GP 2	Generative Programming	ST 17, Exercise 2

Deadline: 21.04.2017, 13:50

☐ **Gr. 1,** J. Karder, MSc.
☐ **Gr. 2,** P. Fleck, MSc.

Name _____ **Effort in h** _____

Points _____ **Lecturer** _____

1. JSON Validator (10 Points)

JSON (JavaScript Object Notation) is a popular format for data exchange between servers and (web) applications.

Implement a simple parser with Boost Spirit that is able to read a JSON file and verify its syntactical correctness. It should at least be able to parse the following snippet:

```
{
    "Image": {
        "Width": 800,
        "Height": 600,
        "Title": "View from 15th Floor",
        "Opacity": 0.5,
        "Thumbnail": {
            "Url": "http://www.example.com/image/481989943",
            "Height": 125,
            "Width": 100
        },
        "Location": {
            "Latitude": 37.7668,
            "Longitude": -122.3959
        },
        "Animated": false,
        "IDs": [116, 943, 234, 38793]
    }
}
```

Please note that the parser should be able to handle objects, members, pairs, arrays, elements and values. You do not need to take care of special characters as well as special number formats like the E notation.

You can find the JSON specification at http://json.org/ and the RFC at http://tools.ietf.org/html/rfc7159.

2. Mini-LOLCODE with Boost.Spirit (4 + 1 + 2 +3 + 2 + 2 Points)

LOLCODE is an esoteric programming language with a syntax resembling lolspeak (the language of the lolcats), e.g.:

```
HAI
CAN HAS STDIO?
VISIBLE "HAI WORLD!"
KTHXBYE
```

Implement a simple interpreter for LOLCODE using Boost Spirit. It should be able to parse LOLCODE source code, execute the code and print to the console:

- First, implement an interpreter that can handle the above example. The `VISIBLE` statement can take any expression that returns a value and prints it to the console. Note that by default a line break is added to the output. To permit line breaks, VISIBLE can be terminated with an exclamation mark. `CAN HAS` is used to include libraries. The parser should be able to handle includes, but no further action has to be performed.

- Single line comments in LOLCODE start with `BTW` and end at the end of a line.

- Multi line comments start with `OBTW` and end with `TLDR`.

- Variables are declared with the `I HAS A` statement followed by the name of the variable. To assign a value to a variable, the `R` keyword is used:

  ```
  I HAS A variable
  variable R 5.0
  ```

  LOLCODE automatically detects the data type of the variable. Implement at least the data type double (`NUMBAR`) and bool (`TROOF`).

- Implement at least 4 basic arithmetic operations, e.g.:

  ```
  SUM OF 4 AN 5
  ```

  It's sufficient for the interpreter to be able to handle numbers and nested mathematical operations. The result of an arithmetic operation can be assigned with `R` to a variable. If there is no assignment, it is saved in a temporary variable called `IT`.

- Implement at least 4 Boolean operations, e.g.:

  ```
  BOTH OF WIN AN FAIL
  ```

  The result of a Boolean operation can be again assigned to a variable or saved in `IT` if omitted.

For more information about the LOLCODE syntax you can have a look at the specification at https://github.com/justinmeza/lolcode-spec/blob/master/v1.2/lolcode-spec-v1.2.md.

1 JSON Validator

1.1 Lösungsidee

Der *JSON*-Validierer wurde in der *Header*-Datei *json.hpp* implementiert. Die Grammatik wurde gemäß der Spezifikation auf `http://json.org` implementiert.

1.1.1 Source

Listing 1: json.hpp

```
#ifndef _json_hpp_
#define _json_hpp_

#include "boost.hpp"
#include <map>
#include <string>

namespace json_parser {

  namespace qi = boost::spirit::qi;
  namespace phoenix = boost::phoenix;

  template<typename Iterator>
  struct json_grammar : qi::grammar<Iterator, qi::space_type> {
    json_grammar() : json_grammar::base_type(json) {

      json    =   object
              | array;

      object  =    qi::lit('{')
              >> -member
              >> qi::lit('}');

      array   =    qi::lit('[')
              >> -element
              >> qi::lit(']');

      member  =    pair
              >> *(qi::lit(',') >> pair);

      element =    value
              >> *(qi::lit(',') >> value);

      pair    =    literal
              >> qi::lit(':')
              >> value;

      value   =   literal
              | qi::double_
              | qi::true_
              | qi::false_
              | qi::lit("null")
              | object
              | array;

      literal = qi::lexeme['"' >> *(qi::char_ - '"') >> '"'];
    };

    qi::rule<Iterator, qi::space_type> json, object, array, element, value, pair, member;
    qi::rule<Iterator, std::string(), qi::space_type> literal;
  };
```

```
52 }
53
54 #endif _json_hpp_
```

