

# Weak language draft

epoll-reactor

since December 2021

## Contents

<b>1</b>	<b>Scope</b>	<b>2</b>
<b>2</b>	<b>Lexical elements</b>	<b>2</b>
2.1	Keywords . . . . .	2
2.2	Operators and punctuators . . . . .	2
<b>3</b>	<b>Grammar summary</b>	<b>2</b>
<b>4</b>	<b>Environment</b>	<b>5</b>
4.1	Data types . . . . .	5
4.2	Inside-iteration statements . . . . .	5
4.3	Iteration statements . . . . .	5
4.4	Conditional statements . . . . .	6
4.5	Jump statements . . . . .	6
4.6	Translation environment . . . . .	6
<b>5</b>	<b>Implementation</b>	<b>6</b>
5.1	Intermediate representation format . . . . .	6

# 1 Scope

This document describes requirements for implementation of weak programming language.

## 2 Lexical elements

### 2.1 Keywords

<b>boolean</b>	<b>break</b>	<b>char</b>
<b>continue</b>	<b>do</b>	<b>false</b>
<b>float</b>	<b>for</b>	<b>if</b>
<b>int</b>	<b>return</b>	<b>string</b>
<b>true</b>	<b>void</b>	<b>while</b>

### 2.2 Operators and punctuators

<b>=</b>	<b>*=</b>	<b>/=</b>	<b>%=</b>	<b>+=</b>	<b>-=</b>
<b>&lt;&lt;=</b>	<b>&gt;&gt;=</b>	<b>&amp;=</b>	<b> =</b>	<b>≐</b>	<b>&amp;&amp;</b>
<b> </b>	<b>^</b>	<b>&amp;</b>	<b>==</b>	<b>!=</b>	<b>&gt;</b>
<b>&lt;</b>	<b>&gt;=</b>	<b>&lt;=</b>	<b>&lt;&lt;</b>	<b>&gt;&gt;</b>	<b>+</b>
<b>-</b>	<b>*</b>	<b>/</b>	<b>%</b>	<b>++</b>	<b>--</b>
<b>[</b>	<b>]</b>	<b>(</b>	<b>)</b>	<b>{</b>	<b>}</b>

## 3 Grammar summary

$\langle \text{program} \rangle$	$::= \langle \text{function-declaration} \rangle^*$
$\langle \text{function-declaration} \rangle$	$::= \langle \text{ret-type} \rangle \langle \text{id} \rangle ( \langle \text{parameter-list-opt} \rangle ) \{ \langle \text{stmt} \rangle^* \}$
$\langle \text{ret-type} \rangle$	$::= \langle \text{type} \rangle$ $\quad   \quad \langle \text{void-type} \rangle$
$\langle \text{type} \rangle$	$::= \textit{int}$ $\quad   \quad \textit{float}$ $\quad   \quad \textit{char}$ $\quad   \quad \textit{string}$ $\quad   \quad \textit{boolean}$
$\langle \text{void-type} \rangle$	$::= \textit{void}$
$\langle \text{constant} \rangle$	$::= \langle \text{integral-literal} \rangle$ $\quad   \quad \langle \text{floating-literal} \rangle$ $\quad   \quad \langle \text{string-literal} \rangle$ $\quad   \quad \langle \text{boolean-literal} \rangle$

