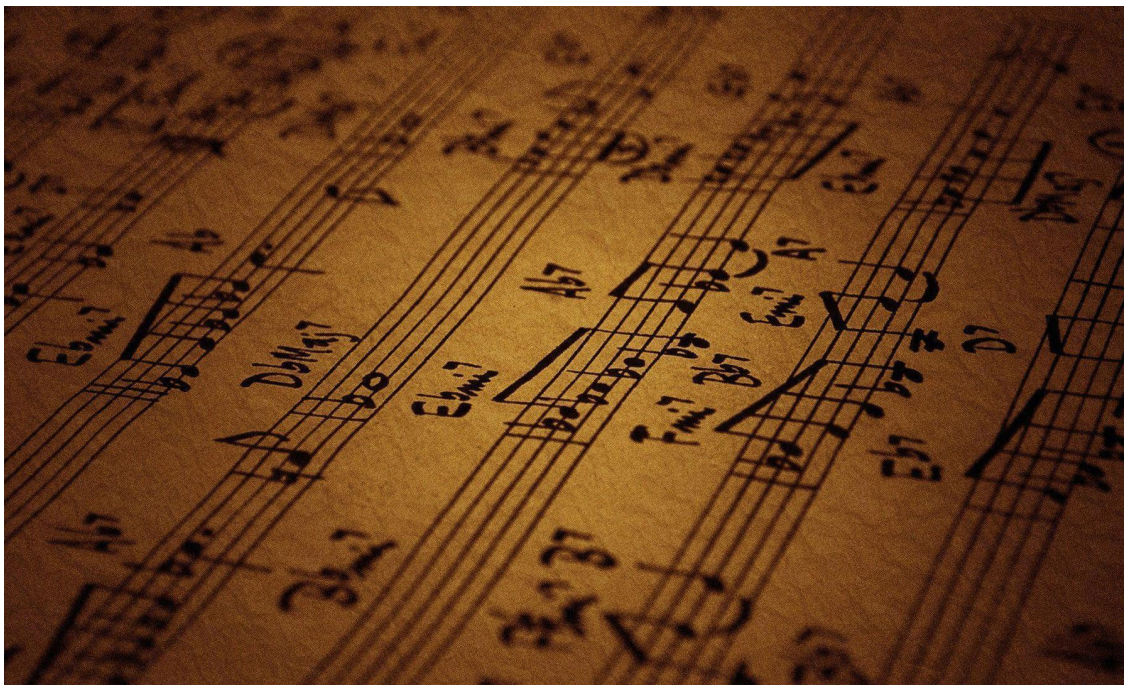
Bazilio Programming Language

ANTLR Programming Masterclass with Python



Lucas Bazilio

Introduction

This document describes the final project of the *ANTLR Programming Masterclass with Python*. Your task is to implement a double interpreter for a music programming language called *Bazilio*. The output of this double interpreter will be a score and some sound files that will reproduce the melody described by the composer.

We call it a double interpreter because it works in both the computer sense (plays a program) and the musical sense (plays a piece of music).

Presentation of the Bazilio language

Bazilio is an algorithmic composition-oriented programming language.
With *Bazilio*, algorithmic compositions are used to generate compositions that give places to scores that can be saved in different digital formats.

Bazilio has many common instructions and has a particular syntax.
This is the *Hello Bazilio*.

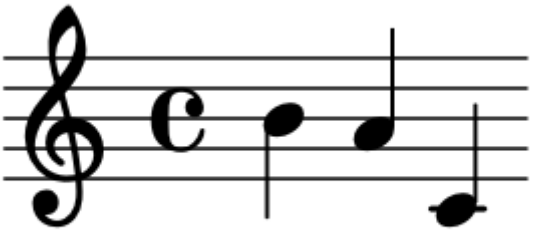
```
### A program in Bazilio ###

Main |:
     <w> “Hello Bazilio”
     (:) {B C A}
:|
```

Thus, we see that the comments are written between triple hashtags (*###*).

Programs are made up of procedures (the order is not relevant), and by default, they start with the *Main* procedure. Each procedure has a name, parameters (there are none in this example), and an associated code block. The blocks are inscribed between the symbols |: and :|.

In *Bazilio*, procedures must start with a capital letter.
On the other hand, variables start with a lowercase letter. The names of musical notes are also capitalized. The first instruction of the program <w> "Hello Bazilio" is a write instruction. The write command is not very useful for composing, but it is useful for debugging because it allows you to write text (in double quotes), integers, and lists. The second statement in the program (:) {B C A} is a play statement. This instruction adds the given note or list of notes to the score. Lists are enclosed in keys with their elements separated by spaces. In this case, the elements are the musical notes B, C, and A. Bazilio uses the English musical notation system. Thus, this program generates the melody Si, La, Do.

