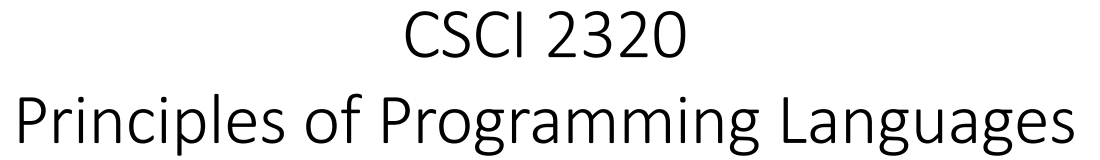


CSCI 2320
Principles of Programming Languages

Semantics

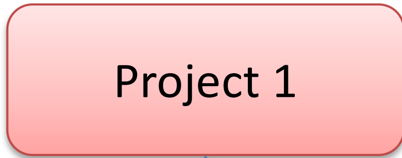
Reading: Ch 7 (Tucker & Noonan)

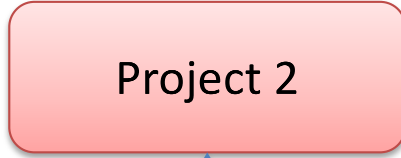
Mohammad T. Irfan

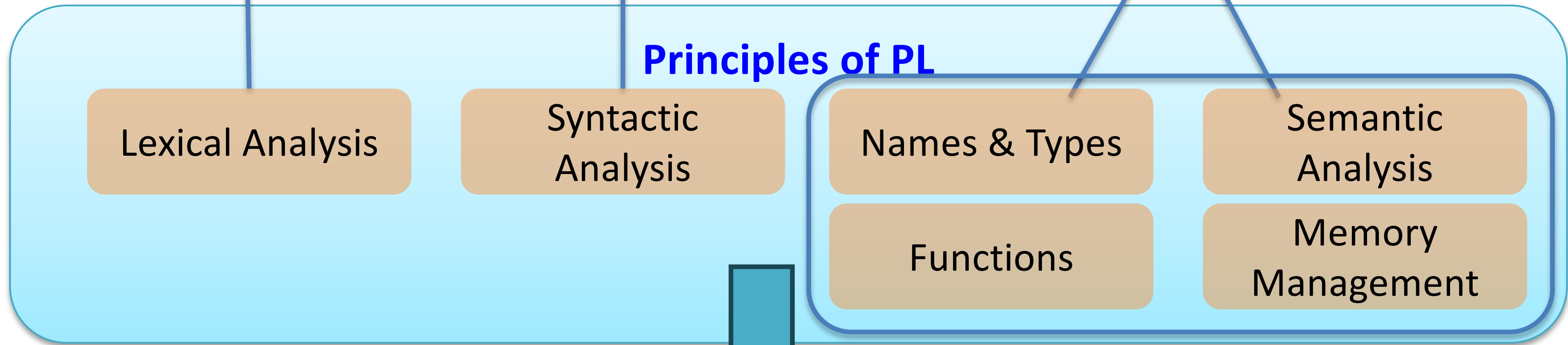
Complete
implementation

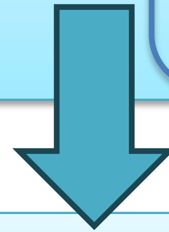
Some simplifications:
no for loop

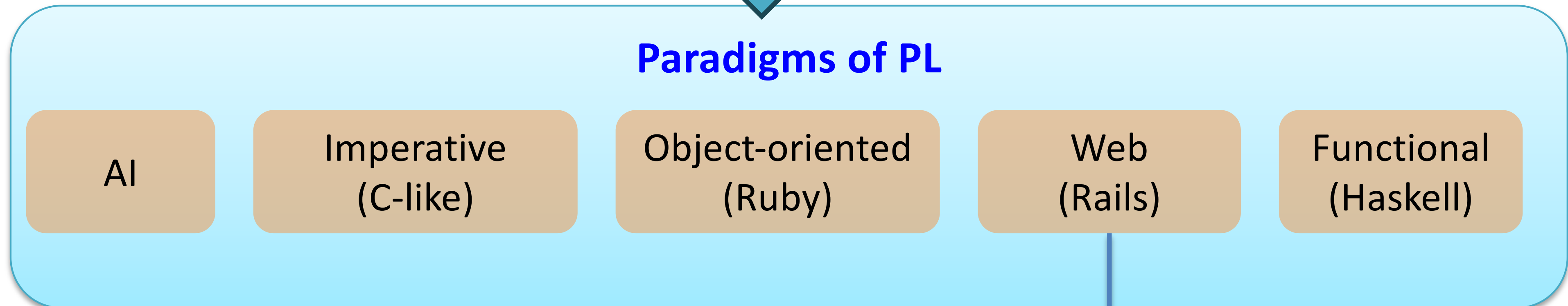
Full-fledged interpreter that can handle
language features like nested loops and
conditionals. Simplification: one function only.



