Principles of Programming Languages

Lecture 10

Variables and Assignment
Variable ≠ Variable

(Math) (CS)

- **Math**
  - \( \pi \)
  - \( 3.1415828... \)
- **CS**
  - \( \pi \)
  - \( 3.14 \)

\( x \)

definite singular description

\( \ldots \)

real

\( \text{ref real} \)

\( \text{real} \)

\( \text{e.g. Prolog & FP (IFP)} \)

\( \text{e.g. Pascal & Algol 68} \)

\( \text{symbol} \)

\( \text{referent} \)
Constants

- Algol 68: `real pi = 3.1416;`
- C: (not quite) `#define PI 3.1416`
- Pascal: `const pi = 3.1416;`
- Ada: `pi: constant FLOAT := 3.1416;`

- No reference available
- `No ref real object exists`
- `pi` replaced by literal
- `No stack (or any other) space allocated (stored in instruction)`
- `Pure definition`
Variables

• Algol 68: \texttt{begin} real \(x, y\);
• C: \{ float \(x, y\);
• Pascal: \texttt{var} \(x, y\): real;
• Ada: \(x, y\): FLOAT;

\[
x := 2.0; \quad y := 3.0;
\]

\[
\begin{array}{c}
\text{x} \\
\text{ref real} \\
\text{real}
\end{array}
\quad \begin{array}{c}
\text{y} \\
\text{ref real} \\
\text{real}
\end{array}
\quad \begin{array}{c}
\text{real} \\
\text{real} \\
\text{3.0}
\end{array}
\quad \begin{array}{c}
\text{2.0}
\end{array}
\]
Assignment (after \( y \) gets 3)

- **Algol 68:** \( x := y; \)
- **C:** \( x = y; \)
- **Bliss:** \( x \leftarrow .y \)
Variables: Bliss v. C

Bliss

- Explicit dereferencing

\[
x = .y + .z
\]
\[
a +.j \leftarrow (a +.i)
\]

C

- Context dereferencing

\[
x = y + z
\]
\[
a[j] = a[i]
\]

\text{Synonym: } *(a+j) = *(a+i)
\]
\text{Only lvalue is actually computed}
Algol 68 aside: identity declaration

- `real x := c` NOT same as `real x = c`
- Identity declaration `real x = 2.718;` means

```
    x
     |
      2.718
```
- Initialized variable `real x := 2.718;` means

```
    x
     |
      2.718
```
- Why? Implicit `local` (stack) allocation
real $x$; abbreviates loc real $x$; which abbreviates the following identity declaration:

ref real $x = loc$ real;

**Local generator**

Yields a ref real

**ref real (lvalue)**

Ascribed to identifier $x$
Initialized Variables

- **Algol 68:** `begin real p := 3.1416;`
- **C:** `{ float p = 3.1416;`
- **Pascal:** `var p: real; begin p := 3.1416;`
- **Ada:** `p: FLOAT := 3.1416;`

**What is "variable" about a variable?**
- Not reference (location, lvalue)
  - Ref cannot be changed; "ascribed to" or "possessed by" identifier
- Not identifier
- Not type (in *most* languages)
- The value (rvalue)
Algol 68 identity declaration: unifies variable & constant declaration

1. `loc real m := 3;`
2. `real x = m;`
3. `ref real x = m;`
4. `real x := m; ⇒ loc real x := m; ⇒ ref real x = loc real := m;`

5. `real t = x*m;`

- `t` const. In new environment
- `x*m` evaluated when declaration encountered
Pointers

