Документация по языку программирования ALC

Чубенко Семён 11И, Тимоша Яснов 11И

27 апреля 2023 г.

1 Пользовательское руководство

1.1 Введение

В пользовательском руководстве мы представляем к ознакомлению инструкцию по использованию языка ALC. В этом блоке мы описали основные особенности его устройста, а также решения проблем, которые могли возникунуть. Для более глубоко понимания рекомендуем ознакомиться с технической документацией и формальной грумматикой.

1.2 Основные парадигмы языка ALC

- Динамическая типизация - принцип, при котором тип переменной назначется не в момент объявления, а в момент присваивания, то есть в одной части программы в **arr** может быть присвоен массив, а вдругой уже целочисленная переменная.
- Отсутсвие глобалной облости видимости. Доступ к переменной ограничен блоком кода, в которой она была объявлена. Это способствет написанию более лаконичного кода.

1.3 Структура программы

Программа разделена на функции, которые вызывают другу друга и выполнюят свою часть кода. В программе всегда должны быть объявлена функция **main**.

1.3.1 Функции

Вся прогрмма состоит из функций, то есть блоков кода, которые могут быть вызваны из другой части програмы и которые также могут вернуть какой-либо результат. Начала описания функции задается ключевым словом **function**. Затем указывается имя функции и в круглых скобках через запятую - передоваемые в блок кода значения. Фигурными скобками ограничивается код для данной функции. Формально: **function** <имя функции> (<аргументы>) { <блок> } После ключегого слова **return** указывается возвращаемое значение.

```
function f(num){
    if (num == 2) {
        return num * 2;
    }
    return num;
}
```

Листинг 1: пример функции

1.3.2 Функция main

Это функция вызывается при запуске программа, иными словами это стратавая точка. Она всегда должна быть опрделена после всех иных функций в коде.

```
    function f(num){ // f() is called from main
        if (num == 2) {
            return num * 2;
        }
        return num;
    }

    function main(){ // program starts from main
        x = 2;
        y = 3;
        print(f(x));
        print(f(y));
    }

Output:
4
3
```

Листинг 2: функция main

1.4 Выражения

Выражением может являться любая последовательность скобок, операторов, переменных, различных литералов и вызовов функций.
Пример выражения, значение которого присваивается в x

```
    x = f(3) + (a - 3 * (b / 2));
```

Листинг 3: выраженеи

1.5 Массивы

В массиве могут быть элемнеты разных типо, например число и строка. Массив объявляется при помощи **[ ]**, внутри которых перечисляются элементы. Размер указывать не нужно, так как он меняться динамически. Пример объявления массива.

```
    x = [1, 3, "2323"];
```

Листинг 4: массив

- функция **append** принимет имя массива, в который добавить элемент, и сам элемент

```
        x = [1, 3, "2323"];
        append(x, "12");
        print(x);

    Output:
    [1, 3, "2323". "12"]

```

Листинг 5: append()

- функция **remove** принимет имя массива, в котором удалить элемент, и индекс элемента

```
        x = [1, 3, "2323"];
        remove(x, 1);
        print(x);

    Output:
    [1, "2323"]

```

Листинг 6: remove()

1.6 Циклы

1.6.1 Цикл while

Цикл while позволяет повтороять одну и ту же часть кода, пока истинно условное выражение. Для использования цикла нужно написать ключевое слово **while**, затем в круглых скобках условное выражение, а далее в фигурных скобках блок кода который будет циклически выполняться.
while (<выражение>) { <блок> }
Пример цикла while

```
    x = 4;
    while(x > 0){
        print(x);
        x = x - 1;
    }

Output:
4
3
2
1
```

Листинг 7: while

1.6.2 Цикл for

Цикл for позволяет пройтись по элементам контейнера. Для использования нужно написать ключевое слово **for**, затем в крруглых скобках указывается итератор, а после ключевого слова **in** - контейнер для перебора. Далее в фигурных скобках блок кода который будет циклически выполняться.
for (<имя переменной> in <контейнер>) { <блок> }
Пример цикла for

```
    arr = [1, 2, 3, 4];
    for (i in arr){
        print(i);
    }

Output:
1
2
3
4
```