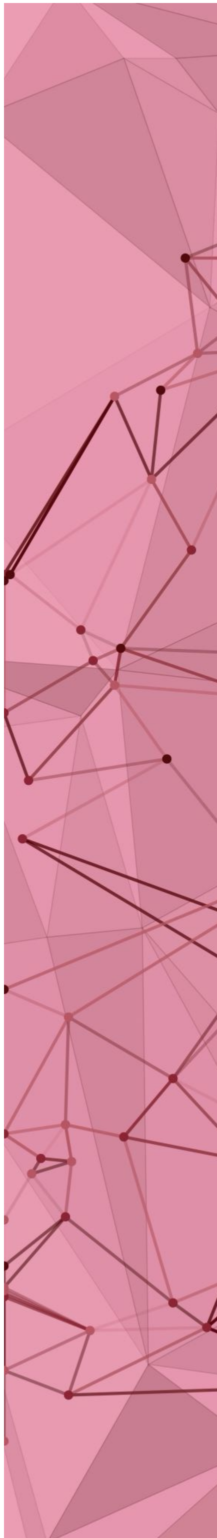








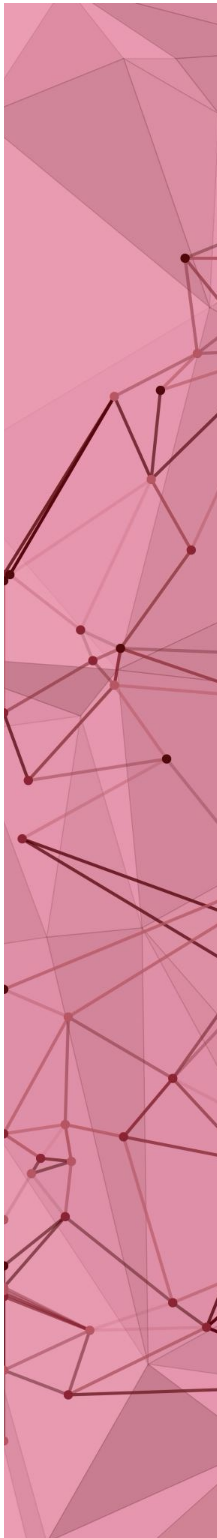
- What is semantics?
- How to define semantics?
  - Expressions
  - Assignment
  - Whole program: denotational semantics
- Lots of examples



Semantics

Precisely describes the meaning of all language constructs for:

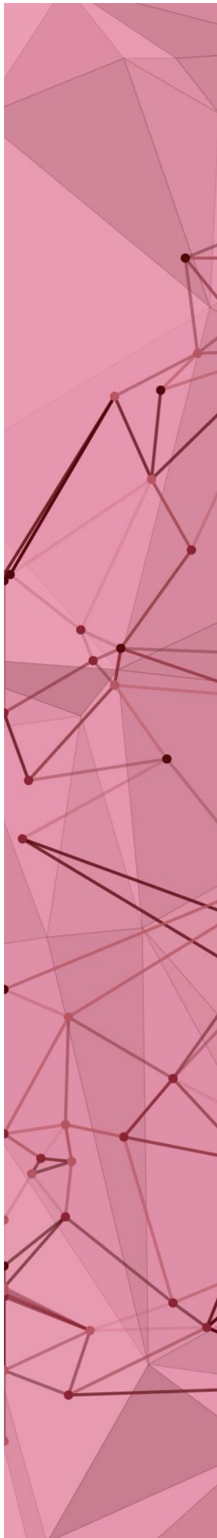
1. Programmers
2. Compiler writers
3. Standards developers



How to define semantics?

