

# Install and Configure ANTLR 4 for C++

Ronald Mak

Department of Computer Engineering

Department of Computer Science

January 20, 2019

## Introduction

ANTLR 4 (“Another Tool for Language Recognition”) is a “compiler-compiler”, a tool that generates components of a compiler for a programming language. See <https://www.antlr.org>. We will use this tool in our compiler design class. Given a grammar file for a programming language, it will generate a parser, lexer (scanner), and parse tree classes written in Java or C++. Other included tools create graphical syntax diagrams and parse tree diagrams.

ANTLR is easy to use inside Eclipse if we install its plugin. Therefore, the first step is to install the plugin, which was originally designed for Java. To do so, follow the instructions in “Install and Configure ANTLR 4 on Eclipse and Ubuntu” (<http://www.cs.sjsu.edu/~mak/tutorials/InstallANTLR4.pdf>).

Then we will concentrate on having ANTLR 4 generate compiler components written in C++.

### Related documents:

- “Install and Configure VirtualBox on Windows”  
<http://www.cs.sjsu.edu/~mak/tutorials/InstallVirtualBox.pdf>
- “Install and Configure Ubuntu on a VirtualBox Virtual Machine”  
<http://www.cs.sjsu.edu/~mak/tutorials/InstallUbuntu.pdf>
- “Install Eclipse on Ubuntu for Java and C++ Development”  
<http://www.cs.sjsu.edu/~mak/tutorials/InstallEclipse.pdf>
- “Install and Configure ANTLR 4 on Eclipse and Ubuntu”  
<http://www.cs.sjsu.edu/~mak/tutorials/InstallANTLR4.pdf>

## A simple expression grammar

In your home directory, **create subdirectory “example” and change to it**. Create a grammar “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. To generate parser and lexer files in C++ instead of Java, add the `-Dlanguage=Cpp` option to the `antlr4` alias:

```
antlr4 -Dlanguage=Cpp Expr.g4
```

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

## Install developer tools

The Ubuntu distribution does not include developer tools such as the GNU `gcc` and `g++` compilers and GNU `make`. See <https://help.ubuntu.com/community/InstallingCompilers> and follow its instructions:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install build-essential
```

**Note:** If you receive the error message “Temporary failure resolving us.archive.ubuntu.com”, the fix is to enter the following command:

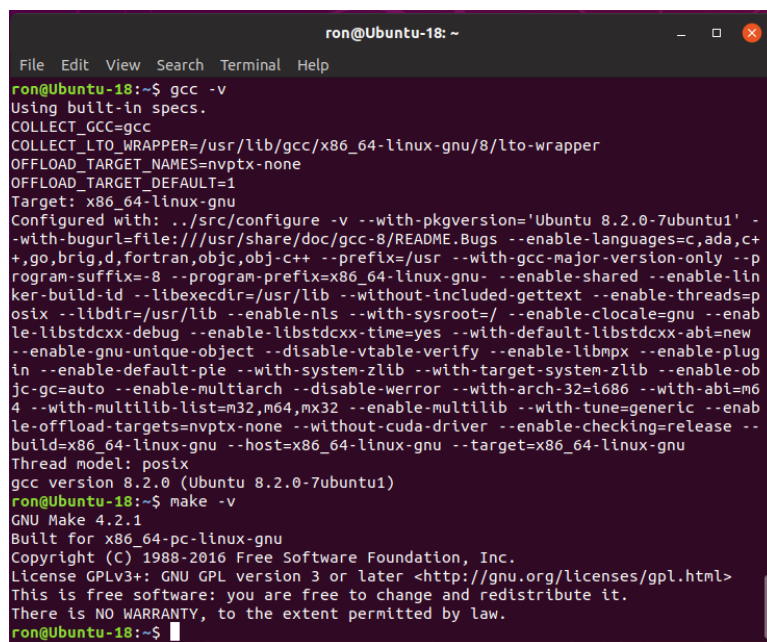
```
sudo dhclient -v -4
```

and then try again. See <https://askubuntu.com/questions/884604/temporary-failure-resolving-us-archive-ubuntu-com-live-usb-rescue>.

After installing the developer tools, you should be able to verify that `gcc`, `g++`, and `make` were properly installed by entering the following commands in the Ubuntu terminal (Figure 1):

```
gcc -v
g++ -v
make -v
```

Figure 1. Verify that `gcc`, `g++`, and `make` were properly installed.



```
ron@Ubuntu-18: ~
File Edit View Search Terminal Help
ron@Ubuntu-18:~$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/8/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 8.2.0-7ubuntu1' --with-bugurl=file:///usr/share/doc/gcc-8/README.Bugs --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++ --prefix=/usr --with-gcc-major-version-only --program-suffix=-8 --program-prefix=x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --libdir=/usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-libmpx --enable-plugin --enable-default-pie --with-system-zlib --with-target-system-zlib --enable-objc-gc=auto --enable-multiarch --disable-werror --with-arch=32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-offload-targets=nvptx-none --without-cuda-driver --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 8.2.0 (Ubuntu 8.2.0-7ubuntu1)
ron@Ubuntu-18:~$ g++ -v
GNU Make 4.2.1
Built for x86_64-pc-linux-gnu
Copyright (C) 1988-2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
ron@Ubuntu-18:~$ make -v
```

## Install cmake

In the Ubuntu terminal window, enter the following command to install the `cmake` utility:

```
sudo apt-get install cmake
```

## Download and install the C++ runtime for ANTLR

Go to <https://github.com/antlr/antlr4> and click the green *Clone or download* button. Then click the *Download ZIP* button. Save the downloaded zip `antlr4-master.zip` in your home directory and unzip it. You will get a directory `antlr4-master`. **Change to directory `antlr4-master/runtime/Cpp`**. Go to <https://github.com/antlr/antlr4/tree/master/runtime/Cpp> and see the instructions under “Compiling on Linux”. Type the following in the Ubuntu terminal window:

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

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

**Enter subdirectory `antlr4-master/runtime/Cpp/run/usr/local/include`**. Type the following command to copy the directory containing the ANTLR header files to the standard system location:

```
sudo cp -r antlr4-runtime /usr/local/include
```

**Enter subdirectory `antlr4-master/runtime/Cpp/run/usr/local/lib`**. Type the following commands to copy the ANTLR libraries to the standard system location and make them available to programs:

```
sudo cp * /usr/local/lib
sudo ldconfig
```

Now you can **change to your home directory** and delete all of directory `antlr4-master`:

```
rm -r antlr4-master
```

## Install the ANTLR plugin for Eclipse

If you haven’t already, install the ANTLR plugin for Eclipse according to “Install and Configure ANTLR 4 on Eclipse and Ubuntu” (<http://www.cs.sjsu.edu/~mak/tutorials/InstallANTLR4.pdf>).

## 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).

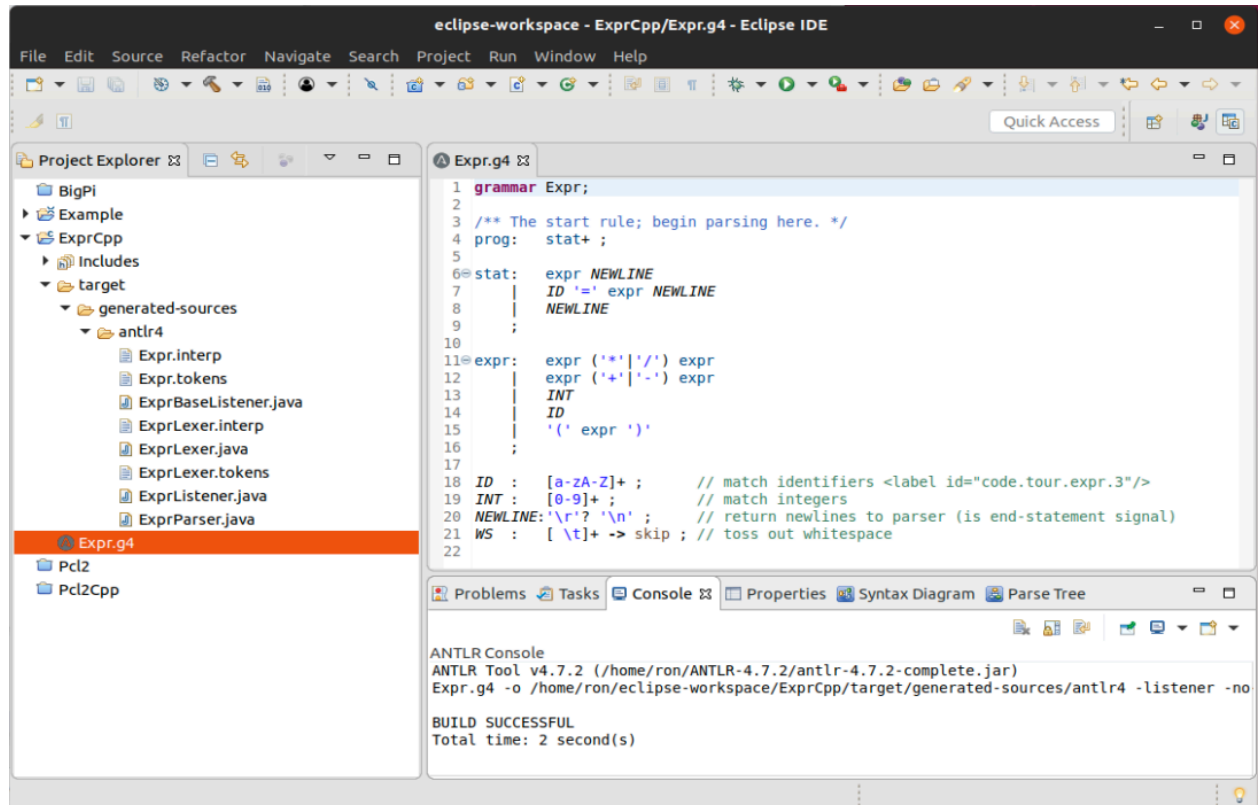


Figure 2. Grammar file Expr.g4 in a C++ project.

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. Make sure in *Distributions* that the 4.7.2 jar file is present and selected (Figure 3). 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>).

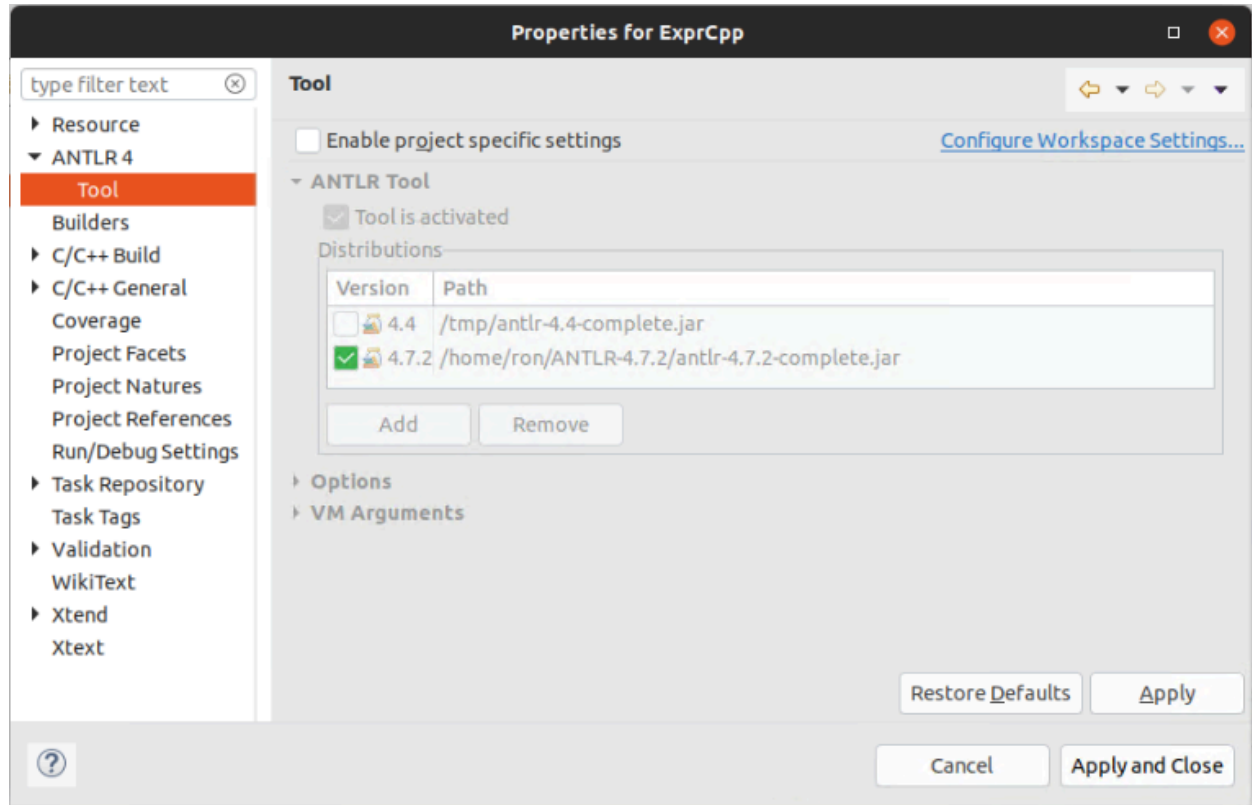


Figure 3. Make sure the 4.7.4 jar file is present and selected.

To change the generated code to C++, we must create an external tool run configuration. From the *Run* dropdown menu at the top, select *External Tools* | *External Tools Configurations ...* In the **External Tools Configurations** dialog box (Figure 4), select ANTLR in the left panel and click the *New launch configuration* button (the button with the yellow + sign in the upper left). Name the run configuration “ExprCpp” and select /ExprCpp/Expr.g4 for the grammar file. Enter the following for *Arguments*:

```
-listener -no-visitor -encoding UTF-8 -Dlanguage=Cpp
```

Click the *Apply* button.

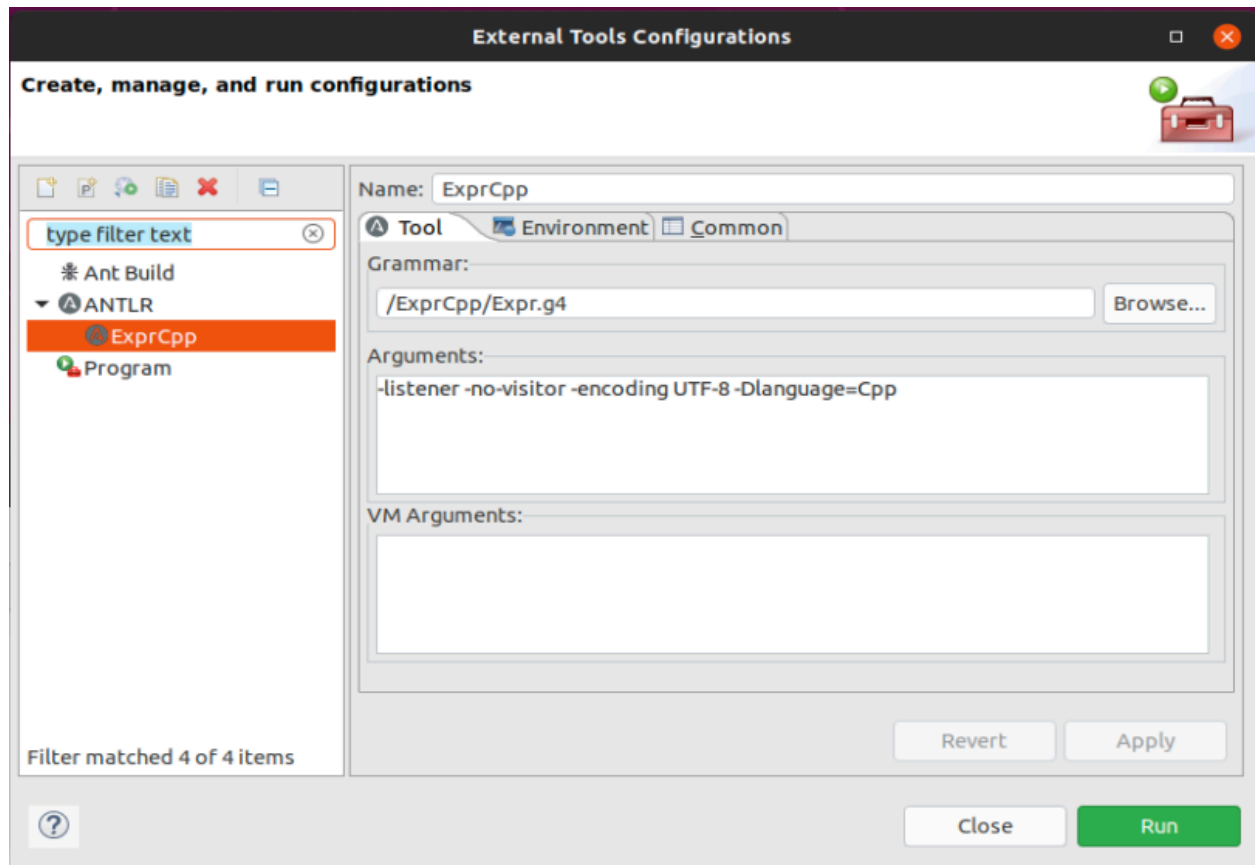


Figure 4. External tools configuration for the grammar file.

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.

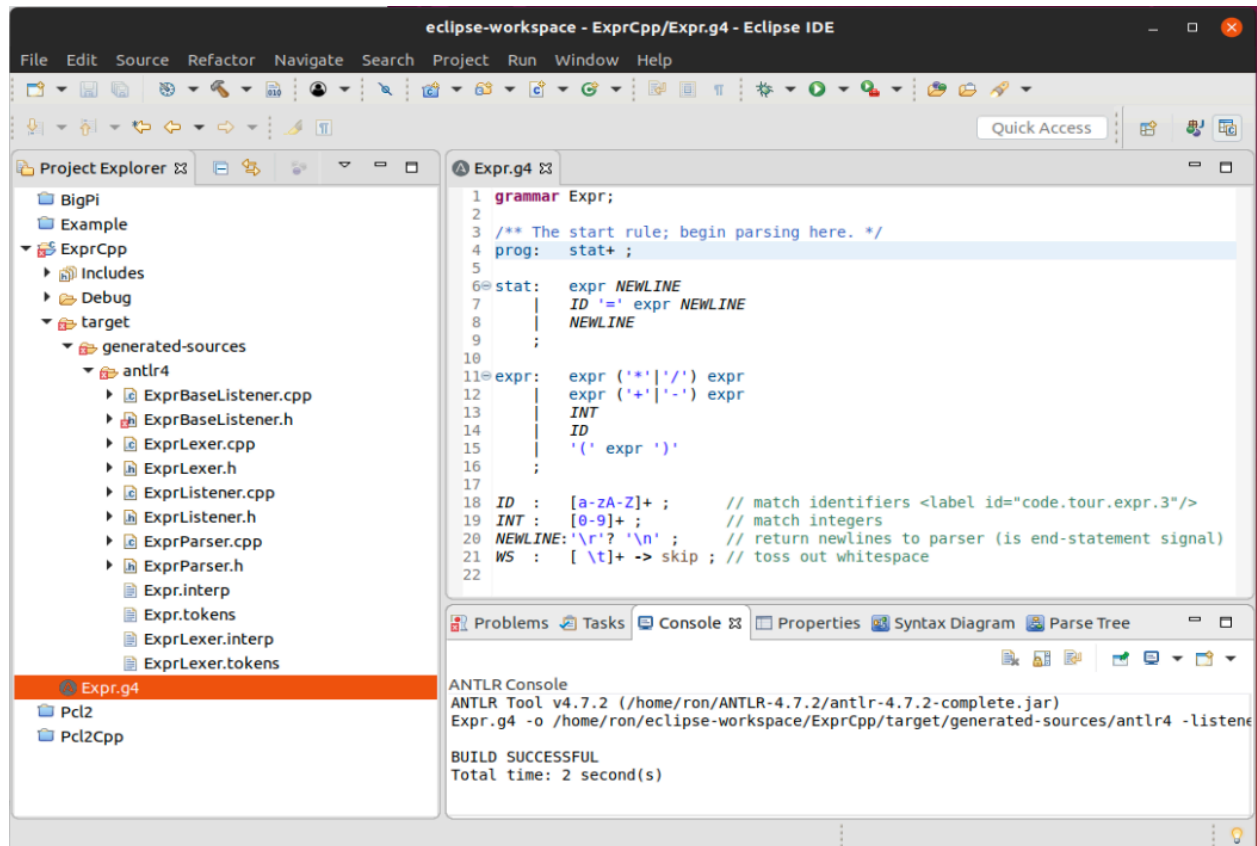


Figure 5. Generated C++ files.

## R. Mak, Install and Configure ANTLR 4 for C++ on Eclipse and Ubuntu

From the *Window* dropdown menu at the top, select *Show View | Other ...* Select *Parse Tree* and *Syntax Diagram* 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).

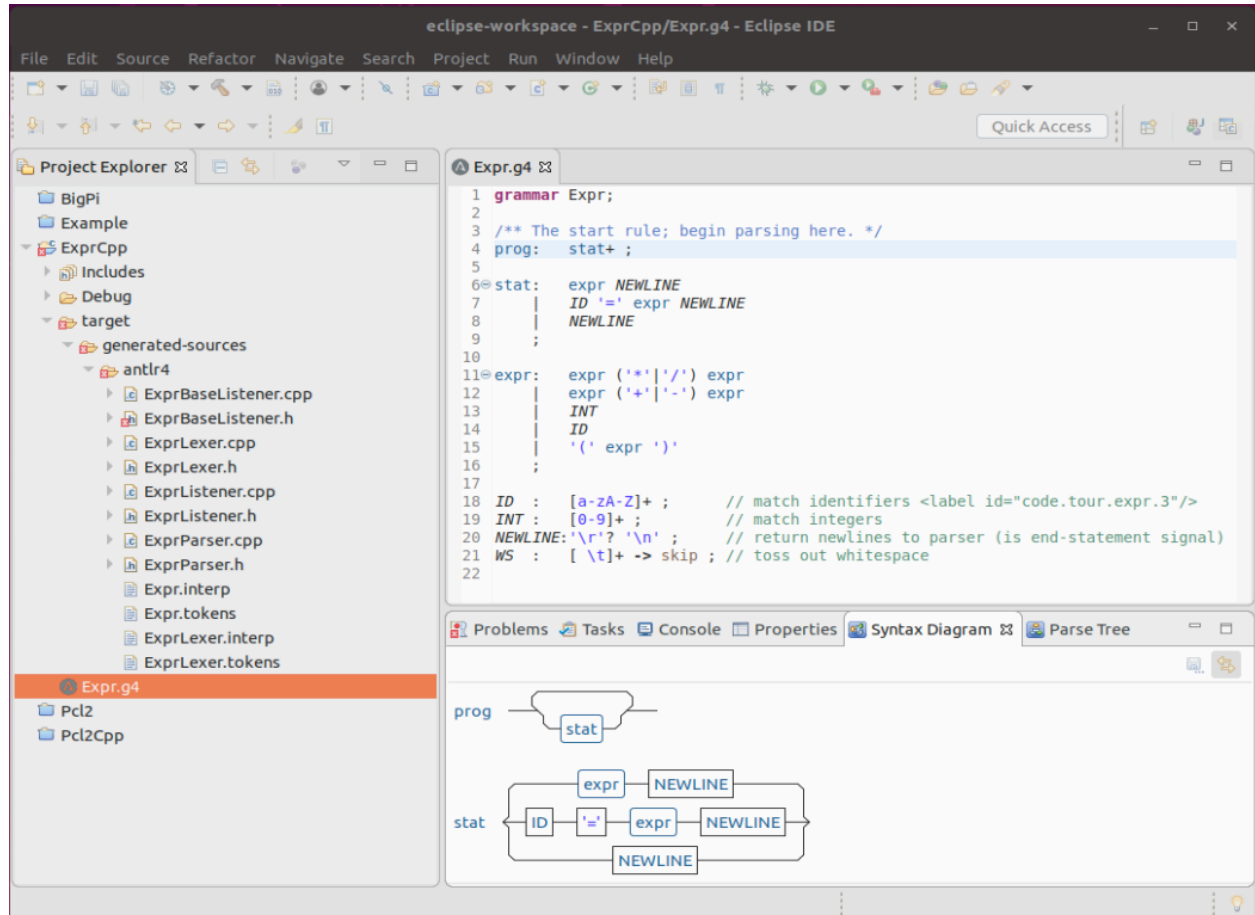


Figure 6. The syntax diagram generated from the *Expr.g4* grammar file *Expr.g4*.



## R. Mak, Install and Configure ANTLR 4 for C++ on Eclipse and Ubuntu

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).

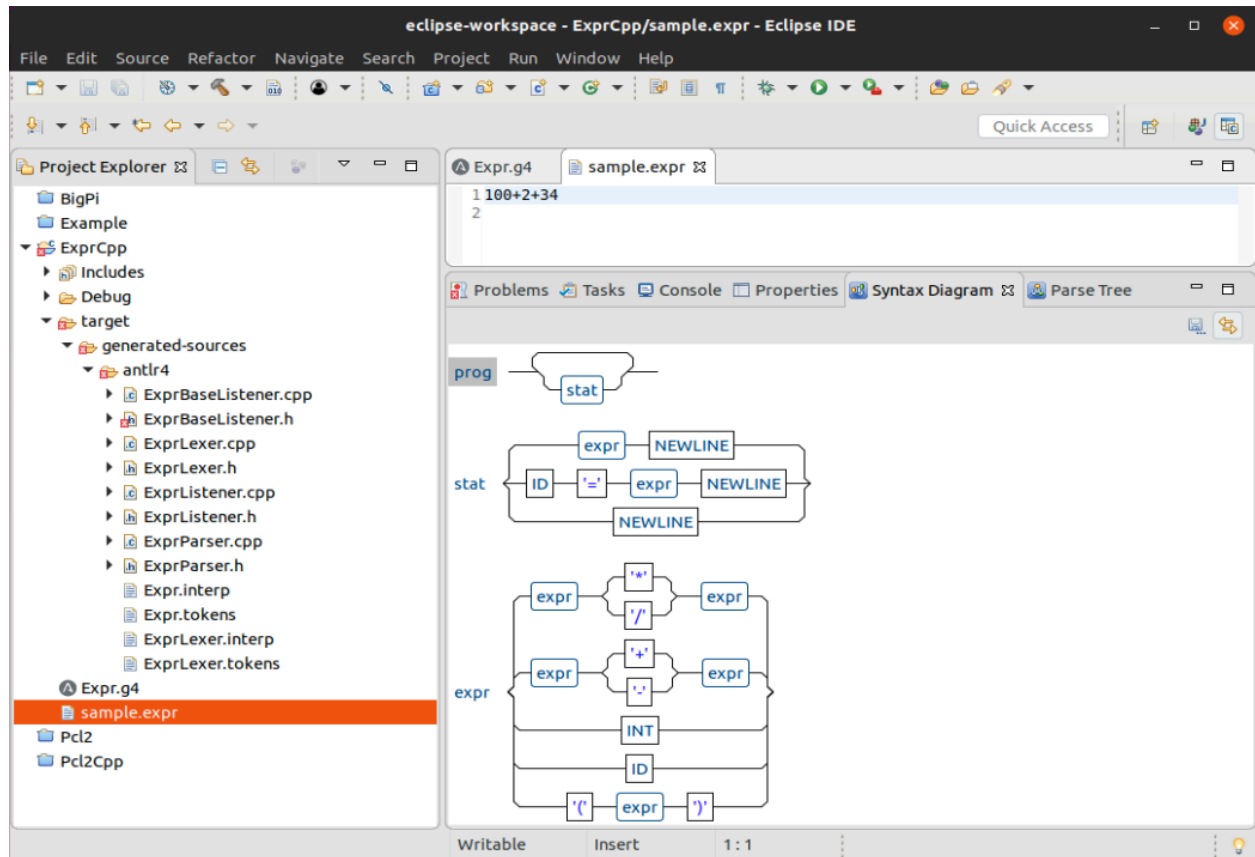


Figure 7. Input source file `sample.expr`.

