

EECS 498 W26 System Design of a Search Engine

13 Expression parser

Due Sunday, March 29, 2026, 11:59 pm

This homework is designed to teach you how to create a grammar and parser. You will write a program that takes in a math expression and evaluates it.

You are given some starter code that implements certain functions, but you will need to write the rest to have a functional parser that can create and evaluate a parse tree.

Here is a preview of what the finished homework will do. The driver does not take any command line arguments but will read and evaluate a math expression from stdin.

```
$ ./driver
(((((((8))))))) * 1) + (((9 + 2 * (4 + 4) + 2 * 4 + 3)) * 2 )
80
```

If the expression is invalid, it should print "Syntax error".

```
$ ./driver
3 +
Syntax error
```

Setup

First download and unzip the starter files. The files should compile as-is.

```
$ ls
Makefile  expression.cpp  parser.cpp  tokenstream.cpp
driver.cpp expression.h  parser.h    tokenstream.h
$ make
g++ -std=c++17 -g -c parser.cpp -o parser.o
g++ -std=c++17 -g -c tokenstream.cpp -o tokenstream.o
g++ -std=c++17 -g -c expression.cpp -o expression.o
g++ -std=c++17 -g parser.o tokenstream.o expression.o driver.cpp -o driver
$
```

driver.cpp

This is where you will run your program. This file is already implemented for you, but you are welcome to modify it if you wish.

The main function will take in an input, tokenize it, create an expression tree from the tokens, and then evaluate and print the answer.

tokenize.cpp

Tokenize will take an input string and separate the string into operators and digits. This file is already implemented for you, but you are welcome to modify it if you wish.

You will want to use the member functions in your parser.cpp to find the next desired token/number, and to check if you are at the end of the input stream.

Expression.cpp

This is where you will want to create new expressions for your grammar. Start by implementing expressions for adding and multiplying. Later, you will build on that to add support for unary minus, division and subtraction.

Parser.cpp

This is where you will parse the tokens into expressions. The function definitions are already implemented for you in parser.h. Your job is to implement parser.cpp and decide how to read in the token stream.

Start by implementing either of the following grammars, whichever seems more natural to you. An `<integer>` is a conventional string of decimal digits.

```
<AddSub>    ::=  <MulDiv> { '+' <MulDiv> }
<MulDiv>    ::=  <Factor> { '*' <Factor> }
<Factor>    ::=  '(' <AddSub> ')' | <integer>
```

----- OR -----

```
<AddSub>    ::=  <MulDiv> [ '+' <AddSub> ]
<MulDiv>    ::=  <Factor> [ '*' <MulDiv> ]
<Factor>    ::=  '(' <AddSub> ')' | <integer>
```

Extend the grammar by adding support for unary minus (negative numbers) and left-associative division ("`/`") and subtraction ("`-`") operators.

Submitting and Grading

Files to submit:

1. driver.cpp
2. expression.cpp
3. expression.h
4. parser.cpp
5. parser.h
6. tokenstream.cpp
7. tokenstream.h

If you are failing some of the tests, consider if '--2' is legal. (It is.) Also, make sure you added '/' as a relevant character in your tokenizer.