1. Operational semantics
   - The meaning attached by compiling using compiler C and executing using machine M. Ex: Fortran on IBM 709.
2. Axiomatic semantics
   - Use mathematical logic
3. Denotational semantics
   - Use mathematical functions
   - These functions transform the “state” of the program

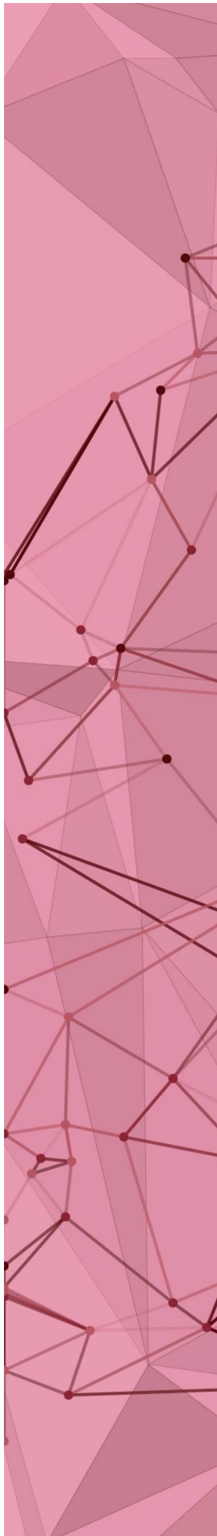
- We’ll use the concept of denotational semantics *informally*



Expressions

Expression semantics

Meaning of



- Infix (C, Java): a + b - c * d
  - Ambiguous without rules of prec. & assoc.
- Polish Prefix (Ambi): - + a b * c d
  - Unambiguous, but cannot use both unary and binary operators
- Polish Postfix (Postscript, calculator):
  a b + c d * -
- Cambridge prefix (LISP, Scheme):
  (- (+ a b) (* c d))

Associativity of Operators

Language	+ - * /	Unary -	**	== != < ...
C-like	L	R		L
Ada	L	R	non	non
Fortran	L	R	R	L

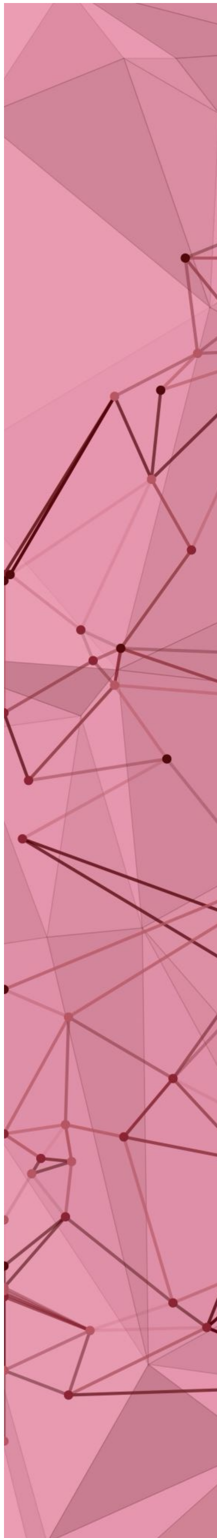
- Semantics of: a > b > c in C/C++?
- Why did Ada make ** and relational operators non-associative?

Precedence of Operators

Operators	C-like	Ada	Fortran
Unary -	7	3	3
**		5	5
* /	6	4	4
+ -	5	3	3
== !=	4	2	2
< <= ...	3	2	2
not	7	2	2

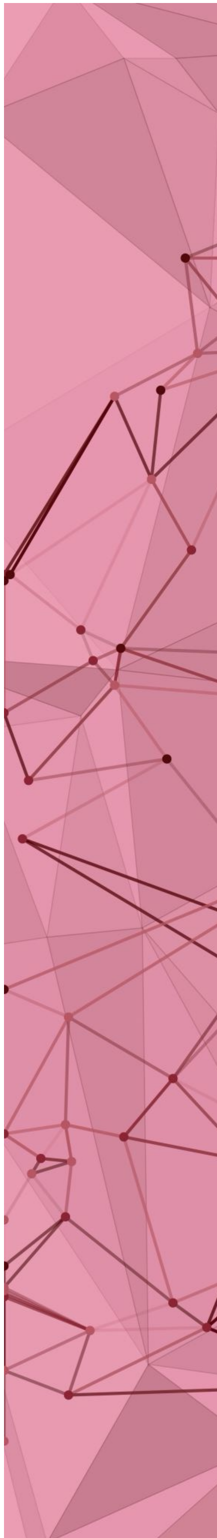
Short Circuit Evaluation

- a and b evaluated as:
  - if a then b else false
- a or b evaluated as:
  - if a then true else b