## R. Mak, Install and Configure ANTLR 4 for C++ on Eclipse and Ubuntu

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).

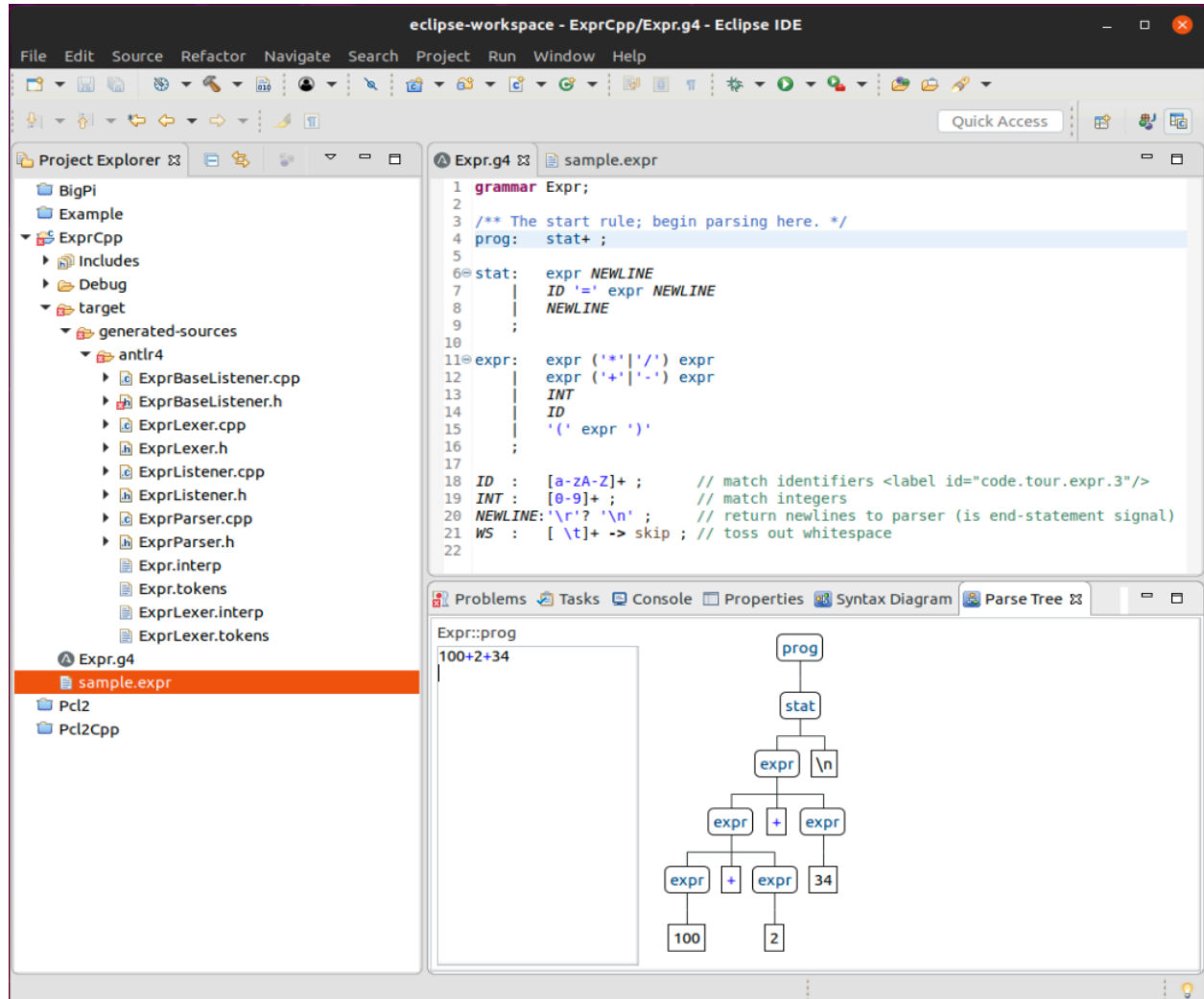


Figure 8. A parse tree generated from the input source.

## Run a C++-based ANTLR project

To successfully compile an ANTLR project, we must specify where to find the necessary 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).

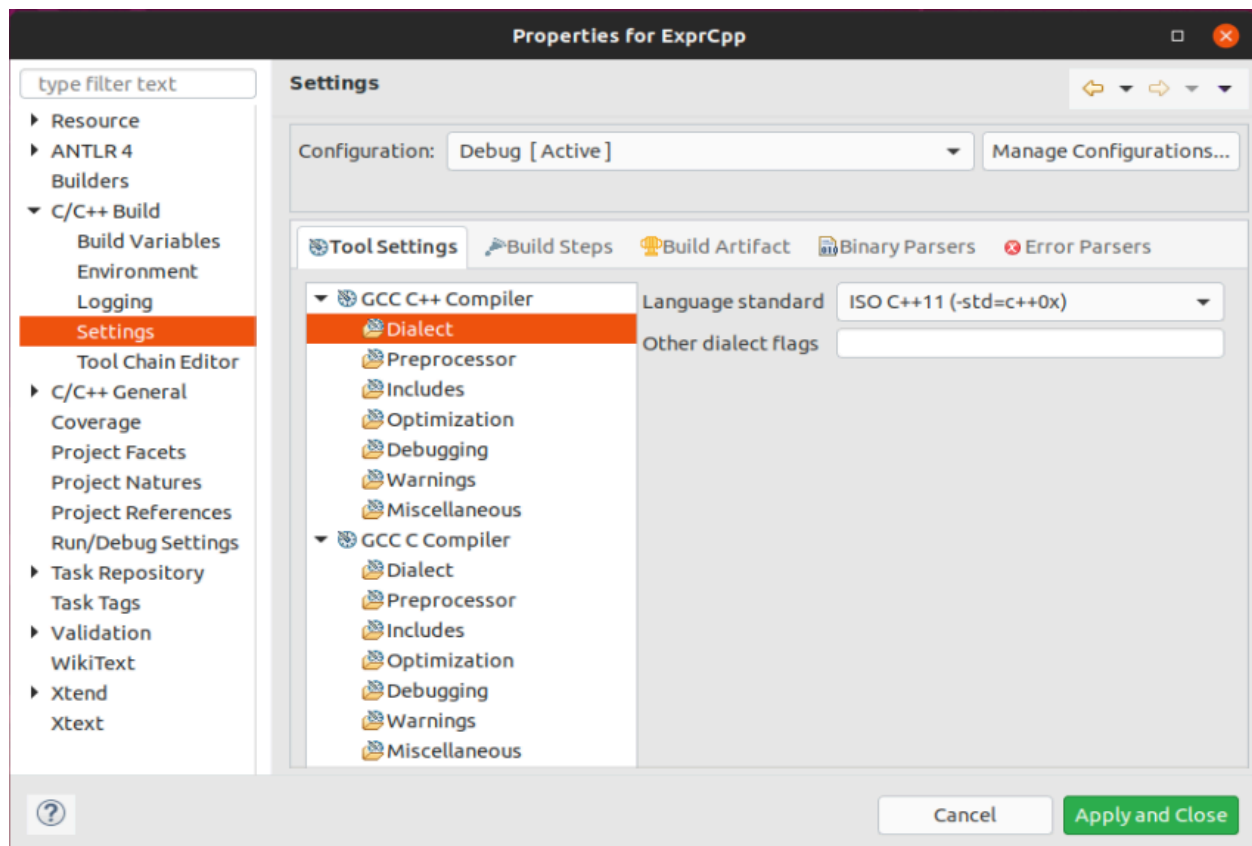


Figure 9. Setting the C++ 11 standard.

Select *GCC C++ Compiler | Includes* in the main panel (Figure 10).

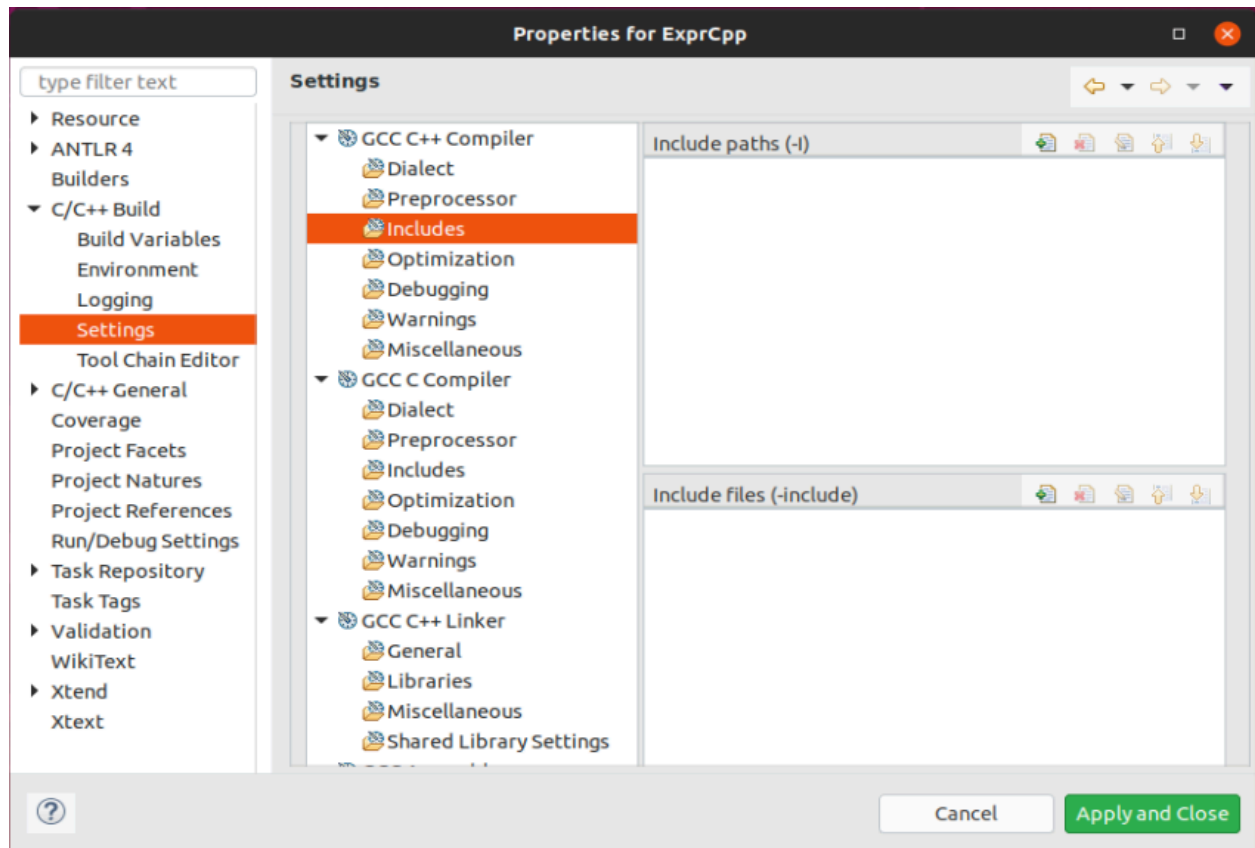


Figure 10. The dialog box to specify include paths to directories that contain needed include files.

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

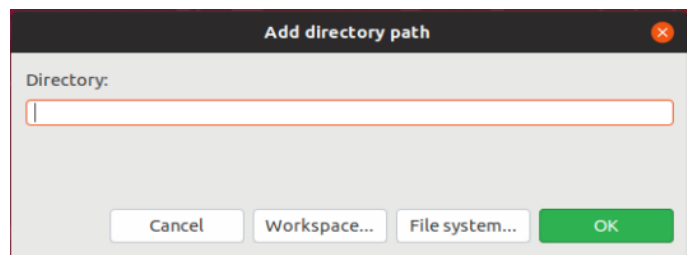


Figure 11. Adding a directory path.

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.

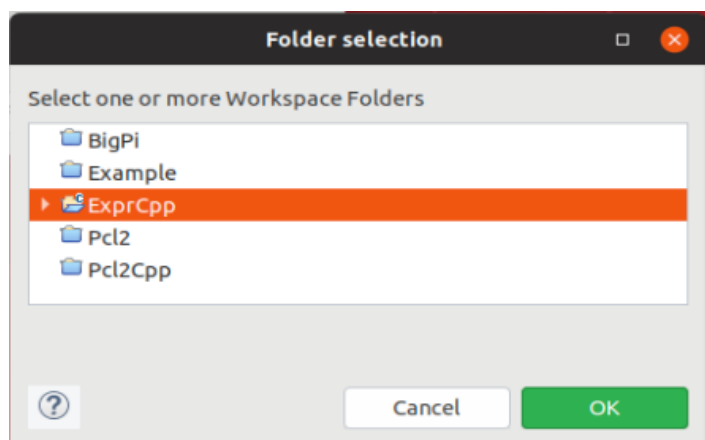
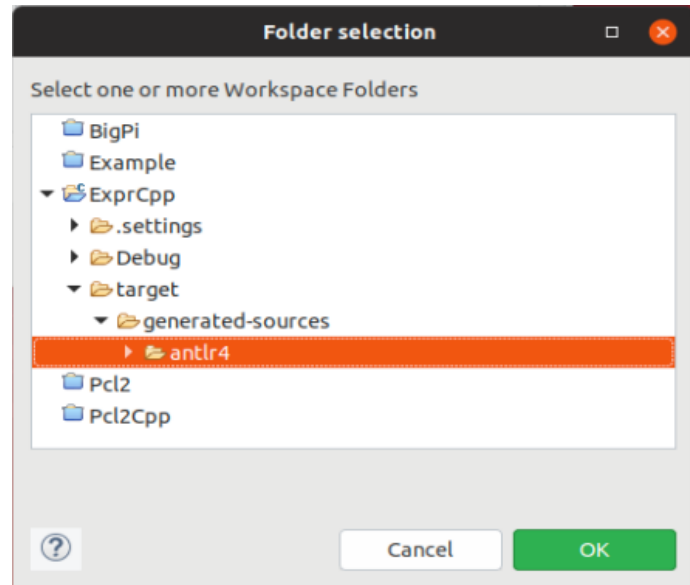