Listing 2: test.json

```
1  {
2    "Image": {
3      "Width": 800,
4      "Height": 600,
5      "Title": "View from 15th Floor",
6      "Opacity": 0.5,
7      "Thumbnail": {
8        "Url": "http://www.example.com/image/481989943",
9        "Height": 125,
10       "Width": 100
11     },
12     "Location": {
13       "Latitude": 37.7668,
14       "Longitude": -122.3959
15     },
16     "Animated": false,
17     "IDs": [ 116, 943, 234, 38793 ]
18   }
19 }
```

Listing 3: json-test.hpp

```
1  #ifndef _json_test_hpp_
2  #define _json_test_hpp_
3
4  #include <iostream>
5  #include <string>
6  #include <fstream>
7
8  #include "json.hpp"
9
10 using namespace std;
11 namespace qi = boost::spirit::qi;
12
13 void test_json(string file) {
14   cout << "testing json parser with file " << file << endl;
15   std::ifstream fileStream(file);
16   std::string json((std::istreambuf_iterator<char>(fileStream)),
17     std::istreambuf_iterator<char>());
18
19   auto begin = json.begin();
20   auto end = json.end();
21   json_parser::json_grammar<decltype(begin)> grammar;
22   cout << "-------------------------------------------------------"
23     << endl
24     << "parsing ... "
25     << endl
26     << "-------------------------------------------------------"
27     << endl
28     << json
29     << endl
30     << "-------------------------------------------------------"
31     << endl;
32
```

```
33    bool success = qi::phrase_parse(begin, end, grammar, qi::space) && begin == end;
34
35    cout << "success = " << boolalpha << success
36      << endl
37      << "-------------------------------------------------------"
38      << endl;
39    fileStream.close();
40  }
41
42  #endif _json_test_hpp_
```

```
testing json parser with file ./test.json
---------------------------------------------------------
parsing ...
---------------------------------------------------------
{
  "Image": {
    "Width": 800,
    "Height": 600,
    "Title": "View from 15th Floor",
    "Opacity": 0.5,
    "Thumbnail": {
      "Url": "http://www.example.com/image/481989943",
      "Height": 125,
      "Width": 100
    },
    "Location": {
      "Latitude": 37.7668,
      "Longitude": -122.3959
    },
    "Animated": false,
    "IDs": [ 116, 943, 234, 38793 ]
  }
}
---------------------------------------------------------
success = true
---------------------------------------------------------
```

Abbildung 1: Test JSON Parser

1.2 Mini-LOLCODE with Boost.Spirit

Der Parser mit semantischen Aktionen für die Programmiersprache *LOLCODE* wurde in der *Header*-Datei *lolcode.hpp* implementiert. Es wurden die beiden Datentypen *Numeric* und *Bool* implementiert, wobei auch der Datentyp *Bool* als *Numeric* behandelt wird, also *0=FAIL* und *1=WIN*. Variablen werden in einem assoziativen *Container (map<string, double>)* gespeichert. Ein zweiter assoziativer *Container (map<string, bool>)* speichert die Information, ob die Variable ein *Numeric* Datentyp ist oder ob es wich um einen *Bool* Datentyp handelt.

boolsche und arithmetische Ausdrücke können verschachtelt werden, sowie Variablen in den Ausdrücken verwendet werden. Somit Ausdrücke wie:

- *BIGGR OF WIN AN SMALLR OF WIN AN FAIL*
- *BIGGR OF 10 AN QUOSHUNT OF 90 AN 10*
- *SUM OF 100 AN PRODUKT OF 50 AN num1*
- *BOTH SAEM bool1 AN DIFFRINT 100 AN num1*

Da auch boolsche Variablen als numerische behandelt werden, sind auch boolsche Variablen in arithmetischen Ausdrücken erlaubt, da in arithmetischen Ausdrücken der numerische Wert einer boolschen variable herangezogen werden.

