DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING IIT HYDERABAD

Compilers-II Project

Red Panda

Group-9

Chittepu Rutheesh Reddy
(CS21BTECH11014)

Kotikalapudi Karthik
(CS21BTECH11030)

Sadineni Abhinay
(CS21BTECH11055)

G Harsha Vardhan Reddy
(CS21BTECH11017)

Contents

1 Introduction

This language aims to simplify working with data that is in delimiter separated format. This language provides a datatype named dataframe that enables us to perform many useful actions on the data like replacing all null values in a column with a value, dropping some columns, manipulating some values in rows, searching rows in the data that satisfy certain conditions, updating and deleting rows based on some conditions, extracting the data into other datatypes of the language and processing it, etc. To assist processing data, non primitive datatype matrix and it's associated operations were added to this language. In addition to these all typical programming constructs are present in this language.

The goal of this language is to combine the features of general purpose languages with some features of high level languages in way that facilitates working with data. Suitable high level datatypes have been added to achieve this.

2 Lexical Conventions

2.1 Comments and Whitespaces

Single line comments begin with #. Multi-line comments are enclosed between #* and *#.

```
# I am a single line comment
#* I am a multi
line comment *#
```

Whitespaces, tabs and newlines are ignored in this language, except in places where they are necessary to separate tokens. Hence the following two programs are equivalent.

Ex-1

```
int a, b, c;
a = 2;
```

Ex-2

```
int a,b,c;a=2;
```

2.2 Reserved keywords

The following are the reserve keywords and they cannot be used as regular identifiers.

int	float	bool	char	string
void	dataframe	if	elif	else
matrix	read	sort	for	while
AND	OR	return	break	continue

Table 1: Reserved Keywords

2.3 Identifiers

This language uses the C-type identifiers i.e. accepts identifiers which start with underscore or a letter and rest of the characters are alphanumeric or underscore.
The following are examples of valid identifiers in this language.

```
_12
__
a23_
x_dim
```

Identifiers like 23gt,1var,2var,234 are invalid.

2.4 Punctuation

The allowed punctuation symbols are:

1. **Semi-Colon:** The statements should end with a semi-colon(;).
   Ex

```
int a = 4;
y = a*b;
```

2. **Comma:** Comma(,) can be used for multiple declarations.
   Ex

```
int x,y,z;
```

3. **Quotation marks:** Strings and characters are enclosed in double-quotes and single-quotes respectively.
   Ex

```
string s = "ant";
char c = 'a';
```

Data Type	Description
int	32 bit signed integer
float	32-bit signed floating-point number
char	stores a single character
string	character sequence
bool	for storing values that are either 1 or 0

Table 2: Primitive data types

3 Data Types

3.1 Primitive

3.2 Non-Primitive

Data Type	Description
dataframe	Main datatype that defines this programming language
matrix	A highlevel data type that abstracts matrix operations

Table 3: Non-Primitive data types

4 Operators

The below table gives the list of operators in this language in the their order of precedence from highest to lowest.

Ex-1:

```
int x = 5+3; #x evaluates to 8
int x = y-z; #x evaluates to the value of (y-z)
float x = 4.2*8; #x evaluates to 33.6
float y = 7.02/3; #y evaluates to 2.34
int x = 5*(3-2); #x evaluates to 5
x++; #value of x increases by 1
x--; #value of x decreases by 1
int x = 7%3; #x evaluates to 1
int x = 5**3; #x evaluates to 125
int x = 5>>2; #x evaluates to 20
int x = 4>>1 #x evaluates to 2
bool x = 5 >= 3 #x evaluates to true
bool x = 3 != 3 #x evaluates to false
```

Ex-2:

```
string s1 = "Hello ";
```

Usage	Symbol	Associativity	Valid Operands
To prioritise/separate evaluation of an expression	()	left to right	int,float,bool
To access members	.	left to right	dataframe
Negation operator	!	right to left	bools and boolean expressions
Increment	++	right to left	int,float
Decrement	–	right to left	int,float
left shift	<<	left to right	int
right shift	>>	left to right	int
power	**	left to right	int,float,matrix
Modulo	%	left to right	int
Multiplication	*	left to right	int,float,matrix
Division	/	left to right	int,float
Addition	+	left to right	int,float,matrix
Subtraction	-	left to right	int,float,matrix
Comparison Operators	<= >= ==	left to right	int,float,bool
Comparison Operators	< >	left to right	int,float,bool
Logical operators	AND OR	left to right	int,bools
Assignment Operators	= += *= /= -= %=	right to left	identifiers on LHS, expressions on RHS

Table 4: Operators

```
string s2 = "World";
string res = s1+s2; #res corresponds to "Hello World"
```

5 Statements

5.1 Declaration statements

Used for declaring variables, functions etc..
Examples:

```
    int x,y,z;
    bool arr[x];
    char x;
    string y;
    matrix m;
```

