Overview

- Some more LISP stuff: user input, trace, cons, more setf, etc.
- Symbolic Differentiation:
  [q] does it need intelligence?
- Expression Simplification
- Programming Assignment (due 2/15/02, Friday).

READ: User Input

READ: keyboard input from user

```
>(read)
hello
HELLO

>(if (equal (read) 'hello)
        'good
        'bad
)
hello
GOOD
```


TRACE/UNTRACE: call tracing

```
>(trace fibo)
(FIBO)
>(fibo 4)
  1> (FIBO 4)
    2> (FIBO 3)
      3> (FIBO 2)
      <3 (FIBO 2)
      3> (FIBO 1)
      <3 (FIBO 1)
    <2 (FIBO 3)
    2> (FIBO 2)
    <2 (FIBO 2)
  <1 (FIBO 5)
5
>
```


List stuff

- CONS: append an atom and a list
  `(cons 'a '(1 2 3)) -> (A 1 2 3)`
  `(cons '(a) '(1 2 3)) -> ((A) 1 2 3)`
- APPEND: append two lists
  `(append '(1 2) '(4 5)) -> (1 2 4 5)`

Fun with SETF

Replace list element with SETF. Note: SETQ will not work!

```
>(setf b '(1 (2 3) 4))
(1 (2 3) 4)

>(caadr b)
2

>(setf (caadr b) 'abcdefg)
ABCDEFG

>b
(1 (ABCDEFG 3) 4)
```


Symbolic Differentiation

Basics: given variable x, functions $f(x), g(x)$, and constant (i.e. number) a:

1. $$\frac{da}{dx} = 0, \frac{d(a \times x)}{dx} = a$$
2. $$\frac{d(f+g)}{dx} = \frac{df}{dx} + \frac{dg}{dx}$$
3. $$\frac{d(f \times g)}{dx} = \frac{df}{dx} \times g + f \times \frac{dg}{dx}$$

The operators can be extended to: binary minus (e.g. `(- x 1)`), unary minus (e.g. `(- x)`), division (e.g. `(/ x 10)`, etc.

Describing in LISP (I)

```
(deriv <expression> <variable>)
```

1. $$\frac{da}{dx} = 0, \frac{d(a \times x)}{dx} = a$$

```
(deriv '10 'x) -> 0
(deriv '(* 10 x) 'x) -> 10
```


Describing in LISP (II)

```
(deriv <expression> <variable>)
```

1. $$\frac{d(f+g)}{dx} = \frac{df}{dx} + \frac{dg}{dx}$$

```
(deriv '(+ (* x 10) (+ 25 x)) 'x)
== (list
       '+
       (deriv '(* x 10) 'x)
       (deriv '(+ 25 x))
   )
```

Describing in LISP (III)

```
(deriv <expression> <variable>)
```

1.

$$\frac{d(f \times g)}{dx} = \frac{df}{dx} \times g + f \times \frac{dg}{dx}$$

```
(deriv '(* (+ 14 x) (* x 17)) 'x)
==(list
     '+
     (list '* (deriv '(* 14 x) 'x) '(* x 17))
     (list '* '(+ 14 x) (deriv '(* x 17)))
  )
```


DERIV: the core function

Pseudo code (basically a recursion):

```
(defun deriv (expression var) BODY)
```

1. if `expression` is the same as `var` return 1
2. if `expression` is a number return 0
3. if `(first expression)` is `'+`, return

```
'(+ (deriv (second expression) var)
    (deriv (third expression) var)
```

4. and so on.

