Programming Paradigms

Control Flow (Part 1)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2020

Control Flow

Control flow: Ordering of instructions

- Fundamental to most models of computation
- Common language mechanisms
  - Sequencing, selection, iteration, recursion, concurrency, exceptions
- Each PL defines its rules
  - Think in terms of concepts, not specific syntax

Quiz: Argument Evaluation

What does the following Java code print?

```
class Warmup {
    static void f(int a, int b) {
        System.out.println(a + ", " + b);
    }

    public static void main(String[] args) {
        int i = 5;
        f(i++, --i);
    }
}
```

Please vote via Ilias

Quiz: Argument Evaluation

What does the following Java code print?

```
class Warmup {
    static void f(int a, int b) {
        System.out.println(a + ", " + b);
    }

    public static void main(String[] args) {
        int i = 5;
        f(i++, --i);
    }
}
```

Result: 5, 5

Please vote via Ilias

Quiz: Argument Evaluation

What does the following Java code print?

```
class Warmup {
    static void f(int a, int b) {
        System.out.println(a + ", " + b);
    }

    public static void main(String[] args) {
        int i = 5;
        f(i++, --i);
    }
}
```

Post-increment: Returns `i` and then increments it

Result: 5, 5

Please vote via Ilias

Quiz: Argument Evaluation

What does the following Java code print?

```
class Warmup {
    static void f(int a, int b) {
        System.out.println(a + ", " + b);
    }

    public static void main(String[] args) {
        int i = 5;
        f(i++, --i);
    }
}
```

Pre-decrement: Decrements `i` and then returns it

Result: 5, 5

Please vote via Ilias

Quiz: Argument Evaluation

What does the following Java code print?

```
class Warmup {
    static void f(int a, int b) {
        System.out.println(a + ", " + b);
    }

    public static void main(String[] args) {
        int i = 5;
        f(i++, --i);
    }
}
```

Evaluation order:

Left-to-right

Result: 5, 5

Please vote via Ilias

Overview

- Expression Evaluation 
- Structured and Unstructured Control Flow
- Selection
- Iteration
- Recursion

Expressions

Operator vs. operand

- Operator: Built-in function with a simple syntax
- Operand: Arguments of operator
- Examples:

```
i++

foo() + 23

(a * b) / c
```

Expressions: Notation

Three popular notations

- Prefix
  - `op a b` or `op(a, b)` or `(op a b)`
- Infix
  - `a op b`
- Postfix
  - `a b op`

Expressions: Notation

Three popular notations

- Prefix
  - `op a b` or `op(a, b)` or `(op a b)`
- Infix
  - `a op b`
- Postfix
  - `a b op`

Example: Lisp

```
(* (+ 1 3) 2)
```

Expressions: Notation

Three popular notations

- Prefix
  - `op a b` or `op(a, b)` or `(op a b)`
- Infix
  - `a op b` —— **Example: Java**
- Postfix
  - `a b op`

`(1 + 3) * 2`

Expressions: Notation

Three popular notations

- Prefix
  - `op a b` or `op(a, b)` or `(op a b)`
- Infix
  - `a op b`
- Postfix
  - `a b op`

Example: C

`a++`

Multiplicity

Number of arguments expected by an operator

- Unary
  - `a++` or `!cond`
- Binary
  - `a + b` or `x instanceof MyClass`
- Ternary
  - `cond ? a : b`
- (More are possible, but uncommon in practice)

Order of Evaluating Expressions

Given a complex expression, in what order to evaluate it?

Examples:

- Multiple arithmetic operations in Python:

```
2 + 3 * 4
```

- Mix of boolean and other expressions in Java:

```
!x && a == false
```

- Dereference and increment a pointer in C:

```
*p++
```

Precedence and Associativity

Choice among evaluation orders:

Specified by precedence and associativity rules of the PL

- Precedence: Specify which operators group "more tighly" than others
- Associativity: For operators of equal precedence, specify whether to group to the left or right

Precedence Levels in C

Operator	Meaning
++, −−	Post-increment, post-decrement
++, −−	Pre-increment, pre-decrement
*	Pointer dereference
<, >	Inequality test
==, !=	Equality test
&&	Logical and
||	Logical or
=, +=	Assignment

Higher means higher precedence

This list is incomplete.

Precedence Levels in C

Operator	Meaning
++, --	Post-increment, post-decrement
++, --	Pre-increment, pre-decrement
*	Pointer dereference
<, >	Inequality test
==, !=	Equality test
&&	Logical and
||	Logical or
=, +=	Assignment

Same precedence level

This list is incomplete.