Declaration statements are further explained in section 6

5.2 Expression statements

Contains single expression followed by semicolon. Used for evaluating expressions. Expressions can include variables, constants, arithmetic operations, logical operations, predicates, function calls and valid combinations of these. It contains two parts

1. An id in LHS , and
2. An expression in RHS

Example

```
x = x + y;
H = x AND y;
k = "Car";
y = a ( c ) + x;
```

5.2.1 Arithmetic operations

It is of 2 types

1. Binary operation.
   It is a part of RHS. It consists of 3 parts
   (a) $operand_1$
   (b) arithmetic operator
   (c) $operand_2$
2. Unary operation
   It is a simple statement. It is of two types.
   (a) Increment operation (operand++)
   (b) decrement operation (operand- -)

Examples

```
x = y + z;
cai++;
```

Where operand can either be valid identifier/ constant or valid return value of a function call or valid output of another operation.

5.2.2 Logical operation

It is of two types.

1. Unary operations (⟨*operator*⟩ *operand*)
2. Binary operations (*operand* ⟨*operator*⟩ *operand*)

Examples

```
    a = b AND c;
    cailin = !qb;
```

5.3 Control Flow:

5.3.1 Conditional Statement:

Conditional statement contain if clauses , else if clauses and else clause .The syntax is as follows if-clause followed by else-if clauses followed by else clause. if and else if clauses contain predicate enclosed by parenthesis. All there clauses contain action that can be either in same line as the clause header or action can enclosed by braces.

```
if(x>>3 && y < 5 || z ==6 ){
  m2 = m4 + m6;
  m7 = m9 * m10;
}
elif(i++) w = z+y;
else{
 matrix m<int> = {{4 ,5 ,6},{2,3.5}};
}
```

5.3.2 Loop Statements:

Loops can introduced into the code by either using for or while .Both of their syntax is same as that in C.

1. For loop contains an expression or a declaration with initialization followed by predicate followed by an increment statement all these statements are separated by semicolon (;) the action is either in same line as the for header or enclosed by braces

```
matrix m <int> = W;
for( int i =0 ; i<= m[0][1] ;i++ ){
         m [i][i] = 0;
   }
for(int i =0 ; i<= m[0][1] ;i++ )m[i][i]=0;
```

2. while loop contains a predicate enclosed by brackets the action is either in same line as the for header or enclosed by braces.

```
while(m[0][1] < x) {
            x++;
    }
while(m[0][1] < x)   x++;
```

3. continue break statements can used inside the loop statements to change the direction the execution of the code.

```
while(m[0][1] < x){
if(m[1][1] == x)break;

elif(m[2][1] == x)continue;

}
```

5.3.3 Call statements

A call statement is a statement that invokes a function or a method. It is used to execute a specific block of code defined in the function or method. When a call statement is encountered during the program's execution, control transfers to the function or method being called, executes the code inside it, and then returns control to the calling statement after the function or method execution is complete.

Syntax

$$\langle fun\_name \rangle \, (arg\_list)$$

where fun_name is name of the caller function and arg_list is the list of arguments to be passed.

Example

```
R = Y ( C );
pain(nagato);
caier();
```

6 Declarations:

There are different types of declarations possible like function declaration, variable declaration etc...

6.1 Variable declaration:

Every variable used in the code should be declared before.

Syntax $\langle datatype \rangle \; \langle id \rangle \, ;$

And one can intitialize a value to the variable during declaration
Syntax ⟨*datatype*⟩ ⟨*id*⟩ = ⟨*const*⟩ ;
Examples

```
int x;
bool y = 0;
char z, a, b;
string st;
float f=0.16181;
matrix m;
```

And there are 2 special variables

1. **Static variables:** Static variables are created when the program starts and destroyed when the program terminates and they initialized only once and retains its value between function calls.

2. **Constant variables:** Const variables must be initialized when declared, either with a value or through a constructor, their value cannot be changed after initialization. It is used to create read-only variables.

6.2 Array declaration

One can declare an array by specifying its type, name, and size. Arrays are used to store a collection of elements of the same data type. Here's the basic syntax for declaring arrays:
⟨*datatype*⟩ ⟨*id*⟩ [*arr_size*] ;
where *arr_size* is size of the array. and similar to simple variables arrays can be initialised along with declaration. Here's the basic syntax for the same
⟨*datatype*⟩ ⟨*id*⟩ [ ] = $\{const_1, const_2, \ldots, const_n\}$;
where $n \geq 0$
Examples

```
int arr[5];
```

```
int arr[] = {1,2,3};
```

```
int n;
int arr[n];
```

6.3 Matrix declaration:

A matrix can declared by its type and name. Type of a matrix can only int or float.

```
matrix m1<int>;
matrix m2<float>;
```

you can initialize a matrix at declaration itself like or you can initialize it within an existing matrix whose type should be same as rhs matrix

```
    matrix m1 <int> = {{1,2,3},{3,4,5}};
    matrix m2 <int> = m1;
```

you can also give the order to the matrix which will by default sets each entry to zero

```
    matrix m3 <int>(3,4);
```

