GeoCode Language Reference Manual

Contents

1 Introduction

With the advent of 3d machining and the availability of affordable routers, CNC machines have become quite common. The driving force behind a CNC machine is G-Code, an old language that is very cumbersome and verbose. The G-Code language focuses on the mechanics of the machine and this verbosity obscures the meaning of the final program.

GeoCode is a language that focuses on allowing the user to describe the program through the definition of the geometrical entities that should be machined. By allowing the user to focus on higher logic, the program will be less obscured by the details of the machining process. The user will create the geometric entities that compose the target motions and Geocode will produce the output in a G-Code file that can be run on a CNC machine.

2 Lexical Elements

2.1 Identifiers

Identifiers are strings used for naming variables, functions, and objects. All identifiers must start with an upper of lower case letter and can be followed by any upper or lower case character, digit, or underscore. The identifier is case sensitive.

2.2 Keywords

Special identifiers are reserved by the Geocode language. These identifiers are:

```
     bool class double else false for geoclass if include int LCS list return
string text true while
```

2.3 Literals

Literals are notations for constant values of primitive types.

1) Integer Constant: A sequence of digits (0-9). Example: 42
2) Double Constant: A sequence of digits (0-9) that contain the decimal . character. The decimal may be the first character in the sequence as long as digits follow the decimal. Example: 13.2 or .17

3) String Constant: A sequence of zero or more characters, digits, and escape sequences surrounded by quotation marks. Example: “test@”
4) Text Constant: A sequence of upper and lower case letters a-z, A-Z, or the digits 0-9 signified with the starting constant letter t and surrounded by quotation marks. Example: t”SimpleText1”
5) List Constant: A sequences of the same data type surrounded in square brackets. Any of the above 4 data types can be used in a list. Example: [1, 2, 3, 4, 5]

2.4 Line Structure

A Geocode program is divided into a group of lines that are terminated by the Unix form using ASCII LF (linefeed). A statement must be on a single line (cannot cross multiple lines).

2.5 Indentation

Leading tabs at the beginning of a line are used to signify the indentation level. The indentation level for a group of statements must be the same.

2.6 Whitespace

Spaces and tabs (except for leading tabs) are ignored.

2.7 Comments

Comments begin with the character sequence /* and end with character sequence */. Everything in a comment is ignored by the compiler.

3 Data Types

3.1 Primitive Data Types

There are 5 primitive types in the Geocode language: integer, double, bool, string, and text.

3.1.1 Integer Types

Integer values are numbers from -2^{30} to $2^{30}-1$, that is −1073741824 to 1073741823. Uninitialized integers are set to the value 0.

3.1.2 Double Types

Double values are numbers in floating-point representation. The current implementation uses double-precision floating-point numbers conforming to the IEEE 754 standard, with 53 bits of mantissa and an exponent ranging from −1022 to 1023. Uninitialized doubles are set to the value 0.0.

3.1.3 Bool Types

Boolean values are the truth values true or false. Uninitialized bools are set to the value false.

3.1.4 String Types

String values are a sequences of characters. All characters are valid except for the character **"** which is used to signify the termination of the string. Uninitialized strings are set to the value "".

3.1.5 Text Types

Text values are similar to strings, except they are more limited in the character set they can hold. Text values can only contain upper and lower case letters a-z, A-Z, and the digits 0-9. Uninitialized texts are set to the value "".

3.2 Class

A class is a user-defined data type made up of variables of primitive types and other classes. The inclusion of class types are allowed with two exceptions:

1) A class may not include itself (recursion).
2) Two classes may not mutually include each other (mutual recursion).

3.2.1 Defining Classes

A class is defined with the keyword class followed by the name and then a colon. On the next line, a series of variables can be defined. A member variable is declared the same as a regular variable. Following the member variable declaration is the member function declaration. A member function declaration is the same as a regular function declaration. A class must have either at least one member variable or one member function. The class definition is signified by the end of the indentation for the class member declarations.

