Intro to WLP4

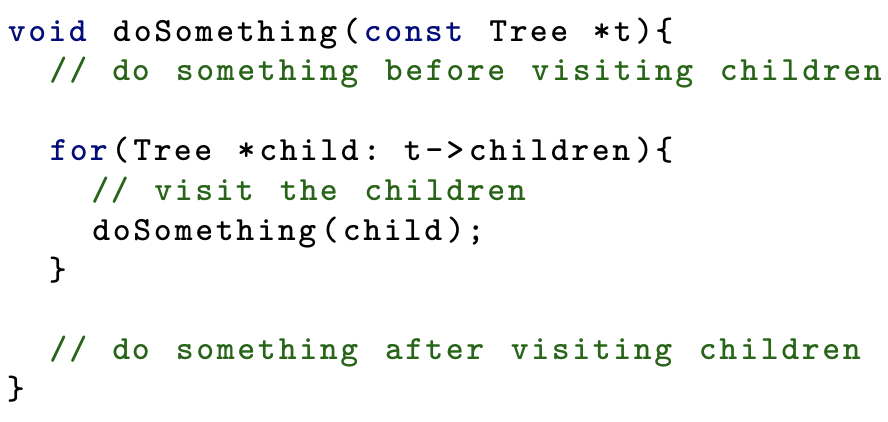
* WLP4 stands for Waterloo Language Plus Pointers Plus Procedures
  + WLP4 won’t pass through parsing stage unless appropriate wain function has been defined
    - wain procedure must always have 2 parameters and 2nd one must be an int
* procedure can have an arbitrary, but fixed, number of parameters
* parameters and variables can be int or int\* type
* supports while loops, if statements (must have else clause), and println statements
* return statements must appear only once in a procedure as the last statement
* supports heap memory through keywords new and delete

Context-Sensitive Analyses in WLP4

* not all language rules can be enforced by a CFG
  + e.g. type checking, variable declaration before use, and scoping (i.e. variable is defined in correct scope and there’s only one variable with the same name in the same scope)
* context-sensitive languages (CSLs) are more powerful than CFLs but aren’t very useful in formalism
* we use more practical method of analysis rather than parsing to deal with CSLs
  + i.e. use a code-based solution to traverse the parse tree to gather information and enforce rules that require context-sensitive information
* to traverse a parse tree, below is a simple code template in C++:

Logo

Description automatically generated



* parse tree created by parser is often called a concrete syntax tree
  + this parse tree is passed through tree transformation stage (i.e. tree is pruned by removing useless nodes like those needed to ensure grammar is unambiguous and to satisfy the requirements of a specific parsing algorithm) before applying context-sensitive analyses
  + transformed tree is called an abstract syntax tree (AST)
* two types of identifiers in WLP4 programs: variable and procedure name
* for variables, check there’s no duplicate variables within any procedure and all variables that are ever used have already been declared
  + use symbol table for variables
* e.g.

Text

Description automatically generated

* + code is legal because although variable x is defined twice in the program, it’s defined only once within each procedure
  + separate symbol table is needed for each procedure to keep track of variable names
  + can map each procedure name to its symbol table:



* + - ideally use better datatype than string for storing variable information
    - key is procedure name and symbol table is value
      * key in sub-map is variable name and value is its type
* e.g.

Text

Description automatically generated

* + program is not valid because x is not defined in procedure wain
  + when detecting duplicates, must use variable symbol table for that specific procedure
* e.g.

Text

Description automatically generated

* + variables are fine because there’s only one x defined in each scope
  + program is not valid because there are 2 procedures with the same name f
* e.g.

Text

Description automatically generated with medium confidence

* + program is valid because it’s legal to have variables that have the same name as any procedure
  + if a variable is declared in a procedure p, all occurrences of x within p refer to variable x even if a procedure named x has been declared as well

Text

Description automatically generated with medium confidence

* + - the above code should not compile
* to implement context-sensitive rules for WLP4 using tree traversals:
  + begin traversal at root of parse tree and determine when we have encountered a tree node that rep a procedure
    - tree node that has one of the below rules rep a new procedure

