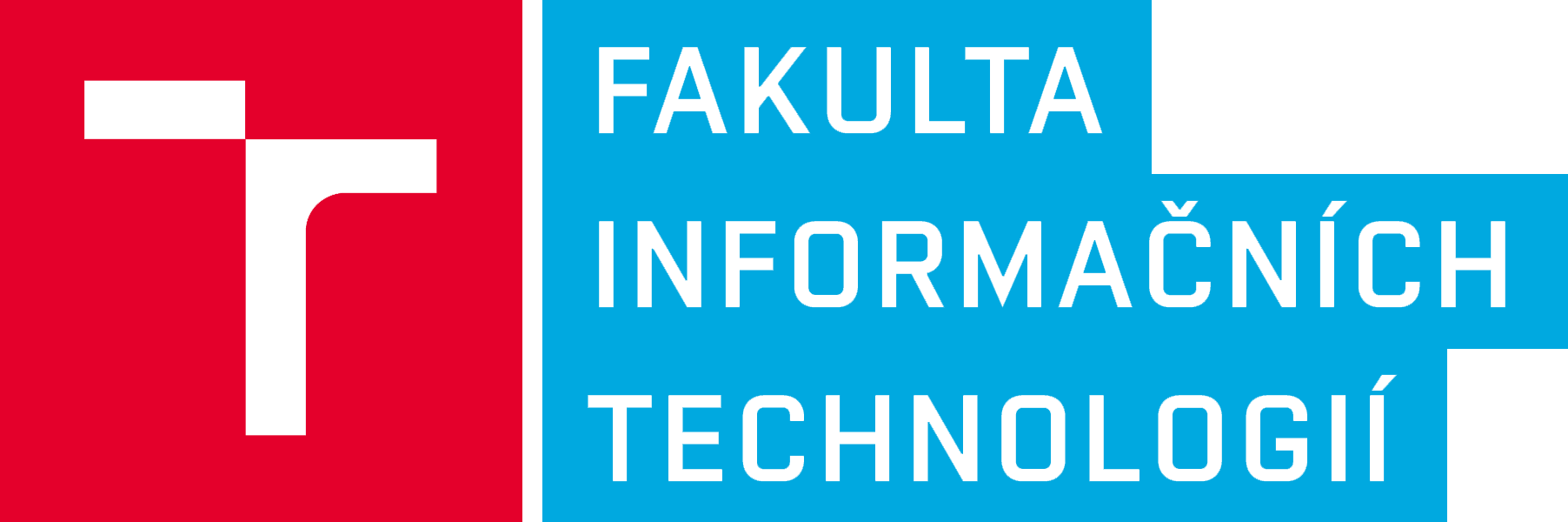
Faculty of information technology

Brno University of Technology



Documentation to the project for the IFJ and IAL courses

**Implementation of the compiler of the IFJ18 programming language**

Team 037, variant I.

Gabriel Quirschfeld (xquirs00)

Marek Varga (xvarga14)

Marek Imrich (ximric01)

Michal Plšek (xplsek03)

**Contents**

1. [**Teamwork**](#_Teamwork)
   1. **Work division**
   2. **Communication**
   3. **Use of IDEs**
2. [**Implementation**](#_Implementation)
   1. **Scanner**
   2. **Parser**
   3. **Code generation**
   4. **Testing**
3. [**Conclusion**](#_Conclusion)
4. [**Sources**](#_Sources)
5. [**Attachments**](#_Attachments)

# **Teamwork**

* 1. **Work division**

Our team consisted of four members. We grouped ourselves together mostly randomly. The members are as follows: Gabriel Quirschfeld (leader), Marek Varga, Michal Plšek and Marek Imrich. The work division was discussed and accepted by all members.

Gabriel Quirschfeld → scanner, symtable, debugging, testing, documentation

Marek Varga → syntactic analysis, precedence analysis, semantic analysis, debugging, testing

Michal Plšek → code generation, debugging, testing

Marek Imrich → scanner, testing, documentation

* 1. **Communication**

We mainly used Facebook as a communication device. We also had team meetings mostly located in Ventana Café or school library.