Here is an example of defining a simple object for a person:

```
class Person:
   string name
   int age
   Print():
      print(name)
```

This defines a class named Person which has two member variables (name and age) and one member function (Print).

3.2.1 Declaring Class Variables

Variables of a class are declared the same as primitive variables. The line must include the name of the class and then the instance name.

```
Person bill, ted
```

That example declares two variables of type Person named bill and ted.

3.2.2 Initializing Class Members

You can initialize the members of a class on the same line as the declaration of the class variable. The initialization must start on the same line as the variable declaration, be surrounded in parentheses, and follow the pattern member name : value. Multiple member initializations must be separated by a comma operator.

```
Person bill(name:”Bill”, age:42)
```

Uninitialized variables will be set to the default variables for the primitive type.

3.2.3 Accessing Class Members

Class members are accessed using the member access operator, which is the period symbol. To access a member, type the name of the class variable followed by the access operator followed by the member name.

```
class Person:
      string name
      Print():
         print(name)

   main():
        Person bill
        bill.name = "Bill"
        bill.Print()
```

3.3 List

A list is a container type that holds a collections of primitive types or classes. All members of a list must be the same data type.

3.3.1 Declaring Lists

List variables are declared similar to primitive variables, except the type that the list contains must be enclosed in square brackets.

```
     list[int] numbers
```

3.3.2 Initializing Arrays

List variables can be initialized by enclosing the values in square brackets. All values in the initialization list must be of the same type as the declared list type.

```
     list[int] numbers = [ 0, 1, 2, 3, 4 ]
```

3.3.3 Accessing Array Elements

Elements of the list can be accessed by zero-based index value. To access an element of the list, surround the index value by square brackets.

```
     numbers[0] = 1
```

Negative integer values index from the back of the list. To access the last element of a non-empty list, use the follow syntax:

```
     print(numbers[-1])
```

3.3.4 List Functions

The list type provides the following functionality:

count(): Return the number of elements in the list.

insert(index, element): Insert element into the list at index value.
pop(): Remove and return the last element on the list.
push(element): Add element to the end of the list.
remove(index): Remove and return the element at index value.
reverse(): Reverse the order of the list.

List functions are called the same as class functions:

```
list[int] numbers = [ 0, 1, 2, 3, 4 ]
print(numbers.count())
```

3.3.5 Concatenating Lists

Lists of the same type can be concatenated with the + operator. The order of newly created list1 + list2 will be list1 followed by list2.

```
list[int] small_numbers = { 0, 1, 2, 3, 4 }
list[int] big_numbers = { 1000, 1001, 1002, 1003, 1004 }
list[int] all_numbers

all_numbers = small_numbers + big_numbers
```

3.4 Point2

The Point2 class represents a 2-dimensional point. The class contains 2 double members: x, y. The data members can be accessed with the class access operator:

```
Point3 start

start.x = 4.2
```

3.5 Point3

The Point3 class represents a 3-dimensional point. The class contains 3 double members: x, y, z. The data members can be accessed with the class access operator:

```
Point3 start

start.z = 4.2
```

3.6 LCS: Local Coordinate System

The variable LCS exists at all times to represent the current coordinate system. The local coordinate system consists of a 3-dimensional origin and axis.

3.6.1 LCS Origin

The LCS origin is a Point3 variable and may be accessed the same as a Point3 class.

```
double current_x

current_x = LCS.origin.x
```

The default value for the LCS origin is 0.0 for all members.

3.6.2 LCS Axis

The LCS axis is an object consisting of 3 Point3 members: u, v, w. The data members can be accessed through the class access operator:

```
double current_w_x

current_w_x = LCS.axis.w.x
```

The default value for the LCS axis is the standard XYZ axes: w=X, y=Y, w=Z.