The execution of the previous program would produce the output of the Hello Bazilio message on the screen. In addition, it would generate a score.
For instance, an score example of { B A C} is:



along with the following files:

- baz.pdf
- baz.midi
- baz.wav
- baz.mp3

You can find these files in the downloadable resources of this lesson.
As you can see, the output of the interpreter are PDF, MIDI, WAV and MP3 files.

Bazilio allows you to write simple programs using integers in a similar way to regular PLs. For example, the following program shows how to read two numbers and compute their greatest common divisor using Euclid's algorithm with two procedures and input/output:

```
### program that reads two integers and writes their greatest
common divisor ###

Main |:
    <w> "Write two numbers"
    <?> a
    <?> b
    Euclides a b
:|

Euclides a b |:
    while a /= b |:
        if a > b |:
            a <- a - b
        :| else |:
            b <- b - a
        :|
    :|
    <w> "Their GCD is" a
:|
```

Variables are local to each invocation of each procedure, and procedures can communicate through parameters. Procedures list the names of their formal parameters, but do not include their types. Parameters are separated with blanks.

Variables must not be declared, and can be of type integers or lists. Musical notes, it will be seen later, are nothing more than constants for integers.

As seen in the example, the syntax for reading and writing is using <?> and <w> respectively. The comparison operator for equality is = and for difference is /=. The assignment is done with the <- instruction.

In addition, the *Bazilio* programming language features recursion. This program shows how to fix the Tower of Hanoi problem:

```
Main |:
    <?> n
    Hanoi n 1 2 3
:|

Hanoi n ori dst aux |:
    if n > 0 |:
        Hanoi (n - 1) ori aux dst
        <w> ori "->" dst
        Hanoi (n - 1) aux dst ori
    :|
:|
```

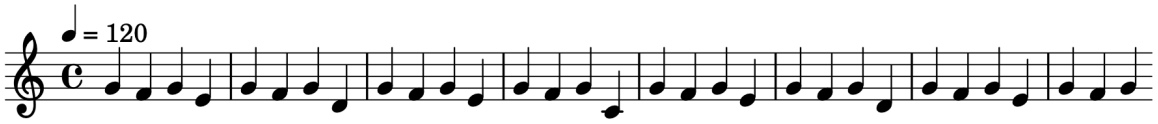
With input **3** the output is:

1 -> 2
1 -> 3
2 -> 3
1 -> 2
3 -> 1
3 -> 2
1 -> 2

Recursion can be exploited in music. The following program composes a nice melody by following the sound of the disks moving, playing the note that corresponds to the disk moving at each step:

```
### Notes of Hanoi ###

Hanoi |:
    src <- {C D E F G}
    dst <- {}
    aux <- {}
    HanoiRec #src src dst aux
:|

HanoiRec n src dst aux |:
    if n > 0 |:
        HanoiRec (n - 1) src aux dst
        note <- src[#src]
        8< src[#src]
        dst << note
        (:) note
        HanoiRec (n - 1) aux dst src
    :|
:|
```

The score generated in this program is:



You can listen to it in the hanoi.mp3 file in the downloadable resources of this lesson.

In the program above you can see more operations for lists:

- l[i] queries the ith element of a list l. Since *Bazili*o is for musicians, the list indexes start at 1.
- l << x adds the element x to the end of the list l.
- #l returns the length or size of the list l.
- 8< l[i] trims (removes) the i-th element of a list l.

Bazilio language specification

The instructions of *Bazilio* are:

- assignment with <-
- input with <?>
- print with <w>
- the reproduction (play) with (:)
- the invocation of procedures
- the conditional with if and maybe else
- iteration with while
- adding to lists with <<
- the cut of lists with 8<

Instructions written one after another are executed sequentially.

Assignment

The assignment must first evaluate the expression on the right side of the <- and then store the result in the local variable on the left side. Example: a <- a - b. In the case of assigning lists, copy the values (without doing aliasing). For simplicity, only a variable can appear on the left side of an assignment (for example, l[2] <- 5 cannot be done).

Reading

The input reading statement must read an integer value from the standard input channel and store it in the variable to the right of the <?>. Example: <?> x.

Writing

The write statement must evaluate the <w> expression and write it, on one line, to the standard output pipe. Example: <w> x + y. In the case of writing a list, it is necessary to write all its values between square brackets and separated by blanks. <w> can contain several parameters, it is necessary to write each of them on the same line, separated by spaces. Parameters can contain text (closed in double quotes). The texts do not appear anywhere else.

Reproduction

The reproduction (play) statement must evaluate the expression of the (:). If it is a note, you must add it to the score, with the value of a quarter note. If it is a list, you must add each of your notes (from left to right).

Conditional

The conditional statement has its usual semantics. The block of *else* is optional.
Examples: `if x = y |: z <- 1 :| else |: z <- 2 :|`. Note that block limiters are always required (both in conditions and in procedures and whiles).

