Exercise 10: Language Extension: Enumeration Types

What changes would be necessary in the compiler if there were enumeration types in MicroJava? The declaration

```
enum Color {RED, BLUE, GREEN}
```

would define an enumeration type with three enumeration constants represented internally by the values 0, 1, and 2. Using an attributed grammar, show how the declaration of enumeration types and the access to enumeration constants (e.g., Color.RED) are handled. Changes are required in the scanner, the parser, the symbol table and the code generator.

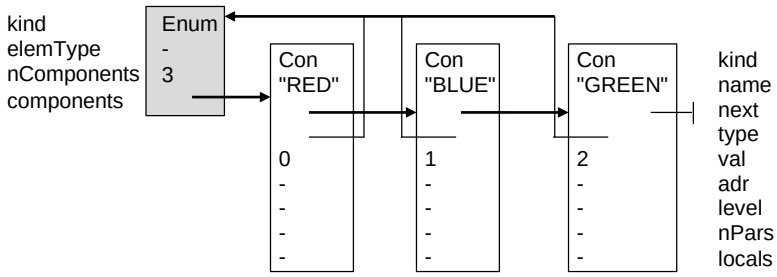
Solution

First of all, we have to declare a new keyword enum with a corresponding token code enum_ in the scanner and in the parser.

Furthermore, we need to add a kind Enum to the class Struct. The structure node of an enumeration type must refer to a list of enumeration constants. To do this, we rename the field nFields to nComponents and fields to components

```
public class Struct {
   static final int None  = 0, Int   = 1, Char  = 2, Arr   = 3, Class = 4, Enum = 5;
   public int    kind;        // None, Int, Char, Arr, Class, Enum
   public Struct elemType;    // Arr: element type
   public int    nComponents; // Class, Enum: number of fields or constants
   public Obj    components;  // Class, Enum: fields or constants
   ...
}
```

The enumeration type Color is then represented in the symbol table as follows:



The grammar of MicroJava programs has to be modified:

```
Program = "program" ident {ConstDecl | ClassDecl | EnumDecl | | VarDecl} '{' {MethodDecl} '}'.
```

Finally, the processing of EnumDecl looks like this:

```
EnumDecl
=  "enum" ident          (. Struct type = new Struct(Struct.Enum);
                          Tab.insert(Obj.Type, t.val, type);
                          Tab.openScope();
                          int val = 0; .)
   "{"
      ident              (. Obj obj = Tab.insert(Obj.Con, t.val, type);
                          obj.val = val; val++; .)
      { "," ident        (. Obj obj = Tab.insert(Obj.Con, t.val, type);
                          obj.val = val; val++; .)
      }
   "}"                   (. type.nComponents = val;
                          type.components = Tab.curScope.locals;
                          Tab.closeScope(); .) .
```

The access to enumeration constants (e.g., Color.RED) happens in Designator:

```
Designator <↑x>
= ident                 (. Operand x = new Operand(Tab.find(t.val)); .)
  { "." ident           (. if (x.kind == Operand.Type) {  // enumeration constant
                             if (x.type.kind != Struct.Enum) error("enumeration type expected");
                             Obj con = Tab.findComponent(t.val, x.type);
                             x.kind = Operand.Con;
                             x.val = con.val;
                           } else if (x.kind == Operand.Local || x.kind == Operand.Static ||
                                      x.kind == Operand.Elem || x.kind == Operand.Fld) {  // field access
                             Code.load(x);
                             if (x.type.kind == Struct.Class) {
                               Obj comp = Tab.findComponent(t.val, x.type);
                               x.adr = comp.adr;
                               x.type = comp.type;
                             } else error("dereferenced object is not a class");
                             x.kind = Operand.Fld;
                           } .)
  | ...
  } .
```

A few more changes are required. Operand must have a new operand kind Type and its constructor has to be changed so that it can also process objects of the kind Obj.Type.

```
public class Operand {
  public static final int  Con = 0, Local = 1, Static = 2, Stack = 3, Fld = 4, Elem = 5, Meth = 6, Type = 7, Cond = 8;
  ...
  public Operand (Obj obj) {
    type = obj.type; val = obj.val; adr = obj.adr; kind = Stack; // default
    switch (obj.kind) {
      ...
      case Obj.Type:
        kind = Type; break;
      default:
        Parser.error("wrong kind of identifier"); break;
    }
  }
```

We also rename Tab.findField() to Tab.findComponent(), because the lookup is now not only for fields, but also for enumeration constants. However, the functionality of this method remains the same.

Nothing needs to be changed for type checking. In an assignment

```
Color c = Color.RED;
```

the type of the left and right-hand side is Color, which is expressed by the same structure node.