3.6.3 LCS Functions

The LCS type provides the following functionality:

identity(): Set the LCS to the default value.
rotate(double angle, Point3 orientation): Rotate the coordinate system by angle degrees around the orientation vector.
translate(Point3 shift): Translate the coordinate system by the shift vector.

LCS functions are called the same as class functions:

```
Point3 shift(x: 1.0, y: 1.0, z: 1.0)

LCS.translate(shift)
```

3.6.3 LCS Scope

Modifications made to the local coordinate system are only observable for statements in the same scope.

3.7 Geoclass

The geoclass type is a special class type that must declare and implement a set of predefined functions. The geoclass has an implicit member variable LCS which signifies the class variables coordinate system.

3.7.1 Geoclass LCS

The geoclass LCS is assigned upon creation by using the current scope LCS. This data member is available to the class by using the LCS keyword.

3.7.2 Geoclass Required Functions

The following are the set of functions that all geoclass classes must declare and define:

1) Translate(Point2 shift): Translate the geometry class in the u,v plane.
2) Rotate(double angle): Rotate the geometry around the w vector in the u,v plane.
3) Scale(Point2 scale): Scale the geometry in the u,v plane.
4) Print(): Output the geometry as NC Code.

3.7.3 Geoclass Declaration

A geoclass declaration and definition is identical to a regular class:

4 Expressions and Operators

4.1 Expressions

An expression is a literal, an identifier, a binary operator, an assignment, a function call, of a grouping of expressions surrounded by parentheses. Here are some examples:

```
42
21 * 2
print(42)
```

4.2 Assignment Operators

Assignment operators store values in variables. The assignment operator = stores the value of its right operand in the variable specified by its left operand.

```
String test = “Hello World”
```

4.3 Arithmetic Operators

Addition +, subtraction -, multiplication *, division /, and modular division % are accomplished with the use of the operators identified. Arithmetic operators only operate on variables of the same type.

Arithmetic operators are fully supported for integers:

```
x = 5 + 3
x = 101 % 2
```

All arithmetic operators can be used with double types except for modular division.

```
x = 5.1 * 2.1
```

The string and text type can allow the use of the addition operator.

```
x = “Hello “ + “World”
```

4.4 Comparison Operators

Two variables of the same type can be compared through comparison operators. The comparison operators are equality ==, not equal !=, less than <, less than or equal <=, greater than >, and greater than or equal >=.

All primitive data types can use the equality and not equal operators.

```
    if (“Hello “ != “World”)
```

The rest of the comparison operators are only supported for double and integers.

```
    if (2 < 3)
```

4.5 Type Casts

Type casts are not supported at this time.

4.6 Operator Precedence

The operators are listed in the order of precedence:

Left Associative	multiplication and division	* \
Left Associative	addition and subtraction	+ -
Left Associative	less than and greater than	< <= > >=
Left Associative	equality and not equal	== !=
Right Associative	assignment	=

5 Statements

Statements cause actions and control flow within the program.

5.1 if Statement

The if statement executes a group of code when the predicate is true. The form of the if statement is the keyword if followed by a predicate surrounded by parentheses, ending with a colon. There may be an else associated with the if statement, which will be executed if the predicate is not true.

```
if (predicate):
  predicate-true
else
  predicate-false
```

The if statement and the statement to be executed can be on the same line as long as a single statement is to be executed:

```
x = 5 + 3
if (x == 8): print(“Eight is great!”)
```

If there is more than one statement to execute, then the statements must be entered on separate lines with an indentation level greater than the if statement:

```
x = 5 + 3
if (x == 8):
   x = x + 5
   print(“Thirteen is great!”)
else:
   x = 8
   print(“Now x is great!”)
```

5.2 while Statement

The while statement is similar to the if statement except that the statement loops and continues looping until the predicate is false:

```
int counter = 0

while (counter < 7):
   print("Need more counter”);
   counter = counter + 1
```

