For next class, install LaTeX (pronounced "LAH-tech")

CS152 – Advanced Programming
Language Principles
Prof. Tom Austin, Fall 2014

Macros

What is a macro?

- Short for *macroinstruction.*
- A rule or pattern that specifies how a certain input sequence should be mapped to a replacement sequence.

Types of macros

Text substitution

Procedural macros

Syntactic macros

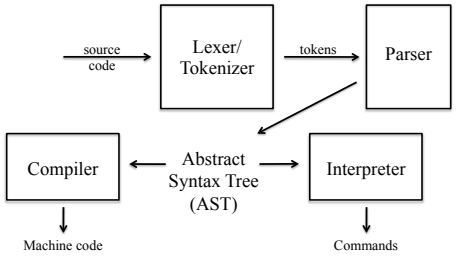
Text Substitution Macros

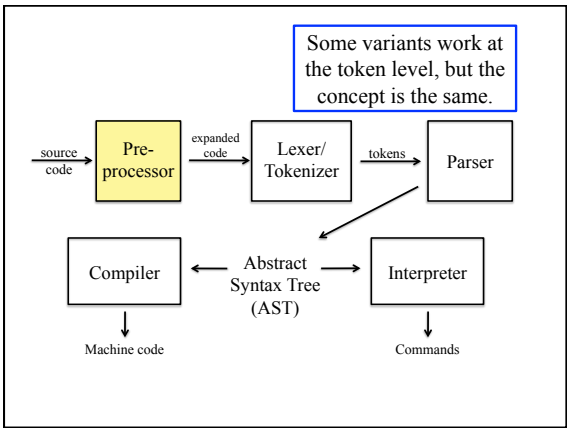
This type of macro works by *expanding text.*

Fast, but limited power. Some examples:

- C preprocessor
- Embedded languages (PHP, Ruby's erb, etc.) are similar, but more powerful

A Review of Compilers





C preprocessor example

```
#define PI 3.14159

#define SWAP(a,b) {int tmp=a;a=b;b=tmp;}

int main(void) {
   int x=4, y=5, diam=7, circum=diam*PI;
   SWAP(x,y);
}
```

```
int main(void) {
   int x=4, y=5, diam=7, circum=diam*PI;
   SWAP(x,y);
}
```


```
int main(void) {
  int x=4, y=5, diam=7,
      circum=diam*3.14159;
  {int tmp=x;x=y;y=tmp;};
}
```

Procedural macros

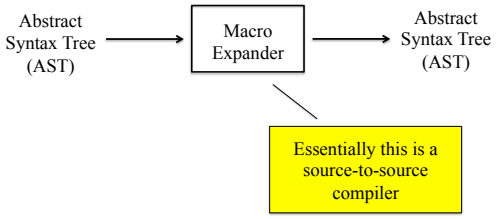
Procedural macros execute preprocessor statements at compilation time.

- Allows macros to be written in a procedural language.
- More powerful than text substitution…
- …but slower and more complex compiler.

Syntactic macros

- Work at the level of abstract syntax trees
- From the Lisp family (including Scheme)
  - Why Lisp? Because Lisp programs *are* ASTs
- Powerful, but expensive
- Where our attention will be focused

Macro expansion process



Many macro systems suffer from a problem with *inadvertent variable capture.*

Let's look at an example…

Accidental Capture Example
(in class)

Hygiene

Hygienic macros are macros whose expansion is guaranteed not to cause the *accidental capture of identifiers*.

Macros in Scheme

- Scheme is noted for its powerful (and hygienic) macro system.
- Is it needed? Aren't lambdas enough?

```
(define (swap x y)
  (let ([tmp x])
    (set! x y)
    (set! y tmp)))
(let ([tmp 7][b 3])
  (swap tmp b)
  (cons tmp b))
```

What is the result?

Pattern Based Macros

- Preserves hygiene
- `define-syntax-rule` matches the given pattern and transforms is into the specified template
- `define-syntax` is allows you to specify multiple patters, including those with a variable number of arguments (using the … syntax)

```
(define-syntax-rule
 (swap x y)
  (let ([tmp x])
    (set! x y)
    (set! y tmp)))

(let ([tmp 7][b 3])
  (swap tmp b)
  (cons tmp b))
```

What is the result now?

Define-syntax swap function

```
(define-syntax swap
  (syntax-rules ()
      [(swap x y)
         (let ([tmp x])
           (set! x y)
           (set! y tmp))]))
```

Lab

Using define-syntax, create a switch statement.
Sample usage:

```
(define x 99)
(switch x
    [3 (displayln "x is 3")]
    [4 (displayln "x is 4")]
    [5 (displayln "x is 5")]
    ['default (displayln
              "none of the above")])
```

For more reading on macros:

- Matthew Flatt, "Pattern-based macros", section 16.1 of "The Racket Guide". http://docs.racket-lang.org/guide/pattern-macros.html
- Greg Hendershott, "Fear of Macros". http://www.greghendershott.com/fear-of-macros/index.html