Assignment 1 - plpy

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 3 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 `plpy.pl`. 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
1
2
3
4
$ ./plpy.pl iota.pl|python3.5 -u
1
2
3
4
$ ./plpy.pl iota.pl>iota.py
$ cat iota.py
#!/usr/local/bin/python3.5 -u
x = 1
while x < 5:
    print(x)
    x = x + 1
$ python3.5 -u iota.py
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.

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 a large piece of software can 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: examples (examples/0/)

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

Subset 1: examples (examples/1/)

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

Subset 2: examples (examples/2/)

- logical operators: `|| && ! and or not`

- comparison operators: `<, <=, >, >=, <=>, !=, ==`
- bitwise operators: `| ^ & << >> ~`
- if/else/elsif, for/foreach, while statements
- last, next

Subset 3: examples (examples/3/)

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

Subset 4: examples (examples/4/)

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

Subset 5: examples (examples/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.

Not Included

The only features you need implement are described in the subsets above. If uncertain ask in the class forum.

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. You can assume this Perl program executes without errors or warnings (with -w specified).

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 lecture and lab example Perl programs. 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 `plpy.pl`.

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.

The Python you produce should be compatible with Python 3.5 (as installed at CSE). Use the #! line shown in examples.

Hints

You should follow discussion about the assignment in class forum Questions about the assignment should be posted there so all students can see the answer. If you need an urgent reply also e-mail Andrew.
Get the easiest transformations working first, make simplifying assumptions as needed, and get some simple small Perl scripts successfully transformed. Then look at handling more constructs and removing the assumptions.

You won't be able to output Python as you generate it e.g. you won't know which import statements are needed to be printed first. Push the Python code to an array as you generate it.

If you want a good mark, you'll need to be careful in your handling of syntax which has no special meaning in Perl but does in Python.

The bulk of knowledge about Perl syntax & semantics you need to know has been covered in lectures. But if you want to get a very high mark, you may need to discover more. Similarly much of the knowledge of Python you need can be determined by looking at a few of the provided examples but if you want to get a very high mark you will need to discover more. This is deliberate as extracting this type of information from documentation is something you'll do a lot of when you graduate.

Testing

As usual some autotests will be available:

```
$ ~cs2041/bin/autotest plpy
```

You will need to do most of the testing yourself.

Test Perl Scripts

You should submit five Perl scripts named `test00.pl .. test04.pl` which each test a single aspect of translation. They should be short scripts containing Perl code which is likely to be mis-translated. The `test??.pl` scripts do not have to be examples that your program translates successfully.
You may share your test examples with your friends but the ones you submit must be your own creation.

The test scripts should show how you've thought about testing carefully. They should be as short as possible (even just a single line).

Demo Perl Scripts

You should submit five Perl scripts named `demo00.pl .. demo04.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.
If your have implemented most of the subsets these should be longer Perl scripts (20+ lines). They should if possible test many aspects of Perl to Python translation.

Setting up a Git Repositories

A repository has been created for your assignment on `gitlab.cse.unsw.edu.au` .
You need to add your CSE ssh key to your `gitlab.cse.unsw.edu.au` .

If you have not done so already, follow the instructions for doing this can be found at the start of lab 7 to add your CSE ssh key to your `gitlab.cse.unsw.edu.au` .

After you have done that create a git repository for the assignment in your CSE account.

These commands will create a copy of the gitlab repository in your CSE account.

Make sure you replace *5555555* below by your student number!

```
$ cd
$ git clone gitlab@gitlab.cse.unsw.EDU.AU:z5555555/16s2-comp2041-ass1 ass1
Cloning into 'ass1'...
$ chmod 700 ass1
$ cd ass1
$ ln -sf /home/cs2041/public_html/scripts/autotest-pre-commit-hook .git/hooks/pre-commit
$ ls
diary.txt examples  plpy.pl
```

Make sure you do the chmod above, so your work is not accessible to other students.

Place all your files for the assignment in this **ass1** directory. A file named plpy.pl (plpy.pl) should be present in your repository. It contains a small amount of Perl to get you started. There should also be a directory named examples (examples).

Try out the starting-point code to see if it works.

```
$ chmod 700 plpy.pl
$ ./examples/0/hello_world.pl
hello world
$ ./plpy.pl examples/0/hello_world.pl >hello_world.py
$ cat hello_world.py
#!/usr/local/bin/python3.5 -u
print("hello world")
$ ./hello_world.py
hello world
$ ./examples/0/hello_world.pl >pl.output
$ ./hello_world.py >py.output
$ diff py.output pl.output && echo success
success
```

` Note the starting-point code has successfully translated a small Perl program (examples/0/hello_world.pl (examples/0/hello_world.pl)) to Python.
Sadly this is almost the only input it can correctly handle.

Your job is to fix that.

