DATUM

LEXER DESIGN

TEAM MEMBERS

- **K. S. Ananth**
- **Nimai Parsa**
- **M. Bhavesh Chowdary**
- **Arugonda Srikar**
- **G. S. S. Koushik**
- **M. Kartikeya**
- **Mohammed Gufran Ali**

SMALL INTRO TO DATUM

SMALL INTRO TO DATUM

- In lexical phase, we extracts tokens from the input program. We pass these tokens for next phase
- We wrote the rules in .l file and generated lexical analyzer (executable) using flex tool.
- The whitespaces and comments gets ignored.
- Open comments, illegal characters gets detects as errors and prints it to console.

CODE SNIPPET

```
"$!"([^!]|(!+[^!$]))*!*"!$" ;
"$$".*\n        ;
"integer"   {count(); printf("INTEGER");}
"float"     {count(); printf("FLOAT");}
"string"    {count(); printf("STRING");}
"char"      {count(); printf("CHAR");}
"bool"      {count(); printf("BOOL");}
"dataset"   {count(); printf("DATASET");}
"array"     {count(); printf("ARRAY");}
"if"        {count(); printf("IF");}
"else"      {count(); printf("ELSE");}
"loop"      {count(); printf("LOOP");}
"break"     {count(); printf("BREAK");}
"continue"  {count(); printf("CONTINUE");}
"return"    {count(); printf("RETURN");}
"function"  {count(); printf("FUNCTION");}
"or"        {count(); printf("OR");}
"and"       {count(); printf("AND");}
"not"       {count(); printf("NOT");}
"from"      {count(); printf("FROM");}
"to"        {count(); printf("TO");}
"step"      {count(); printf("STEP");}
"also"      {count(); printf("ALSO");}
"row"       {count(); printf("ROW");}
"col"       {count(); printf("COL");}
"filter"    {count(); printf("FILTER");}
"sum"       {count(); printf("SUM");}
```

```
"max"       {count(); printf("MAX");}
"min"       {count(); printf("MIN");}
"mean"      {count(); printf("MEAN");}
"join"      {count(); printf("JOIN");}
"read"      {count(); printf("READ");}
"write"     {count(); printf("WRITE");}
"unique"    {count(); printf("UNIQUE");}
"show"      {count(); printf("SHOW");}
"split"     {count(); printf("SPLIT");}
"sort"      {count(); printf("SORT");}
"shuffle"   {count(); printf("SHUFFLE");}
"add"       {count(); printf("ADD");}
"shape"     {count(); printf("SHAPE");}
"drop"      {count(); printf("DROP");}
"show_bar"      {count();printf("SHOW_BAR");}
"show_scatter"  {count();printf("SHOW_SCATTER");}
"show_line"     {count();printf("SHOW_LINE");}
"show_box"      {count(); printf("SHOW_BOX");}
"function_declarations:"    {count(); printf("FUNC_LABEL");}
"start:"                    {count(); printf("START_LABEL");}
{L}({L}|{D})*           {count(); printf("IDENTIFIER");}
\"(\\.|[^\\"])*\"       {count(); printf("STRING_LITERAL");}
'(\\.|[^\\'])'          {count(); printf("CONSTANT");}
{D}+                    {count(); printf("CONSTANT");}
{D}*"."{D}+             {count(); printf("CONSTANT");}
{D}+"."{D}*             {count(); printf("CONSTANT");}
"+="        {count(); printf("ADD_ASSIGN");}
"-="        {count(); printf("SUB_ASSIGN");}
"*="        {count(); printf("MUL_ASSIGN");}
```

```
"*="        {count(); printf("MUL_ASSIGN");}
"/="        {count(); printf("DIV_ASSIGN");}
"%="        {count(); printf("MOD_ASSIGN");}
"=="        {count(); printf("EQ_OP");}
"="         {count(); printf("=");}
"+"         {count(); printf("+");}
"-"         {count(); printf("-");}
"/"         {count(); printf("/");}
"*"         {count(); printf("*");}
"%"         {count(); printf("%%");}
"++"        {count(); printf("INC_OP");}
"--"        {count(); printf("DEC_OP");}
">"         {count(); printf(">");}
"<"         {count(); printf("<");}
"≥"         {count(); printf("GE_OP");}
"≤"         {count(); printf("LE_OP");}
"≠"         {count(); printf("NE_OP");}
"→"         {count(); printf("FLOW");}
";"         {count(); printf(";");}
"("         {count(); printf("(");}
")"         {count(); printf(")");}
","         {count(); printf(",");}
"{"         {count(); printf("{");}
"}"         {count(); printf("}");}
"["         {count(); printf("[");}
"]"         {count(); printf("]");}
```

SAMPLE CODE

```
function_declarations:

function (dataset ds) -> normalize() -> dataset {
    array(int) mean_arr = ds->mean(1);
    loop i from 0 to ds->shape[0] {
        ds->col(i) -= mean_arr[i];
    }
    array(int) std_arr = ds*ds->sum(1) / ds-
>shape[0];
    loop i from 0 to ds->shape[0] {
        ds->col(i) /= std_arr[i];
    }
    return ds;
}

start:

dataset ds = read("./dataset", "csv")
ds->col(1)
  ->fill_null_values(0)
  ->row({
        return r[0] == null;
    })->drop(0);

ds->normalize()->write("./dataset.csv", "csv");
```

SAMPLE OUTPUT

```
FUNC_LABEL

FUNCTION  ( DATASET  IDENTIFIER )  FLOW  IDENTIFIER ( )  FLOW  DATASET  {
     ARRAY ( IDENTIFIER )  IDENTIFIER  =  IDENTIFIER FLOW MEAN ( CONSTANT ) ;
     LOOP  IDENTIFIER  FROM  CONSTANT  TO  IDENTIFIER FLOW SHAPE [ CONSTANT ]  {
         IDENTIFIER FLOW COL ( IDENTIFIER )  SUB_ASSIGN  IDENTIFIER [ IDENTIFIER ]
;
     }
     ARRAY ( IDENTIFIER )  IDENTIFIER  =  IDENTIFIER * IDENTIFIER FLOW SUM (
CONSTANT )  /  IDENTIFIER FLOW SHAPE [ CONSTANT ] ;
     LOOP  IDENTIFIER  FROM  CONSTANT  TO  IDENTIFIER FLOW SHAPE [ CONSTANT ]  {
         IDENTIFIER FLOW COL ( IDENTIFIER )  DIV_ASSIGN  IDENTIFIER [ IDENTIFIER ]
;
     }
     RETURN  IDENTIFIER ;
}

START_LABEL

DATASET  IDENTIFIER  =  READ ( STRING_LITERAL ,  STRING_LITERAL )
IDENTIFIER FLOW COL ( CONSTANT )
   FLOW IDENTIFIER ( CONSTANT )
   FLOW ROW ( {
         RETURN  IDENTIFIER [ CONSTANT ]  EQ_OP  IDENTIFIER ;
     } ) FLOW DROP ( CONSTANT ) ;

IDENTIFIER FLOW IDENTIFIER ( ) FLOW WRITE ( STRING_LITERAL ,  STRING_LITERAL ) ;
```

DEMO FOR LEXER

https://drive.google.com/file/d/1GwMW1hHCxU5I6jFpPWGmZTH70Ek3j6te/view?usp=drive_link

CODE LINK

Datum/lexer/lexer.l at main · ksananth4424/Datum (github.com)

REFERENCES USED

ANSI C Lex: ANSI C grammar (Lex) (liu.se)