CSc 372 - Project 2

Trong Nguyen - Anh Nguyen Phung

Language Design

- When we design the language, we want to minimize the amount of symbols as much as possible to make our language become more accessible to people who are not familiar with programming.

- Besides the basic types such as alphabetical and digit characters, we also allow to use symbols in non strings and non comments which are " (for string), # (for comment), space, tab, and new line (for usual code writing).

- If the users type in any inappropriate symbols or syntax in any block of code, our language will check and report the specific errors at exact line to the users - which will help the users to debug much easier.

- In our language, basic comparators and comparisons such as $+$, $-$, $\geq$, etc will be replaced with keywords.

- Each line of code will need to be on separate lines (our language does not require the users to do indentation like Python but it will help to improve code readability).

Overcoming challenges

When we design the language, we had a problem when we do the translator to translate code from our language to Java.

To be specific, we support two expressions which are int expression and boolean expression in our language. We have two steps when translating code from our language to Java, the first step is to check whether the given expression is a valid expression or not and if it is correct then we will move to the second step which is to calculate to value of the expression.

The challenging part is at the first step when we found it quite difficult to validate whether the given expression is correct or not. To overcome this problems, we tried to break the integer expression and boolean expression in our language into small parts and then analyze the common features of the correct expression. After that, we write functions utilizing recursions to check small parts of the given expressions to see if they are matched with the corrects ones or not and if it is totally correct then we will move those expressions to the next step.

Language Tutorial

How to run the program in our language ?

Suppose that the users want to write a program called Program.txt in our language. To compile and run, the users will need to run the following commands in the terminal:

- javac Translator.java to compile the language.

- java Translator Program.txt to convert the program from our language into Java. If there is any parsing errors, the error message will be printed out to the terminal with specific description at exact line of code.

- javac Program.java to compile the program in Java.

- java Program to run the program with any command line arguments. For example, java Program 3 4 5

Language Tutorial

Grammar

Basic Syntax

Comment

In our language, comment starts with #. After # can be any ASCII characters except for ''.

Integer Expression

Our language supports add (+), sub (-), mult (*), div (/), mod (%), negate(-). There should be a space before and after each keyword, except for negate that only needs the space after.

All operators are left-associative, with order of highest to lowest precedence (from left to right) and any operations that have equal that have equal precedence are placed in {}.

negate, {mult, div, mod}, {add, sub}.

Example of valid integer expression:

```
2 add 3 mod 2
3 sub 2 mult 4
negate 4 div 2
```

Language Tutorial

Grammar

Boolean Expression

Our language support and (&&), or (||), not(!). For comparators, our language supports gt (>), lt (<), gte (>=), lte (<=), equal (==), diff (!=). There should be a space before and after each keyword, except for not (!) which only needs a space after.

All operators are left-associative, with order of highest to lowest precedence (from left to right) and any operations that have equal that have equal precedence are placed in {}.

{<int_expr> <comparator> <int_expr>, not}, and, or.

Note: Our language does not have comparison between boolean values. So something like 2 gt 3 equal true is invalid in our language.

Something such as not 2 gt 3 is invalid in our language.

Example of valid boolean expression:

```
2 lt 3 and 3 mod 1 equal 0
not x                          # with 'boolean x assign true'
2 diff 3 or true
```

Language Tutorial

Grammar

Declaring variable

In our language, you can declare of type integer and boolean. The name of a variable (non-command line argument variable) should only contain alphabetical characters (both upper case and lower case).

```
integer a assign 5
integer B assign 2 add 3
boolean X assign true
boolean y assign 2 gte 3
```

You cannot re-declare the variable. For example, the code below will be considered invalid in our language:

```
integer a assign 5
integer a assign 4
```

```
boolean a assign true
integer a assign 4
```

Language Tutorial

Grammar

Reassign variable

In our language, you can reassign the value of a variable if that variable is already declared. If reassigned, the value must be the same type as the declared type. Moreover, you can not reassign the value of a variable if that variable is not declared first.

Example of valid variable reassignment:

```
integer x assign 3      # int x = 3
x assign 4              # x = 4
x assign x add 1        # x = x + 1
```

Print

In our language, you can print either boolean/integer expressions (including variables) and strings. #print for printing without new line and printout for #printing with new line

```
printout 2
print true
printout "Hello World!"
printout 2 add 3 sub 5
print 2 lt 3
```

Language Tutorial

Grammar

If/else conditionals

Examples of valid if/else conditionals:

```
if 2 lt 3
  print "2 less than 3"
end
```

```
if 2 lt 3
  print "2 less than 3"
else
  if 2 equal 2
    printout "Hello!"
  end
end
```

Language Tutorial

Grammar

Loops

Examples of valid loops:

```
integer i assign 0
during i lt 2          # while i < 2
  printout i
  i assign i add 1     # i = i + 1
end
```

```
integer i assign 0
during i lt 2          # while i < 2
  if i equal 1
    printout "World!"
  else
    printout "Hello"
  end
  i assign i add 1     # i = i + 1
end
```

Language Tutorial

Grammar

Command line arguments

In our language, to use command line arguments, you have to specify how many command line arguments you want to use at the beginning of the file. For example, if you want to use 3 command line arguments, at the beginning of the file, you should have this line of code:

```
use 3 cmd args
```

Note: You have to follow the exact syntax above with a single space between keywords and the number, otherwise there would be a parsing error.

Then you can use 3 command line arguments arg0, arg1, arg2 as variables directly in your code. The name of a command line argument variable starts with arg followed by a number indicating the index.

For example, suppose you want to read in one command line argument and print out the square of that command line argument, then the desired program should be:

```
use 1 cmd args
printout arg0 mult arg0
```

Expanding language

Ideas for expansion of language

If we can expand our language further, there will be more improvements that we want to add to our language:

- Support other types besides integer and boolean such as float, etc and other mathematical operations such as sin, cos, tan, ...

- Support parentheses in our language. We currently do not support brackets and sometimes it could be inconvenient for the users when they have more complex expression so we hope to support the parentheses in our expansion language.

- Support data structure such as array, dictionary, list, etc so that the users would find it more comfortable to store data, information or when they need to deal with loops and indexing.