Macro Programming

https://clojure.org/reference/reader

Ken Q Pu, Faculty of Science, Ontario Tech University

Strategy in macro programming

- Define the macro's functionality and its expected parameters.
- Think about the code that you want to be generated. Sketch it out on a paper to help to visualize the generated code.
- Figure out how the generated code is related to the macro parameters.
- Utilize the macro programming features to compute the generated code from the macro parameters.

Echo

The ***println*** function prints its argument to the output. However, the parameter is always evaluated first, and the result is printed.

We want to implement ***echo*** which prints the argument without evaluation.

```
(echo (+ 1 2 3))
⇒ (+ 1 2 3)
```

Echo

The ***println*** function prints its argument to the output. However, the parameter is always evaluated first, and the result is printed.

We want to implement ***echo*** which prints the argument without evaluation.

```
(echo (+ 1 2 3))
⇒ (+ 1 2 3)
```

```
(defmacro echo [form]
  ...)
```

Echo

The ***println*** function prints its argument to the output. However, the parameter is always evaluated first, and the result is printed.

We want to implement ***echo*** which prints the argument without evaluation.

```
(echo (+ 1 2 3))
⇒ (+ 1 2 3)
```

```
(defmacro echo [form]
  ...)
```

```
(println "(+ 1 2 3)")
```

Echo

The ***println*** function prints its argument to the output. However, the parameter is always evaluated first, and the result is printed.

We want to implement ***echo*** which prints the argument without evaluation.

```
(echo (+ 1 2 3))
⇒ (+ 1 2 3)
```

```
(defmacro echo [form]
  ...)
```

```
(str form)
```

```
(println "(+ 1 2 3)")
```

Echo

The ***println*** function prints its argument to the output. However, the parameter is always evaluated first, and the result is printed.

We want to implement ***echo*** which prints the argument without evaluation.

```
(echo (+ 1 2 3))
⇒ (+ 1 2 3)
```

```
(defmacro echo [form]
  `(println ~(str form)))
```

```
(str form)
```

```
(println "(+ 1 2 3)")
```

Echo

Why not these other erroneous alternatives?

```
(defmacro echo [form]
  `(println ~(str form)))
```

```
(defmacro echo [form]
  `(println (str form)))
```

```
(defmacro echo [form]
  (println (str form)))
```

```
(defmacro echo [form]
  `(println (str ~form)))
```

Echo-eval

We want to extend ***echo*** *to also evaluate the input form and print the output.*

```
(echo-run (+ 1 2 3))
⇒ (+ 1 2 3)
  6
```

```
(defmacro echo-eval [form]
  `(do (println ~(str form))
       ...))
```

Echo-eval

We want to extend ***echo*** *to also evaluate the input form and print the output.*

```
(echo-eval (+ 1 2 3))
⇒ (+ 1 2 3)
  6
```

```
(defmacro echo-eval [form]
  ...))
```

Echo-eval

Let's think about the generated code.

Then figure out the strategy using macro programming features.

```
(defmacro echo-eval [form]
  ...))
```

```
(do
  (println "(+ 1 2 3)")
  (let [result (+ 1 2 3)]
    (println result)))
```

```
(str form)
form
```

Echo-eval

Let's think about the generated code.

Then figure out the strategy using macro programming features.

```
(defmacro echo-eval [form]
  `(do
     (println ~(str form))
     (let [result# ~form]
       (println result#)))
```

```
(do
  (println "(+ 1 2 3)")
  (let [result (+ 1 2 3)]
    (println result)))
```

```
(str form)
form
```

Echo-eval

What's wrong with this code?

```
(defmacro echo-eval [form]
  `(do
     (println ~(str form))
     (let [result ~form]
       (println result)))
```

Challenge

Can you implement a macro that can accept multiple forms?

```
(echo-eval
  (+ 1 2 3)
  (str 1 2 3))

=>

(+ 1 2 3)
6

(str 1 2 3)
"123"
```

Summary

- Macros are just Clojure functions, except that they must generate data that is valid Clojure code.
- Macros execute during compile time, so they do not have access to any runtime data.
- Macro programming features of Clojure make writing complex macros more manageable.
- Macros are difficult to develop and maintain. Consider using runtime functions instead.