Name:	**Mubashir Ali**
Reg #	**FA21-BCS-009**
Course Name:	**Compiler Construction**
Submitted To:	**Syed Bilal Haider Bukhari**
Date:	**03 Jan 2025**

Lab Terminal

Question 5: Explain the functions which performs the semantic analysis in mini compiler.
Answer: In a mini compiler, the **semantic analysis** phase checks for logical errors that cannot be detected during syntax analysis. It ensures that the program is meaningful and conforms to the language's rules. The semantic analysis phase typically checks for things like type errors, undeclared variables, and type mismatches in expressions.

Functions Performing Semantic Analysis in Your Mini Compiler

From the provided code structure, the key functions related to **semantic analysis** include:

1. **checksym**:
    - **Purpose:** This function checks whether a variable has already been declared.
    - **How it works:** When a new variable is encountered, checksym verifies if it already exists in the symbol table. If the variable is not declared previously, a new entry is created in the symbol table.
    - **Semantic Check:** This ensures that variables are declared before being used and that multiple declarations of the same variable are prevented.
2. **addfn0, addfn1, addfn2**:
    - **Purpose:** These functions add variable information to the symbol table based on the data type.
    - **How they work:** Depending on the type of the variable (int, float, char), these functions update the symbol table with the variable's name, data type, and initial value.
    - **Semantic Check:** These functions enforce proper variable declaration by checking if the variable already exists in the symbol table and ensuring no redeclaration is allowed for the same variable within the same scope.
3. **checkvalid**:
    - **Purpose:** This function checks whether a variable exists in the symbol table and whether it has been properly declared before use.
    - **How it works:** For every variable used in expressions or assignments, checkvalid is called to ensure the variable is present in the symbol table and has a valid type.
    - **Semantic Check:** It ensures that the program does not use undeclared variables, which would lead to runtime errors.

4. **Type Checking in Expressions (within exp, E, and T rules)**:
    - **Purpose:** These rules check that the operands in expressions match in type.
    - **How it works:** For example, in arithmetic operations, the types of operands are checked to ensure compatibility (e.g., trying to add an integer and a float).
    - **Semantic Check:** These functions prevent type mismatches, such as trying to assign a string to an integer variable or performing operations between incompatible types.
5. **printsymtable**:
    - **Purpose:** Although this function's primary task is to display the symbol table, it can also be useful for semantic analysis.
    - **How it works:** It prints the symbol table, allowing the compiler developer to inspect the types, names, and values of all declared variables in the current program.
    - **Semantic Check:** This function can be used as a diagnostic tool to verify the integrity of the semantic analysis phase by ensuring the correct information is stored in the symbol table.

Overall Flow of Semantic Analysis

- **Declaration Check:** Ensures variables are declared before use.
- **Type Checking:** Ensures that operations are performed between compatible types.
- **Symbol Table Management:** Ensures that variable names are unique within their scope, and that values assigned to variables match their declared types.

Example

Consider the following example:

```
int a;

a = 10;

float b;

b = a + 5.5;

// Error: Type mismatch (int + float)
```

- **checksym** will verify if a and b are declared.
- **addfn0** will add a and b to the symbol table.
- When parsing the expression a + 5.5, the **semantic analysis** will catch the **type mismatch** error because a is an int and 5.5 is a float.