Étude 03 Koch Snowflake Interactive

Ruiyi Qian

Abstract

This is a task about drawing the Koch Snowflake. Koch Snowflake is a fractal curve, which is a curve that can never be drawn in a finite number of steps. For most of the fractals, they are self-similar, which means that they can be divided into smaller parts that are similar to the whole.

In this project, I have implemented a program that can draw the Koch Snowflake interactively, and any order of the snowflake can be drawn in affordable time.

You can find the source code of the program in this repository.

1 Introduction

This task is to process the given dates and check whether they are valid or not.

Input (stdin)	Output (stdout)	Error Output (stderr)
4-6-92	04 Jun 1992	
04/06/92	04 Jun 1992	
3 AUG 97	03 Aug 1997	
12-Sep-1955	12 Sep 1955	
03 JUN 3004	03 JUN 3004 - INVALID	Year out of range

The task input and output are shown in the table above. For the valid dates, the program will output the date in the format of day, month and year, which are separated by a space. For the invalid, the program needs to identify the error and output the error message. The valid date format is as follows consisting of day, month and year, which are separated by a hyphen, slash or space. The exact rules are well defined in the task description. Such as the month can be written in digits or abbreviation, upper or lower case. The year can be written in two or four digits. The day can be written in one or two digits.

2 The Program

Those rules can be summerized as grammer rules and range rules. Grammer rules are used to check the format of the date, quite easy and can be written in ANTLR4. Range rules are used to check the validity of the date, which is more complicated and needs to be checked in the program. I will list the grammer rules and range rules in the following sections. And the detailed source code can be found in the repository.

Grammer Rules

```
grammar date;
date:    day '-' month '-' year
     |   day '/' month '/' year
     |   day ' ' month ' ' year
     ;
day:     DIGIT DIGIT | DIGIT;
month:   DIGIT DIGIT | DIGIT
     |   UPPER UPPER UPPER
     |   UPPER LOWER LOWER
     |   LOWER LOWER LOWER
     ;
year:    DIGIT DIGIT DIGIT DIGIT | DIGIT DIGIT;
DIGIT:   [0-9]
UPPER:   [A-Z]
LOWER:   [a-z]
```

Range Rules

In this section, we assume that the date is valid in the grammer rules, and already converted to the format of day, month and year, note that the they are still in string format.
So the first thing we need to do is to convert the string to integer. The following pseudo code shows what I have done.

Algorithm 1: String to Integer

Input: raw day: d_{str}, raw month: m_{str}, raw year: y_{str}
Output: day: d, month: m, year: y
Data: $M_{map} \leftarrow \{$"JAN" : 1, "FEB" : 2, "MAR" : 3, "APR" : 4, "MAY" : 5, "JUN" : 6, "JUL" : 7, "AUG" : 8, "SEP" : 9, "OCT" : 10, "NOV" : 11, "DEC" : 12$\}$

if $m_{str}.int() \notin$ **N then**
  if $m_{str}.upper() \notin M_{map}.keys()$ **then**
    return *Invalid month*
  end
  $m_{str} \leftarrow M_{map}.get(m_{str}.upper()).str()$
end
if $y_{str}.length()$ *is 2* **then**
  if $y_{str}.int() < 50$ **then**
    $y_{str} \leftarrow$ "20" $+ y_{str}$
  end
  else
    $y_{str} \leftarrow$ "19" $+ y_{str}$
  end
end
return $d \leftarrow d_{str}.int()$, $m \leftarrow m_{str}.int()$, $y \leftarrow y_{str}.int()$

Then we can check the range rules. It is quite easy to check the year range, but the month and day range are more complicated. One thing to note is that the month and day range are different for leap year and non-leap year. So it is better to check the leap year first. I choose to use the following pseudo code to check the month and day range.