• Refs to variable refs
• Algol 68: \texttt{ref real px ;}
• C: \texttt{float *px ;}
• Pascal: \texttt{var px: ^real ;}
• Ada: \texttt{type PTR is access FLOAT ;}
px : PTR; (PTR=`access type” to `base type”)

\begin{tikzpicture}[scale=0.8]
\node (px) at (0,0) {px};
\node (ref) at (-1,-1) {\texttt{^ref real}};
\node (real) at (-2,-2) {\texttt{ref real}};
\node (ref_real) at (-2,-3) {\texttt{ref real}};
\draw (px) -- (ref);
\draw (ref) -- (real);
\draw (real) -- (ref_real);
\end{tikzpicture}
Algol 68 pointer declaration

\[
\begin{align*}
\text{ref real } & px \Rightarrow \\
\text{ref ref real } & px = \text{loc ref real}; \\
\end{align*}
\]
Pointers (cont’d)

- **Algol 68:** \( px := x \);
- **C:** \( px = &x \);
- **Bliss:** \( px \leftarrow x \)

- **NOT** Pascal, Ada picture. Closest is:
  - Pascal: \( \text{new}(px); \ px^@ := x; \)
  - Ada: \( px := \text{new} \ FLO\ A\ T(x); \)

```
```

Pointers in Pascal/Ada: point into heap

- Defeats "dangling pointers" at block exit
- Pointers cannot point into the stack
- \texttt{var x: real; var px, py: ^real;}

\begin{center}
\begin{tikzpicture}
    \node (x) {x};
    \node (px) [below of=x] {px};
    \node (py) [right of=px] {py};
    \node (stack) [above of=py] {stack};
    \node (heap) [right of=py] {heap};
    \node (2.0) [below of=py] {2.0};

    \draw [->] (x) -- (px);
    \draw [->] (px) -- (py);
    \draw [->] (py) -- (2.0);
\end{tikzpicture}
\end{center}
Pascal/Ada pointers (cont’d)

- new(px); px^ := x;

``anonymous variable px^``

Pascal/Ada pointers point only to ```unnamed```` data objects
Pascal/Ada pointers (cont’d)

- \( x := 4.0; \)
- \( \text{new}(py); \; py := px; \)

\[ \begin{array}{c}
\text{x} \\
\text{px} \\
\text{py}
\end{array} \quad \text{stack} \quad \begin{array}{c}
\text{heap} \\
\text{4.0} \\
\text{2.0}
\end{array} \]
Pascal/Ada pointers (cont’d)

- \( x := 4.0; \)
- \( py^ := px^; \)

```
pointer value out of legal range```

Diagram showing stack and heap with pointers and values.
Pascal/Ada pointers (cont’d)

- \( x := 4.0; \)
- \( \text{new}(py); \quad py^ := px^; \)
Algol68 has a heap allocator

- `ref ref real px = heap real;`
- `ref real hx = heap real;`  
  - Abbreviated `heap real hx;`

- Pascal: lvalues of all var identifiers are in stack
  - lvalues of all anonymous variables are in the heap
- Algol68: no restriction
  - allocation orthogonal to declaration
Refs and Aliasing

- Algol68: `ref real py; py := px;`
Refs and Aliasing (cont’d)

- Algol 68: `ref real y = x;` (identity declaration)
Bliss: untyped bit strings

- `local x; x ← 2; local px;`
- `px ← x``ptr to x stored in px``
- picture after assignment
Semantics of Variable Uses: C v. Bliss

- On LHS: $x = / \leftarrow$
- On RHS: $x = \& x \leftarrow$

C:

Meaning of $x$ **context-sensitive**
Not ``referentially transparent''

Bliss:

Meaning of $x$ *is* ``referentially transparent''
Variable Uses: C v. Bliss (cont’d)

- **C:** & **not** a unary operator
  - & (x+y) meaningless
  - &&x meaningless (can’t reverse a pointer)
  - &2 meaningless

- **Bliss:** . **is** a unary operator
  - . (x+y) sensible
  - . . x sensible
  - . 2 sensible

\[ \begin{array}{c}
2 = 0010 \\
.2 = 0010: 1101 \\
..2 = 1101: 1000
\end{array} \]
C * ≠ Bliss.

Bliss variable occurrences are "referentially transparent"
Semantics of x is independent of context