* 1. **Use of IDEs**

We worked with the JetBrains CLion IDE for coding and debug but some of us also used vim. PHP script was used for testing purposes and valgrind controlled the work with memory allocation and deallocation. Git was used for version control. While working on separate parts at the same time we used branches that we later merged. Also, we used CMake for

# **Implementation**

* 1. **Scanner**

The scanner was implemented as a finite-state automaton (att. 1). It has two final states, one being the error state and the other is a success final state. It also utilizes the epsilon routes, mostly to reach the success final state. The token is a structure that encapsulates its size (allocated memory), type and the content. The type of a token is a structure on its own. It states which state the token reaches after it is fully read. The content is simply the string that is read. We use one global token that is declared in the header file of the scanner.

* 1. **Parser**

The whole compiler works in two transits. During the first transit all variables and functions are loaded in the symtables. Functions and global variables are loaded into the global symtable and local variables and function parameter are loaded into an array of local symtables. Symtable is constructed as BST – binary search tree. Our implementation was inspired by homework we did for IAL course but there also were some adjustments made. Every node in BST contains an ID which is hashed according to our hash function, hash id of the function it belongs to or 0 it is a function, number of function parameters if it is a function, type, name, information wheter it was defined and used in code generation yet and pointers to left and right descendant.

During the second transit all syntax and semantic actions are made. Parsing is made by predictive analytics with the help of LL-parsing table (att.2) and precedence table is used for parsing expressions. Firstly ‘<start>’ is pushed into the predictive stack and based on given token it is rewritten according to LL-parsing table to appropriate rule defined in LL-grammar (att.3). When an expression needs to be evaluated it is done so with the help of operator precedence table and expression is written to AST – abstract syntax tree as a result.

If there are only constants occurring in the expression it is also checked for semantic errors. If there is a variable occurring in the expression it is only partially checked. By partially checked, definition of variable is checked and type compatibility is not checked.

* 1. **Code generation**
  2. **Testing**

A testing script in php language was written for our testing purposes. Also each one of us wrote test files for each part of the compilation process. In the end we ended up with over 200 tests which was sufficient enough for discovering and fixing major bugs in lexical, syntax and semantic parts but not enough to discover and fix all the bugs. Where we could do a little better was testing code generating mainly because of the lack of time.

# **Conclusion**

Work division was not a problem for us however we had some difficulties explaining to each other the work we did separately. Especially when the code generating was supposed to be completed into the syntax. We feel that if we had had more regular meeting sessions during the development this communication would have been easier. We also found out that pushing our demands through the internet was less effective rather than meeting in person.

# **Sources**

# **Attachments**

**Attachment 1: Finite-state automaton**

**Obrázok, na ktorom je text, mapa

Popis sa automaticky vygeneroval**

**Attachment 2: LL grammar**

1. <start> → <function> <st-list>
2. <function> → ‘def’ <function-head> <st-list> <function-tail> <function>
3. <function> → ε
4. <function-head> → <function-id> ‘(’ <par> ‘)’ ‘EOL’
5. <function-tail> → ‘end’ ‘EOL’
6. <par> → <id> <next-par>
7. <par> → ε
8. <next-par> → ‘,’ <par>
9. <next-par> → ε
10. <st-list> → <stat> ‘EOL’ <st-list>
11. <st-list> → ‘EOL’<st-list>
12. <st-list> → ε
13. <stat> → <id> <eval>
14. <eval> → ε
15. <eval> → ‘=’ <assign>
16. <assign> → <expr>
17. <assign> → <function-id> <f-params>
18. <stat> → ‘if’ <expr> ‘then’ ‘EOL’ <st-list> ‘else’ ‘EOL’ <st-list> ‘end’
19. <stat> → ‘while’ <expr> ‘do’ ‘EOL’ <st-list> ‘end’
20. <stat> → ‘print’ <print-expr>
21. <print-expr> → ‘(‘ <print-expr>‘)’
22. <print-expr> → <id> <next-print-expr>
23. <print-expr> → ε
24. <next-print-expr> → ‘,’ <print-expr>
25. <stat> → <function-id> <f-params>
26. <f-params> → ‘(‘ <f-param> ‘)’
27. <f-params> → <id> <next-f-param>
28. <f-params> → ε
29. <next-f-params> → ‘,’ <f-params>
30. <next-f-params> → ε

