C--

PROJECT 2 REPORT GROUP 5

SEC 02

Farid Mirzayev 21603178

Okan Alp Unver 21702632

Nihat Bartu Serttas 21702301

<main> : Means the beginning of the program and shows how the statements should be written between { and } in order to understand when the program ends.

<stmnts> : Statements are lines of code. There can be zero or more lines of code.

<stmnt>: Just one instruction. There are many kinds of instructions in C--. Most of them are very similar to known languages such as C and Java. There are the exceptions of set based operations.

<declaration> : Statement which contains a basic declaration of a variable by indicating the data type and the identifier name.

<dec_init_operation>: In addition to declaration, this statement contains the initialization of the identifier with a value.

<assignment_operation>: This is a function for assigning a new value to the variable. User can use equal sign for assigning a value to variable

<assignment_operator_right>: The right hand side of the equals sign in an assignment operator can be different than the left hand side. In order to use different statements such as function calls with return values or simply values alone, we have <assignment_operator_right>

<create>: Create is used with an identifier to create a set. Basically it is the same as a normal declaration for a set.

<delete>: This function is for deleting the set mentioned. Users will use the "delete" keyword with a variable name in order to delete a set.

<if>: This means if conditional statements. If the Boolean condition inside the parentheses is true, the statement inside the curly braces will be executed. If there exists an else part as well, and the condition is false, the statements inside the curly braces that come after the else will be executed.

<while>: This means while loop. Inside the parenthesis, is a Boolean condition. Until this condition becomes false, the statements inside the curly braces will keep repeating.

<funct_definition>: Function definitions contain the return value of the function, name of the function, and inside a parenthesis, all the parameters and their types for being used in the statements.

<function_definition_params>: This is the entire collection of function parameters. A function can take zero or more function parameters, so it calls itself recursively.

<non_empty_function_definition_params>: Inside the <function_definition_params> if there are more than one parameter we use <non_empty_function_call_params> to recursively represent as many as we want.

<funct_param> : A function parameter is the required item's data type and name. functions can take zero or more parameters.

<funct_call>: Function calls are similar to function definitions, without the return types

<function_call_params>: Function call parameters are just variable names or just values.

<non_empty_function_call_params>: Just like <non_empty_function_definition_params> we use <non_empty_function_call_params>

<read>: Read is the input function which can be used to take inputs from the user.

<write>: This function is used to print out outputs in the terminal.

<return>: Return statement of a function definition.

<boolean_expression>: Defines all Boolean expressions in this language. <boolean_expression> represents <bool_set_operation> and <logic_expression> at the same time. Which means that all types of operations which return bool type can represent with this notation.

<boolean_logical_expression>: All kinds of logical expressions excluding logical set operators. This is an expression that returns a boolean value.

<bool_set_operation>: Logical set operations such as subset, superset etc.

<logical_operands> : This contains all logical operators that will be used between variables.

<set_bool_operator>: Defines subset operator, proper subset operator, super set operator, proper superset operator, and equivalent set operator.

<set_operation>: Contains union operation, intersection operation, difference operation, complement operation and cartesian multiplication.

<union_operation>: Is a function for setting all elements in a collection into one set. Users will use union_operator to merge two sets.

<intersection_operation>: This function will define the contents of the largest set with common elements which are in both sets. Users will be able to use this operation by using intersection_operator.

<difference_operation>: It is a function for defining the contents of a set as elements which appear only in one set. In order to use difference operations, users need to use difference operations .

<complement_operation>: This function is for setting the contents with elements which are not in set.

<cartesian_operation>: This statement is used to cartesian multiply two sets and return the output set.

<set_type>: This is the value of the data type for set and setofsets, such as \$1,2\$ or \$\$1\$, \$2,3,4\$\$

<var_type>: These are the values of primitive data types such as 1, “fsdlfksd”, 37.40 etc.

<data_type>: Data types of primitive values such as int, double etc.

<set_data_type>: Data types of set values set and setofsets.

<identifier>: Defines the identifier of this language. Identifiers are used for defining variable names, function names and other fields that need an identifier to be named. According to this definition, identifiers of this language must start with a <alphabetic> notation and continue with any sequence of <alphabetic> and <digit> combination.

