CS 603: Programming Language Organization

Lecture 6

Spring 2004

Department of Computer Science

University of Alabama

Joel Jones

Outline

- Questions
- Impcore Interpreter
- Reading for next time

Abstract Syntax Trees

- A representation of source code that separates the internal representation from the external representation, by removing extraneous details
- Example from Impcore:

```
Toplevel =  EXP      (Exp)
         |  DEFINE (Name, Namelist, Exp)
         |  VAL      (Name, Exp)
         |  USE      (Name)
```

```
Exp =  LITERAL (Value)
    |  VAR       (Name)
    |  SET       (Name, Exp)
    |  IF        (Exp, Exp, Exp)
    |  WHILE     (Exp, Exp)
    |  BEGIN     (Explist)
    |  APPLY     (Name, Explist)
```

Interpreter—noweb

- Implemented in C, using *noweb* literate programming system
- Conventions of *noweb*:
  - Chunks contain code and references to other chunks
  - Names are italicized and in angle brackets <*name*> and also contain page number
  - Definition is show with ≡, continuation with +≡
  - Pointers in right margin, use also (33b)

Interpreter—Impcore

- `typedef` used to avoid writing `struct` explicitly, doesn't contain "`*`", so pointers are explicit
- Names of functions begin lower case, except automatically generated ones
- Appendix A covers uninteresting parts, lexing and parsing

Interpreter—Impcore: Interfaces

- `ast` – Abstract syntax
- `env` – Environments
- `err` – error reporting
- `eval` – The evaluator—core of the language semantics
- `interp` – the real-eval-print loop
- `list` – routines for constructing and using lists
- `print` – an extensible version of `printf`
- `reader` – Reading abstract syntax from various inputs

Impcore Interpreter: ASTs

- Type of abstract syntax tree is a *sum type*, also known as a *discriminated-union* type
  - Not directly supported in C

Pair Up: • So how are they represented?

- Use tag and *undiscriminated union*

Pair Up: • Write structure definition for top-level and be prepared to explain.

```
Toplevel = EXP (Exp*)
         | VAL (Name *name, Exp *exp)
         | DEFINE (Name *name, Userfun userfun)
         | USE (Name*)
```

Impcore Interpreter: Environments

- Environments map names to values (or types), therefore interpreter needs to represent names, values, and environments
- Names are an abstract data type, `Name`, which support comparison and conversion to and from `char*`
  - `nametostr` converts a `Name` to `char*`
  - `strtoname` converts a `char*` to a `Name` and acts as a constructor

Impcore Interpreter: Environments (cont.)

- Uses one C type for `Value` and another for `Fun`
- Constructors
  - `mkValenv`
  - `mkFunenv`
- Accessors
  - `fetchval, isvalfound, bindval`
  - `fetchfun, isfunbound, bindfun`