System Programming Semester Assignment

Submitted by - Ekamjit Singh

Roll number - 2019CSC1001

Project uploaded on:

https://github.com/ekamjit105/System-Programming-Assignment

Project description

Consider a simple SQL query that retrieves a character string from a column of the Table "Formulae" as below:

```
Select "Formula" from "Formulae" where "ID"=5;
```

The returned value may be like:

```
'(<A1>+<A2>)*0.5 - <A3>'
```

where <A1>, <A2>, <A3> etc. Are to be substituted by the fieldnames stored in a Table "FormulaFields". An example schema can be:

```
("TupleID", "VariableName","ColumnName")
(1,'<A1>','Salary.Basic')
(2,'<A2>','Salary.TA')
(3,'<A3>','Salary.PF')
...
```

(i.e, tablename.columnname is stored in the third column of the table)

Having obtained the formula from the table by a query such as above, you need to convert it to a corresponding SQL query. For the example above, the resulting query should be

```
Select ((Basic + TA) * 0.5 - PF) as "Result" from "Salary"
where ... (some condition here)
```

To do:

1. Use *lex* (*flex*) to scan and interpret formula that is always constructed from symbols of the form *<A1>*, *<A2>* etc., and numeric literals, and arithmetic operators +, −, *, / and () [2 Marks]
2. Use *yacc* (*bison*) to generate parser for checking if formula is correctly formed according to standard grammer of arithmetic expressions involving tokens from 1 above [3 Marks]
3. Generate code equivalent to the example SQL code given above by specifiying appropriate instructions in parser using C/C++. [2 Marks]
4. Make the tables "Formulae" and "Salary" in some DBMS, say MySQL, and test run the successful execution of the generated query. [2 Marks]
5. Include documentation for your design, standard comments in source code, and any special assumptions you make. [1 Mark]

Assumptions made and approach followed

- According to the project description, we have created a database called mydb.db in **SQLite3 DBMS**. The database contains three tables with schema as described below.

```
C:\Users\ekamjit singh\Desktop\assgn\assgn\2019CSC1001_SemAssignment>sqlite3 mydb.db
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
sqlite> .tables
formulae       formulafields  salary
sqlite> .schema formulae
CREATE TABLE formulae (id INT primary key, formula varchar);
sqlite> .schema formulafields
CREATE TABLE formulafields (TupleId INT primary key, VariableName varchar, ColumnName varchar);
sqlite> .schema salary
CREATE TABLE salary (id INT primary key, basic FLOAT, TA FLOAT, PF FLOAT, ENT FLOAT, rate FLOAT);
sqlite>
```

I. Formulae
   - Contains various formulas used to calculate resultant salary, each having a unique id.

II. Formulafields
   - Used to map all the variables (like <A1>, <A2> etc) to actual fields required to calculate salary (Basic, Rate, TA etc).
   - We have assumed that the variables start from <A1>, <A2>… hence considering <A0> as an invalid variable. The same has been taken care of while tokenizing in Flex.

III. Salary
   - We have assumed that this table contains various salary descriptions, with each row (identified by a unique id) defining various field values (Basic, rate, TA etc) for calculating a particular resultant salary.

- Inserting Values into the tables Formulae, Formulafields and Salary

I. Formulae

```
sqlite> .header on
sqlite> .mode column
sqlite> select * from formulae;
id  formula
--  --------------------
1   <A1>+2*(<A2>-<A3>)
2   (<A1>*<A4>)-<A2>
3   3.5*<A4>+<A2>
4   (<A1>/<A2>)*4
5   <A4>+<A1>*(3+2)
6   <A1>-(<A2>*3
7   <A1>-<A>*3+4
8   <A3>+(4*<A0>)
9   (<A1>+<A2>)*0.5-<A3>
sqlite>
```

**Here tuple ids 6-8 have been deliberately inserted with incorrect formulas to test erroneous cases. Tuple id 9 contains the formula as in problem statement example.

