Compiler Construction / Compiler Design
Assignment Stage 3

Stage3: Completing all Phases of Compiler (Deadline 24th April)

You will be implementing following modules of compiler

- **Type Extraction and Checking**
- **Semantic Analyzer**
- **Code Generation**

1. **Type Extractor and Checker:**
    - **Propose appropriate type checking rules for your language.**
    - **Try to have 10 reasonable rules.**
    - **You can look into sample rules given in Appendix A for C type language for your reference.**
    - **The type checker verifies the type of an expression appearing at the right hand side of the assignment statement and checks if it matches with that of the identifier on the left hand side.**
    - **Type Coercion, Conversion (Narrowing or Broadening) and equivalence concepts learnt in PPL course should be applied here where ever is applicable.**

2. **Semantic Analyzer:**

    - **This module verifies the semantics of the code.**
    - **Identify 10 valid semantic rules which suits your language design.**
    - **Sample semantic rules for C type language is given in Appendix B. You can use this as reference and propose suitable semantic rules for your language.**
    - **Each valid semantic rule proposed and implementation will carry appropriate credits during evaluation.**

4. Code Generator : This module may take as input the abstract syntax tree (AST) as intermediate representation. The function generates 8086 assembly code.
The task of code generation is made simpler with the following constraints on the given source code.

- **The _main is the only function in the input source code.**
- **The input source code does not have function definitions.**
- **All other features of the language such as iterative and conditional constructs, all statement constructs, all types of variables (including records), expression evaluation etc. exist in the source code.**

Only trivial optimization such as avoiding redundant code, appropriate register usage etc are needed while the detailed code optimization techniques are not expected to be implemented.

Appendix A

The sample type checking rules are given below for reference:

i. **The type of an identifier is the type appearing while declaring the variable.**

ii. **The type of TK_NUM is integer.**

iii. **The type of TK_RNUM is real.**

iv. **The type of an identifier of a record type expanded with a dot followed by the field name is same as that of the field.**

v. **The type of a record type is a tuple (type of field1, type of field 2, type of field3,.......and so on). Example : Consider the following type definition**

```
record #marks
type real : maths;
type int: rollnum;
type real: physics;
type real: chemistry;
endrecord;
```

The type of the above record is (real, integer, real, real) (say T).

vi. **The type of a simple expression (say E) of the form expression(say E1) <operator> Expression(say E2)**

   a. **is integer, if both expressions are of type integer and the operator is arithmetic operator.**
   b. **is real, if both the expressions are of type real and the operator is arithmetic operator.**
   c. **is boolean, if both expressions are of type integer and the operator is relational operator.**
   d. **is boolean, if both expressions are of type real and the operator is relational.**
   e. **is boolean, if both expressions are of type boolean and the operator is logical.**
   f. **is record type T if both E1 and E2 are of the same record type T.**

The type of the expression is ERROR, if the above rules do not derive the type of E appropriately.

The type of an identifier or an expression is computed by traversing the abstract syntax tree.

Appendix B

Sample semantic rules :

i. **An identifier cannot be declared multiple times in the same scope.**
ii. **An identifier must be declared before its use.**
iii. **An identifier declared globally cannot be declared anywhere else in function definitions.**
iv. **The types and the number of parameters returned by a function must be the same as that of the parameters used in invoking the function.**
v. **The parameters being returned by a function must be assigned a value. If a parameter does not get a value assigned within the function definition, it should be reported as an error.**
vi. **The function that does not return any value, must be invoked appropriately.**
vii. **Function input parameters passed while invoking it should be of the same type as those used in the function definition. Number of input parameters must be same as that of those used in the function definition.**
viii. **An if statement must have the expression of boolean type.**
ix. **Function overloading is not allowed.**
x. **The function cannot be invoked recursively.**
xi. **An identifier used beyond its scope must be viewed as undefined**
xii. **A record type definition is visible anywhere in the program.**
xiii. **The right hand side expression of an assignment statement must be of the same type as that of the left hand side identifier.**
xiv. **A function definition for a function being used (say F1) by another (say F2) must precede the definition of the function using it(i.e. F2).**