Figure 12. Select ExprCpp as a directory that contains include files.

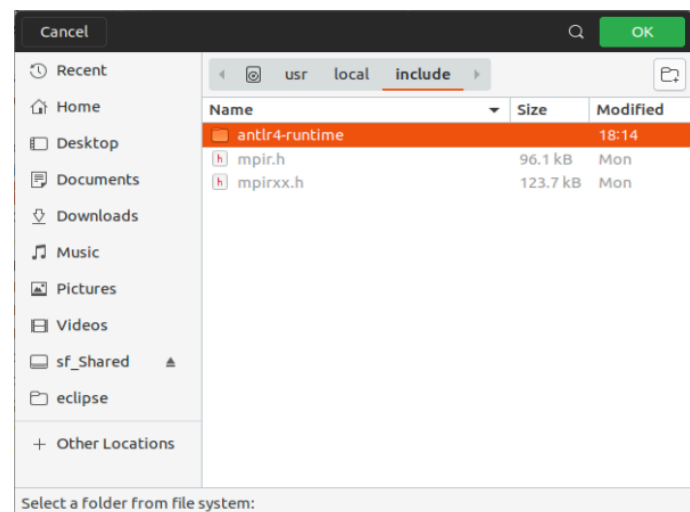
Repeat the process again, but this time in the **Folder selection** dialog box, select the folder ExprCpp/target/generated-sources/antlr4 (Figure 13). Click the *OK* button, and then click the *OK* button of the **Add directory path** dialog box.

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



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. Select /usr/local/include/antlr4-runtime which contains the ANTLR header files.



The project will have three directories that contain needed include files (Figure 15). Click the *Apply and Close* button and then the **Yes** button of the **Settings** dialog box.

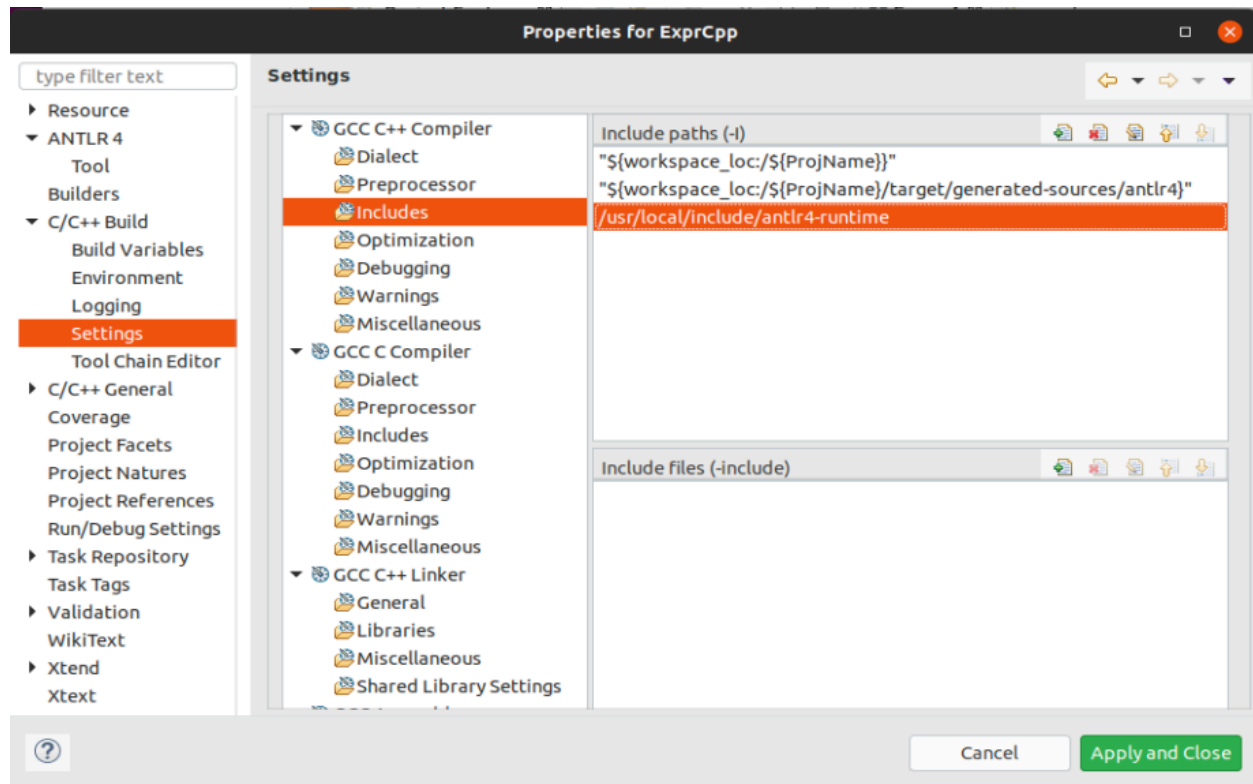


Figure 15. Three directories that contain needed include files.

Open the **Properties for ExprCpp** dialog box again, and select *GCC C++ Linker | Libraries* (Figure 16).

