Computational Logic Optional Assignment

Technical Documentation

Student:
Andrei Bratu
912
2018

Statement

Implement an application with the following functionalities:

- Addition, subtraction, multiplication, division by one digit in base $p \in \{2, 3, ..16\}$
- Conversion of natural numbers between arbitrary bases $p,\ q \in \{2, 3, ..16\}$ using substitution method or successive divisions
- Rapid conversions between bases $\{2, 4, 8, 16\}$
- User interface that allows verifying all features independently

Features

1. Adding two numbers in a base
2. Subtracting two numbers in a base
3. Multiplying a number by a digit in a base
4. Dividing a number by a digit in a base
5. Converting a number from a base to another
6. Fast conversion between base2 and base4
7. Fast conversion between base2 and base8
8. Fast conversion between base2 and base16
9. Possibility of exiting the application

Accomplished tasks

1. Adding two numbers digit by digit, with carry
2. Subtracting two numbers digit by digit, with borrow
3. Multiplying a number by a digit in a given base
4. Dividing a number by a digit in a given base
5. Converting from a base q to base 10 via substitution
6. Converting from base 10 to base q via repeated divisions
7. Converting from a base p to a base q using (5) and (6)
8. Rapid conversion between bases $\{2, 4, 8, 16\}$
9. Layered architecture of the application, using Python as a programming language
10. Terminal based UI

Used data types

Number

Number represents a custom data structure that defines how a number is stored and interacts with other entities. Number is a Python class with the following attributes: **val**, the value of the number, represented as a string, and **base**, the base of the number, represented as an integer. When doing arithmetic operations on Number entities, the val field is transformed into a Python list, in which digits are held in reverse order, in order to allow digit by digit operations. Being a high programming language, Python allows defining the behaviour of arithmetic operations on user-defined types, making the implementation more suggestive(see the ui module), while encapsulating the behaviour into the Number class.

List

The Python List data type, as described above, is used internally in representing Number entities into a working form. A list holds the digit values, in reversed order.

Dictionary

The Python dictionary is an associative data type, used in order to map characters to the corresponding digit value, and more suggestively, during the fast conversions. An example of Python dictionary is given below:

```
base2_base_16_dict = {
    '0000': '0', '0001': '1', '0010': '2', '0011': '3',
    '0100': '4', '0101': '5', '0110': '6', '0111': '7',
    '1000': '8', '1001': '9', '1010': 'A', '1011': 'B',
    '1100': 'C', '1101': 'D', '1110': 'E', '1111': 'F'
}
```

User Interaction

The application offers a menu-based interface, which prompts the user to select one of the nine features. After introducing the desired feature, the application prompts the user for the required inputs, usually the value of the numbers, their bases and the base of the final computation. The result of the operation is displayed, and the interaction loop starts over.

Code structure

The project uses a layered, object oriented approach.

- **number.py** Holds the Number class, which describes a number entity
- **validator.py** Holds the NumberValidator class, delegated with validating a number introduced by the user
- **ui.py** Holds the UI class, delegated with the user interface
- **exceptions.py** Holds the InvalidNumberException class, raised by invalid user input
- **test.py** Unit tests of the application
- **main.py** The entry point of the application

Used algorithms

Below are the main algorithms used in the program. The fast conversions and the auxiliary functions have been left out, but are thoroughly documented in the code.

List Representation

```
FUNCTION GET_LIST_REPRESENTATION
    INPUT val // A string
    char_to_digit = {
        '0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7,
        '8': 8, '9': 9, 'A': 10, 'B': 11, 'C': 12, 'D': 13, 'E': 14,
        'F': 15
    }
    repr = []

    FOR EACH CHARACTER c in val:
        APPEND char_to_digit[c] to repr
```

Addition

```
FUNCTION ADD
    INPUT self
    INPUT other
    base = self.base
    self_repr = get_list_representation(self)
    other_repr = get_list_representation(other)
    carry = 0

    WHILE other_reprs has fewer digits than self_repr:
        APPEND 0 to other_repr

    WHILE self_repr has fewer digits than other_repr:
        APPEND 0 to self_repr
    FOR EACH digit in self_repr:
        digit_self_repr += digit_other_repr + carry
        carry = digit_self_repr / base
        digit_self_repr = digit_self_repr % base

    if carry != 0:
        APPEND carry to self_repr
```

Subtraction

```
FUNCTION SUBTRACTION
    INPUT self
    INPUT other

    base = self.base
    self_repr = get_list_representation(self)
    other_repr = get_list_representation(other)
    carry = 0

    WHILE other_repr has fewer digits than self_repr:
        APPEND 0 to other_repr

    FOR EACH digit in self_repr
        digit_self_repr = digit_self_repr - digit_other_repr - carry
        IF digit_self_repr < 0:
            carry = 1
        ELSE:
            carry = 0
        IF carry != 0:
            digit_self_repr += base
```

Multiplication

```
FUNCTION MULTIPLICATION
    INPUT self
    INPUT digit

    self_repr = get_list_representation(self)

    FOR EACH digit in self_repr:
        digit_self_repr = digit_self_repr * digit + carry
        carry = digit_self_repr / base
            digit_self_repr = digit_self_repr % base

    WHILE carry != 0:
        self_repr.append(carry % base)
        carry = carry / base
```

Division

```
FUNCTION DIVISION
       INPUT self
       INPUT digit

       self_repr = get_list_representation(self)

       FOR EACH digit of self, starting from most significant:
             remainder = base * remainder + digit_self_representation
             digit_self_representation = remainder / digit
             remainder = remainder % digit

       WHILE most significant digit of self_repr == 0:
             POP most_significant_digit
```

From any base to base 10

```
FUNCTION ANY_BASE_TO_BASE10
       INPUT number_repr // The number as a list
       base_10 = 0
       power = 1
       for i in range(0, len(number_repr)):
                   base_10 += number_repr[i] * power
                   power *= q

       repr = []
       WHILE base_10:
                   repr += [base_10 % 10]
                   base_10 = base_10 / 10

       REVERSE repr
```

From base 10 to any base

```
FUNCTION BASE10_TO_ANY_BASE
       INPUT number // The number as a string
       number = INT number
       repr = []
       WHILE number != 0:
                   repr += [number % q]
                   number = number / q

    REVERSE repr
```