Text

Description automatically generated

* + check for duplicate procedures at this point
    - if master map of all symbol tables already contains mapping with same procedure name, we’ve discovered duplicate procedure and can generate appropriate error
  + if procedure is not duplicate, create new symbol table for this procedure and store it in master map
  + continue with traversal inside procedure
    - use global variable to store the currently active symbol table
  + within procedures, variables are either declared in parameter or at start of body
  + to find variable declarations, only need to look for parse tree nodes where the rule is dcl -> type ID
    - to populate symbol table, retrieve lexeme for ID token (child 2) and actual type from type token (child 1)
    - if name is already in table, give error
* all variables must be declared at top of procedure so only one pass is needed to generate entire symbol table for a procedure
  + i.e. by the time the first traversal reaches the statements child of the following 2 rules, symbol table is complete

Text

Description automatically generated

* to check for declaration of variables before use, traverse statements subtree and return expr while looking for these rules that use variables: factor -> ID and lvalue -> ID
  + when node is encountered, retrieve lexeme for ID token (child 1) and check that an entry with this name exists in the symbol table for the current procedure
* procedures must be declared before use so we can’t collect list of all procedures that are declared in a program and then check that each procedure call matches the name of a declared procedure
* analysis must validate names of procedures being called during creation of top-level symbol table
  + look for tree node with these rules: factor -> ID LPAREN RPAREN and factor -> ID LPAREN arglist RPAREN
  + check name of procedure against the current procedures already declared and in the symbol table
* when new procedure declaration is discovered, should also store procedure’s signature (i.e. information needed to correctly call it)
  + in WLP4, it’s a sequence rep the types for parameters
    - can use a vector of strings to rep
  + when calling a procedure, must check correct number and types of parameters are passed
* best way to keep track of everything is to have a top-level map with procedure name to Procedure objects
  + create a class Procedure that stores the signature and specific symbol table for that procedure
* e.g.

Text

Description automatically generated

* + overall data structure for top-level symbol table would look like:

Diagram

Description automatically generated

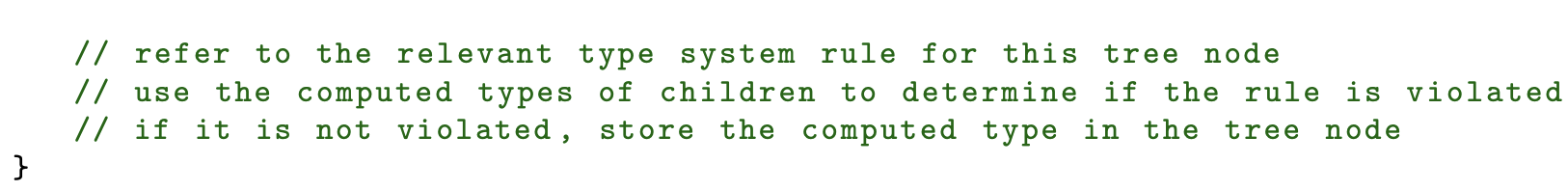
* + - (…) is a map
    - <…> is a pair
    - […] is a vector

Catching Type Errors

* to determine type correctness, we need a tree traversal and infer the types of child nodes at any given node
  + look at grammar rule for current node and determine if the type rule has been validated

A picture containing text

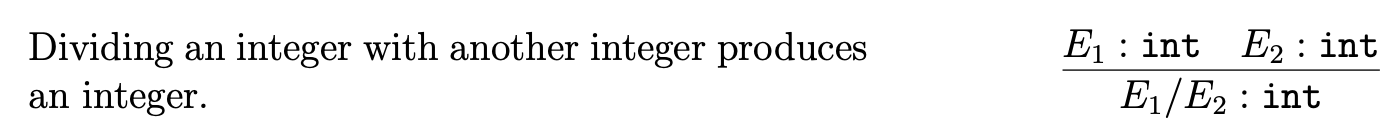
Description automatically generated



* type system for WLP4:
  + premises for a conclusion go above a horizontal bar
  + conclusion is below the bar
  + empty premises mean the conclusion always holds

Table

Description automatically generated



Table

Description automatically generated

* + an absence of a type rule implies that it’s not allowed
* while statements don’t produce values and don’t have an inferred type, we still need to check type correctness of statements
* statement is well-typed if its components are well-typed

Text, letter

Description automatically generated

Text, letter

Description automatically generated

* to type check the correctness of control flow statements (if and while), must have way to check correctness of conditional expression that’s evaluated
  + typically, these expressions would evaluate to a Boolean but because WLP4 doesn’t have a bool type, we say a test is well-typed if the operands for the comparison are of the same type

Graphical user interface

Description automatically generated with low confidence

Text, letter

Description automatically generated