IRIs Language Reference Manual

xl2784 st3174
hg2498 pt2508

October 15, 2018

Contents

1 Language Tokens and Basic Rules

The token of IRIs has 5 categories: operators, identifiers, literals, key words and comments. This chapter will introduce each kind of IRIs tokens and some basic rules of IRIs.

1.1 Operators

IRIs has the following 5 basic arithmetic operator:

```
+ - * / %
```

They stand for plus, minus, times, divide and mod respectively. Times, divide and mod has the highest precedence and then the plus and minus. IRIs will not conduct auto-casting, so you have to make sure each side of the operator has the same data type or there will errors. IRIs does not have = for assignment, instead, **pipe(|)** takes the functionality of assignment. Different from = which is right to left associative, **pipe(|)** is actually left to right associative and in IRIs, we recommend to write the 2 parts of an assignment in the precedence and following line of a **pipe(|)** . For example:

```
5
|
a
```

will assign 5 to a.

```
3
|
+2
|
a
```

will first calculate 3+2 and assign the result to a. You can also add different variable in this way. You can use parentheses to change the precedence of an expression, but you have to write this in a single line:

```
2
|
*(1+2)
|
a
```

IRIs also has some logic operators:

```
|| && > < >= <= != = !
```

They stand for or, and, greater, less, greater or equal, less or equal, not equal and not. Since IRIs does not use = as assign operator, here just a single = stands for equal.

1.2 Identifiers

In IRIs, identifiers is used to identify variables and functions. A valid identifier should start with a letter from a to z or A to Z and followed by any English letters or numbers and underscore(_). Identifiers in IRIs are case sensitive and should be unique within its scope. For variables declared within a function, the scope will be the whole function. For variables declared in a loop or if statement, the loop or if statement will be the scope. For variables declared our side a function, the scope will be global.

1.3 Literals

IRIs has 5 kinds of Literals: integer, float, string, list and bool. An integer literal consists of only numeric numbers but can start with arbitrary number of 0. The leading 0 will be omitted. A float literal consists of an optional integer part, a decimal point and an optional fraction part. But the two numeric parts are not optional simultaneously. A string literal should be within of two quotes. A valid string can consists of ascii code from 0 to 127. If you want to contain a quote in a string, you have to use a **backslash(\)** as an escape character. A bool literal will use the key words **true** and **false**. A list literal will be introduced in chapter 2.

1.4 Keywords

There are totally 14 key words in IRIS:

```
int float string bool if else elif fi while return end continue break Siri
```

Each will be introduced in the following chapters.

1.5 Comments

There is only one kind of comments in IRIs, that is start with * and end with *\. The comment can be multiple lines and cannot be nested.

2 Data type

2.1 Integer

2.1.1 Declaration

An integer variable is declared with an **int** keyword, and an value can be assigned with a pipe: | operator

```
6
|
int var
```

Declaration of multiple integers

```
1, 2, 3
|
int var1, var2, var3
```

2.2 Float

2.2.1 Declaration

A float variable is declared with an **float** keyword, and an value can be assigned with a pipe: | operator

```
6.0
|
float var

1.0, 2.0, 3.0
|
float var1, var2, var3
```

2.2.2 Operation

$+$, $-$, $*$, $/$, operations are supported for float numbers.

2.3 String

String is immutable. Once created, it can?t be updated.

2.3.1 Declaration

A string variable is declared with the keyword **string**.

```
 "hello world"
|
string var

"hello", "world", "haha"
|
string var1, var2, var3
```

2.3.2 Operation

- Concatenation of two strings (str1+str2)
- Get the length of a string (len(my_string))

2.4 Tuple

A tuple is a set of elements. The type of elements in a tuple can be different, they can be integers, floats, strings, lists and booleans. Tuple is immutable. Once created, it cannot be updated.

2.4.1 Declaration

```
(1,2,3,"hello","world")
|
tuple a

(1,2,3,"hello","world"), (1,2,3), ("hello", 3, 10)
|
tuple a, b, c
```

2.4.2 Operation

- Access to an element of a tuple: a[0] (the first element), a[1] (the second element)
- Get the length of a tuple: len(a) (get the length of the tuple named "a")

2.5 Boolean

2.5.1 Declaration

A boolean variable is declared with the keyword **bool**. A boolean variable can a value of True or False.

```
true
|
bool var

true, false
|
bool var1, var2
```

2.6 List

List is a mutable set of the same type of elements. The type of members of a list can be int, float, string or a list of a certain type.

