

Install and Configure ANTLR 4 for C++

Ronald Mak

Department of Computer Engineering

Department of Computer Science

Department of Data Science

January 20, 2019

Revised: January 7, 2020

Introduction

This tutorial describes the extra steps required to enable ANTLR 4 to generate compiler components written in C++ rather than its native Java, and it also describes how to make ANTLR's Eclipse plugin ANTLR work with C++.

If you haven't already installed and configured ANTLR 4, read the tutorial "Install and Configure ANTLR 4 for Ubuntu and MacOS X" (<http://www.cs.sjsu.edu/~mak/tutorials/InstallANTLR4.pdf>).

A simple expression grammar

In your home directory, **create a directory named `example` and `cd` to it**. Create a grammar file **`Expr.g4`** which is a text file containing:

```
grammar Expr;

prog: (expr NEWLINE)* ;
expr: expr ('*' | '/') expr
     | expr ('+' | '-') expr
     | INT
     | '(' expr ')'
     ;

NEWLINE : [\r\n]+ ;
INT     : [0-9]+ ;
```

This is a grammar for simple arithmetic expressions. Use your `antlr4` alias to invoke ANTLR 4 to process the grammar file, but add the option `-Dlanguage=C++` to generate C++ code instead of Java:

```
antlr4 -Dlanguage=C++ Expr.g4
```

You should now see that ANTLR generated `.h`, `.cpp`, and other files. Unfortunately, there is no C++ equivalent of the Java test rig `grrun`, so later, we will need to write a main program that invokes the generated lexer and parser.

Download and install the C++ runtime for ANTLR

Go to <https://www.antlr.org/download.html> and scroll down to the section “C++ Target”.

Download `antlr4-cpp-runtime-4.7.2-macos.zip` if you are on MacOS X, or `antlr4-cpp-runtime-4.7.2-source.zip` if you are on Ubuntu. (Download and install the version for Windows users at your own risk.)

MacOS X installation

Unzip `antlr4-cpp-runtime-4.7.2-macos.zip` and move the two subdirectories `antlr4-runtime` and `lib` to the `ANTLR-4.7.2` directory that you had earlier created in your home directory and which contains `antlr-4.7.2-complete.jar`.

Ubuntu installation

Unzip `antlr4-cpp-runtime-4.7.2-source.zip` and `cd` to the resulting directory `antlr4-cpp-runtime-4.7.2-source`.

First, make sure you have all the required tools. Type the following commands in the Ubuntu terminal window:

```
sudo apt install cmake
sudo apt install uuid-dev
sudo apt install pkg-config
```

Then type the following commands:

```
mkdir build && mkdir run && cd build
cmake ..
DESTDIR=../run make install
```

The last command above will take a while to approach 100% completion.

Now copy the ANTLR 4 include files to `/usr/local/include` and the ANTLR 4 libraries to `/usr/local/lib` by entering the following commands (which assume that you’re still in the `build` directory):

```
cd ../run/usr/local/include
sudo cp -r antlr4-runtime /usr/local/include
cd ../lib
sudo cp * /usr/local/lib
sudo ldconfig
```

Now you can delete `antlr4-cpp-runtime-4.7.2-source.zip` and the directory `antlr4-cpp-runtime-4.7.2-source`.

Create a C++-based ANTLR project

Create a C++ project in Eclipse called `ExprCpp`. Copy the `Expr.g4` grammar file into the project and open the file in the editor window. By default, the ANTLR plugin will generate Java files in `target/generated-sources/antlr4` (Figure 2).

R. Mak, Install and Configure ANTLR 4 for C++

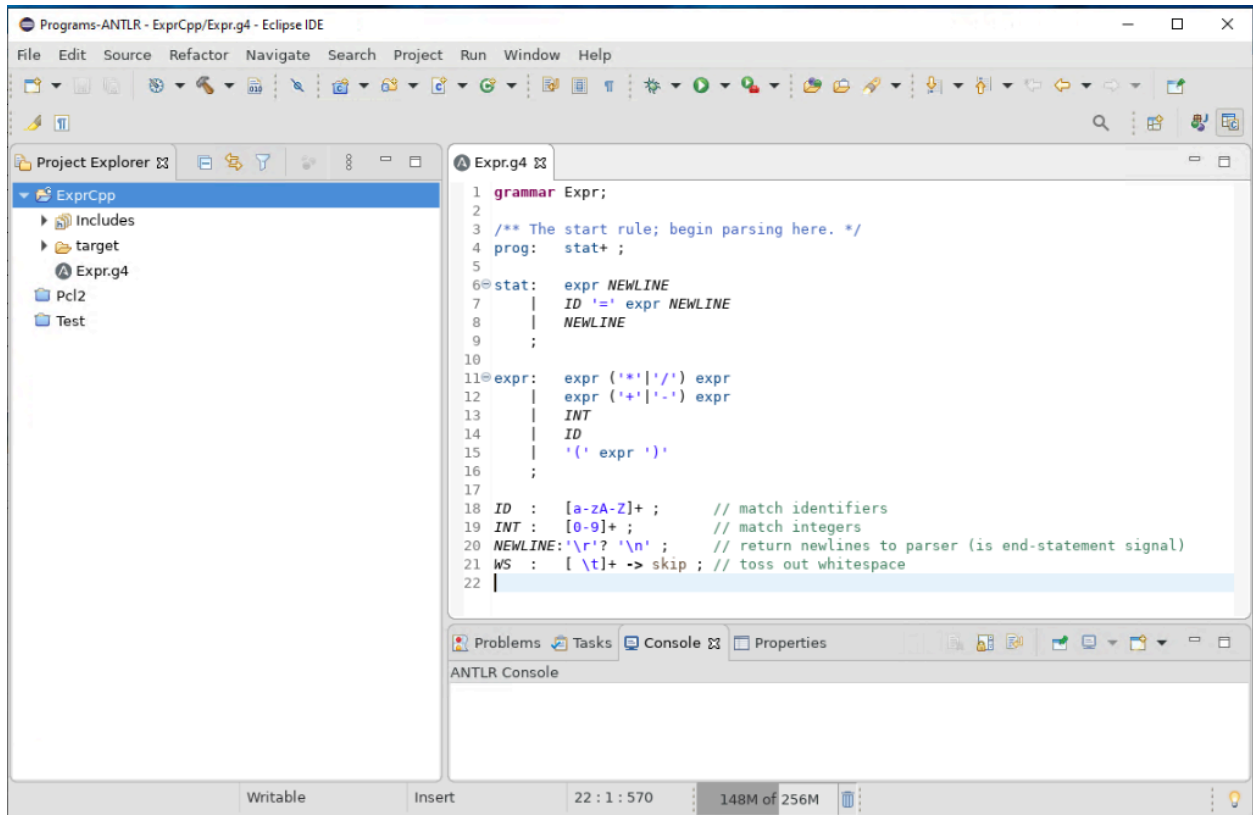


Figure 2. Grammar file **Expr.g4** in a C++ project.

Right-click the project name and select *Properties* from context menu. In the **Properties for ExprCpp** dialog box, select *ANTLR 4 | Tool* in the left panel. (You will not see *ANTLR 4* unless you've opened the grammar file in the editor window.)

Check *Enable project specific settings* and add and select your copy of **antlr-4.7.2-complete.jar**. Uncheck *Enable project specific settings*, click the *Configure Workspace Settings* link, and do the same (Figure 3).

Make sure in *Distributions* that the **antlr-4.7.2-complete.jar** file is present and selected. If not, see "Create a Java-based ANTLR project" in "Install and Configure ANTLR 4 on Eclipse and Ubuntu" (<http://www.cs.sjsu.edu/~mak/tutorials/InstallANTLR4.pdf>). In the *Options* section of both dialog boxes, uncheck *Generate a parse tree listener* and check *Generate parse tree visitors*. Check *Tool is activated*.

