CS315- Programming Languages Project-1

NJOY

GROUP 9

İrem Ural
21502278
Section 1

Barış Eymür
21502879
Section 1

Deniz Şen
21502997
Section 1

BNF GRAMMAR

<program> -> **PROGRAM** <prog_identifier> **NJOY** <**program_statements**>
END_PROGRAM

<program_statements> -> <program_statements><program_statement> |
<program_statement>

<program_statement> -> <comment> | <declarative_statement> | <predicate_dec>

<predicate_dec> -><identifier> (<parameters>) **NJOY**<statements> <return_statement>**END**

<statements> -> <statements> <statement>| <statement>

<statement> -> <iteration_statement> | <comment> | <assignment_statement>|
<declarative_statement>| <predicate_call>; | <input_statement> |
<output_statement> | <if_statement>| <return_statement>

<if_statement> -> **IF** (<logical_expression>) **NJOY**<statements> **END** |
IF (<logical_expression>) **NJOY**<statements> **END**
ELSE NJOY<statements> **END**

<iteration_statement> -> <while_statement> | <doWhile_statement>

<while_statement> -> **NJOY_WHILE**(<logical_expression>) **NJOY**<loop_statements> **END**

<doWhile_statement> -> **DO NJOY**<loop_statements> **END**
NJOY_WHILE(<logical>_expression>);

<loop_statements> -> <loop_statements><loop_statement> | <loop_statement>

<loop_statement> -> <iteration_statement> | <comment> | <assignment_statement>|
<declarative_statement>| <break_statement > | <predicate_call>; |
<input_statement> | <output_statement> | <break_statement> |
<if_inside_while_statement> | <return_statement>

<if_inside_while_statement> -> **IF** (<logical_expression>) **NJOY** <loop_statements> **END**|
IF (<logical_expression>) **NJOY** <loop_statements> **END**
ELSE NJOY <loop_statements> **END**

<return_statement> -> **RETURN** <logical_expression>;

<predicate_call> -> <identifier>(<arguments>)

<parameters> -> <parameters>, <parameter> | <parameter>

<parameter> -> STRING <identifier> | BOOL <identifier>

<arguments> -> <arguments>, <argument> | <argument>

<argument> -> <logical_expression> | STRING

<logical_expression> -> <logical_expression> IFF <implies_term> | <implies_term>

<implies_term> -> <or_term> IMPLIES <implies_term> | <or_term>

<or_term> -> <or_term> OR <and_term> | <and_term>

<and_term> -> <and_term> AND <not_term> | <not_term>

<not_term> -> NOT(<factor>) | <factor>

<factor> -> <const_identifier> | <identifier> | <bool> | <array_item> | (<logical_expression>) |
<predicate_call>

<comment> -> /* <comment> < string_elements>*/

<break_statement> -> **STOP_NJOY**;

<declarative_statement> -> **STRING** <identifier>; | **BOOL** <identifier>;|
<constant_declaration>| **STRING** <assignment_statement> |
BOOL <assignment_statement> | <array_declaration>

<constant_declaration> -> **CONST BOOL** <const_identifier>=<bool>; | **CONST STRING**
<const_identifier>=<string>;

<array_declaration> -> STRING @<identifier> = {<array_string_declarations>}; | BOOL
@<identifier> = {<array_bool_declarations>};

<array_string_ declarations> -> <array_string_declarations>, <array_string_declaration> |
<array_string_declaration>

<array_bool_declarations> -> <array_bool_declarations>, <array_bool_declaration> |
<array_bool_declaration>

<array_bool_declaration> -> <string>:<logical_expression>

<array_string_declaration> -> <string>:<string> | <string>: <const_identifier> |
<string > : <identifier>

<array_item_assignment> -> @<identifier>[<string>] = <string>; |
@<identifier>[<string>] = <logical_expression>;

<array_item> -> @<identifier>[<string>]

<assignment_statement> -> <identifier> = <logical_expression>;
| <identifier> = <predicate_call>;
| <identifier> = <string>;
| <identifier> = <array_item>
| <array_item_assignment>
| <identifier> = <input_statement>