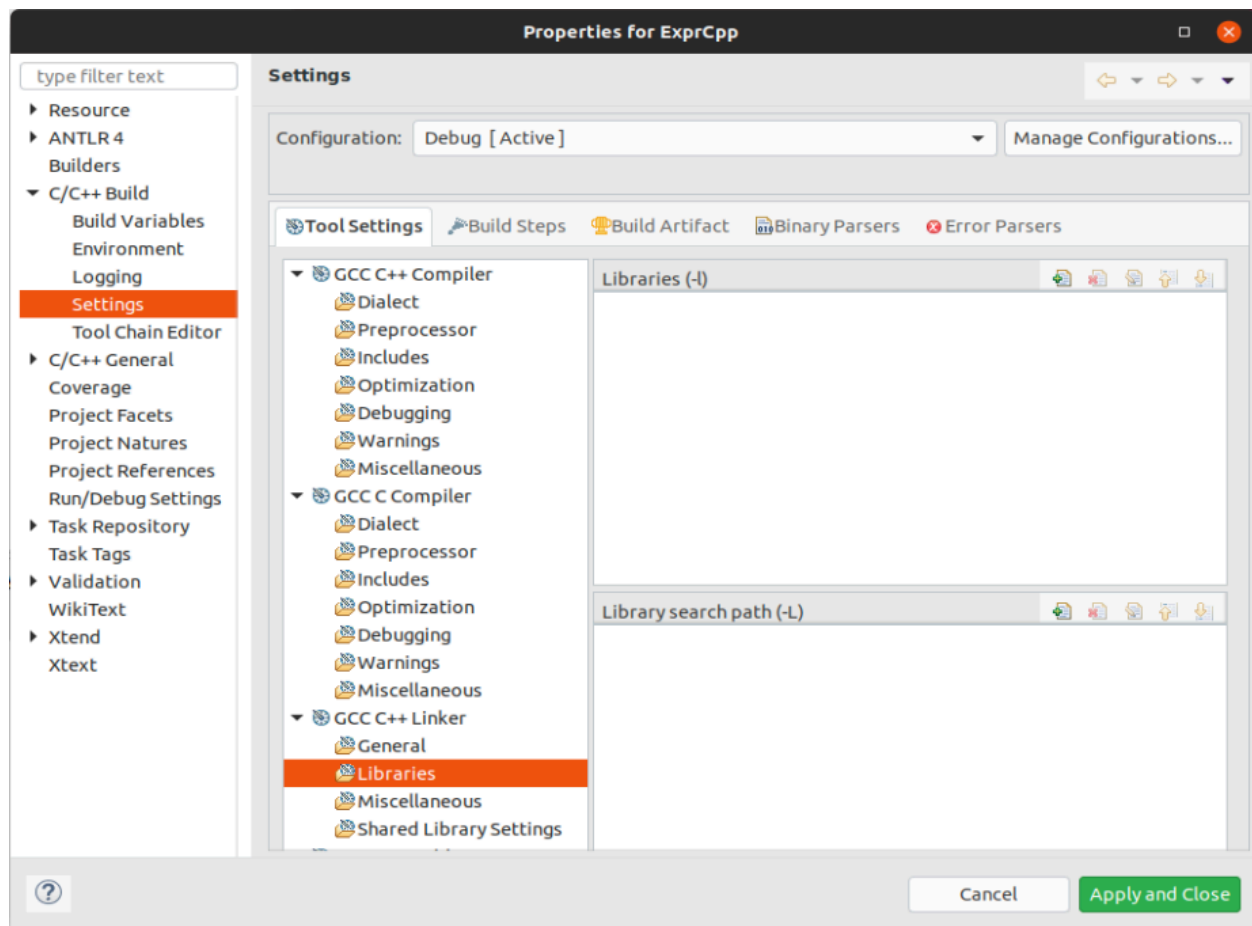


Figure 16. The dialog box to specify needed libraries.

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)

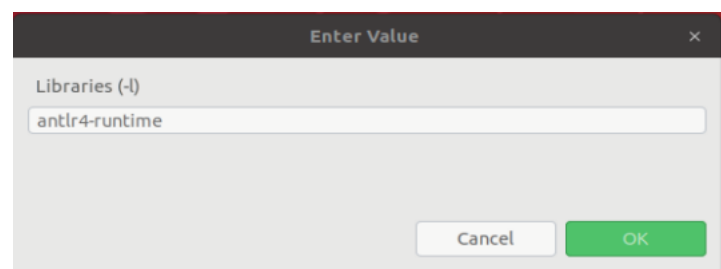


Figure 17: Specifying the antlr4-runtime library.

## R. Mak, Install and Configure ANTLR 4 for C++ on Eclipse and Ubuntu

Now the project can access needed libraries (Figure 18). Click the *Apply and Close* button.

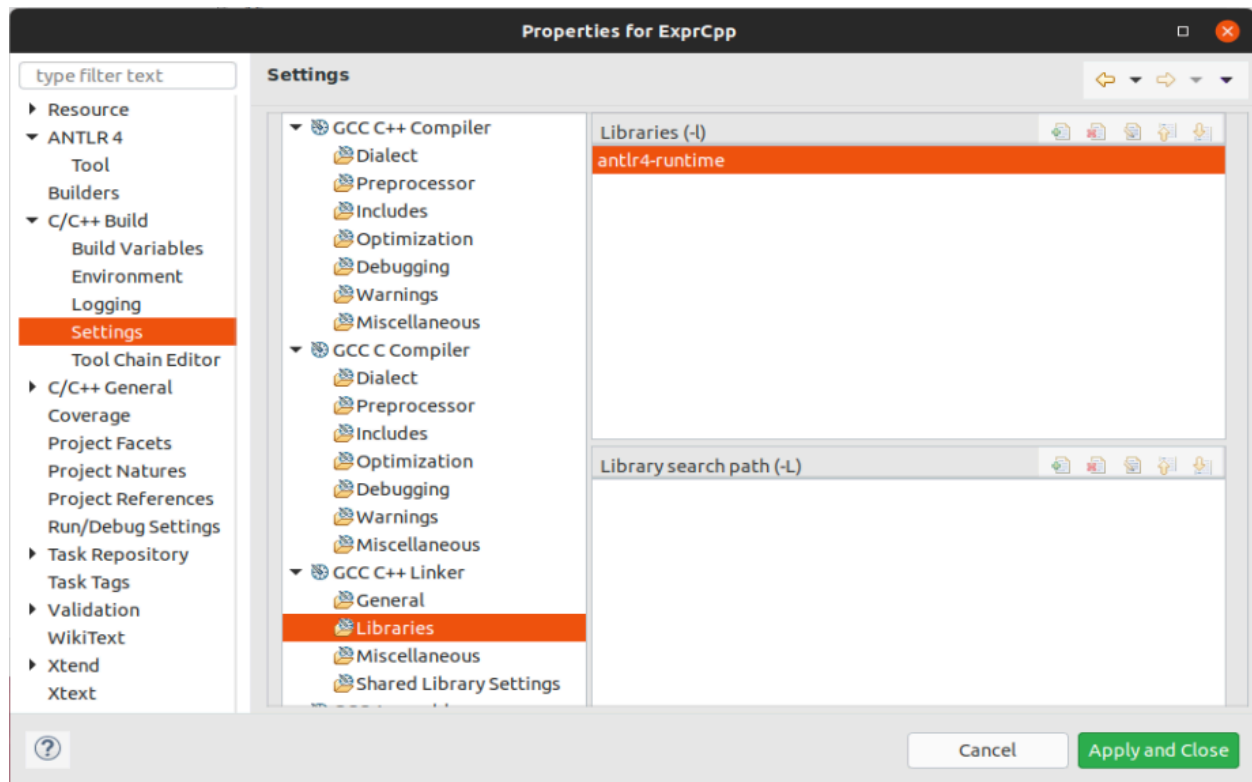


Figure 18. Libraries needed by the project.

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.

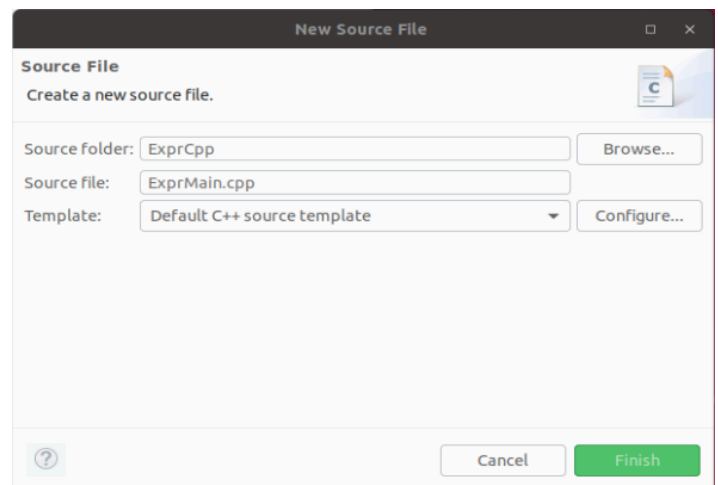


Figure 19. Creating the main source file.



Enter the following code for source file ExprMain.cpp (Figure 20):

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

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

using namespace antlr4;
using namespace antlr3;
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;
}
```

