CSC 171
Introduction to Computer Programming
Lab 1

Dr. Stephen Bloch
sbloch@adelphi.edu
http://www.adelphi.edu/sbloch/class/171/

Review: simpler, more consistent rules for arithmetic

1) All operators go *before* however many operands they need.

2) All subexpressions *must* have parentheses around them (including the operator).

3) = means “are these equal?”;
define means “make this stand for that”.
Both count as operators (see rules 1 & 2)

4) No hidden operators; if you mean *, say it.

5) No extra parentheses allowed; *exactly one* pair of parentheses per operator.

Review: Old vs. new notation

- 3 + 4
- 3 + 4 * 5
- (3+4) * 5
- 2 + cos(0)
- 3 + 4 = 7
- 3 + 4 > 7
- Let x = 7
- 6x + ((2))
- Let f(x) = cos(x) + 2
- f(0)

- (+ 3 4)
- (+ 3 (* 4 5))
- (* (+ 3 4) 5)
- (+ 2 (cos 0))
- (= (+ 3 4) 7)
- (> (+ 3 4) 7)
- (define x 7)
- (+ (* 6 x) 2)
- (define (f x) (+ (cos x) 2))
- (f 0)

Conversion recipe

1) Expand abbreviations

- 3z becomes 3 * z
- z^2 becomes z * z
- 3*3+4*5 becomes ((3*3)+(4*5))
- $(-b+\sqrt{(b^2 - 4ac)})/2a$ becomes (((-b)+sqrt((b*b)-(4*a*c)))/(2*a))

2) Parenthesize *all* subexpressions that have an operator. Should end up w/same number of operators, left-parentheses, and right-parentheses.

3) Move each operator to just after its own left parenthesis.

becomes `(+ 3 x)`

becomes `(+ 3 (* 4 5))`

becomes `(- (* 7 x) (/ (+ 3 x) (+ y 2)))`

Using DrScheme: interactions

```
Welcome to DrScheme, version 301.
Language: Beginning Student.
> (+ 3 (* 4 5))
23
>
```

Using DrScheme: Definitions

```
(+ 3 4)
(+ 3 (* 4 5))
(* (+ 3 4) 5)
(+ 2 (cos 0))
(= (+ 3 4) 7)
(> (+ 3 4) 7)
(define x 7)
(+ (* 6 x) 2)
(define (f x) (+ (cos x) 2))
(f 0)
```

```
Welcome to DrScheme, version 301.
Language: Beginning Student.
7
23
35
3
true
false
44
3
>
```

Using DrScheme: Stepper

Common errors

- Mismatched parentheses
- Omitting space after operator or between numbers
- Putting operator between operands

Checking your work

```
(+ 3 4) "should be 7"
(+ 3 (* 4 5)) "should be 23"
(* (+ 3 4) 5) "should be 35"
(= (+ 3 4) 7) "should be true"
(> (+ 3 4) 7) "should be false"
(* 3 5) "should be 8"
```

```
Welcome to DrScheme, version 301.
Language: Beginning Student.
7
"should be 7"
23
"should be 23"
35
"should be 35"
true
"should be true"
false
"should be false"
15
"should be 8"
>
```

Exercise

- Make up some arithmetic expressions.
- On paper, write down their correct values.
- Convert them to Scheme syntax.
- Type them, with their correct values, into the Definitions window.
- Hit “Run”; did they all work correctly?

Exercise: find the cubes of…

a) 0

b) 5

c) 17

d) 1234567890

e) The result of (d)

One way

a) (* 0 0 0) “should be 0”

b) (* 5 5 5) “should be 125”

c) (* 17 17 17) “should be a few thousand”

d) (* 1234567890 1234567890 1234567890) “should be about thirty digits long”

e) (* yecch….)
“should be about ninety digits long”

Better: define variables

(define big 1234567890)

(* big big big) “should be about 30 digits”

(define bigger (* big big big))

(* bigger bigger bigger)
“should be about 90 digits”

Even better: define function

```
(define (cube num) (* num num num))

(cube 1234567890)
    "should be about 30 digits"

(cube (cube 1234567890))
    "should be about 90 digits"

…
```

But are we sure cube is right?

- “cube” function takes in a number and returns another number
- Must produce right answer for *every possible* input
- Problem: infinitely many possible inputs; can't test them all!
- Choose test cases carefully:
  - Cover all “special cases”
  - Do the easy ones first, building up to harder ones
  - Number of test cases increases with complexity of program

Writing “cube” methodically

; cube : takes in a number and returns a number

(cube 0) “should be” 0

(cube 5) “should be” 125

(cube -6) “should be” -216

Writing “cube” methodically

```
; cube : takes in a number and returns a number
(define (cube num)
   … something involving num … )
```

(cube 0) “should be” 0

(cube 5) “should be” 125

(cube -6) “should be” -216

Writing “cube” methodically

```
; cube : takes in a number and returns a number
(define (cube num)
  (* num num num))
```

(cube 0) “should be” 0

(cube 5) “should be” 125

(cube -6) “should be” -216

Recipe for defining functions

1) **Contract**
specify function name, number/types of arguments,
type returned

2) **Examples**
write down examples with known right answers

3) **Function skeleton**
inserted between contract and examples

4) **Function body**
replaces function skeleton

5) **Testing**
Do the examples produce the expected answers?

Why do we need a recipe?

Programming requires creativity, and sometimes that's enough.

But when programs get big and complex, it helps to have a structure to follow.

Analogous to rules of counterpoint, harmony, etc. in music. Sonata, rondo, virelai, fugue, …

Helps avoid "blank page syndrome".

Exercise

- Define a function “hours->minutes” which takes in a number of hours & returns how many minutes it is.
- Follow the design recipe.

Functions with Multiple Arguments

- Write a function named “difference-of-cubes” which takes two numbers and computes the difference of their cubes.
- Follow the design recipe.

Functions with Multiple Arguments

; difference-of-cubes : number number -> number

“Examples of difference-of-cubes:”
(difference-of-cubes 0 0) “should be 0”
(difference-of-cubes 2 0) “should be 8”
(difference-of-cubes 3 2) “should be 19”

Functions with Multiple Arguments

```
; difference-of-cubes : number number -> number
(define (difference-of-cubes x y)
  … something involving x and y …)
"Examples of difference-of-cubes:"
(difference-of-cubes 0 0) "should be 0"
(difference-of-cubes 2 0) "should be 8"
(difference-of-cubes 3 2) "should be 19"
```

Functions with Multiple Arguments

```
; difference-of-cubes : number number -> number
(define (difference-of-cubes x y)
  (- (* x x x) (* y y y)))
"Examples of difference-of-cubes:"
(difference-of-cubes 0 0) "should be 0"
(difference-of-cubes 2 0) "should be 8"
(difference-of-cubes 3 2) "should be 19"
```

Functions with Multiple Arguments

```
; difference-of-cubes : number number -> number
(define (difference-of-cubes x y)
  (- (cube x) (cube y)))
"Examples of difference-of-cubes:"
(difference-of-cubes 0 0) "should be 0"
(difference-of-cubes 2 0) "should be 8"
(difference-of-cubes 3 2) "should be 19"
```

Exercise

- Define a function “total-minutes” which takes in a number of hours and a number of minutes and returns the total number of minutes.
- Ex. (total-minutes 1 30) “should be 90”
- Follow the design recipe.

Pre-semester survey

www.adelphi.edu/sbloch/class/171/survey.html

Daily survey

www.adelphi.edu/sbloch/class/171/daily.html