Compilers course project 2022 - Ossi Lehtonen

This is a project for course Compilers 2022 in University of Helsinki.
The program is written in C#.
The source code can be found from the folder 'src/MiniPLInterpreter'.

MiniPL Language

Please read MiniPL.pdf for detailed description of the language. This section is meant for answers required in the project setting (project_description.pdf)

Token patterns as regular expressions

Assignment = \:\=
Operators = \+|-|*|\/|< |=|&|!
Parentheses = \(|\)
String = \”[^\”]*\”
Int = \d*
Idents_and_keywords = [A-Za-z]+(_|\d)*[A-Za-z]*

Token = String | Int | Idents_and_keywords | \; | .. | : | Assignment | Operators | parentheses

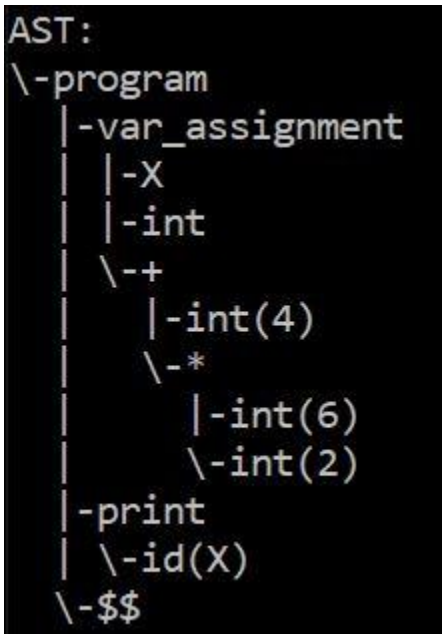
A modified context-free grammar suitable for recursive-descent parsing

TODO

ASTs

The programs are from MiniPL.pdf

Program 1 ast:



The syntax of the tree is as follows. “Program” is the root node. Node “var_assignment” is the first child of “program” and “print” the second. “X”, “Int” and “+” are the children of “var_assignment” and so on…

Program 2 ast:

See picture doc/asts/ast_2.JPEG

Program 3 ast:

See picture doc/asts/ast_3.JPEG

MiniPL Interpreter

In this section I try to briefly explain the structure of the program (from folder src).
The actual source code is in src/MiniPLInterpreted. The MiniPLUnitTests folder can be dismissed, as it was just used when developing the interpreter.

Structure

The structure of the interpreter program is quite straightforward.

```
static void Main(string[] args)
{
    MiniPLFileNameInput miniPLInput = new MiniPLFileNameInput();

    while (true)
    {
        string[] program = miniPLInput.readMiniPLProgram();

        MiniPLScanner scanner = new MiniPLScanner();
        MiniPLParser parser = new MiniPLParser();
        Interpreter interpreter = new Interpreter(scanner, parser);

        interpreter.interpret(program);
    }
}
```

The Interpreter class gets MiniPLScanner and MiniPLParser in its constructor. MiniPLScanner and MiniPLParser implement Interfaces which the Interpreter class expects them to implement.

Interpreter class' "interpret"-function gets a list of strings as input. This list of strings is a MiniPL program, each string representing a row of a program.

The "interpret"-function calls Scanner's "scan"-function, which returns the tokens from the string list. Then, the Interpreter calls Parser's "parse"-function, which returns an AST. The interpreter then prints out the AST. Finally the interpreter calls it's private function "interpret", which gets the AST as input.

Implementations of the Scanner and Parser follow the guidance obtained from the course. In a "production" interpreter, at least the parser would require some refactoring as the code is not clean at all, but it works.

I will not go into more details of the implementation, except when it comes to error handling.

Error handling

Running and Usage instructions

The program is written in C# and it has been tested with Windows 10 and Ubuntu 20.04 WSL, so it should work in the university environment.

The source code can be found from folder 'src/MiniPLInterpreter'

Running

If Visual Studio is installed, one should be able to run the project easily with Visual Studio by opening the MiniPLInterpreter.sln solution in the src -folder.

If dotnet command line tools are installed, navigate to src/MiniPLInterpreter folder and run commands:

dotnet restore
dotnet run

Usage

When the program is running, it asks the user to provide the filename of a MiniPLProgram. It reads the MiniPLPrograms from folder src/MiniPLInterpreter/miniPL-programs. The files should be in .txt format. There are three example programs in the folder named 1.txt, 2.txt and 3.txt. These are the example programs from project_description.pdf.

Example:

```
Give the name of MiniPL program in folder (miniPL-programs):
1.txt
```

By pressing enter, the interpreting begins.

```
Give the name of MiniPL program in folder (miniPL-programs):
1.txt
AST:
\-program
  |-var_assignment
  | |-X
  | |-int
  | \-+
  |   |-int(4)
  |   \-*
  |     |-int(6)
  |     \-int(2)
  |-print
  | \-id(X)
  \-$$
Program:
------------
16
------------
```

The program first prints out the AST of the MiniPLProgram. The actual interpreted program is printed between two “--------”. In this case the MiniPL program prints out 16.

Work Hours

Please see file Workhours.MD