Compiler Design Project
Phase-1

Language “X”

By-
Sourabh Rohit (201551065)
Mehak Piplani (201551072)
Ujwal Tewari (201551085)
Neelansh Sahai (201551086)

Contents

Intro to the Language

Our language namely X is an fiscal combination of a predominant part of C grammar and syntax and some parts and function definition of python while at the same time differs from each of their syntax in many places to simplify writing programs.

Basic Idea

To list out the features of X:

a. **Keywords** - if, for, funk, end, else, break, return, continue, while, do, int, float, char, string, boolean
b. **Recursion** in the functions is recognized.
c. **Declaration of Variable**: The type of the variable can be mentioned while declaration.
d. **Arrays are supported**. Also, the default values can be initialized at the time of declaration of an array.
Things the language doesn't support:
e. **Pointers** are not supported by this language.

Compile and Run

On getting compiled our compiler will first of all **“Prettify”** or in basic terms shall indent the code properly and rearrange it as per norms and required indentation. We use braces as our means of indentation.

-

Grammar for our language-

The starting variable for our language is CODE. The capitalize words represent the non-terminals and rest are the terminals for the language.

<CODE> ⇒ <DECLARATION>
| <DECLARATION> <CODE>

<DECLARATION> ⇒ <FUNCTION_DEFINITION>
| <VARIABLE_DECLARATION>
| <STATEMENTS>

<FUNCTION_DEFINITION>
⇒ 'funk '<TYPE> 'function_name' '(' <PARAMETER>')' '{'
<STATEMENTS>
'}'

<PARAMETER> ⇒ <ID_LIST>
| ϵ

<ID_LIST> ⇒ <TYPE> id
| ',' <ID_LIST>

<VARIABLE_DECLARATION>
⇒<TYPE> <DECLARATION_LIST> ';'
| <DECLARATION_LIST> ';'

<DECLARATION_LIST> ⇒ <VARIABLE>
| <VARIABLE>',' <DECLARATION_LIST>

<VARIABLE> ⇒ ASSIGNMENT_EXPRESSION
| id[size]
| id
| id[size][size]

<TYPE> ⇒ int
| float
| char
| string
| Boolean
| long

<EXPRESSION> ⇒ IDENTIFIER OPERATOR
| (EXPRESSION)
| RELATIONAL_STATEMENT
| FUNCTION_CALL

IDENTIFIER ⇒ id //This refers to the variable names
| <STRING> // this is for all kinds of strings
|<CONSTANT> // this is for constant values like in C
|<CONST> // these are basically all types of numbers and size includes only Positive integers

<STATEMENTS> ⇒ <LOOPING_STATEMENTS>
| <CONDITIONAL_STATEMENTS>
| <JUMP>
| <STATEMENTS>
| <EXPRESSION_STATEMENT>
| <VARIABLE_DECLARATION>
|<INPUT_STATEMENT>
|<OUTPUT_STATEMENT>

ASSIGNMENT_EXPRESSION ⇒ id ‘ =’ <EXPRESSION>

<EXPRESSION_STATEMENT> ⇒ <EXPRESSION> ‘;’
| <ASSIGNMENT_EXPRESSION> ‘;’

FUNCTION_CALL ⇒ ‘function_name’ (PASS_PARAMETER);

PASS_PARAMETER ⇒ IDENTIFIER
| ‘,’ PASS_PARAMETER

CONDITIONAL_STATEMENTS ⇒ if '(' <EXPRESSION> ')' ‘{‘
<STATEMENTS> ‘}’
| if '(' <EXPRESSION> ')' ’{‘
<STATEMENTS> ‘}’
else ‘{’ <STATEMENTS> ‘}'
|if '(' <EXPRESSION> ')' ‘{‘
<STATEMENTS> ‘}’
elseif '(' <EXPRESSION> ')' ’{‘
<STATEMENTS> ’}’
else ‘{‘ <STATEMENTS> ‘}’

LOOPING_STATEMENTS ⇒ while ‘(‘ <EXPRESSION>’)’ ‘{‘
<STATEMENTS> ‘}’
| for (<VARIABLE_DECLARATION> ;
< EXPRESSION>;
<EXPRESSION>) ‘{‘
<STATEMENTS> ‘}’

JUMP ⇒ continue ;
| break;
| return ;
| return <EXPRESSION>;

OPERATOR ⇒ UNARY_OPERATOR
| OPERATORS_IDENTIFIER_LIST
| ϵ

OPERATORS_IDENTIFIER_LIST ⇒ OPR IDENTIFIER
| OPERATORS_IDENTIFIER_LIST
| OPR <EXPRESSION>

OPR ⇒ +
| *
| /
| %
| -

UNARY_OPERATOR ⇒ ++
| --

RELATIONAL_OPR ⇒ >=
| <=
| >
| <
| !=
` | ==
| && (... Logical AND)
| || (...Logical OR)

RELATIONAL_STATEMENT ⇒ <EXPRESSION> RELATIONAL_LIST
| FUNCTION_CALL RELATIONAL_LIST

RELATIONAL_LIST ⇒
RELATIONAL_OPR <EXPRESSION>
| RELATIONAL_OPR FUNCTION_CALL
| RELATIONAL_OPR RELATIONAL_LIST

<INPUT_STATEMENT> ⇒ ‘scan’ ‘(’ <PASS_PARAMETER> ‘)’ ‘;’

<OUTPUT_STATEMENT> ⇒ ‘print’ ‘(‘ <EXPRESSION> ‘)’ ‘;’

Ambiguity-

_____

In the above explained grammar, the case that *s = (a+b) - (c+d)* was not covered.

If for the <EXPRESSION> we define it to be

<EXPRESSION> ⇒ <EXPRESSION> <OPR> <EXPRESSION>
| ‘(‘ <EXPRESSION> ‘)’
| <EXPRESSION>
| id

But here we can observe that if we try to build a parse tree for an expression (a+b+c), then we get an ambiguity.

Example Codes-

---

1) **Finding the factorial of a number-**

```
int  a = 10;
funk int factorial(int a){
    int q=1;
    for ( int i=1; i <= a; i ++){
                q=q*i;
        }
   return q;
   }
  int s = factorial(a);
  print(s);
```

2) **Finding Armstrong number-**

```
 int  a = 10;
funk int armstrong(int a){
   int m=a;
       Int d=0,s=0;
       while(m>0){
       d=m%10;
       s=s+cube(d)
       m=m/10;
         }
       }
       if(s==a){
       print(a);
       }
```

```
funk int cube(m){
return (m*m*m);
}
```

3) Finding the sum of cubes for number range one to n-

```
funk int cube(int a) {
        return a*a*a;
}
int b = 1;
int n;
scan(n);
int sum =0;
for(int i=0; i<=n; i++) {
        sum = sum + cube(i);
}
print(sum);
```