<input_statement> -> IN();

<output_statement> ->OUT(<string>); | OUT(<identifier>); | OUT(<bool>); |
OUT(<array_item>);

<const_identifier> -> #<identifier>

<prog_identifier> -> ~ <uppercase> <alphanumerics>

<identifier> -> <letter><alphanumerics>

<connective> -> **AND** | **OR** | **IFF** | **IMPLIES | NOT**

<string> -> “<string_elements>”

<string_elements> -> <string_elements><string_element> | <string_element>

<string_element> -> <alphanumerics> | ! | @ | # | $ | % | ^ | & | * | (|) | _ | - |+ | = | ; | : | ‘
| ? | . | , | > | < | { | } | [|] | `| ~ | \ | / | [ ]

<alphanumerics> -> <alphanumerics><alphanumeric> | <alphanumeric>

<alphanumeric> -> <letters> | <digits>

<digits> -> <digits><digit> | <digit>

<letters> -> <letters><letter> | <letter>

<digit> -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<letter> -> <lowercase> | <uppercase>

<bool> -> **TRUE | FALSE**

<lowercase> -> a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | r | s | t | u | v | y | z | x | w | q

<uppercase> -> A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | T | U | V | Y | Z | X | W | Q

<keywords> -> **NJOY | END |** <connective> **| CONST |** <bool> **| IF | ELSE | NJOY_WHILE | DO | RETURN | STOP_NJOY | STRING | BOOL | END_PROGRAM**

<program> :
This construct signifies the beginning of the program, thus every program that is written in NJOY language must start with the keyword PROGRAM which can only be written at the beginning of the program. After the keyword PROGRAM, the name of the program and the **NJOY** keyword must be written. Between NJOY and END_PROGRAM keywords, the code of the program can be written. This syntax was written for readability and writability issues. The users should be able to code in NJOY as if they are writing in English.

<program_statements>:
Program statements belong to the program between NJOY and END_PROGRAM keywords. It contains the program_statement and it is defined in a recursive way to allow several usage of program statements.

<program_statement>:
In Java language, programmers can write commands/statements in a function body. Likewise in NJOY language, programmer must write a run() predicate because when the program is executed by using interpreter, the system will first call the run() predicate in run-time. Then the run method will call other methods and will take the necessary actions.Inside the predicate programmer is allowed to use any statements defined in <statements> nonterminal. Outside of the predicate, programmer can write a comment, declare a constant or a variable. As these two statements have differences, <program_statements> and <statements> are declared differently.

<predicate_dec>:
Predicate is basically a function which always returns a truth value, true or false. Hence, in NJOY language while declaring a predicate the user should not use a return type before the name of the identifier. Simply an identifier must be defined for the predicate followed by the parameters(<parameters>). Parameters must be enclosed in left and right parenthesis. And the body of the predicate must be inside NJOY and END keywords again. Before the END keyword the user must write a return statement inside the body part.

<prog_identifier>:
The program identifier is a special kind of identifier. It is also a variable identifier but in this case, it holds the name of the program. The program identifier can be written in any alphanumeric form, except that it must start with ~ and an uppercase letter respectively. This restriction is important for the lexical analyzer not to confuse <prog_identifier> with <identifier>. Also, it is better for the name of the program to fit into general programing conventions.

<statements>:
Statements consists of several statements. They are defined in a left recursive manner in order to emphasize left associativity rule of NJOY language. They are used in the body of the program, hence they can be any valid statement defined in <statement>.

<statement>:
This construct generalizes any kind of code in the language. It is important for creating an interface between codes. They can be used in predicate body. It contains iterative statements, like while and do while statements, if statements,comments, assignment statements, input/output statements and declarative statements. In predicate, programmers are allowed to call other functions, and can use return statements inside while or if statements as well.

<if_statament>:
In bnf grammar if_statements create dangling-else problem, NJOY language resolves this issue by using NJOY and END keywords, which must be declared in capital letters. Hence these keywords prevent ambiguity. Programmer can define if statements as well as if-else statements and nested if statements can be defined as well.

