Recursion

• General Algorithm for Recursion

• When to use and not use Recursion

• Recursion Removal

• Examples

• Comparison of the Iterative and Recursive Solutions

• Exercises
General Algorithm for Recursion

- As we studied in the last unit recursion consists of two steps.
  - The Base Step
  - The Recursive Step

- The general algorithm for a recursive problem can be outlined as follows:
  
  - if stopping condition then
    - solve simple problem (base)
  else
    - use recursion to solve smaller problem

  - combine solutions from smaller problem
When to use and not use Recursion

- **When to use recursion:**
  - The problem definition is recursive
  - The problem is simpler to solve recursively
  - When the produced results are used in the reverse order of their creation

- **When *not* to use recursion:**
  - The recursion can be replaced with only a loop
  - Run-Time or space limitation
Recursion Removal

- Recursion can be removed by replacing the selection structure (if-statement) with a loop.

- If some data need to be stored for processing after the end of the recursive step, a data structure is needed in addition to the loop.

- The data structure vary from a simple string or an array to a stack.
Example 1 – The factorial program.

• Recall the recursive version of our factorial program in the last unit:
  
  ```java
  public static long factorial(long number)
  {
    if(number == 0) //base step
      return 1;
    else //recursive step
      return number * factorial(number - 1);
  }
  ```

• It is easy to provide an iterative version based on a loop and a condition:
  
  ```java
  public static long factorial(long number)
  {
    long fact = 1;
    for(long i = number; i >= 1; i--)
      fact = fact*i;
    return fact;
  }
  ```
Example 2 – Fibonacci Numbers

• Recall the recursive version of our fibonacci program in the last unit:
  public static int fib(int number)
  {
    if(number <= 2) //base step
      return 1;
    else //recursive step
      return fib(number - 1) + fib(number - 2);
  }

• It is easy to provide an iterative version based on a loop and a condition:
  public static int fib(int number)
  {
    int fib1 = 1, fib2 = 1, fib = 1;
    for(int i = 3; i <= number; i++)
    {
      fib2 = fib1;
      fib1 = fib;
      fib = fib2 + fib1;
    }
    return fib;
  }
Comparison of the Iterative and Recursive Solutions

- If both the iterative and recursive versions of the examples are considered we find that:
  - The recursive version is simpler
  - The iterative version involves the use of variables to store the intermediate values during computation.
  - The iterative version is more time-efficient as no activation records are created.
  - The iterative version can be easily created for these examples which are tail recursive.
    - *Tail Recursion* is a special form of recursion in which the last statement of a recursive method is a recursive call.
    - Tail Recursive methods can easily be converted to iterative methods.
Recursion – A Complete Example

- **Converting a decimal integer to its binary equivalent.**
  - The algorithm (in pseudocode) for converting a decimal integer into a binary integer as follows:

  1. If the integer is 0 or 1, its binary equivalent is 0 or 1.
  2. If the integer is greater than or equal to 2 do the following:
  3. Divide the integer by 2.
  4. Separate the result into a quotient and remainder.
  5. Divide the quotient again and repeat the process until the quotient is zero.
  6. Write down all remainders in reverse order as a string.
  7. This string is the binary equivalent of the given integer.

- As an example consider the conversion of 14 to its equivalent binary representation.

  - 14/2 = Quotient:7 Remainder: 0 ⇒ 7/2 = Quotient: 3 Remainder: 1
    ⇒ 3/2 = Quotient:1 Remainder:1 ⇒ 1/2 ⇒ Quotient: 0 Remainder: 1

- Therefore the binary representation of 14 is 1110.
Pseudocode Version

- This method converts an integer number to its binary equivalent.
- Base step
  - \text{dec2bin}(n) = n \text{ if } n \text{ is 0 or 1}
- Recursive step
  - \text{dec2bin}(n) = \text{dec2bin} \left( \frac{n}{2} \right), (n \mod 2)
- Algorithm \text{dec2bin}(n):
  - If \( n < 2 \)
    - return \( n \) as a string.
  - else
    - \text{dec2bin}(n/2)
    - append \( n \mod 2 \)
import java.io.*;

public class Dec2Bin
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter an integer: ");
        int n = Integer.parseInt(in.readLine());
        String answer = dec2bin(n);
        System.out.println("The binary equivalent of "+n+" is "+answer);
    }

    public static String dec2bin(int n){
        if (n < 2) return n + ";
        else return (dec2bin(n/2) + n%2);
    }
}
import java.io.*;

public class Dec2BinIterative
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter an integer: ");
        int n = Integer.parseInt(in.readLine());
        String answer = dec2bin(n);
        System.out.println("The binary equivalent of " + n + " is " + answer);
    }

    public static String dec2bin(int n){
        int q = n, r = 0;
        String x = new String();
        do
        {
            r = q%2;
            q = q/2;
            x = r + x;
        } while(q != 0);
        return x;
    }
}
Exercises

1. Write iterative solutions to exercises in Unit 17.