Revisit Java Programming
(Recursion)

Fall, 2020

Table of Contents

Recursion

- Another way of repeating a logic like loop
- In some problems, it may be natural to define the problem in terms of the problem itself.
- Recursion is useful for problems that can be represented by a simpler version of the same problem.
- Example: the factorial function

```
6! = 6 * 5 * 4 * 3 * 2 * 1
```

We could write:

```
6! = 6 * 5!
```

Recursion: factorial example

In general, we can express the factorial function as follows:

```
n! = n * (n-1)!
```

Is this correct? Well… almost.

The factorial function is only defined for *positive* integers. So we should be a bit more precise:

```
n! = 1                (if n is equal to 1)
n! = n * (n-1)!       (if n is larger than 1)
```

Recursion: factorial example

```
public static int fac(int numb){
    if(numb<=1)
        return 1;
    else
        return numb * fac(numb-1);
}
```

***recursion* means that a function calls itself**

Recursion: factorial example

- Assume the number typed is 3, that is, numb=3.

```
fac(3) :
3 <= 1 ?                    No.
fac(3) = 3 * fac(2)
   fac(2) :
      2 <= 1 ?              No.
      fac(2) = 2 * fac(1)
         fac(1) :
            1 <= 1 ?        Yes.
            return 1
       fac(2) = 2 * 1 = 2
       return fac(2)
 fac(3) = 3 * 2 = 6
 return fac(3)
 fac(3)  has the value 6
```

```
int fac(int numb){
  if(numb<=1)
      return 1;
   else
      return numb * fac(numb-1);
}
```

Recursion: factorial example

For certain problems (such as the factorial function), a recursive solution often leads to short and elegant code. Compare the recursive solution with the iterative solution:

Recursive solution

```
int fac(int numb){
   if(numb<=1)
     return 1;
   else
     return numb*fac(numb-1);
}
```

Iterative solution

```
int fac(int numb){
  int product=1;
  while(numb>1){
     product *= numb;
     numb--;
  }
  return product;
}
```

Recursion

We have to pay a price for recursion:

- calling a function consumes more time and memory than using a loop.
- Using a stack not heap memory
- high performance applications (graphic action games, simulations of nuclear explosions) hardly ever use recursion.

Please use recursion for the right problems!

Recursion

We must always make sure that the recursion *bottoms out*:

- A recursive function must contain at least one non-recursive branch.
- The recursive calls must lead to a non-recursive branch.

- Recursion is one way to decompose a task into smaller subtasks. At least one of th e subtasks is a smaller example of the same task.
- The smallest example of the same task has a non-recursive solution.

Example: The factorial function

`n! = n * (n-1)! and 1! = 1`

```
public static int fac(int numb){
    if(numb<=1)
        return 1;
    else
        return numb * fac(numb-1);
}
```

Practice 1

- Write a recursive factorial

Recursion: Fibonacci numbers

- Fibonacci numbers:

  `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...`

  where each number is the sum of the preceding two.

- Recursive definition:
  - `F(0) = 0;`
  - `F(1) = 1;`
  - `F(number) = F(number-1)+ F(number-2);`

Recursion: Fibonacci numbers

```
public static int fib(int number)
{
  if (number == 0) return 0;
  if (number == 1) return 1;
  return (fib(number-1) + fib(number-2));
}
```

Practice 2

- Write a recursive fibonacci

Practice 3

- Recursive Sum
  - https://github.com/JaewookByun/h02406/blob/master/basic/src/main/java/kr/ac/sejong/icse/advanced_programming/basic/lecture10/P3RecursiveSum.java

```
3   public class P3RecursiveSum {
4
5           public static int recursiveSum(int i) {
6                   if( i == 0 )
7                           return i;
8                   return i + recursiveSum(i-1);
9           }
10
11          public static void main(String[] args) {
12                  System.out.println(recursiveSum(10));
13          }
14  }
```

Practice 4

- Recursive Sum
  - https://github.com/JaewookByun/h02406/blob/master/basic/src/main/java/kr/ac/sejong/icse/advanced_programming/basic/lecture10/P4RecursiveSum2.java

```
3   public class P4RecursiveSum2 {
4
5           public static int recursiveSum(int[] intArray, int index) {
6                   if (index == 0)
7                           return intArray[index];
8                   return intArray[index] + recursiveSum(intArray, index - 1);
9           }
10
11          public static void main(String[] args) {
12                  int[] intArray = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
13                  System.out.println(recursiveSum(intArray, 8));
14          }
15  }
```

Recursive: Binary Search

- Search for an element in an array
  - Sequential search
  - Binary search
- Binary search
  - Compare the search element with the middle element of the array
  - If not equal, then apply binary search to half of the array (if not empty) where the search element would be.

Recursion: Binary Search

```
// Searches an ordered array of integers using recursion
Public static int bsearchr(const int data[], // input: array
                int first,          // input: lower bound
                int last,          // input: upper bound
                int value           // input: value to find
          )// output: index if found,  otherwise return -1

{   //cout << "bsearch(data, "<<first<< ", last "<< ", "<<value << "); "<<endl;
    int middle = (first + last) / 2;
    if (data[middle] == value)
        return middle;
    else if (first >= last)
        return -1;
    else if (value < data[middle])
        return bsearchr(data, first, middle-1, value);
    else
        return bsearchr(data, middle+1, last, value);
}
```

Practice 5

- Write a recursive binary search

Practice 5

- Write a recursive Euclid method
  - https://github.com/JaewookByun/h02406/blob/master/basic/src/main/java/kr/ac/sejong/icse/advanced_programming/basic/lecture11/P2RecursiveEuclid.java

```
20  public class P2RecursiveEuclid {
21          // recursive implementation
22          public static int gcd(int p, int q) {
23                  if (q == 0)
24                          return p;
25                  else
26                          return gcd(q, p % q);
27          }
28          public static void main(String[] args) {
29                  int p = 1440;
30                  int q = 408;
31                  int d = gcd(p, q);
32                  System.out.println("gcd(" + p + ", " + q + ") = " + d);
33          }
34  }
```

```
//gcd(1440, 408)
//   gcd(408, 216)
//      gcd(216, 192)
//         gcd(192, 24)
//            gcd(24, 0)
//               return 24
//            return 24
//         return 24
//      return 24
//   return 24
```

Summary

- Recursion