<iteration_statement>:
This the interface of <while_statement> and <doWhile_statement>. This part was created for the sake of simplicity because in the language grammar, there are many places where we need to handle each loop at once and by the help of this construct, it becomes easier.

<while_statement>:
This is the standard while loop that can be found on any programming language. It keeps going through the same statements inside loop, until the specified condition becomes false. In NJOY, every while loop must start with a NJOY_WHILE keyword. It is followed by a left parenthesis and a boolean expression(<logical_expression>). This expression is the condition that specifies whether the loop should continue executing or not. The end of the boolean expression should be specified by using right parenthesis. The code that the loop should execute is written between NJOY and END keywords. The code inside of while_statement must be a part of <loop_statements>.

<doWhile_statement>:
The usual do-while loop is included in NJOY as well. The difference between while statement and do-while statement is that the condition checking is done after the code execution rather than before. So the code inside the loop is guaranteed to be executed at least one time. The syntax of the do while statement is as follows; first, a DO keyword must be written. Then the code block must be written between a NJOY-END block. The code must also be a part of <loop_statements>. After the code block, a NJOY_WHILE keyword should be written. After that, the condition should be written between parentheses, as a boolean expression. The do-while loop must end with a semicolon, so we need to put it at the end of the right parenthesis.

<loop_statement>:
The statements that can be written inside a loop are different than usual statements. It is forbidden to declare predicates inside loops, which is the part where <loop_statement> differs from <statement>. However, it does not have any difference in terms of syntax.

<loop_statements>:
This part only represents multiple <loop_statement> inside a loop, as the user can write more than one statement inside a loop. By using recursion, this construct handles every combination of multiple <loop_statement>.

<if_inside_while_statement>:
This construct is defined in order to define break statements inside of if statement. However programmer can only define break inside of if_statement, if if_statement is located in an iteration statement, which are while and so while statements in NJOY language.

<return_statement>:
Return_statement is used inside of an predicate. It is similar to any other programming languages. The user must start with RETURN keyword, which is like all other keywords written in uppercase letters**.** The return value must be specified afterwards. It can be an identifier, a logical expression or a predicate call, as predicate call and identifier are both inside the logical expression, to avoid ambiguity we prefered to specify it as logical_expression. The value of each type will be returned to the caller. At the end of the return statement there must be a semicolon to show the end of the statement.

<predicate_call>:
Predicate call is used in order to call a predicate inside of the run() predicate or another predicates. The identifier which is used in declaration must be used in predicate call. The arguments must be specified as well, if parameters are specified while declaring. There must be always left and right parenthesis after the identifier. Left and right parenthesis allows our NJOY language to recognize it as a predicate. Arguments must be written inside of these parenthesis. When it is used as a statement programmer should not forget to put semicolon after the predicate call.

<parameters>:
This construct is to represent multiple parameters inside a <predicate_dec>. It also covers every possible way of writing parameters, by using left recursion. When declaring parameters, the user must put a comma between different <parameter> declarations. It is a very conventional language grammar, therefore it is good for NJOY's writability.

<parameter>:
Parameters are written only with predicate declarations(<predicate_dec>). In NJOY, every variable has a initially specified type, on predicate declarations, the parameter's type must also be declared. Therefore, <parameter> is always constituted by a type, STRING or BOOL, along with an <identifier> for the variable that will be used inside the predicate definition, though programmer can define it as empty as well. It is a very standard representation across the programming languages.

<arguments>:
This nonterminal stands for the several arguments which can be defined during predicate call. By using left recursion different number of arguments can be accepted, however it should match the number of parameters defined in declaration. The arguments are separated by a comma.

<argument>:
Arguments are used during the predicate call. The NJOY language provides pass by value, hence the value of each argument will be used inside of the predicate. The value of each variable cannot be changed inside the predicate as well. Arguments can be an identifier, its type can be string or bool, a predicate call can be used as an argument as well, its return type will be used. Additionally TRUE, FALSE keywords can be used directly, a string in quotes can be passed into predicate and lastly a constant identifier can be used in the arguments. The type of each argument must match with the type of each parameter that are used in declaration respectively.