R. Mak, Install and Configure ANTLR 4 for C++

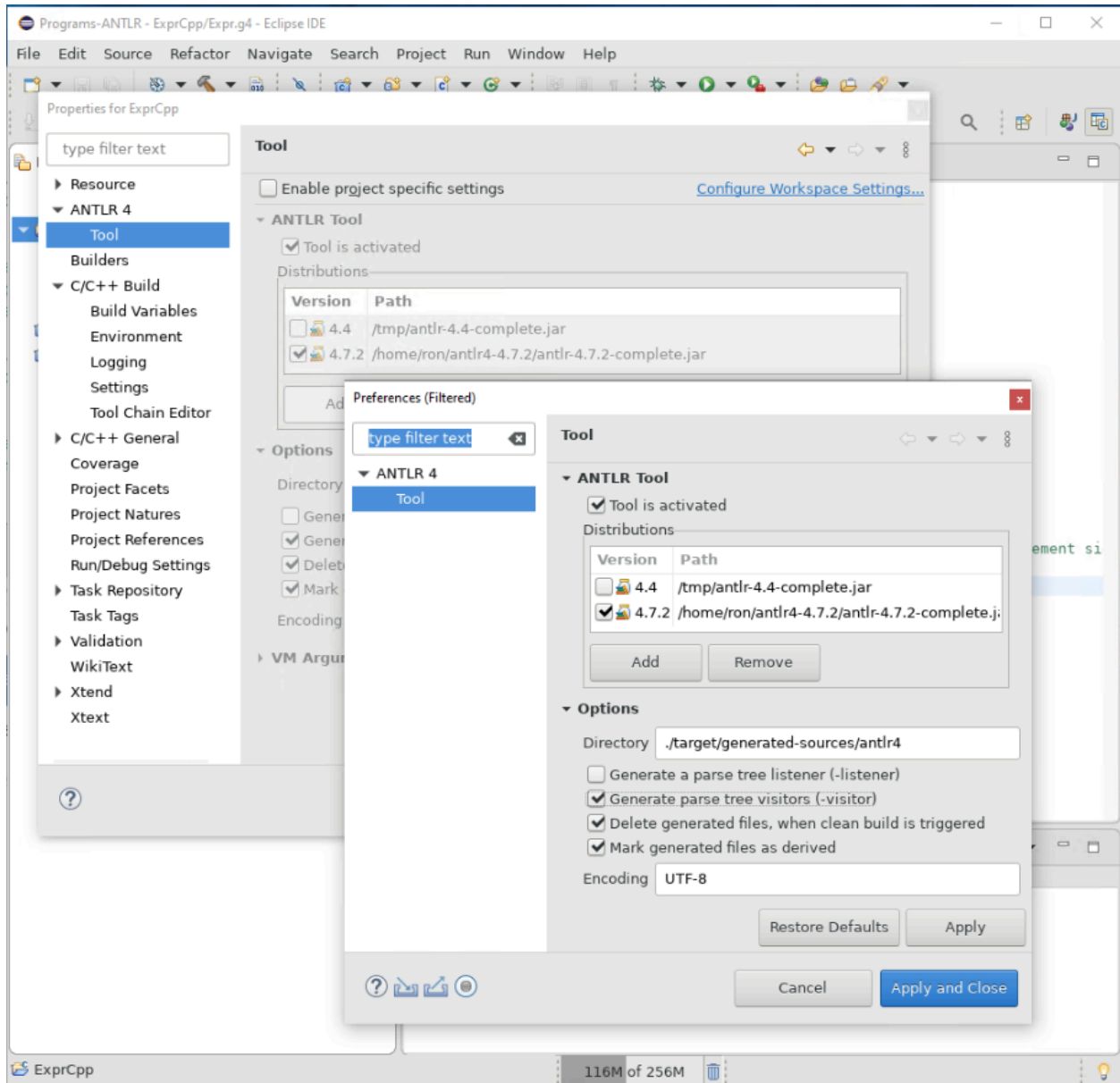


Figure 3. Select the `antlr-4.7.2-complete.jar` file and set the correct options.

To change the generated code to C++, we must create an external tool run configuration. Right-click the grammar file `Expr.g4` in the left panel and select `Run As | External Tools Configurations` in the context menu. Create a configuration named "ExprCpp" for the grammar and enter the following for `Arguments` (Figure 4):

```
-no-listener -visitor -encoding UTF-8 -Dlanguage=C++
```

Click the `Apply` button.

R. Mak, Install and Configure ANTLR 4 for C++

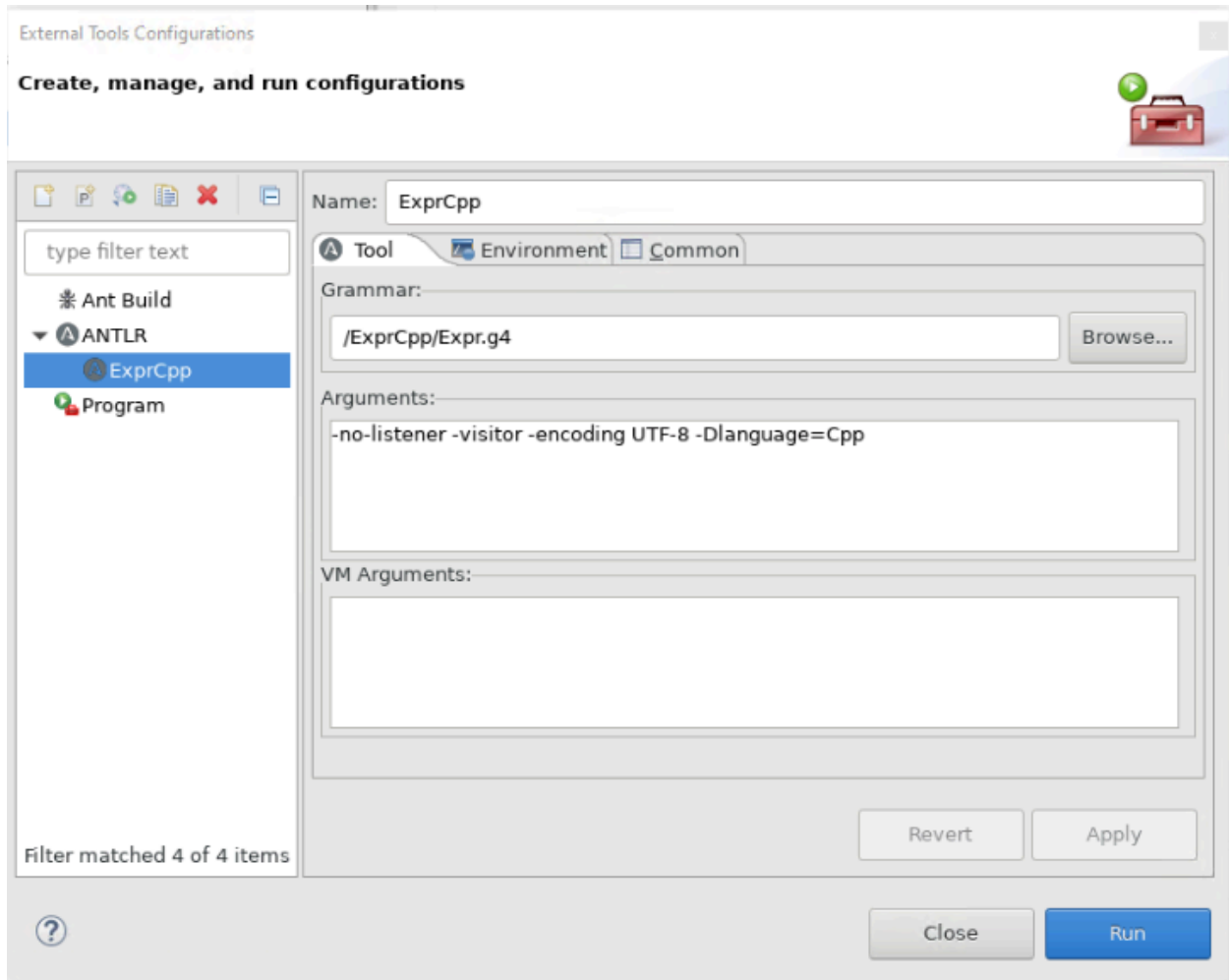


Figure 4. External tools configuration for the `Expr.g4` grammar file.

R. Mak, Install and Configure ANTLR 4 for C++

Click the *Run* button to generate C++ files (Figure 5). These files will regenerate automatically after every modification of the grammar file. You can manually cause a regeneration by right-clicking the name of the grammar file and selecting *Run As | Generate ANTLR Recognizer* from the context menu.

