CS 2SC3 and SE 2S03 Fall 2008

04 Records and Arrays

William M. Farmer

Department of Computing and Software
McMaster University

5 November 2008

Records

- A record is a data structure that holds a tuple whose members are named.
- The members of a record are called fields and are indexed by their names.
- Records are also called structures (as in C).
- The type of a record is declared before a record is defined.
- The fields of a record are usually mutable.
- Records of a type t have a constructor, an indexed selector, and possibly an indexed mutator.

Records in OCaml

- Record type declaration:

  `type` rec-type `=` $\{$name$_1$ `:` t_1 `;` $\ldots$ `;` name$_n$ `:` $t_n\}$ `;;`

- Constructor:

  $\{$name$_1$ `=` expr$_1$ `;` $\ldots$ `;` name$_n$ `=` expr$_n\}$

- Selector:

  e `.` n

  where e is an expression of a record type having a field named n.

- A field in a record is mutable if it is declared mutable:

  `type` rec-type `=` `{` $\ldots$ `;` `mutable` name$_i$ `:` t_i `;` $\ldots$ `}` `;;`

- Mutator:

  e_1 `.` n `<-` e_2

  where e_1 is an expression of a record type having a mutable field named n and e_2 is of the same type as e_1 `.` n.

Arrays

- An array is a data structure that holds a list such that each member in the list can be directly accessed and modified.
- The members of an array are called cells and are indexed by natural numbers.
- Arrays usually have a constructor, an indexed selector, and an indexed mutator.
- Arrays may be multidimensional.
  - A one-dimensional array is called a vector (as in OCaml).
- Strings are often implemented as arrays (as in C and OCaml).

Arrays in OCaml

- Constructor:

  `[|`expr_1`;` $\ldots$ `;` expr_n`|]`

- Selector:

  e_1`.(`e_2`)`

  where e_1 is an expression denoting a vector and e_2 is an expression denoting a value i of type `int` with $0 \leq i \leq n-1$ where n is the length of the vector.

- Mutator:

  e_1`.(`e_2`)` `<-` e_3

- The `Array` module contains several other constructors, selectors, and mutators including `create`, `append`, and `length`.

Character Strings in OCaml

- Strings are a special kind of array in OCaml.
- Constructor:

  $"c_1 c_2 \cdots c_n"$

- Selector:

  $e_1 \,.\, [e_2]$

- Mutator:

  $e_1 \,.\, [e_2] \texttt{ <- } e_3$

Anomalies and Exceptions

- An anomaly is an unexpected behavior by a service.
- An exception is a signal to the client of a service that an anomaly has been exhibited by a service.
  - An exception is thrown or raised when the anomaly occurs.
- A thrown exception is caught by an appropriate exception handler that tries to handle the exception.
- Ways an exception can be handled:
  1. An attempt is made to recover from the anomaly.
  2. The exception is thrown higher up the uses chain.
  3. The state of the program providing the service is repaired as best as possible and then the service is allowed to fail.
- Exceptions can be used to change the normal flow of control.

Exceptions in OCaml

- Exception declaration:

  `exception` cap-name `;;`

- Raising an exception:

  `raise` cap-name `;;`

- Catching and handling an exception:

  `try` expr `with`
  `|` p_1 `->` expr_1
  ⋮
  `|` p_n `->` expr_n

Stacks

- A stack is a data structure that holds a list such that the elements are accessed according to the principle of last in first out (LIFO).
- Stacks are employed extensively in computer systems.
- Constructor:

  $$\text{bottom} : \text{void} \longrightarrow \text{stack}.$$

- Selectors:

  $$\text{height} : \text{stack} \longrightarrow \text{nat}.$$
  $$\text{top} : \text{stack} \longrightarrow \text{element}.$$

- Mutators:

  $$\text{push} : \text{element}, \text{stack} \longrightarrow \text{void}.$$
  $$\text{pop} : \text{stack} \longrightarrow \text{void}.$$

Queues

- A queue is a data structure that holds a list such that the elements are accessed according to the principle of first in first out (FIFO).
- Constructor:

  $\text{empty} : \text{void} \longrightarrow \text{queue}.$

- Selectors:

  $\text{length} : \text{queue} \longrightarrow \text{nat}.$
  $\text{front} : \text{queue} \longrightarrow \text{element}.$

- Mutators:

  $\text{push} : \text{element}, \text{queue} \longrightarrow \text{void}.$
  $\text{pop} : \text{queue} \longrightarrow \text{void}.$

Abstract Data Types

- An abstract data type (ADT) is a set of data and a set of operations that can be applied to the data.
  - The data and operations are described abstractly without reference to how they are implemented.
- ADTs are essentially the same as algebras in algebraic specification and mathematical structures in mathematics.
- In many ADTs, the set of data is recursively defined by the operations.
  - Examples: Natural numbers, lists, stacks, trees.
- An ADT is a special case of a module that consists of an interface and an implementation.
- ADTs support the Principle of Separation of Concerns: the interface of an ADT is separated from its implementation.
- ADTs support the Principle of Information Hiding: The "secret" of the ADT is hidden from the user.

