Compilers -2(CS3423)
Project - Overview

- Team 19
S Abhinay (CS21BTECH11055)
C Rutheesh (CS21BTECH11014)
G Harsha (CS21BTECH11017)
K Karthik (CS21BTECH11030)

Problem Aiming to Solve:

Everyday we deal with data, in a lot of fields we need to manipulate data. There is no special language designed to read and manipulate data that is in the form of csv files. Though there are libraries in normal programming languages to handle csv and other files in similar format there is no special language which has a built-in module to handle these types of files and is structured around manipulating data easily. So our language aims to be a DSL that helps working with csv files.

Why is the problem important to Solve:

These days there is a lot of data available and almost all fields are beginning to use some AI techniques. These techniques need data to be trained but the data need to be pre processed before fetching into these models. So a DSL helping these tasks is important. Also we may want to extract the data present in the files and want to perform some actions, for example the data present in the file may be matrix and we may want to find the inverse of it, our language has support for matrices and helps that and other similar operations on data.

Difficulties Faced in the Implementation:

We have class construct in our language, but when writing semantics we first wrote without considering the existence of classes, thinking that we can add these basic checks first and then on top it the class relevant semantic checks. But, later we realized that the existence of classes has just increased the complexity of semantic analysis by at least five times. Each grammar rule has to be changed. Also, we had so many semantic checks, some of them we totally forgot and found that we had not implemented them when we were testing some test cases. The experience of implementing and testing our semantic analysis was like digging an inexhaustible gold mine where we kept finding errors each time we tested.We wanted our language to be very flexible and high level but as the target language was C++ and due to limited time, we had to curtail some features in our language because either required constructs were not present in C++ or were difficult to implement in the given time.

Pipeline of Compilation Process:

The input program passes through the Lexer, which tokenises the input and sends them to the parser. The parser checks whether the input program satisfies the language rules. The semantic analysis is written as SATG grammar in the action parts of grammar rules. So semantic analysis happens along with the parsing. The output is in the target language C++.

Running Test Cases:

Compile the the parser file located in the codes folder using the following command

```
bison -d parse.y
```

It outputs parse.tab.c, rename the file using the following command

```
mv parse.tab.c parse.tab.cpp
```

Compile the lex file located in the codes folder using the following command

```
flex  lex.l
```

It outputs lex.yy.c, rename the file using the following command

```
mv lex.yy.c lex.yy.cpp
```

Compile the lex and parser outputs to generate the compiler by running the below command

```
g++ lex.yy.cpp parse.tab.cpp helper_functions.cpp -o
compiler
```

Or run the 1.sh script file included in the codes folder which does the above things in one go

```
Chmod +x 1.sh
./1.sh
```

The test file should be in the same directory as compiler binary i.e. in the codes folder. Let's say the name of the test file is inp.txt, it is given as command line argument like below

```
./compiler inp.txt
```

Now the generated code in C++ will be found in the same directory that is codes directory and the name of the output file would be out_inp.cpp. Executing this file is equivalent to executing the input file DSL file, run the following command to do this

```
g++  out_inp.cpp
./a.out
```