Figure 20. The main file ExprMain.cpp.

Select the project name and click the hammer icon at the top to compile and link the project (Figure 21).

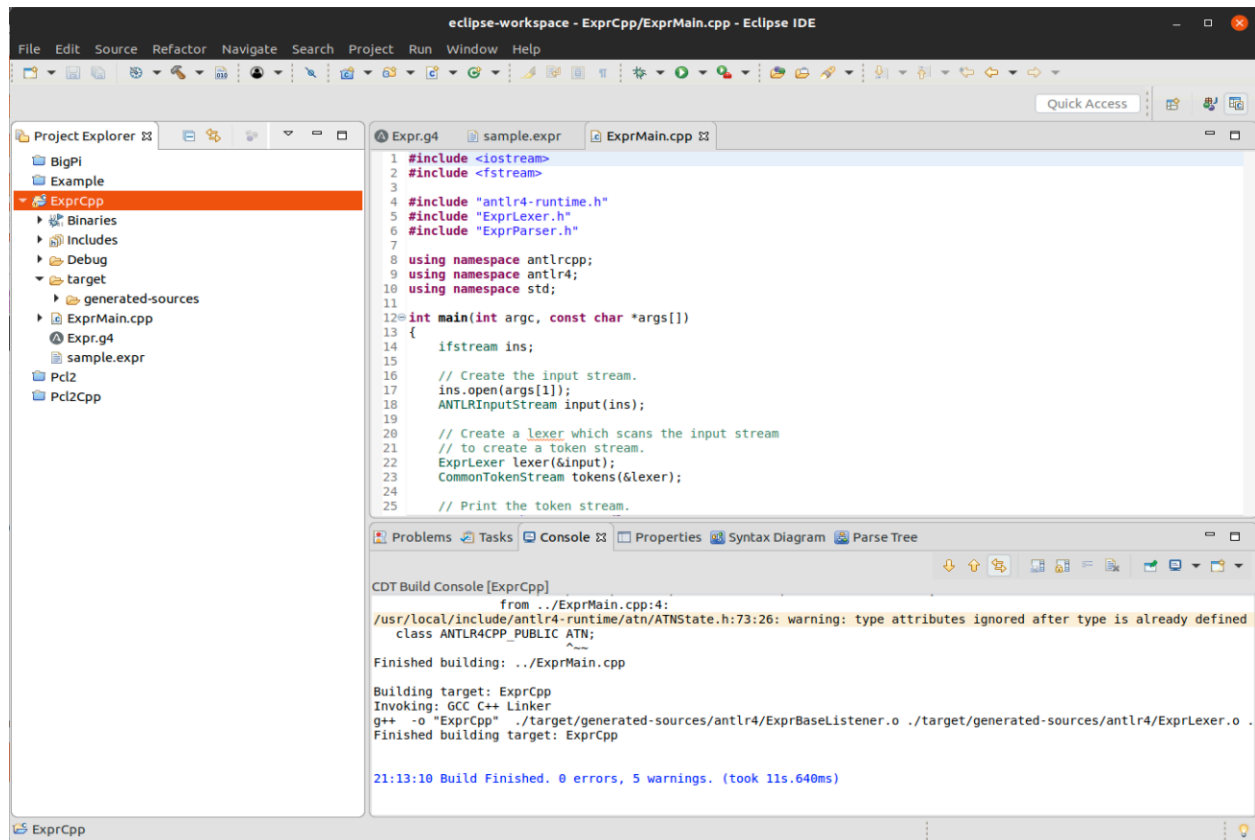


Figure 21. Compile and link.

Before we can run the program, we 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).

In the right panel, type “ExprCpp” for the configuration name (Figure 22). If the *Project* field is empty, click the first *Browse ...* button to select the project. If the *C/C++ Application* field is empty, click the *Search Project ...* button.

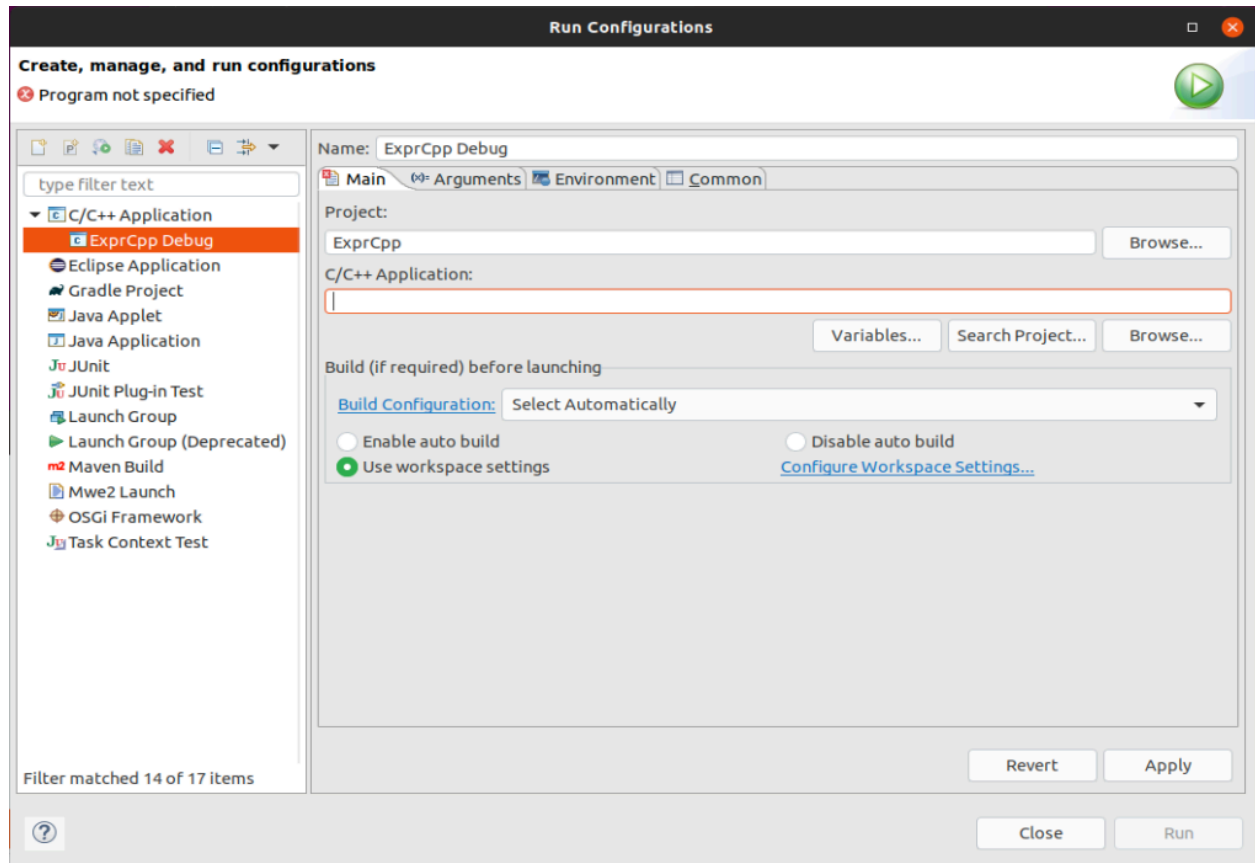


Figure 22. Creating a run configuration.