Листинг 8: for

1.7 Условный оператор if

Логический оператор позволяет ограничить выполение кода в заимисмотсти от значние выражения. То есть, если условное выражение принимает значение истина, то блок кода внутри if выполниться, в проивном случае он будет пропущен. Для использования условного оператора нужно написать ключевое слово **if**, затем в груглых скобках написать условное выражение. Далее в фигурных скобках указывается блок кода.
Пример if

```
    x = 4;
    if (x >= 4){
        print("in first if");
    }
    if (x < 2) {
        print("in second if");
    }

Output:
```

```
in first if
```

Листинг 9: if

Продолжением идеи условных операторов является использование их в связке с **else** и **else if**. **else** позволяет указать блок кода, который выполниться, если условное выражение приняло значение ложь. **else if** позволяет в случе ложного значние одного условного выражение проверить другое условное выражение.
Пример конуструкции из **if**, **else** и **else if**

```
    if (x >= 4){
        print("in first if");
    } else if (x < 2) {
        print("in second if");
    } else {
        print("in else");
    }
```

Листинг 10: else if

2 ЧаВо

- **ПОЧЕМУ НЕ ЗАПУСКАЕТСЯ ПРОГРАММА???**
  Мы специально для такой неприятной ситуации реализовали вывод в консоль контекста для возникшей ошибки. Внимательно ознакомьтесь с ним.

- **Почему не могу обратиться к переменной?**
  В языке ALC область видимиости жёстко ограничена фигурными скобками. Из внешней области переменные видны, но в обртаную сторону доступ будет отсутствовать.

3 Формальная грамматика

```

<program> ::= {<function>} | function main() <block>
<function> ::= function <name> ( <arguments> ) <root block>
<root block> ::= "{" {<statement>} [return <name>] "}"
<block> ::= "{" <statement> {<statement>} "}"


<statement> ::= <exp> ;
<value exp> ::= <value> | <function call> | <container>


<operand> ::= <name> | <num> | <string> | <function call> | <list element>
<value> ::= <priority 1>


<priority 1> ::= <priority 2> {<operation 1> <priority 2>}
<operation 1> ::= ||
<priority 2> ::= <priority 3> {<operation 2> <priority 3>}
<operation 2> ::= &&
<priority 3> ::= <priority 4> {<operation 3> <priority 4>}
<operation 3> ::=  == | !=
<priority 4> ::= <priority 5> {<operation 4> <priority 5>}
<operation 4> ::=  <= | >= | > | <
<priority 5> ::= <priority 6> {<operation 5> <priority 6>}
<operation 5> ::= + | -
<priority 6> ::= <priority 7> {<operation 6> <priority 7>}
<operation 6> ::= * | / | // | %
<priority 7> ::= <operation 7> <priority 8> | <priority 8>
<operation 7> ::= - | !
<priority 8> ::= <operand> | (<priority 1>)


```

```
34
35 <function call> ::= <name> ( <arguments> )
36 <arguments> ::= <value exp> { , <value exp> }
37
38 <container> ::= <string> | <list> | <range> | <temporary list>
39 <range> ::= range ( <operand> , <operand> )
40 <string> ::= "  <symbol><string> | <num><string> "
41 <name> ::= <symbol> <string>
42 <symbol> ::= ['a'...'z' | 'A'...'Z']
43 <numb> ::= ['0'...'9']
44 <list> ::= <name>
45 <list element> ::= <name> "[" <name> | <list element> | <num>"]"
46 <temporary list> ::= "[" <name> | <list element> | <num>"]"
47
48 <exp> ::= <variable declaration> | <function call> | <special operators> | <
       conditional special operators> | <assign> ";"
49
50 <assign> ::= <name> = <value exp>
51
52 <special operators> ::= <input operator> | <output operator>
53 <input operator> ::= read(<input arguments>)
54 <input arguments> ::= <name> | <list element>
55 <output operator> ::= print(<arguments>)
56 <return> ::= return <name> ;
57
58
59 <conditional special operators> ::= <for> | <while> | <if>
60 <if> ::= if ( <value exp> ) <block> [<else>]
61 <else> ::= else <block> | else <if>
62 <for> ::= for ( var <name> in <container> ) <block>
63 <while> ::= while ( <value exp> ) <block>
```