**Attachment 3: LL parsing table**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **<start>** | **<function>** | **<function-head>** | **<function-tail>** | **<par>** | **<next-par>** | **<st-list>** | **<stat>** | **<eval>** | **<assign>** | **<print-expr>** | **<next-print-expr>** | **<f-params>** | **<next-f-params>** |
| **def** | 1 | 2 |  |  |  |  | 1 |  |  |  |  |  |  |  |
| **EOL** | 1 | 3 |  |  |  | 9 | 11 |  | 14 |  | 23 | 23 | 28 | 28 |
| **end** |  |  |  | 5 |  |  | 12 |  |  |  |  |  |  |  |
| **,** |  |  |  |  |  | 8 |  |  |  |  | 24 | 24 | 29 | 29 |
| **'=’** |  |  |  |  |  |  |  |  | 15 |  |  |  |  |  |
| **id** | 1 | 3 | 4 |  | 6 |  | 10 | 13 |  | 16 | 22 | 22 | 27 | 27 |
| **expr** |  |  |  |  |  |  |  |  |  | 16 | 22 | 22 | 27 | 27 |
| **(** |  |  |  |  |  |  |  |  |  | 16 | 21 |  | 26 |  |
| **)** |  |  |  |  | 7 | 7 |  |  |  |  | 23 | 23 | 28 | 30 |
| **if** | 1 | 3 |  |  |  |  | 10 | 18 |  |  |  |  |  |  |
| **then** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **else** |  |  |  |  |  |  | 12 |  |  |  |  |  |  |  |
| **while** | 1 | 3 |  |  |  |  | 10 | 19 |  |  |  |  |  |  |
| **do** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Function-id** | 1 | 3 | 4 |  |  |  | 10 | 25 |  | 17 |  |  |  |  |
| **$** |  | 3 |  |  |  | 9 | 12 |  | 14 |  | 23 | 23 | 28 | 28 |
| **print** |  |  |  |  |  |  | 10 | 20 |  | 17 |  |  |  |  |

**Attachment 4: Precedence table**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **+** | **-** | **\*** | **/** | **<** | **<=** | **>** | **>=** | **'==’** | **!=** | **(** | **)** | **I** | **$** |
| **+** | > | > | < | < | > | > | > | > | > | > | < | > | < | > |
| **-** | > | > | < | < | > | > | > | > | > | > | < | > | < | > |
| **\*** | > | > | > | > | > | > | > | > | > | > | < | > | < | > |
| **/** | > | > | > | > | > | > | > | > | > | > | < | > | < | > |
| **<** | < | < | < | < | '=’ | '=’ | '=’ | '=’ | > | > | < | > | < | > |
| **<=** | < | < | < | < | '=’ | '=’ | '=’ | '=’ | > | > | < | > | < | > |
| **>** | < | < | < | < | '=’ | '=’ | '=’ | '=’ | > | > | < | > | < | > |
| **>=** | < | < | < | < | '=’ | '=’ | '=’ | '=’ | > | > | < | > | < | > |
| **'==’** | < | < | < | < | < | < | < | < | '=’ | '=’ | < | > | < | > |
| **!=** | < | < | < | < | < | < | < | < | '=’ | '=’ | < | > | < | > |
| **(** | < | < | < | < | < | < | < | < | < | < | < | '=’ | < | E |
| **)** | > | > | > | > | > | > | > | > | > | > | E | > | E | > |
| **I** | > | > | > | > | > | > | > | > | > | > | E | > | E | > |
| **$** | < | < | < | < | < | < | < | < | < | < | < | E | < | E |