Zhaba-script syntax🐸

Примечания:

1. Определение синтаксической диаграммы вида: $<a_x>$ `::=` b, где $x \in S$ и $S \cap Z = S$ эквивалентна следующей записи:
   ```
    <a_{S0}> ::= b
    <a_{S1}> ::= b
      …
    <a_{Sn}> ::= b
   ```
2. Использование синтаксической диаграммы вида: $<a_x>$, где $x \in S$ и $S \cap Z = S$ эквивалентна следующей записи: $[<a_{S0}>|a_{S1}>|…|<a_{Sn}>]$
3. Данный синтаксис не учитывает препроцессинг потому что да.

```
<zhaba-script-program> ::=
   {<op-def_0>
   |<type-def>
   |<impl-def>
   |<spaces_x><nl>}

<spaces_0> ::= ""
<spaces_{n,n > 0}> ::= " "<spaces_{n-1}>
<nl> ::= "\n"
<op-char> ::=
  |"~"|","|"."|"+"|"-"|"*"|"\"
  |"%"|"<"|">"|"="|"^"|"&"|":"
  |"|"|"/"|"!"|"#"|"$"|"@"|"?"
<digit>
::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
<letter> ::=
  |"A"|"B"|"C"|"D"|"E"|"F"|"G"|"H"|"I"|"J"|"K"|"L"|"M"|"N"
  |"O"|"P"|"Q"|"R"|"S"|"T"|"U"|"V"|"W"|"X"|"Y"|"Z"|"a"|"b"
  |"c"|"d"|"e"|"f"|"g"|"h"|"i"|"j"|"k"|"l"|"m"|"n"|"o"|"p"
  |"q"|"r"|"s"|"t"|"u"|"v"|"w"|"x"|"y"|"z"|"_"
<id> ::=
  | <letter>{<letter>|<digit>}
  | <op-char>{<op-char>}

<op-def_n> ::= <op-t>[<type>]<id>{<arg>}<block_{x + n,x > 0}>
<op-t> ::= "fn"|"lop"|"rop"|("bop"<int>)
```

```
<arg> ::= <type><id>

<type-def> ::=
  "type"(<normal-type-def>|<generic-type-def>)<type-block$_{x,x > 0}$)>
<normal-type-def> ::= <id>
<generic-type-def> ::= <id><id>{<id>}
<type-block$_n$> ::=
  | <nl><type-block-ln$_n$>{<type-block-ln$_n$>}
  | ":"<nl><type-block-content>{<type-block-ln$_n$>}
  | ":"<type-block-content><nl>
<type-block-ln$_n$>   ::= <spaces$_n$><type-block-content><nl>
<type-block-content> ::= <type-literal><id>{<id>}<nl>

<impl-def> ::= "impl"<type-literal><impl-block$_{x,x > 0}$>
<impl-block$_n$> ::=
  (":"<op-def$_{x,x > 0}$>|":"<nl>)<impl-block-item$_n$>{ impl-block-item$_n$}
<impl-block-item$_n$> ::=
  | <op-def$_n$>
  | <spaces$_{x,x >= 0}$><nl>

<bin-digit> ::= “0”|”1”
<hex-digit> ::=
  |<digit>
  |"A"|"B"|"C"|"D"|"E"|"F"
  |"a"|"b"|”c"|"d"|"e"|"f"
<int> ::=
  |”0x”<hex-digit>{<hex-digit>}
  |”0b”<bin-digit>{<bin-digit>}
  |<digit>{<digit>}

<i8-literal>  ::= <int>"i8"
<i16-literal> ::= <int>"i16"
<i32-literal> ::= <int>"i32"
<i64-literal> ::= <int>["i64"|"i"]
<u8-literal>  ::= <int>"u8"
<u16-literal> ::= <int>"u16"
<u32-literal> ::= <int>"u32"
<u64-literal> ::= <int>("u64"|"u")
```

```
<float> ::=
  | <digit>{<digit>}"."<digit>{<digit>}
  | "."<digit>{<digit>}
  | <digit>{<digit>}"."
<f32-literal> ::= (<float>|<int>)"f32"
<f64-literal> ::= (<float>["f64"|"f"]|<int>("f64"|"f"]))
<f80-literal> ::= (<float>|<int>)"f80"
<bool-literal> ::= “tru”|”fls”|"true"|"false"
<type-literal> ::= <base-type>|<user-type>
<user-type> ::=
<id>["<"<type-literal>{<type-literal>}">")]{"P"|"*"}["R"|"&"]
<base-type> ::=
  |"int"|"i8"|"i16"|"i32"|"i64"|"u8"|"u16"
  |"u32"|"u64"|"char"|"str"|”f32”|”f64”|”f80”|"bool"
<str-literal> ::=
  | "'"{<escaped-char>|<char-not-slash>}"'"
  | "`"{“\`”|<char-not-backtick>}"`"
<escaped-char> ::= "\\"<char>
<literal> ::=
  | <type-literal>
  | <i8-literal>
  | <i16-literal>
  | <i32-literal>
  | <i64-literal>
  | <u8-literal>
  | <u16-literal>
  | <u32-literal>
  | <u64-literal>
  | <f32-literal>
  | <f64-literal>
  | <bool-literal>
  | <str-literal>

<block$_n$> ::=
  | <nl><block-ln$_n$>{<block-ln$_n$>}
  | ":"<nl><block-content>{<block-ln$_n$>}
  | ":"<block-content><nl>
<block-ln$_n$> ::= <spaces$_n$><block-content$_n$><nl>
```

```
<block-content$_n$> ::=
  |<exp>
  |<if$_n$>
  |<loop$_n$>
  |<explicit-var-decl>
  |<ret>

<ret> ::= “<<<”<exp>

<if$_n$> ::= "?"<exp><block$_{n+x, x>0}$>{<elif$_{n+x, x>0}$>}[<else$_{n+x, x>0}$>]
<elif$_n$> ::= "|"<exp><block$_{n+x, x>0}$>
<else$_n$> ::= "\"<block>
```