Pointers in C

- A pointer in C is a variable of a reference type.
  - Reference types in C are called pointer types.
  - The value of a pointer is a memory address that refers or points to a value.
- Constructor:
  - `int * ip = NULL;`
  - `int * ip = &i;`

  creates a pointer of pointer type `int *`.
- Selector for dereferencing:
  - `*ip`

  denotes the value that the pointer `ip` points to.
- Mutator for dereferencing:
  - `*ip = i;`

  sets the value that the pointer `ip` points to.
- Pointers are used extensively in C.
  - Dangerous and tricky, they must be used very carefully!

Pointer Arithmetic in C

- The `sizeof` operator takes a variable x or type t as input returns an integer that is the number of 8-bit bytes reserved for x or t.
- A pointer p of type t can be viewed as an index into a giant array of cells of size `sizeof`(t):
  - $p + 1$ (pointer addition) is the next index into this giant array.
  - $p - 1$ (pointer subtraction) is the previous index into this giant array.
- Pointer arithmetic is not valid with `void` pointers because values of type `void` do not have a fixed size.
- Pointer arithmetic provides a powerful and uniform mechanize for accessing memory, but it can lead to dangerous and undesired memory access.

Evaluation Strategies

- Let p be a procedure with parameters $x_1, \ldots, x_n$ that is applied to arguments $a_1, \ldots, a_n$.
- An evaluation strategy is a set of rules for evaluating $p(a_1, \ldots, a_n)$, an application of p to $a_1, \ldots, a_n$.
- There are several different evaluation strategies.
- A programming language employs one or more evaluation strategies.
- A programming language may also be able to simulate evaluation strategies that is does not directly support.
- The three main types of evaluation strategies are:
  1. Call by value.
  2. Call by reference.
  3. Call by name.

Call by Value

- The most common evaluation strategy is call by value.
- Call by value works as follows on $p(a_1, \ldots, a_n)$:
  1. The arguments $a_1, \ldots, a_n$ are evaluated resulting in values $v_1, \ldots, v_n$.
  2. The values of the parameters $x_1, \ldots, x_n$ of p are set to the values $v_1, \ldots, v_n$.
- Call by value is used by both OCaml and C.
- In OCaml, step 2 is done by binding the parameters to the values.
- In C, step 2 is done by copying the values to new memory locations and then binding the parameters to these memory locations.
  - This is a costly operation if the values occupy a large amount of space.
- In OCaml and C, call by value is relaxed for boolean expressions and conditions.

Call by Reference

- The evaluation strategy call by reference works as follows:
  1. The arguments $a_1, \ldots, a_n$ are evaluated resulting in values $v_1, \ldots, v_n$.
  2. The parameters $x_1, \ldots, x_n$ of p are bound to references for the values $v_1, \ldots, v_n$.
- Call by reference is more space- and time-efficient than C-style call by value, but widens the access to the values.
- Call by reference is not directly supported in OCaml or C.
- In functional programming languages like OCaml, call by reference is used internally — so a parameter bound to a mutable value v behaves as if it were bound to a reference for v.
- Call by reference can be simulated in OCaml by using references and in C by using pointers.

Call by Name

- The evaluation strategy call by name works as follows:
  1. The arguments $a_1, \ldots, a_n$ are not evaluated.
  2. The arguments $a_1, \ldots, a_n$ are directly substituted for the parameters $x_1, \ldots, x_n$ in the body of p.
- Call by name is used to implement:
  - Lazy evaluation (or delayed evaluation).
  - Macro expansion (for example, as in C).

Records in C

- Records are called structures in C.
- Structure type declaration:

```
typedef struct {
  [const] t1 field-name1;
    ⋮
  [const] tn field-namen;
} struct-type;
```

  where `const` is an optional type qualifier that makes the field immutable.

- Constructor:

```
struct-type struct-name = {expr1, ..., exprn};
```

- Selector:

```
struct-name.field-name;
```

- Mutator:

```
struct-name.field-name = expr;
```

Arrays in C.

- Constructor:

  ```
  int a[5];
  int b[5] = {10,20,30,40,50};
  ```

  create arrays of type `int[5]` of length 5.

- Selector:

  `a[`index`]`

  where index is $0, \ldots, 4$.

- Mutator:

  `a[`index`] =` expr`;`.

  where index is $0, \ldots, 4$.

- Note: An array in C is not a variable: it can not be directly modified by assignment:
  - `a = b;` gives an error.

Pointers and Arrays

- Arrays are implemented in C like constant pointers that point to a fixed amount of space.
- Suppose

```
int a[5];
int * ip;
ip = a;
```

- Then

```
ip == &a[0]
ip + 3 == &a[3]
*(ip + 3) == a[3]
ip[3] == a[3]
```

- Since an array is accessed via a pointer, it is possible to access memory outside of the array (called a buffer overflow).
- Buffer overflows are the cause of many insidious bugs and dangerous security breaches.

Strings in C

- A string in C is a array of elements of type `char` that end with the null character `\'0'`.
- Constructors:

  "$c_1 c_2 \cdots c_n$"

  $\{c_1, c_2, \ldots, c_n,$ `\'0'`$\}$

- A variable of type `char` array can hold a string.

```
char x[] = "abc";
```

- The C library with header `<string.h>` contains various string processing functions.