Alle Konstanten wie z.B. *VISIBLE* wurden als Literale in der Grammatik definiert *(qi::lit("VISIBLE"))*.

1.2.1 Source

Listing 4: lolcode.hpp

```
#ifndef lolcode_h
#define lolcode_h

#include "boost.hpp"
#include <map>
#include <string>

#define IT_VAR "IT"

namespace lolcode {

  // Used namespaces
  namespace qi = boost::spirit::qi;
  namespace phoenix = boost::phoenix;

  // associatives containers for the variables
  std::map<std::string, bool> var_is_bool_dict;
  std::map<std::string, double> var_dict;

  // for registering a new variable
  struct new_var {
    void operator()(std::string const &var) const {
      var_dict[var] = 0.0;
      var_is_bool_dict[var] = false;

      std::cout << std::endl << "semantics-info: created new var '" << var << std::endl;
    }
  };

```

```
30  // for printing a variable
31  struct print_variable {
32    void operator()(std::string const &var) const {
33      if (var_dict.find(var) != var_dict.end()) {
34        if (var_is_bool_dict[var]) {
35          std::cout << ((var_dict[var] == 0.0) ? "FAIL" : "WIN") << std::endl;
36        }
37        else {
38          std::cout << var_dict[var] << std::endl;
39        }
40      }
41      else {
42        std::cout << std::endl << "semantics-info: Variable '" << var << "' not found";
43      }
44    }
45  };
46
47  // for getting a varaible value
48  double get_var(std::string const &var) {
49    double result = 0.0;
50
51    if (var_dict.find(var) != var_dict.end()) {
52      result = var_dict[var];
53    }
54
55    std::cout << std::endl << "semantics-info: resolved variable '" << var << "' to value: " <<
    ↪  result << std::endl;
56
57    return result;
58  }
59
60  // for setting a variable numeric value
61  void set_var(std::string const &var, double const &val) {
62    var_is_bool_dict[var] = false;
63    var_dict[var] = val;
64
65    std::cout << std::endl << "semantics-info: set numeric var '" << var << " to: " << val <<
    ↪  std::endl;
66  }
67
68  // for seting a bool variable value
69  void set_var_bool(std::string const &var, double const &val) {
70    var_is_bool_dict[var] = true;
71    var_dict[var] = (val != 1.0) ? 0.0 : 1.0;
72
73    std::cout << std::endl << "semantics-info: set boolean var '" << var << " to: " <<
    ↪  ((var_dict[var] == 0.0) ? "FAIL" : "WIN") << std::endl;
74  }
75
76  // for printing a plain bool result
77  void print_bool_result(double const &_1) {
78    std::cout << ((_1 == 1.0) ? "WIN" : "FAIL") << std::endl;
79  }
80
81  template<typename Iterator>
82  struct lolcode_grammar : qi::grammar<Iterator, qi::blank_type> {
83    lolcode_grammar() : lolcode_grammar::base_type(program) {
84
85      program =    qi::lit("HAI")
86              >> qi::eol
87              >> *stat
88              >> qi::lit("KTHXBYE")
89              >> *qi::eol;
```

```
90
91      stat =    -(
92               includeStat
93               | vardeclStat
94               | assignmentStat
95               | arithemticExpr[phoenix::bind(&set_var, (const char *)IT_VAR, qi::_1)]
96               | boolexpr[phoenix::bind(&set_var_bool, (const char *)IT_VAR, qi::_1)]
97               | visibleStat
98             )
99             >> -comment
100            >> qi::eol;
101
102     includeStat   =     qi::lit("CAN")
103                >> qi::lit("HAS")
104              >> include
105                    >> qi::lit('?');
106
107     vardeclStat   =     qi::string("I")
108                >> qi::lit("HAS")
109                >> qi::lit("A")
110                >> ident[new_var()];
111
112     assignmentStat =    ident[qi::_a = qi::_1]
113                    >> qi::lit("R")
114                    >> (
115                   arithemticExpr[phoenix::bind(&set_var, qi::_a, qi::_1)]
116                   | boolexpr[phoenix::bind(&set_var_bool, qi::_a, qi::_1)]
117                 );
118
119     additionStat     =    qi::lit("SUM")
120                >> qi::lit("OF")
121                >> arithExOrVar[qi::_val = qi::_1]
122                >> qi::lit("AN")
123                >> arithExOrVar[qi::_val += qi::_1];
124
125     substractionStat =    qi::lit("DIFF")
126                >> qi::lit("OF")
127                >> arithExOrVar[qi::_val = qi::_1]
