Ceng 111 – Fall 2021
Week 9b

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.

Expressions

- An expression is a calculation which has a set of *operations*.
- Operations have *operators* and *operands*.
- Example: 3 + 4
  - + → operator
  - 3, 4 → operands

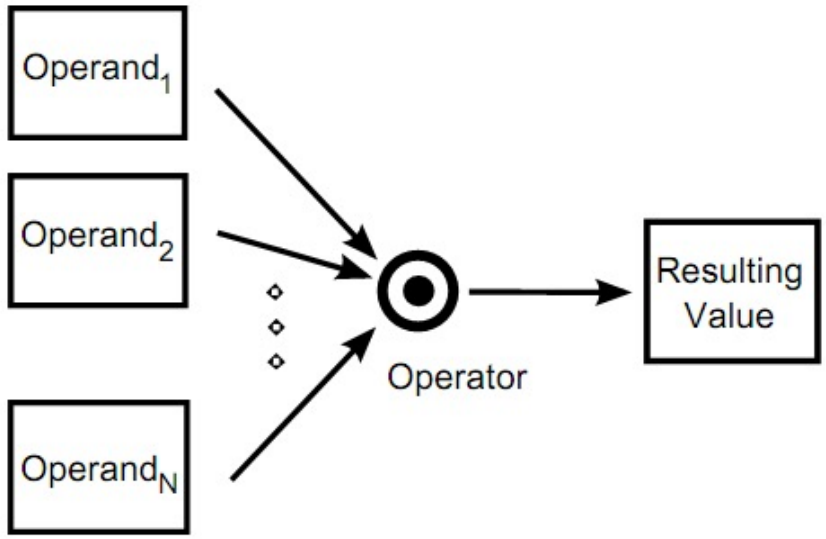
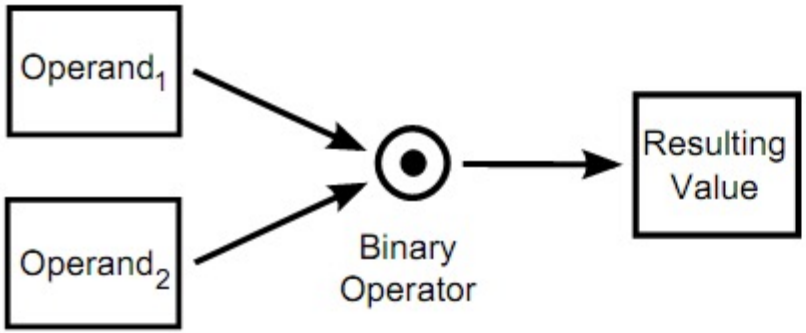
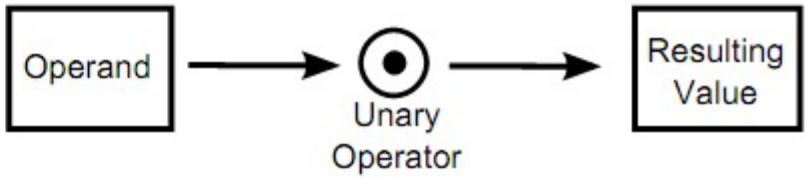


Figure 3.1: N-ary operation



(a) Binary operation



(b) Unary operation

Previously on CENG111!

Expressions in Python
- Involving logical operators -

- Precedence & Associativity

Operator	Type	Associativity	Description
[]	Binary	Left-to-right	**Indexing**
**	Binary	Right-to-left	Exponentiation
+, -	Unary	Right-to-left	Positive, negative
*, /, //, %	Binary	Left-to-right	Multiplication & Repetition, Division, Remainder, Modulo
+, -	Binary	Left-to-right	Addition, Subtraction, Concatenation
in, not in, <, <= >, >=, ==, !=	Binary	Right-to-left	Membership, Comparison
not	Unary	Right-to-left	Logical negation
and	Binary	Left-to-right	Logical AND
or	Binary	Left-to-right	Logical OR

