Bilkent University

CS 315 Programming Languages

FOTRAN Project 2

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 <logic_expr>SC

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

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

<elif_stmt> → elif LP <logic_expr> RP LCB <stmt_list>
RCB
| elif LP <logic_expr> RP LCB
<stmt_list> RCB <elif_stmt>

<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
| foreach <identifier> in LP <const_list> RP LCB <stmt_list> RCB

<while_loop> → while LP <logic_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

<const_list> → <const> | <const>COMMA <const_list>

1.6 Expressions

<logic_expr> → <logic_expr> <double_imp_op> <implies_term>
| <implies_term>

<implies_term> → <or_term> <imp_op> <implies_term>
| <or_term>

<or_term> → <or_term> <or_op> <and_term>
| <and_term>

<and_term> → <and_term> <and_op> <not_term>
| <not_term>

<not_term> → <not_op>(<value>) | <value>

<value> → <identifier> | <const>

		| (<logic_expression>) | <func_call>
<expr_list>	→	<logic_expr>
		| <logic_expr>COMMA <expr_list>

1.7 Operators

<and_op>	→	&
<or_op>	→	|
<imp_op>	→	->
<double_imp_op>	→	<=>
<not_op>	→	~

1.8 Characters

<characters>	→	<char> | <char><characters>
<char>	→	<alphanumeric> | <whitespace> | ; | ,
		| = | ~ | & | | | (|) | [|] | { | }
<alphanumeric>	→	<digit> | <letter>
<letter>	→	[A-Za-z]
<digit>	→	[0-9]
<whitespace>	→	“ ” | \t | \n | \r

2.General Properties and Conventions

2.1 Beginning of the Execution

In FOTRAN language, execution starts at the very beginning of the file. FOTRAN does not require any special words or functions to indicate the program's starting point.

2.2 Parameter Passing Methods

All functions in the FOTRAN language pass parameters by value. Pass-by-reference is not allowed in FOTRAN.

2.3 Operator Precedence

FOTRAN language defines strict operator precedence rules. The priority of operators is implemented as follows:
NOT > AND > OR > IMPLICATION > DOUBLE IMPLICATION.

If multiple operations exist in a single logical expression, operators will be prioritized according to the order given above. Also, FOTRAN prioritizes inside of the parentheses first.

FOTRAN defines additional logical expressions such as “not_term” “and_term”, “or_term”, and “implies_term” to implement the operator precedence. For example, an “and_term” consists of “not_terms” and other “and_terms”. As a result, “not_terms” are pushed down in the parse tree. Additionally, FOTRAN considers logical expressions inside parentheses as "value", which is why they are pushed down in the parse tree.

2.4 Input Handling

Input statements are used for input handling. As described above in the BNF rules, input statements are defined as “<input_stmt> → input <identifier_list>SC”.

The user will enter values one by one for every identifier in the identifier list of the input statement. String inputs like "t" and "true" are considered TRUE, while "f" and "false" are considered FALSE. Other string inputs are invalid and will cause errors. All integer inputs are considered TRUE, except 0, which refers to FALSE.

Note: There are no conflicts in the FOTRAN language. Also, resolved ambiguities are explained throughout this report.

3. 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, if-then-else, if-elif block, or if-elif block-then-else. if and elif need logic expressions in parenthesis, which determines whether the if-block will be executed or not. Curly brackets

are required for if, elif, or else blocks to avoid ambiguity. Besides, in order to prevent ambiguity, an endif token must be used after the if block.

<elif_stmt>
This construct is used to create "elif blocks" in the FOTRAN language. elif blocks are control statements that are executed if both conditions:

1. no if or elif statements are satisfied before them in the same if block.
2. their condition, which is given as a logical expression in parentheses, is satisfied.

are met.

<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 the if, elif, and else blocks.

<foreach_loop>
This construct is used to form a foreach loop. A foreach loop will iterate through a given array or a constant list and execute its statements for each of the array's elements. It is mandatory to use curly braces to define foreach blocks.

<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 while blocks.

<identifier_list>
This construct represents a list of identifiers. Identifiers are separated with 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.

<const_list>
This construct represents a list of constants separated by commas. It can be used in a foreach loop to enable iterating through a constant list.

<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. This expression is also used to implement double implication. The double implication operator has the lowest priority among FOTRAN's logical operators and is left-associative.

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

<implies_term>
This term consists of "or_terms" and other "implies_terms" in order to prioritize it over DOUBLE IMPLIES. IMPLIES operation is right-associative.

<or_term>
This term consists of "and_terms" and other "or_terms" in order to prioritize it over IMPLIES and DOUBLE IMPLIES. OR operation is left-associative.

<and_term>
This term consists of "not_terms" and other "and_terms" in order to prioritize it over OR, IMPLIES, and DOUBLE IMPLIES. AND operation is left-associative.

<not_term>
This construct represents the resulting logical expression from the "not" operation. It has the highest priority among all operations. Therefore, it will be at the lowest level compared to other logical expressions. Because it is at the lowest level, this construct is also used to represent "value" itself. NOT operation is distinct from other operations as it requires one operand instead of two. NOT operation is left-associative.

<value>
This construct is the building block of expressions. A value can be constructed by an identifier, a constant, a function call, or a logical expression inside parentheses.

<and_op>
This construct represents the logical AND (&) operator.

<or_op>
This construct represents the logical OR (|) operator.

<imp_op>
This construct represents the logical IMPLIES (→) operator.

<double_imp_op>
This construct represents the logical DOUBLE IMPLIES (⟷) operator.

<not_op>
This construct represents the logical NOT (~) operator.

<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 " ", \t, \r, and \n.

4. Terminals

LP	→	(
RP	→	)
LCB	→	{
RCB	→	}
LSB	→	[
RSB	→	]
NOT_OP	→	~
OR	→	|
AND	→	&
IMPLIES	→	->
DOUBLE_IMPLIES	→	<=>
COMMA	→	,
SC	→	;
ASSIGN	→	=
HASHTAG	→	#

5. Non-Trivial Tokens

5.1 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.

5.2 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)
```

5.3 Literals

String: Strings are used in output statements. Their usage purpose may be communicating to users. For example, a programmer may demand boolean values by printing a string to the terminal.

Value: Values are boolean values. They can take true or false.

5.4 Reserved words

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

elif elif is used for else-if 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 else part of a conditional statement.

endif endif is used to indicate the end of an if block

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 true value.
FALSE	FALSE represents false value.

6. Language Design Criteria

6.1 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 blocks or while blocks, 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. Similarly, the endif keyword is used to indicate where an if block ends.

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

6.2 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 FOTRAN provides automatic operator precedence, it avoids using unnecessary parentheses for logical expressions.

6.3 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.

FOTRAN does not allow any aliasing.