Lectures 3-4-5-6
Basics in Procedural Programming: Machinery

prof. Marco Bellia, Dip. Informatica, Università di Pisa

February 21, 2014

Basics in Procedural Programming: Machinery

- Naming and Binding
- Mutable Values: Denotable, Storable and Expressible Value
- Env, Store, AR and Blocks: Motivations
- Blocks: Inline blocks and Procedure/function (body) block
- Blocks: Static and Dynamic Scope
- Activation Records: Structure and Implementation
- Programming Unit
- Aliasing, Closures, Lambda Lifting
- Env: Formalization and Implementation
- Store: Formalization and Implementation

Naming and Binding

- **Naming** = Use of *identifiers* to refer to *definitions* of programming entities
- **Definition** of the entity = Definition results a *Denotable Value* of the language **semantic domain** *Den*

Example

final double pigreco = 3.15; /*an example of constant*/
int y = 5; /*an example of variable*/
See next slide, for other examples of definitions.

- **Binding** = Association between the name and its definition
- Bindings of a program are all collected in the **semantic structure** *Env*

Mutable and Immutable values

- Names for *Constants* and *Variables* are in common use in the introduction (i.e. declaration) of values
- **Variable** = It is synonym of **Mutable Value**
  - Mutable values are basics in Imperative Languages but they are non incompatible with Descriptive Languages, in principle
  - However, problems arise from the different ways in which such values can be used in the program: Haskell preserves Transparency Property, instead Ocaml does not.
- **Constant** = It is an example of **Immutable Value**

Example

```
Var x: int /*Pascal Declaration of a mutable value*/
Const y:int /*Pascal Declaration of a immutable value*/
int z[] /*C Declaration of an immutable, structured, value with mutable components of type int */
*int y /*C Declaration of an mutable value that it is, in turn, a mutable value yet, namely a pointer*/
label u /*Pascal Declaration of a immutable value, namely a position in program */
void p(...){...} /*Declaration of an immutable value, namely a procedure*/
public class A{...} /*Declaration of an immutable value, namely a class (of Java)*/
struct S{...} /* Declaration of an immutable value, namely a type record (of C) */
type B = ... /*Again, declaration, in OCaml, of an immutable value, namely a (concrete) type*/
```

Mutable and Immutable values: The Equality Property

The two classes of values definitely, differ in some form of state that is underlying of mutable values. This is clearly, reflected from the behavior of (almost all) operations of the two classes, and then, from their use in programming.

- However, what about comparing two values?
- The **Equality Property** = Two values are equals only if they can be exchanged with one another, in the program.
  - Each mutable value is equal only to itself

Example

Comment the equality predications in the following C++ text:

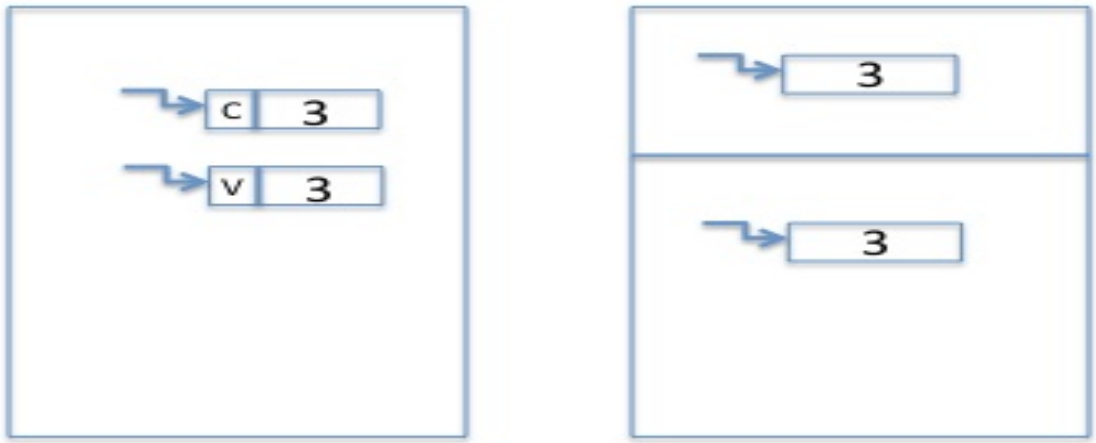
```
const int c1 = 3;
const int c2 = 3;
int x1 = 3;
int x2 = 3;
int *y1 = &x1;
int *y2 = &x2;
printf("%2d\n",c1==c2);
printf("%2d\n",x1==x2); /*.... */
printf("%2d\n",y1==y2); /* .... */
```