Expressions in Python
- assignment (**not** an operator) -

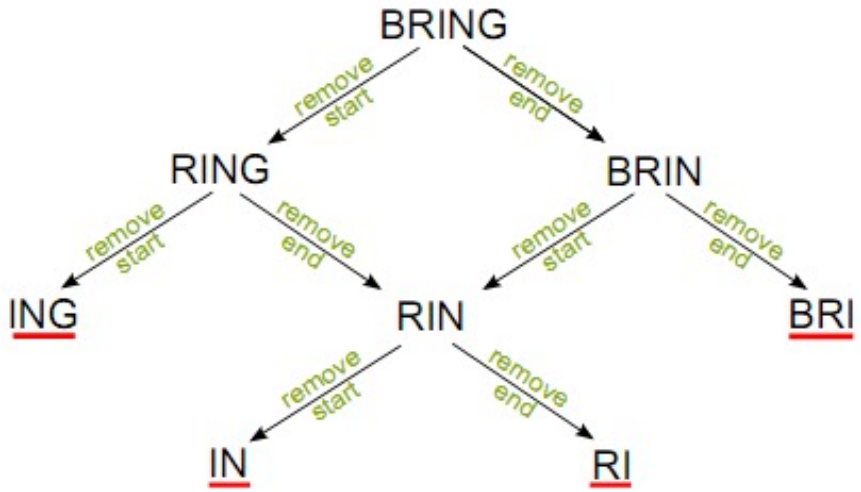
- Single assignment:
  - a = 4
- Multiple assignment:
  - a = b = c = 4
- Combined assignment:
  - a = a + 4 → a += 4
  - +=, *=, -=, /=, etc.

```
>>> b += 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
>>> b = 5
>>> b **= 2
>>> b
25
```

Previously on CENG111!

Church-Rosser Property

- A reduction/re-writing system has the Church-Rosser Property if the set of rules always lead to the same results independent of the order of application of the rules.
- Evaluation of a mathematical expression is said to have the Church-Rosser Property:
- A simple example:
  - "If both ends of a string are consonants, remove one"



Previously on CENG111!

Church-Rosser Property

- How about expressions in programming languages? Do they have Church-Rosser Property?
- Answer it yourself considering these:
  - Limitations due to fixed size representations of numbers: Remember that a+(b+c) may not be equivalent to (a+b)+c?
  - Side-effects in evaluating some operations and function calls
    - f(2) + x

LESSON: A programmer has to know the order an expression is evaluated!

Side effect

```
1 def f(L):
2       L[0] += 2
3       return L[0]
4
5 M=[2, 3, 4]
6 x = f(M) + M[0]
```

Evaluation order yields two different results

Evaluate f(M) first

Evaluate M[0] first

Today

- Expression evaluation

Administrative Notes

- THE2 announced:
  - Due date: 26 December, 23:59
- Midterm:
  - 22 December, Wednesday, 18:00

So, how are expressions evaluated in HLPL?

- Consider these:
  - `2 - 3 ** 4 / 8 + 2 * 4 ** 5 * 1 ** 8`
  - `4 + 2 - 10 / 2 * 4 ** 2`
  - `3 / 3 ** 3 * 3`
- or these:

a) `not a == b + d < not a`

b) `a == b <= c == True`

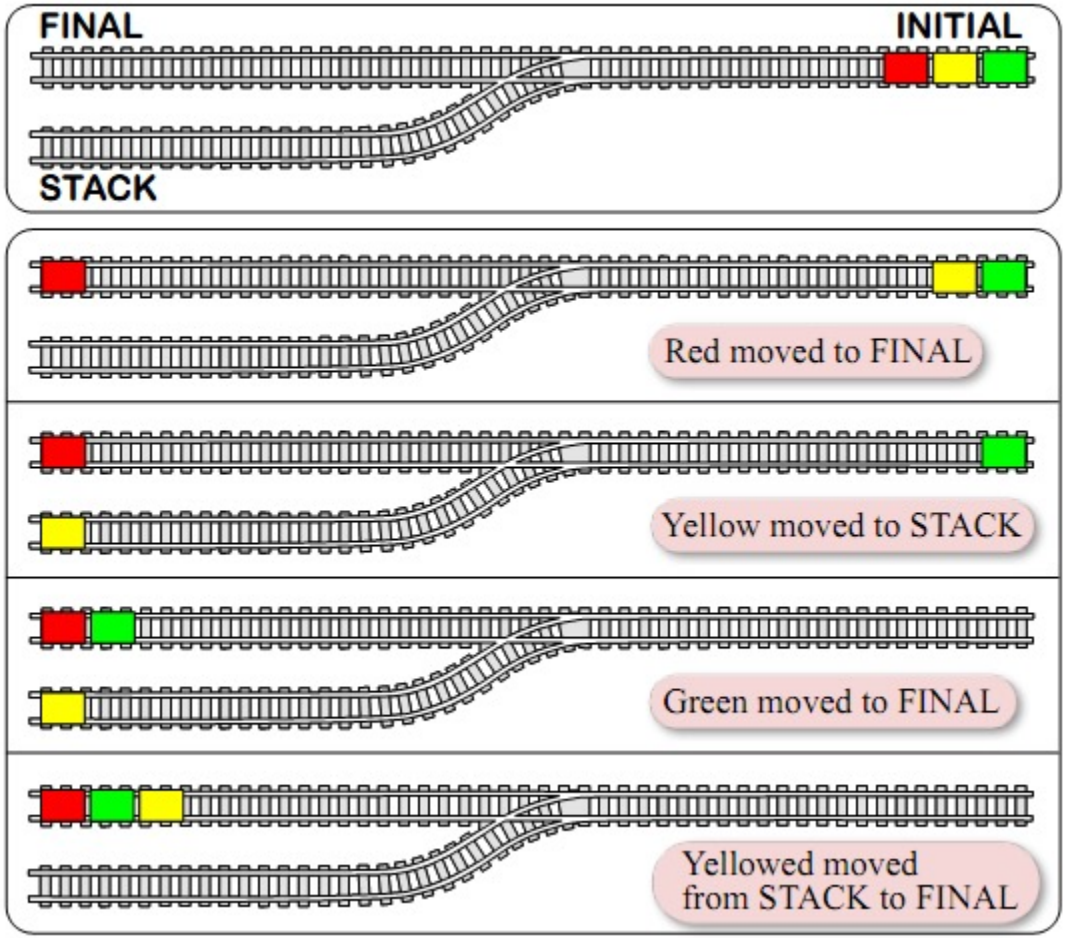
c) `True <= False == b + c`

d) `c / a / b`

Expression Evaluation in PLs

- In most cases,
  - First, *Dijkstra's shunting-yard algorithm* is used to convert an expression into postfix notation.
  - Then, the postfix expression is evaluated using a *postfix evaluation algorithm*.

Dijktsra's Shunting-Yard Algorithm

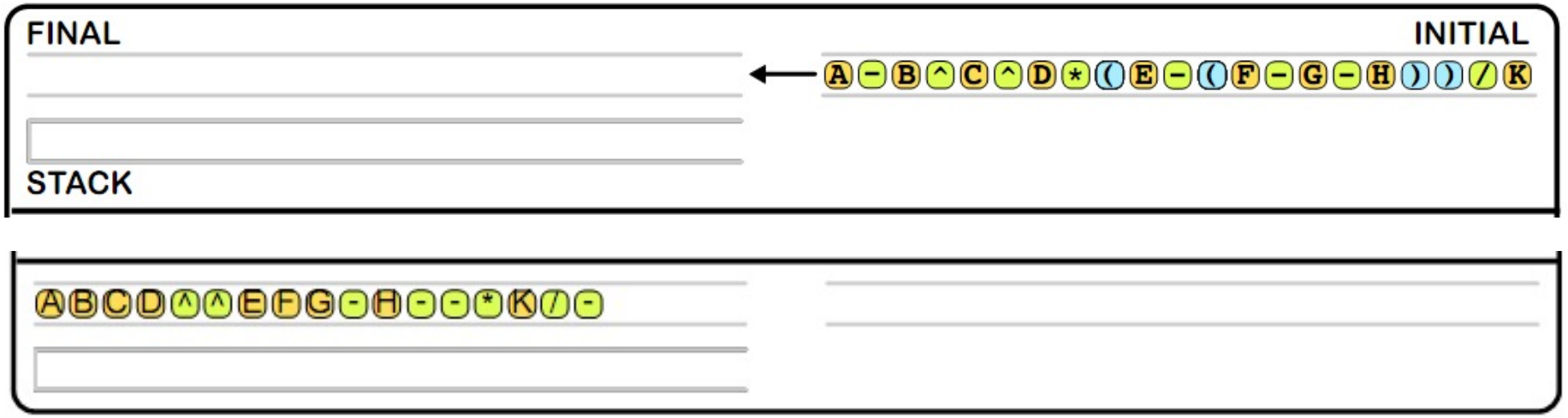


Algorithm 1 Dijkstra's Shunting-yard algorithm.

