Bilkent University

CS 315 Programming Languages

FOTRAN Project 1

Eren Karakaş | 22002722 | Section 1
Oğuz Kuyucu | 21902683 | Section 1
Safa Eren Kuday | 21902416 | Section 1

1.BNF of FOTRAN

1.1 Program

<program> → <stmt_list>

<stmt_list> → <stmt> | <stmt><stmt_list>

1.2 Statement types

<stmt> → <while_loop> | <foreach_loop>
| <decleration_stmt> | <output_stmt>
| <input_stmt> | <return_stmt>
| <assignment_stmt> | < if_stmt>
| <fixed_array_decleration> | <function>
| <func_call> | <comment>

<decleration_stmt> → <identifier_list>SC

<output_stmt> → output <string_const>
| output <logic_expr>SC

<input_stmt> → input <identifier_list>SC

<return_stmt> → return <identifier>SC
| return <logic_expr>SC

<assignment_stmt> → <identifier> ASSIGN <logic_expr>SC

<if_stmt> → if LP <logic_expr> RP LCB <stmt_list> RCB
| if LP <logic_expr> RP LCB <stmt_list> RCB
else LCB <stmt_list> RCB

<fixed_array_decleration> → <identifier> LSB <expr_list> RSB SC

<function> → func <type> <identifier> LP
<identifier_list> RP LCB <stmt_list> RCB

<func_call> → <identifier> LP <expr_list> RP

<comment> → HASHTAG <characters> HASHTAG

1.3 Loop types

<foreach_loop> → foreach <identifier> in <identifier>
LCB <stmt_list> RCB

<while_loop> → while LP <logical_expr> RP LCB <stmt_list>
RCB

1.4 Variables

<identifier_list> → <identifier>
| <identifier>COMMA <identifier_list>

<identifier> → <letter> | <letter> <identifier>

1.5 Types

<type> → bool | array

<string_const> → “” | “<characters>”

<const> → TRUE | FALSE

1.6 Expressions

<logic_expr> → <value>
| LP <logic_expr> <op> <logic_expr> RP
| <not_op> <logic_expr>

<expr_list> → <logic_exp> | <logic_exp>COMMA <expr_list>

<value> → <identifier> | <func_call> | <const> |
<not_op> <value>

1.7 Operators

<op> → & | | | → | ↔ | ^

<not_op> → ~

1.8 Characters

<characters> → <char> | <char> <characters>

<char> → <alphanumeric> | <whitespace> | ; | , | =
| ~ | & | | | (|) | [|] | { | }

<alphanumeric> → <digit> | <letter>

<letter> → [A-Za-z]

<digit> → [0-9]

<whitespace> → “ “ | \n

2.Descriptions of Language Constructs

<program>

This construct represents a FOTRAN program's entirety and consists of statements.

<stmt_list>

This construct represents a statement list, which consists of many statements making up a program or statement block.

<stmt>

This construct represents a statement. A statement can be any unit that expresses an action to be carried out. In FOTRAN, every statement ends with a semicolon. Various types of statements available in FOTRAN are described below.

<decleration_stmt>

This construct is used to declare one or more boolean variables separated by commas in a statement.

<output_stmt>

This construct is used to display output to the console. An output statement can either take string constants or values of logical expressions as parameters.

<input_stmt>

This construct is used to take input from the user. To use the construct, the list of identifiers whose values are going to be assigned has to be given in the statement.

<return_stmt>

This construct is used to return values from functions. In FOTRAN, functions can return boolean values and arrays.

<assignment_stmt>
This statement is used to assign values to boolean variables. Its syntax expects an identifier to assign a value on the left-hand side and a logical expression on the right-hand side.

<if_stmt>
This construct is the conditional statement of FOTRAN. It can be structured as if-then or if-then-else by giving a logical expression in parenthesis, which decides whether the if-block will be executed or not. Curly brackets are required for if-else blocks to avoid ambiguity.

<fixed_array_decleration>
This construct is used to declare fixed-length arrays. It expects the array's name and a list of logical expressions between squared brackets.

<function>
This construct is used to declare and implement functions. It expects the reserved word "func" at the beginning and then takes the return type, the function's name, a list of parameters, and statements inside the function's block. Function blocks are defined using curly braces.

<func_call>
This construct is used to represent function calls. It expects the function's name and a list of expressions in parentheses as parameters.

<comment>
This construct is used to represent comments. In FOTRAN, # symbols are used to indicate where a comment starts and ends. The compiler ignores the comments and does not execute them. However, comments cannot be used between if and else blocks.

<foreach_loop>

This construct is used to form a for-each loop. A for-each loop will iterate through a given array and execute its statements for each of the array's elements. It is mandatory to use curly braces to define a for-each block.

<while_loop>

This construct is used to form a while loop. A while loop will continue to execute the statements under it as long as the given boolean condition results in TRUE. It is mandatory to use curly braces to define a while block.

<identifier_list>

This construct represents a list of identifiers. Identifiers are separated using commas between them.

<identifier>

This construct represents an identifier. It is used to identify and differentiate variables, arrays, and functions.

<type>

This construct represents a type. In FOTRAN, only two types exist, bool and array. It is used to specify whether a function returns a boolean or an array.