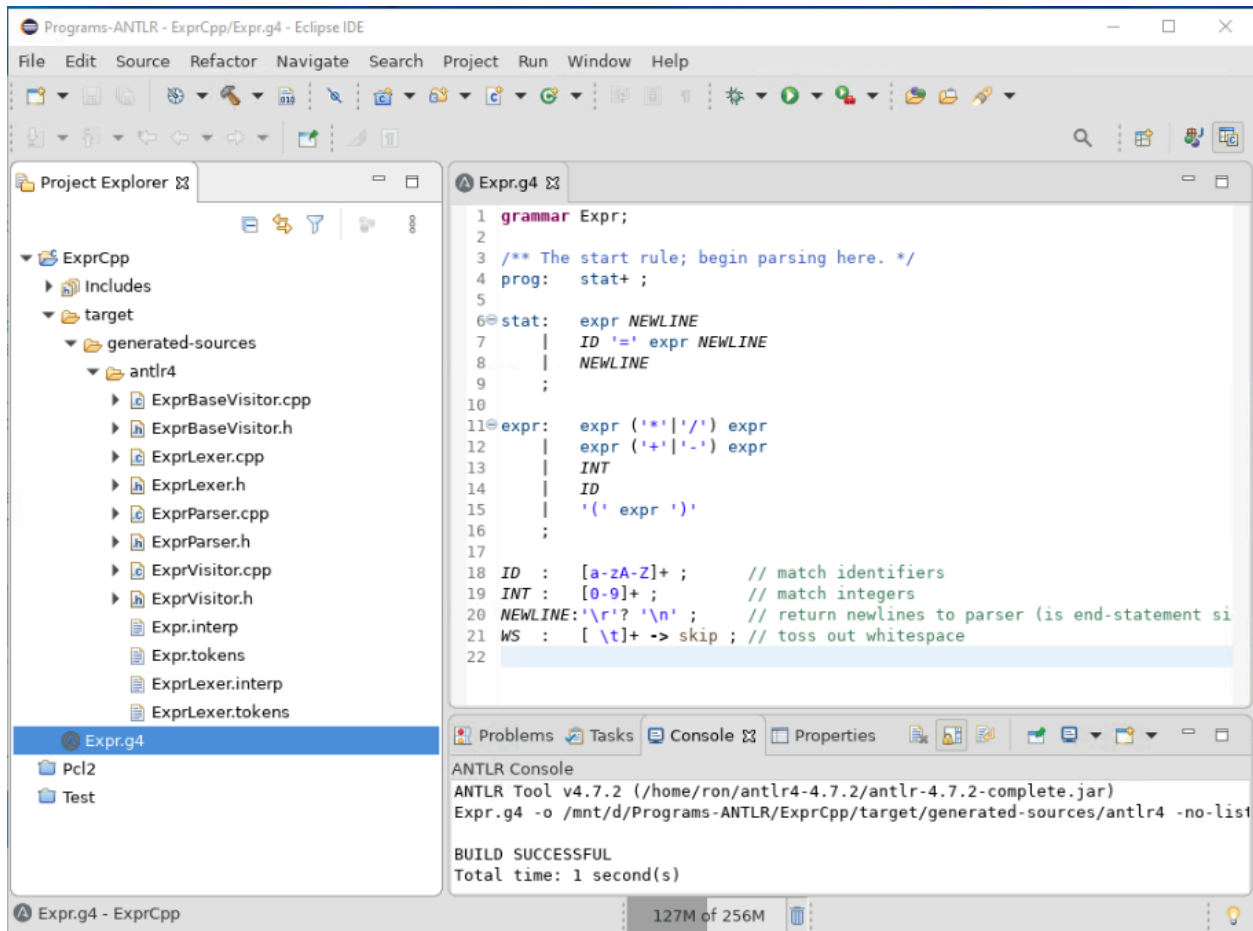


Figure 5. Generated C++ files.

R. Mak, Install and Configure ANTLR 4 for C++

From the *Window* dropdown menu at the top, select *Show View | Other*. Select *Syntax Diagram* and *Parse Tree* and click the *Open* button. Open the **Expr.g4** grammar file in the editor window. Select the *Syntax Diagram* tab. You should see a syntax diagram that ANTLR generated from the grammar file (Figure 6).

The screenshot displays the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The Project Explorer on the left shows a project named 'ExprCpp' with subfolders 'Pcl2' and 'Test'. The main editor window shows the 'Expr.g4' grammar file with the following content:

```
1 grammar Expr;
2
3 /** The start rule; begin parsing here. */
4 prog: stat+ ;
5
6 stat: expr NEWLINE
7     | ID '=' expr NEWLINE
8     | NEWLINE
9     ;
10
11 expr: expr ('*' '/' ) expr
12     | expr ('+' '-' ) expr
13     | INT
14     | ID
15     | '(' expr ')'
16     ;
17
18 ID : [a-zA-Z]+ ; // match identifiers
19 INT : [0-9]+ ; // match integers
20 NEWLINE: '\r'? '\n' ; // return newlines to parser (is end-state)
21 WS : [ \t]+ -> skip ; // toss out whitespace
22
```

The Syntax Diagram tab at the bottom shows the generated diagram. It features a root node 'prog' which branches to a 'stat' node. The 'stat' node further branches into three paths: 'expr NEWLINE', 'ID '=' expr NEWLINE', and 'NEWLINE'. The 'expr' node branches into four paths: 'expr ('*' '/') expr', 'expr ('+' '-') expr', 'INT', and 'ID'. The diagram uses boxes to represent nodes and arrows to show the flow of the parse tree.

Figure 6. The syntax diagram generated from the grammar file **Expr.g4**.

R. Mak, Install and Configure ANTLR 4 for C++

Right-click the project name and select *New | File* to create an input source file **sample.expr**. Enter the arithmetic expression **100+2*34** that we used before into the file and then save the file (Figure 7).

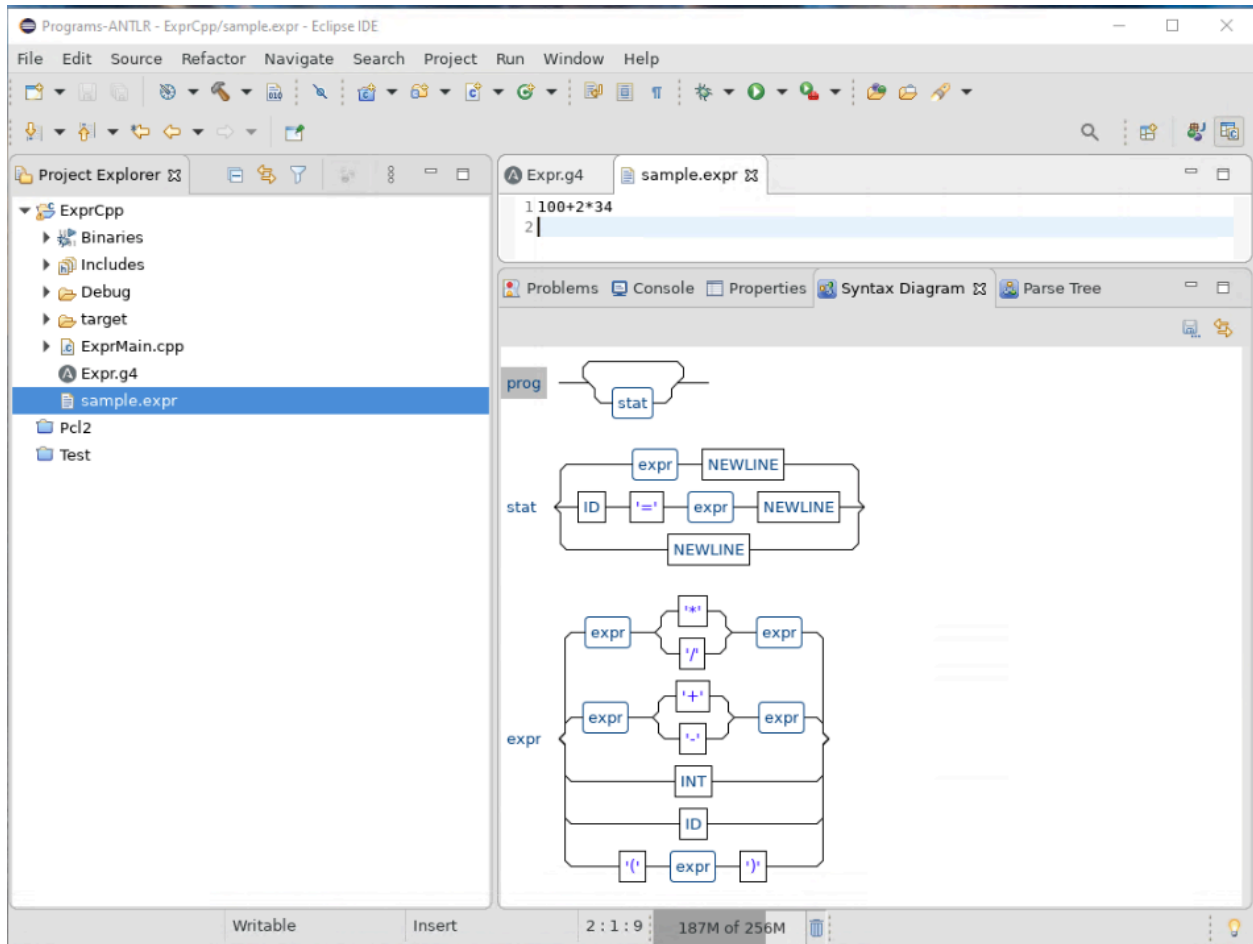


Figure 7. Input source file **sample.expr**.