Get next token t from the input queue
if t is an operand **then**
  Add t to the output queue
if t is an operator **then**
  while There is an operator τ at the top of the stack, and either t is left-associative and its precedence is less than or equal to the precedence of τ, or t is right-associative and its precedence is less than the precedence of τ **do**
    Pop τ from the stack, to the output queue.
  Push t on the stack.
if t is a left parenthesis **then**
  Push t on the stack.
if t is a right parenthesis **then**
  Pop the operators from the stack, to the output queue until the top of the stack is a left parenthesis.
  Pop the left parenthesis.
if No more tokens to get **then**
  Pop the operators on the stack, if any, to the output queue.

Dijktsra's Shunting-Yard Algorithm: Example

$$A - \frac{B^{C^D} \times (E - (F - G - H))}{K}$$



Postfix Evaluation

1. Go from left to right
2. When you see an operator:
   a) Apply it to the last two operands
   b) Remove the last two operands and put the result in place of the operator.

Output in Python

```
>>> print "I am %f tall, %d years old and have %s eyes" % (1.86, 20, "brown")
I am 1.860000 tall, 20 years old and have brown eyes
```

- %f → Data identifier
- We have the following identifiers in Python:

Identifier	Description
d, i	Integer
f, F	Floating point
e, E	Floating point in exponent form
s	Using the `str()` function
r	Using the `repr()` function
%	The % character itself

Output in Python

```
>>> print "I am {0} tall, {1} years old and have {2} eyes".format(1.86, 20, "brown")
I am 1.86 tall, 20 years old and have brown eyes
```

- {0}, {1}, {2} → Data fields
- Instead of numbers, we can give names to the fields:

```
>>> print "I am {height} tall, {age} years old and have {color} eyes".\
        format(height=1.86, age=20, color="brown")
I am 1.86 tall, 20 years old and have brown eyes
```

- We can re-use the fields

```
>>> print "I am {height} tall, {age} years old. I am {height} tall.".\
         format(age=20,height=1.86)
I am 1.86 tall, 20 years old. I am 1.86 tall.
```

Basic Statements

Examples:

del L[2]

print “this is a string”

Compound Statements

- Involves more than one expression or statement
- Example:

$$\textbf{if } \beta \textbf{ then } \sigma$$

$$\textbf{if } \beta \textbf{ then } \sigma_1 \textbf{ else } \sigma_2$$

$$\textbf{while } \beta \textbf{ do } \sigma$$

$$\textbf{for } v = 1 \textbf{ to } 5 \textbf{ do print } v, v * (v - 1)$$

Conditional Statements

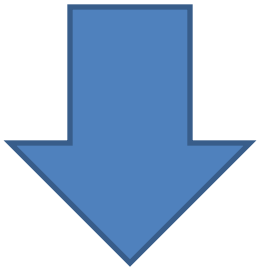
if $\langle boolean\ \ expression \rangle$ **then** $\langle action \rangle$

Translated to:

compute the $\langle boolean\ \ expression \rangle$, leave the result in the relevant register r
branch to α if $r \stackrel{?}{=} 0$
carry out $\langle action \rangle$
α : $\langle some\ actions\ that\ follow\ the\ \mathbf{if} \rangle$

Conditional Statements

if $\langle boolean\ \ expression \rangle$ **then** $\langle action_{TRUE} \rangle$
if $\neg\langle boolean\ \ expression \rangle$ **then** $\langle action_{FALSE} \rangle$



if $\langle boolean\ \ expression \rangle$ **then** $\langle action_{TRUE} \rangle$
else $\langle action_{FALSE} \rangle$

Conditional Statements in Python

```
1 if <condition-expression> :
2         <statements-1>
3 else:
4         <statements-2>
```

- the syntax is important!
- indentation is extremely important!
- “else”-part can be omitted!

You can indent your Python code using tabs or space. However, it is a good programming practice to use only one of them while indenting your code: *i.e.*, do not mix them!

Multiple If Statements in Python

```
1  if <condition-expression-1> :
2          <statements>
3  elif <expression-2> :
4          <statements>
5  .
6  .
7  .
8  elif <expression-M> :
9          <statements>
10 else:
11         <statements>
```

Multiple **Nested** If Statements in Python

```
1 if <condition-expression-1> :
2         <statements-1>
3         if <condition-expression-2>:
4                 <statements-2>
5         else:
6                 <statements-3>
7 else:
8         <statements-4>
```

Conditional Expression in Python

<exp-1> if <cond-exp> else <exp-2>

Note that this is an expression not a statement!!