128                >> qi::lit("AN")
129                >> arithExOrVar[qi::_val -= qi::_1];
130
131     productStat      =    qi::lit("PRODUKT")
132                >> qi::lit("OF")
133                >> arithExOrVar[qi::_val = qi::_1]
134                >> qi::lit("AN")
135                >> arithExOrVar[qi::_val *= qi::_1];
136
137     divisionStat     =    qi::lit("QUOSHUNT")
138                >> qi::lit("OF")
139                >> arithExOrVar[qi::_val = qi::_1]
140                >> qi::lit("AN")
141                >> arithExOrVar[qi::_val /= qi::_1];
142
143     sameStat    =    qi::lit("BOTH")
144             >> qi::lit("SAEM")
145             >> boolOrVar[qi::_val = qi::_1]
146             >> qi::lit("AN")
147               >> boolOrVar[qi::_val = (qi::_val == qi::_1)];
148
149     notsameStat =    qi::lit("DIFFRINT")
150             >> boolOrVar[qi::_val = qi::_1]
151             >> qi::lit("AN")
152             >> boolOrVar[qi::_val = qi::_val != qi::_1];
```

```
153
154         smallerStat =    qi::lit("SMALLR")
155                 >> qi::lit("OF")
156                 >> boolOrVar[qi::_val = qi::_1]
157                 >> qi::lit("AN")
158                 >> boolOrVar[qi::_val = qi::_val < qi::_1];
159
160         biggerStat  =    qi::lit("BIGGR")
161                 >> qi::lit("OF")
162                 >> boolOrVar[qi::_val = qi::_1]
163                 >> qi::lit("AN")
164                 >> boolOrVar[qi::_val = qi::_val > qi::_1];
165
166         visibleStat    =  qi::lit("VISIBLE")
167                     >>
168                         (
169                     literal[std::cout << qi::_1]
170                     | arithemticExpr[std::cout << qi::_1]
171                     | boolexpr[phoenix::bind(&print_bool_result, qi::_1)]
172                     | ident[print_variable()] // "print_variable.operator(qi::_1)"
173                   )
174                   >> (
175                         qi::char_('!')
176                       | qi::eps[phoenix::ref(std::cout) << std::endl]
177                     );
178
179         arithemticExpr =  qi::double_
180                       | additionStat
181                       | substractionStat
182                       | productStat
183                       | divisionStat;
184
185         boolexpr       =   qi::lit("WIN")[qi::_val = 1.0]
186                   | qi::lit("FAIL")[qi::_val = 0.0]
187                   | sameStat[qi::_val = qi::_1]
188                   | notsameStat[qi::_val = qi::_1]
189                   | smallerStat[qi::_val = qi::_1]
190                   | biggerStat[qi::_val = qi::_1]
191                   | arithemticExpr[qi::_val = qi::_1];
192
193         arithExOrVar =    arithemticExpr[qi::_val = qi::_1]
194                       | ident[qi::_val = phoenix::bind(&get_var, qi::_1)];
195
196         boolOrVar      =   boolexpr[qi::_val = qi::_1]
197                       | ident[qi::_val = phoenix::bind(&get_var, qi::_1)];
198
199         comment        = (
200                         qi::lit("BTW")
201                     >> qi::lexeme[*(qi::char_ - qi::eol)]
202                 )
203                 |
204                 (
205                     qi::lit("OBTW")
206                   >> qi::lexeme[*((qi::char_ | qi::eol) - qi::lit("TLDR"))]
207                   >> qi::lit("TLDR")
208                 );
209
210         ident          = qi::lexeme[qi::alpha >> *(qi::alnum | '_')];
211
212         include        =   qi::lit("STDIO")
213                 | qi::lit("STRING")
214                 | qi::lit("SOCKS")
215                 | qi::lit("STDLIB")
```

```
216                 ;
217
218       literal       = qi::lexeme[qi::lit('"') >> *(qi::char_ - qi::lit('"')) >> qi::lit('"')];
219     }
220
221     qi::rule<Iterator, qi::blank_type> program, stat, includeStat, vardeclStat, comment,
    ↪   visibleStat;
222     qi::rule<Iterator, qi::blank_type, qi::locals<std::string>> assignmentStat;
223     qi::rule<Iterator, double(), qi::blank_type> boolexpr, arithemticExpr, arithExOrVar,
    ↪   boolOrVar, additionStat, substractionStat, productStat, divisionStat, sameStat, notsameStat,
    ↪   smallerStat, biggerStat;