R. Mak, Install and Configure ANTLR 4 for C++

Copy the contents of the source file onto the clipboard (control-A followed by control-C). Select the **Expr.g4** tab and click on **prog**, the start rule. Paste the contents of the clipboard into the **Expr : prog** panel, and you should see a parse tree that the plugin generated from the source file contents (Figure 8).

The screenshot displays the Eclipse IDE interface for a project named 'ExprCpp'. The main editor shows the ANTLR 4 grammar file 'Expr.g4' with the following content:

```
1 grammar Expr;
2
3 /** The start rule; begin parsing here. */
4 prog: stat+ ;
5
6 stat: expr NEWLINE
7     | ID '=' expr NEWLINE
8     | NEWLINE
9     ;
10
11 expr: expr ('*' '/') expr
12     | expr ('+' '-') expr
13     | INT
14     | ID
15     | '(' expr ')'
16     ;
17
18 ID : [a-zA-Z]+ ; // match identifiers
19 INT : [0-9]+ ; // match integers
20 NEWLINE: '\r'? '\n' ; // return newlines to parser (is end-statement signal)
21 WS : [\t]+ -> skip ; // toss out whitespace
22
```

The 'Parse Tree' panel shows the following tree structure for the input '100+2*34':

```
graph TD
    prog --> stat
    stat --> expr1[expr]
    stat --> n1[\n]
    expr1 --> expr2[expr]
    expr1 --> plus[+]
    expr2 --> 100
    expr2 --> expr3[expr]
    expr3 --> 2
    expr3 --> star[*]
    expr3 --> expr4[expr]
    expr4 --> 34
```

The console panel shows the input text: 'Expr::prog' and '100+2*34'.

Figure 8. A parse tree generated from the input source.

Run a C++-based ANTLR project

To successfully compile an ANTLR project, we must specify where to find the ANTLR 4 header files and libraries.

Right-click on the project name and select *Properties* from the context menu. In the **Properties for ExprCpp** dialog box, select *C/C++ Build | Settings* in the left panel and *GCC C++ Compiler | Dialect* in the main panel. Select the C++ 11 standard (Figure 9).

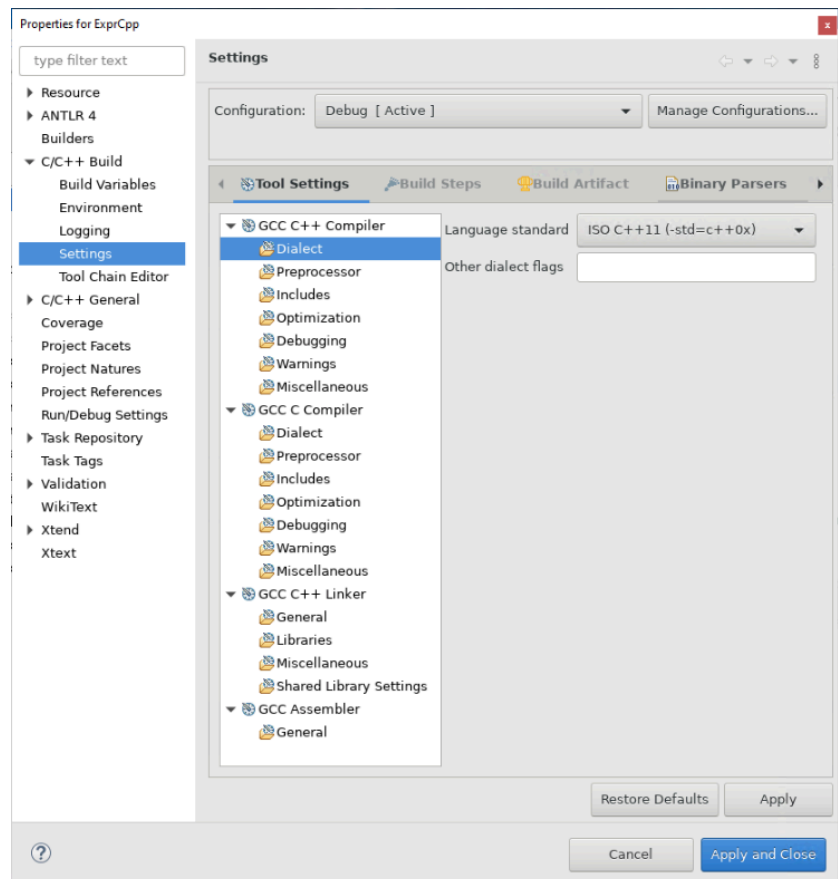


Figure 9. Setting the C++ 11 standard.

R. Mak, Install and Configure ANTLR 4 for C++

You must specify paths to directories that contain include files that the project requires. Select *GCC C++ Compiler | Includes* in the main panel (Figure 10).

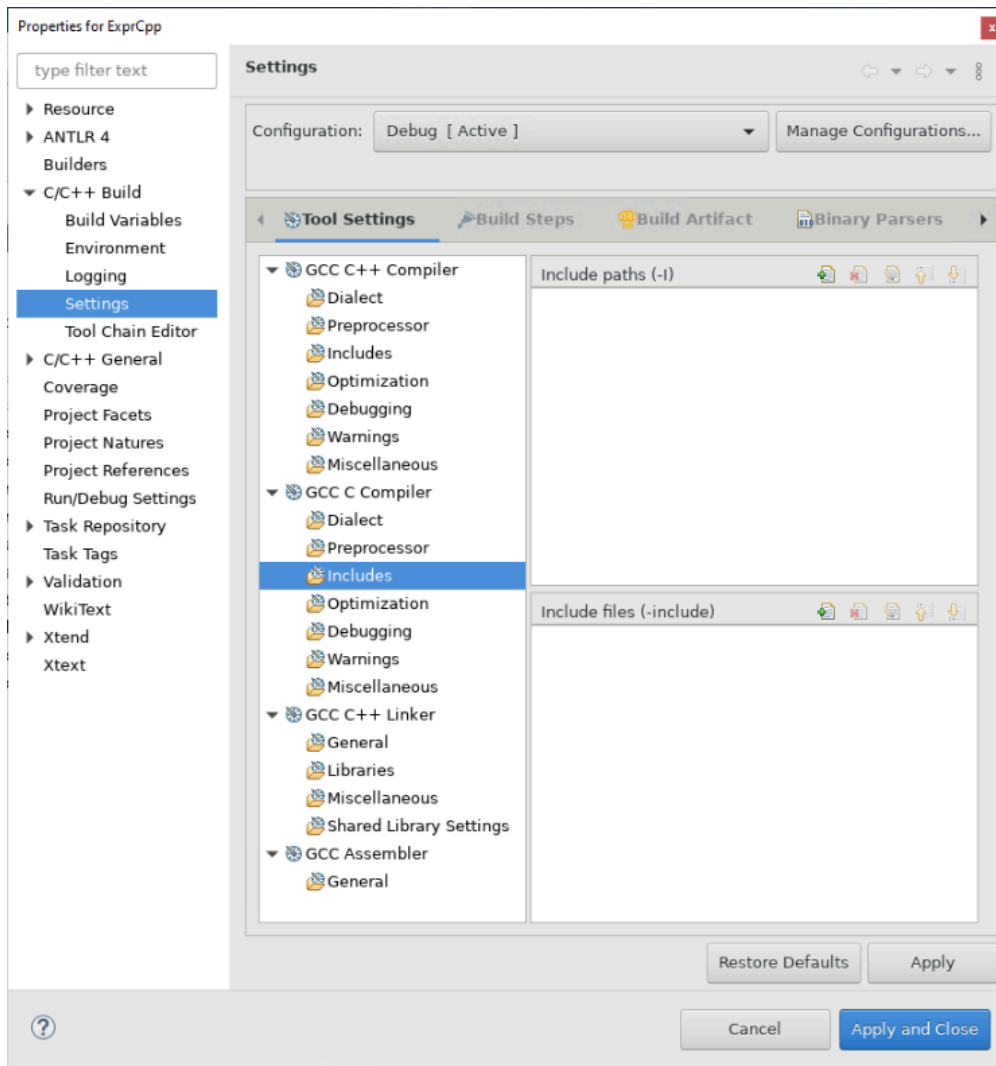


Figure 10. The dialog box to specify include paths to directories that contain needed include files.

In the *Include paths* panel, click the green + to add a path to a directory that contains include files (Figure 11).

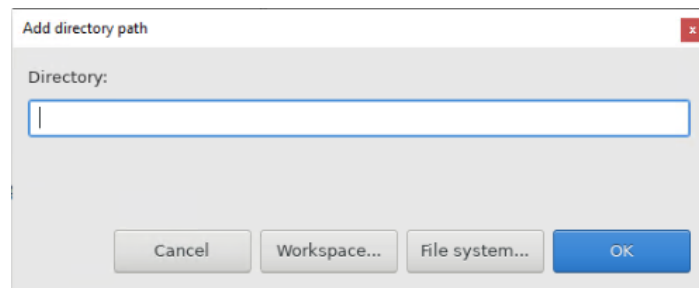


Figure 11. Adding a directory path.

