Assignment 1 - Perl2Python

- Mon Sep 09 15:30 Version 0.1 - alpha release
- Mon Sep 16 15:30 Version 0.2 - beta release (more information about subsets but examples TBA)
- Tue Sep 07 12:00 Version 0.3 - indictive assessment revised, subsets refined
- Mon Sep 23 13:20 Version 0.4 - subset refined, examples added, various typos fixed

This assignment aims to give you:

- practice in Perl programming generally
- experience in translating between complex formats with Perl
- introduce you to Python semantics & clarify your understanding of Perl semantics

Note: the material in the lecture notes will not be sufficient by itself to allow you to complete this assignment. You may need to search on-line documentation for Perl etc. Being able to search documentation efficiently for the information you need is a *very* useful skill for any kind of computing work.

Introduction

Your task in this assignment to write a Perl compiler. Generally compilers take a high level language as input and output assembler, which can then can be directly executed. Your compiler will take a Perl script as input and output a Python script. Such a translation is useful because programmers sometimes need to convert Perl scripts to Python.
Possible reasons for this include integration of existing Perl code into a Python program and shifting a complete Perl prgram to a new platform which requires Python, such as Google's app engine.

Your task in this assignment is to automate this conversion. You must write a Perl program which takes as input a Perl script and outputs an equivalent Python program.

The translation of some Perl code to Python is straightforward. The translation of other Perl code is difficult or infeasible. So your program will not be able to translate all Perl code to Python. But a tool that performs only a partial translation of Perl to Python could still be very useful.

You should assume the Python code output by your program will be subsequently read and modified by humans. In other word you have to output readable Python code. For example, you should aim to preserve variable names and comments, and to output properly indented Python code.

You must write your translator in Perl (for assignment 2 there will be no restriction on language).

You should call your Perl program `perl2python`. It should operate as Unix filters typically do, read files specified on the command line and if none are specified it should read from standard input (Perl's <> operator does this for you). For example:

```
% cat iota.pl
#!/usr/bin/perl
$x = 1;
while ($x < 5) {
        print "$x\n";
        $x = $x + 1;
}
% perl iota.pl
0
1
2
3
4
% ./perl2python iota.pl|python
0
1
2
3
4
% ./perl2python iota.pl >iota.py
% cat iota.py
#!/usr/bin/python
x = 1
while x <  5:
    print x
    x = x + 1
% python iota.py
0
1
2
3
4
```

In many cases the natural translation of Perl code into Python code will have slightly different semantics. For example, Python's `print` statement will print an initial space if the preceding character printed was not a new line - Perl's `print` does not do this. Python's (pre-3.0) division operator (/) truncates its result to an integer if given two integer arguments . Perl does not.

This is a general issue with translating between languages. It is essential that a compiler such as `gcc` produce a translation to assembler exactly matching the semantics of the program, but our purposes are different.

Our goal is to provide automated-assistance in converting a piece of software. The software will need to subsequently maintained and modified by humans so the simpler natural translation to more human-readable code is more desirable even if the semantics do not match exactly. Translation of large piece of software will not be completely automatic anyway and will require subsequent manual modification.

Requirements

To assist you tackling the assignments requirements have been broken into subsets in approximate order of difficulty. Each subset contains the previous subsets. This implementation order is recommended not required. You will receive marks for successfully translating features regardless of which subset they are

in, even if previous subsets are unimplemented.

Subset 0

- simple print statements (with explicit new lines)
- strings

examples of subset 0 (examples/0)

Subset 1

- variables
- numeric constants
- arithmetic operators: `+ - * / % **`

examples of subset 1 (examples/1)

Subset 2

- logical operators: `|| && ! and or not`
- comparison operators: `<, <=, >, >=, <>, !=, ==`
- bitwise operators: `| ^ & << >> ~`
- if, for, while statements
- break, continue

examples of subset 2 (examples/2)

Subset 3

- simple use of
- more complex print statements (e.g. without new lines)
- simple uses of ++ & --
- ..
- chomp, split, join, ARGV

examples of subset 3 (examples/3)

