January 2025: CSE310 – Compiler Sessional
Getting Started with ANTLR4

Prepared by – Nafis Tahmid

May 2025

1 Introduction

ANTLR (ANother Tool for Language Recognition) is a robust parser generator designed for reading, processing, executing, or translating structured text or binary data. It is extensively used in the development of programming languages, tools, and frameworks. Given a grammar, ANTLR produces a parser capable of constructing and traversing parse trees.

2 Setting things up

This section outlines the installation process for Linux keeping details limited to much basic as possible. ANTLR can be installed using two different methods. The first method is simpler and suitable for initial experimentation or quick testing. The second method is more appropriate for regular or long-term use. It is recommended to try the first method initially to become familiar with the setup, and then proceed with the second method for standard usage.

2.1 Just getting started with antlr4-tools

To play around with ANTLR without having to worry about installing it and the Java needed to execute it, use antlr4-tools. The only requirement is `Python3`. But at first, we may require creating a virtual environment for installation using `pip`. The command to create and activate the environment

```
$ python3 -m venv <environment_name>  # this creates virtual environment
$ source <environment_path>/bin/activate # for environment activation
```

Suppose you want to create an environment named `antrl4_venv` in `~/3-1/CSE310` directory. So, navigate to the directory and then execute the following commands.

```
$ python3 -m venv antlr4_venv  # this creates virtual environment
$ source ./antlr4_venv/bin/activate # for environment activation
```

Now you should see the shell like this

```
(antlr4_venv) $
```

Now as you have the environment activated, use the following command which creates `antlr4` and `antlr4-parse` executables. (You can deactivate an environment by simply writing `deactivate` and then pressing enter)

```
$ pip install antlr4-tools
```

2.1.1 First run will install Java and ANTLR

If needed, antlr4 will download and install Java 11 and the latest ANTLR jar:

```
$ antlr4
Downloading antlr4-4.13.2-complete.jar
ANTLR tool needs Java to run; install Java JRE 11 yes/no (default yes)? y
Installed Java in /Users/parrt/.jre/jdk-11.0.15+10-jre; remove that dir to uninstall
ANTLR Parser Generator  Version 4.13.2
 -o ___              specify output directory where all output is generated
 -lib ___            specify location of grammars, tokens files
...
```

Now let's play with simple grammar. Save the following text into a file named `Expr.g4`. Note that the grammar name and file name must be exactly same, otherwise it will generate error.

```
grammar Expr;
prog:  expr EOF ;
expr:  expr ('*'|'/') expr
    |  expr ('+'|'-') expr
    |  INT
    |  '(' expr ')'
    ;
NEWLINE : [\r\n]+ -> skip;
INT     : [0-9]+ ;
```

2.1.2 Try parsing with a sample grammar

To parse and get the parse tree in text form, use:

```
$ antlr4-parse Expr.g4 prog -tree
10+20*30
^D
(prog:1 (expr:2 (expr:3 10) + (expr:1 (expr:3 20) * (expr:3 30))) <EOF>)
```

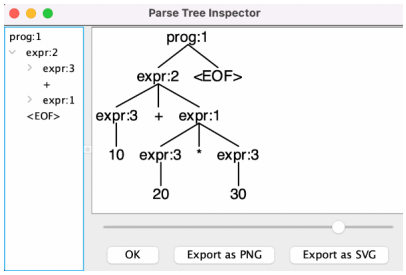
(Note: `^D` means control-D and indicates "end of input".)
Here's how to get the tokens and trace through the parse:

```
$ antlr4-parse Expr.g4 prog -tokens -trace
10+20*30
^D
[@0,0:1='10',<INT>,1:0]
[@1,2:2='+',<'+'>,1:2]
[@2,3:4='20',<INT>,1:3]
[@3,5:5='*',<'*'>,1:5]
[@4,6:7='30',<INT>,1:6]
[@5,9:8='<EOF>',<EOF>,2:0]
enter   prog, LT(1)=10
enter   expr, LT(1)=10
consume [@0,0:1='10',<8>,1:0] rule expr
enter   expr, LT(1)=+
consume [@1,2:2='+',<3>,1:2] rule expr
enter   expr, LT(1)=20
consume [@2,3:4='20',<8>,1:3] rule expr
enter   expr, LT(1)=*
consume [@3,5:5='*',<1>,1:5] rule expr
enter   expr, LT(1)=30
consume [@4,6:7='30',<8>,1:6] rule expr
exit    expr, LT(1)=<EOF>
exit    expr, LT(1)=<EOF>
exit    expr, LT(1)=<EOF>
consume [@5,9:8='<EOF>',<-1>,2:0] rule prog
exit    prog, LT(1)=<EOF>
```