Algorithm 2: Check the range of the date

Input: day: d, month: m, year: y
Output: valid date: *date* or error message

$leap \leftarrow [4 \mid y] \wedge (\neg[100 \mid y] \vee [400 \mid y])$
if $y \notin [1753, 3000]$ **then return** *Year out of range* ;
if $m \in \{1, 3, 5, 7, 8, 10, 12\}$ **then**
  if $d \notin [1, 31]$ **then return** *Day out of range* ;
end
else if $m \in \{4, 6, 9, 11\}$ **then**
  if $d \notin [1, 30]$ **then return** *Day out of range* ;
end
else if $m \in \{2\}$ **then**
  if $d \notin [1, 28 + leap]$ **then return** *Day out of range* ;
end
else return *Month out of range* ;
return $date \leftarrow (d, m, y)$

3 Data Generator

The data generator is used to generate the test data for the program. And I have written a python script to generate the data. Just take it for granted, those date can be split into two groups, one is the valid date, the other is the invalid date. Considering the range of the year, the valid date is generated in the range of $[1753, 3000]$. So the valid date must be finite, and the invalid date is infinite. Finite data is easy to generate, but infinite data is hard to generate. So let's focus on the valid date first.

Valid Date

For convenience, I use the `<datetime>` module in python to generate the valid date.

```
import datetime
begin_date = datetime.date(1753, 1, 1)
end_date = datetime.date(3000, 12, 31)
date = begin_date
while date <= end_date:
    # do something
    date += datetime.timedelta(days=1)
```

For each date, get the day, month and year from the `datetime` object, choose the format randomly, and then convert the date to string. Then we get the valid date that needs to be checked by the program.

```
separator_list = ['/', '-', ' ']        # separator list
day_list   = [str(d), str(d).zfill(2)] # day list
month_list = [str(m), str(m).zfill(2)] # month list
year_list  = [str(y), str(y)[-2:]]     # year list
m = abbr[m][0:3]
month_list.extend([m.upper(), m.lower(), m]) # month list
```

We first generate any valid variation of the date, organize them into a list, and then randomly choose one of them to generate the test data. Or just print them all out.

```
valid_list = []
for day in day_list:
    for month in month_list:
        for year in year_list:
            for sep in separator_list:
                date_string = day + sep + month + sep + year
                valid_list.append(date_string)
```

Now we have the valid date list, you can print them out or write them into a file.

Table 1: Valid Date(The first 5 rows)

Input (stdin)	Output (stdout)	Error Output (stderr)
09-07-1766	09 Jul 1766	
7 jun 2153	07 Jun 2153	
02/jul/2319	02 Jul 2319	
09-Nov-2343	09 Nov 2343	
03/08/2387	03 Aug 2387	

Invalid Date

As I mentioned before, the invalid date is infinite, so we can not generate all of them. My strategy is to generate the invalid date base on the valid date. Randomly choose a valid date, and then change one of its component to make it invalid. For example, if the valid date is `29-07-1766`, we can

- change the day to `32` to make it invalid
- change the month to `feb` to make it invalid
- change the year to `1752` to make it invalid
- change the separator to `'.'` to make it invalid
- delete one of the component to make it invalid
- insert some random character to make it invalid
- replace the whole date with some random string to make it invalid
- etc.

Apart from this strategy mentioned above, I have tried about 12 tricks to generate the invalid date. These tricks are assigned with different weights, during the generation process, the program will randomly choose one of the tricks to generate the invalid date. Now we have the invalid date list, you can print them out or write them into a file.

