

COS 301 Programming Languages
Fall 2009
Project Assignment #3 Due Tuesday Nov 3

Typing, Scope and Binding

Write 7-to-10 page paper that addresses the following topics for your selected language:

- Types available **in the language** beyond basic data types such as ints, floats, etc. This might include arrays, vectors, structures, strings, pointers, objects and classes. You should include higher-level structures such as hashes, stacks, etc. **if they are part of the language definition rather than part of a standard or other library.**
- Summary of standard libraries or classes that extend the type system.
- Discuss the units or levels of scope and the nature and type (run-time or compile-time) of name bindings within the different levels of scope.

Remember that the goal is to summarize and compare, not to provide reference manuals. Continue developing the annotated bibliography of your references. As discussed in class, your bibliography may include references not cited in this particular paper.

Programming Assignment #3

Write a lexer or tokenizer and recursive descent parser (or recognizer) for the query language developed in class. For output show (a) the token stream from the lexer, with values if applicable, and (b) show the call tree for the parser/recognizer by including output code into each function that recognizes a non-terminal. A recursive descent recognizer is similar to a recursive descent parser, but it does not construct an abstract syntax tree. For each non-terminal in the syntactic grammar there is a Boolean-valued function that returns true a production rule was successfully recognized; false otherwise.

For example, the Assignment function from the text's CLite parser could be rewritten as:

```
private Boolean assignment () {
    // Assignment --> Identifier = Expression ;
    Boolean result = Variable(match(TokenType.Identifier))
        && match(TokenType.Assign)
        && expression()
        && match(TokenType.Semicolon) ;
    System.out.println ("Recognized Assignment");
    return result;
}
```

For output your program should show the input query, the final result of the recognizer (True or False, Valid or Invalid), the token stream (at least up the point where query recognition failed) and values associated with tokens – for example, a word token would have a word associated with it. The query

melon* -watermelon apple grape (berry OR berries) "wash carefully"

corresponds to the following token stream

```
[prefixterm, melon*] [exclusion, watermelon] [word, apple] [word,
grape] [lparen, ] [word, berry] [OR, ] [word, berries] [rparen, ]
[phrase, "wash carefully"]
```

Where tokens are represented by [token, value] pairs.

Submit a program listing, sample output for the 5 valid queries listed below well as two additional valid queries and two invalid queries.

Query Grammar - syntactic rules

```
orterm -> andterm { OR andterm }
andterm -> term { AND term }
term -> pw | exclusion | inclusion | prefixterm | ( query )
prefixterm -> word*
inclusion -> +pw
exclusion -> -pw
pw -> phrase | wordseq
wordseq -> word { word }
```

Query Grammar - lexical rules

```
word -> nothyphen {char}
char -> a..zA..Z0..9@$%&-
nothyphen -> a..zA..Z0..9@$%&
phrase -> "string"
string -> any character except "
```

What are the tokens in this grammar? They are all of the symbols recognized at the lexical level plus any other terminal symbols used in the grammar – in this case only the left and right parentheses.

Example Queries

```
cats dogs ponies
"domestic pets" parakeet -snakes +gerbil
(cow OR goat) AND (hen OR rooster)
pup* "rabies vaccination" +booster
horse* pony AND (riding saddles stirrups)
```