Chapter 9

Numeric

Unless stated otherwise the alias n denotes a standard 32 bit integer.

9.1 Primality Test

A simple algorithm that determines whether or not a given integer is a prime number, e.g. 2, 5, 7, and 13 are **all** prime numbers, however 6 is not as it can be the result of the product of two numbers that are < 6.

In an attempt to slow down the inner loop the $\sqrt{n}$ is used as the upper bound.

```
1) algorithm IsPrime(n)
2)     Post: n is determined to be a prime or not
3)     for i ← 2 to n do
4)         for j ← 1 to sqrt(n) do
5)             if i * j = n
6)                 return false
7)             end if
8)         end for
9)     end for
10) end IsPrime
```

9.2 Base conversions

DSA contains a number of algorithms that convert a base 10 number to its equivalent binary, octal or hexadecimal form. For example 78_{10} has a binary representation of 1001110_2.

Table 9.1 shows the algorithm trace when the number to convert to binary is 742_{10}.

```
1) algorithm ToBinary(n)
2)     Pre:  n ≥ 0
3)     Post: n has been converted into its base 2 representation
4)     while n > 0
5)         list.Add(n % 2)
6)         n ← n/2
7)     end while
8)     return Reverse(list)
9) end ToBinary
```

n	$list$
742	{ 0 }
371	{ 0, 1 }
185	{ 0, 1, 1 }
92	{ 0, 1, 1, 0 }
46	{ 0, 1, 1, 0, 1 }
23	{ 0, 1, 1, 0, 1, 1 }
11	{ 0, 1, 1, 0, 1, 1, 1 }
5	{ 0, 1, 1, 0, 1, 1, 1, 1 }
2	{ 0, 1, 1, 0, 1, 1, 1, 1, 0 }
1	{ 0, 1, 1, 0, 1, 1, 1, 1, 0, 1 }

Table 9.1: Algorithm trace of ToBinary

9.3 Attaining the greatest common denominator of two numbers

A fairly routine problem in mathematics is that of finding the greatest common denominator of two integers, what we are essentially after is the greatest number which is a multiple of both, e.g. the greatest common denominator of 9, and 15 is 3. One of the most elegant solutions to this problem is based on Euclid's algorithm that has a run time complexity of $O(n^2)$.

```
1) algorithm GreatestCommonDenominator(m, n)
2)     Pre:  m and n are integers
3)     Post: the greatest common denominator of the two integers is calculated
4)     if n = 0
5)        return m
6)     end if
7)     return GreatestCommonDenominator(n, m % n)
8) end GreatestCommonDenominator
```

9.4 Computing the maximum value for a number of a specific base consisting of N digits

This algorithm computes the maximum value of a number for a given number of digits, e.g. using the base 10 system the maximum number we can have made up of 4 digits is the number 9999_{10}. Similarly the maximum number that consists of 4 digits for a base 2 number is 1111_2 which is 15_{10}.

The expression by which we can compute this maximum value for N digits is: $B^N - 1$. In the previous expression B is the number base, and N is the number of digits. As an example if we wanted to determine the maximum value for a hexadecimal number (base 16) consisting of 6 digits the expression would be as follows: $16^6 - 1$. The maximum value of the previous example would be represented as $FFFFFF_{16}$ which yields 16777215_{10}.

In the following algorithm *numberBase* should be considered restricted to the values of 2, 8, 9, and 16. For this reason in our actual implementation *numberBase* has an enumeration type. The *Base* enumeration type is defined as:

$$Base = \{Binary \leftarrow 2, Octal \leftarrow 8, Decimal \leftarrow 10, Hexadecimal \leftarrow 16\}$$

The reason we provide the definition of *Base* is to give you an idea how this algorithm can be modelled in a more readable manner rather than using various checks to determine the correct base to use. For our implementation we cast the value of *numberBase* to an integer, as such we extract the value associated with the relevant option in the *Base* enumeration. As an example if we were to cast the option *Octal* to an integer we would get the value 8. In the algorithm listed below the cast is implicit so we just use the actual argument *numberBase*.

1) **algorithm** MaxValue(*numberBase*, n)
2) **Pre:** *numberBase* is the number system to use, n is the number of digits
3) **Post:** the maximum value for *numberBase* consisting of n digits is computed
4) **return** Power(*numberBase*, n) -1
5) **end** MaxValue

9.5 Factorial of a number

Attaining the factorial of a number is a primitive mathematical operation. Many implementations of the factorial algorithm are recursive as the problem is recursive in nature, however here we present an iterative solution. The iterative solution is presented because it too is trivial to implement and doesn't suffer from the use of recursion (for more on recursion see §C).

The factorial of 0 and 1 is 0. The aforementioned acts as a base case that we will build upon. The factorial of 2 is 2∗ the factorial of 1, similarly the factorial of 3 is 3∗ the factorial of 2 and so on. We can indicate that we are after the factorial of a number using the form $N!$ where N is the number we wish to attain the factorial of. Our algorithm doesn't use such notation but it is handy to know.

```
1) algorithm Factorial(n)
2)     Pre:  n ≥ 0, n is the number to compute the factorial of
3)     Post: the factorial of n is computed
4)     if n < 2
5)         return 1
6)     end if
7)     factorial ← 1
8)     for i ← 2 to n
9)         factorial ← factorial * i
10)    end for
11)    return factorial
12) end Factorial
```

9.6 Summary

In this chapter we have presented several numeric algorithms, most of which are simply here because they were fun to design. Perhaps the message that the reader should gain from this chapter is that algorithms can be applied to several domains to make work in that respective domain attainable. Numeric algorithms in particular drive some of the most advanced systems on the planet computing such data as weather forecasts.