224     qi::rule<Iterator, std::string(), qi::blank_type> include, ident, literal;
225   };
226 }
227
228 #endif
```

```
HAI

OBTW This section declares includes the needed libs
         All supported libs are included
TLDR

CAN HAS STDIO?                    BTW Include STDIO lib
CAN HAS STRING?                   BTW Include STRING lib
CAN HAS SOCKS?                    BTW Include SOCKS lib
CAN HAS STDLIB?                   BTW Include STDLIB lib

OBTW This section declares all numeric variables
         Two numeric variables are declared and intialized
TLDR
I HAS A num1                     BTW declare variable num_ 1
I HAS A num2                     BTW declare variable num_ 2
num1 R 10.0                      BTW assign 10.0 to num1
num2 R 20.0                        BTW assign 10 to num2

OBTW This section declares all bool variables
         Two bool variables are declared and intialized
TLDR

I HAS A bool1                    BTW declare variable bool_ 1
I HAS A bool2                    BTW declare variable bool_ 2
bool1 R WIN                      BTW assign WIN to bool1
bool2 R FAIL                     BTW assign FAIL to bool2
SUM OF 1 AN 2

OBTW This section prints the current variable values
         Two bool and numeric variables are print
TLDR

VISIBLE "Value of num1:  = " !
VISIBLE num1                     BTW print value of num1
VISIBLE "Value of num2:  = " !
VISIBLE num2                     BTW print value of num2
```

```
VISIBLE "Value of bool1:  = " !
VISIBLE bool1                    BTW print value of bool1
VISIBLE "Value of bool2:  = " !
VISIBLE bool2                    BTW print value of bool2
VISIBLE "Value of IT:  = " !
VISIBLE IT                       BTW print value of bool2

BTW This section performs simple arithmetic expressions and prints them

VISIBLE "1 + 2 = 3 = " !
VISIBLE SUM OF 1 AN 2
VISIBLE "10 - 5 = 5 = " !
VISIBLE DIFF OF 10 AN 5
VISIBLE "2 * 2.5 = 5 = " !
VISIBLE PRODUKT OF 2.5 AN 2
VISIBLE "100 / 5 = 20 = " !
VISIBLE QUOSHUNT OF 100 AN 5

BTW This section performs complex arithmetic expressions and prints them

VISIBLE "10 - (100 / 10) = 0 = " !
VISIBLE DIFF OF 10 AN QUOSHUNT OF 100 AN 10
VISIBLE "2 * (3 + 2) = 10 = " !
VISIBLE PRODUKT OF 2 AN SUM OF 3 AN 2
VISIBLE "100 / (100 - 90) = 10 = " !
VISIBLE QUOSHUNT OF 100 AN DIFF OF 100 AN 90

BTW This section performs complex arithmetic expressions with variables and prints them

VISIBLE "num1 + num2 = 30 = " !
VISIBLE SUM OF num1 AN num2
VISIBLE "num1 + (num1 * num2) = 210 = " !
VISIBLE SUM OF num1 AN PRODUKT OF num1 AN num2
VISIBLE "100 + (50 * num1) = 600 = " !
VISIBLE SUM OF 100 AN PRODUKT OF 50 AN num1

BTW This section performs simple bool expressions and prints them

VISIBLE "WIN == WIN = WIN = " !
VISIBLE BOTH SAEM WIN AN WIN
VISIBLE "WIN != FAIL = WIL = " !
VISIBLE DIFFRINT WIN AN FAIL
VISIBLE "WIN < FAIL = FAIL = " !
VISIBLE SMALLR OF WIN AN FAIL
VISIBLE "FAIL > WIN = FAIL = " !
VISIBLE BIGGR OF FAIL AN WIN

VISIBLE "10 == 10 = WIN = " !
VISIBLE BOTH SAEM WIN AN WIN
VISIBLE "10 != 11 = WIN = " !
VISIBLE DIFFRINT WIN AN FAIL
```

```
VISIBLE "10 < 9 = FAIL = " !
VISIBLE SMALLR OF 10 AN 9
VISIBLE "9 > 10 = FAIL = " !
VISIBLE BIGGR OF 9 AN 10

VISIBLE "WIN == (WIN == FAIL) = FAIL = " !
VISIBLE BOTH SAEM WIN AN BOTH SAEM WIN AN FAIL
VISIBLE "WIN != (WIN != FAIL) = FAIL = " !
VISIBLE DIFFRINT WIN AN DIFFRINT WIN AN FAIL
VISIBLE "FAIL < (WIN > FAIL) = WIN = " !
VISIBLE SMALLR OF FAIL AN BIGGR OF WIN AN FAIL
VISIBLE "WIN > (WIN < FAIL) = WIN = " !
VISIBLE BIGGR OF WIN AN SMALLR OF WIN AN FAIL

VISIBLE "10 == (10 - 10) = FAIL = " !
VISIBLE BOTH SAEM 10 AN DIFF OF 10 AN 10
VISIBLE "10 != (9 + 1) = FAIL = " !
VISIBLE DIFFRINT 10 AN SUM OF 9 AN 1
VISIBLE "10 < (1 * 11) = WIN = " !
VISIBLE SMALLR OF 10 AN PRODUKT OF 1 AN 11
VISIBLE "10 > (90 / 10) = WIN = " !
VISIBLE BIGGR OF 10 AN QUOSHUNT OF 90 AN 10

BTW This section performs complex bool expressions with variables and prints them

VISIBLE "bool1 == bool2 = FAIL = " !
VISIBLE BOTH SAEM bool1 AN bool2
VISIBLE "bool1 == (bool1 != bool2) = WIN = " !
VISIBLE BOTH SAEM bool1 AN DIFFRINT bool1 AN bool2
VISIBLE "bool1 == (100 != num1) = WIN = " !
VISIBLE BOTH SAEM bool1 AN DIFFRINT 100 AN num1

VISIBLE "End of program"

KTHXBYE
```