R. Mak, Install and Configure ANTLR 4 for C++

Click the *Workspace* button. In the **Folder selection** dialog box, select the project folder **ExprCpp** (Figure 12). Click the *OK* button, and then click the *OK* button of the **Add directory path** dialog box.

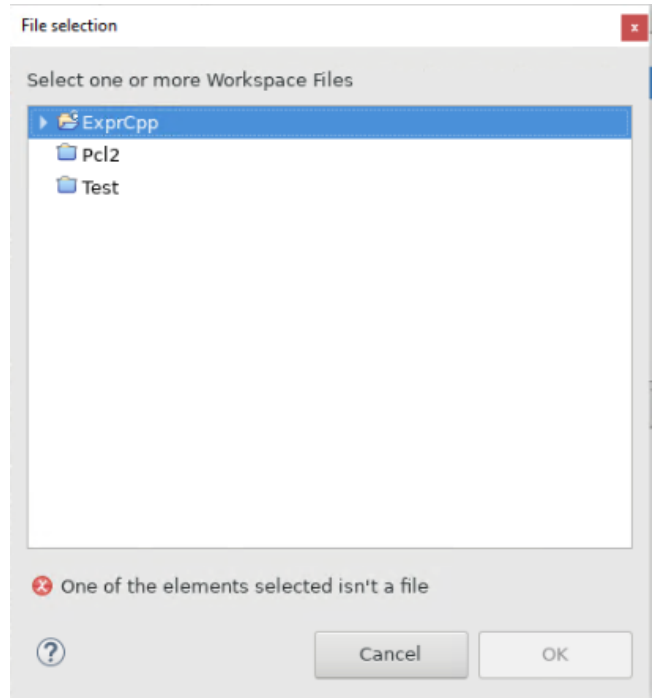


Figure 12. Selecting **ExprCpp** as a directory that contain needed include files.

Repeat the process again, but this time in the **Folder selection** dialog box, select the folder **ExprCpp/target/generated-sources/antlr4** (Figure 13). Click the *OK* button, and then click the *OK* button of the **Add directory path** dialog box.

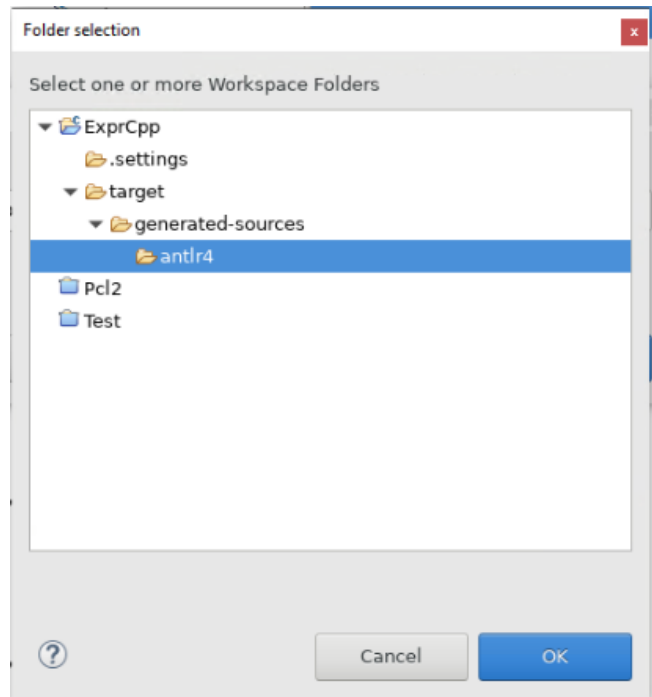


Figure 13. Selecting **ExprCpp/target/generated-sources/antlr4** as a directory that contains needed include files.

R. Mak, Install and Configure ANTLR 4 for C++

Repeat the process one more time, but this time in the **Add directory path** dialog box, click the *File system* button. Click *Other Locations* in the left panel, then *Computer* in the main panel. Select the directory `/usr/local/include/antlr4-runtime` which contains the ANTLR header files (Figure 14). Click the *OK* button.

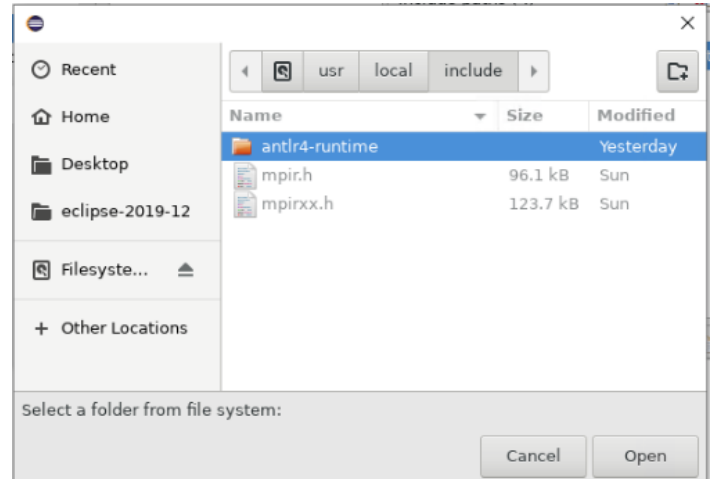


Figure 14. Selecting directory `/usr/local/include/antlr4-runtime` which contains the ANTLR header files.

The project will have three directories that contain the required include files (Figure 15). Click the *Apply and Close* button.

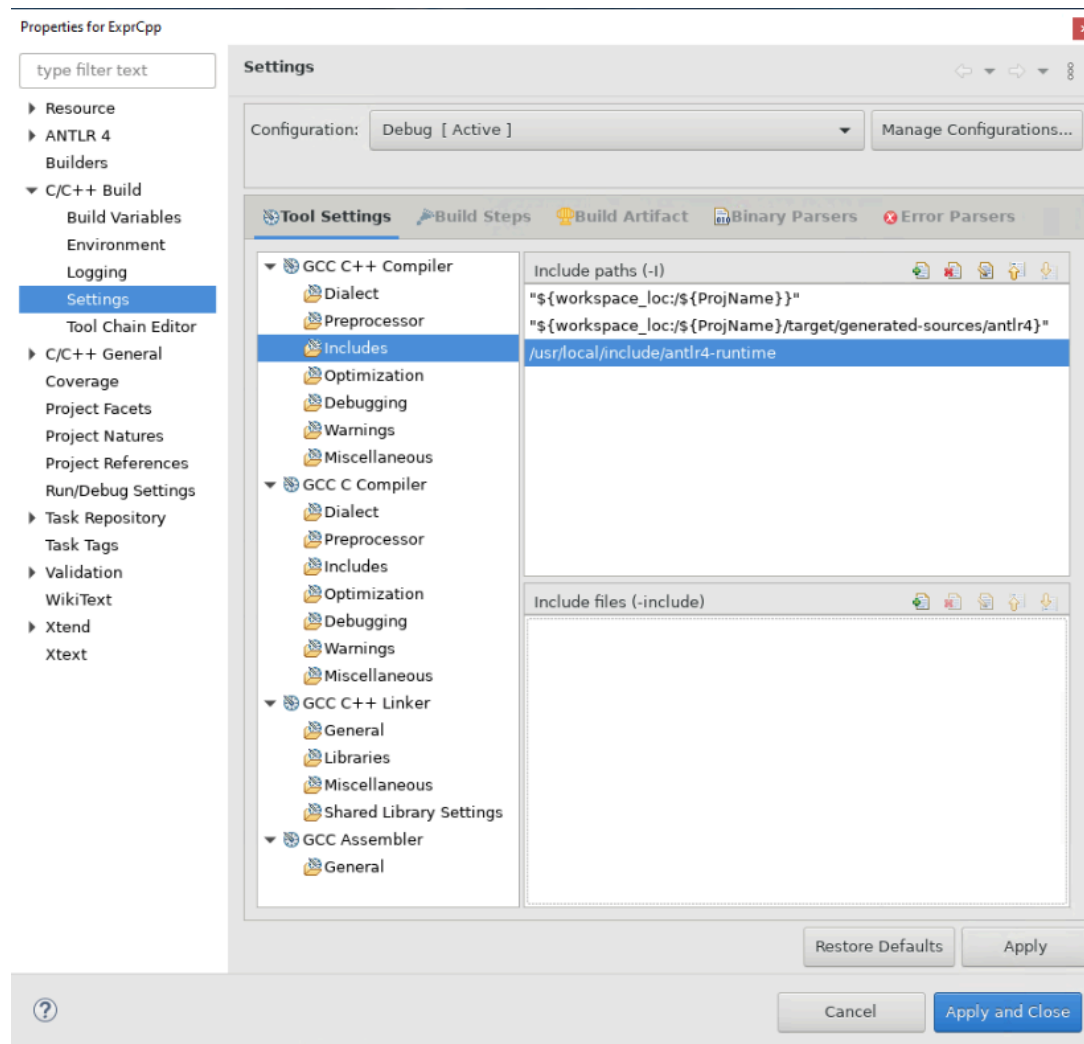


Figure 15. Three directories that contain required include files.