Mutable and Immutable values: Implementation Skills

The two classes of values definitely, differ in some form of state that is underlying of mutable values. This is clearly, reflected from the behavior of (almost all) operations of the two classes, and then, from their use in programming.

- However, what can we say about the internal representation of such values?
- Implementation Skills

Example



One immuable and one immutable value in two different memory organizations: Memory on the right has a *constant pool memory*.

Den, Mem, Val: The Value Domains of a Language

- The Value (semantic) Domains are a fundamental characteristic of a language
- They highly constrain the way in which algorithms may be written, in the language
  - **Val** = Domain of the Values that can be involved in the language programs
  - **Den** = D. of the Values that can be expressed in definitions
  - **Mem** = D. of the Mutable Values of the language
- The following hold: (1) $Den \subseteq Val$; (2) $Mem \subseteq Val$
- But: $Den \cup Mem = Val$; $Mem \subseteq Val$; and so on ... hold or not depending on the language

Example

int A[3] = {3,5,17}; /* define, in C, a mutable value in a binding for A */
A = {3,5,12}; /* is not permitted */
Then: Is {3,5,17} defining a mutable or immutable value?
In providing for an answer, compare it with:
int B = 3; /* define, in C, a mutable value in a binding for B */
B = 15; /* a common statement in C */
Then: 3 and 15 are immutable integers.
What can you say about the other languages that you know?

Expressible Values

- Expressible Values = Have an explicit, syntactic, presentation in the language;
- These values are useful for introducing constant values in expressions;
- Hence, values of common use in the expressions of the language are also expressible value;

Example

Integers are expressible values of C
Are *arrays* expressible values of C?
Are *lists* expressible values of Ocaml (Haskell)?
Are *vectors* expressible values of Java?
Are *functions* expressible values of Ocaml (Haskell)?
Are *methods* expressible values of Java?

Env, Store, AR

- **Env** = Semantic Structure for collecting the bindings of the program
  - It is quite related to the symbol tables of the front-end of language executors and Compilers
  - Machine Languages do not use naming and do not require Env
- **Store** = Semantic Structure for Mutable Values, i.e. Mem
  - Additional Memory components are always present in the implementation machinery, to handle immutable values and the program representation of all languages (including pure Functional ones)
- **AR** = Implementation Machinery Component for the computation control
  - It is used to support the program sectioning into parts that:
    - can be executed separately,
    - and, include inline blocks, procedures/functions, modules, monitors, threads,..

Program Sectioning: Blocks

- Block = It is used for creating sections of program that are:
  - (partially) autonomous in the definitions that may be used, and
  - may exhibit specific functionalities, and
  - may be valid supports in program verification and modification

- Two main kinds in Procedural Programming:
  - inline blocks
  - procedures and functions

Blocks: Inline vs. Procedures

- Inline Blocks
  - anonymous
  - contain two parts: Local Definitions and End/Exit Code;
  - may be nested: Execution exits nested blocks in reverse order to the entering

- Procedures and Functions
  - named
  - contain three parts: Parameter Transmission, Local Definitions and Return/Exit Code;

Example

(a) According to the above features, describe the features of the blocks of the compound statements of the language C.
(b) Moreover, answer to: in what features the inline blocks of Java differ from the ones described in the slide

- Suggested Reading:
  Gabrielli M., S. Martini, Programming Languages: Principles and Paradigms, Springer, 2006 - Chapter 4-4.2