$\langle \text{integral-literal} \rangle$	$::= \langle \text{digit} \rangle^*$
$\langle \text{floating-literal} \rangle$	$::= \langle \text{digit} \rangle^* . \langle \text{digit} \rangle^*$
$\langle \text{string-literal} \rangle$	$::= \text{ " } ( \backslash \text{x00000000} - \backslash \text{x0010FFFF} )^* \text{ "}$
$\langle \text{boolean-literal} \rangle$	$::= \text{ true }$   $\text{ false }$
$\langle \text{alpha} \rangle$	$::= \text{ a }   \text{ b }   \dots   \text{ z }   \_$
$\langle \text{digit} \rangle$	$::= \text{ 0 }   \text{ 1 }   \dots   \text{ 9 }$
$\langle \text{id} \rangle$	$::= \langle \text{alpha} \rangle ( \langle \text{alpha} \rangle   \langle \text{digit} \rangle )^*$
$\langle \text{parameter} \rangle$	$::= \langle \text{type} \rangle \langle \text{id} \rangle$
$\langle \text{parameter-list} \rangle$	$::= \langle \text{parameter} \rangle , \langle \text{parameter-list} \rangle$   $\langle \text{parameter} \rangle$
$\langle \text{parameter-list-opt} \rangle$	$::= \langle \text{parameter-list} \rangle   \epsilon$
$\langle \text{stmt} \rangle$	$::= \langle \text{selection-stmt} \rangle$   $\langle \text{iteration-stmt} \rangle$   $\langle \text{jump-stmt} \rangle$   $\langle \text{expr} \rangle$
$\langle \text{iteration-stmt} \rangle$	$::= \langle \text{stmt} \rangle$   $\text{ break; }$   $\text{ continue; }$
$\langle \text{selection-stmt} \rangle$	$::= \text{ if } ( \langle \text{expr} \rangle ) \{ \langle \text{stmt} \rangle^* \}$   $\text{ if } ( \langle \text{expr} \rangle ) \{ \langle \text{stmt} \rangle^* \} \text{ else } \{ \langle \text{stmt} \rangle^* \}$
$\langle \text{iteration-stmt} \rangle$	$::= \text{ for } ( \langle \text{expr-opt} \rangle ; \langle \text{expr-opt} \rangle ; \langle \text{expr-opt} \rangle ) \{ \langle \text{iteration-stmt} \rangle^* \}$   $\text{ while } ( \langle \text{expr} \rangle ) \{ \langle \text{iteration-stmt} \rangle^* \}$   $\text{ do } \{ \langle \text{iteration-stmt} \rangle^* \} \text{ while } ( \langle \text{expr} \rangle )$
$\langle \text{jump-stmt} \rangle$	$::= \text{ return } \langle \text{expr} \rangle ? ;$
$\langle \text{assignment-op} \rangle$	$::= =$   $\text{ *= }$   $\text{ /= }$   $\text{ \%= }$   $\text{ += }$   $\text{ -= }$

		<<=
		>>=
		&=
		=
		^=
$\langle expr \rangle$	$::=$	$\langle assignment\text{-}expr \rangle$
$\langle expr\text{-}opt \rangle$	$::=$	$\langle expr \rangle \mid \epsilon$
$\langle assignment\text{-}expr \rangle$	$::=$	$\langle logical\text{-}or\text{-}expr \rangle$   $\langle unary\text{-}expr \rangle \langle assignment\text{-}op \rangle \langle assignment\text{-}expr \rangle$
$\langle logical\text{-}or\text{-}expr \rangle$	$::=$	$\langle logical\text{-}and\text{-}expr \rangle$   $\langle logical\text{-}or\text{-}expr \rangle \parallel \langle logical\text{-}and\text{-}expr \rangle$
$\langle logical\text{-}and\text{-}expr \rangle$	$::=$	$\langle inclusive\text{-}or\text{-}expr \rangle$   $\langle logical\text{-}and\text{-}expr \rangle \ \&\& \ \langle inclusive\text{-}or\text{-}expr \rangle$
$\langle inclusive\text{-}or\text{-}expr \rangle$	$::=$	$\langle exclusive\text{-}or\text{-}expr \rangle$   $\langle inclusive\text{-}or\text{-}expr \rangle \mid \langle exclusive\text{-}or\text{-}expr \rangle$
$\langle exclusive\text{-}or\text{-}expr \rangle$	$::=$	$\langle and\text{-}expr \rangle$   $\langle exclusive\text{-}or\text{-}expr \rangle \ \sim \ \langle and\text{-}expr \rangle$
$\langle and\text{-}expr \rangle$	$::=$	$\langle equality\text{-}expr \rangle$   $\langle and\text{-}expr \rangle \ \& \ \langle equality\text{-}expr \rangle$
$\langle equality\text{-}expr \rangle$	$::=$	$\langle relational\text{-}expr \rangle$   $\langle equality\text{-}expr \rangle \ == \ \langle relational\text{-}expr \rangle$   $\langle equality\text{-}expr \rangle \ \ != \ \langle relational\text{-}expr \rangle$
$\langle relational\text{-}expr \rangle$	$::=$	$\langle shift\text{-}expr \rangle$   $\langle relational\text{-}expr \rangle \ > \ \langle shift\text{-}expr \rangle$   $\langle relational\text{-}expr \rangle \ < \ \langle shift\text{-}expr \rangle$   $\langle relational\text{-}expr \rangle \ >= \ \langle shift\text{-}expr \rangle$   $\langle relational\text{-}expr \rangle \ <= \ \langle shift\text{-}expr \rangle$
$\langle shift\text{-}expr \rangle$	$::=$	$\langle additive\text{-}expr \rangle$   $\langle shift\text{-}expr \rangle \ << \ \langle additive\text{-}expr \rangle$   $\langle shift\text{-}expr \rangle \ >> \ \langle additive\text{-}expr \rangle$
$\langle additive\text{-}expr \rangle$	$::=$	$\langle multiplicative\text{-}expr \rangle$   $\langle additive\text{-}expr \rangle \ + \ \langle multiplicative\text{-}expr \rangle$   $\langle additive\text{-}expr \rangle \ - \ \langle multiplicative\text{-}expr \rangle$

