CS 2SC3 and SE 2S03 Fall 2008

03 Control Structures

William M. Farmer

Department of Computing and Software
McMaster University

8 October 2008

Data Structures

- A data structure is a portion of memory that holds a structured collection of values.
  - A data structure may be itself a value.
- Various operators are associated with each kind of data structure:
  - Constructors for creating data structures.
  - Selectors for retrieving the values in data structures.
  - Mutators for modifying the values in data structures.
- Access to these operators needs to be controlled to ensure data privacy, integrity, and availability.
- Some data structures do not have mutators.
- The imperative programming paradigm heavily uses mutable data structures.
- The functional programming paradigm avoids using mutable data structures.

Example: Pairs

- A pair is a data structure that holds an order pair $\langle a, b \rangle$ of two values a and b with unspecified types.
- Constructor:
  - pair(a, b) creates a data structure p holding $\langle a, b \rangle$.
- Selectors:
  - get-fst(p) returns a, the first value in p.
  - get-snd(p) returns b, the second value in p
- Mutators:
  - set-fst(p, x) sets a, the first value in p, to x.
  - set-snd(p, x) sets b, the second value in p, to x.
  - Note: a and x (as well as b and x) need not have the same type.

Use of Pairs

- The pair data structure can be used to build many other useful data structures.
  - It is the chief data structure of Lisp.
- Pairs can be used to define tuples:

  $$(a_1, a_2) = \langle a_1, a_2 \rangle.$$
  $$(a_1, \ldots, a_n) = \langle a_1, (a_2, \ldots, a_n) \rangle \text{ for } n \geq 3.$$

- Pairs can be used to define lists:
  - $[\,] =$ nil, some special value.
  - $[a_1] = \langle a_1, [\,] \rangle$.
  - $[a_1, \ldots, a_n] = \langle a_1, [a_2, \ldots, a_n] \rangle$ for $n \geq 2$.

Example: References

- A reference of type t is a data structure that holds a value of type t.
- A reference is said to reference or point to its value.
- In OCaml, `ref` is a polymorphic type of references.
- Constructor: `ref` expr constructs a reference of the type of t `ref` where t is the type of expr.
  - Example: `let x = ref 8 ;;`
- Selector: If expr is a reference, `!`expr selects the referenced value of the reference.
  - Example: `!x ;;`
- Mutator: If expr_1 is a reference of type t and expr_2 is a value of type t, then expr_1 `:=` expr_2 sets the referenced value of expr_1 to expr_2.
  - Example: `x := 7 ;;`

References in C

- References are implemented in C as memory addresses.
- A reference of type t is a memory address of a location that can hold a value of type t.
- In C, a variable of type t is bound to a reference of type t.
- Constructor: `int x;` constructs a reference of type `int` and binds `x` to it.
- Value selector: `x;` selects the referenced value (of the reference `x` is bound to).
- Address selector: `&x;` selects the address (of the reference `x` is bound to).
- Mutator: `x = 3;` sets the referenced value (of the reference `x` is bound to) to `3`.

Control Structures

- A control structure controls the execution of statements in a program.
- Before control structures were invented, execution was controlled in an unstructured manner using conditionals and goto statements.
  - This made the control flow of the program exceedingly difficult to understand.
- There are three main categories of control structures:
  1. Sequential control structures allow a sequence of statements to be executed one after another.
  2. Conditional control structures allow a statement to be selected for execution on the basis of whether a condition evaluates to true or false.
  3. Iterative control structures allow a statement to be repeatedly executed.
- There are several kinds of control structures in each of these categories.

Block

- A block is a sequential control structure that treats a sequence of statements as a single statement.
- The statements in a block are executed left to right.
- OCaml has two syntaxes for blocks:

  $$\texttt{(}\mathsf{expr}_1\,\texttt{;}\ \cdots\ \texttt{;}\ \mathsf{expr}_n\texttt{)}$$
  $$\texttt{begin}\ \mathsf{expr}_1\ \texttt{;}\ \cdots\ \texttt{;}\ \mathsf{expr}_n\ \texttt{end}$$

- C has the following syntax for blocks:

  $$\{\mathsf{stmt}_1\ \cdots\ \mathsf{stmt}_n\}$$

For Loop

- The for loop is an iterative control structure that executes a statement for a certain number of times.
- For loop iteration is normally bounded.
- OCaml has two syntaxes for blocks:

  ```
  for name = $\mathrm{expr}_1$ to $\mathrm{expr}_2$ do $\mathrm{expr}_3$ done
  for name = $\mathrm{expr}_1$ downto $\mathrm{expr}_2$ do $\mathrm{expr}_3$ done
  ```

  where expr_1 and expr_2 are expressions of type `int` and expr_3 is an expression of type `unit`.
- C has the following syntax for for loops:

  ```
  for ($\mathrm{expr}_1$; $\mathrm{expr}_2$; $\mathrm{expr}_3$) stmt
  ```

  which is equivalent to

  ```
  $\mathrm{expr}_1$;
  while ($\mathrm{expr}_2$) {
    stmt
    $\mathrm{expr}_3$;
  }
  ```

While Loop

- The while loop is an iterative control structure that executes a statement as long as a condition is true.
- While loop iteration is unbounded.
- The while loop is more general than the for loop; it can simulate a for loop.
- OCaml has the following syntax for while loops:

  `while` expr$_1$ `do` expr$_2$ `done`

  where expr$_1$ is of type `bool` and expr$_2$ is of type `unit`.
- C has the following syntax for while loops:

  `while (`expr`)` stmt