Here's how to get a visual tree view:

```
$ antlr4-parse Expr.g4 prog -gui
10+20*30
^D
```

The following will pop up in a Java-based GUI window:



2.1.3 Generating parser code

The previous section used a built-in ANTLR interpreter but typically we will ask ANTLR to generate code in the language used by our project (there are about 10 languages to choose from as of 4.11).

To generate Java code from a grammar:

```
$ antlr4 Expr.g4
$ ls Expr*.java
    ExprBaseListener.java  ExprLexer.java         ExprListener.java      ExprParser.java
```

And, to generate C++ code from the same grammar:

```
$ antlr4 -Dlanguage=Cpp Expr.g4
$ ls Expr*.cpp Expr*.h
    ExprBaseListener.cpp  ExprLexer.cpp         ExprListener.cpp      ExprParser.cpp
    ExprBaseListener.h    ExprLexer.h           ExprListener.h        ExprParser.h
```

2.2 Installation with libraries and runtimes

ANTLR is really two things: a tool written in Java that translates grammar to a parser/lexer in Java (or other target language) and the runtime library needed by the generated parsers/lexers.

Installing Java

First, we need to have Java (version 11 or higher) installed in our system. We can check whether we have Java already installed in our system by using

```
$ java --version
```

If we do not have it, we can install version 17 (LTS) using the following commands.

```
$ sudo apt update
$ sudo apt install openjdk-17-jdk
```

Download

Now let's run the following commands

```
$ cd /usr/local/lib
$ sudo curl -O https://www.antlr.org/download/antlr-4.13.2-complete.jar
```

If you do not have curl installed, run `sudo apt install curl` command to install curl.
However, we can just download the `jar` using browser from website https://www.antlr.org/download.html and put it somewhere rational like `/usr/local/lib`.

Configuring `CLASSPATH` and Using Aliases

The `CLASSPATH` environment variable informs Java where to locate user-defined classes and packages, such as `antlr-4.13.2-complete.jar`. If this variable is not set correctly, Java tools may produce a "class not found" error. If we want to **avoid manually configuring aliases and the `CLASSPATH`** we can **simply activate the virtual environment** we created above and use `anltr4 Expr.g4` command. However, in certain systems, specifying the version explicitly—for example, using `antlr4 -v 4.13.2 Expr.g4` instead of `antlr4 Expr.g4`—can help prevent

errors. On some systems, omitting the version may lead to errors, while on others, including the version might also cause issues. Users are advised to use the form that is compatible with their specific system configuration.
However, let's add `antlr-4.13.2-complete.jar` to our CLASSPATH using -

```
$ export CLASSPATH=".:/usr/local/lib/antlr-4.13.2-complete.jar:$CLASSPATH"
```

It's also a good idea to put this in our `.bash_profile` because when we open a new terminal, it doesn't remember environment variables like `CLASSPATH` unless we explicitly tell it to. By putting it in our startup script, it gets automatically set every time we start a new shell session.
Here is how we can do it using the following commands

1. In your home directory, open `.bashrc` in a text editor using `nano ~/.bashrc`
2. Add - `export CLASSPATH=".:/usr/local/lib/antlr-4.13.2-complete.jar:$CLASSPATH"` at the end of your `.bashrc`.
3. Save and exit (`Ctrl+X`, then `Y`, then `Enter`)
4. Apply the changes - `source ~/.bashrc`
5. Test it using `echo $CLASSPATH` and we should see our path including the ANTLR jar.

Typing long java commands to run ANTLR all the time would be painful, so it's best to make an alias.

1. In your home directory, open `.bashrc` in a text editor using `nano ~/.bashrc`
2. Add at the end of your `.bashrc` -

```
alias antlr4='java -Xmx500M -cp "/usr/local/lib/antlr-4.13.2-complete.jar:$CLASSPATH"
↪  org.antlr.v4.Tool'
```

3. Save and exit (`Ctrl+X`, then `Y`, then `Enter`)
4. Apply the changes - `source ~/.bashrc`
5. Test it using `anltr4 Expr.g4`.