<logical_expresion>:
Logical expressions include the basics of propositional calculus. The main focus of the NJOY programming language is to allow programmers an efficient programming based on propositional calculus. In our language the logical expressions has precedence, IFF is used for if and only if. User must use it in capital letters. If only if has the lowest priority in NJOY language. Hence while computing other operations will be prioritized. It is defined in a left recursive manner. It consists of logical expression IFF and <implies_term> or just an <implies_term>.

<implies_term>:
This nonterminal is defined in order to specify the precedence of logical operators. Implies_term consists of or_term followed by IMPLIES keyword, then an implies_term at right hand side, or just an or_term can be used. It is defined in a recursive manner as well, and most importantly, implies operation is right associative. It has a higher presedense than IFF however lower presedence than OR.

<or_term>:
The precedence of operators is considered during declaration of or_term. It consists of an or_term at the left hand side of the expression followed by a OR keyword, written in capital letters and an and_term or only an and term can be used. It is defined in recursive manner as well. The keyword must be used in capital letters to indicate it is a logical operator.

<and_term>:
And_term is defined in order to explain how can be used in our language, and to show in which precedence level it belongs. And has a higher precedence than or hence it can be seen in deeper levels in parse trees. And_term is defined recursively. It consists of an and_term left hand side followed by AND keyword. At the right hand side a not_term is defined. Instead of these, a not_term can be used as well. AND keyword must be written in uppercase letters.

<not_term>:
This term was included to signify the precedence of NOT connective. Not operator has a higher precedence than and, therefore it was written such that <and_term> would lead the parse tree to <not_term>. Not operator can be expressed in NJOY by writing NOT keyword along with left and right parentheses. Inside the parentheses, any expression that defines a boolean value can be written. <not_term> will take the inverse of that expression.

<factor>:
Factor is defined to show the highest precedence in NJOY language. Parenthesis has the highest priority. Factor can be a constant identifier, identifier which belongs to a bool expression, truth values, which are TRUE and FALSE in NJOY language, predicate call, which returns a bool result and the result will be used while computing the result of the whole expression. An array_items value and lastly an logical expression which enclosed in left and right parenthesis can be used as well. It shows that parenthesis has the highest precedence. It contains the basic units which participate in logical expressions.

<assignment_statement>:
When we give a variable a value, we use the assignment statements. However, <assignment_statement> does not include variable declarations. Assignment statements always start with an <identifier> and equal sign(=). On the right hand side of the equation, there can be 1 of the following 8 constructs: <bool>, <identifier>, <logical_expression>, <predicate_call>, <const_identifier> , <string_elements>, <array_item> and <input_statement>. Additionally, it contains <array_item_assignment>. Also, assignment statements must end with a semicolon, like most of the statements. More importantly, the 2 hand sides of the equation must be from the same type. <assignment_statement> is written so that the convention of equality is not broken, because most of the programming languages use equal sign to indicate assignment and NJOY is no exception in this regard. It is focused on writability mostly, but it also helps with the readability.

<declarative_statement>:
Our language has a requirement that every variable must first declared, along with its type, that can be either STRING or BOOL. <declarative_statement> is used in this case. When a variable is declared, it can either take its value in the same line of code, or it can be declared without initially having a value. <declarative_statement> also includes <const_declaration>. Array_declaration belongs to this construct as well. Declaration can be done on any scope inside the PROGRAM-ENDPROGRAM scope. Declaring a variable along with its type is a fact that increases readability of the code as the coder can easily remember the type of a particular variable with a single look at its declaration.

<constant_declaration>:
Constants are immutable throughout the program and they must be assigned value at the same time they are declared. Constants can be of any type which are either BOOL or STRING. The type of the constant must be given just as any declaration. In terms of syntax, there is a substantial difference between <declarative_statement> and <constant_declaration>, which is the fact that constant declarations must start with the keyword CONST, which must be written in capital letters. At the end of declaration a semicolon must be used as well. This type of syntax is good for writability because it is in a form such that the assignment part is same as regular assignment.

