CIIC 4030/ICOM 3046 Programming Languages
Assignment #3

Add the following semantics (context-sensitive) specifications to the parsing code in assignment #2

Each sequence derived from defdef is a **declaration** of a **procedure**. Each sequence derived from vardef is a **declaration** of a **variable**. The **name** of the procedure or variable is the lexeme corresponding to the child ID of the defdef or vardef. Names are case sensitive; for example, FOO and foo are distinct names.

Each procedure is associated with exactly one **scope**. The **declaration subtrees** of a procedure are the sequences derived from the children parmsopt, vardefsopt, and defdefsopt of the declaration of *p*. A procedure *q* whose declaration is in the sequences derived from the declaration subtrees of *p* is said to be **nested** in *p*. The scope of a procedure *p* **contains** all variables and procedures whose declarations are in the sequences derived from the declaration subtrees of *p*, **except** for those variables and procedures whose declarations are in the subsequences derived from the declaration subtrees of some other procedure *q*that is nested in *p*. A scope **declares** the names of all of the variables and procedures that it contains.

A scope **must** not contain two or more declarations with the same name.

A name is said to be **used** in procedure *q* when it appears as an ID in a sequence derived from the child expras of the declaration of *q*. When a name *n* is used in procedure *q*, a **declaring scope** of the use of *n* is a scope that declares *n*, and is either the scope of *q* or of some other procedure *p* in which *q* is nested. Every use of a name **must** have some declaring scope. The **closest declaring scope** of a use of *n* is the scope of some procedure *p* whose scope is a declaring scope of the use of *n*, and such that no other procedure *q* nested in *p* has a scope that is a declaring scope of the use of *n*. Each use of a name **denotes** the variable or procedure with that name declared in the closest declaring scope of the use.

A **type** is a sequence derived from type. A type is called a **procedure type** if it is derived from LPAREN typesopt RPAREN ARROW type. The sequence of types derived from the child typesopt of the procedure type is called the **parameter types**. The child type of the procedure type is called the **return type**. The **type** of a variable is the sequence derived from the child type of the vardef that declares the variable. The **type** of a procedure *p* is a procedure type whose parameter types are the types of the variables declared in the sequence derived from the child parmsopt of the declaration of *p*, and whose return type is the sequence derived from the child type of the declaration of *p*.

- **ID**
  - The type of an ID that is a use of a name is the type of the variable or procedure denoted by the use of the name. The context-free grammar ensures that whenever an ID is a use of a name, the ID is the child of either an expra or a factor.

- **factor**
  - The type of a factor deriving an ID is the type of that ID.
  - The type of a factor deriving NUM is Int.
  - The type of a factor deriving LPAREN expr RPAREN is the type of the expr.
  - When a factor derives factor LPAREN argsopt RPAREN, the type of the child factor **must** be a procedure type whose parameter types are the same as the types of the exprs occurring as children of the args derived from the argsopt. The type of a factor deriving factor LPAREN argsopt RPAREN is the return type of the child factor.
- **term**
  - The type of a term that derives a factor is the type of the factor.
  - When a term derives term STAR factor, term SLASH factor, or term PCT factor, the types of the child term and factor **must** be Int, and the type of the resulting term is Int.
- **expr**
  - The type of an expr that derives a term is the type of the term.
  - When an expr derives expr PLUS term, or expr MINUS term, the types of the child expr and term **must** be Int, and the type of the resulting expr is Int.
  - When an expr derives IF LPAREN test RPAREN LBRACE expras RBRACE ELSE LBRACE expras RBRACE, the type of the two exprs derived from test **must** be Int, the types of the two child expras**must** be the same, and the type of the resulting expr is the type of the two child expras.
- **expra**
  - When an expra derives ID BECOMES expr, the type of the ID and the expr **must** be the same, and the ID **must** denote a variable, not a procedure. An expra always derives either expr or ID BECOMES expr; in both cases, the type of the expra is the type of the child expr.
- **expras**
  - The type of an expras that derives expra SEMI expras is the type of the child expras.
  - The type of an expras that derives expra is the type of the expra.
- **defdef**
  - A defdef derives DEF ID LPAREN parmsopt RPAREN COLON type BECOMES LBRACE vardefsopt defdefsopt expras RBRACE. The type of the child expras **must** be the same as the child type.

The first procedure declared in a program is the **main** procedure. The type of the main procedure **must** be (Int, Int) => Int.