1 Purpose

The purpose of this project is to expose any gaps in your Linux-based C++ development. This project provides an early opportunity to address shortcomings before before we approach the complex tasks involving the operating system. Additionally, we explore the reality of a world including generative AI. We will begin with an AI-generated solution and *refactor* it into the indicated build.

2 Task

Create a command line tool named `evaluate-boolean` that correctly evaluates infix notation boolean expressions involving boolean variables and the boolean operators `AND` and `OR`. Correct evaluation means that order of operation is observed. Note this is a simplified algebra without parenthesis `()` and `NEGATION`.

Application input should be accepted as a string argument followed by `n` characters representing the true `T` and false `F` values of the characters, e.g.,

```
./bin/evaluate-boolean "a + b * c" F T F
```

The order in which the variables appear is irrelevant to the order of their values, i.e., `a` always takes the first `T|F`, `b` always takes the second, `c` the third, etc.

3 Deliverables

```
proj1/
+-- Makefile                # Build configuration file for make program
|
+-- src/
|   +-- boolean_expression_parser.cc  # Source file for the parser logic
|   +-- main.cc                       # Main entry point of program
|   +-- util.cc                       # Source file for utility functions
|
+-- include/
|   +-- boolean_expression_parser.h   # Header file for the parser logic
|   +-- util.h                        # Header file for utility declarations
|
+-- bin/                    # Build directory (generated during build process)
|   +-- evaluate-boolean              # Main entry point of the program
|
+-- build/                  # Compile directory (generated during build process)
|   +-- boolean_expression_parser.o   # object for the parser logic
|   +-- main.o                        # object for program main entry point
|   +-- util.o                        # object for utility functions
|
+-- README.md               # Details contents of source, header, and other files
```

3.1 Submission

You must submit a zip archive and only a zip archive. Individual files and other archive types will not be accepted; 100% penalty.

All names are case-sensitive

Your archive must hold a topmost/root directory named `proj1` containing the following; 100% penalty.

Your project must adhere to basic C++ best practices. To that end, I am providing the following files and structure. Note these files are yours and, aside from Makefile, you should expect to create or modify them all.

- `proj1/README.md`
  A non-RTF text file, though you are free to use the standard markdown language of this file type if you choose. The file must describe the purpose of each included file (other than itself).
- `proj1/include/boolean_expression_parser.h`
  A C++ header file containing only the declaration of your BooleanExpressionParser class. Implementation details **MUST NOT** appear in this file.
- `proj1/include/util.h`
  A C++ header file containing only the function or class declarations of your helper classes or functions, e.g., the function which maps letters to **true** or **false**. Implementation details **MUST NOT** appear in this file.
- `proj1/src/boolean_expression_parser.cc`
  A C++ source file containing only the implementation details of your BooleanExpressionParser class.
- `proj1/src/main.cc`
  A C++ source file containing only your application's entry point. The only code allowable in this file is your `main` function. Any helper functions used to process input must appear in the `util` lib. Functions other than `main` **MUST NOT** appear in this file.
- `proj1/src/util.cc`
  A C++ source file containing only the implementation details of your helper classes or functions, e.g., the function which maps letters to **true** or **false**.
- `Makefile`
  I have provided you with a `Makefile` that will build all code, assuming it is placed in the correct locations. You **MUST NOT** alter this file. I will use my own version for testing and your `Makefile` will be ignored.

All source code must be styled. Use the Google style guidelines for your code; 15% penalty. Assuming you have access to Python 3 and Pip 3, install cpplint for ease of checking your code's style. Code so poorly styled as to be unreadable will not be considered.

Cont.

3.2 Behavior

This task is **very well studied** and I assume many, if not most or all, will ask a generative AI for help. To that end, I have done so for you. It created the following, correct, **grammar**

Expression	→	Term {`"+"` Term}
Term	→	Factor {`"*"` Factor}
Factor	→	`"T"` | `"F"`

for a **top-down, recursive-descent parser**. You will find in this directory a file, `proj1.cc` which contains the AI's best attempt at implementing parser for the grammar. I went through multiple iterations helping it notice its own mistakes, but it missed three. None of these inhibit compilation and still allow the parser to correctly handle some expressions.

I encourage you to start working from the code generated by ChatGPT.

1. Start with the given file use

   ```
   g++ -std=c++17 proj1.cc -o proj1
   ```

   To compile the monolithic file and ensure that works in your environment. It should pass some but not all tests. Passing all tests only gets you the last 20%, so focus on setup first.

2. Next, break it into the components describe above and use the provided `Makefile` to get the code compiling on your machine as the five separate files:

   `proj1/include/boolean_expression_parser.h`,

   `proj1/include/util.h`,

   `proj1/src/boolean_expression_parser.cc`,

   `proj1/src/main.cc`, and

   `proj1/src/util.cc`.

3. Ensure your source and header files are correct as per the instructions and modern C++ coding standards[1].

4. Update your `README.md` to reflect the changes you have made.

5. Fix the logic three errors ChatGPT embedded in the program. All three result in yet-to-be-parsed characters remaining in the expression. Each causes the issue for a different reason. **Notice this is the least valuable step. Get your structure first; fix the code second.**

[1]In testing, we **will** make a `main.cc` file which includes your headers twice to test your define guards.

Cont.

3.3 Points

This section describes the points awarded for each correct project deliverable. There may be additional penalties assigned; see the Submission subsection for details.

- Correct `README.md`: 1 point. You must explain exactly what is contained in each file and your design decisions for its contents.
- Correct(compilable) entry file (`main.cc`): 1 point[2]. If you have anything more than a main function which calls other functions and merges their results, you will receive 0 points for this code, i.e., the code to map variables to their values must not be in this source file.
- Correct(compilable) boolean parser class library (`boolean_expression_parser.cc/h`): 1 point[3]. If you include any code in this file not necessary to the class, you will receive 0 points, e.g., the code to map variables to their values must not be in this class library.
- Correct(compilable) helper class/function library (`util.cc/h`): 1 point[4]. Your implementation code **MUST** be in the cc file and your declaration code **MUST** be in the h file. If you do otherwise, you will receive 0 points. This is where the mapping from character to boolean and any explode function should reside.
- Correct evaluation of described boolean expressions: 1 point[5]

Let it be stated one more time, you are begin given 95% of the code for this project; any deviation from the specified format will result in zero credit. That format is what is actually being graded. Make sure you do your own work, your code will be inspected for similarities and you may be shocked how easy it is to tell who copied from whom in a situation like this.

[2]Code that compiles, but contains warnings will only receive 0.85/1.0 points.
[3]Code that compiles, but contains warnings will only receive 0.85/1.0 points.
[4]Code that compiles, but contains warnings will only receive 0.85/1.0 points.
[5]There is no partial credit here.

The End.