<const_identifier>:
This construct is declared to specify the restrictions in naming the constants. The constant names must start with # and it should be followed by an uppercase or lowercase letter. In later characters digits and any letters can be used.

<break_statement>:
Programming languages which contain any kind of loop, often provide a keyword that stops the loop at any instance. In NJOY, this keyword is defined as STOP_NJOY, as it stops the execution of the loop which contains the <break_statement>. As every loop declaration has a “NJOY” part inside, the keyword that “stops” the loop should have a “NJOY” in it. Therefore, <break_statement> also includes “NJOY” in its keyword. It can be declared only inside of the iteration statements and the if statements which are inside of the iteration statements.

<connective>:
Connectives are used in propositional calculus. NJOY language offers and, or, if and only if , implies and negotiation operations. Different than the other connectives, implies operation is defined as right associative. The other operations like xor, xnor can be defined using these basic logical operations. They are reserved words and written in capital letters, it is preferred in this way to increment readability in NJOY language. The user will be able to understand each operation easily without recognizing any symbol or keyword. AND, OR, IFF, IMPLIES, OR keywords are used for these operations.

<bool>:
Bool defines the truth values used in NJOY. They are basically true and false and they are written as TRUE and FALSE respectively, It increments readability in NJOY language, it allows user to recognize values of each boolean easily.

<identifier>:
Variable names are called identifiers. The language does not have a restriction for the length of the identifiers. However, the identifiers must start with a letter and can only contain alphanumeric characters. Containing only alphanumeric characters increases the writability of the language as most of the non-alphanumerics are hard to type on keyboards.

<alphanumerics>:
Alphanumerics can be letters or digits. In order to allow users to employ several letters and digits together in any order alphanumerics is defined. Therefore, it is declared recursively in BNF, it consists of alphanumerics or alphanumeric.

<alphanumeric>:
Alphanumeric can be either letters in English language or digits in Mathematics.

<digits>:
Digits is constructed to write several digits together sequentially. Hence it is defined recursively. A digit can be followed by a digit is acceptable in identifiers or string in NJOY like in other several programming languages.

<letters>:
Letters is defined in order to allow programmers to write several letters consecutively. The definition of letters is recursive to provide this feature. It is used in the nonterminal alphanumerics.

<digit>:
Digit is constructed to represent each digit character from 0 to 9. They can be used in a string or in an identifier.

<uppercase>:
NJOY uses the letters from English language, therefore <uppercase> covers these letters' uppercase representations.

<lowercase>:
English letters' lowercase representations are covered by <lowercase>.

<letter>:
All the possible letters are covered in this construct. <letter> is either a <lowercase> or a <uppercase> in NJOY.

<string>:
String consists of string elements in quotes. It can be any character, space, symbols, digits and letters in both lowercase and uppercase forms. It must be defined in a single line by the programmer, hence inside of the string programmer should not use any new lines.

<string_elements>:
String elements are defined because of the fact that programmer must be able to write a string by using several string elements in any order.

<string_element>:
String_element is basically any character on the keyboard, and the programmer is allowed to define a string like in all other languages.Hence, this nonterminal is defined in the definition of string.

<keywords>:
Keywords are defined to make a readable, writable and reliable language. It enables programmers to develop several features in a reliable way. NJOY and END keywords are declared to highlight the beginning and end of a predicate, selection statements, iteration statements or a program. For end of the program to make our program more readable END_PROGRAM keyword is used. They bring out the body of these statements. The connectives are keywords as well. Additionally, the types of variables which are BOOL and string are in this category. The words used in iteration_statement, NJOY_WHILE, DO are also reserved word. CONST used in constant declaration, RETURN which is used in return statement in a predicate, TRUE and FALSE used as truth values and IF, ELSE used for selection are all in the keyword nonterminal. These keywords are reserved by NJOY language, hence programmers cannot use these keywords in the same writing style in any identifier.