Functions: Reusable Actions

- In programming, we often combine the statements that we use frequently together into functions.

```
1   void main()
2   {
3       hello();
4
5       ... // Some execution here
6
7       hello();
8   }
9
10  void hello()
11  {
12      ...
13      // I am looong function involving lots and lots of statements
14      ...
15  }
```

Functions: Reusable Actions (cont'd)

- Functions in programming are similar to functions in Mathematics but there are differences.
- Difference to mathematical functions:
  - A function in programming may not return a value.
  - A function in mathematics only depends on its arguments unlike the functions in programming.
  - A mathematical function does not have the problem of side effects.

Functions: Reusable Actions

- Why do we need functions?
  - Reusability
  - Structure
  - Other benefits of the functional paradigm

Functions in Python

```
1 def function-name(parameter-1, ..., parameter-N):
2         statement-1
3         .
4         .
5         statement-M
```

- Syntax is important!
- Indentation is extremely important

Functions in Python

- Write a Python function that reverses a given number
  - Example: If 123 is given, the output should be 321

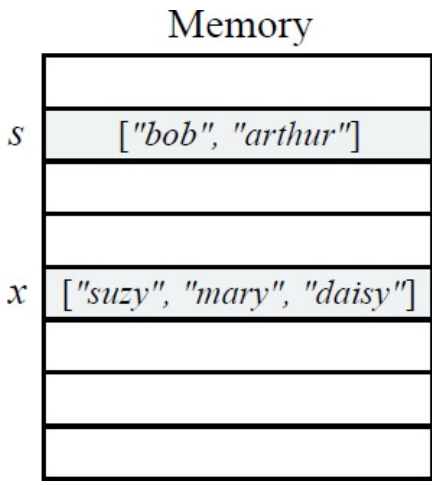
```
1  def reverse(a):
2      return int(str(a)[::-1])
3
```

Parameter passing in functions

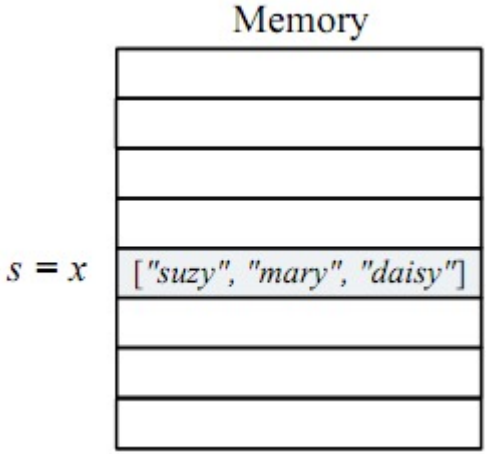
```
define f(x)
    x[0] ← "jennie"
    x ← ["suzy","mary","daisy"]
    return x
...
s ← ["bob","arthur"]
print f(s)
print s
...
```

(a) `["suzy","mary","daisy"]`
`["bob","arthur"]`

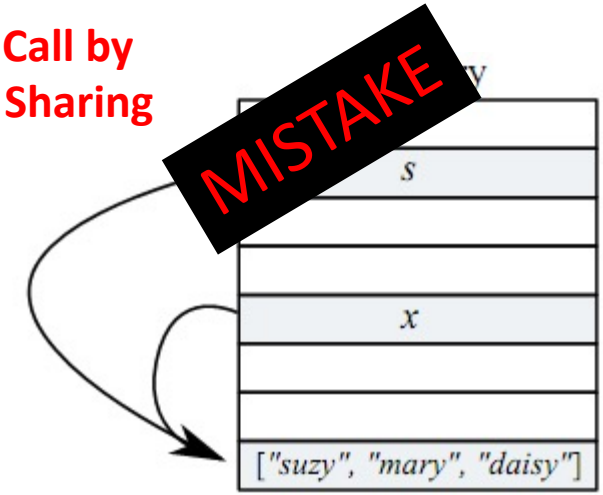
Call by Value



Call by Reference



Call by Sharing



(b) `["suzy","mary","daisy"]`
`["suzy","mary","daisy"]`

(c) `["suzy","mary","daisy"]`
`["jennie","arthur"]`