Listing 5: lolcode-test.hpp

```
1  #ifndef _lolcode_test_hpp_
2  #define _lolcode_test_hpp_
3
4  #include <iostream>
5  #include <string>
6  #include <fstream>
7
8  #include "lolcode.hpp"
9
10 using namespace std;
11 namespace qi = boost::spirit::qi;
12
13 void test_lolcode(string file) {
14   cout << "testing lolcode parser with file " << file << endl;
15   std::ifstream fileStream(file);
16
17   string code((std::istreambuf_iterator<char>(fileStream)),
```

```
    std::istreambuf_iterator<char>());

  auto begin = code.begin();
  auto end = code.end();
  lolcode::lolcode_grammar<decltype(begin)> grammar;

  cout << "-------------------------------------------------------"
    << endl
    << "parsing ... "
    << endl
    << "-------------------------------------------------------"
    << endl
    << code
    << endl
    << "-------------------------------------------------------"
    << endl;

  bool success = qi::phrase_parse(begin, end, grammar, qi::blank) && begin == end;

  cout << endl
    << "-------------------------------------------------------"
    << endl
    << "success = " << boolalpha << success
    << endl
    << "-------------------------------------------------------"
    << endl;
  fileStream.close();
}

#endif _lolcode_test_hpp_
```

Listing 6: main.cpp

```
#include <iostream>
#include <string>
#include <fstream>

#include "json-test.hpp"
#include "lolcode-test.hpp"

using namespace std;

int main() {
  test_lolcode("./test.lolcode");

  cout << endl << endl;

  test_json("./test.json");

  // console won't keep open in VS
  cin.get();

  return 0;
}
```

```
testing lolcode parser with file ./test.lolcode
-----------------------------------------------------
parsing ...
-----------------------------------------------------
HAI

OBTW This section declares includes the needed libs
         All supported libs are included
TLDR

CAN HAS STDIO?                   BTW Include STDIO lib
CAN HAS STRING?                  BTW Include STRING lib
CAN HAS SOCKS?                   BTW Include SOCKS lib
CAN HAS STDLIB?                  BTW Include STDLIB lib

OBTW This section declares all numeric variables
         Two numeric variables are declared and intialized
TLDR
I HAS A num1                    BTW declare variable num_ 1
I HAS A num2                    BTW declare variable num_ 2
num1 R 10.0                     BTW assign 10.0 to num1
num2 R 20.0                       BTW assign 10 to num2

OBTW This section declares all bool variables
         Two bool variables are declared and intialized
TLDR

I HAS A bool1                   BTW declare variable bool_ 1
I HAS A bool2                   BTW declare variable bool_ 2
bool1 R WIN                     BTW assign WIN to bool1
bool2 R FAIL                    BTW assign FAIL to bool2
SUM OF 1 AN 2

OBTW This section prints the current variable values
         Two bool and numeric variables are print
TLDR

VISIBLE "Value of num1:  = " !
VISIBLE num1                    BTW print value of num1
VISIBLE "Value of num2:  = " !
VISIBLE num2                    BTW print value of num2
VISIBLE "Value of bool1:  = " !
VISIBLE bool1                   BTW print value of bool1
VISIBLE "Value of bool2:  = " !
VISIBLE bool2                   BTW print value of bool2
VISIBLE "Value of IT:  = " !
VISIBLE IT                      BTW print value of bool2

BTW This section performs simple arithmetic expressions and prints them

VISIBLE "1 + 2 = 3 = " !
VISIBLE SUM OF 1 AN 2
VISIBLE "10 - 5 = 5 = " !
VISIBLE DIFF OF 10 AN 5
VISIBLE "2 * 2.5 = 5 = " !
VISIBLE PRODUKT OF 2.5 AN 2
VISIBLE "100 / 5 = 20 = " !
VISIBLE QUOSHUNT OF 100 AN 5

BTW This section performs complex arithmetic expressions and prints them
```

