TEAM 5

COMPILER DESIGN LAB PROJECT

SRIPRANAV MANNEPALLI *(CS18B036)*

S.R.V.S.MAHESWARA REDDY *(CS18B032)*

CHIRAG GUPTA *(CS18B006)*

AVS HRUDAI *(CS18B001)*

SCOPE OF THE WORK

IMPLEMENTATIONS

1. Can Be Done :

- Data Types : Integer , Floating Point , Boolean , Character , String , List , Matrix
- Operators : + , - , * , / , logical and bitwise operators
- Comments
- Expressions
- Conditional Statements
- Iterative statement
- Input-Output Statements
- Functions

2. We Are not sure about :

- Structs / Classes
- Pointers
- Dynamic Memory Allocation
- Arrays

SOURCE LANGUAGE SPECIFICATIONS

DATA TYPES

1) **INTEGER**
   - Keyword Used for representing : “int”
   - Pattern : [+|-]?[0-9][0-9]*
2) **FLOATING POINT**
   - Keyword Used for representing : “float”
   - Pattern : [+|-]?[0-9]+.[0-9][0-9][0-9]
3) **BOOLEAN**
   - Keyword Used for representing : “boolean”
   - Pattern : [FALSE|TRUE]
4) **CHARACTER**
   - Keyword Used for representing : “character”
   - Pattern : [‘][A-Za-z0-9]?[‘]
5) **STRING**
   - Keyword Used for representing : “string”
   - Pattern : [“][A-Za-z0-9]*[“]

6) LIST

Keyword Used for representing : “list”
Pattern : [[[0-9],]*[0-9];]
List element is of Integer Type
List is 1-Dimensional only
List Element can be accessed as L[i]
L is the name of the variable identifier and “i” is the position
Default values of all elements of list are zero.
List is static.
Example :
list p[2];
list r[2]=[1,2;]

7) MATRIX

Keyword Used for representing : “matrix”
Pattern : [[[0-9],]*[0-9];]+
Matrix element is of Integer Type
Matrix is 2-Dimensional only
Matrix Element can be accessed as M[i,j]
M is the name of the variable identifier and “i” and “j” are row and column number
Default values of all elements of matrix are zero.
Matrix is static.
Example :
matrix p[2,3];
matrix r[2,2]=[1,2;3,4;]

IDENTIFIERS

1) **VARIABLE IDENTIFIERS**

Pattern : [A-Za-z]+[0-9]?

Rules :

Variable Identifiers Do not have an Underscore.

Can have at most one digit at the end.

Max length is 20

2) **FUNCTION IDENTIFIER**

Pattern : [_][A-Za-z]+[0-9]?

Rules :

Functional Identifiers Begin with an Underscore.

Can have at most one digit at the end.

Max length is 20

OPERATORS

1) **PLUS** “+” : a+b
   Plus Operation is valid if Both the operands are :
   a) Integers: Then output is an Integer.
   b) Float : Then the output is float.
   c) String : Then the output is a concatenation of the both input strings.
   d) List : Then output is the addition of two input lists (If same dimensions)
   e) Matrix : Then the output is the addition of the two input matrices (if same dimensions)
2) **MINUS** “-” : a+b
   Plus Operation is valid if Both the operands are :
   a) Integers: Then output is an Integer.
   b) Float : Then the output is float.
   c) List : Then output is the subtraction of two input lists (If same dimensions)
   d) Matrix : Then the output is the subtraction of the two input matrices (if same dimensions)
3) **MULTIPLICATION** “*” : a*b
   Multiplication Operation is valid if Both the operands are :
   a) Integers: Then output is an Integer.
   b) Float : Then the output is float.

5) DIVISION “/” : a/b

Division Operation is valid if Both the operands are :

a) Integers : Then the output is Float.
b) Float : Then the output is Float

6) SPECIAL OPERATOR “@” : @a

This operator is valid only on string , list and matrix
This operator returns the size of the operand.
This operator returns a matrix of size 1x2

Example:

```
string p=”hello”
matrix r = @p
Then r is [5;-1;]    ( outputs [length;-1;] for string )

list p=[1,2,3,4;]
matrix r = @p
Then r is [4;0;]     ( outputs [length;0;] for list )

matrix p=[1,2,3;4,5,6;7,8,9;]
matrix r = @p
Then r is [3;3;]     ( outputs [rows;columns;] for matrix )
```

7) **Logical operators** : “&&” and “||” (Logical ‘and’,’or’)

a && b ; a || b;
These operators are used for boolean expressions only.

