COMPILER DESIGN PROJECT

CS-306

Version : 3.0

INSTRUCTOR : Ashish Phophalia

BHASH LANGUAGE

GROUP MEMBERS -
Kotha Praneeth Sai - 201551044
P Aditya Sai - 201551013
Bhavana Kurra - 201551071
Harsha Vardhan D - 201551043

Contents -

1.INTRODUCTION

BHASH is the name of the programming language that we have tried to implement. We have taken inspiration from many languages and created a new language. We added new features based on the problems we faced while programming.

Features of BHASH -

1. **Keywords** - block, double, int , defVar, break, else, while, switch, case, def, list, boolean, char, string, return, elif, end, continue, for, default, bridge, Label, value, do, if
2. **Statement Terminator** - ';'
3. **Indentations** - Not required. The keyword end is used to point to the end of conditionals (if, if-else etc.) or loop statements (for, while, do while).
4. **Recursion** in the functions is recognized.
5. **Declaration of Variable**: The type of the variable can be mentioned while declaration.
6. Lists are supported.

Things that are not supported by this language -

1. **Pointers** are not supported by this language.
2. They don't support other **abstract data types** like struct, enum, union etc.
3. **Dynamic allocation of memory** for arrays are not supported.

2. GRAMMAR

The words in uppercase are tokens,words in inverted commas { ‘ , ’} are terminals , whereas the words in lowercase are non-terminals.

```
program : declarationlist {printf("Parse Successfully\n");}

declarationlist: declaration declarationlist
                | declaration

declaration : function_definition
            | variable_declaration

function_definition : FUNC ID '(' params ')' statement

variable_declaration : variable_declaration datatype init_dec_list
                     | datatype  init_dec_list

init_dec_list     : init_dec ';'
                       |init_dec ',' init_dec_list

init_dec : ID EQUALTO initializer
         | ID
         | ID '[' NUM ']'   {}
         | ID '[' NUM ']' EQUALTO initializer
         | ID '[' ID  ']'
         | ID '[' ID ']' EQUALTO initializer

initializer : expression
                | CONSTANT
                | list_init
                | NUM
```

```
                | CHAR_CONSTANT

list_init : '{' list_constant_list '}'

list_constant_list :  list_constant_list ',' CONSTANT
                    | CONSTANT

list_constant_list : list_constant_list ',' NUM
                    | NUM

list_constant_list : list_constant_list ',' CHAR_CONSTANT
                |CHAR_CONSTANT


params : paramlist | ε
paramlist: datatype ID
        | paramlist ',' datatype ID

datatype : INT
        | DOUBLE
        | DEFVAR
        | CHAR
        | STRING
        | BOOLEAN
        | BLOCK

expression : ID EQUALTO simpleexpression
           | immutable B_OP simpleexpression
           | immutable inc_dec
           | simpleexpression

inc_dec : INC
        | DEC

simpleexpression : simpleexpression OR_B_RELOP andexpression
```

```
                | andexpression

andexpression : andexpression AND_B_RELOP unaryrelexpression
                | unaryrelexpression

unaryrelexpression : NOT_B_RELOP unaryrelexpression
                | relexpression

relexpression : relexpression RELOP sumexpression
                | sumexpression

sumexpression : sumexpression SUMOP term
                | term

term : term MULOP unaryexpression
        | unaryexpression

unaryexpression : UNARYOP unaryexpression
                | factor

factor : immutable
        | mutable

immutable : ID
mutable : '(' expression ')'
        | CONSTANT
        | call
        | NUM
        | CHAR_CONSTANT
        | TRUE_
        | FALSE_
        | ID'[' expression ']'
        | ID '[' NUM ']'

call : ID '(' args ')'
```

```
args : arglist
        | ε
arglist : arglist ',' expression
        | expression

statement : expressionstmt
                | iterationstmt
                | selectionstmt
                | compoundstmt
                | returnstmt
                | breakstmt
                | continuestmt
                | labeledstmt
                | typecaststatement
                | blockstmt
                | outputstatement
                | typedefstatement
                | inputstatement
                | local_declaration
                | funccallstmt
funccallstmt : funcid_list EQUALTO call
                |func_var_dec_list EQUALTO call

funcid_list : ID | funcid_list ',' ID

func_var_dec_list :  func_var_dec_list ',' datatype ID
                | datatype funcid_list

returnstmt : RETURN ';' | RETURN expression ';' | RETURN args ';'

breakstmt : BREAK ';'
                | BREAK ID ';'
```

```
continuestmt : CONTINUE ';'

selectionstmt : IF '(' simpleexpression ')' statement
              | IF '(' simpleexpression ')' statement ELSE statement
              | IF '(' simpleexpression ')' statement ELIF '(' simpleexpression
')'statement ELSE statement
              | SWITCH '(' simpleexpression ')' statement

labeledstmt : CASE '(' simpleexpression ')' statement labeledstmt
                | DEFAULT statement
                | CASE '(' simpleexpression ')' statement

compoundstmt : ':'  statementlist   END
                       | ':'  END

local_declaration : datatype init_dec_list

typecaststatement : datatype ID EQUALTO
                datatype '(' simpleexpression')' ';'
                | ID EQUALTO datatype '(' simpleexpression ')' ';'
blockstmt : ID compoundstmt

statementlist : statementlist statement
                | statement

iterationstmt : WHILE '(' simpleexpression ')' statement
              | DO statement WHILE '(' simpleexpression ')' ';'
              | FOR '(' for_datatype_dec ID EQUALTO start ':'
                start ':' start ')' statement

start : NUM | ID

for_datatype_dec : datatype
                 | ε
```

```
expressionstmt : expression  ';'
               | ';'

typedefstatement : STRING ID EQUALTO TYPEDEF '(' ID ')' ';'
               | ID EQUALTO TYPEDEF '(' ID ')' ';'
               | datatype ID EQUALTO TYPEDEF '(' ID ')' ';'

outputstatement :  PRINT '(' outvariable ')'  ';'
               | PRINTLN '(' outvariable ')' ';'

outvariable : outvariable ',' value |
                    value
               | expression

value : ID


inputstatement : datatype ID  EQUALTO  INPUT '(' input_value ')' ';' |
                     ID  EQUALTO  INPUT '(' input_value ')' ';'  |
                    ID '[' NUM ']' EQUALTO INPUT '(' input_value ')' ';' |
                    ID '[' ID ']' EQUALTO INPUT '(' input_value ')' ';'

input_value : CONSTANT
      | ε
```