2.6.1 Declaration and initialization

To define a list, you should use type[]

```
int[]
|
li
```

Then li will be a list of integers. You can initialize the list when declaring it.

```
int[1,2,3]
|
li2
```

Then li2 will be [1,2,3]. You can also create a list of list. The inner list type has to be declared.

```
int[int[]]
|
li2
```

You have to declare the type of element of the list if you declare a list. If you want to initialize a list of list, you should do:

```
int[int[1,2],int[3,4],int[5,6]]
```

```
|
li3
```

2.6.2 Operation

List Concatenation Two lists can be concatenated by the + operator.

```
int[1, 2, 3]
|
li1
int[4, 5, 6]
|
li2
|
+l1
|
li3
```

li3 will be [4,5,6,1,2,3].

Append elements to list One can append a list with the **append** function.

```
int[1,2,3]
|
li
append(li, 4)
```

Then li will be [1,2,3,4].

Pop elements from list One can pop out elements from a list with the **pop** function.

```
int[1,2,3,4,5]
|
li
pop(li)
|
ele
```

li will be [1,2,3,4] and ele will be 5.

Length Get the length of a list with **len** function.

```
int[1,2,3,4,5]
|
```

```
li
len(li)
|
a
```

a will equal to 5.

Element Accessing Element in the list can be retrieved with [**id**].

```
int[1,2,3,4,5]
|
li

li[2]
|
int ele
```

The variable ele should be 3.

3 Control Flow

3.1 If-else

The if -else statement is used to express decisions. The syntax of If-else in our language is

```
if (expression)
   (statement)
elif (expression)
   (statement)
else
   statement
fi
```

The end of if -else is denoted by the keyword fi, and the elif and else are optional. Each if-else statement can have multiple elif but can only have zero or one else. The expression is evaluated; if it is true (expression has a non-zero value), statement is executed.

3.1.1 Nested if-else

Because each if-else statement need to be ended with an fi keyword, there is no ambiguity if the else statement is omitted in the inner if-else statement.

3.2 while loop

While is the only loop implementation in our language. The syntax of while in our language is

```
while (expression)
   (statement)
endwile
```

The expression is evaluated first; If it is true, statement is executed, and then the expression is re-evaluated. The cycle continues until the expression is False.

3.3 Continue and Break

Continue and break can only be used in while loop.The continue keyword causes next iteration of the while begin.

```
-10
|
int x

0
|
int a

while (x<3)

   3
   |
   +x
   |
   x

   if (x == -1)
      continue
   fi

   x
   |
   +1
   |
   a

endwile
```

For the program above, the result a is $(-7) + (-4) + 2$

A break causes the innermost while loop to be exited immediately.

```
-10
|
int x

0
|
int a

while (x<3)

   3
   |
   +x
   |
   x

   if (x == -1)
      break
   fi

   x
   |
   +1
   |
   a

endwile
```

For the program above, the result a is $(-7) + (-4)$

4 Standard Input and Output

4.1 Input

The input mechanism is to read one string at a time from the standard input, normally the user input, with input():

```
input(datatype)
   |
(datatype) var
```

Programmers should manually set the only argument of input function specifying the datatype for the return value of the function. Below are examples for input in all formats. Notice that we can only read input as integer, float and string.

```
input(int)
|
int a

input(string)
|
string a

input(float)
|
float a
```

Notice that if you have already declared the variable in the previous statement, you can also use input()

```
int a

input(int)
|
a
```

4.2 Output

The output function showresult() concatenates string variable and other strings and then prints its arguments on the standard output.

```
"hello"
 |
 string s1
 |
 + "world"
 |
 string s2
 showresult(s2)
```

Then "hello world" will be printed.

5 Function

In IRIs, every program should have a main function and will start at main function. main can accept arguments from standard input and return 0 if exits without exception. In IRIs, you can declare a function in such a form:

```
type fun_name([para_list])
function body
return value
Siri
```

For example,

```
int fun(int a)
   int b

    a
    |
    +1
    |
    b

    return b
    Siri
```

This function will increment a by 1 and assign its value to b and return b. Any IRIs statement can be used in a function. But IRIs does not support nested function which means you cannot declare a function in a function. In IRIs, each function must have a return value. When reached to a return keyword, the function will terminate immediately. To call a function, you can just use: fun_name(a,b,c...). For example,

```
fun(5)
|
c
```

c will be 5 after execution. You can call a function inside a function.