Assignment 2: sheeple, Sheeple

version: 0.1 last updated: 2020-07-22 17:00

Aims

This assignment aims to give you

- practice in Perl programming generally
- experience in translating between complex formats with Perl
- clarify your understanding of shell syntax & semantics

Introduction

Your task in this assignment to write a shell compiler. Generally compilers take a high level language as input and output assembler, which can then can be directly executed. Your compiler will take shell scripts as input and output Perl. Such a translation is useful because programmers sometimes convert shell scripts to Perl

Possibly motivations including integration of the code to existing Perl, adding extra functionality more easily available from Perl or a need for data structures unavailable in shell.

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

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

You should assume the Perl code output by your program will be subsequently read and modified by humans. In other word you have to output readable Perl code. For example, you should preserve variable names and comments and the code you generate should be sensible formatted.

Your compiler must be written in Perl. You should call your Perl program `sheeple.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 gcc.sh
#!/bin/dash
for c_file in *.c
do
    gcc -c $c_file
done
$ ./sheeple.pl gcc.sh
#!/usr/bin/perl -w
foreach $c_file (glob("*.c")) {
    system "gcc -c $c_file";
}
```

If you look carefully at the example above you will notice the Perl code does not have exactly the same semantics as the shell code - if there are no `.c` files in the current directory the for loop in the shell program executes once and tries to compile a non-existent file named `*.c` whereas the Perl for loop does not execute.

This is a general issue with translating shell to Perl. In many cases the natural translation of the shell code will have slightly different semantics. For some purposes it might be desirable to produce more complex Perl code that matches the semantics exactly, e.g.

```
#!/usr/bin/perl -w
if (glob("*.c") {
    foreach $c_file (glob("*.c")) {
       system "gcc -c $c_file";
    }
} else {
    system "gcc -c '*.c'";
}
```

This is not desirable for our purposes. Our goal is to produce the clearest most-human-readable code so the (simpler) first translation is more desirable.

Subsets

The shell features you need to implement is described in the table below as a series of nested subsets.

It is suggested you tackle the subset in the order listed but this is not required.

Subset Subset	Syntax Syntax	Keywords Keywords	Builtins/ Builtins/ Programs Programs	Variables Variables	Explanation Explanation
0	= $ #		echo		Only simple & obvious statements.
1	? * [ ]	for do done	exit read cd		Only simple & obvious statements. No nesting of for loops. **? * [ ]** for file matching only.
2	' "	if then else elif fi while	test expr	$1 $2 $3 ...	Only simple & obvious statements. No nesting of for/while/if statements.
3	` $() $(()) []		echo -n	$# $* $@	for/while/if statements can be nested. **$()** as equivalent to back quotes. **$(())** for arithmetic. **[]** as equivalent to test
4	< > && || ; {}	case esac local return	mv chmod ls rm		**)** as part of case only. **{}** for functions only. Simple uses of mv/chmod/ls/rm with no command line options.

Examples

Some examples of shell code and possible translations are available here and as a zip file

These examples should provide most the information you need to tackle subsets 0 & 1.

Translating subsets 2-4 will require you to discover information from online or other resources. This is a deliberately part of the assignment.

The Perl you generate can and probably will be different to the examples you've been given.

But a good check of the Perl you generate is to execute it and then and then use `diff` to compare its output is identical to the output from the shell script

Assumptions/Clarifications

Like all good programmers, you should make as few assumptions about your input as possible.

You can assume the code you are given is Shell that works with the version of on CSE systems (essentially POSIX compatible).

Other shells such as Bash contain many other features. If these features and not present in `/bin/dash` CSE machines you do not have to handle these

Notice for example the pipe operator ('|') does not appear above so you not need to translate pipelines. The right parenthesis (')') is in the subset for use in `case` statements only.

You should implement the keywords & builtins listed the above table directly in Perl, you cannot execute them indirectly via `system` or other Perl modules. For example, this is not an acceptable translation.

```
system "for c_file in *.c; do gcc -c $c_file; done";
```

The only shell builtins which you must translate directly into Perl are:

```
exit read cd test expr echo
```

The builtins (`exit read cd`) must be translated into Perl to work. For example this Perl code does not work:

```
system "exit";
```

The last 3 (`test expr echo`) can be, and often are implemented by stand-alone programs. So, for example, this Perl code will work:

```
system "echo hello world";
```

Doing this will receive no marks, instead of using `system.call` you should translate uses of `test`, `expr` and `echo` directly to Perl, e.g.:

```
print "hello world\n";
```

You do have to handle *test* operators such `-r` and `-d`. They look like options but are operators in *test*'s expression language. There will be little testing of *test*'s infrequently used operators.

You don't have to handle *test*'s `-ef` and `-t` operators as they don't have a simple translation to Perl.

The only shell builtin option you need to handle is echo's -n option.

You do not need to handle other echo options such as `-e`. You do not need to handle options for other builtins. For example you do not need to handle the various (rarely used) *read* options.

Bash has many special variables. You need only handle a few of these, which indicate the shell script's arguments. These special variables need be translated:

```
$# $* $@ $1 $2 $3 ...
```

You can assume the shell scripts you are given execute correctly with `/bin/dash` on a CSE lab machine. You program does not have to detect errors in the shell script it is given.

You should assume as little as possible about the formatting of the shell script you are given. However most of the evaluation of your program will be on sensibly formatted shell scripts.

Most of the evaluation of your program will be on shell code that a programmer might normally write to solve a tasks. ocus on translating typical shell scripts, rather than unusual difficult-to-translate Shell.

It is a requirement that any comments in the shell code must be transferred to the Perl.

With some approaches it can be difficult to transfer comments in exactly the same position, in this case it is OK if comments are shifted to some degree.

The Perl generated must be sensible indented and formatted, if input the shell script is sensibly formatted.

If there are shell keywords, e.g. `case`, that you cannot translate the preferred behaviour is to include the untranslated shell construct as a comment. Other sensible behaviour is acceptable.

Your Perl (`sheeple.pl`) should work with the default Perl on a CSE lab machine. You are permitted to use any Perl modules installed on CSE systems.

The Perl you generate should also work with Perl on a CSE lab machine. The Perl you generate should not use Perl modules.

The Perl you generate should not generate warnings from Perl's '-w' flags unless the warning corresponds to an issue with the Shell.

Hints

You should follow discussion about the assignment in the course forum. Questions about the assignment should be posted there so all students can see answers.

Get the easiest transformations working first, make simplifying assumptions as needed, and get some simple small shell scripts successfully transformed. Then look at handling more constructs and removing the assumptions.

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

The bulk of knowledge about shell 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 Perl 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.

Demo Shell Scripts

You should submit five shell scripts named `demo00.sh .. demo04.sh` which your program translates correctly (or at least well). These should be realistic shell 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 shell scripts (20+ lines). They should if possible test many aspects of shell to Perl translation.

Test Shell Scripts

You should submit five shell scripts named `test00.sh .. test04.sh` which each test a single aspect of translation. They should be short scripts containing shell code which is likely to be mis-translated. The `test??.sh` 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).

Diary

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

Date	Start	Stop	Activity	Comments

29/09/15	16:00	17:30	coding	implemented exit
30/09/15	20:00	10:30	debugging	found bug in for loops

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

Change Log

Version 0.1
(2020-07-22 17:00)

- Initial release

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest sheeple
```

Assessment

Submission

When you are finished working on the assignment, you must submit your work by running `give`:

```
$ give cs2041 ass2_sheeple sheeple.pl diary.txt test??.sh [any-other-files]
```

You must run `give` before **Sunday August 09 21:59 2020** to obtain the marks for this assignment. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

You can run `give` multiple times. Only your last submission will be marked.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

You cannot obtain marks by e-mailing your code to tutors or lecturers.

You check the files you have submitted here.

Automarking will be run by the lecturer after the submission deadline, using test cases different to those `autotest` runs for you. (Hint: do your own testing as well as running `autotest`.)

Manual marking will be done by your tutor, who will mark for style and readability, as described in the **Assessment** section below. After your tutor has assessed your work, you can view your results here; The resulting mark will also be available via give's web interface.

Due Date

This assignment is tentatively due **Sunday August 09 21:59 2020**.

If your assignment is submitted after this date, each hour it is late reduces the maximum mark it can achieve by 2%. For example, if an assignment worth 74% was submitted 10 hours late, the late submission would have no effect. If the same assignment was submitted 15 hours late, it would be awarded 70%, the maximum mark it can achieve at that time.

Assessment Scheme

This assignment will contribute 15 marks to your final COMP(2041|9044) mark

15% of the marks for assignment 2 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 assignment 2 will be based on the test suite you submit.

80% of the marks for assignment 2 will come from the performance of your code on a large series of tests.

An indicative assessment scheme follows. The lecturer may vary the assessment scheme after inspecting the assignment submissions, but it is likely to be broadly similar to the following:

100	Subsets 0-4 handled perfectly, sheeple.pl is beautifully readable; excellent test suite
HD (85+)	Subsets 0-3 handled, sheeple.pl is good clear code; great test suite
DN (75+)	Subsets 0-2 handled, sheeple.pl is readable ; good test suite
CR (65+)	Subsets 0-1 handled, sheeple.pl is reasonable readable; OK test suite
PS (55+)	Subset 0 translated more-or-less, test suite partly completed
PS (50+)	Good progress on assignment, but not passing autotests
0%	knowingly providing your work to anyone and it is subsequently submitted (by anyone).
0 FL for COMP(2041|9044)	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.

Intermediate Versions of Work

You are required to submit intermediate versions of your assignment.

Every time you work on the assignment and make some progress you should copy your work to your CSE account and submit it using the `give` command below. It is fine if intermediate versions do not compile or otherwise fail submission tests. Only the final submitted version of your assignment will be marked.

All these intermediate versions of your work will be placed in a Git repository and made available to you via a web interface at `https://gitlab.cse.unsw.edu.au/`*`z5555555`*`/20T2-comp2041-ass2_sheeple` (replacing *z5555555* with your own zID). This will allow you to retrieve earlier versions of your code if needed.

Attribution of Work

This is an individual assignment.

The work you submit must be entirely your own work, apart from any exceptions explicitly included in the assignment specification above. Submission of work partially or completely derived from any other person or jointly written with any other person is not permitted.

You are only permitted to request help with the assignment in the course forum, help sessions, or from the teaching staff (the lecturer(s) and tutors) of COMP(2041|9044).

Do not provide or show your assignment work to any other person (including by posting it on the forum), apart from the teaching staff of COMP(2041|9044). 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 that work was submitted without your knowledge or consent; this may apply even if your work is submitted by a third party unknown to you. You will not be penalized if your work is taken without your consent or knowledge.

Submissions that violate these conditions will be penalised. Penalties may include negative marks, automatic failure of the course, and possibly other academic discipline. We are also required to report acts of plagiarism or other student misconduct: if students involved hold scholarships, this may result in a loss of the scholarship. This may also result in the loss of a student visa.

Assignment submissions will be examined, both automatically and manually, for such submissions.

COMP(2041|9044) 20T2: Software Construction is brought to you by
the School of Computer Science and Engineering
at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs2041@cse.unsw.edu.au
CRICOS Provider 00098G