Примечание:
В выражениях с <optional-comma> может быть вставлена запятая между 2 токенами если левый токен - (“)”|литерал|идентификатор) и правый токен - (“(”|литерал|идентификатор).

```
<optional-comma> ::= [","]

<loop$_n$> ::= "@"<loop-exp><block$_{n+x, x>0}$>
<loop-exp> ::=
  | <loop-exp-while>
  | <loop-exp-foreach>
  | <loop-exp-for>
<loop-exp-while> ::= <exp>
<loop-exp-foreach> ::= <id><optional-comma><exp>
<loop-exp-for> ::= <exp><optional-comma><exp><optional-comma><exp>

<expl-var-decl> ::=
  (<type-literal>|"auto")<expl-var-decl-tuple>
<expl-var-decl-item> ::= <id>|<id>"="<exp>
<expl-var-decl-tuple> ::=
   <expl-var-decl-tuple><optional-comma><expl-var-decl-item>
```

Примечание:

1. Согласно следующему определению `<exp>` и `<tuple>` эквивалентны, но подразумевается использование `<exp>` когда тип не void и это не перечисление `<exp>`, и `<tuple>` когда это некоторое перечисление `<exp>`.

2. lop - left operator - левый(префиксный) оператор
3. rop - right operator - правый(постфиксный) оператор
4. bop - binary operator - бинарный оператор

```
<exp> ::=
  | "("<exp>")"
  | <literal>
  | <tuple>
  | <var>
  | <type-cast>
  | <lop-call>
  | <rop-call>
  | <bop-call>
  | <fn-call>
  | <member-call>
  | <member-access>
  | <ptr-member-call>
  | <ptr-member-access>
  | <assign>
  | <implicit-var-decl>

<var> ::= <id>
<assign> ::= <exp>”=”<exp>

<type-cast> ::= <exp>"as"<type-literal>

<lop-call> ::= <lop><tuple>
<lop> ::= <builtin-lop>|<used-defined-lop>
<builtin-lop> ::=
  |"*"|"!"|"&"|"out"|"put"|"sizeof"|"malloc"|"free"
  |"in_i8"|"in_i16"|"in_i32"|"in_i64"|"in_u8"|"in_u16"
  |"in_u32"|"in_u64"|"in_char"|"in_str"|"in_bool”
  |"in_f32"|"in_f64"|"in_f80"
<lib-defined-lop> ::= ”--”|"++"|"+"|"-"
<user-defined-lop> ::= <id>

<rop-call> ::= <tuple><rop>
<rop> ::= <lib-defined-rop>|<used-defined-rop>
```

```
<lib-defined-rop> ::= ”--”|"++"
<user-defined-rop> ::= <id>

<bop-call> ::= <exp><bop><tuple>
<bop> ::=
  | <builtin-bop>
  | <lib-defined-bop>
  | <used-defined-bop>
<builtin-bop> ::=
  |"+"|"-"|"/"|"%"|"*"
  |"=="|"!="|"<"|">"|"<="|">="|"&&"|"||"
<lib-defined-bop> ::= ".."|"**"|"%%"
<user-defined-bop> ::= <id>
<implicit-var-decl> ::= <id>":="<exp>

<member-access> ::= <exp>"."<id>
<member-call> ::= <exp>"."<id>"("[<populated-tuple>]")"
<ptr-member-access> ::= <exp>"->"<id>
<ptr-member-call> ::= <exp>"->"<id>"("[<populated-tuple>]")"

<fn-call> ::= <id>"("[<populated-tuple>]")"

<tuple> ::= <populated-tuple>|<void-tuple>
<void-tuple> ::= “()”
<populated-tuple> ::=
  | <exp>
  | <tuple><optional-comma><exp>
```