COMPSCI 210
Computer Systems 1

Assignment 1 - Floating Point

Worth 5%

final date 20th of March, 2018, 11:59pm

Introduction

This assignment gets you to implement a simple floating point format using Python. The assignment leads you to develop the floating point format in a series of steps.

CodeRunner

All of the programming is to be tested on CodeRunner https://coderunner2.auckland.ac.nz, however don't develop your code in CodeRunner as it is a testing environment, not a development environment. IDLE or some other IDE or text editor is much better.

The programs submitted through CodeRunner will be checked for similarities to all other submissions, so please ensure that the work is your own. The checker is not fooled by changing attribute names and comments or reordering the code, and other alterations.

Python provides a number of easy ways to do some of these exercises. Unfortunately using these solutions does not demonstrate the algorithms which produce the output. The markers will check your submitted programs and you will lose any marks allocated by the CodeRunner system if you haven't used the indicated algorithms. This is mostly for Program 1 and Program 2.

Program 1 (2 marks)

The function has this signature:

```
def integer_to_binary(integer_string)
```

Convert a string of decimal digits up to 32 digits long representing a positive decimal integer value into its equivalent binary digit string.

You must use the decimal to binary conversion algorithm described in class: repeated division by two. Print each binary digit as it is produced and print the final result.

The following method produces the expected output but does not use the correct algorithm, hence it would pass in CodeRunner but you would lose all marks when the markers review the code.

```
def integer_to_binary(integer_string):
    result = bin(int(integer_string))[2:]
    for bit in reversed(result):
        print(bit, end=' ')
    print()
    print(result)
```

Program 2 (4 marks)

The function has this signature:

```
def fraction_to_binary(fraction_string, significant_binary_digits)
```

Convert a string of decimal digits up to 32 digits long representing a positive decimal fractional value into its equivalent binary digit string. For this function to terminate the algorithm accepts the number of significant binary digits to produce. Binary digits in the result are significant if they are the first "1" in the result or any digits following the first "1". The last significant binary digit must be rounded in the final result. So the algorithm will print more digits than the specified number

because leading "0"s don't count and the algorithm must go one more digit than the specified number of digits and the final result must show exactly the number of specified significant binary digits using the extra digit to round up if necessary.

You may assume that the decimal fraction will NOT be zero.

e.g.

`0.5` to two significant binary digits would be `0.10`

`0.125` to two significant binary digits would be `0.0010`

`0.875` to one significant binary digit would be `1.`, because that is `0.111` rounded to two significant binary digits.

You must use the decimal to binary conversion algorithm described in class: repeated multiplication by two. Print each binary digit as it is produced and print the final result.

All decimal fractions will start with "0." and the binary final result must also start with "0." except when the value rounds to "1.".

The 32 digits in the decimal fraction do not include the "0.".

You may find it useful to use the Decimal class. See https://docs.python.org/3/library/decimal.html

---

Program 3 (2 marks)

The function has this signature:

```
def valid_float(number_string)
```

Check whether a string consists only of decimal digits (of any length) and the "-" and "." characters and matches the following requirements. The function returns True if the string is valid, and False otherwise.

The string must start with a decimal digit or "-". The "-" can only occur as the first character if it appears at all.

If the first decimal digit is a "0" the next character must be ".".

The string does not have to include a "." There can be only one "." in the string.

If the string includes a "." there must be a string of 1 or more decimal digits before the "." and there must be 1 or more decimal digits after the "."

These strings are valid:

```
1234
-1234
12.4
0.6
-0.6
-1234567890.123456789
```

These strings are invalid:

```
+123
123.
.6
00.6
12-.6335
```

You may use regular expressions. See https://docs.python.org/3/library/re.html

An alternative could be to use a finite state machine.

Program 4 (2 marks)

The function has this signature:

```
def break_into_parts(decimal_string)
```

Break the decimal string into the sign, the integer part and the fractional part. The decimal string is guaranteed to be valid as in Program 3. The function must print the sign, integer and fraction part as below. If the sign is not negative it is printed as "+", if the fractional part does not exist it is printed as "0".

If the decimal string is `-12.40` the output would be:

```
sign: - integer: 12 fraction: 40
```

If the string is `1234` the output would be:

```
sign: + integer: 1234 fraction: 0
```

Program 5 (2 marks)

The function has this signature:

```
def print_binary(number_string, fraction_length)
```

Print the binary version of the decimal number string. The decimal string is guaranteed to be valid as in Program 3. The printed binary number must have at least one digit before and after the ".". The fraction length parameter is the number of binary digits to print after the ".".

Unlike Program 2 you do NOT round the binary fraction at the last digit.

Positive numbers should be printed with a sign.

If the number is `0` and the fraction length is `4` the output would be :

```
+0.0000
```

If the decimal number is `-12.4` and the fraction length is `10` the output would be:

```
-1100.0110011001
```

Program 6 (8 marks)

The function has this signature:

```
def float_string_to_binary(float_string)
```

It takes a decimal number string representing a floating point value and prints out the binary equivalent using the floating point format described below. You may assume that all values of the decimal string will fit within this format.

The format

The floating point format to use is similar to the IEEE 32-bit floating point format as covered in class. It is a 16-bit format (not quite the same as the IEEE 16-bit format).

Sign: the first bit (most significant bit: bit 15). 1 means negative, 0 means positive

Exponent: four bits for the exponent (bits 14–11). The exponent has a bias of 7.

Mantissa (fractional part): 11 bits (bits 10–0). The mantissa is normalised and the most significant 1 is not represented. **When converting a decimal number into this format you must round the 12th significant binary digit of the number before removing the most significant 1.**

Zero is represented by all 16 bits being 0.

e.g. The largest number which can be represented by this approach is 511.875

```
0 1111 11111111111
```

The sign is positive.

The exponent is 1111_2 minus the bias or 15 - 7 = 8.

The mantissa is 1.11111111111_2 remember the non-showing 1 before the binary point.

$1.11111111111_2 * 2^8 = 111111111.111_2 = 511.875$

I leave you with the exercise to find the smallest non-zero value which can be represented by this format.

And

Include a comment with your login name (UPI) at the top of each program you enter into CodeRunner. Markers will remove a mark if you have not done this on every program.

Any work you submit must be your work and your work alone.

To share assignment solutions and source code is not permitted under our academic integrity policy. Violation of this will result in your assignment submission attracting no marks, and you will face disciplinary actions in addition.