Main Function: DERIV

You can make separate functions for each operator:

```
(defun deriv (expr var)
  (if (atom expr)
    (if (equal expr var) 1 0)
    (cond
        ((eq '+ (first expr))     ; PLUS
            (derivplus expr var))
        ((eq '* (first expr))     ; MULT
            (derivmult expr var))
        (t    ; Invalid
            (error "Invalid Expression!"))
     )
  )
)
```


Calling DERIV from DERIVPLUS

Then, you can call `deriv` from `derivplus`, etc.

```
(defun derivplus (expr var)
    (list '+
          (deriv (second expr) var)
          (deriv (third expr) var)
    )
)
```

Expression Simplification

Problem: a lot of nested expression containing

`(* 1 x) (* x 1) (+ 0 x) (+ x 0) (+ 3 4) ...`

which are just `x, x, x, x,` and 7.

Use simplification rules:

1. `(+ <number> <number>)`: return the evaluated value
2. `(* <number> <number>)`: return the evaluated value
3. `(+ 0 <expr>) (+ <expr> 0)`: return `<expr>`
4. `(* 1 <expr>) (* <expr> 1)`: return `<expr>`
5. `(- (- <expr>))` : return `<expr>`

HINT: look at the raw result and see what can be reduced.

SPLUS: Simplify `(+ x y)`

```
(defun splus (x y)
    (if (numberp x)
        (if (numberp y)
            (+ x y)
            (if (zerop x)
                y
                (list '+ x y)
            )
        )
        (if (and (numberp y) (zerop y))
            x
            (list '+ x y)
        )
    )
)
```


Programming Assignment 1

1. Implement `deriv` to support:
   addition, subtraction, unary minus, multiplication, and division.
   → HINT: use slide 11 as a skeleton.
2. Implement simplification routines `splus` etc. for all operators and integrate it into `derivplus`, etc.
   → HINT: Integrate code in slide 14 into code in slide 12.
3. Implement a function
   `(deriv-val <expr> <var> <value>)`
   to evaluate the final expression where the number `<value>` replaces the symbol `<var>`.
   → HINT: Use the `eval` function to recursively evaluate.
4. You *may* (i.e. not required) write a separate `(simplify <expr>)` function using `splus`, etc.

Programming Assignment 1: other conditions

All operators are either binary or unary:
i.e. expressions like `(+ 1 2 3 4 5)` do not need to be supported. Only those in the form of `(+ 1 2)` or `(- 5)` are expected to be used.

Programming Assignment 1: Example Inputs and Outputs

1. `(deriv ’(* (+ x 4) (+ x 5)) ’x)`
   `-> (+ (+ X 4) (+ X 5)))`
2. `(deriv ’(/ (+ x 1) x) ’x)`
   `-> (/ (- X (+ X 1)) (* X X))`
3. `(deriv-val ’(* (+ x 4) (+ x 5)) ’x 10)`
   `-> 29`
4. `(deriv-val ’(/ (+ x 1) x) ’x 20)`
   `-> -1/400`

Programming Assignment 1: Required Material

Use the exact filename as shown below (in **bold**).

- Program code (**deriv.lsp**): put it in a single text file.
  – Ample indentation and documentation is required.
- Documentation (**README**): user manual
- Sample inputs and outputs (include in **README**)
  – 10 non-trivial (4 or more terms) examples should be given.
- 10% Extra Credit for the top 3 submissions that can produce the shortest expressions (average of about 10 expressions will be used as a measure). Only on-time submissions will be considered for extra credit. If there are ties, the closest number of students to 3, but not less than 3, will be awarded with extra credit.

Programming Assignment 1: Submission

- Send as email to the TA (attached text files):
  `ltapia@tamu.edu`,
  and also CC: `choe@tamu.edu`
- Submission deadline is 2/15/02 Friday midnight (23:59:59).
- Late policy: initial penalty -25%, and additional -25% per week thereafter (i.e. 2/16–2/22: -25%, 2/23–3/1: -50%, ...).
- If more than half have problems meeting the deadline, I will consider an extension.
- Only send plain text ASCII files. **Do not send MS-Word documents or other formatted text.**

Next Week: Search Methods

- Chapter 3
- Required: sections 3.3–3.7.
- Other sections are optional.