R. Mak, Install and Configure ANTLR 4 for C++

Now and select *GCC C++ Linker | Libraries* (Figure 16).

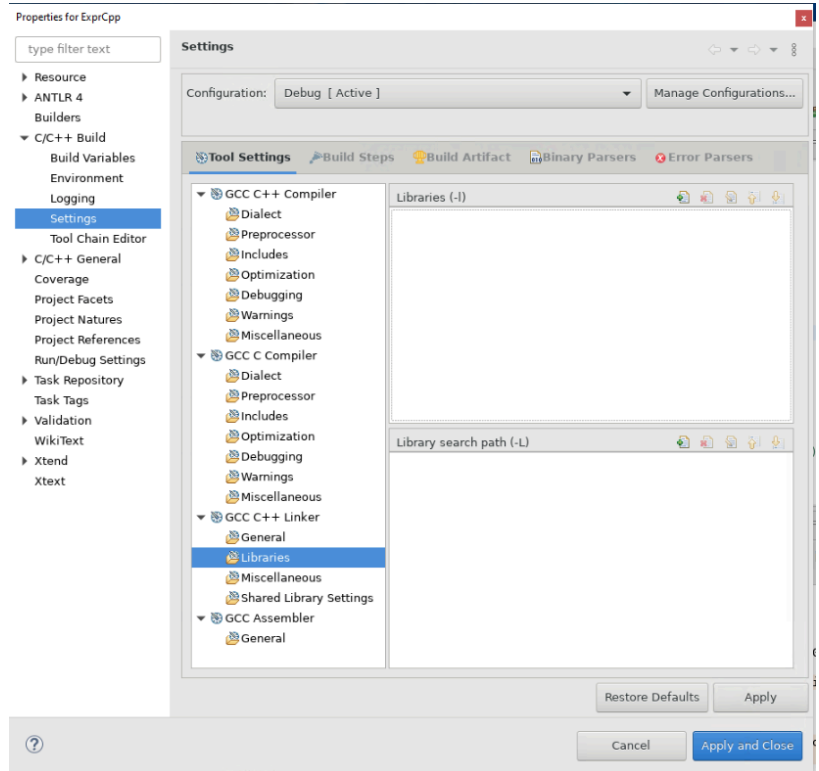


Figure 16. The dialog box to specify needed libraries.

In the top Libraries panel, click the green + to add a library. In the Enter Value dialog box, enter “antlr4-runtime” and click the OK button. (Figure 17)

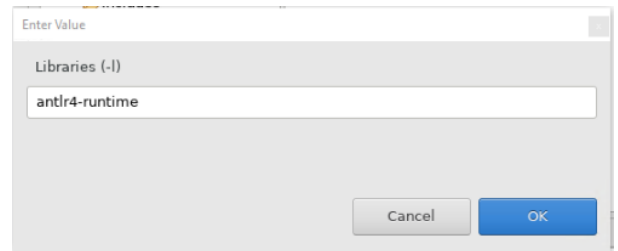


Figure 17: Specifying the antlr4-runtime library.

If the ANTLR 4 runtime library is not in the standard library directory `/usr/local/lib`, then you must also set the path of the directory containing the library. Do this in the *Library search path* panel (Figure 18).

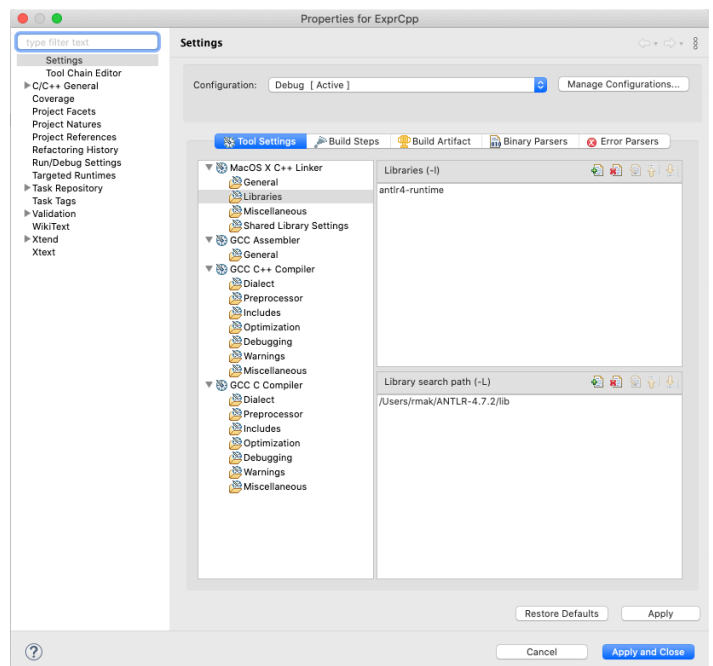


Figure 18: Specifying the directory that contains the ANTLR 4 runtime library.

R. Mak, Install and Configure ANTLR 4 for C++

Now the project can access the ANTLR 4 runtime library. Click the *Apply and Close* button.

To run in Eclipse, our sample project needs a main. Right-click the project name and select *New | Source file* from the context menu. Enter **ExprMain.cpp** as the name of the source file (Figure 19). Click the *Finish* button.

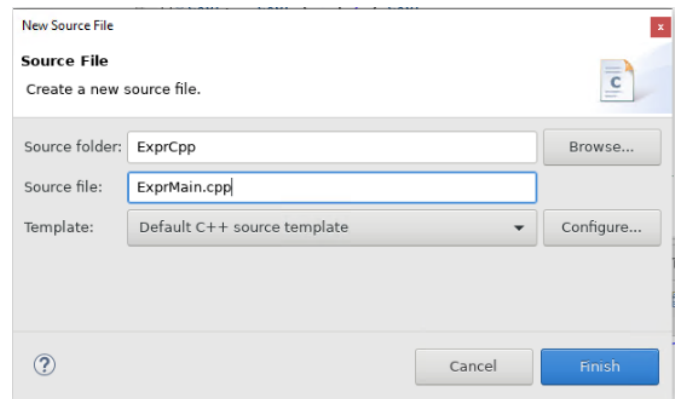


Figure 19. Creating the main source file.

Enter the following code in the editor for source file `ExprMain.cpp`:

```
#include <iostream>
#include <fstream>

#include "antlr4-runtime.h"
#include "ExprLexer.h"
#include "ExprParser.h"

using namespace antlr3cpp;
using namespace antlr4;
using namespace std;

int main(int argc, const char *args[])
{
    ifstream ins;

    // Create the input stream.
    ins.open(args[1]);
    ANTLRInputStream input(ins);

    // Create a lexer which scans the input stream
    // to create a token stream.
    ExprLexer lexer(&input);
    CommonTokenStream tokens(&lexer);

    // Print the token stream.
    cout << "Tokens:" << endl;
    tokens.fill();
    for (Token *token : tokens.getTokens())
    {
        std::cout << token->toString() << std::endl;
    }

    // Create a parser which parses the token stream
    // to create a parse tree.
    ExprParser parser(&tokens);
    tree::ParseTree *tree = parser.prog();

    // Print the parse tree in Lisp format.
    cout << endl << "Parse tree (Lisp format):" << endl;
    std::cout << tree->toStringTree(&parser) << endl;

    return 0;
}
```


R. Mak, Install and Configure ANTLR 4 for C++

Select the project name and click the hammer icon at the top to compile and link the project (Figure 20). There may be compiler warnings that you can safely ignore.

