Language Description: Gurt©

Gurt©

By Robert Wood

York College of Pennsylvania

CS340 – Programming Language Design

Professor Dean Zeller

Fall 2023

Wednesday, December 6, 2023

Table of Contents

Preface

Foreword By Robert Wood

Welcome to Gurt, an introductory programming language designed to make coding simpler for young students to understand. As the world today continues to advance its technologies, it is critical that the next generation are introduced relatively early to the basics of programming.

Gurt is intentionally designed to be straightforward and easy to grasp, making it a great starting point for young students new to programming. Because of its simple syntax, Gurt provides a very basic but wide-open door into the world of problem solving and logical thinking.

Throughout this introduction, one will see how Gurt's low-complexity design and simple syntax lay out a solid foundation for young students to develop basic programming skills. The skills that students will acquire through learning and programming with Gurt will enable them to broaden their horizons to more complex programming languages such as Java, C#, C, and so forth.

Buckle in and enjoy the ride as you discover how Gurt can inspire the innovators of tomorrow!

Contributions

The author would like to recognize the following people for their role they played in the creation of Gurt:

- Original template and assignment requirements by Dean Zeller.
- Interpreter design by CodePulse on YouTube

Language Description

History

Gurt is an original, upcoming programming language that was initially designed in the Fall of 2023. The inspiration behind the name of the language is straightforward: I have always joked with my friends about having a pet goose named Gurt when I was older. Seeing as this has always been a joke and not a real goal for me, I decided what better way to put the joke to use than to create a language that has the name, with its logo being a goose, that I can use to develop real world applications.

Description

The functional programming language Gurt is a language based off of a mix of the core concepts behind the design of JavaScript and Python. Gurt can currently be used for very basic programs and has been designed with the goal of being used as a tool to introduce those new to programming to the core concepts such as data types, loops and branches, and functions.

Language Notes

The following are notes regarding the Gurt language.

- Comments are single line, denoted by the double forward slash (//) symbol. // is not included in the cono-table, as it is handled during the tokenization process.
- Variables are defined first in the program file, one per line using the following keywords:
    - `int (integer values)`
    - `flt (single floating-point values)`
    - `dbl (double floating-point values)`
    - `str (string of characters)`
- The supported data types include integers, single floating-point values, double floating-point values, and strings.
- Functions may be defined anywhere in the program at the leftmost level on the same line.
- Recursion is <u>not</u> available.
- Function definitions require the keyword `define`. Function calling is simply the name of the function, followed by parentheses with or without parameter values.
- Return statements are not available in the language but may be implemented later.
- The main program is defined and runs anywhere outside of any functions.
- Use of `open` (`{`) and `closed` (`}`) curly braces to define the beginning and end of the logic block for functions
- Mathematics is calculated left to right, and colons (**:**) are interpreted as parentheses for order of operations.
- A newline (`\n`) denotes the end of a line of programming, similar to python
- Assignment is completed using the equals = operation. This is similar to many other languages. For example: `dbl sum = dbl1 + dbl2`
- Within the functions and main program, the only use of parentheses is in function calls, input statements, and print statements.
- The logical keywords `and`, `or`, and `not` are available for use. Examples:
    - `If a not b then return 0`
    - `If a and b then return 0`
    - `If a or b then return 0`
- Arrays are not available, however lists are using the square bracket symbols `[` and `]`
- The `print` statement can take multiple arguments, which must be a variable, literal, or string. Example:

- print(“Hello World!”)
- print(variable)
- print(400)

Features to Add

The following features could conceivably be added to future versions of the Gurt compiler.

