Complex Types

Programming Languages

William Killian

Millersville University

Outline

- Strings (again?)
- Arrays
- Associative Arrays
- Lists
- Memory Representations

Strings

Definition

- *Descriptor* - additional information required to use an instance of a type during program execution

What information does a string need to have?

String Descriptors

Compile-Time

- The string itself
  - May or may not have an empty character (null) at the end
- Address
  - where it is located in memory
- Length
  - length of the string

Run-Time

- A buffer of characters
  - where the first “current length” characters denote the current string
- Address
  - where it is located in memory
- Current Length
  - number of characters in the current representation
- Maximum Length
  - length of the buffer

Strings – Design Questions

- Should we be able to modify strings?
- Should we be able to compare strings to characters?
- Should we be able to resize strings?
- What **<u>operations</u>** do we want to support?

Arrays

The Array Type

- **Homogeneous** aggregate of data elements
- Individual elements are identified by **position**, relative to the first element

Array Indexing

- Indexing (or subscripting) is mapping from indices to elements
- Syntax:
  - Can use square brackets e.g. arr[10]
  - Can use parentheses e.g. a(10)

Array Storage Categories

- **Static**
  - Subscript ranges are known before runtime
  - Storage allocation (on stack) is known before runtime
- **Fixed stack-dynamic**
  - Subscript ranges are known before runtime
  - Storage allocation (on stack) is done at declaration time
- **Fixed heap-dynamic**
  - Subscript ranges are known before runtime
  - Storage allocation (on heap) is done at declaration time
- **Heap dynamic**
  - Subscript ranges change change during runtime
  - Storage allocation (on heap) can change during runtime

Multi-Dimensional Arrays

- Ability to define an array that spans more than one dimension (e.g. a 2-D grid or 3-D volume)

Design Decisions:

- Indexing order vs. storage order
  - row-major vs column
- Syntax for accessing elements
  - `arr[0][1]` vs `arr(0, 1)`
- Allow for "jagged" arrays
  - inner dimensions need not be the same size
- Allow for "slices" of arrays (also called a view)
  - extract a sub-piece `a[1:-1][1:-1]` (ignores edges)

Compile-Time Descriptors

Single-Dimensional Array

- Element Type
- Index Type
- Index Lower Bound
- Index Upper Bound
- **Address**

Multi-Dimensional Array

- Element Type
- Index Type
- Number of Dimensions
- Index 1 Lower Bound
- Index 1 Upper Bound
- …
- Index N Lower Bound
- Index N Upper Bound
- **Address**

Array Initialization

- We may want to be able to initialize elements of an array when we declare it!

C, C++, Java, C# example

```
int list [4] = {4, 5, 7, 83}
```

Character strings in C

```
char name [] = ”freddie”;
```

Arrays of strings in C

```
char* names [] = {”Bob”, ”Jake”, ”Joe”};
```

Java initialization of String objects

```
String[] names = {”Bob”, ”Jake”, ”Joe”};
```

Array Design Decisions

- Storage?
  - Static (C/C++), Fixed stack-dynamic (C/C++), Fixed heap-dynamic (Java, C++ with **new**), Heap Dynamic (Python)
- Heterogeneous?
  - Elements do not need to all be of the same type
  - Supported in Perl, Python, JavaScript, Ruby
- Multi-dimensional Shape?
  - Square
  - Upper-Triangular (Linear Algebra Applications)
  - Jagged

Associative Arrays

Associative Array

- **Unordered** collection of data elements that are **indexed by** an equal number of values called **keys**
- User-defined keys must be stored
- Design issues:
  - What is the form of references to elements?
  - Is the size static or dynamic?
- Built-in type in Perl, Python, JavaScript, Ruby

Associative Arrays in Python/Ruby

```
# called dict in python
data = {"a" : 1}
data["b"] = [1, 2, 3]
data["c"] = "cool"

# called Hash in Ruby
data = Hash["a" => 1]
data["b"] = [1, 2, 3]
data["c"] = 'cool'
```

Lists

Not the kind you're thinking of

List Types (in Functional Languages)

- Lists are defined as a **Cons**

- **Cons** is like a “node” containing two parts:
  - CAR – the value stored at the node
    - Often refer to the car as first or head
  - CDR – refer to another **Cons**
    - Often refer to the cdr as rest or tail

List Types

in Functional Languages

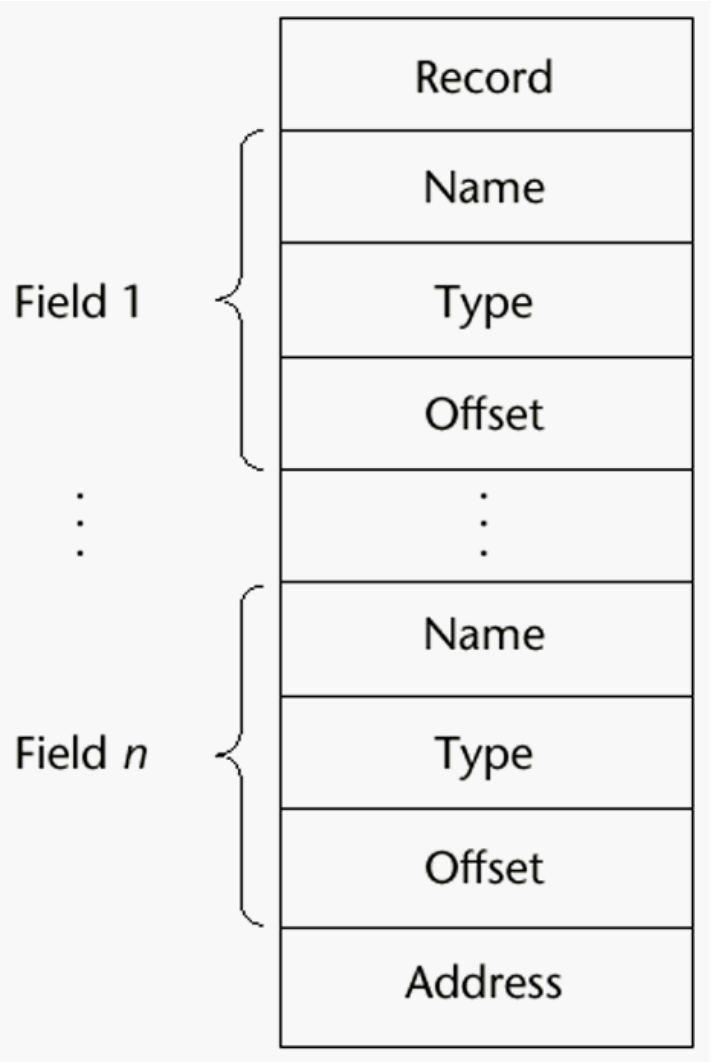
```
;; Lisp
(CAR (1 2 3))
(CDR (1 2 3))
(CONS 1 (2 3))
(CONS 1 (CONS 2 (CONS 3 NIL)))

(* OCaml *)
List.hd [1;2;3]
List.tl [1;2;3]
1::[2;3]
1::2::3::[]
```

Memory Representation of Aggregate Types

Tuples/Records

Note: All fields can have different types/sizes



Arrays

- Arrays are stored **contiguously** in memory

0	1	2	3	4	5	6	7	8	9

- No extra space between elements
- Cover more in a Computer Architecture class

Lists

- View lists as being “nodes” in memory