Funcons
Basics of Imperative Programming

L. Thomas van Binsbergen

Royal Holloway, University of London

7 March, 2015
Section 1

Programming Constructs
Subsection 1

Expressions
Expressions

- An expression is evaluated to yield a value.
- Many expressions are ‘pure’.
- In imperative languages, expressions may have side effects.
- For example:
Expressions

- An expression is evaluated to yield a value.
- Many expressions are ‘pure’.
- In imperative languages, expressions may have side effects.
- For example:
  - \( i++ \)
Expressions

- An expression is evaluated to yield a value.
- Many expressions are ‘pure’.
- In imperative languages, expressions may have side effects.
- For example:
  - \( i++ \)
  - \( printMe(x) \)
Expressions

- An expression is evaluated to yield a value.
- Many expressions are ‘pure’.
- In imperative languages, expressions may have side effects.
- For example:
  - `i ++`
  - `printMe(x)`
  - In some languages: `x := 3`
Subsection 2

Statements
A *statement* has the primary goal of updating variables and printing.

Statements return no value, or the empty tuple ( ).

Examples:

- \( x := 3; \)
- **print** “hello”;
- What do you think of? \( 3 + 2; \)
If-Then-Else

Is **if-then-else** an expression or a statement?
Is **if-then-else** an expression or a statement?

- It can be both, based on the contents of the branches.
Is **if-then-else** an expression or a statement?

It can be both, based on the contents of the branches.

For example:

- **if** `true` { **print** “yea”; } **else** { **print** “nay”; }
- `true ? 10 : 5`
If-Then-Else

- Is **if-then-else** an expression or a statement?
- It can be both, based on the contents of the branches.
- For example:
  - ```
    if true { print "yea"; } else { print "nay"; }
    true ? 10 : 5
  ```
- Is **if-then** an expression or a statement?
Subsection 3

Declarations
A *declaration* has the primary goal of yielding an *environment*. An environment contains bindings from identifiers to values. A declaration may have side-effects, e.g. updating variables. For example:

- `int x;
- int y = 0;
- procedure void printMe(int x) { print x; return x; }

Section 2

Lab Preparation
Subsection 1

Effects
Expressions as Statements

- An expression can be considered a statement if we:
  - Evaluate the expression, optionally performing side-effects.
  - Discard the yielded value, and yield () instead.

- The `effect` funcon has this behaviour.

- `effect(X : T) : ()`
  - Descends `X`.
  - Replaced by `()`. 
Subsection 2

Variable Declarations
Imperative Variables

The following slides discuss how

- The *inherited* entity *environment*
- The *mutable* entity *store*

are used to define imperative variables with scoping rules.
The store binds variables to arbitrary values.
The store represents the computer’s memory.
A variable is a reference to a slot in memory.
Slots are allocated with a fresh variable referring to it.
A slot stores arbitrary values (no size restrictions).
The environment binds identifiers to variables.

Declarations extend the current environment with bindings.
Allocating Variables

- `var x = 0;` binds `x` to a fresh `variable`, whose value in the store is 0.
- `scope(bind("x", allocate-initialised-variable(0)), ...)`
Accessing Variables

- **print** \( x; \)
  - prints the value assigned to the variable bound to \( x \).
- **print**(assigned**(bound**(“x”))))
- What is the funcon translation of the expression \( x \)?
- In the lab you will implement the funcon translation of \( x := 3 \)
Scoping

- Bindings are local, as **environment** is an inherited entity.
- Therefore:
  - An **identifier** can be *out of scope*.
  - A **variable** can be *unbound*, in a certain scope.
- **Variables** are global, as **store** is a mutable entity.
- An assignment to a **variable** changes it everywhere.

**Examples**

- `seq(scope(bind("x", 3),...), bound("x"))`
- `seq(assign(bound("x"), 5), assigned(bound("x")))`
Subsection 3

Normal Control Flow
The *flow of control* is the sequence of statements in a program’s execution.

We have seen that *sequential* places statements in sequence.
Control flow can *branch* in two or more directions.
Which direction is taken is decided by evaluating an expression.
For example, **if-then-else** has:

- A **then** branch.
- An optional **else** branch.
- A Boolean expression known as the **condition**.

**Figure**: Control flow of **if-then** and **if-then-else**.
Normal Control Flow - while

- For example, **while** has:
  - An body which may or may not be executed.
  - A Boolean expression known as the **condition**.
  - Note the similarity with **if-then**.

![Control Flow Diagram](image)

**Figure**: Control flow of **while**.
Normal Control Flow - switch

For example, **switch** has:

- One or more cases.
- An expression yielding a value that can be *matched*.

Figure: Control flow of **switch**.
In the lab

- In the lab you will be asked to implement:
  - Boolean expressions
  - if-then-else using **if-then-else**
  - while using **while**
Subsection 4

Abnormal Control Flow
Abnormal control flow interrupts a sequence of statements. 
Control flow is continued elsewhere, or the program halts. 
Examples are: 
- GOTO 
- throw 

Funcons `throw` and `handle-thrown` are used to define most forms of abnormal control flow. 
In the lab you are asked to implement `return` statements.
Abnormal Control Flow

- Abnormal control flow interrupts a sequence of statements.
- Control flow is continued elsewhere, or the program halts.
- Examples are:
  - GOTO
  - throw
  - continue

- Funcons **throw** and **handle-thrown** are used to define most forms of abnormal control flow.
- In the lab you are asked to implement **return** statements.
Abnormal Control Flow

Abnormal control flow interrupts a sequence of statements.
Control flow is continued elsewhere, or the program halts.
Examples are:
- GOTO
- throw
- continue
- break

Funcons **throw** and **handle-thrown** are used to define most forms of abnormal control flow.
In the lab you are asked to implement **return** statements.
Abnormal Control Flow

- Abnormal control flow interrupts a sequence of statements.
- Control flow is continued elsewhere, or the program halts.
- Examples are:
  - GOTO
  - throw
  - continue
  - break
  - return
- Funcons **throw** and **handle-thrown** are used to define most forms of abnormal control flow.
- In the lab you are asked to implement **return** statements.