Домашнее задание №7

Starovoytov Alexandr

Скриптовый язык, на котором будет выполняться лабораторная работа, студентом выбирается самостоятельно. Примеры возможных скриптовых языков: JavaSciрt (Node.js), Python, Ruby, Lua, Perl, Racket и т.д.

1. Утилита `tree`

Реализуйте собственный вариант утилиты `tree`. Пусть ваша программа поддерживает по меньшей мере ключи `-d` и =-o= так же, как реализация утилиты `tree` в ОС Linux. Поддержку других ключей можно не реализовывать.

Для «рисования» дерева в консоли используйте символы псевдографики.

Программа не должна аварийно завершаться, если права доступа запрещают получение списка файлов какого-либо каталога.

```
#!/usr/bin/env raku

sub output-node(@prefix) {
    for @prefix -> $el {
        given $el {
            when 1 {
                print "├── ";
            }
            when 2 {
                print "└── ";
            }
            when 3 {
                print "│   ";
            }
            when 4 {
```

```
                print "    ";
            }
        }
    }
}

sub MAIN(
    Str $path = ".",
                Bool :$d = False,
                           Str :$o
) {
    if $o.defined {
        my $output-file = open $o, :w;
        $*OUT = $output-file;
    }
    my $dirs = 0;
    my $files = 0;
    my @todo = [($path.IO, []),];
    while @todo {
        my $pair = @todo.pop;
        my $path = $pair[0];
        my @prefix = $pair[1];
        if @prefix {
            output-node(@prefix);
            print $path.basename;
            given $path {
                when .d {$dirs += 1};
                when .f {$files += 1};
            }
        } else {
            print $path;
        }
        if $path.d {
            try {
                for @prefix <-> $el {
                    given $el {
                        when 2 {$el = 4};
                        when 1 {$el = 3};
                    }
                }
```

```
                @prefix.push(1);
                my @contents = [($_, @prefix.clone)
                                for ($d
                                     ?? dir($path,
                                            test => { "$path/$_".IO.d
        ?& ! /^^ \./ })
                                     !! dir($path,
                                            test => { ! /^^ \./
        })).sort(&lc)];
                if @contents {
                    @contents[*-1;1;*-1] = 2;
                }
                @contents.=reverse;
                @todo.append(@contents);
                CATCH {
                    default {
                        note " [error opening dir]";
                    }
                }
            }
        }
        say "";
    }
    if $d {
        say "$dirs directories";
    } else {
        say "$dirs directories, $files files";
    }
}
```

2. Утилита grep

Реализуйте собственный вариант утилиты `grep`. Допускается ограничиться работой только с текстовыми файлами. Так же, как и стандартная утилита `grep`, ваша программа должна обрабатывать как стандартный ввод, так и файлы, пути к которым указаны в командной строке. Ключ `-e` должен позволять передать программе регулярное выражение вместо строки для поиска. Пусть ваша реализация также поддерживает ключи `-i`, `-m`, `-n` так же, как это делает стандартная реализация утилиты `grep`. Поддержку других ключей можно не реализовывать.

Программа не должна аварийно завершаться, если какой-либо из файлов, перечисленных в аргументах командной строки, не может быть прочитан.

Сообщения об ошибках и предупреждения должны направляться в стандартный поток вывода ошибок. **Направление таких сообщений в стандартный поток вывода не допускается.**

```
#!/usr/bin/env raku

sub MAIN(
    Str $path?,
    Str $pattern = "",
    Str :$e,
    Bool :$i,
    Bool :$n,
    Int :$m
) {
    my @data;
    with $path {
        with $path.IO.r {
            @data = slurp($path).lines;
        } else {
            note "error reading $path";
            return;
        }
    } else {
        @data = slurp().lines;
    }
    my $regex;
    with $e {
        $regex = $e;
    } else {
        $regex = $pattern;
    }
    my $k = 1;
    for @data -> $line {
        if $i {
            $line ~~ rx/:i $<result>=<$regex>/;
        } else {
            $line ~~ rx/$<result>=<$regex>/;
        }
```

```
        if $<result> {
            if $n {
                say "$k:$line"
            } else {
                say $line;
            }
        }
        with $m {
            last if $m == $k;
        }
        $k += 1;
    }
}
```

3. Утилита wc

Реализуйте собственный вариант утилиты `wc`. Так же, как и стандартная утилита `wc`, ваша программа должна обрабатывать как стандартный ввод, так и файлы, пути к которым указаны в командной строке. Пусть ваша реализация поддерживает ключи `-c`, `-m`, `-w`, `-l` так же, как это делает стандартная реализация утилиты `wc`. Поддержку других ключей можно не реализовывать.

