The First Programming Problem

JAN 20TH 2012

Problem: Converting decimal numbers to binary

- Given a non-negative integer, convert it into its **binary equivalent**.

- **Example:**
  - **Input:** 123 **Output:** 1111011
  - **Input**: 1363 **Output**: 10101010011
  - **Input**: 12 **Output**: 1100

Plan of Action

1. Understand the problem. What does "binary equivalent" mean?

2. Design algorithms for the problem. How would we solve the problem with a pencil and paper?

3. Write down pseudocode for the algorithm.

4. Translate the pseudocode to Python code.

5. Test, test, test.

This example will illustrate

- Constants
- Variables
- Operators
- Data types
- Expressions
- Function calls
- Input statements
- Output statements
- Control flow statements

Decimal numbers revisited

Consider the decimal number 8,374.

	8	3	7	4
Place value	1000	100	10	1

Therefore, the "value" of this number is

$$8 \times 1000 + 3 \times 100 + 7 \times 10 + 4 \times 1$$

What are binary numbers?

Similarly, consider the binary number 10110110.

1	0	1	1	0	1	1	0

Place values:	128	64	32	16	8	4	2	1

Just like the place values for decimal numbers are powers of 10, the place values for binary numbers are powers of 2.

Therefore, the "value" of this number is

$$128 + 32 + 16 + 4 + 2 = 182$$

Table of Binary Equivalents

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100

Two observations based on this table

Observation 1:

If n is even, then its binary equivalent ends with a 0; otherwise if n is odd, its binary equivalent ends with 1.

Two observations based on the table

Observation 2:

Suppose that the binary equivalent of n is

$$b_k \dots b_2\, b_1\, b_0$$

If n is even, then the binary equivalent of n/2 is

$$b_k \dots b_2\, b_1$$

and if n is odd, then the binary equivalent of (n-1)/2 is

$$b_k \dots b_2\, b_1$$

This suggests an algorithm

- Check if the given number n is odd or even.
- If n is even, we know that its binary equivalent ends with 0. Furthermore, to get the rest of n's binary equivalent, we need to "consult" n/2.
- If n is odd, we know that the binary equivalent ends with 1. Furthermore, to get the rest of n's binary equivalent, we need to "consult" (n-1)/2.

Ilustration of this algorithm

Let the given input be n = 203.

1. n = 203 is odd. So rightmost bit is 1.
To get the rest of the answer we should "consult" (n-1)/2 = 101.
2. n = 101 is odd. So the rightmost bit is 1.
To get the rest of the answer we should "consult" (n-1)/2 = 50
3. n = 50 is even. So the rightmost bit is 0.
To get the rest of the answrt we should "consult" n/2 = 25.
4. n = 25 is odd. So the rightmost bit is 1.
To get the rest of the answer we should "consult" (n-1)/2 = 12.
5. n = 12 is even. So the rightmost bit is 0.
To get the rest of the answrt we should "consult" n/2 =6.
6. n = 6 is even. So the rightmost bit is 0.
To get the rest of the answrt we should "consult" n/2 =3.
7. n = 3 is odd. So the rightmost bit is 1.
To get the rest of the answer we should "consult" (n-1)/2 = 1.
8. n = 1 is odd. So the rightmost bit is 1.
To get the rest of the answer we should "consult" (n-1)/2 = 0.

So the output (right to left) is 1 1 0 1 0 0 1 1.

Pseudocode

1. Read the number n given as input.
2. If n is even, output 0. Replace n by n/2.
3. If n is odd, output 1. Replace n by (n-1)/2.
4. If n is 0, stop. Otherwise go to Line 2.

Note that this algorithm produces the binary equivalent of n in "right to left order."

Our first program

```
n = int(raw_input("Enter a positive integer:"))
while n > 0:
    print n % 2
    n = n/2
```