Subset 4

- <>
- . .=
- chomp
- variable interpolation
- simple uses of %
- arrays and hashes
- push, pop, shift, unshift, reverse
- simple use of regexes (// and s///)

example of subset 4 (examples/4)

Subset 5

- anonymous variable ($_)
- open
- functions
- more complex uses of features from subsets 1-4
- self-application (translating itself to python)

You may suggest in the forum other features to be added to the list for subset 5. Some features in subset 5 present great difficulties to translation perfect handling of these will not be required for full marks.

examples of subset 5 (examples/5)

Not Included

The features you need implement are described in the subsets above so, for example, you don't have to translate these Perl features:

Assumptions/Clarifications

Like all good programmers, you should make as few assumptions about your input as possible.
You can assume that the input to your program is a valid Perl program and it executes without error.

It is possible to format Perl programs so that they are difficult to translate. Most of the Perl programs your program will be given will be formated in a similar way to use for lecture and lab examples. But your should attempt to successfully translate at least small variations to that style and preferably any formatting style that a reasonable Perl programmer is likely to use.

It is possible to construct obscure and difficult to translate Perl programs using the features list in the above subsets. Most of the Perl programs your program will be given as input will use the features in an obvious straight-forward manner.

You may use any Perl package which is installed on CSE's system.

You may submit multiple files but the primary file must be named `perl2python`.

If there is Perl that you cannot translate the preferred behaviour is to include the untranslated Perl construct as a comment. Other sensible behaviour is acceptable.

Hints

You should follow discussion about the assignment in the course forum (https://cgi.cse.unsw.edu.au/~forums/support/viewforum.php?f=1190). Questions about the assignment should be posted there so all students can see the lecturer's answer. If you need an urgent reply also e-mail the lecturer.

Demo & Test Inputs

You should submit ten Perl scripts named `demo00.pl .. demo09.pl` which your program translates correctly (or at least well). These should be realistic Perl scripts containing features whose successful translation indicates the performance of your assignment. Your demo scripts don't have to be original, e.g. they might be lecture examples. If they are not original they should be correctly attributed.
You should submit ten Perl scripts named `test00.pl .. test09.py` which test single aspects of translation. They should be short scripts containing Perl code belonging to the subsets for this assignment, which are likely to be good tests of a Python to Perl translator. In other words they are examples of Perl a student assignment might mis-translate. The `test??.pl` scripts do not have to be examples that your program translates successfully.

In summary the demo scripts should show off your translation - they can be long and test many aspects of Perl to Python translation. The test scripts should show how you've thought about testing carefully - and they should be short and each test only a few aspects of Perl to Python translation. p>

Getting Started

Here is the source (perl2python.level0) to a Perl script which translates correctly a very small amount of Python to Perl including examples//0/hello_world.pl (examples/0/hello_world.pl). For example:

```
% d=/home/cs2041/public_html/assignments/perl2python/
% cp $d/perl2python.level0 perl2python
% ln -s $d/examples .
% perl examples/0/hello_world.pl
hello world
% perl2python examples/0/hello_world.pl >h.py
% perl h.py
hello world
%
```

Version History & Diary

You need to maintain a version history for your assignment in a git repository. You should submit this in a tar file `git.tar`.
Git will be covered later in lectures but here is a quick tutorial on the what to do:

```
% mkdir -p 2041/ass1
% chmod 700 2041/ass1
% cd 2041/ass1
% git init
Initialized empty Git repository in ...
% cp /home/cs2041/public_html/assignments/perl2python/perl2python.level0 perl2python
% git add perl2python
% git commit -a -m 'starting point'
....
% vi perl2python
% git commit -a -m 'added code to handle while loops'
....
% vi perl2python
% git commit -a -m 'fix bug with formatting'
....
% ....
% ....
% ....
% git commit -a -m 'tidied up assignment for submission'
% tar jcf git.tar .git
% give cs2041 ass1 perl2python diary.txt git.tar test??.py demo??.py
```

You must keep notes of each piece of you make work on this assignment. The notes should include date, starting&finishing time, and a brief description of the work carried out. For example:

Date	Start	Stop	Activity	Comments
29/09	16:00	17:30	coding	implemented for loop
30/09	20:00	10:30	debugging	found bug in string handing

Include these notes in the files you submit as diary.txt.