In the **Program Selection** dialog box, select ExprCpp and click the *OK* button (Figure 23).

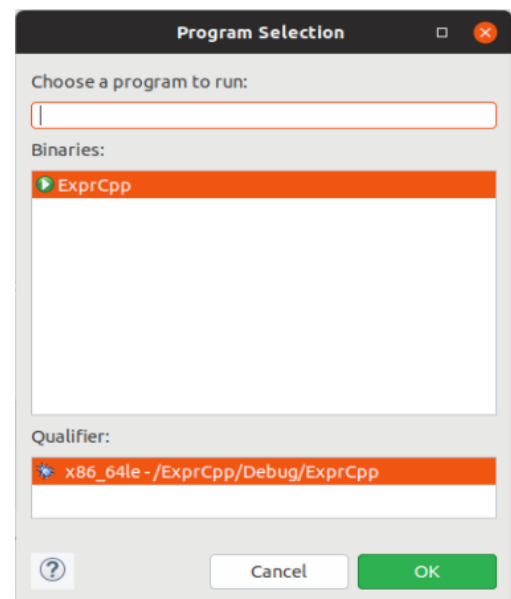


Figure 23. Selecting the program to run.

Back in the **Run Configurations** dialog box, click the *Arguments* tab and type “sample.expr” as the program’s command-line argument, the source file to compile. (Figure 24).

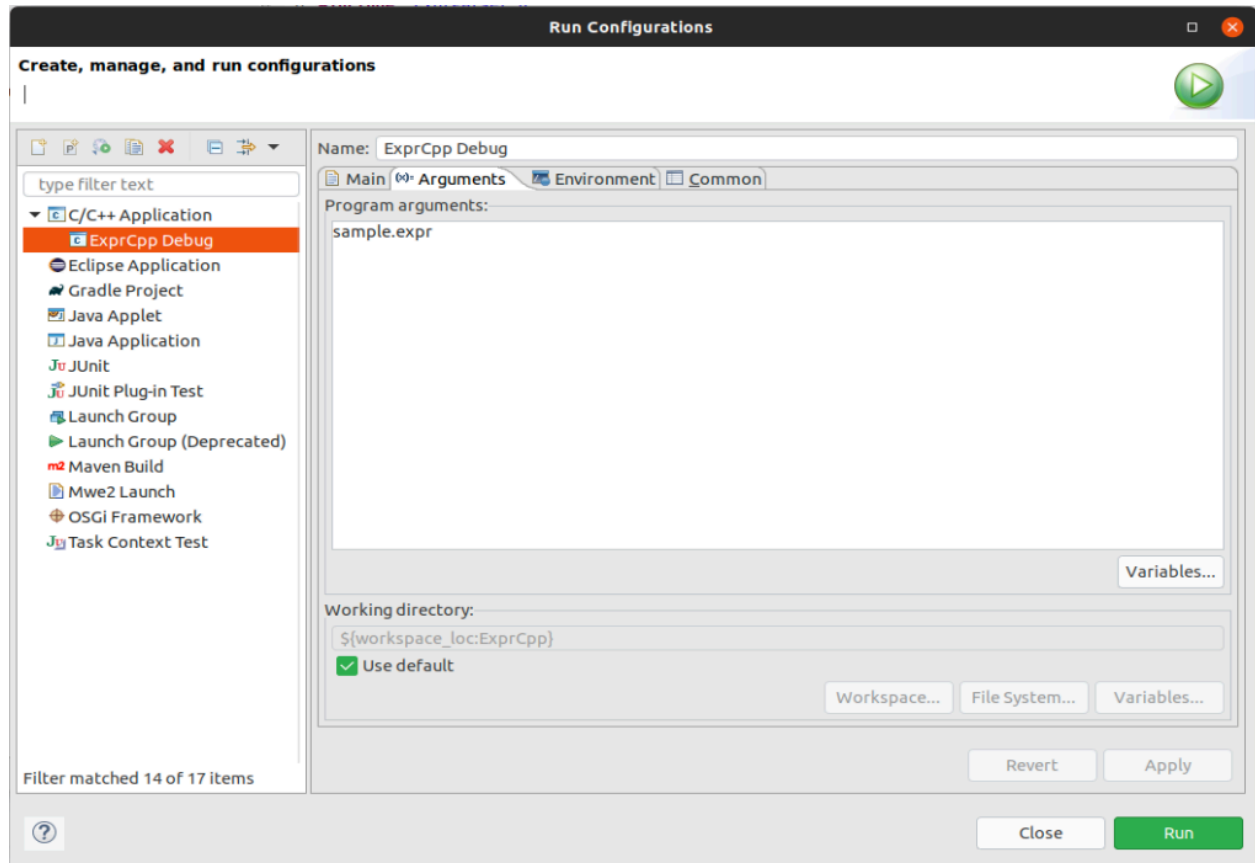


Figure 24. Specifying the command-line argument sample.expr, the source file to compile.

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 25).

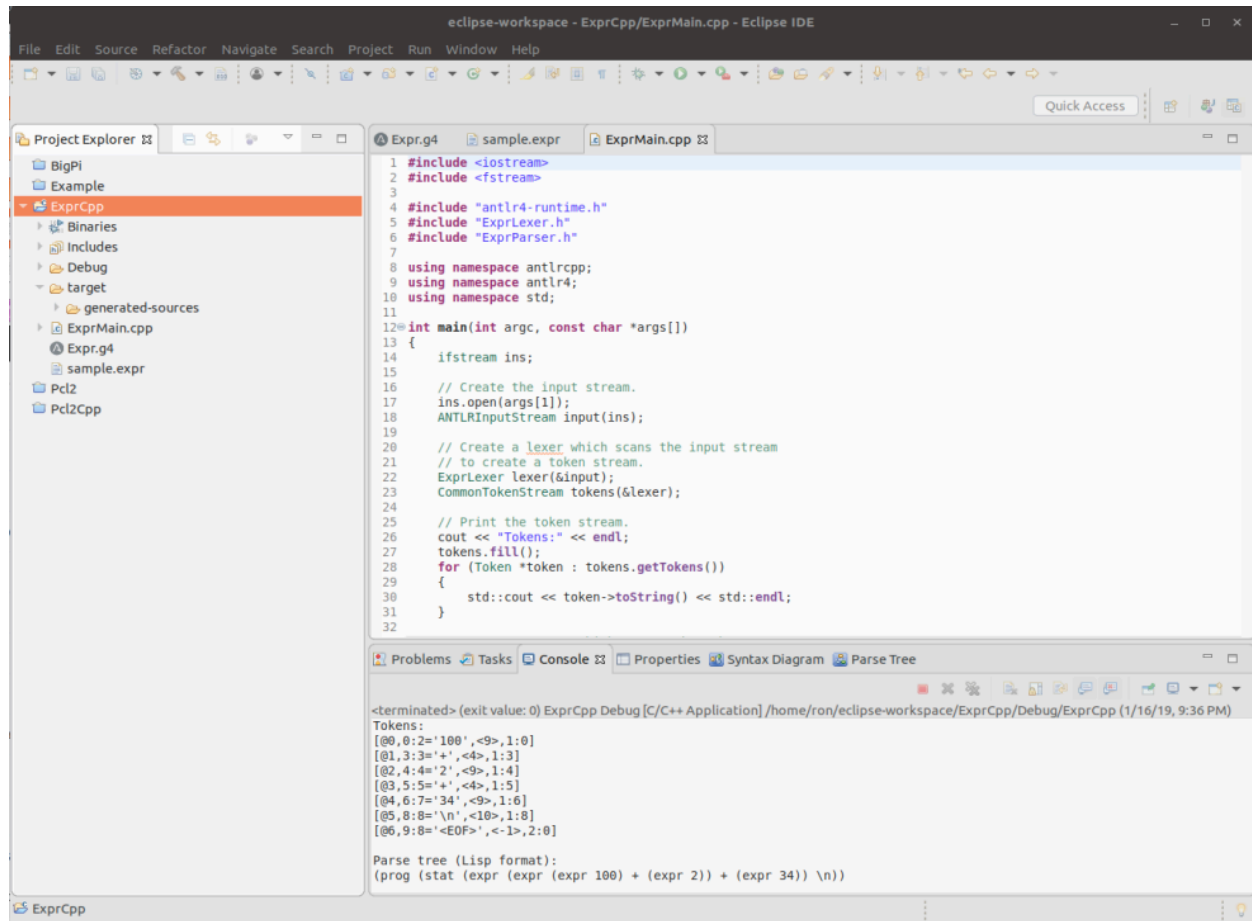


Figure 25. Output from running the generated compiler.