sum = A[0] + calSum(A[1:]). Now, when you tried to get sum of (A[1:]), that guy told you he can give you the sum of all elements starting from the third element (index 2) to the end of the list, so the same logic applies, total sum = A[0]+A[1]+calSum(A[2:]). That story kept repeating over and over until only one element is left in A (A[N-2:]). The rounds of that story is listed below:

1. total sum = A[0] + calSum(A[1:])
4. ...

We can note here that each round tries to simplify the sum by making smaller versions of the sum and combines the results of these small versions by adding them together.

The rounds stopped when there was only one element left in A because we cannot shrink A further, this is called the terminating condition in recursion, which basically tells the program when to stop making smaller versions of the problem. You can see the terminating condition in the code.

Each execution of the recursive call can be mapped to one of the rounds.

```python
def calSum(A):
    if(len(A)==1):  # terminating condition
        return A[0]  # terminating condition
```
return A[0] + calSum(A[1:]) # recursive call

Now we present, the execution rounds of this function starting from the call in the main()

```
myList=[3, -1, 5, 7, 2, 4]
# first call    calSum(myList) --> returns myList[0] + myList[1:]  
# second call   myList[1:]    --> returns myList[1] + 
myList[2:]    
myList[3:] + myList[4:]    
# fourth call   myList[3:]    --> returns myList[3] + myList[4:]  
myList[4:] + myList[5:]    
```

Example 2

Fibonacci numbers are a list of integers with special relationship between each other:

- fib(0) = 0
- fib(1) = 1
- fib(n) = f(n-1) + f(n-2)

Hence, the Fibonacci sequence is: 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...

It's required to write the function “fib(n)” that gets the Fibonacci number of the input parameter 'n'.

For example: fib(5) = fib(4) + fib(3)

fib(4) = fib(3) + fib(2)

fib(3) = fib(2) + fib(1)

fib(2) = fib(1) + fib(0) = 1 + 0 = 1

then, fib(3) = 2, fib(4) = 2 + 1 = 3, so fib(5) = 3 + 2 = 5
def fib(n):
    if n==0:  # terminating condition
        return 0
    if n==1:  # terminating condition
        return 1
    return fib(n-1) + fib(n-2)  # recursive call

From:
http://courses.cs.purdue.edu/ - Computer Science Courses

Permanent link:
http://courses.cs.purdue.edu/cs17700:fall15:prelab:prelab14

Last update: 2015/11/29 00:32