Basic BASIC Language Specification

Maddy Wu Amir Estejab

May 20, 2024

Introduction

As student's that took intro to CS at Williams, we learned how to program with Python. There were, of course, many benefits to learning how to code this way. For example, Python syntax, at the level that we were learning it, was not difficult to learn. However, there were also many challenges – especially for individuals that have never coded before or have only had experience coding with languages like Scratch. One of the main challenges that we came across as students in 134, was trying to gain a solid foundational understanding of the specifications in a language. Python is obviously a large language with many specifications which is daunting to people that have never programmed before.

BASIC is a programming language that was originally designed to remedy this problem. Designed by John G. Kemeny and Thomas E. Kurtz at Dartmouth College in 1963 as a way to make it easy for non-STEM students to learn how to code. For our project, we are looking to create an even simpler version of the language that will hopefully further lower the barriers of entry to learning how to the program and really reinforce what we believe to be the foundations of programming.

Design Principles

Because Basic BASIC is meant to be accessible even to most non-technical of users, both our primitive types and our combining forms will try to resemble plain English as much as possible. Furthermore, Basic BASIC will have even fewer specifications than BASIC which will hopefully be conducive to ease of learning and a stronger understanding of "foundational" concepts. If the user chooses to continue programming. BASIC Basic will lend itself nicely to other more complex languages.

Examples

1. test1.bbas:

```
10 PRINT 2^3
20 PRINT "That was easy!"
30 PRINT
```

   Run "dotnet run test1.txt" in the project file. The output should be the following:

```
8
```

2. test-2.bbas: Factorial without INPUT

```
LET num = 5
LET factorial = 1
FOR i = 1 TO num
    LET factorial = factorial * i
```

```
NEXT i
PRINT "The factorial of "; num; " is: "; factorial
```

Run ”dotnet run test2.bbas” in the project file.

```
"The factorial of 5 is 120"
```

3. test-3.bbas: Factorial with INPUT

```
INPUT "Enter a number: ", num
LET factorial = 1
FOR i = 1 TO num
    LET factorial = factorial * i
NEXT i
PRINT "The factorial of "; num; " is: "; factorial
```

Run ”dotnet run test3.bbas” in the project file. The user will then be prompted for a number.
In this case assume they inputed 5

```
"The factorial of 5 is 120"
```

Language Concepts

In order to use BASIC Basic the user should have an understanding of basic (haha) math operations. They also need to have some an understanding of strings and print statements. Strings are a primitive data type in this language meaning they cannot be broken down into constiuent parts. The print statment as well as the arithmetic operators and comparison operations, on the other hand, are combining forms.

In Basic BASIC we also have conditionals and GO TO statements. Although GO TOs are technically considered bad practice when coding, we chose to implement them because it is the most simplistic way of introducing control flow in programs. It allows the user to think more like a computer, by stepping through a program the same way that a computer would, without any previous or advanced programming knowledge.

Formal Syntax

```
<Expr>         ::= <Statement>
                 | <Statement> <Expr>
<Statement>    ::= <Assignment> | <PRINT> | <Conditional> | <Arithmetic> | <Comparison>
<Assignment>   ::= <Var> '=' <Primitive>
<Var>          ::= [a-zA-Z]+
<Primitive>    ::= <Bboolean> | <Bstring> | <Num>
<Bboolean>     ::= True | False
<Bstring>      ::= " "
<Num>          ::= n\in\Z
<PRINT>        ::= PRINT <Expr>
<Arithmetic>   ::= (<Expr> + <Expr>) | (<Expr> - <Expr>) |
                   (<Expr> * <Expr>) | (<Expr> / <Expr>) |
                   (<Expr> ^ <Expr>)
```

```
<Comparison>  ::= (<Expr> == <Expr>) | (<Expr> != <Expr>) |
                  (<Expr> < <Expr>)  | (<Expr> <= <Expr>) |
                  (<Expr> > <Expr>)  | (<Expr> >= <Expr>)
<Conditional> ::= IF <Comparison> THEN <Statement>
```