<string>: Defines strings of this language. According to this definition string type contains <char_array> between two “ signs**.**

<comment>: A comment represents everything that is not a statement and they need to be written between // and //.

<char_array>: Defines the char array of this language. According to this definition, <char_array> can represent zero, one or more than one element in itself. Since <char> represents <alphabetic> and <digit> type, char array can be interpreted as any combination of all digits and alphabetic characters.

<char>: Defines the chars of this language. According to this definition chars of this language can contain only <alphabetic> and <digit> types.

<double>: Defines all double values in this language.

<integer>: Defines all the integers in this language.

<number>: Defines all positive integers and 0.

<signs>: Defines the signs of this language. <signs> contains the ‘+’ and ‘-‘ signs.

<digit>: Defines the digit characters in this language, <digit> contains all digits at once.

<alphabetic>: Defines the alphabetic characters in this language, <alphabetic> contains all letters with their capital version.

<bool>: Defines bool types in it which are true and false.

<empty> -> ε

Terminals of C--
identifier - This terminal produces tokens that are used in the named constructs such as
predicates, variables, etc.
comment - comments of C--, example comment: // This is a comment create -
terminal for set declaration in C--.
delete- terminal for deleting sets.
write - Terminal used in printing an output.
read - Terminal used to get an input from the user.
uni- Terminal used for assigning the union of two sets
intx- Terminal used for assigning the intersection of two sets
diff- Terminal used for assigning the difference of two sets
comp- Terminal used for assigning the complement of a set
if- the if statement
else- the else statement
while-the while statement
return- the return command
“ = ” - terminal of assignment operator in C--.
“ (“ left and “) ” right parentheses of C--.
“ [“ left and “] ” right square brackets of C--.

“ { “ left and “ } ” right curly brackets of C--.
“ ! ” - Negation mark of C--
“ , ” - Comma in C--
“ < “ - Terminal corresponding to less than
“ <= ” - Terminal corresponding to less than or equal to “ > ” - Terminal corresponding to greater than
“ >= ” - Terminal corresponding to greater than or equal to “ == “ - Terminal corresponding to is equal.
“ != “ - Terminal corresponding to is not equal.
“ && “- Terminal for logical and operation.
“ ||”- Terminal for logical or operation.
“ = “ - Terminal corresponding to assignment.
“ $ ”- Terminal for starting and ending the sets.
“ $< “ - Terminal corresponding to proper subset of sets .
“ $<= ” - Terminal corresponding to subset of set data type.
“ $> ” - Terminal corresponding to proper superset of set data type.
“ $>= ” - Terminal corresponding to superset of set data type.
terminals corresponding to “ + ” addition, “ - ” subtraction, “ * ” multiplication, “ / ” division in C--.

Conflicts:

There are not any conflicts or warnings caused by the yacc file.

Readability & Reliability & Writability:

Comments: The user-friendly syntax is just like the other user-friendly languages like java and C++. For the sake of readability of the program the comment symbol (//) will be written on the start in order to distinguish comments from the code. To lessen the mistake rate comments are used on every new line. This makes the program reliable in case of difference between code and comments. Also, this feature makes the program more writable.
Identifiers: The concept of variable names starting with letters are used as common code convention, symbols cannot be used in the variables. Only characters and alphabet with digits can be used which is assisting the writability and readability of the program. The feature of not being able to use complex characters makes the program appropriate in case of making mistakes and makes it a more reliable language. In addition to these benefits, the debugging process of the program is convenient.
Reserved Words: set, String, int, double, bool, read, write, if, else, while, return. To make the language readable and easy to implement its syntax, reserved words are

kept short and simple. With this feature, our program becomes understandable for the developers. Understanding the program will help decrease the mistakes of the coder.

Explanation for yacc file of our language

%token MAIN - Indicates main program
%token COMMENT - Indicates comment line
%token BOOLEAN, INTEGER, DOUBLE, STRING - Indicates primitive data values
%token SET, SETOFSETS - Indicates set values
%token SET_TYPE, SET_OF_SET_TYPE - Indicates set types
%token STRING_TYPE, INT_TYPE, VOID_TYPE, DOUBLE_TYPE, BOOL_TYPE - Indicates primitive data types

%token READ, WRITE - Indicates reading and printing operations
%token IF, ELSE, WHILE - Indicates if, else, while statements
%token RETURN - Indicates return command

%token CREATE_OP, DELETE_OP, UNION_OP, INTERSECTION_OP, DIFFERENCE_OP, COMPLEMENT_OP - Indicates for creating, deleting, union, intersection, difference and complement operations.

