Rime with PenguinJudge Tutorial

Author: @shiodat @kazuki

はじめに

本資料はプログラミングコンテストを開催するためのチュートリアルです。

ジャッジシステムには PenguinJudge を、問題作成には Rime を使用します。

- PenguinJudge: https://github.com/penguin-judge/PenguinJudge
  - プライベート競技プログラミングコンテスト用のジャッジシステムです
- Rime: https://github.com/icpc-jag/rime
  - プログラミングコンテストの問題セット作成補助ツールです
  - PenguinJudge 向けに一部改良したものを使用します
    - https://github.com/penguin-judge/PenguinJudge

準備

PenguinJudge 対応版 Rime をインストールします。

```
$ pip install git+https://github.com/penguin-judge/rime.git
```

プロジェクト用のディレクトリを作成します。今回は `sample-contest` という名前にしてみましょう。

```
$ mkdir sample-contest
$ cd sample-contest
```

プロジェクトの初期化

`rime_init` コマンドによりRimeプロジェクトを作成することができます。

```
$ rime_init --git
Initialized empty Git repository in /home/***/sample-contest/.git/
[master (root-commit) c731ac9] Initial commit
 3 files changed, 5049 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 PROJECT
 create mode 100644 common/testlib.h
```

次ページ以降で、`PROJECT` ファイルにコンテスト情報を記述します。

PROJECT ファイルの編集

`penguin_config` を編集してPenguinJudgeで読み込みできるようにします

```
use_plugin('rime_plus')
use_plugin('judge_system.penguin')
#use_plugin('judge_system.atcoder')
...

project(library_dir='common', required_rime_plus_version='1.0.0')

penguin_config(
    id="sample",  # ユニークなコンテスト名を設定します
    title="サンプルコンテスト",  # コンテストの名前を設定します
    start="2020-07-17T13:30:00+09:00",  # 開始時刻を設定します
    end="2020-07-17T15:30:00+09:00",  # 終了時刻を設定します
)


...
```

問題の作成

Rimeプロジェクトに問題を追加します。`welcome` という名前にしてみましょう。

```
$ rime add . problem welcome
[   ADD    ] /home/***/sample-contest/welcome/PROBLEM

Error Summary:
Total 0 errors, 0 warnings
```

vi が起動しますので、問題設定を記述します。詳細は次ページで説明します。

問題設定は後で編集できますので、そのまま編集せずに保存しても問題ありません。

PROBLEM ファイルの編集

`time_limit`, `id`, `title`, `memory_limit`, `score` を設定します。

```
problem(
  time_limit=5,  # 実行時間制約(秒)を設定します
  id="welcome",  # ユニークな問題名を設定します
  title="Welcome to Penguin Judge",  # 問題の名前を設定します
  #reference_solution='???',  # 想定解法がある場合、その解法ディレクトリを設定します
  ...
)

penguin_config(
  memory_limit=512,  # メモリ制約(MB)を設定します
  score=100,  # 正解時のスコアを設定します
)
...
```

問題文の追加

PenguinJudge では `README.md` で問題文を記述します。Markdown記法に加えてTeX記法も使うことができます。以下は問題文のテンプレートです。

```
## 問題文

## 制約

## 入力

## 出力

## 注意事項

## 入力例1

## 出力例1
```

問題文の追加 | 問題文と制約の記述

入出力を受け取ってその合計を表示させるような問題を出題してみましょう。

```
## 問題文

自然数 $a$, $b$ が与えられたとき、$a + b$ を求めてください。

## 制約

- $a$, $b$ はともに自然数です。
- $1 \leq a \leq 100$
- $a \leq b \leq 100,000$
```

問題文の追加 | 入出力の記述

a, b はスペース区切りで与えられることを問題文として明示します。

````
## 入力

```
a b
```

## 出力

$a + b$ を出力してください。なお、最後に改行を出力すること。
````

問題文の追加 | 入出力例の記述

解答をテストするための入出力をいくつか例示しましょう。

````
## 入力例1

```
10 20
```

## 出力例1

```
30
```
````

問題を難しくする場合、入力の制約の境界値付近のテストケースを追加したり、誤ったアルゴリズムを通さないためのコーナーケースを追加することがあります。

想定解答の追加

`sample-solution` というフォルダにPython解答を追加してみましょう。
`rime add <problem_directory> solution <solution_directory>` コマンドを使います。

```
$ rime add welcome solution sample-solution
[   ADD    ] /home/***/sample-contest/welcome/sample-solution/SOLUTION

Error Summary:
Total 0 errors, 0 warnings
```

vi が起動しますので、解答設定を記述します。詳細は次ページで説明します。
解答設定は後で編集できますので、そのまま編集せずに保存しても問題ありません。

SOLUTION ファイルの編集

今回はPython解答を追加しますので、15行目のコメントアウトを外します。

```
# -*- coding: utf-8; mode: python -*-

## Solution
#c_solution(src='main.c') # -lm -O2 as default
#cxx_solution(src='main.cc', flags=[]) # -std=c++11 -O2 as default
...
#script_solution(src='main.pl') # shebang line is required
script_solution(src='main.py') # shebang line is required <- ここのコメントアウトを外します。
#script_solution(src='main.rb') # shebang line is required
#js_solution(src='main.js') # javascript (nodejs)
#hs_solution(src='main.hs') # haskell (stack + ghc)
#cs_solution(src='main.cs') # C# (mono)

## Score
#expected_score(100)
```

Python解法の追加

`sample-solution` ディレクトリに `main.py` を追加します。

```
$ touch welcome/sample-solution/main.py
$ vi welcome/sample-solution/main.py
```