- Defining multiple variables on a line
- Return statements
- Multi-line functions
- Clearer syntax
- Char and byte datatypes
- Recursion
- End-of-line marks the end of a programming line, such as semi-colons
- Arrays
- Multiple arguments and literals for print statement

Example Programs

Program 1 – Mathematics

```
// display math operations with integers
int integer1 = 5
int integer2 = 5
int integer3 = 2
int intResult = :integer1 + integer2: * integer3

// display math operations with doubles
dbl double1 = 5.5
dbl double2 = 4.5
dbl double3 = 4.5
dbl doubleResult = :double1 + double2: / double3

// output the results to the console
print(intResult)
print(doubleResult)
print("Ending Program")
```

Program 2 – Branches and Loops

```
// initialize num1 and sum variables
print(“Enter a number: “)
int num1 = inputInt()
int sum = 0

// while sum is less than 25
while sum <  25 then print(num1) int sum = sum + 1
if num > 25 then int num = num + 4 else int num = 0

str stringVar = "Final value in sum: "

// output result to console
print("Final value in num1:")
print(num1)
print(stringVar)
print(sum)
```

Program 3 – Functions

```
    // function that calculates the sum of two given integers
    define calculateSum(num1, num2) { int sum = num1 + num2 }

    // function that calculates the quotient of two doubles
    define calculateQuotient(dividend, divisor) { dbl quotient =
dividend / divisor }

     // initialize two integer variables and calculate their sum
    int x = 2
    int y = 8
    int sum = calculateSum(x, y)
    // initialize two double variables and calculate their quotient
    dbl top = 2.0
    dbl bottom = 0.155
    dbl quotient = calculateQuotient(top, bottom)

    // print results to console
    print("Sum:")
    print(sum)
    print("Quotient:")
    print(quotient)
}
```

