C

```
char *s="char *s=%c%s%c;main(){printf(s,34,s,34,10);}%c";main(){printf(s,34,s,34,10);}
```

```
#define p(x) main () { puts ("#define p(x) "# x""); puts ("p("# x")"); }
p(main () { puts ("#define p(x) "# x""); puts ("p("# x")"); })
```

Java

```
public class PrintSelf {
   static StringBuffer self = new StringBuffer(
     "public class PrintSelf {
         static StringBuffer self = new StringBuffer();
         static char quote = 34;
         public static void main(String[] args) {
            self.insert(68,self);
            self.insert(68,quote);
            self.insert(295,quote);
            System.out.println(self);
         }
      }");
   static char quote = 34;
   public static void main(String[] args) {
      self.insert(68,self);
      self.insert(68,quote);
      self.insert(295,quote);
      System.out.println(self);
   }
}
```

Unix shell

```
cat $0
```

```
Unmatched '.
```

```
x='y=`echo .|tr . "\47"`;echo "x=$y$x$y;$x"';y=`echo .|tr . "\47"`;echo "x=$y$x$y;$x"
```

SML

```
let val s = "let val s = #$#
  in String.translate (fn x => if Char.ord x=36 then s
    else implode [if Char.ord x=35 then Char.chr 34 else x])s end"
in String.translate (fn x => if Char.ord x=36 then s
  else implode [if Char.ord x=35 then Char.chr 34 else x])s end
```

Scheme

```
(define (printself <method>)
  (bind (((x <list>)
          (quote (method ()
                   (list (quote define) (quote (printself <method>))
                         (list (quote bind)
                               (list
                               (list (quote ( x <list>))
                                     (list (quote quote) x))) x))))))
    (method ()
      (list (quote define) (quote (printself <method>))
            (list (quote bind)
                  (list
                  (list (quote (x <list>))
                        (list (quote quote) x))) x)))))
```

Lisp

```
((lambda (x) (format t "(~s’~s)" x x))’(lambda (x) (format t "(~s’~s)" x x)))
```

```
(setq p "(setq p ~S)(format T p p)")(format T p p)
```

```
(let ((let ’‘(let ((let ’,let)) ,let)))
    ‘(let ((let ’,let)) ,let))
```