Semantics

Semantics				
Syntax	Abstract Syntax	Type	Prec./Assoc.	Meaning
”s”	Bstring of string	string	n/a	A sequence of characters enclosed in double quotes (”). It is a primitive
true/false	Bbool of bool	string	n/a	Truth values indicating whether the result of a comparitive operation is true or false. It is a primitive
n	Num of int	int	n/a	N is any positive or negative integer. It is a primitive.
x	Var of string	int	n/a	N is any positive or negative integer. It is a primitive.
=	Assignment of string * Expr	char	n/a	Assigns the value of the right expression to the value of the left expression
`==, <, <=, >, >=, !=`	EqualsEquals, Less, LessEqual, Greater, GreaterEqual of Expr * Expr	char	left	Compares the value of the right expression to the value of the left expression and returns a boolean.
+	Plus of Expr * Expr	char	left-associative	Adds the value of the left expression to the value of the right expression.

-	Minus of Expr * Expr	char	left-associative	Subtracts the value of the right expression from the value of the left expression.
*	Times of Expr * Expr	char	left-associative	Multiplies the value of the left expression to the value of the right expression.
/	Divide of Expr * Expr	char	left-associative	Divides the value of the left expression by the value of the right expression.
ˆ	Exp of Expr * Expr	char	right-associative	Raises the value of the left expression to the value of the right expression.
()	Parens of Expr	char	n/a	parentheses
GO TO	GoTo of Expr * Expr	string	n/a	Control flow statement that directs the flow of the program to a certain line number.
IF THEN	IfThen of Expr * Expr	string	n/a	An extension of a control statement. Operates similarly to a conditional move in C. Only if a certain condition is met, will the flow of the program change.

1. What are the primitive kinds of values in your system? For example, a primitive might be a number, a string, a shape, or a sound. Every primitive should be an idea that a user can explicitly state in a program written in your language.
   (a) The primitives we have are strings, numbers, and booleans
2. What are the combining forms in your language? In other words, how are values combined in a program? For example, your system might combine primitive "numbers" using an operation like "plus." Or perhaps a user can arrange primitive "notes" within a "sequence."
   (a) The combining forms we have are your typical arithmetic operators like addition, subtratction, multiplication, division, and exponentiation. We also have comparison operators like ==, <, <=, >, >=, and !=
3. How is your program evaluated? In particular
   (a) Do programs in your language read any input?
      i. Our language does take in user input. As of right now our program does not take in user input, but at some point it is something that we would like to implement as we believe that it will make programming with Basic BASIC more dynamic.
   (b) What is the effect (output) of evaluating a program? Does the language produce a file or print something to the screen? Use one of your example programs to illustrate what you expect as output.
      i. Depending on the program that the user writes, Basic BASIC should be able to interpret any program written in the language and output what the user is hoping to output. The most complex example we have provided is a game called STM which should generate a playable game.

Remaining Work and Limitations

Unfortunately, we were not able to implenet conditionals by the final submission. On our first try, it kept trying to parse ”IF” as a variable. We were able to reconcile this with the addition of things called ”keywords.” Examples of keywords are: PRINT, GO TO, IF, and THEN, words that should not be allowed to be used as variable names as they have other meanings and uses. However, after we got to this point, we realized that we also had to implement comparison operations and then implement the parsing of those into our conditional statements.

We would eventually like to implement conditionals as we do believe that it would enhance the capabilities of our language, we just, unfortunately able to get to it during this iteration.

In later versions of Basic BASIC we would also like to have an interpreter that would be able to handle programs like STM which is a memory game that was introduced in Games for TI-99/4A. To create an interpreter that would be able to run the STM program, we would also need to implement FOR NEXT, which operates similarly to a for loop in python, CALL