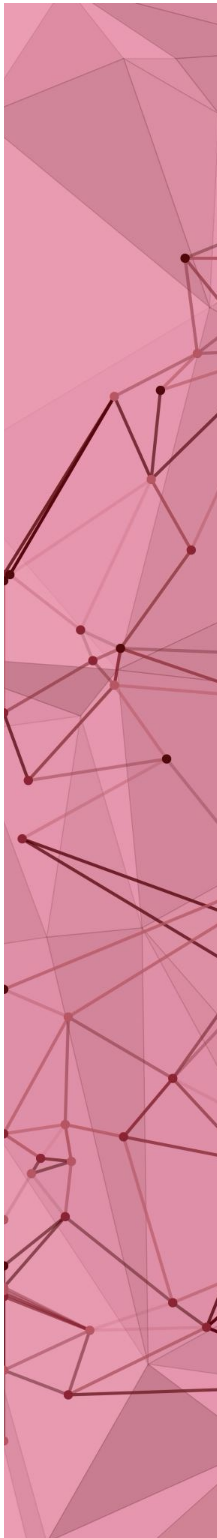
Example

```
Node p = head;
while (p != null && p.info != key)
	p = p.next;
if (p == null) // not in list
	…
else // found it
	…
```



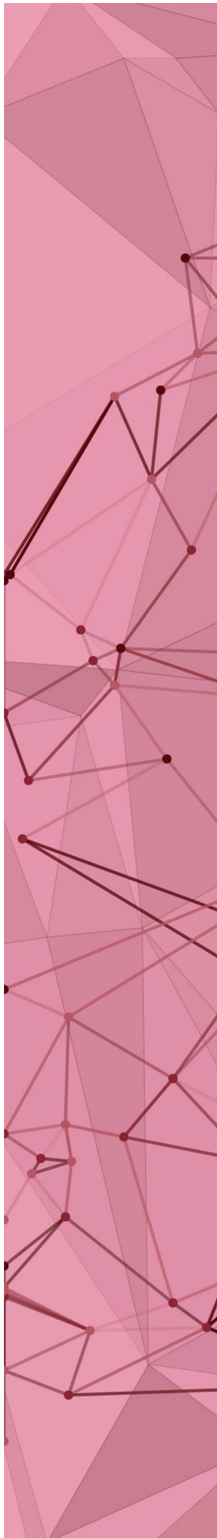
Question (see textbook)

Is the meaning of
$(a + b) + c$ the same as $a + (b + c)$?



Question (see textbook)

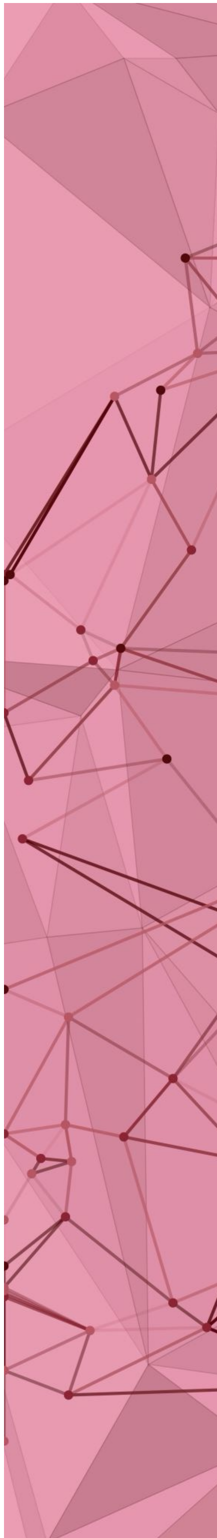
- What is the value of a below?
  i = 2;
  a = i++ * i++;
- The semantics of the RHS expression above is undefined in C.
- [ANSI C, 1990] "Between the previous and next sequence point [expr eval] an object shall have its stored value modified at most once by the evaluation of an expression. Furthermore, the prior value shall be accessed only to determine the value to be stored."
  - a[i] = i++; also undefined