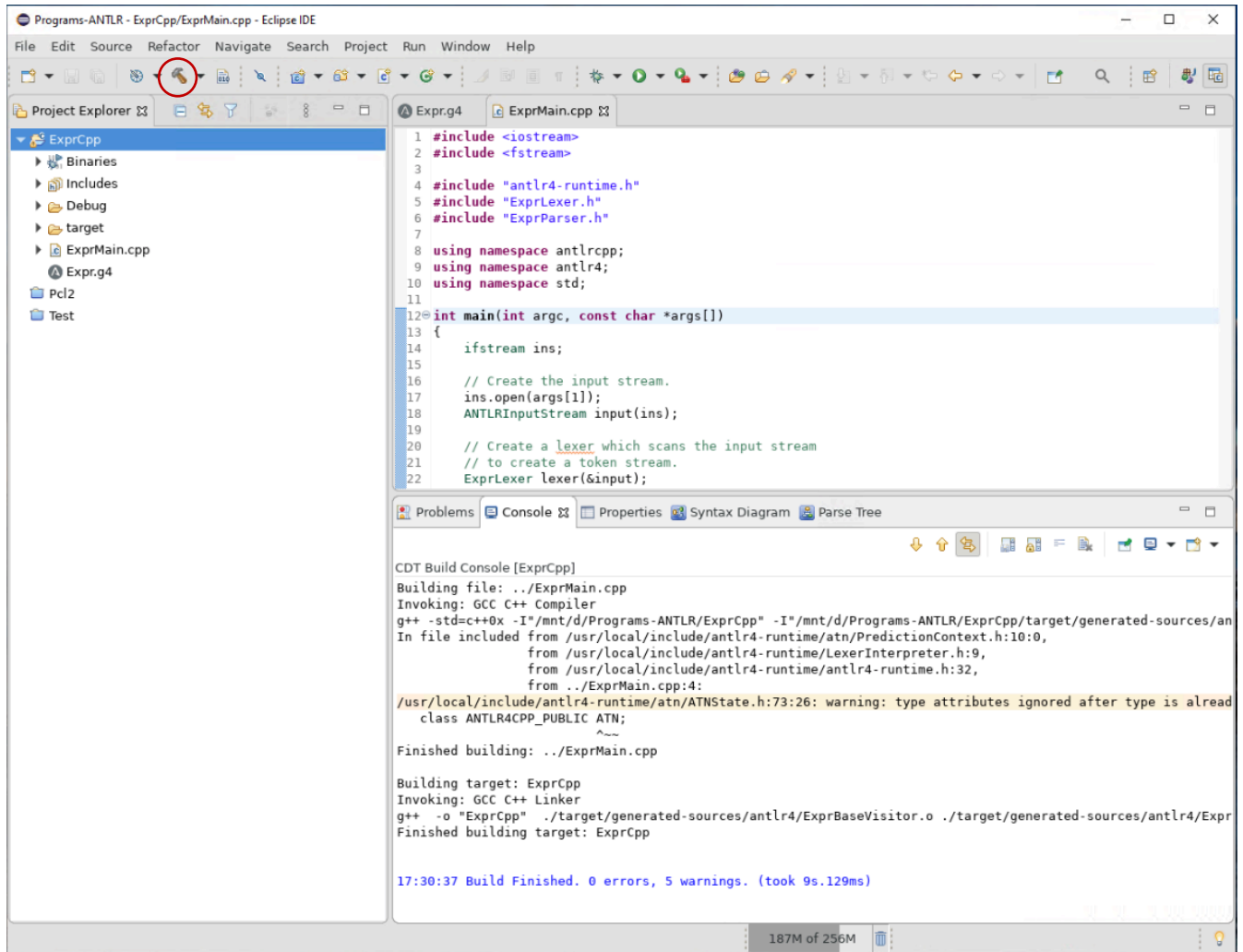


Figure 20. Compiling and linking project **ExprCpp**.

Before you can run the program, you must create a run configuration for this project. Select the project name. From the *Run* dropdown menu at the top, select *Run Configurations*. In the **Create, manage, and run configurations** dialog box, select *C/C++ Application* in the left panel and click the *New Launch Configuration* button above it (the document icon with the yellow plus sign).

R. Mak, Install and Configure ANTLR 4 for C++

In the right panel, type “ExprCpp” for the configuration name (Figure 21). If the *Project* field is empty, click the first *Browse* button to select the project.

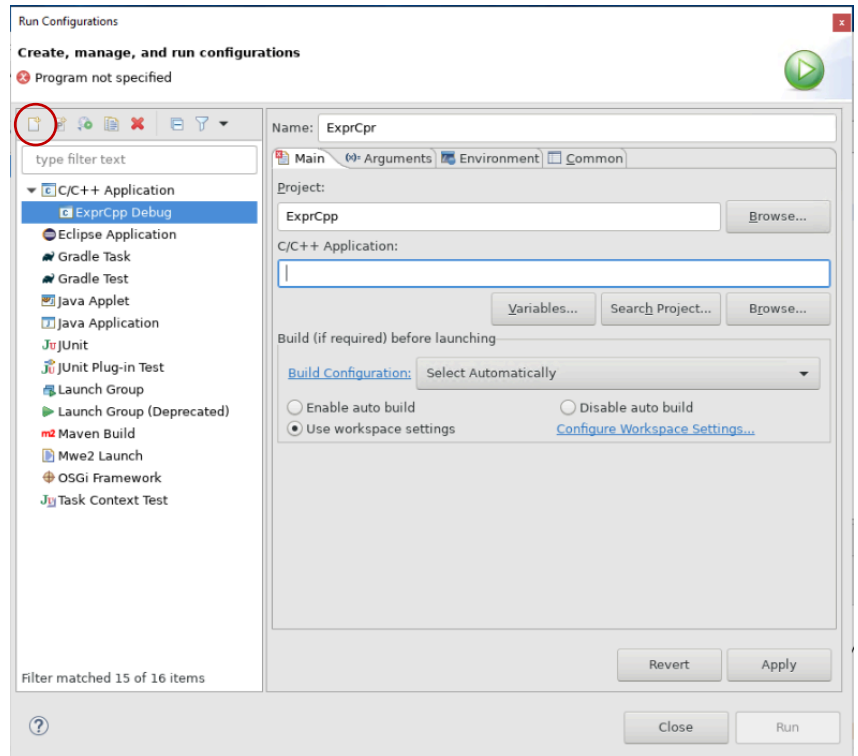


Figure 21. Creating a run configuration.

If the *C/C++ Application* field is empty, click the *Search Project* button. In the **Program Selection** dialog box, select **ExprCpp** and click the *OK* button (Figure 22).

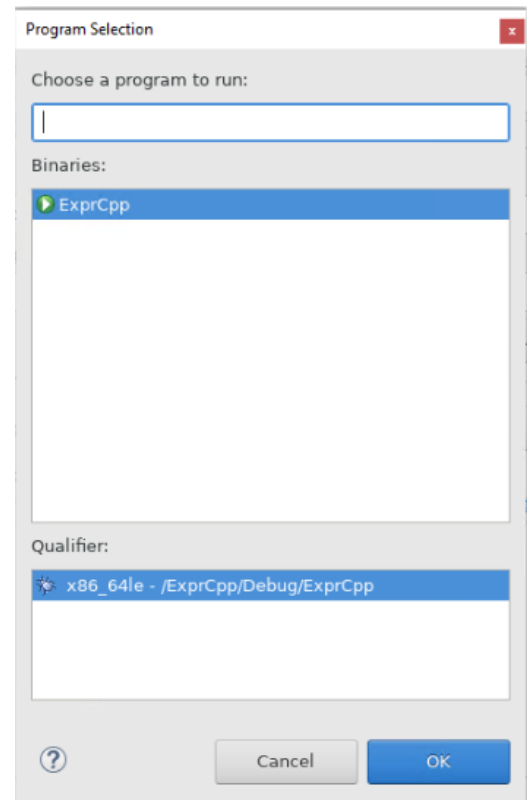


Figure 22. Selecting the program to run.

R. Mak, Install and Configure ANTLR 4 for C++

Click the *Arguments* tab and enter **sample.expr** as the program's command-line argument, the source file to compile. (Figure 23).