Iteration with while

The iterative statement with while has its usual semantics.
Example: `while a > 0 |: a <- a / 2 :|.`

Procedure invocation

A procedure call has its usual semantics.
If the number of parameters passed does not correspond to those declared, an error occurs. Procedures are not functions and cannot return results. But procedures can be called recursively. The syntax is as follows: no parentheses or commas. Example:
`Writing x + y 2.`

Expressions

If a variable has not yet received a value, its value is zero. Arithmetic operators are the usual ones (+, -, *, /, %) and with the same priority as in C. Parentheses can of course be used. Relational operators (=, /=, <, >, <=, >=) return zero for false and one for true.

Lists

Lists are enclosed in keys, with their elements separated by blanks. List elements can only be integers (or notes, which are just integer constants). Lists are passed by reference. #l returns the length of the list l. l[i] queries the ith element of a list l; y must be between 1 and #l. l << x adds the element x to the end of the list l. 8< l[i] removes the i-th element from a list l; it is also required to be between 1 and #l.

Scope of visibility

The order in which the procedures are declared does not matter. Variables are local to each invocation of each procedure. There are no global variables and no way to access variables from other procedures (only through parameters).

Notes

Bazilio provides some names that represent the white notes on a piano. The first three notes are A0 (La0), B0 (Si0), C1 (Do1). The last three are A7 (La7), B7 (Si7), C8 (Do8). In https://es.wikipedia.org/wiki/Frecuencias_de_afinaci%C3%B3n_del_piano you have an explanation of this nomenclature. Furthermore, the notes C, D, E, F, G, A, B (without number) are synonyms of C4 (central Do), D4, E4, F4, G4, A4, B4. The notes of *Bazilio* are just constants, so A0 is equal to 0, B0 equals to 1, ... and C8 equals ???. Thus, a note can be transposed one octave higher or lower by adding or subtracting 7 units.
(yes, musicians are peculiar people and 7 notes they call an octave).

The following procedure would play all the white piano keys from lowest to highest (from left to right):

```
All_Keys |:
    note <- A0
    while note <= C8 |:
        <:> note
        note <- note + 1
    :|
:|
```

Errors

Although *Bazilio* is quite simple, the programs can cause many errors at runtime. For this practice, we only ask you to catch the most likely errors (division by zero, call to undefined procedure, repetition of already defined procedure, number of incorrect parameters, repeated formal parameter names, access to a non-existent index of a list ...) and have the program throw an exception when they occur. You should not perform semantic analysis for type errors.

Invocation of the interpreter

Your interpreter must be invoked with the command python3 bazilio.py, passing as a parameter the name of the file that contains the source code (the extension of the files by programs in *Bazilio* is .bzl). For example:
python3 bazilio.py music.bzl

If you want to start from a procedure other than Main, you can give its name as a parameter. For example:
python3 bazilio.py music.bzl Hanoi

If the program runs correctly, the files musica.pdf with the score in PDF format, music.midi with music in MIDI format, musica.wav with music in WAV format, and music.mp3 with music in format MP3 will be generated.

If you wish, you can also have the music play at the end of the program.

Libraries

Use ANTLR to write the grammar and the interpreter. You can freely use any standard Python library.

External programs

To generate the scores, you must use the LilyPond program. Lilypond already generates MIDI and PDF. To generate WAV from MIDI you must use timidity++. To generate MP3 from WAV you must use ffmpeg. All of these programs can be easily installed on Mac, Linux and (I assume) Windows. Your program will be corrected in an environment where the lilypond, timidity and ffmpeg binaries are in the path.

Brief demonstration of external programs

LilyPond is a music typography program. It is free software and is part of the GNU Project. The scores are described from a fairly complex language, but when you only need to know a few rudiments to do this project.
For example, this example.lily program in LilyPond describes a stair with empty blacks (Do, Re, Mi, Fa, Sol, La, Si, Do) at a tempo of 120:

```
\version "2.20.0"
\score {
    \absolute {
        \tempo 4 = 120
        c'4 d'4 e'4 f'4 g'4 a'4 b'4
    }
    \layout { }
    \midi { }
}
```

When you run

```
lilypond example.lily
```

Lilypond generates example.pdf and example.midi (because the \layout and \midi requests have been given respectively).

TiMidity++ is another open source program that transforms MIDI files into sound files, such as WAV files. Can be run like this

```
timidity -Ow -o example.wav example.midi
```

in order to get the example.wav file.

The free program ffmpeg allows you to convert between different video and audio formats. Can be run like this

```
ffmpeg -i example.wav -codec:a libmp3lame -qscale:a 2 example.mp3
```

to get the example.mp3 file.

Lastly, MP3 files can be played with many audio players. On a Mac, it can be done like this from the command line:

```
afplay exemple.mp3
```

Your job

Your job is to implement a (double) interpreter for *Bazilio*.

To carry out his work, he must use Python3 and ANTLR4, as explained in the course classes. To generate the scores, you must use the Lilipond program. To generate WAV and MP3, the programs Timidity++ and ffmpeg.