Homework Assignment 8

Any automatically graded answer may be manually graded by the instructor. Submissions are expected to only use functions taught in the course. If a submission uses a disallowed function, that exercise can get zero points. Excluding promises, *all functions that mutate values are disallowed* (mutable functions usually have a `!` in their name).

Translating SimpleJS into LambdaJS

1. Implement the following translation function from SimpleJS into LambdaJS. LambdaJS variables are underlined, while SimpleJS are not. We use the abstract-syntax notation for the let-binding, sequencing, the object constructor, and the function declaration (λ). We use indentation to highlight the scope of let-binders and of sequencing. You are encouraged to peruse `hw8-util.rkt`, as it gives usage examples and has helpful documentation to complete this assignment.

$$J[\![x.y]\!] \stackrel{\text{def}}{=\!=} (\texttt{get-field} \ (\texttt{deref} \ \texttt{x}) \ \texttt{"y"})$$

$$J[\![x.y := e]\!] \stackrel{\text{def}}{=\!=} \texttt{let} \ \underline{data} = [\![e]\!] \ \texttt{in}$$
$$\texttt{let} \ \underline{o} = ((\texttt{deref} \ \underline{x})) \ \texttt{in}$$
$$(\texttt{set!} \ \underline{x} \ (\texttt{set-field} \ \underline{o} \ \texttt{"y"} \ \underline{data}));$$
$$\underline{data}$$

$$J[\![x.y(e\cdots)]\!]\!] \stackrel{\text{def}}{=\!=} \texttt{let} \ \underline{m} = (\texttt{get-field} \ (\texttt{deref} \ \texttt{x}) \ \texttt{"y"}) \ \texttt{in}$$
$$\texttt{let} \ \underline{f} = (\texttt{get-field} \ (\texttt{deref} \ \texttt{m}) \ \texttt{"\$code"}) \ \texttt{in}$$
$$(\underline{f} \ \underline{x} \ [\![e\cdots]\!])$$

$$J[\![\texttt{function}(x\cdots) \ \{e\}]\!] \stackrel{\text{def}}{=\!=} (\texttt{alloc} \ \{\texttt{"\$code"} : \lambda(\underline{this}, \underline{x}\cdots).[\![e]\!], \texttt{"prototype"} : (\texttt{alloc} \ \{\})\})$$

$$J[\![\texttt{new} \ e_f(e\cdots)]\!] \stackrel{\text{def}}{=\!=} \texttt{let} \ \underline{ctor} = (\texttt{deref} \ [\![e_f]\!]) \ \texttt{in}$$
$$\texttt{let} \ \underline{obj} = (\texttt{alloc} \ \{\texttt{"\$proto"} : (\texttt{get-field} \ \underline{ctor} \ \texttt{"prototype"})\}) \ \texttt{in}$$
$$\texttt{let} \ \underline{f} = (\texttt{get-field} \ \underline{ctor} \ \texttt{"\$code"}) \ \texttt{in}$$
$$(\underline{f} \ \underline{obj} \ J[\![e]\!]\cdots);$$
$$\underline{obj}$$

$$J[\![c]\!] \stackrel{\text{def}}{=\!=} c$$

$$J[\![x]\!] \stackrel{\text{def}}{=\!=} \underline{x}$$

$$J[\![\texttt{let} \ x = e_1 \ \texttt{in} \ e_2]\!] \stackrel{\text{def}}{=\!=} \texttt{let} \ \underline{x} = [\![e_1]\!] \ \texttt{in} \ [\![e_2]\!]$$