CS536 HW7

Lingfei Li

Question 1

typedef -> TYPEDEF STRUCT ID ID SEMICOLON
type -> ID

Question 2

(a)
Add a new class "TypedefSym" that is a subclass of SemSym. It has only one field: type (of type string), representing the type being defined. Then, during the process of name analysis, do the following:

1. Check if T is a built-in type. If no, go to next step. If yes, pass the analysis.
2. Check if T is defined by a typedef by finding the symbol T globally in the SymTable, and check if it's an instance of TypedefSym. If no, go to next step. If yes, pass the analysis.
3. Check if T is in the form *struct xxx*. If yes, find the symbol xxx globally in the SymTable and check if it is an instance of StructSym. If no, fail the analysis. If yes, pass the analysis.

If the name analysis doesn't fail, create a new TypedefSym and insert it to the SymTable.

(b)
Add an additional check: After we find the symbol named "xxx" from symbol table, check if the type of the symbol is TypedefSym. It yes, it should be rejected as "undeclared variable".

(c)
Firstly check if the name "xxx" has been occupied.
Then, check if T is a pre-defined type, a struct type, or an alias. If it's an alias (not int, bool, string, void or struct), search for symbol named "T" globally in the symbol table and check if it is an instance of TypedefSym. If no, reject the declaration as "bad type". If yes, replace type alias "T" with the real type and redo the name analysis.

(d)
The "real" type of the name, not the alias.

Symbol table:

Key	**Value**
MonthDayYear	StructSym { varDeclList: int month, int day, int year }
date	TypedefSym { type: “struct MonthDayYear” }
today	VarSym { type: “struct MonthDayYear” }
dollars	TypedefSym { type: “int” }
salary	VarSym { type: “int” }
moreDollars	TypedefSym { type: “int” }
md	VarSym { type: ”int” }
d	VarSym { type: “int” }