CSC 526, Spring 2020, Assignment 6

Purpose: User Defined Functions

Due: 11:59pm, Wednesday, April 8, 2020

Get the assignment code

These instructions assume that your course git repository is set up. Change into your course repository directory and enter the following commands.

```
git fetch assignments
git checkout assignments/master -- assignment6
git add assignment6
git commit -a
```

This will copy the `assignment6` directory into your working directory, start tracking the files in the `assignment6` directory, and commit those files to your local git repository.

Assignment Description

The purpose of this assignment is to augment the previous assignment with user-defined functions.

Syntax

```
<program> :=
    | ((define main (input) <expr>))
    | (<definition list> (define main (input) <expr>))

<definition list> :=
    | <definition>
    | <definition> <definition list>

<definition> :=
    | (define <identifier> () <expr>)
    | (define <identifier> (<parameter list>) <expr>)

<parameter list> :=
    | <identifier>
    | <identifier> <parameter list>

<expr> :=
    | <integer>
    | <identifier>
    | true
    | false
    | (let (<bindings>) <expr>)
    | (if <expr> <expr> <expr>)
    | (add1 <expr>)
    | (sub1 <expr>)
    | (is-num <expr>)
```

```
    | (is-bool <expr>)
    | (+ <expr> <expr>)
    | (- <expr> <expr>)
    | (* <expr> <expr>)
    | (< <expr> <expr>)
    | (> <expr> <expr>)
    | (== <expr> <expr>)
    | (print <expr>)
    | <function app>

<bindings> :=
    | (<identifier> <expr>)
    | (<identifier> <expr>) <bindings>

<function app> :=
    | (<identifier>)
    | (<identifier> <expr list>)

<expr list> :=
    | <expr>
    | <expr> <expr list>
```

Functions

Functions are defined using the `define` keyword and require a list of formal parameters and a function body. The formal parameters represent variables that can be used inside the function body. A function can be called from any other function regardless of the order in which they appear in the source program.

Here is an example program:

```
((define add3 (n) (+ n 3))
 (define main (input) (add3 input)))
```

The main aspects of this program are:

- The function `add3` is defined with one argument. The return value of the function is the result of evaluating the function body (expression).
- The function `main` is defined with one argument. The body of the main function calls the `add3` function passing the `input` value as the argument. Note that `input` is an explicit argument to the `main` function.

The body of the main function should be generated under the label `code_entry_point` and follow the standard x86-64 calling convention. This is just like the previous assignment since the `runtime.c` program calls the `code_entry_point` function with the `input` argument. The remaining functions are called within the context of the generated assembly and can have any calling convention that you wish.

Error Checking

The error checking of the previous assignments should be extended to handle the following cases:

- Duplicate function names: all function names must be unique.
- `main`: the function `main` must be present and have a single formal parameter.
- Duplicate formal parameters: all the formal parameters for a given function must have unique names.
- Arity: function application must check that the number of arguments given matches the number of arguments expected.
- Application of unknown function: application of something that is not a function or a function name that does not exist.

For the sake of simplicity and maintaining consistency with the x86-64 calling convention, you may choose to disallow functions that have more than six formal parameters. If you choose to do so, then you need to perform an extra error check that signals an error if a user-defined function has more than six formal parameters.

Turning in the Assignment

To turn in the assignment execute the following git commands from within your repository:

```
git add <file>
git commit -a
git push origin master
```

where a `git add <file>` command is needed for every file that is required for building the assignment executable. Failure to add any required files will result in a failing grade for the