Bachus-Naur Form

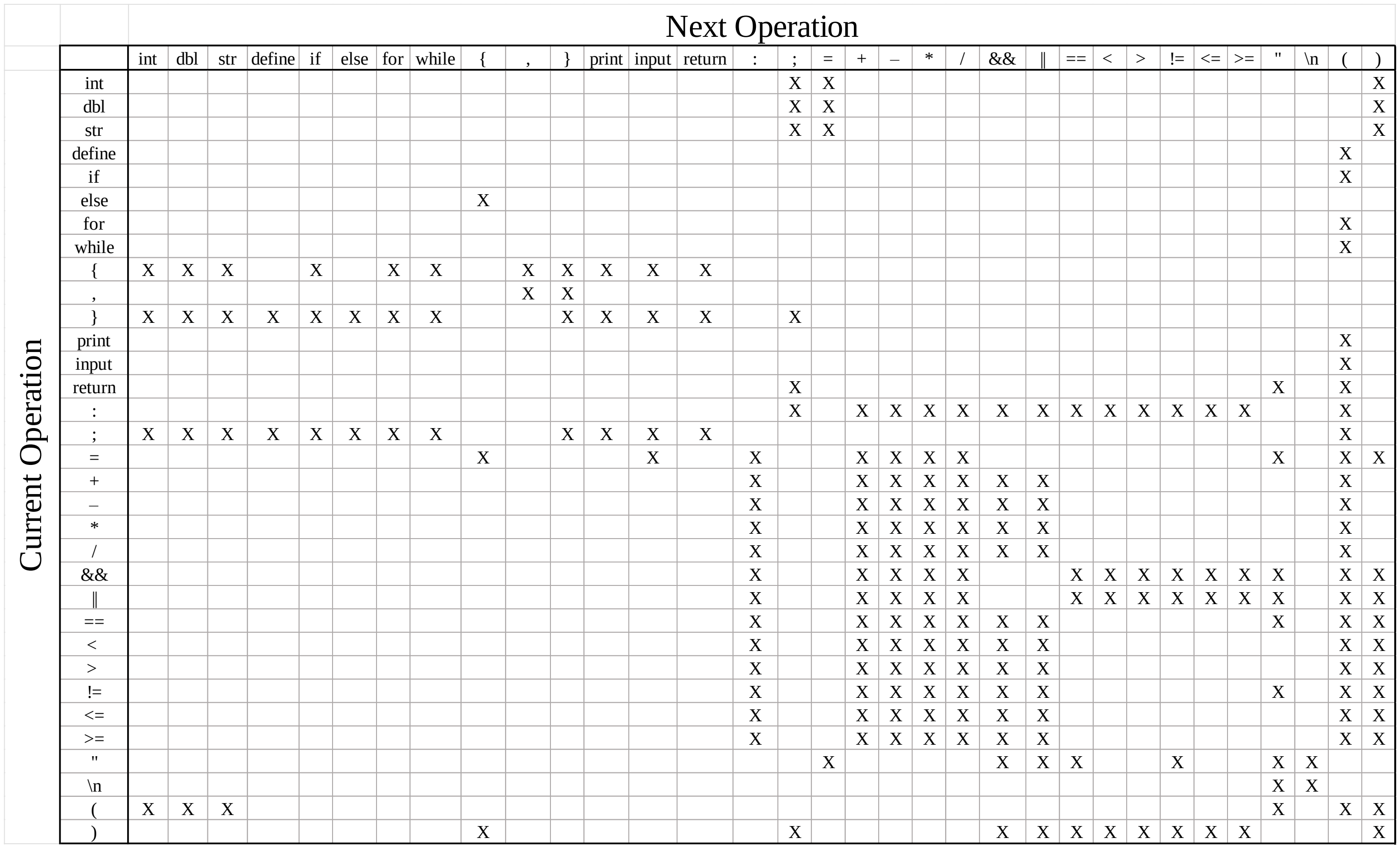
The following is the Bachus-Naur form to define the language Gurt.

```
<program>        ::= <variables> <functions> <statements> |  <functions>
                     <statements> <variables>  |  <statements> <functions>
                     <variables>  |  <statements> <variables> <functions>  |
                     <variables> <statements> <functions>  |  <functions>
                     <variables> <statements>

<variables>      ::= <variable def>  |  <variable def> <variables>

<variable def>   ::= int <symbol> = <operand>    |  dbl <symbol> = <operand>    |
                     str <symbol> = <operand>    |  int <symbol> = <function call>
                     |  dbl <symbol> = <function call>    |  str <symbol> =
                     <function call>

<functions>      ::= <function def>  |  <function def> <functions>

<function def>   ::= define <symbol> ( ) { <statements> }  |  define <symbol> (
                     <parameters> ) { <statements> }
<parameters>     ::=  <symbol>

<symbol>         ::= <letter>  |  <letter> <letterdigits>

<letterdigits>   ::= <letter> <letterdigits>  |  <digit> <letterdigits>

<letter>         ::= a  |  b  |  c  |  d  |  e  |  f  |  g  |  h  |  i  |  j  |
                     k  |  l  |  m  |  n  |  o  |  p  |  q  |  r  |  s  |  t  |
                     u  |  v  |  w  |  x  |  y  |  z  |  A  |  B  |  C  |  D  |
                     E  |  F  |  G  |  H  |  I  |  J  |  K  |  L  |  M  |  N  |
                     O  |  P  |  Q  |  R  |  S  |  T  |  U  |  V  |  W  |  X  |
                     Y  |  Z  |   _

<number>         ::= <digit>  |  <digit> <number>

<digit>          ::= 0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8  |  9

<statements>     ::= <statement>  |  <statement> <statements>

<statement>      ::= <assign stmt>    |  <if stmt>  |  <for stmt>  |  <while stmt>
                     |  <print stmt>    |  <function call>    |  <return stmt>

<assign stmt>    ::= <symbol> = <expression>

<expression>     ::= <operand>  |  <operand> <operation> <expression>

<operand>        ::= <symbol>  |  <literal>  |  "<letter>"

<operation>      ::= +  |  -  |  *  |  /

<if stmt>        ::= <if>  |  <if-else>

<if>             ::= if ( <condition> ) { <statements> }

<if-else>        ::= if <condition> then <statements> ifagain <condition> then
                     <statements> else <statements>

<condition>      ::= <operand> <comparison> <operand> | ( <operand> <comparison>
                     <operand> )

<comparison>     ::= ==  |  !=  |  <  |  >  |  <=  |  >=  |  and  |  or  |  not
```

```
<for stmt>       ::= for <symbol> <condition> <expression> then <statements>
<while stmt>     ::= while <condition> then <statements>
<print stmt>     ::= print ( <operand> )
<function call>  ::= <symbol> ( )  |  <symbol> ( <parameters> )
```

CONO Table

The following is a screenshot of an Excel spreadsheet storing the cono table for Gurt.