II. Formula fields

```
sqlite> Select * from formulafields;
TupleId  VariableName  ColumnName
-------  ------------  ------------
1        <A1>          salary.basic
2        <A2>          salary.TA
3        <A3>          salary.PF
4        <A4>          salary.ENT
5        <A5>          salary.rate
```

**Assuming variables start from <A1>, <A2> and so on.

III. Salary

```
sqlite> Select * from salary;
id  basic    TA      PF    ENT     rate
--  -------  ------  ----  ------  ----
1   20000.0  3400.0  5.34  2500.0  24.0
2   30000.0  4300.0  6.34  1500.0  21.0
3   35900.0  5420.0  5.0   2500.0  24.4
4   54000.0  4300.0  2.5   1500.0  21.0
5   22000.0  3400.0  4.5   1300.0  17.2
sqlite>
```

Approach followed:

I. User is asked to enter an id to select a particular formula from the formula table.
II. The id entered fetches a formula from the database and the formula is scanned using Flex (file with .l extension).
III. After flex tokenizes the formula, its correctness is checked according to standard grammar of arithmetic using Bison (file with .y extension).
IV. Once this generated parser successfully parses the formula, all the variables in the formula <A1>, <A2>, <A3> etc. are substituted with field names using the formula fields table.
V. A query is then generated using this final generated formula that calculates salary for a particular salary id (taken as input from the user). Hence the final query becomes

Select <Formula> AS RESULT FROM Salary where id = <user entered id>;

We can use other conditions as well in the where clause of this query like '…where basic > 20000 ' , '…where rate < 9.5' etc.

Hence the final query is executed and result is printed.

Output Screenshots

The following commands are used to compile and run the code.

```
\2019CSC1001_SemAssignment> flex assgn.l
\2019CSC1001_SemAssignment> bison -d assgn.y

\2019CSC1001_SemAssignment> g++ -o output lex.yy.c assgn.tab.c sqlite3.o
\2019CSC1001_SemAssignment> .\output.exe
```

The command 'Flex assgn.l' is used to compile the .l file as a Flex file and command 'Bison -d assgn.y' is used to compile the .y file as a bison file. These generate intermediate and header files to be further compiled together for creating an executable file.

The intermediate files generated (lex.yy.c and assgn.tab.c) are then compiled together with sqlite3 object and header files required for database connectivity using **g++ compiler**.

Output.exe is the final generated executable file for the project.

Running the output.exe file

We first display all the tables created in the database

```
------------------------------------------------
       DISPLAYING ALL THE TABLES IN DATABASE
------------------------------------------------

TABLE FORMULAE

id: 1
formula: <A1>+2*(<A2>-<A3>)

id: 2
formula: (<A1>*<A4>)-<A2>

id: 3
formula: 3.5*<A4>+<A2>

id: 4
formula: (<A1>/<A2>)*4

id: 5
formula: <A4>+<A1>*(3+2)

id: 6
formula: <A1>-(<A2>*3
```

```
TABLE FORMULAFIELDS

TupleId: 1
VariableName: <A1>
ColumnName: salary.basic

TupleId: 2
VariableName: <A2>
ColumnName: salary.TA

TupleId: 3
VariableName: <A3>
ColumnName: salary.PF

TupleId: 4
VariableName: <A4>
ColumnName: salary.ENT

TupleId: 5
VariableName: <A5>
ColumnName: salary.rate
```

```
TABLE SALARY

id: 1
basic: 20000.0
TA: 3400.0
PF: 5.34
ENT: 2500.0
rate: 24.0

id: 2
basic: 30000.0
TA: 4300.0
PF: 6.34
ENT: 1500.0
rate: 21.0

id: 3
basic: 35900.0
TA: 5420.0
PF: 5.0
ENT: 2500.0
rate: 24.4

id: 4
basic: 54000.0
TA: 4300.0
PF: 2.5
ENT: 1500.0
rate: 21.0
```

Then we ask the user to input a formula id to fetch a formula. Once parsed, we replace its variables and ask the user for a salary id for which the final result is generated. Following are the output screenshots of some of the test cases.