6.4 declaration Scope:

Every variable declared in the program have definite scope i.e.. Every variable has definite boundaries in which it can be accessed or modified. In this language scope is enclosed by $'\{'$,$'\}'$.
Every variable declared between these brackets (in this scope) cannot be used outside it. Based on scope Variables are categorized into 2 types

1. Global Variables (Variables outside the scope which can be accessed)
2. Local Variables (Variables within the scope)

```
int Var = 10;  #Global variable

void function1() {
    print("Global variable: $Var$\n");
}

void function2() {
    Var = 20;  #Modifying the global variable
    int var = 10;
    print("Global variable : $Var$\n");
    print("Global variable : $Var$\n");
}

int main() {
    function1();
    function2();
    return 0;
}
```

6.5 Function declaration

Similar to variables every function must me declared first before calling (using) it or one can define the function before calling it.

Syntax

$$\langle \text{return\_type} \rangle \quad \langle \text{function} \rangle \, (\langle \text{argument\_list} \rangle) \, ;$$

$$\langle \text{argument\_list} \rangle \rightarrow \langle \text{datatype} \rangle \, \langle \text{id} \rangle, \langle \text{datatype} \rangle \, \langle \text{id} \rangle, \ldots \langle \text{datatype} \rangle \, \langle \text{id} \rangle$$

Where ⟨return_type⟩ is the datatype of the variable which the function returns. A function may not have any arguments and a function may return nothing (void).

6.6 Initialization:

Initialization can further be divided into two types

1. **Default initialization:** Unless explicitly declared every variable will be initialised to '0', 'NULL' respectively.

```
int n; # Here n <- 0
string s; # s <- ""
```

2. **Explicit initialization:** We initialise variables to their corresponding constants during declaration.

```
char c ='c' ; # c <- 'c'
int arr[]={1,2,3}; # arr[3] <- {1,2,3}
```

7 In-built functions:

7.1 Dataframe functions:

Any function given below that has expression/predicate as argument may change its syntax and semantics slightly. We also want to implement Union, Intersection and set difference of dataframes.

1. Create a dataframe

```
dataframe df = dataframe();
```

2. Read data from file into dataframe. Read can be any of these two types listed below.

```
df.read(file_name,datatype);
df.read(file_name,datatype,delimiter);
```

Ex:

```
string datatype[ ] = ["int","float","string"];
df1.read("data.csv",datatype);
df2.read("data.csv",datatype,",");
```

3. Read data from an existing dataframe you copy the dataframe as it is or you can request some columns only.

```
string headers[ ] = ["col1" ,"col2", "col3"];
df2 = df.get(headers);
df3 = df2;
```

4. Get columns names of a dataframe

```
string col_names[ ] = df.columns();
```

5. Add columns

```
string headers[ ] = ["new1" ,"new3"];
string datatype[ ] = [ "string" ,"bool"];
df.add_columns(headers,datatype);
```

6. Drop columns

```
 df.drop(headers);
```

7. Delete row

```
 df.delete(predicate);
```

8. Query

```
df.select(headers, predicate);
```

9. Add row

```
df.add_rows(A,null,B,W,null);
```

10. Set all null values to a default value.

```
df.set_null(column, expression);
```

11. Update a column which satisfies an expression with a value.

```
df.update(column,predicate,expression);
```

12. Write data to another CSV file. If headers is null, then all the columns will be written.

```
df.write(file_name,headers,df);
```

13. Basic analysis of data

```
df.describe();
```

14. Operations on Dataframe. op is some operator.

```
    df[column_i] = df[column_j] op df[column_k];
    df[column_i] = df[column_j] op integer;
```

Ex:

```
    df["day"] = df{"date"}%7;
```

7.2 Matrix functions:

1. Addition, Subtraction of matrices

```
matrix m1,m2,m3,m4;
m3 = m2 + m1;
m4 = m2 - m1;
```

2. Multiplications of matrices (Both must compatible)

```
m5 = m2 * m1;
```

3. Transpose of matirx

```
m2.T()
```

4. Inverse of matrix

```
m2.Inv()
```

5. power of matrix(Only Square matrices)

```
m2.pow(4)
```

6. Determinant of matrix(Only Square matrices)

```
m2.det()
```

7.3 Sorting functions:

1. Sorting function can only sort arrays with all int , all float , all characters , all string string entries

```
int arr [ ] = { 1,2,3,4,213};
sort(arr, arr + 2 );
string SS [ ] = { "SSA", "WWE", "Dfdsf" };
sort(SS,SS + 2);
```

2. Default sorting is either ascending or lexicographic-ally , put -1 as extra parameter to do the opposite

```
sort(SS,SS + 2,-1);
```