Jvav 24 Release

Released July 2024

Jvav 24 is the first major Jvav Release, supports a few syntaxes and features. The versions of Jvav are named by year, e.g. a Jvav version released in 2024 should be called Jvav 24.

This is an early release of Jvav, many if the details are have yet to be finalized, so this is just an introduction to the syntax

1. Compilation unit

In general, a Jvav source file corresponds to a *compilation unit*, and a *compilation unit* contains several members.

Members are:
- Function declaration *(since Jvav 24)*
- Global statement *(since Jvav 24)*

Global statement are any valid statement *(since Jvav 24)*

2. Scope

2.1. General

In a Jvav source file, the outermost scope are called *global scope*. When we declare a function, the scope within the function is called *local scope*.

A *local scope* can be created by surrounding statements with curly bracket syntax.

[Example
```
Complation unit #1:

...

// global scope
{ // local scope
  ...
}
```
 – *end example*]

2.2. Block scope

The following syntaxes introduces a *block scope* that includes statements:
- selection or iteration statements (Section 4.2, Section 4.3)
- compound statement

3. Expressions

3.1. Preamble

An expression is a sequence of operators and operands that specifies a computation. An expression can result in a value and can cause side effects.

3.2. Primary expressions

Primary expression are:
- literal expression

- name expression
- (expression)

3.2.1. Literals

There are several kinds of literals:

- number literal
- string literal
- boolean literal

3.2.1.1. Number literal

3.2.1.1.1. Kinds of number literal

- *binary-literal*
- *octal-literal*
- *decimal-literal*
- *hexadecimal-literal*

For number literals other than decimal, theyall have specific prefixes.

3.2.1.1.2. Binary literal

0b binary-digit

0B binary-digit

binary digit is one of:

0 1

3.2.1.1.3. Octal literal

0 octal-digit

octal digit is on of:

0 1 2 3 4 5 6 7

3.2.1.1.4. Decimal literal

Numbers start with non-zero digit

decimal digit is on of:

0 1 2 3 4 5 6 7 8 9

3.2.1.1.5. Hexadecimal literal

0x hexadecimal-digit

0X hexadecimal-digit

hexadecimal digit is one of:

0 1 2 3 4 5 6 7 8 9

a b c d e f

A B C D E F

3.2.1.2. String literal

“ character-sequence$_{optional}$ *“*

If any line break character appears in *character-sequence*, the string is unterminated, the compiler reports diagnostics.

The backslash(\) is *escape character*, when a backslash is detected in source file, the subsequent character is included into the *character-sequence*, the backslash character and the meaning of the subsequent character is ignored. For example, the quote(") symbol represents the end of *character-sequence* which would not appear in the *character-sequence*, but a backslash before it adds it to the *character-sequence*.

3.2.1.3. Boolean literal

The Boolean literals are the keywords false and true. Such literals have type bool.

3.2.2. Parenthesized expression

A parenthesized expression (*E*) is a primary expression whose type and result are identical to those of *E*. The parenthesized expression can be used in exactly the same contexts as those where E can be used, and with the same meaning, except as otherwise indicated.

3.2.3. Name expression

A name expression holds an identifier that refers to function, variable or a constant.

3.3. Compound expressions

3.3.1. Postfix expressions

3.3.1.1. General

Postfix expressions group left-to-right.

3.3.1.2. Call expression

A call expression is a postfix expression followed by parentheses containing a possibly empty, comma-separated list of expressions which constitute the arguments to the function.

Recursive calls are permitted.

4. Statements

Statements are executed in sequence except where noted elsewhere.

Statements are starts with a keyword, otherwise it is a expression statement. *(since Jvav 24)*

4.1. Block statement

A *block statement* (also known as a compound) groups a sequence of statements into a single statement.

compound-statement:

  { *statement-sequence*$_{\text{optional}}$ }

statement-sequence:

  statement

  statement-sequence statement

A block statement defines a block scope (Section 2.2).

4.2. Selection statements

4.2.1. If statement

If statement executes statements conditionally, if the condition yields *true* the first sub-statement is executed. If the *else* part is present and the condition yields *false*, the second sub-statement is executed.

if condition { statement-true } (1)

if condition { statement-true } else { statement-false } (2)

1. *If* statement without an *else* branch
2. *If* statement with an *else* branch

condition	-	a *expression* which will yield a value of type *bool*
statement-true	-	the *statement* to be executed if the *condition* yields *true*
statement-false	-	the *statement* to be executed if the *condition* yields *false*

4.3. Iteration statements

4.3.1. General

Iteration statements specify looping, and have following syntax:

while condition { statement }

do { statement } while expression

for init-statement$_{optional}$; *condition*; *expression { statement }*

The sub-statement in an *iteration-statement* defines a block scope which is entered and exited each time through the loop.

4.3.2. While statement

While statement executes the sub-statement repeatedly until the value of *condition* becomes *false*. The expression of the *condition* is evaluated before each execution of the sub-statement.

4.3.3. Do-while statement

Do-while statement executes the sub-statement unconditionally first, then executes the sub-statement repeatedly until the value of *condition* becomes *false*. The expression of the *condition* is evaluated before each execution of the sub-statement except the first time.

4.3.4. For statement

For statement executes the sub-statement repeatedly until the value of *condition* becomes *false*, while the statement does not need to manage the loop condition.

The *init-statement* is executed before the first execution of sub-statement, the *expression* is evaluated after each execution of the sub-statement.

4.4. Jump statements

4.4.1. General

Jump statements unconditionally transfer control, has following syntax:

break

continue

return expression

4.4.2. Break statement

A *break* statement shall be enclosed by *iteration-statement*. The *break* statement causes the termination of the *iteration-statement*; control passes to the statement following the terminated statement, if any.

4.4.3. Continue statement

A *continue* statement shall be enclosed by *iteration-statement*. The *continue* statement causes the termination of current loop and immediately starts the next loop if any.

4.4.4. Return statement

A function returns control to its caller by the return statement.

The *expression* of *return* statement is called its operand. A *return* statement with no operand shall be used only in a function whose has no return type. The type of the operand must match the type of the function's return type.

5. Declarations

5.1. Preamble

A declaration is a statement (Section 4)

5.2. Type clause

: identifier

Type clause specifies the type of the declaration, such as the variable's type and the function's return type, and can usually be empty.

5.3. Variable declaration

A variable declaration is a statement that introduces and optionally initialize one identifiers.

let variable-name type-clause$_{optional}$ = *initializer*	(1)
var variable-name type-clause$_{optional}$ = *initializer*	(1)

variable-name	-	the name of the variable, any valid identifier
type-clause	-	possibly empty, the type of the variable
initializer	-	the initial value of the variable, any valid expression

1. Declare a variable with the type and the initializer, the value of the variable is mutable.
2. Declare a constant with the type and the initializer, which the value cannot be changed after declaration.

If the *type-clause* is empty, then the type of the variable is deduced from the initializer.

5.4. Function declaration

A function declaration declares a function in current scope and associates the function's name, parameters and return type.

fun function-name (parameter-list) { statement }	(1)
fun function-name (parameter-list) type-clause$_{optional}$ *{ statement }*	(2)

function-name	-	the name of the function, any valid identifier
parameter-list	-	a list of parameter syntax
type-clause	-	possibly empty, the return type of the function

1. Declare a function with no *type-clause*, which means no return value.
2. Declare a function with a *type-clause*.

The parameters of the function are in the same scope of the *statement*

5.4.1. Parameter syntax

Parameter syntax declares a parameter of a function.

parameter-name : *type-clause*