Table 2: Invalid Date(The first 5 rows)

Input (stdin)	Output (stdout)	Error Output (stderr)
12-Apr-23.0	12-Apr-23.0 - INVALID	Invalid Year
11/13/2125	11/13/2125 - INVALID	Month out of range
not a date at all	not a date at all - INVALID	Invalid Grammer
4-feb-?84?	4-feb-?84? - INVALID	Invalid Year
32-12-3000	32-12-3000 - INVALID	Day out of range

Those script can be found in the repository, `"src/gen/valid_generator.py"` and `"src/gen/invalid_generator.py"`.
You can run them with the following command:

```
# generate 10 valid dates
## the generated dates are stored in the file "valid-10.in"
## the expected output is stored in the file "valid-10.ans"
./scripts/generate.sh valid -n 10 -o valid-10

# generate 10 invalid dates
## the generated dates are stored in the file "invalid-10.in"
## the expected output is stored in the file "invalid-10.ans"
./scripts/generate.sh invalid -n 10 -o invalid-10
```

4 Local Judge

This is an trial version of the local judge. I just implemented the basic functions, to make sure the program can run correctly, and compare the spped of my implementation both in C++ and Python.

It can works both on Linux and Mac OS. Make sure you have installed the `coreutils` in your system if you are using Mac OS.

```
brew install coreutils
```

You can run the local judge with the following command:

```
# Test python version
./scripts/run_py.sh data/samples/
./scripts/run_py.sh data/generated/

# Test c++ version
./scripts/run_cpp.sh data/samples/
./scripts/run_cpp.sh data/generated/
```

Appendix

Here are 12 tricks I used to generate the invalid date.

1. Totally not a date, e.g. `not a date at all`. I add this trick to make sure the program can handle the invalid input. Even if the input is not a date, the program should still print out the error message.

2. Manually edge case, e.g. `32-12-3000`. These case are hard to be covered by the random generation.

3. Randomly edge case. I randomly choose a valid date, and then change one of its component to make it invalid. For example, if the valid date is `29-07-1766`, we can change the day to `32` to make it invalid.

4. Totally random string. In the visible ASCII range, I randomly generate a string with length between 1 and 20. I assume that the string is not a valid date. But it is possible that the string is a valid date, just ignore it.

5. Valid date with valid but different seperator. I randomly choose a valid date, and then change the one of the separators to `'/'`, `'-'` or `' '`, which is different from the original separator. The generated date will be like `29-07 1766` or `29-07/1766`.

6. Valid date with invalid seperator. I randomly choose a valid date, and then change the one of the separators to `'.'`, `'\'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`, `'^'`, `'~'`, `'`'`,

7. Randomly insert a non digit character. I randomly choose a valid date, and then insert a non digit character to the date. For example, if the valid date is `29-07-1766`, we can insert a non digit character to the date to make it invalid. The generated date will be like `29-07-1766?` or `29-07?-1766`.

8. Ramdomly replace a character in day with a non digit character. I randomly choose a valid date, and then replace a character in the day with a non digit character. For example, if the valid date is `29-07-1766`, we can replace the first character in the day to make it invalid. The generated date will be like `2#-07-1766` or `&9-07-1766`.

9. Ramdomly replace a character in year with a non digit character. I randomly choose a valid date, and then replace a character in the year with a non digit character. For example, if the valid date is `29-07-1766`, we can replace the first character in the year to make it invalid. The generated date will be like `29-07-1&66` or `29-07-#766`.

10. For month represented by digits, replace a charactor with non digit character like what we have done in day and year. For month represented by string, randomly make one of the character upper case or lower case, stop if the month is valid.

11. Duplicate a valid date. I randomly choose a valid date, and then duplicate it to make it invalid. For example, if the valid date is `29-07-1766`, we can duplicate it to make it invalid. The generated date will be like `29-07-1766 29-07-1766`. This can be used to test the program can handle the next line.

12. Leap year `29 02 ????`. Ramdomly generate a leap year, and then append `29 02` to the year.

References

[1] COSC 326. Etude 01 - Dates. *https://www.cs.otago.ac.nz/cosc326/2022/etudes/etude01-dates.pdf*.

[2] Wikihow. How To Calculate Leap Years. *https://www.wikihow.com/Calculate-Leap-Years*.

[3] Real Python. Parsing Dates in Python. *https://realpython.com/python-datetime/*.

[4] Wikipedia. Parse Tree. *https://en.wikipedia.org/wiki/Parse_tree*.