3. SYNTAX

This section presentation the syntax and parse trees of various Control structures:

● Selection Statement:

The standard “If... Else” selection statement.
The following is the syntax:

IF *<conditions>*

:

<statement_1>

<statement_2>

.

.

<statement_n>

END

“If... elif... else” is as follows:

IF *<conditions>*

:

<statement_1>

.

.

<statement_n>

END

elif <conditions>

:

<statement_1>

.

.

<statement_n>

END

ELSE

:

<statement_1>

.

.

<statement_n>

END

Compound Selection statements in the same fashion:

IF <condition1> :

<statements>

IF <condition2>:

<statements>

END

elif <condition3>:

<statements>

END

END

• Iteration statements:

Three kinds of Iterative control statements are supported.
The "DO… WHILE" statement:

do:

<statements>

END

while

<expression> ;

The "WHILE" statement

while *<expression>:*

<statements>

END
The "FOR" statement:

for(*<expresion_statements: <expresion_statements>:<expression>***):**

<statements>

end

Compound Iterations are implemented in the same fashion:

while *<expression>*:

do: for(*<expression_stmnts>*:*<expresion_stmnts>*:*<expression>***):**

<statements>

end

while *<expression>*

end

● **Jump statements**

The following Jump control keywords are implemented:
break, return, continue
//return can take 0, 1 or more than 1 values
return ;
return a ;
return a-b, b-a, a+b, b+a ;
//break can take BLOCK ID and break out of that BLOCK
break ;
break loop ;

● Variable Declarations:

The variables in BHASH need to be declared with a data type.

```
int a = 1;
 double b = 0.5 ;
char c = 'r' ;
string d = "string" ;
int f[3] = [1, 2, 3] ;
boolean s = true ;
defvar a;
```

```
block loop1;
```

● **Functions:**
Functions in BHASH are declared and defined at the same time anywhere in the program.
A parameter accepted by a function can be kept free to accept any data type or can be fixed to accept only a certain data type.

```
func Func_name(a, b, int c, char d):
// Do something to a, b
// Do something to c, d
return a ; //single return
//OR
return a-b, c, int (a)//Multiple returns
END
```

● **Unary Operators:**
Common increment and decrement unary operators “++” and “--” are supported in both postfix and prefix only postforms.
i++, i--

● **Logical operators:**
Most commonly used Logical operators for OR, AND and EQUIVALENCE are supported:
i == (a - b), j < 9, k > 10, l <= 5, m >= 6, n != 14

4. A SAMPLE PROGRAM

1. Program to find the factorial of the given number

```
func main():
int a = input(“Enter a number”);
int b = factorial(a);
print(b);
end
func factorial(int x):
int temp=1;
for(int i = 1 : x : -1) :
temp = temp*i ;
end
return temp;
end
```