Abbildung 2: Test Teil 1

```
VISIBLE "10 - (100 / 10) = 0 = " !
VISIBLE DIFF OF 10 AN QUOSHUNT OF 100 AN 10
VISIBLE "2 * (3 + 2) = 10 = " !
VISIBLE PRODUKT OF 2 AN SUM OF 3 AN 2
VISIBLE "100 / (100 - 90) = 10 = " !
VISIBLE QUOSHUNT OF 100 AN DIFF OF 100 AN 90

BTW This section performs complex arithmetic expressions with variables and prints them

VISIBLE "num1 + num2 = 30 = " !
VISIBLE SUM OF num1 AN num2
VISIBLE "num1 + (num1 * num2) = 210 = " !
VISIBLE SUM OF num1 AN PRODUKT OF num1 AN num2
VISIBLE "100 + (50 * num1) = 600 = " !
VISIBLE SUM OF 100 AN PRODUKT OF 50 AN num1

BTW This section performs simple bool expressions and prints them

VISIBLE "WIN == WIN = WIN = " !
VISIBLE BOTH SAEM WIN AN WIN
VISIBLE "WIN != FAIL = WIL = " !
VISIBLE DIFFRINT WIN AN FAIL
VISIBLE "WIN < FAIL = FAIL = " !
VISIBLE SMALLR OF WIN AN FAIL
VISIBLE "FAIL > WIN = FAIL = " !
VISIBLE BIGGR OF FAIL AN WIN

VISIBLE "10 == 10 = WIN = " !
VISIBLE BOTH SAEM WIN AN WIN
VISIBLE "10 != 11 = WIN = " !
VISIBLE DIFFRINT WIN AN FAIL
VISIBLE "10 < 9 = FAIL = " !
VISIBLE SMALLR OF 10 AN 9
VISIBLE "9 > 10 = FAIL = " !
VISIBLE BIGGR OF 9 AN 10

VISIBLE "WIN == (WIN == FAIL) = FAIL = " !
VISIBLE BOTH SAEM WIN AN BOTH SAEM WIN AN FAIL
VISIBLE "WIN != (WIN != FAIL) = FAIL = " !
VISIBLE DIFFRINT WIN AN DIFFRINT WIN AN FAIL
VISIBLE "FAIL < (WIN > FAIL) = WIN = " !
VISIBLE SMALLR OF FAIL AN BIGGR OF WIN AN FAIL
VISIBLE "WIN > (WIN < FAIL) = WIN = " !
VISIBLE BIGGR OF WIN AN SMALLR OF WIN AN FAIL

VISIBLE "10 == (10 - 10) = FAIL = " !
VISIBLE BOTH SAEM 10 AN DIFF OF 10 AN 10
VISIBLE "10 != (9 + 1) = FAIL = " !
VISIBLE DIFFRINT 10 AN SUM OF 9 AN 1
VISIBLE "10 < (1 * 11) = WIN = " !
VISIBLE SMALLR OF 10 AN PRODUKT OF 1 AN 11
VISIBLE "10 > (90 / 10) = WIN = " !
VISIBLE BIGGR OF 10 AN QUOSHUNT OF 90 AN 10

BTW This section performs complex bool expressions with variables and prints them
```