You may choose to store this information in git commit messages. and generate `diary.txt` from `git log` before you submit.

Getting Started

Here is the source (perl2python.pl) to a Perl script which translates correctly a very small amount of Perl to Python including examples//0/hello_word.sh (examples/0/variables0.pl). For example:

```
% cd
% mkdir -p 2041/ass1
% chmod 700 2041/ass1
% cd 2041/ass1
% d=/home/cs2041/public_html/assignments/perl2python/
% cp $d/perl2python.pl .
% ln -s $d/examples .
% pl examples/0/hello_world.pl
hello world
% perl2python.pl examples/0/hello_world.pl >hello_world.py
% cat hello_world.py
#!/usr/bin/python2.7 -u
print 'hello world'
% python hello_world.py
hello world
% pl examples/0/hello_world.pl >pl.output
% python -u hello_world.py >py.output
% diff py.output pl.output && echo success
success
% git init
Initialized empty Git repository in ...
% git add perl2python.pl
% vi perl2python.pl
% git commit -a -m 'added code to handle for loops'
% vi perl2python.pl
% git commit -a -m 'added code to produce arithmetic'
% ...
% git commit -a -m 'tidied up assignment for submission'
% tar jcf git.tar .git
% give cs2041 ass1 perl2python.pl diary.txt git.tar demo??.pl test??.pl
```

You need to maintain a version history for your assignment in a git repository which you should submit in a tar file named `git.tar`.

Git will be covered later in lectures but the above commands are enough to get by.

Assessment

Assignment 1 will contribute 15 marks to your final COMP2041/9041 mark

15% of the marks for assignment 1 will come from hand marking. These marks will be awarded on the basis of clarity, commenting, elegance and style. In other words, you will be assessed on how easy it is for a human to read and understand your program.

5% of the marks for stage 1 will be based on the test suite you submit.

80% of the marks for assignment 1 will come from your translators performance on a set of input Perl programs.

<

100%	Subsets 0-4 handled perfectly, subset 5 largely working, Perl is beautiful
90%	Subsets 0-4 handled, Perl is very clear & very readable
75%	Subsets 0-3 handled, Perl is good clear code
65	Subsets 0-2 handled, Perl is reasonably readable
55%	Subset 0-1 translated more-or-less
0%	Knowingly providing your work to anyone and it is subsequently submitted (by anyone).
0 FL for COMP2041/9041	Submitting any other person's work. This includes joint work.
academic misconduct	Submitting another person's work without their consent.

The lecturer may vary the assessment scheme after inspecting the assignment submissions but its likely to be broadly similar to the above.

Originality of Work

The work you submit must be your own work. Submission of work partially or completely derived from any other person or jointly written with any other person is not permitted. The penalties for such an offence may include negative marks, automatic failure of the course and possibly other academic discipline. Assignment submissions will be examined both automatically and manually for such submissions.

Relevant scholarship authorities will be informed if students holding scholarships are involved in an incident of plagiarism or other misconduct.

Do not provide or show your assignment work to any other person - apart from the teaching staff of COMP2041/9041. If you knowingly provide or show your assignment work to another person for any reason, and work derived from it is submitted you may be penalized, even if the work was submitted without your knowledge or consent. This may apply even if your work is submitted by a third party unknown to you.

Note, you will not be penalized if your work is taken without your consent or knowledge.

Submission

This assignment is due at 23:59pm Monday October 7 Submit the assignment using this *give* command:

```
    tar jcf  git.tar .git
give cs2041 ass1 perl2python.pl diary.txt git.tar demo??.py test??.py [any-other-files]
```

If your assignment is submitted after this date, each hour it is late reduces the maximum mark it can achieve by 1%. For example if an assignment worth 76% was submitted 12 hours late, the late submission would have no effect. If the same assignment was submitted 36 hours late it would be awarded 64%, the maximum mark it can achieve at that date.