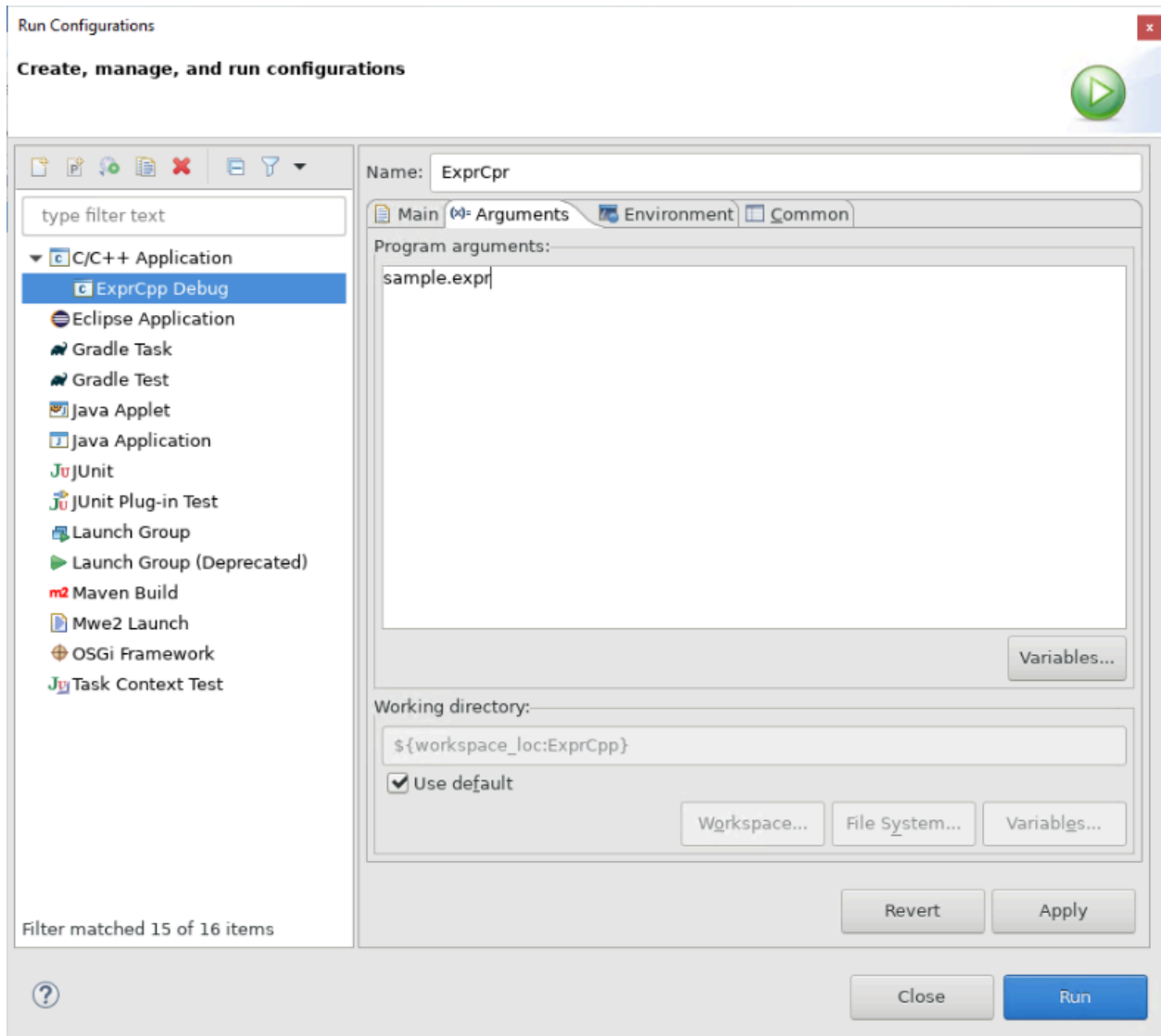
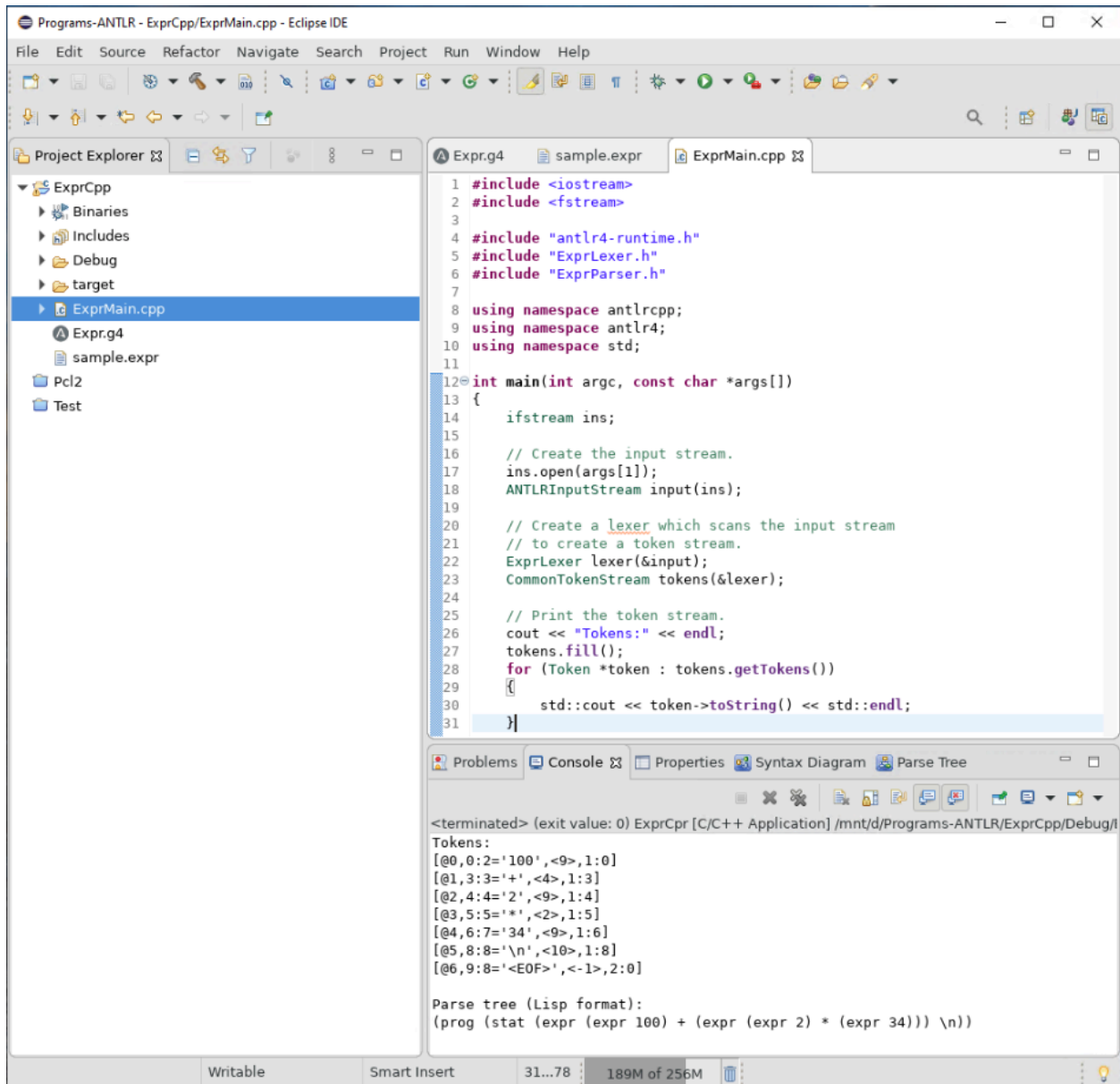


Figure 23. Specifying the command-line argument **sample.expr**, the source file to compile.

R. Mak, Install and Configure ANTLR 4 for C++

Click the *Apply* button and then the *Run* button. You should see output from running the generated compiler in the *Console* tab of the main Eclipse window. First is a list of tokens from the token stream generated by the lexer, and then the parse tree generated by the parser in Lisp format (Figure 24).



The screenshot shows the Eclipse IDE interface. The Project Explorer on the left shows a project named 'ExprCpp' with sub-projects 'Expr.g4' and 'sample.expr'. The main editor displays the source code for 'ExprMain.cpp'. The code includes headers for ANTLR4 runtime and the generated lexer and parser. The main function reads an input stream, creates a lexer and parser, and prints the token stream and parse tree.

```
1 #include <iostream>
2 #include <fstream>
3
4 #include "antlr4-runtime.h"
5 #include "ExprLexer.h"
6 #include "ExprParser.h"
7
8 using namespace antlr3cpp;
9 using namespace antlr4;
10 using namespace std;
11
12 int main(int argc, const char *args[])
13 {
14     ifstream ins;
15
16     // Create the input stream.
17     ins.open(args[1]);
18     ANTLRInputStream input(ins);
19
20     // Create a lexer which scans the input stream
21     // to create a token stream.
22     ExprLexer lexer(&input);
23     CommonTokenStream tokens(&lexer);
24
25     // Print the token stream.
26     cout << "Tokens:" << endl;
27     tokens.fill();
28     for (Token *token : tokens.getTokens())
29     {
30         std::cout << token->toString() << std::endl;
31     }
32 }
```

The Console tab shows the output of the program:

```
<terminated> (exit value: 0) ExprCpr [C/C++ Application] /mnt/d/Programs-ANTLR/ExprCpp/Debug/1
Tokens:
[@0,0:2='100',<9>,1:0]
[@1,3:3='+',<4>,1:3]
[@2,4:4='2',<9>,1:4]
[@3,5:5='*',<2>,1:5]
[@4,6:7='34',<9>,1:6]
[@5,8:8='\n',<10>,1:8]
[@6,9:8='<EOF>',<-1>,2:0]

Parse tree (Lisp format):
(prog (stat (expr (expr 100) + (expr (expr 2) * (expr 34))) \n))
```

Figure 24. Output from running the generated compiler.