$\langle \text{multiplicative-expr} \rangle$	$::=$	$\langle \text{unary-expr} \rangle$   $\langle \text{multiplicative-expr} \rangle * \langle \text{unary-expr} \rangle$   $\langle \text{multiplicative-expr} \rangle / \langle \text{unary-expr} \rangle$   $\langle \text{multiplicative-expr} \rangle \% \langle \text{unary-expr} \rangle$
$\langle \text{unary-expr} \rangle$	$::=$	$\langle \text{postfix-expr} \rangle$   $++ \langle \text{unary-expr} \rangle$   $-- \langle \text{unary-expr} \rangle$
$\langle \text{postfix-expr} \rangle$	$::=$	$\langle \text{primary-expr} \rangle$   $\langle \text{postfix-expr} \rangle [ \langle \text{expr} \rangle ]$   $\langle \text{postfix-expr} \rangle ++$   $\langle \text{postfix-expr} \rangle --$
$\langle \text{primary-expr} \rangle$	$::=$	$\langle \text{constant} \rangle$   $\langle \text{id} \rangle$   $( \langle \text{expr} \rangle )$

## 4 Environment

### 4.1 Data types

The language must implement static strong typing. All casts must be explicit.

- **Int** – Signed 32-bit;
- **Float** – Signed 32-bit;
- **Bool** – 8-bit;
- **String** – Character sequence, that ends with Null character;
- **Void** – Empty type, used as return type only.

### 4.2 Inside-iteration statements

- **Break** – Usable only inside the **while**, **do-while** and **for** statements and performs exit from a loop.
- **Continue** – Usable only inside the **while**, **do-while** and **for** statements and performs jump to the next iteration.

### 4.3 Iteration statements

- **While** – Loop statement that performs its body until the condition evaluates to true.
- **Do-While** – Loop statement with similar to **While** semantics, but it executes body before condition check at first time.

- **For** – Loop statement with three initial parts and body. This includes:
  - **Initial** part with the variable assignment;
  - **Conditional** part with the some condition;
  - **Incremental** part with the some statement, that should change assigned variable.

#### 4.4 Conditional statements

- **If** – Conditional statement, that should execute If-part when it's condition evaluates to true. Otherwise, Else-part should be executed.

#### 4.5 Jump statements

- **Return** – The end point of control flow, may return value, may not (void functions).

#### 4.6 Translation environment

The whole program must be placed in one file to simplify translation and linking (lack of it as such).

### 5 Implementation

#### 5.1 Intermediate representation format

The IR has three-address format, which consists of following instructions:

- *res = arg op arg;*
- *if arg op arg goto L;*
- *L:* (label);
- *goto L.*

**NOTE:** *arg* can be either temporary variable or immediate value (integer or float).

**NOTE:** *if* instruction explicitly has two operands, so if we have expression like

*if (condition)*

, *if* instruction should be represented as

*if condition != 0 goto L...*

.

**NOTE:** when *if* instruction argument cannot explicitly be converted to bool, the result of argument implicitly converted to bool.