Сообщения об ошибках и предупреждения должны направляться в стандартный поток вывода ошибок. **Направление таких сообщений в стандартный поток вывода не допускается.**

```
#!/usr/bin/env raku

sub get-stats($data, $name, $m, $c, $w, $l) {
    if !$m ?& !$c ?& !$w ?& !$l {
        print $data.chars, " ", $data.words.elems, " ",
         $data.encode.bytes, " ";
    } else {
        if $l {
            print $data.lines.elems, " ";
        }
        if $w {
            print $data.words.elems, " ";
        }
        if $m {
```

```
            print $data.chars, " ";
        }
        if $c {
            print $data.encode.bytes, " ";
        }
    }
    if $name {
        say $name;
    } else {
        say "";
    }
}

sub MAIN(
    Bool :$m = False, #= print the character counts
    Bool :$c = False, #= print the byte counts
    Bool :$w = False, #= print the word counts
    Bool :$l = False, #= print the newline counts
    *@files
) {
    if !@files {
        get-stats(slurp(), "", $m, $c, $w, $l);
    }
    for @files -> $file {
        if $file.IO.r {
            get-stats($file.IO.slurp(), "$file", $m, $c, $w, $l);
        } else {
            note "can't read $file";
        }
    }
}
```

4. Поиск опечаток

Реализуйте простейшую программу проверки орфографии. Пусть программа принимает на вход словарь и текст на естественном языке и выводит список и координаты слов (строка, колонка), которые не встречаются в словаре.

Например, пусть =dictionary.txt= — словарь, а =example-missprint.txt= — текст, где в строке 1 допущена опечатка в слове `general`, во 2 строке — в слове `emphasizes` и в 7 строке — в слове `supports` (1-е буквы этих слов находятся в 25, 23 и 8 колонках соответственно). Тогда вызов и результат работы вашей программы `speller.py` должен выглядеть так:

```
> ./speller.py dictionary.txt example-missprint.txt
1,  25    gneral
2,  23    emphasises
7,   8    suports
```

Считайте, что в проверяемом тексте переносы слов отсутствуют. Различные формы одного слова рассматривайте как разные слова. Апостроф считайте частью слова.

Рекомендации

В виде отдельного модуля реализуйте сканер, преобразующий текст в токены — слова и знаки пунктуации. Для каждого токена храните его координаты в исходном тексте — позицию от начала текста, номер строки, номер колонки.

Тестирование программы выполните на примерах коротких английских текстов.

Словарь получите из текста, в котором, как вы считаете, отсутствуют опечатки. Для получения отдельных слов из этого текста используйте разработанный вами сканер. Напишите вспомогательную программу, которая будет строить словарь по тексту, поданному на вход этой программы.

Решение

```
#!/usr/bin/env raku

grammar G {
    token TOP { [<separator> <word> <separator>]* }
    token punctuation { <+:punct -[']>* }
    token separator { <.ws> <punctuation> <.ws> }
    token word { ["'" | \w]+ }
}

multi MAIN(
    Str $dictionary where *.IO.f,
```

```
    Str $text where *.IO.f
) {
    my @dict = $dictionary.IO.lines;
    my $i = 0;
    for $text.IO.lines -> $line {
        my $match = G.parse($line);
        for $/<word> -> $word {
            my $str-word = $word.Str;
            if !($str-word (elem) @dict) {
                say $i, " ", $word.pos, " ", $str-word;
            }
        }
        $i += 1;
    }
}

multi MAIN(
    Str $text where *.IO.f,
    Bool :$c
) {
    my $output-file = open "dict", :w;
    $*OUT = $output-file;
    my $match = G.parse($text.IO.slurp);
    for [$_.Str for $/<word>] -> $word {
        say $word;
    }
}
```

Ачивка

В качестве скриптового языка выбрать какой-нибудь редкий или необычный язык **(1 балл).**