Examples

- **Dereference and increment a pointer:**
  - `*p++`
- **Mix of logical operators:**
  - `a && b || c`
- **Mix of inequality and equality tests:**
  - `x < y == foo`

Examples

- **Dereference and increment a pointer:**
  - `*p++` means `*(p++)`
- **Mix of logical operators:**
  - `a && b || c` means `(a && b) || c`
- **Mix of inequality and equality tests:**
  - `x < y == foo` means `(x < y) == foo`

Examples

- **Dereference and increment a pointer:**
  - `*p++` means `*(p++)`
- **Mix of logical operators:**
  - `a && b || c` means `(a && b) || c`
- **Mix of inequality and equality tests:**
  - `x < y == foo` means `(x < y) == foo`

General rule:

When in doubt, use parentheses

Associativity Rules

- Decide about same-level operators
- Arithmetic operators:

  Mostly left-to-right a.k.a. left-associative

  - `12 - 3 - 2` yields `7` in most languages
  - Exception: Exponentiation is mostly right-associative
    - `2 ** 3 ** 2` yields `512` in most languages
    - But: `2 ^^ 3 ^^ 2` yields `64` in Excel
- Assignments: Mostly right-associative
  - `a = b = a + c` assigns `a + c` into `b` and then `a`

Quiz: Precedence and Associativity

1) What are the values of `foo` and `bar`

   (a) when assignments are left-associative?

   (b) when assignments are right-associative?

```
int foo = 1, bar = 2;
foo = bar = foo + bar;
```

2) What is the value of `z`

   (a) when `&&` has higher prededence than `||`?

   (b) when `||` has higher prededence than `&&`?

```
bool x = false, y = false, z = true;
bool z = x || y && y || z;
```

Please vote in Ilias

Quiz: Precedence and Associativity

1) What are the values of `foo` and `bar`

foo=2, bar=4

(a) when assignments are left-associative?

(b) when assignments are right-associative?

foo=3, bar=3

```
int foo = 1, bar = 2;
foo = bar = foo + bar;
```

2) What is the value of `z`

true

(a) when `&&` has higher prededence than `||`?

(b) when `||` has higher prededence than `&&`?

false

```
bool x = false, y = false, z = true;
bool z = x || y && y || z;
```

Please vote in Ilias

Ordering within Expressions

- **Discussed so far:**
  Order of performing operations
- **But: In what order are the operands evaluated?**
- **Example:**

```
a - f(b) - c * d
```

Ordering within Expressions

- **Discussed so far:**
  Order of performing operations
- **But: In what order are the operands evaluated?**
- **Example:**

```
a - f(b) - c * d
```

Has precedence over subtraction

Ordering within Expressions

- **Discussed so far:**

  Order of performing operations

- **But: In what order are the operands evaluated?**

- **Example:**

```
a - f(b) - c * d
```

Subtraction is left-associative:

This is computed first

Ordering within Expressions

- Discussed so far:
  Order of performing operations
- But: In what order are the operands evaluated?
- Example:

```
a - f(b) - c * d
```

But: Which of these two operands is evaluated first?

Why Does It Matter?

- **Reason 1: Side effects**
  - Evaluating `f(b)` may modify `c` or `d`
- **Reason 2: Compiler optimizations**
  - Influences register allocation and instruction scheduling

Example:

```
a - f(b) - c * d
```

Ordering: Language-specific

Different PLs: Different ordering within expressions

- Java and C#: Left-to-right
- C and many other languages: Undefined
  - Compiler can choose best order
  - Earlier example again:

```
int i = 5;
f(i++, --i);
```

Ordering: Language-specific

Different PLs: Different ordering within expressions

- Java and C#: Left-to-right
- C and many other languages: Undefined
  - Compiler can choose best order
  - Earlier example again:

```
int i = 5;
f(i++, --i);
```

May pass 5, 5 (left-to-right) or 4, 4 (right-to-left) to `f`

Short-circuit Evaluation

- **Saving time** when evaluating **boolean expressions**
- **Example:**

```
if (very_unlikely && very_expensive())
{
    ...
}
```

Short-circuit Evaluation

- **Saving time when evaluating boolean expressions**
- **Example:**

```
if (very_unlikely && very_expensive())
{
    ...
}
```

If first operand is false, no need to evaluate the second

Short-circuit Evaluation

- **Saving time when evaluating boolean expressions**
- **Example:**

```
if (very_unlikely && very_expensive())
{
    ...
}
```

But: Side effects of second operand may or may not happen

Short-circuit Evaluation (2)

- **Most PLs implement short-circuit evaluation**
  - Boolean and: Ignore second operand if first is false
  - Boolean or: Ignore second operand if first is true
- **One (relatively) popular exception: Pascal**

Short-circuit Evaluation (3)

- Beware that side effects in some boolean expressions may not happen
- Use it to your advantage:

```
// C code
p = my_list;
while (p && p->key != val) {
    ...
    p = p ->next;
}
```