<comment>:
Comments are the parts which cover some strings. In NJOY, comments can be formed in one way. The programmer can write comments by using single-line comment which can be called by writing "/*". This will indicate the start of the comment and until it finds a "*/" token it is considered as comment. "*/" will end the single-line comment. This syntax is helpful in

terms of writability. It is composed of string elements which are declared in <string_elements> nonterminal.

<input_statement>:
In NJOY, the user inputs are taken by a special function called IN() followed by a semicolon. When IN() is called, the program stops executing and waits for the user to enter an input. The input is returned afterwards, meaning that it can be assigned to variables. IN() function never takes an argument, otherwise an error will occur. Considering there might be many inputs, the function identifier is defined as short as possible. This feature increases the writability of the language.

<output_statement>:
To print a string of characters on the screen, the user can call a predefined function, which is OUT. OUT function takes arguments which are desired to be printed on the screen. The syntax of the function is set to contain only one argument at a time and to not allow concatenation inside the argument. The argument can only have one type at a time as there isn't a concatenation feature. This function does not return any value therefore it cannot be used in <assignment_statement>. The identifier of the function is short and explanatory enough for the sake of writability.

<array_declaration>:
The built-in data structure of NJOY is a regular associative array since there is not a data type that can be used as an index such as integer type. This data structure can hold data from 2 supported types: string and bool. Every item has a key which can be only in string type. The declaration of the data structure can only be made by first specifying the type of the data, followed by a special array identifier, which starts with a '@' character. Then, a regular assignment operator, followed by either a <array_string_declaration> or an <array_bool_declaration> inside left and right curly braces. The declarative statement must end with a semicolon. The data structure is static and homogenous in NJOY language hence programmer cannot define both booleans and string types in the same array. Once they are created they cannot be changed as well.

<array_string_declarations>:
This rule consists of several <array_string_declaration> statements. They are defined in a left recursive manner in order to emphasize left associativity rule of NJOY language. They are used inside the <array_declaration>. There must be a comma between multiple <array_string_declaration> statements.

<array_string_declaration>:
An array item from type string can be declared by first writing its key in a string form, followed by a colon and the value which the key holds. The value can be declared by including a string variable or directly a <string>.

<array_bool_declarations>:
This rule consists of several <array_bool_declaration> statements. They are declared in a left recursive format, since the NJOY language has a left-associativity rule. They can be any valid statement defined in <statement>.

<array_bool_declaration>:
A bool array element can be declared by first writing a key which will be of type string, then putting a colon(:) next to it and then writing the logical expression (it can also be a predicate call) which will give the bool value.

<array_item_assignment>:
The value of an array item can be re-assigned by a separate statement. First the identifier of the array should be written(with a '@' character at the beginning). Then, the key which holds the value that is to be changed is written inside left and right square brackets. After that, a regular assignment operator is needed, followed by the new value of the array item. The new value of the array item must be from the same type as the type of the array, otherwise, an error will occur. The RHS of the assignment can be a <logical_expression> or directly a <string>.

<array_item>:
An array item can be obtained by writing the array identifier, then writing the key of the desired item as a string between square brackets([key]).

NJOY is a language that is mainly designed for simplicity of writing. In particular, the tokens are selected such that coders should be able to write their code as if their writing in English. This is why the tokens are mostly explicitly written in English. But simplicity by itself might have caused some problems for the language, for instance if every single token such as “=” had an English equivalent in the syntax, the program may have been easier to write, but it would also be extremely long, which would diminish the readability of the program. On the other hand, there are some commonalities between every high level programming language, such as using “=” as the assignment operator. The tokens and the syntax of NJOY also considers these commonalities and follows them. This fact makes the language easier to learn by coders who have been in the business for long, since coders get used to some conventions across programming. They can easily adapt on NJOY by using their old syntax knowledge. NJOY is reliable in some ways too. In the language, the type checking is essential, because the predefined types should not be confused by the compiler. Also the language is designed to be able to execute detailed errors, which can help the coder debug their code more easily. The scopes are precise in the language definition, which will be helpful for checking processes. Overall, NJOY is designed to address any kind of coder whether they are veterans or beginners. It is mainly focused on writability, but it is also readable and reliable in some ways.