Abbildung 3: Test Teil 2

```
VISIBLE "bool1 == bool2 = FAIL = " !
VISIBLE BOTH SAEM bool1 AN bool2
VISIBLE "bool1 == (bool1 != bool2) = WIN = " !
VISIBLE BOTH SAEM bool1 AN DIFFRINT bool1 AN bool2
VISIBLE "bool1 == (100 != num1) = WIN = " !
VISIBLE BOTH SAEM bool1 AN DIFFRINT 100 AN num1

VISIBLE "End of program"

KTHXBYE

-----------------------------------------------------

semantics-info: created new var 'num1

semantics-info: created new var 'num2

semantics-info: set numeric var 'num1 to: 10

semantics-info: set numeric var 'num2 to: 20

semantics-info: created new var 'bool1

semantics-info: created new var 'bool2

semantics-info: set boolean var 'bool1 to: WIN

semantics-info: set boolean var 'bool2 to: FAIL

semantics-info: set numeric var 'IT to: 3
Value of num1:  = 10

Value of num2:  = 20

Value of bool1:  = WIN

Value of bool2:  = FAIL

Value of IT:  = 3

1 + 2 = 3 = 3
10 - 5 = 5 = 5
2 * 2.5 = 5 = 5
100 / 5 = 20 = 20
10 - (100 / 10) = 0 = 0
2 * (3 + 2) = 10 = 10
100 / (100 - 90) = 10 = 10
num1 + num2 = 30 =
semantics-info: resolved variable 'num1' to value: 10

semantics-info: resolved variable 'num2' to value: 20
30
num1 + (num1 * num2) = 210 =
semantics-info: resolved variable 'num1' to value: 10

semantics-info: resolved variable 'num1' to value: 10

semantics-info: resolved variable 'num2' to value: 20
210
100 + (50 * num1) = 600 =
semantics-info: resolved variable 'num1' to value: 10
600
WIN == WIN = WIN = WIN
```

Abbildung 4: Test Teil 3

```
WIN < FAIL = FAIL = FAIL

FAIL > WIN = FAIL = FAIL

10 == 10 = WIN = WIN

10 != 11 = WIN = WIN

10 < 9 = FAIL = FAIL

9 > 10 = FAIL = FAIL

WIN == (WIN == FAIL) = FAIL = FAIL

WIN != (WIN != FAIL) = FAIL = FAIL

FAIL < (WIN > FAIL) = WIN = WIN

WIN > (WIN < FAIL) = WIN = WIN

10 == (10 - 10) = FAIL = FAIL

10 != (9 + 1) = FAIL = FAIL

10 < (1 * 11) = WIN = WIN

10 > (90 / 10) = WIN = WIN

bool1 == bool2 = FAIL =
semantics-info: resolved variable 'bool1' to value: 1

semantics-info: resolved variable 'bool2' to value: 0
FAIL

bool1 == (bool1 != bool2) = WIN =
semantics-info: resolved variable 'bool1' to value: 1

semantics-info: resolved variable 'bool1' to value: 1

semantics-info: resolved variable 'bool2' to value: 0
WIN

bool1 == (100 != num1) = WIN =
semantics-info: resolved variable 'bool1' to value: 1

semantics-info: resolved variable 'num1' to value: 10
WIN

End of program

-----------------------------------------------------
success = true
-----------------------------------------------------
```

Abbildung 5: Test Teil 4