2.3 Integration of the C++ runtime and execution of compiled code

If you are already decided about using Java, you can skip this section.
When setting up ANTLR for C++ development, it's essential to understand the role of the ANTLR runtime. The runtime is a library that provides the necessary support for the parser and lexer code generated by ANTLR to function correctly within our C++ application.
Including the runtime involves integrating the ANTLR C++ runtime library into our project. This library contains implementations of various components such as token streams, parsers, lexers, and tree walkers, which are crucial for the operation of the generated code. By incorporating the runtime, we ensure that our application has access to these components, enabling it to parse and process input as defined by our grammar.

2.3.1 Steps of including the ANTLR C++ runtime

The minimum C++ version to compile the ANTLR C++ runtime with is C++17 and the minimum CMake version required is CMake 2.8. While earlier versions of CMake (e.g., 2.8) might suffice, it is advisable to use a more recent version to ensure compatibility and access to the latest features.
The default CMake version available through apt on Ubuntu may be outdated, potentially leading to compatibility issues. To install the latest version of CMake, follow these steps:

Remove existing installation
This step ensures that any existing, potentially outdated, versions of CMake are removed from your system. First give

```
$ which cmake
```

- **If installed from source**
  If this returns a path like `/usr/local/bin/cmake`, it indicates that CMake was installed manually and not managed by APT. Navigate to the directory where you built CMake and run:

  ```
  $ sudo make uninstall
  ```

  This will remove the installed files. If you no longer have the build directory, you can recreate it by downloading the same version of the source code used for installation, running `./bootstrap` or `./configure`, `make`, and then `sudo make uninstall`. [Follow the documentation properly and give `uninstall instead of install`]

- **If installed via `.sh` installer:**
  If you used a script like `cmake-3.31.7-linux-x86_64.sh` to install CMake, it likely installed to `/usr/local`. To remove it. **BE AWARE ABOUT THE PATHS AND USING `rm -rf`**

  ```
  $ sudo rm -rf /usr/local/bin/cmake
  $ sudo rm -rf /usr/local/share/cmake*
  $ sudo rm -rf /usr/local/doc/cmake*
  $ sudo rm -rf /usr/local/man/man1/cmake*
  ```

- **If installed via Snap**
  If you see path like `/snap/bin/cmake`, then it indicates that CMake was installed via Snap. To remove this Snap-installed version of CMake, run the following command:

  ```
  $ sudo snap remove --purge cmake
  ```

  Then verify removal with `which cmake`

- **If installed via apt**
  If you see a path like `/usr/bin/cmake`, then it was probably installed using apt. To remove it –

```
$ sudo apt purge --auto-remove cmake
$ hash -r
```

Installation of CMake

You can follow any of the following two for installing CMake 3.* (newer version (3.5 or higher) may have backward compatibility issues).

- **Add Kitware APT Repository**
  Kitware provides up-to-date CMake packages. Adding their repository allows you to install the latest version directly. (Be aware about the line-break)

  However, executing the attached `cmake-installation-script.sh` will also do. A good resource about installing the latest CMake – Ask Ubuntu

```
$ wget -O - https://apt.kitware.com/keys/kitware-archive-latest.asc 2>/dev/null | gpg
↪  --dearmor - | sudo tee /etc/apt/trusted.gpg.d/kitware.gpg >/dev/null
$ sudo apt-add-repository "deb https://apt.kitware.com/ubuntu/ $(lsb_release -cs) main"
$ sudo apt update
$ sudo apt install cmake=3.25.*
$ hash -r
```

  You can simply give `sudo apt install cmake` without specifying any version.

- **Installing a specific version of CMake by compiling source**
  Compiling CMake from source is an excellent option. This method involves downloading, building, and installing the source code manually. LinuxCapable.com provides detailed step. You may follow this article.

2.3.2 Installing the ANTLR4 C++ Runtime

The file `antlr4-runtime.h` is part of the ANTLR4 C++ runtime library. To obtain it, let's follow execute the commands one by one: (executing the attached `antlr-runtime-installation-script.sh` will also do)

```
$ git clone https://github.com/antlr/antlr4.git
$ cd antlr4
$ cd runtime/Cpp
$ mkdir build && cd build
$ cmake ..
$ make
$ sudo make install
```

This will compile the runtime and install the headers and libraries on the system so that compiler can find `antlr4-runtime.h` when included.
Many systems place headers in `/usr/local/include` or `/usr/include`. If the runtime is installed there, the compiler may find them by default. Running `find /usr/local/include -name "antlr4-runtime.h"` , the installation can be verified.