Case 1: Formula id = 3 {3.5*<A4>+<A2>}

Salary id = 1 {TA= 3400.0, ENT= 2500.0}

```
--------------------------------------------------

Enter formula id : 3

Executed query : SELECT formula FROM 'formulae' where id = 3;

fetched : 3.5*<A4>+<A2>
PARSE SUCCESSFUL!

--------------------------------------------------

Now moving to replacing variables

New formula with replaced variables is : 3.5*salary.ENT+salary.TA
--------------------------------------------------

Enter salary id to be selected: 1


Final Query will be :

SELECT (3.5*salary.ENT+salary.TA) AS RESULT FROM salary WHERE id=1

RESULT: 12150.0
```

Case 2: Formula id = 3 {3.5*<A4>+<A2>}

Salary id = 2 {TA= 4300.0, ENT= 1500.0}

```
--------------------------------------------------

Enter formula id : 3

Executed query : SELECT formula FROM 'formulae' where id = 3;

fetched : 3.5*<A4>+<A2>
PARSE SUCCESSFUL!

--------------------------------------------------

Now moving to replacing variables

New formula with replaced variables is : 3.5*salary.ENT+salary.TA
--------------------------------------------------

Enter salary id to be selected: 2


Final Query will be :

SELECT (3.5*salary.ENT+salary.TA) AS RESULT FROM salary WHERE id=2

RESULT: 9550.0
```

Case 3: Formula id = 2 {(<A1>*<A4>)-<A2>}

Salary id = 1 {Basic= 20000, TA= 3400.0, ENT= 2500.0}

```
--------------------------------------------------

Enter formula id : 2

Executed query : SELECT formula FROM 'formulae' where id = 2;

fetched : (<A1>*<A4>)-<A2>
PARSE SUCCESSFUL!

--------------------------------------------------

Now moving to replacing variables

New formula with replaced variables is : (salary.basic*salary.ENT)-salary.TA
--------------------------------------------------

Enter salary id to be selected: 1


Final Query will be :

SELECT ((salary.basic*salary.ENT)-salary.TA) AS RESULT FROM salary WHERE id=1

RESULT: 49996600.0
```

Case 4: Formula id = 2 {(<A1>+<A2>)*0.5-<A3>}

As in example test case given in problem statement

Salary id = 1 {Basic = 20000.0, TA= 3400.0, PF = 5.34}

```
--------------------------------------------------

Enter formula id : 9

Executed query : SELECT formula FROM 'formulae' where id = 9;

fetched : (<A1>+<A2>)*0.5-<A3>
PARSE SUCCESSFUL!

--------------------------------------------------

Now moving to replacing variables

New formula with replaced variables is : (salary.basic+salary.TA)*0.5-salary.PF
--------------------------------------------------

Enter salary id to be selected: 1


Final Query will be :

SELECT ((salary.basic+salary.TA)*0.5-salary.PF) AS RESULT FROM salary WHERE id=1

RESULT: 11694.66
```

References used:

- Compilers: Principles, techniques and tools : https://bit.ly/3mt9l4X
- https://www.geeksforgeeks.org/sql-using-c-c-and-sqlite/
- https://www.devdungeon.com/content/compiling-sqlite3-c

All the necessary project files have been uploaded on GitHub link given below

https://github.com/ekamjit105/System-Programming-Assignment

The repository includes the following files:

- assgn.l - lex file
- assgn.y – bison file
- output.exe – project executable file
- mydb.db – database created using SQLite3 DBMS

- assgn.tab.c and assgn.tab.h – files generated on compiling bison file
- lex.yy.c – file generating after compiling flex file

- sqlite3.c – used for compiling sqlite3
- sqlite3.o and sqlite.h – object and header files to be included in project for database connectivity.

Sincere regards.