<string_const>

This construct represents a string. FOTRAN defines strings as a sequence of characters between quotation marks and allows empty strings.

<const>

This construct represents a constant. Since FOTRAN only defines booleans, a constant can be either TRUE or FALSE.

<logic_expr>
This construct represents logical expressions that can be evaluated. Since FOTRAN has only boolean variables, only one type of expression is used in conditional or other types of statements. In this construct, parentheses are mandatory to avoid the ambiguity that can arise from operation precedence.

<expr_list>
This construct represents a sequence of expressions in order to use them in function calls or array declarations. Expressions are separated using commas between them.

<value>
This construct is the building block of expressions. A value can be constructed by an identifier, a constant, a function call, or another value prefixed by a NOT operator.

<op>
This construct represents operations. In FOTRAN, defined operations are AND, OR, IMPLIES, and DOUBLY-IMPLIES.

<not_op>
This construct represents the NOT operation. NOT operation is separate from other operations as it requires one operand instead of two.

<characters>
This construct represents a sequence of characters.

<char>
This construct represents a character, which can be a letter, digit, or whitespace.

<alphanumeric>
This construct represents characters that can be either a letter or a digit.

<letter>
This construct represents a letter. In FOTRAN, both uppercase and lowercase English letters from A to Z are valid.

<digit>
This construct represents a digit. In FOTRAN, all digits from 0 to 9 are valid.

<whitespace>
This construct represents whitespace. In FOTRAN, whitespace includes " " and the newline character \n.

3.Terminals

LP	→	(
RP	→	)
LCB	→	{
RCB	→	}
LSB	→	[
RSB	→	]
NOT_OP	→	~
OR_OP	→	|
AND_OP	→	&
IMPLIES_OP	→	⇒
DOUBLY_IMPLIES_OP	→	⟺
XOR_OP	→	^
COMMA	→	,
SC	→	;
ASSIGN	→	=
HASHTAG	→	#

4.Non-Trivial Tokens

4.1 Comments

In FOTRAN programming language, every comment starts with a # and ends with a #. The comments are intended to increase the readability of the program. In comments, every character usable in a string is allowed.

4.2 Identifiers

Every logical variable, function, and array has an identifier. An identifier can contain both letters and digits as long as it starts with a letter. It is strongly advised to use meaningful variable names to improve the program's readability.

Some examples:

```
arr1       (for an array)
isAllTrue  (for a function)
isUpper    (for a boolean variable)
12pens     (not allowed)
```

4.3 Literals

String constants: String constants are literals that contain a sequence of characters. They can be used in output statements to communicate with a user or display information. For example, a programmer can specify input requirements by printing a string literal to the terminal.

Boolean literals: Boolean literals are the reserved words TRUE and FALSE. They can represent a state, the result of a logical expression, etc.

4.4 Reserved words

if if is used for conditional statements. It can be used either with or without an else block.

else else has to be used after an if block. It is used to represent the else part of a conditional statement.

func func is used to indicate the start of function declaration.

while	while is used for loops. It indicates the start of a while block.
foreach	foreach is used to indicate a foreach block. It is very similar to the for-each loop of Java.
in	in is used to build a foreach block. It refers to an array, which means an array of boolean values must come after it in FOTRAN.
return	return is used to return values to functions.
input	input is used to take input from users.
output	output is used to print output to users.
bool	bool is used to indicate the return type of functions.
array	array is used to indicate the return type of functions.
TRUE	TRUE represents the true boolean value.
FALSE	FALSE represents the false boolean value.

5.Language Design Criteria

5.1 Readability

The FOTRAN language is simple, which increases its readability. It does not have separate syntaxes for assignment statements. For example, if someone wants to make a boolean reverse of its value, they have to do "a = ~a". Syntaxes such as "a~" are not allowed. Operator overloading is not allowed for similar reasons. Since FOTRAN does not have integers, boolean values cannot be represented using 0 or 1. This similarly increases readability since some programming languages, such as C, introduce unexpected behavior using this conversion.

Reserved words such as while, if, etc. define their usage clearly, and these reserved words can not be used to define variables.

For if block or while block, it is mandatory to use curly braces. This convention is useful to clarify which else belongs to which if or where blocks start and end.

Functions are declared with their return types, allowing users to understand what the program is doing without runtime tests.

5.2 Writability

FOTRAN programming language is quite easy to adapt to for programmers. It has a similar structure to many popular languages in terms of loops, functions, and conditional statements' syntaxes. Declarations, arrays, input and output statements are straightforward to write. Since parentheses are mandatory for logical expressions, expressions might become longer than required; however, this avoids confusion and unexpected results, making the language more reliable. Additionally, a lack of expressivity for the sake of readability would make FOTRAN harder to grasp, decreasing its writability.

5.3 Reliability

The FOTRAN programming language performs its specifications under all conditions.

Type mismatch cannot be experienced because FOTRAN only operates using booleans, arrays of booleans, and string constants. Booleans cannot be assigned to boolean arrays as they can take either TRUE or FALSE. String constants cannot be assigned to or represent either of them. Since data types are so distinct, FOTRAN does not perform any implicit conversions, which are known to confuse programmers.

There is yet to be an exception-handling mechanism for FOTRAN language, as currently, only its BNF specification and lex description exist.

FOTRAN does not allow any aliasing.