Logo, company name

Description automatically generated



**COMSATS University Islamabad (CUI)**

**Lab terminal**

**Submitted to:**

**Submitted By: ANIS MAJID**

**Reg No: FA21-BCS-024**

**Course: COMPILER CONSTRUCTION**

**Date: 3 JANUARY 2025**

**Question No 05)**

**Key Areas of Semantic Analysis (Such as they are):**

1. **Symbol Table Management:** The core of the semantic analysis lies in how it manages the symbol table.
   * **Declaration Check:** During variable and function declarations, the compiler checks if an identifier has already been declared in the same scope (sym\_lookup function). This prevents name collisions (multiple declarations of the same variable) within the current scope.
   * **Scope Management:** Each function has its own local scope (tracked by the locals array, the new\_scope function which reset the local\_no and param\_no counters). This ensures that variables declared inside functions don't conflict with global variables or variables in other functions.
   * **Symbol Lookup:** Before using an identifier in an expression, the compiler checks if it has been declared. If it hasn't, the compiler reports an error. This is primarily the job of sym\_lookup. It searches both globals and locals symbol tables.
   * **Function declaration/implementation check:** If the compiler encounters a function implementation without a previous declaration, it generates an error.
2. **Lvalue Analysis:** The compiler keeps track of "lvalues" during expression parsing to detect assignment and modifiable object operations.
   * **lvalue flag:** A global lvalue flag is set when parsing certain expressions such as a variable identifier or array index operations.
   * **Error Detection:** The needs\_lvalue function checks if the lvalue flag is set before performing an assignment (=) or a pre/post increment/decrement operation (++ and --). If the flag is not set, it means that the assignment or increment/decrement is being attempted on a non-modifiable expression (e.g., a constant), and an error is produced.
   * **Address Generation:** When the lvalue is true the compiler uses lea instruction to get the address of a variable or an array element instead of the content using mov.
3. **Function Call Argument Count:** When parsing a function call, the compiler pushes parameters to the stack. The object() function counts the number of arguments and will use the total number to adjust the stack after the call, using add esp, %d. There is no checking of the number of parameter declared with the number of arguments passed to the functions.

**Functions Involved in Semantic Analysis:**

* **sym\_lookup(char\*\* table, int table\_size, char\\* look):** Used to look up an identifier in a given symbol table. Used for both globals and locals.
* **new\_global(char\* ident):** Adds a global variable or function identifier to the global symbol table (globals).
* **new\_fn(char\* ident):** Adds a function identifier to the global symbol table (globals) and sets the is\_fn flag.
* **new\_local(char\* ident):** Adds a local variable identifier to the local symbol table (locals) and sets its stack offset.
* **new\_param(char\* ident):** Adds a parameter identifier to the local symbol table (locals). Also computes the stack offset for parameter based on the cdecl convention.
* **new\_scope():** Resets the local symbol table and parameters to start a new function scope.
* **needs\_lvalue(char\* msg):** Checks if the lvalue flag is set, prints an error if not, and resets the flag.
* **decl(int kind):** Handles variable and function declarations and interactions with the symbol table.
* **object():** The call operation (expr ("," expr)\*)? from object calls expr recursivelly which performs semantic checks for expressions.

**Limitations:**

* **No Type Checking:** There are no explicit types, so expressions using different types of variables are considered valid.
* **No Array Bound Checking:** No runtime check is done, and accessing out of bound of an array will result in undefined behavior.
* **Minimal Error Reporting:** The error messages are relatively simple, and the compiler doesn't attempt complex error recovery. It may produce cascading errors.
* **Limited Scope:** Global and local scopes are managed with simple arrays without a full scope hierarchy.
* **No Function Signature Checking:** The compiler doesn't check parameter types or the number of arguments during function calls, potentially leading to runtime errors.
* **No variable initialization order tracking:** The compiler doesn't enforce variable initialization order or detect uninitialized variables.