8) **Logical operator** : “!” (Logical ‘not’)

!a
This operator is used only for boolean expression.

9) **Bitwise operators** : “&” , “|” , “^” (Bitwise ‘and’,’or’,’xor’)

a&b ; a|b; a^b;
This operators are used only for integer type variables.

10) **Increment** “++” and **decrement** “--” operators.

a++ ; a-- ;

COMMENTS

Single Line comments : #

```
# Hello World
```

Multi Line Comments : #* …. *#

```
#* Hello World
How are you ? *#
```

Every Line in our language should end with a Semicolon “;”

EXPRESSIONS

1) **Arithmetic Expressions**:

   This language supports all the expression in the usual “Infix” form and we are following the standard precedence rules and the parenthesis are given the highest precedence.
   Ex: ((a+b)+5)

2) **Boolean Expressions**:

   We require these expressions for the conditional statements.We can use all the relational operators.
   Ex: (a+10 < b+8)

3) **Logical Expressions** :

   Use ‘&&’ and ‘||’ logical operators.
   Ex: (a<b) && (b>=g)

CONDITIONAL STATEMENTS

Only one type of conditional statement is provided in this language. The 'if' conditional statement is of 3 forms; 'if-then', ‘if-then-elif-then-else and 'if-then-else'. Example code is as follows

1)
```
if(p>10){
        r=5;
}
```

2)
```
if(p<10){
        b=5;
}
else{
        b=6;
}
```

3)
```
if(p<10){
        b=5;
}
elif(p<20){
        b=6;
}
else{
        b=7;
}
```

ITERATIVE STATEMENTS

We support a “FOR” loop and “WHILE” loop.
For loop takes three arguments : initialisation ; condition ; increment;
Example :

```
for(int a=6;a<7;a++){
        Statements;
}
```

```
while(a<b){
        Statements;
}
```

- For loop has Initialisation, Condition statement, Increament and the execution statement.
- While loop has a condition part and the execution statement.

OUTPUT STATEMENT

The syntax for output is print(x).

This is straightforward for all the data types.

print:

print(3) / print(“hello”) / print(3.14) etc are not valid.

print can only have an ID as an argument.

```
int x=3;
print(x); is valid and prints ‘3’ on the console

string  x=”hello”;
print(x); is valid and prints ‘hello’ on the console

matrix p=[1,2,3;4,5,6;];
print(p); is valid and prints ‘[                    on the console
                                    1,2,3;
                                    4,5,6;
                                    ]’
```

INPUT STATEMENT

The syntax for input is scan(x)

These are straightforward if the data types are Integer,Floating Point,Character and string.

scan:

scan can only have an ID as an argument and scan can only take Integer/Floating/Character/String as argument.

```
int x;
scan(x); is valid and takes input from console.

string p;
scan(p); is valid and takes input from console
```

List/Matrix input can be taken through iterative loops.

FUNCTIONS

Function name should start with an underscore.
Other rules regarding the function identifiers are already mentioned.

Syntax:

```
datatype functionname (parameters){
        statements;
        return datatype;
}
```

datatype can only be of integer/float/string/character.

Function should return something.

Example :

```
int printhello(){
        string x=”helloworld”;
        print(x);
        int y=0;
        return y;
}
```

NOT SURE ABOUT : Nested functions / Recursion

KEYWORDS

int

float

string

character

boolean

list

matrix

if

else

print

scan

return

elif

while

TRUE

FALSE

for

TOOLS

LEX:

- Lex is a program that generates lexical analyser.
- The lexical analyser is a program that transforms an input stream into a sequence of tokens.

YACC:

- YACC provides a tool to produce a parser for a given grammar.
- It is used to produce the source of the syntactic analyzer of the language produced by LALR(1) grammar

NASM:

- Using NASM as a assembler

HIGH LEVEL DESIGN

LEXICAL ANALYSER:

- We divide the input program into a sequence of Tokens using the “LEX” tool.

