HKU COMP3235 Assignment 3 (Project - Part I)

Due: Monday, April 24, 2017

** To be done in a group of up to 3; everyone in the group will get the same marks. **

Late penalty: -15% per day.

[[7.4.2017 - Assignment spec posted.]]

Extend c5c (yes, only the compiler) to become c6c by implementing the following.

Task 1. Replace sas by nas - i.e., make c6c generate code for nas instead of sas.

Task 2. Constants. There are these three types: integer, char, and an "nas string" (actually its address). (Sorry, no real numbers, as nas can only handle integers.)

```
x = 'c';
i = 27;
s = "hello, world!"; -- an nas string is created, and its address stored in s
```

Task 3. Variables. The a-z registers in the previous versions are now gone. You can name and create your own "variables". Variable names are alphanumeric; the first letter must be a letter. Case-insensitive. Maximum length 12 characters. A variable can be either local (to a function) or global. Variables are auto-declared on first use. You can interpret "use" to be either an assignment or when values are passed to formal parameters when a function is called (see Task 5 below). There is no typing - a variable can hold an integer or a char, or an "nas string" (actually its address). When inside a function, a reference to a global variable is preceded by the special character "@". Please refer to Slide 18 of L14.pdf for an example.

```
x = 1234; -- x is declared, if this is the first time it appears
...
x = 4567; -- this just changes the value of x

func(a) { ... }
...
i = func(aa); -- local variable a is "declared" in func and initialized with value of aa
```

Task 4. Replace the print and read statements in the c4/5 language by the following, which mirror the I/O instructions of nas:

```
geti(i); getc(c); gets(s); ...
puti(num); puti_(27); putc('x'); ...
puts(s); puts("o"); puts_("Hello!"); ... -- these are nas strings
```

The single argument is actually mapped to the element at the top of the stack in nas.

Task 5. Functions and function calls. A function can have zero or more formal parameters. A parameter is a single/scalar variable (integer, char, or an nas string), and arguments are passed by value. A function's name should not clash with that of a variable. Function definitions may appear anywhere in the code (although defining them all at the beginning is a good practice), and so you might need an extra pass to fill in their addresses in calls. A function has a single return value. Naturally, functions can be recursive.

```
prime(x) {...} -- this defines a function
a = prime(myint); -- this calls a function
if (func(a, b, c)) ...; -- a function with three parameters
```

"nas" is your target machine. If necessary, you could change it to fit your needs, particularly its limits (such as memory and stack sizes). But if you want to change its features such as to add new instructions or the way some of the current instructions work, please discuss with me beforehand.