First make a change to starting point code - e.g. change the initial comment to remove Andrew's name (using your editor of choice) and add your name - then push this to gitlab as your first commit, like this:

```
$ vi plpy.pl
$ git add plpy.pl
$ git commit -a -m "Andrew's code"
[master 4cdfa5f] Andrew's code
 1 file changed, 17 insertions(+)
 create mode 100755 plpy.pl
$ git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 239 bytes, done.
Total 2 (delta 1), reused 0 (delta 0)
To gitlab@calliope1.cse.unsw.EDU.AU:16s2COMP2041/z5555555-ass1.git
   36ccb2b..4cdfa5f  master -> master
```

If you want to check that your gitlab repository has stored your commit visit `https://gitlab.cse.unsw.edu.au/z5555555/16s2-comp2041-ass1.git` (replacing *5555555* with your student number)
Here are examples of how you might push work to gitlab after completing parts of the assignment.

```
$ vi plpy.pl
$ vi diary.txt
$ git commit -a -m 'added code to handle for loops'
$ git push
[master 5a11cef] added code to handle for loops
 2 files changed, 2 insertions(+)
 create mode 100644 diary.txt
$ vi plpy.pl
$ vi diary.txt
$ git commit -a -m 'added code to produce arithmetic'
[master 1e3fe75] added code to produce arithmetic
 2 files changed, 2 insertions(+)
$ git push
Counting objects: 10, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (8/8), 617 bytes, done.
Total 8 (delta 2), reused 0 (delta 0)
To gitlab@gitlab.cse.unsw.EDU.AU/z5555555/16s2-comp2041-ass1.git
   63655f5..1e3fe75  master -> master
$ vi demo00.pl demo01.pl test00.pl test01.pl test02.pl
$ git add demo00.pl demo01.pl test00.pl test01.pl test02.pl
$ git commit -a -m 'created some demo & test examples'
[master 226cddf] created some demo & test examples
 0 files changed
 create mode 100644 demo00.pl
 create mode 100644 demo01.pl
 create mode 100644 test00.pl
 create mode 100644 test01.pl
 create mode 100644 test02.pl
$ git push
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 352 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gitlab@gitlab.cse.unsw.EDU.AU/z5555555/16s2-comp2041-ass1.git
   1e3fe75..226cddf  master -> master
$ ...
$ give cs2041 ass1 plpy.pl diary.txt demo??.pl test??.pl
```

I expect most students will just work in their CSE account and push work to gitlab.cse.unsw.edu.au from CSE, but you can try setting up a git repository on your home machine and pushing work to gitlab.cse.unsw.edu.au from there.
If you do this you'll want to use git's pull command to update the repository in your CSE account.

```
$ git pull
Unpacking objects: 100% (3/3), done.
From gitlab@gitlab.cse.unsw.EDU.AU/z5555555/16s2-comp2041-ass1.git
   226cddf..e64fee9  master     -> origin/master
Updating 226cddf..e64fee9
Fast-forward
 plpy.pl |    1 +
 1 file changed, 1 insertion(+)
```

If you can't get ssh access to gitlab to work for your home repository, you can also use https to access gitlab using a URL equivalent to `https://gitlab.cse.unsw.edu.au/z5555555/16s2-comp2041-ass1.git` (replace *5555555* with your student number) and your z-id & zPass.

```
$ git remote set-url origin https://gitlab.cse.unsw.edu.au/z5555555/16s2-comp2041-ass1.git
$ git push
Username for 'https://gitlab.cse.unsw.EDU.AU': z5555555
Password for 'https://z5555555@gitlab.cse.unsw.EDU.AU': zPass
```

If the supplied files for the assignment are updated you can merge the changes into your git repository like this:

```
$ git remote add example_code gitlab@gitlab.cse.unsw.EDU.AU:16s2COMP2041/16s2-comp2041-ass1-su
$ git pull
```

Git will be covered later in lectures but the above commands should be enough to get by. Help will be available in the forums, and from your tutor if anything goes wrong.

Diary

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	16:00	17:30	coding	implemented assignment statements
30/09/16	20:00	10:30	debugging	found bug in while loops

Include these notes in the files you submit as an ASCII file named diary.txt.

Some students choose to store this information in git commit messages and use a script to generate `diary.txt` from `git log` before they submit. You are welcome to do this.

Assessment

Assignment 1 will contribute 15 marks to your final COMP[29]041 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. The hand marking will also check that the Python you generate maintains the readability of the input Perl program, you have submitted an appropriate diary and have pushed your work to `gitlab.cse.unsw.edu.au` .

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.

50+ input Perl programs will be used to calculate your performance mark. Your translator will be run on each. The Python it produces will be run on sample input. Your will receive marks if your Python produces the correct output.

100%	Subsets 0-4 handled perfectly, subset 5 largely working, plpy.pl is beautiful
90%	Subsets 0-4 handled, plpy.pl is very clear & very readable
75%	Subsets 0-3 handled, plpy.pl is good clear code
65	Subsets 0-2 handled, plpy.pl 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 COMP[29]041	Submitting any other person's work. This includes joint work.
academic misconduct	Submitting another person's work without their consent. Paying another person to do work for you.

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 suspension from UNSW. 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.

Plagiarism or other misconduct can also result in loss of student visas.

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 3 Submit the assignment using this *give* command:

```
$ give cs2041 ass1 plpy.pl diary.txt demo??.pl test??.pl [any-extra-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 5 hours late, the late submission would have no effect. If the same assignment was submitted 30 hours late it would be awarded 70%, the maximum mark it can achieve at that time.