Rime でテストを行うために1行目に **必ず Shebang を記述する**　必要があります。
PenguinJudge では Python3.7 が使えますので、 `python3` を指定します。

```
#!/usr/bin/env python3
a, b = map(int, input().split())
print(a + b)
```

想定解答は以下の処理を行っています。

1. 標準入力を受け取り、それをスペース区切りして int 型にキャストする
2. 和を求める

テストセットの作成

ユーザが投入した解答をテストするために生成器と検証器を作ります。検証器は入力が制約違反をしていないかを確かめるために利用します。

`rime add <problem_directory> testset <test_directory>` コマンドを使います。

```
$ rime add welcome testset tests
[   ADD    ] welcome: /home/***/sample-contest/welcome/tests/TESTSET

Error Summary:
Total 0 errors, 0 warnings
```

vi が起動しますので、解答設定を記述します。詳細は次ページで説明します。
解答設定は後で編集できますので、そのまま編集せずに保存しても問題ありません。

TESTSET の編集

今回はPythonで生成器と検証器を作ることにします。9行目と18行目のコメントアウトを外して `generator.py`, `validator.py` というファイルをこれから作ります。

```
## Input generators.
#c_generator(src='generator.c')
...
#script_generator(src='generator.pl')
script_generator(src='generator.py')  # コメントアウトを外します

## Input validators.
#c_validator(src='validator.c')
...
#script_validator(src='validator.pl')
script_validator(src='validator.py')  # コメントアウトを外します

## Output judges.
#c_judge(src='judge.c')
...
```

問題生成器 generator.py の作成 1/2

ファイルで入出力を処理するため、ファイル出力用の関数を作成すると便利です。

Rime で実行する都合上、ここでも Shebang が必要です。

```
#!/usr/bin/env python3

# a, b が与えられたときに、その内容をファイル出力する関数
def make_file(a, b, prefix, number):
    statement = f'{a} {b}\n'
    with open(f'{prefix}{number:02}.in', 'w') as f:
        f.write(statement)


# 制約を変数として定義しておく
min_a, max_a = 1, 100
min_b, max_b = 1, 100000

# 入力例1 に対応する問題を生成する
make_file(10, 20, 'sample', 1)
```

問題生成器 generator.py の作成 2/2

いくつかランダムに生成した入力もテストケースに入れてみましょう。

```
# maximum values
make_file(max_a, max_b, 'maximum', 1)

# minimum values
make_file(min_a, min_b, 'minimum', 1)

# random solutions
for number in range(1, 11):
    a = random.randint(min_a, max_a)
    b = random.randint(min_b, max_b)
    make_file(a, b, 'random', number)
```

コーナーケースなどは必要に応じて追加してください。

問題検証器 validator.py の生成

入力を標準入力から受け取り、その値が正しいことを確認します。

Rime で実行する都合上、ここでも Shebang が必要です。

```
#!/usr/bin/env python3
a, b = map(int, input().split())

assert 1 <= a <= 100
assert 1 <= b <= 100000
```

Python で検証を行う場合、`assert` を使うことができます。

想定解答の指定

Rime は想定解答を指定しない場合、アルファベット順で一番若い解答を想定解答として利用して出力を生成し、他の回答のテストを行います。

明示的に想定解答を指定したい場合には、`PROBLEM` ファイルの `reference_solution` に想定解答の存在するディレクトリを指定します。

```
$ vi welcome/PROBLEM
```

```
problem(
  time_limit=5,
  id="welcome",
  title="Welcome to Penguin Judge",
  reference_solution='sample-solution',  # 想定解答のディレクトリを指定します
  #wiki_name="Your pukiwiki page name", # for wikify plugin
  #assignees=['Assignees', 'for', 'this', 'problem'], # for wikify plugin
  #need_custom_judge=True, # for wikify plugin
)
```

問題のテスト

`rime test` コマンドを使うことで、入出力の生成からテストまで自動実行されます。

```
$ rime test
[ GENERATE ] welcome/tests: generator.py
[ VALIDATE ] welcome/tests: OK
[  REFRUN  ] welcome/sample-solution
[   TEST   ] welcome/sample-solution: max 0.02s, acc 0.25s

Build Summary:
welcome ... in: 114B, diff: 72B, md5: -
  sample-solution SCRIPT 3 lines, 69B

Test Summary:
welcome ... 1 solutions, 14 tests
  sample-solution  OK  max 0.02s, acc 0.25s

Error Summary:
Total 0 errors, 0 warnings
```

問題を指定してテストする場合には `rime test <problem_directory>` を使います。

PenguinJudge 用に問題を一式揃える 1/2

`rime pack` コマンドを使うことで入出力とREADMEを一式まとめることができます。

```
$ rime pack
[ GENERATE ] welcome/tests: generator.py
[ VALIDATE ] welcome/tests: OK
[  REFRUN  ] welcome/sample-solution

Error Summary:
Total 0 errors, 0 warnings
```

PenguinJudge 用に問題を一式揃える 2/2

生成した問題は `<problem_directory>/rime-out/penguin` 配下に出力されます。

```
$ tree welcome/rime-out/penguin -L 1
welcome/rime-out/penguin
├── README.md
├── input
└── output

2 directories, 1 file
```

想定している入出力ファイルが全て存在していることを確認しましょう。

PenguinJudge に問題をアップロードする

`rime upload` コマンドを使うことで PenguinJudge に問題をアップロードできます。
`API URL`, `UserID`, `Password` は環境に合わせて適宜入力してください。

```
$ rime upload
API URL: http://localhost:5000
UserID: admin
Password:
[ GENERATE ] welcome/tests: generator.py
[ VALIDATE ] welcome/tests: OK
[  REFRUN  ] welcome/sample_solution

Error Summary:
Total 0 errors, 0 warnings
```

ブラウザを開いて PenguinJudge にアクセスしてみましょう！