Current Operation	Next Operation																																
	int	dbl	str	define	if	else	for	while	{	,	}	print	input	return	:	;	=	+	–	*	/	&&	‖	==	<	>	!=	<=	>=	"	\n	(	)
int																X	X																X
dbl																X	X																X
str																X	X																X
define																																X	
if																																X	
else									X																								
for																																X	
while																																X	
{	X	X	X		X		X	X		X	X	X	X	X																			
,										X	X																						
}	X	X	X	X	X	X	X	X			X	X	X	X		X																	
print																																X	
input																																X	
return																X														X		X	
:																X		X	X	X	X	X	X	X	X	X	X	X	X			X	
;	X	X	X	X	X	X	X	X			X	X	X	X																		X	
=									X				X		X			X	X	X	X									X		X	X
+															X			X	X	X	X	X	X									X	
–															X			X	X	X	X	X	X									X	
*															X			X	X	X	X	X	X									X	
/															X			X	X	X	X	X	X									X	
&&															X			X	X	X	X			X	X	X	X	X	X	X		X	X
‖															X			X	X	X	X			X	X	X	X	X	X	X		X	X
==															X			X	X	X	X	X	X							X		X	X
<															X			X	X	X	X	X	X									X	X
>															X			X	X	X	X	X	X									X	X
!=															X			X	X	X	X	X	X							X		X	X
<=															X			X	X	X	X	X	X									X	X
>=															X			X	X	X	X	X	X									X	X
"																	X					X	X	X			X			X	X		
\n																														X	X		
(	X	X	X																											X		X	X
)									X							X						X	X	X	X	X	X	X	X				X

Code Generators

The following is a list of the code generators, their purpose, and examples of when they are called.

Name	Purpose	Example Code
declareFunction	Indicate that a function is being declared	define (
startParam	Indicate that a parameter is being defined	(int || (dbl || (str || , int || , dbl || , str
endParam	Indicate that a parameter definition is done	int , || dbl , || str , || int) || dbl) || str)
beginFunction	Indicate the starting point of the code inside of a function block	) {
nop	Perform no operation when moving to new line that is unrelated to the current token	{ int || { dbl || { str || int || dbl || || { if || { for || { print || { while || str || if || for || print || } int || } str
beginVarDec	Indicate that a variable is being declared	int = || dbl = || str =
endVarDec	Indicate the end of the variable declaration	= || + || - || * || / || :
beginOoO	Indicate the start of the order of operations	= : || + : || - : || * : || / :
endOoO	Indicate the end of the order of operations	+ : || - : || * : || / :
endLine	Indicate the end of the line of code	anything
endFunction	Indicate the end of a function	} || } }
conditionStart	Indicate the start of a condition	if (|| while (|| for (
conditionEnd	Indicate the end of a condition	) {
loadValue	Load value in variable for use in line of code	{ <operator> || <operator>
addValue	Add a value to another	= + || + + || - + || / + || * +
subtractValue	Subtract a value from another	= - || + - || - - || / - || * -

About the Author

Robert Wood is a 21-year-old Senior undergraduate student at York College of Pennsylvania majoring in Computer Science with a minor in Mathematics. His hobbies consist of programming, playing lacrosse, weightlifting, playing music, and playing video games with his friends.

Originally a Bio-Chem major at a previous school, Robert was talked into switching his major to computer science by his older brother, Austin. Neither of the Wood brothers had a good understanding of what a computer science program consisted of, but Robert decided to take the risk as he has always been fascinated by different technologies. Though he was familiar with using different kinds of software, Robert had never written a line of code prior to switching his major. Luckily, he was able to pick up on the base concepts of programming relatively quickly and he has not looked back since. Robert's main goals for when he graduates college are to become a full-stack software engineer and start a family of his own.

Robert's portfolio website can be found at www.robertwood.dev for more information about himself, projects that he has made/has been a contributor to, and for links to his social media profiles.