BIE-PJP Semestral Work - Mila Compiler

DAMIAN MALARCZYK

29 JANUARY 2017

1. Implemented basic features
2. Realization and sample files changes
3. Additional features
4. Standard library
5. Command line tool
6. Compilation, LLVM version

1. Implemented basic features

All basic requested features have been implemented namely:

- Main function
- Printing numbers
- Reading numbers
- Global variables
- Expressions
- Assignment
- Number constants in C like literals (Decimal, Octal and Hexadecimal bases)
- If statement
- While loop
- For loop
- Exit (return) expression
- Nested blocks
- Statically allocated arrays with any index interval
- Procedures
- Functions
- Local variables
- Parameters of functions and procedures
- Recursion and indirect recursion
- String literals

2. Realization and sample files changes

Compiler has dedicated syntax for reference passing which is a `&` sign after a variable name, so this operator has been added to function calls like `readln`.

Few `;` has been added to some `end` keywords.

Write function for printing string literals was replaced by `echo` function.

All changes made in the source files contain comments `/* */` indicating the changes made.

3. Additional features

- Keyword `until` which can be used in for loop statements, in opposite to `to` keyword it does not include last element (unclosed range)
- Keyword `downuntil`
- Single expression `for` loops do not require `begin end` syntax
- `for` loops do not require global variables
- `Switch` statement with `case` constant values as well as ranges with same syntax as with `for` loops, also the `default` statement. All cases are checked so that no same value is included within more than one case.
- External functions
- Passing pointers to functions - I didn't implement ptr unwrapping syntax but it could be useful with external functions
- Function attributes
  - Non returning functions (`@NoReturn`)
  - Forced inline (`@Inline`)
  - No inline (`@NoInline`)
- Basic optimizations
- Protected array access - each attempt to access or set array value will be preceded with safety check whether the given index is within the array's index interval. Of course those are absent when optimizations are enabled.
- Comments - `/* */`
- Abstract syntax tree printing

Syntax for all the mentioned features will be shown in the included additional example source file together with comments indicating.

4. Standard library

There's a set of predefined functions, namely:

- write(i: integer) - prints number to standard output
- writeln(i: integer) - like write but adds new line symbol
- readln(i: integer*) - takes pointer to integer variable, reads value from standard input
- inc(i: integer*) - increment
- dec(i: integer*) - decrement
- print(i: char*) - prints string literal to standard output
- echo(i: char*) - like print but appends new line character
- newLine() - prints new line characters to standard output
- _exit(i: integer) `@NoReturn` - ends executions with i exit code
- _exitMessage(i: integer, msg: char*) `@NoReturn` - like _exit but uses echo with the msg argument before end of execution

The standard library is embedded in the source code in hex format, it uses C++ implementation and external functions feature, it is automatically linked with every created executable. If the source code contains some `extern` declared functions only object file will be generated and standard library source will be exported so that it is possible to link it later. I will also include it in the send files.

5. Command line tool

Product of the source code is a command line tool, the first argument has to be a `path to a file or directory`.

Supported flags:

- -dir - indicates that given path leads to a directory, then all files in the directory with extension `p` will be compiled. `-dumpIR` and `-dumpAST` are ignored when this flag is used. Optimizations can be enabled.
- -O - used to enable optimizations.
- -dumpIR - after successful compilation produced IR output is printed to the standard output.
- -dumpAST - the generated AST is printed to the standard output. With this flag code is not compiled, if used `-dumpIR` and `-O` are ignored.

Usage example:

```
./Mila /path/to/my/source.p -dumpIR -O
```

6. Compilation, LLVM version

Together with the source I included simple `makefile`. Dependently on platform and location of all LLVM tools, linker and headers search path may need to be adjusted.

The project has been tested with:

```
clang version 3.9.1 (tags/RELEASE_391/final)
```

On macOS 10.12.3