Assignment

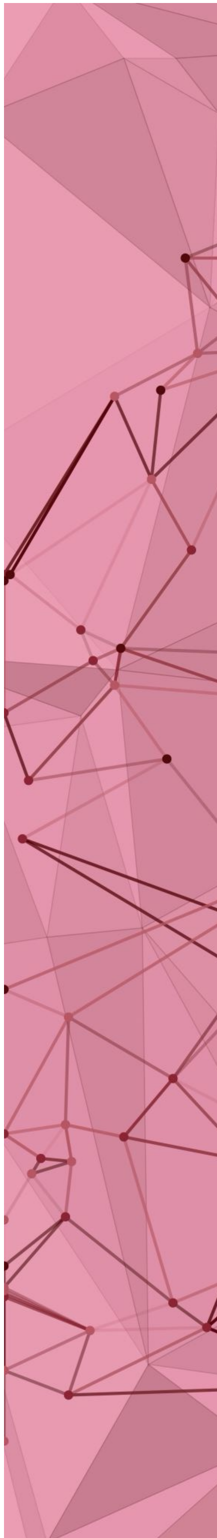
Assignment semantics

- Allow multiple assignments? a = b = c = 0
- OK by C/C++/Java/Python



Assignment semantics

- Is assignment an expression?
  - Why ask? If it’s an expression, you can use assignment as a conditional expression!
- C/C++: Yes [e.g. Kernighan & Ritchie's strcpy]
- Python: No
- Java: Yes only for Boolean expressions

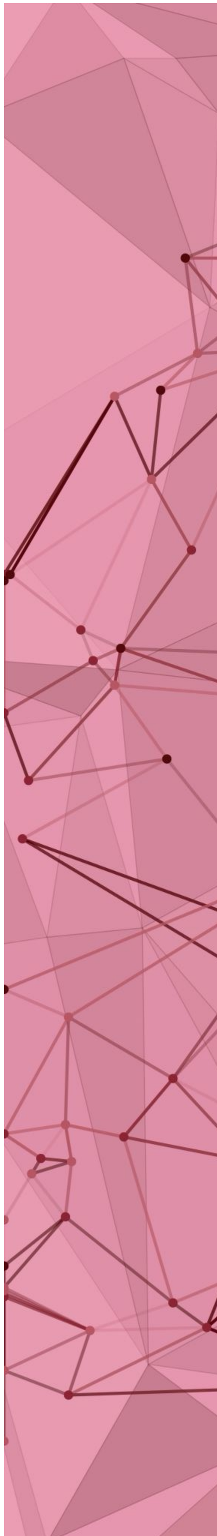


Examples

- C
  - //Kernighan and Ritchie's strcpy(char *p, char *q):
    while (*p++ = *q++) ;
  - while (ch = getc(fp)) ... // ???
  - while (p = p->next) ... // ???
- Java

```
boolean x = false;
if (x = true)
    System.out.println("x is true!"); //this will be printed!
```

- Python
  - These are illegal



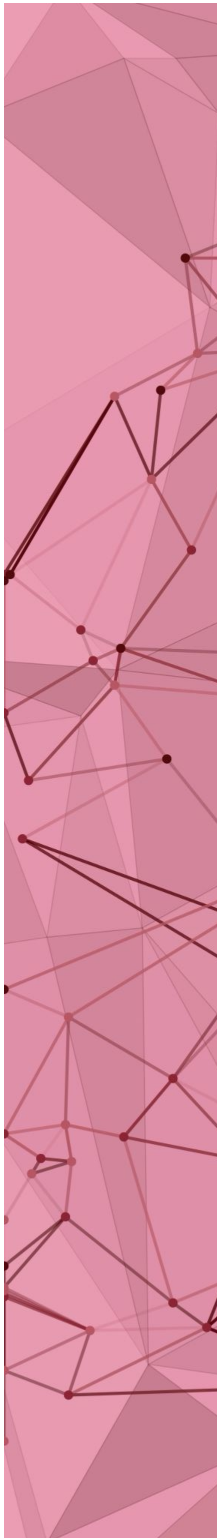
Assignment:
copy vs. reference semantics

