Руководство для чайников.
Язык программирования Yuhan.

Алена Иванова и Тимур Ахметзянов

December 2024

Содержание

1 Введение

1.1 Руководство

1.2 чайника

1.3 для

1.4 языка

1.5 Yuhan

2 Структура программы

Грамматика программы:

```
<program> ::= {<programBody>}
<programBody> ::= <namespace> | <function> | <structBody>
<structBody> ::= <varDefinition> | <struct>
```

Программа представляет из себя набор пространств имён, функций, структур и объявлений переменных.

3 Встроенные типы данных

В языке Yuhan поддерживаются такие базовые типы как:

- **int** - представляет целое число. Занимает 4 байта (32 бита). Диапазон значений от $-2\ 147\ 483\ 648$ до $2\ 147\ 483\ 647$.
- **char** - представляет один символ в кодировке ASCII. Занимает в памяти 1 байт (8 бит). Может хранить любое значение из диапазона от -128 до 127.
- **bool** - переменная этого типа может иметь значения true и false. Занимает 1 байт (8 бит).
- **float** - представляет вещественное число с плавающей точкой. Занимает 4 байта (32 бита).
- **string** - строка, представляет собой массив элементов типа char.
- **array** - массив.

Между типами int, char, bool и float возможны касты.

3.1 Array

При создании массива надо указать тип данных, которые будут храниться в этом массиве и его размер, который должен быть известен на момент компиляции: array<тип, размер>

4 Объявление переменных

Сначала идет тип данных, далее имена переменных, через запятую. Каждую переменную можно сразу инициализировать: ставится = и пишется выражение, задающее переменную.

Более формально:

```
<varDefinition> ::= <type> <var> {, <var>};
<var> ::= <identifier> | <identifier> = <exp12>
```

<type> - обозначение типа. При использовании невстроенных типов нужно прописывать специальное слово **typename**. При использовании встроеных типов допускается их написание без специального слова **typename**.

<identifier> - идентификатор, имя переменной (для каждой переменной должно быть уникальное имя).

5 Выражения

5.1 Приоритеты операций

1) Скобки (круглые и квадратные), точка (.), пространство имён(::)
2) Префиксные инкремент и декремент, отрицание (!), унарные + и -
3) Умножение (*), деление(/) и взятие остатка от деления(%)
4) Сложение (+) и вычитание (-)
5) Битовые сдвиги (« и »)
6) Сравнение (>, <, <=, >=)
7) Сравнение на равенство (==, !=)
8) Битовый AND (&)
9) Битовый XOR (^)
10) Битовый OR (|)
11) Логический AND (and)
12) Логический OR (or)
13) Присваивания (=, +=, -=, *=, /=, %=, &=, |=, ^=, «=, »=)
14) Запятая (,)

6 Statements and blocks

```
<block> ::= { {<statement>} }
<statement> ::= <if> | <while> | <for> | <switch> | break; | continue; |
<varDefenition> | <return> | <block> | ; | <exp14>
```

7 Условные операторы

7.1 If

```
<if> ::= if ( <exp14> ) <statement> [else <statement>]
```

if выполняет проверку выражения, если оно True, то выполниться первый **statement**, если после if идёт else, то **statement**, относящийся к else, выполняетс только в том случае, когда выражение False.

7.2 Switch

```
<switchItem> ::= case <literal> <block>
<switch> ::= switch ( <exp14> ) { {<switchItem>} [default <block>] }
```

Значение выражения последовательно сравнивается со значениями после оператора case. И если совпадение будет найдено, то будет выполняться определенный блок case и все последующие.
Стоит отметить, что сравниваемое выражение в switch должно иметь тип, у которого определён оператор ==.
В конце конструкции switch может стоять блок default. Он необязателен и выполняется в том случае, если switch н прервался ранее.
Чтобы выйти из оператора switch, в блоке можно поставить оператор break.

8 Циклы

8.1 While

```
<while> ::= while ( <exp14> ) <statement>
```

Цикл while выполняет <statement>, пока его условие(<exp14>) истинно, то есть возвращает True.

8.2 For

```
<for> ::= for ( <инициализатор> ; <условие> ; <итерация>) <statement>
```

Более формально:

```
<for> ::= for ( [(<exp14> | <varDefinition>)]; [<exp14>]; [<exp14>] ) <statement>
```

- **инициализатор** - выполняется один раз при начале выполнения цикла.
- **условие** - представляет условие, при соблюдении которого выполняется цикл.
- **итерация** - выполняется после каждого завершения блока цикла и задает изменение параметров цикла.

8.3 Break и continue

Существуют такие специальные слова break и continue. Они используются для управления временем. С помощью бреак можно прервать время, а континуе принудительно передает управление в управляемое выражение наименьшего заключающего действия.

9 Функции

```
<funcVarDefinition> ::= <type>[&] <identifier> {, <type>[&] <identifier>}
<function> ::= func <type> <identifier> ( [<funcVarDefinition>] ) <block>
```

При объявлении функции надо писать специальное слово **func**, после которого нужно имя функции (должно быть уникальным).
В функцию можно передавать 0 и более аргументов, есть возможность передавать аргуметы по ссылке.

10 Пространство имён

```
<namespace> ::= namespace <identifier> { {<programBody>}}
<programBody> ::= <namespace> | <function> | <structBody>
<structBody> ::= <varDefinition> | <struct>
```

Пространство имён задаётся с помощью специального слова **namespace** и идентификатора, который является именем пространства имён (должно быть уникальным).
Структура тела пространства имён совпадает со структурой программы.

11 Структуры

```
<struct> ::= struct <identifier> { {<structBody>} };
<structBody> ::= <varDefinition> | <struct>
```

Структуры задаютсч с помощью специального слова **struct** и идентификатора, который является именем структуры (должно быть уникальным).
В теле структуры могут быть объявления переменных, в этом случаи они являются полями структуры.

Также в теле структуры могут располагаться структуры, в этом случае внешняя структура выступает в качестве пространства имён.

Примечание: структуры находятся в $\alpha-$тестировании. Пока что выдают UB. Использовать под свой страх и риск