Lecture 20: Parser Combinators & Program Units

CSCI 131
Spring, 2011

Kim Bruce

Pointers

- Pointers have been lumped with the goto statement as a marvelous way to create impossible to understand programs
  - K & R, C Programming Language
- Problems
  - Dangling pointers -- leave pointer to recycled space
    - stack frame popped or recycled heap item
  - Dereference nil pointers or other illegal address
  - Unreachable garbage
  - in C: p+1 different from (int)p + 1

More Scala

We're ahead of syllabus!

Scala & Parsing

- Scala provide parser combinators
  - Operators that allow you to glue together parsers
  - "|" is alternative, "~" is concatenation
    - def factor = "(" ~ expr ~ ")" | numericLit
    - def multOp = ("*" | "/")
    - def term = (factor ~ multOp ~ term | factor)
    - def addOp = ("+" | "-")
    - def expr = (term ~ addOp ~ expr | term)

Adding Actions

- ^^ {...} represents action to take with result.
  - def term : Parser[Int] = (
    factor ~ "*" ~ term ^^ { case x ~ "*" ~ y => x * y } |
    factor ~ "/" ~ term ^^ { case x ~ "/" ~ y => x / y } |
    factor)
  - Type says result is an Int
  - If x is result of factor, "*" is just "*", y is result of term, then return x * y.
- See code for interpreter in ArithParser.scala

What's Wrong With Grammar?

- Right recursive -- how does that affect answer?
- We can fix it by using our grammar for arithmetic expressions.
- Introduces new operations
  - "*" for 0 or more repetitions -- gives list as result
  - a ~> b means recognize a then b, but then throw result of a away if at beginning of result
  - a <~ b means recognize a then b, but then throw results of b away if at end of result

See ArithParserBuild.scala

Program Units & Activation Records

Program Units

- Separate segments of code allowing separate declarations of variables
  - Ex.: procedures, functions, methods, blocks
  - During execution represented by unit instance
    - fixed code segment
    - activation record with "fixed" ways of accessing items

Activation Record Structure

- Return address
- Access info on parameters (*how?*)
- Space for local vbles
- *How get access to non-local variables?*

Invoking Function

- Make parameters available to callee
  - E.g., put on stack or in registers
- Save state of caller (registers, prog. counter)
- Ensure callee knows where to return
- Enter callee at first instruction

Returning from Function

- If function, leave result in accessible location
- Get return address and transfer execution
- Caller restores state