x = y

Will the (R-)value of y be copied to x

or

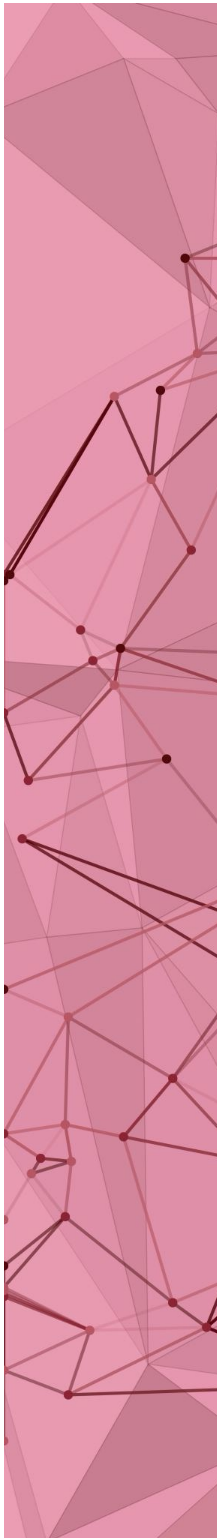
will x and y point to the same data?



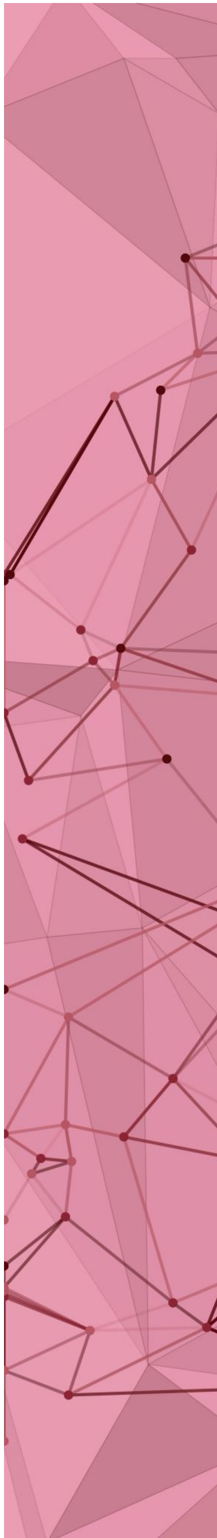
Denotational
Semantics

Program state

Collection of all active objects and their current values (binding!)



```
// compute n!
1        void main ( ) {
2         int n, i, f;
3          n = 3;
4          i = 1;
5          f = 1;
6          while (i < n) {
7                      i = i + 1;
8                      f = f * i;
9          }
10       }
```



```
// compute n!
1        void main ( ) {
2         int n, i, f;
3          n = 3;
4          i = 1;
5          f = 1;
6          while (i < n) {
7                  i = i + 1;
8                  f = f * i;
9          }
10       }
```

n	i	f
undef	undef	undef
3	undef	undef

```
// compute n!
1        void main ( ) {
2         int n, i, f;
3          n = 3;
4          i = 1;
5          f = 1;
6          while (i < n) {
7                  i = i + 1;
8                  f = f * i;
9          }
10       }
```

n	i	f
3	undef	undef
3	1	undef

```
// compute n!
1        void main ( ) {
2         int n, i, f;
3          n = 3;
4          i = 1;
5          f = 1;
6          while (i < n) {
7                  i = i + 1;
8                  f = f * i;
9          }
10       }
```

n	i	f
3	1	undef
3	1	1

// compute n!		n	i	f
1	void main ( ) {			
2	int n, i, f;			
3	n = 3;			
4	i = 1;			
5	f = 1;	3	1	1
6	while (i < n) {	3	1	1
7	i = i + 1;			
8	f = f * i;			
9	}			
10	}			

```
// compute n!
1         void main ( ) {
2          int n, i, f;
3           n = 3;
4           i = 1;
5           f = 1;
6           while (i < n) {
7                   i = i + 1;
8                   f = f * i;
9           }
10        }
```

n	i	f
3	1	1
3	2	1

```
// compute n!
1        void main ( ) {
2         int n, i, f;
3          n = 3;
4          i = 1;
5          f = 1;
6          while (i < n) {
7                  i = i + 1;
8                  f = f * i;
9          }
10       }
```

n	i	f
3	2	1
3	2	2

```
// compute n!
1        void main ( ) {
2         int n, i, f;
3          n = 3;
4          i = 1;
5          f = 1;
6          while (i < n) {
7                  i = i + 1;
8                  f = f * i;
9          }
10       }
```