5.3 for Statement

The for statement is similar to the while statement in that it will continually loop as long as the predicate is true. The for statement has 3 parameters:

1) Initialization expression: The expression will be executed once before entering the for loop body.
2) Predicate expression: The for loop will continue as long as the predicate is true.
3) Step expression: At the end of each iteration of the loop, this expression will be called.

```
for (initialize; test; step)
  statement
```

An example:

```
int x

for (x = 0; x < 10; x++):
  print(x);
```

5.4 return Statement

The return statement is used inside a function to stop the execution of that function and return the value to the caller.

```
string WhatTheCatSay():
  return "Meow"

main():
  string answer
  answer = WhatTheCatSay()
```

6 Functions

Functions are used to group logical code into a unit that can be called later in the program. A function must be declared and defined before it is used in the program.

Every program requires at least one function, called main. That is where the program's execution begins.

6.1 Function Declarations

A function is declared with an optional return type, a function name, and a parameter list surrounded by parentheses. Here is the general form:

```
optional-return-type function-name (parameter-list):
```

The return type is the data type that is being returned from the function. If the function does not return, then no return type should be specified.

A parameter is a variable type followed by a variable name. A parameter list is a sequence of parameters separated with commas. The parameter list can be empty. Here is an example of a function declaration with two parameters:

```
int add(int x, int y):
```

6.2 Function Definitions

A function must be defined right after the declaration. The function definition is a group of statements following the function declaration that have a greater indentation than the function declaration.

```
int add(int x, int y):
   return x + y
```

6.3 Calling Functions

A function is called by its name along with any parameters that are required. The assignment operator can be used to save data returned from the function.

```
a = add(2, 40);
```

6.4 The main Function

Every program must have one function, called 'main'. This is where the program begins executing.   The main function has no return type and no parameter list.

```
main ():
   print("Hello World!")
```

6.5 print Function

The print function directs the parameters to the final output of the program. The following program would print the text "Hello World" upon completion of the program:

```
main ():
   print("Hello World!")
```

The print function takes only one parameter. If this parameter is a primitive type, then print will output a string representation of this data type. If a class variable is passed to the print function, then the print function will try to call the function Print() on the class. If this function does not exist, this statement will result in an error.

```
include "standard.geo"

main ():
  Circle circle(radius:5)

  print(circle)
```

7 Program Structure and Scope

A program is a file that contains the main function. This program may contain any number of other functions that assist in the program's execution.

7.1 Include

A program may include another file by using the include command. The include command inserts the specified file in the current file as if the file was embedded.

```
include "standard.geo"

main ():
  Circle circle(radius:5)

  print(circle)
```

7.2 Scope

Scope refers to what parts of the program can "see" a declared object. Any line in the program has access to:

1) All variables in the current statement group.
2) All variables declared in the previous lines of the same function that are of a lesser indentation level.
3) All function parameters of the enclosing function.
4) All global variables.

The current statement group is defined as all lines that are of the same indentation level.

The LCS keyword makes extension use of the scope. All modifications to the LCS are only available to the lines in the same scope.

```
main ():
  Circle circle1, circle2
  Point3 up_in_z(0.0, 0.0, 42.0)

      LCS.Translate(up_in_z)
      circle1 = Circle(radius:5)

  circle2 = Circle(radius:5)
```

Since circle1 is in the scope of the LCS translate, circle1 will be created with an LCS that is at 42 in the Z. The variable circle2 is not in the same scope as the LCS transform, therefore, it will be created with the default transform.

8 Standard Library

A standard library standard.geo is included with the compiler. This file defines the basic geometry classes and some utility classes.

8.1 Geometry Classes

The standard library will define the following geometry classes for NC Code output:

1) Line
2) Polyline
3) Rectangle
4) Circle
5) Arc
6) Textbox

8.2 Utility Classes

There is only one utility class defined right now, Math. The following functions will be exposed:

1) Sin
2) Cos
3) Tan