|  |  |
| --- | --- |
| <hello> | means that hello is a nonterminal symbol |
| **hello** | (in **bold** font) means that hello is a terminal symbol |
| x\* | means zero or more occurrences of x |
| X+ | means a comma-separated list of one or more x’s |
| | | means separate alternatives |
| → | means deriving |
| {} | means optional occurrences |

S = {<source code>}

VN = {<source code>, <set of affirmations>, <affirmation>, <function declaration>, <more identifiers>, <variable declaration>, < identifier >, <letter>, <digit>, <calculator>, <call function>, <conditions>, <number>, <equal to>, <operation>, <condition>, <variable>, <map declaration>, <map nr and char>, <array use>, <map use>, <break>, <characters>, <array declaration>, <continue>, }

VT = {0, …, 9, a, …, z, A, …, Z, main, let, for, while, break, if, else, case, switch, fn, true, false, if, else, return, continue, do, struct, enum, const, “&”, “|”, “:”, “=”, “!”, “;”, “{”, “}”, “<”, “>”, “(”, “)”, “[”, “]”, “+”, “-”, “\*”, “/”, “””, “\_”, “,”, “.”, “:”, “#”}

P = {

<source code> → <function declaration>\* **main () {** <player> **{**<set of affirmations>**}**

<function declaration> → **fn**<identifier>**(**<more identifiers>+**){**<set of affirmations>**}**

<more identifiers> → <identifier> | <identifier> <more identifiers> | <identifier>**:**<more identifiers>

<identifier> → <letter> | {<letter> | <digit>| \_} \*

<set of affirmations> → < affirmation> | < affirmation><set of affirmations>

<affirmation> → <variable declaration> |

<call function> |

<array declaration> |

<map declaration> |

<array use> |

<map use> |

<calculator> |

<return> |

<continue> |

<break> |

**if (**<conditions>**)** **{**<set of affirmations>**}** |

**if (**<conditions>**)** **{**<set of affirmations>**} else** **{**<set of affirmations>**}** |

**while** **(**<conditions>**)** **{**<set of affirmations>**} |**

**for (**<calculator>**;** <conditions>**;** <equal to>**)** **{**<set of affirmations>**} |**

**do {**<set of affirmations>**} while (**<conditions>**); |**

**switch (**<nr and char>**) {**<set of affirmations> **} |**

**case** <nr and char> **:** <set of affirmations> **|**

**struct** <nr and char> **{**<nr and char>\***}** <nr and char>\***; |**

**enum** <nr and char> **{**<nr and char>+**}; |**

**const** <set of affirmations>

<variable declaration> → **let** <identifier>**; | let** <identifier> **=** <set of affirmations>**;**

<call function> → <identifier> **(**<more identifiers>+**);**

<array declaration> → **let** <identifier> = [<nr and char>+**];**

<map declaration> → **let** <identifier> **= {**<map nr and char>+**};**

<array use> → <identifier> **[**<nr and char>**];**

<map use> → <identifier> **[**<nr and char>**];**

<return> → **return** <call function>**;**

| **return** <array use>**;**

| **return** <map use>**;**

| **return** <equal to>**;**

<calculator> → <identifier> **=** <equal to>**;**

<equal to> → <identifier> |

<identifier> <operation> <equal to> |

<number> |

<number> <operation> <equal to>

<operation> → **+ | - | \* | /**

<break> → **break;**

<continue> → **continue;**

<conditions> → {**!**}<nr and char> <condition> <conditions> | {**!**}<nr and char>

<map nr and char> → <nr and char> **:** <nr and char>

<nr and char> → <variable>|<number> | <variable><nr and char>|<number><nr and char>

<variable> → <identifier> | <letter>

<condition> → **& | | | = | < | > |** <condition><condition>

<number> → <digit> | <digit> <number>

<letter> → **a** | **b** | **c** | … | **A** | **B** | **C** | … | **Z**

<digit> → **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9 | .** }