```
1  //################## Compiler Design (2) ##################
2  //#################### Type Checking ######################
3  // ######## Input: expression var= funcName(par) #########
4  //################ Output: maching result #################
5  //############### Author: Ahmed A. Alsamman ###############
6  using System;
7  using System.Collections.Generic;
8
9  namespace TypeCheckingFunc
10 {
11     public struct Exp
12     {
13         public string lhs;
14         public string fName;
15         public string par;
16         public Exp(string lhs,  string fName, string par)
17         {
18             this.lhs = lhs;
19             this.fName = fName;
20             this.par = par;
21         }
22     }
23     public struct Token
24     {
25         public string name;
26         public string type;
27         public string parType;
28         public Token(string name, string type, string parType)
29         {
30             this.name = name;
31             this.type = type;
32             this.parType = parType;
33         }
34     }
35
36     public class SymbolTable
37     {
38         public List<Token> token = new List<Token>();
39         public int Lookup(string tok)
40         {
41             for (int i = 0; i < token.Count; i++)
42                 if (token[i].name == tok)
43                     return i;
44             return -1;
45         }
46
47         public void Insert(string name, string type, string parType)
48         {
49             if (Lookup(name) == -1) // variable not found
50                 token.Add(new Token(name, type, parType));
51         }
52     }
```

```
53
54      class Program
55      {
56          static string lhsType = null;
57          static string funcType = null;
58          static string parType = null;
59          static string parErr = null;
60          static SymbolTable symbTab = new SymbolTable();
61
62          static void Main(string[] args)
63          {
64              //insert variables
65              symbTab.Insert("a", "int", null);
66              symbTab.Insert("b", "double", null);
67
68              //insert functions
69              symbTab.Insert("fact", "int", "int");
70              symbTab.Insert("sqrt", "double", "int");
71
72              while (true)
73              {
74                  parErr = null;
75                  Console.Write("LHS: ");
76                  string lhs = Console.ReadLine();
77                  Console.Write("Function name: ");
78                  string fName = Console.ReadLine();
79                  Console.Write("Paramater: ");
80                  string par = Console.ReadLine();
81
82                  if (lhs == "" || fName == "" || par=="")
83                      continue;
84                  Console.WriteLine("Expression: {0}={1}({2})", lhs, fName, par);
85                  Exp exp = new Exp(lhs, fName, par);
86                  if (Parse(exp) == "parsed")
87                      MatchTypes(exp);
88              }
89          }
90
91          private static bool validID(string var)
92          {
93              if (var[0] == '_' || Char.IsLetter(var[0]))
94              {
95                  for (int i = 1; i < var.Length; i++)
96                      if (!Char.IsLetter(var[i]) && !Char.IsDigit(var[i]) || var[0] ➥
                          == '_')
97                          return false;
98              }
99              else
100                 return false;
101             return true;
102         }
103
```

```
104
105         private static string Parse(Exp exp)
106         {
107             if (validID(exp.lhs))
108                 symbTab.Insert(exp.lhs, null, null);
109
110             else
111             {
112                 Console.WriteLine("Parsing Error: LHS must be a variable");
113                 return null;
114             }
115            if (validID(exp.fName))
116              symbTab.Insert(exp.fName, null, null);
117              else
118              {
119                 Console.WriteLine("Parsing Error: RHS must be a function Name");
120                 return null;
121              }
122             if (validID(exp.par))
123                 symbTab.Insert(exp.par, null, null);
124             else
125             {
126                 try
127                 {
128                     int.Parse(exp.par);
129                     symbTab.Insert(exp.par, "int", null);
130
131                 }
132                 catch
133                 {
134                     try
135                     {
136                         double.Parse(exp.par);
137                         symbTab.Insert(exp.par, "double", null);
138                     }
139                     catch
140                     {
141                         Console.WriteLine("Parsing Error: Invalid parameter");
142                         return null;
143                     }
144                 }
145             }
146             return "parsed";
147         }
148
149
150
151
152
153
154
155
```

```
156
157         private static void MatchTypes(Exp exp)
158         {
159             lhsType = symbTab.token[symbTab.Lookup(exp.lhs)].type;
160             funcType = symbTab.token[symbTab.Lookup(exp.fName)].type;
161             parType = symbTab.token[symbTab.Lookup(exp.par)].type;
162             string storedParType = symbTab.token[symbTab.Lookup
                  (exp.fName)].parType;
163
164             if( symbTab.token[symbTab.Lookup(exp.lhs)].parType!=null)
165             { // LHS is a function name)
166                 Console.WriteLine("Error: LHS must be a variable");
167                 return;
168             }
169             if (storedParType == null && funcType != null)
170             { // RHS is an existing variable (it has a type and it has no
                  parameter)
171                 Console.WriteLine("Error: RHS must be a function name");
172                 return;
173             }
174
175             if (parType!=null &&  parType!= storedParType)
176             {
177                 parErr = "Parameter Error: Expected parameter of type " +
                      storedParType ;
178             }
179             if (parType == null)
180             {
181                 parErr = "Parameter Error: Undefined variable "  + exp.par ;
182             }
183
184             if (lhsType == funcType && lhsType != null)
185                 Console.WriteLine("Type Mached");
186
187             else if (lhsType == null || funcType == null)
188             {
189                 if (lhsType == null)
190                     Console.WriteLine("Undefined variable {0} at LHS", exp.lhs);
191                 if (funcType == null && exp.lhs != exp.fName)
192                     Console.WriteLine("Undefined variable {0} at RHS",
                          exp.fName);
193                 return;
194             }
195
196             else Console.WriteLine("Type Mismach. Expected a value of type {0} ",
                    lhsType);
197             if (parErr != null)
198                 Console.WriteLine(parErr);
199         }
200     }
201 }
202
```