2.3.3 Using the `-I` option in compilation command

When compiling project (for example, using g++), we need to use the -I option to add the path to the directory that contains antlr4-runtime.
Suppose we have a source file named `main.cpp`, which is designed to parse an input file specified through command-line arguments. To compile this file and produce an executable named `main.out`, one can use the following commands. These commands are provided for illustrative purposes only. Please refer to the next section for the actual steps for generating and running lexer and parser.

```
$ g++ -std=c++17 -I/path/to/antlr4-runtime/include -c ExprLexer.cpp ExprParser.cpp main.cpp
$ g++ -std=c++17  ExprLexer.o ExprParser.o main.o -L/path/to/antlr4-runtime/lib
↪  -lantlr4-runtime -o main.out -pthread
```

3 Parsing C Code Using ANTLR

This section outlines the procedure for parsing C code using ANTLR, with support for both C++ and Java implementations. The directory structure of the provided files is illustrated in Figure 1.

3.1 C++ Implementation

To generate the lexer and parser in C++ and test the setup, navigate to the `cpp` directory and execute the following commands.
To parse a syntactically correct C source file:

```
$ bash run-script.sh input/test.c
```

To parse a C source file that contains a syntax error (to verify error detection capabilities):

```
$ bash run-script.sh input/test_syntax_error.c
```

`./cpp/run-script.sh` file:

```
$ antlr4 -v 4.13.2 -Dlanguage=Cpp C8086Lexer.g4
$ antlr4 -v 4.13.2 -Dlanguage=Cpp C8086Parser.g4
$ g++ -std=c++17 -w -I/usr/local/include/antlr4-runtime -c C8086Lexer.cpp C8086Parser.cpp
↪  Ctester.cpp
$ g++ -std=c++17 -w C8086Lexer.o C8086Parser.o Ctester.o -L/usr/local/lib/ -lantlr4-runtime
↪  -o Ctester.out -pthread
$ LD_LIBRARY_PATH=/usr/local/lib ./Ctester.out $1
```

3.2 Java Implementation

For the Java implementation, follow the same procedure as above. Navigate to the `java` directory and run the corresponding script files using the same commands. This ensures consistent behavior between the C++ and Java-based parsers.
`./java/run-script.sh` file:

```
$ antlr4 -v 4.13.2 C8086Lexer.g4 C8086Parser.g4
$ javac -cp .:/usr/local/lib/antlr-4.13.2-complete.jar  C8086*.java Main.java
$ java -cp .:/usr/local/lib/antlr-4.13.2-complete.jar Main $1
```

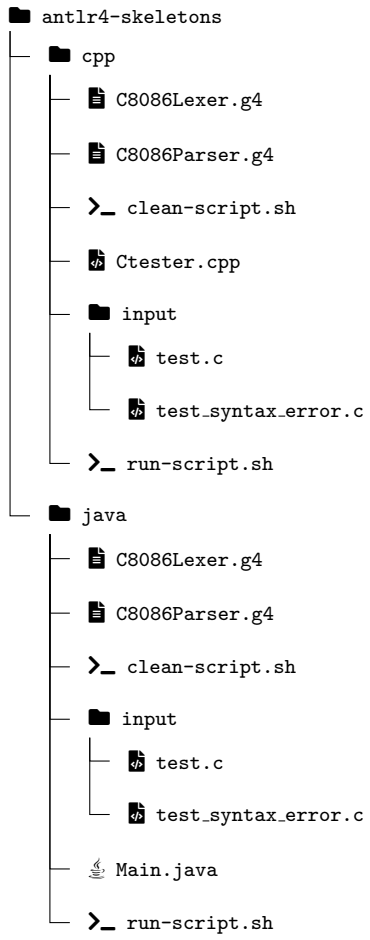


Figure 1: Directory structure of the provided files. The script `run-script.sh` is used to generate the necessary lexer and parser files for parsing. To clean up the generated files, use `clean-script.sh`. Note that this script will remove all "newly generated" files; use it with caution.