SYNTAX ANALYSER:

- Using “YACC” tool , we recognize the syntactic structure of the program and checks if the given program is in the correct syntax of the programming language.

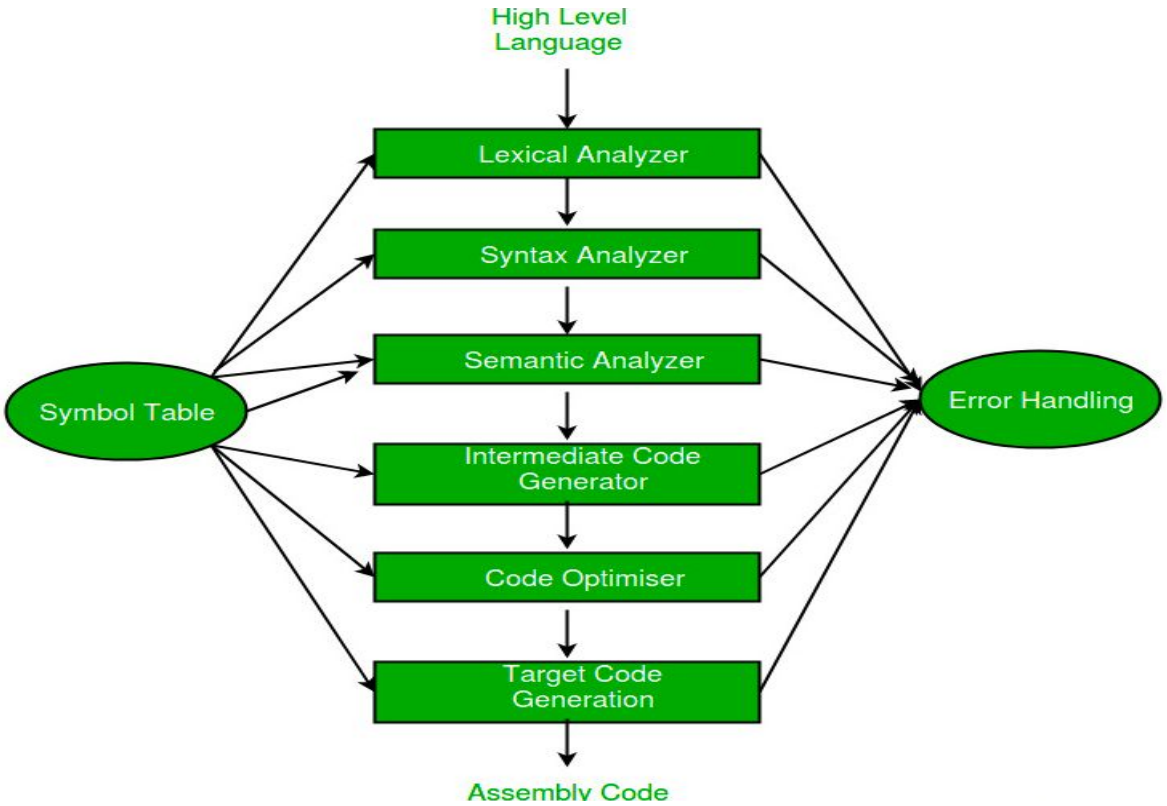
SEMANTIC ANALYSER:

- Using “YACC” tool , we make sure that the declarations and the statements of the program are semantically correct.

INTERMEDIATE CODE GENERATOR:

- Output from the previous phase will be converted into the three-address code.

Compiler phases And Its High level View



EXAMPLES

```
list Li=[3,3,3,3,3;];


for(int i=0;i<5;i++){

      Li[i]=10;

}

print(Li);
```

```
matrix Ma=[1,1;2,2;2,3;];


for(int i=0;i<2;i++){
      for(int j=0;j<3;j++){
            Ma[i,j]=100;
      }
}

print(Ma);
```

TARGET MACHINE SPECIFICATIONS

Architecture:

x86-architecture.

Tool :

NASM assembler: The **Netwide Assembler** (**NASM**) is an assembler for the Intel x86 architecture. It can be used to write 16-bit, 32-bit (IA-32) and 64-bit (x86-64) programs. NASM is considered to be one of the most popular assemblers for linux.

Target Machine:

x86 64-BIT linux machine.