n	i	f
…		
…		

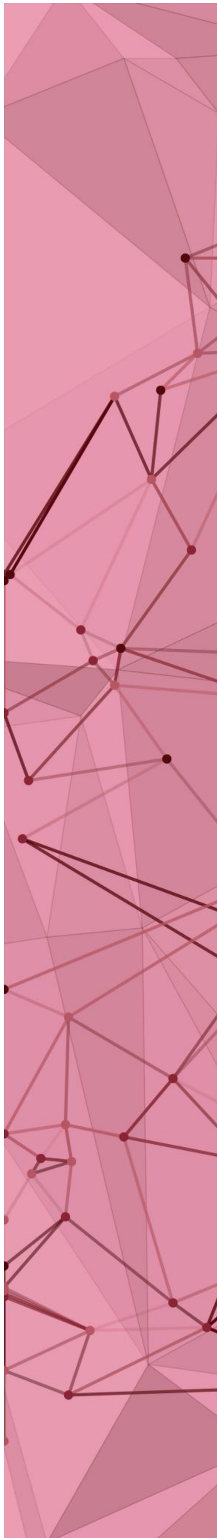
```
// compute n!
1   void main ( ) {
2    int n, i, f;
3     n = 3;
4     i = 1;
5     f = 1;
6     while (i < n) {
7         i = i + 1;
8         f = f * i;
9     }
10  }
```

n	i	f
3	3	6

Control flow semantics

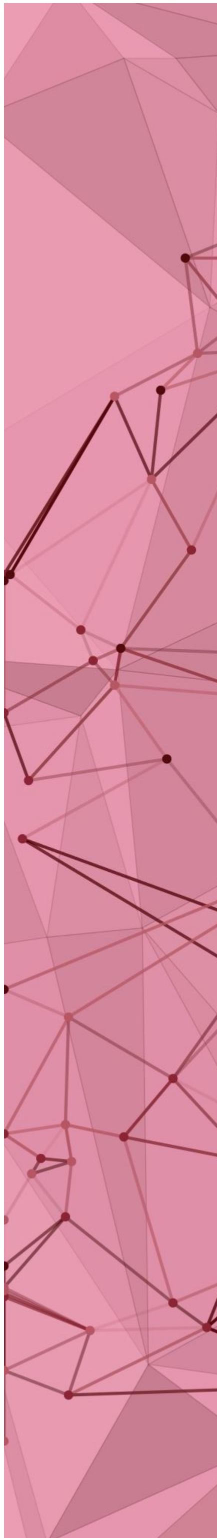
To be complete, an imperative language needs:

- Statement sequencing
- Conditional statement
- Looping statement



Sequence of statements

- s1
  s2
- Semantics:
  - First execute s1
  - Then execute s2
  - Output state of s1 is the input state of s2



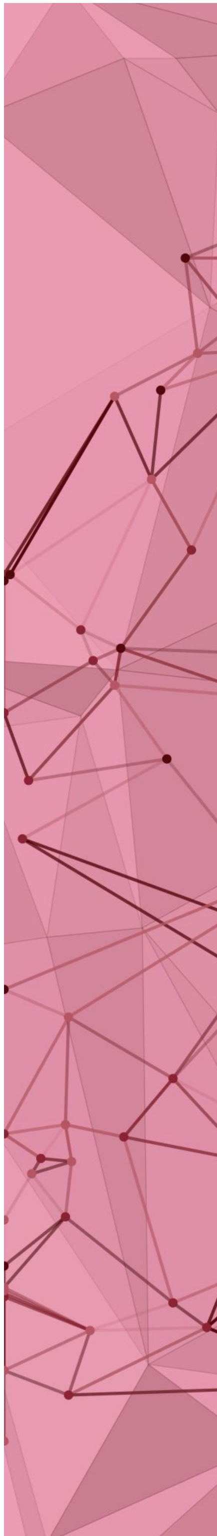
Conditional

- *IfStatement* $\rightarrow$ if (*Expression*) *Statement*
  [else *Statement*]

- Example:

```
if (a > b)
        z = a;
else
        z = b;
```

- If the test expression is true (here, no change of state)
  - then the output state of the conditional is the output state of the then branch,
- else the output state of the conditional is the output state of the else branch.



While loops

- *WhileStatement* $\rightarrow$ while (*Expr*) *Statement*
- The conditional expression is evaluated (state does not change)
- If it is true, first the statement is executed, and then the loop is executed again.
- Otherwise, the loop terminates.