Marius Mitea Assignment 3
5508878 "Semantic Information"

I pass the semantic information around through my predictive recursive descent parser with the aid of a reference to a SemanticInfo class. It holds the name of the production I'm parsing, its type (where applicable, program for example doesn't have a type), the parameters (again, where applicable, variables don't have parameters), array indices (held as an array of strings, since when I'm parsing through an error I may read the wrong information in and I'll deal with that later on).

The parameters are held as an array of another structure holding the name, type and an array of strings for the array indices where applicable. I initially chose a binary search tree but that would lose the order information, so I went with this instead.

Once all the semantic information is gathered in the right nodes, I fill in the information in the symbol table, a huge structure holding a binary search tree keyed by the symbol name and relevant information for each symbol (the kind of the symbol, and another sorted binary search tree that holds properties of the symbol defined on a per-kind basis for speed; it holds properties like child symbol tables for functions and classes, parameters for functions, types for functions and variables etc). Each symbol table also holds a pointer to the "parent" symbol table, while the global symbol table has null as its parent. Symbol tables have a few methods:

- Find – given a symbol name and an option to recurse upwards into the symbol table structures, it tries to find the symbol information and return its structure, or return null if it fails.
- Add – given a symbol name, it adds the specified symbol to the binary search tree contained in the table. The tree's unique key property protects the data from symbol redefinition, though additional error detection is done before calling it to try to recover from such semantic errors.
- CreateChild – creates another symbol table and sets the parent property to the current one, with the intention of using it in nested symbol tables, like for functions and the such.

The parser holds 2 references to symbol tables, one a global symbol table that gets created when the parsing begins, and one the "current" symbol table that gets updated recursively as we traverse the code.

Extra care has been taken to recover from as many errors as possible from a semantic point of view. Every time a symbol definition is encountered, the current scope is checked to see if it already exists, and if it does, the information about it is updated without assuming absolutely anything about it to keep semantic integrity. For example:

```
SymbolTable.Information info;
if ((info = CurrentScope.Find(si.Name, false)) != null)
{
        SemanticErrorSymbolAlreadyDefined(si.Name);
        error = true;

        if (info.Properties.ContainsKey("function_symtable"))
                if (info.Properties["function_symtable"] != null)
                        CurrentScope = info.Properties["function_symtable"] as SymbolTable;
                else
                        info.Properties["function_symtable"] = CurrentScope =
CurrentScope.CreateChild(si.Name);
        else
        {
                info.Properties.Add(si.Name, CurrentScope.CreateChild(si.Name));
                CurrentScope = info.Properties[si.Name] as SymbolTable;
        }
}
else
```

```
{
	info = CurrentScope.Add(si.Name);
	info.Kind = SymbolTable.Information.EKind.Function;
	info.Properties.Add("function_parameters", si.Parameters);
	info.Properties.Add("function_type", si.Type);
	info.Properties.Add("function_symtable", CurrentScope.CreateChild(si.Name));
	CurrentScope = info.Properties["function_symtable"] as SymbolTable;
}
```

And a final note on error reporting, the usual error formal is <type of error>:<line>:error message. And the parser errors (non-semantic errors) usually just dump the token type that they were expecting and what they got, so while for example the end-of-file errors look a bit weird, they are done so for ease of use and uniformity, and all the required information is in there.