%token LP, RP - Indicates Left and Right parenthesis
%token LSB, RSB Indicates left and right square brackets
%token LCB, RCB Indicates left and right curly braces
%token NEGATION Indicates “!” symbol
%token COMMA Indicates “,” symbol

%token MULTIPLICATION, DIVISION, ADDITION, SUBTRACTION, AND, OR, LESS_THAN, LESS_OR_EQ, GREATER_THAN, GREATER_OR_EQ, EQUALS_OPERATOR, NOT_EQUALS_OPERATOR - Indicates logical operands. “*”, “/”, “+”, “=”, “&&”, “||”, “<”, “<=”, “>”, “>=”, “=”, “!=”

%token ASSIGNMENT_OPERATOR - Indicates “=” symbol

%token PROPER_SUBSET_OPERATOR, SUBSET_OPERATOR, SUPERSET_OPERATOR, PROPER_SUPERSET_OPERATOR, EQUIVALENT_OPERATOR, CARTESIAN_OPERATOR - Indicates set boolean operators. “$<”, “$<=”, “$>”, “$>=”, “$==”, “$*” symbols respectively.

%token SEMICOLON - Indicates “;” symbol
%token IDENTIFIER - Indicates name of a variable
%%

Start: detects the defined structure of a program which is written in our language that must start with “start” and should be enclosed with curly brackets.

Ex:

```
main {
        statement;
        statement;
        .
        .
        .
}
```

stmnts, stmnt: stmnts stands for represent all of the statements(stmnt) in our programing languages which are:

Declaration, declaration and initialization, assignment, create, delete, if, while, function definition, function call, read, write, return and comment.

All of the statements must end with “;” in our language.

Ex:

```
statement;
```

Declaration: matches structure of declarations of variables, sets
Dec_init_operation: matches structure of declaring and initializing variables and sets
Assignment_operation:
Create: matches structure of creating set
Delete: matches structure of deleting set
If: matches the structure of if statement

Ex:

```
if ( ! b  ){
    write("ok");
}
```

While: matches the structure of while statement

Ex:

```
while(true){
        write(false);
}
```

Funct_definition: matches structure of functions while defining. 3 different structure for the type of function

Funct_call: matches structure of function while calling. Functions are called using only the name of a function.

Ex:

```
calledFunction(set1);
```

read: matches structure of reading statement
write: matches structure of printing statement
Return matches structure of return command
Boolean_expression indicates all of the code blocks which return boolean type of data structure.

Ex:

```
if( setNumbers <= setOfsets ){
while(true){write(false); }}
```

Bool_set_operation indicates all of the boolean set operations mentioned in the description of the first project. Bool set operator arguments must be set type or so.

Ex:

```
exampleSet $< anotherSet;
```

Set_operation indicates all of the set operations which are union, intersection difference, complement, cartesian.

Ex:

```
set unionSets = set1 uni set2;
```

Union_operation indicates union operation between sets.

Ex:

```
set unionSets = set1 uni set2;
```

Intersection_operation indicates intersection operation between sets.

Ex:

```
set intersectionSets = set1 intx set2;
```

Difference_operation indicates difference operation between sets.

Ex:

```
set differenceSets = set1 diff set2;
```

Complement_operation indicates complement operation between sets.

Ex:

```
set complementSets = set1 comp set2;
```

Cartesian_operation indicates cartesian product operation between sets.

Ex:

```
set cartesianSets = set1 $* set2;
```

Additional Information:

We didn't implement the for loop because the contents between the parentheses for example the condition checking and update statements can vary a lot which means there are plenty of rules to be implemented for loops in known programming languages such as C and Java. Furthermore, since this programming language is highly related with set operations and their all properties (union, intersection, complement, difference, cartesian product) and their additional properties which are(checking whether the set is a subset of the other set) it is impossible to implement between the parentheses in for loop. As a solution we have decided not to implement a for loop because we already have while loops for this purpose.

Moreover in the example code we didn’t give examples for every boolean expression and logical operands because there are lots of them and we have no conflicts whatsoever. We gave a few basic examples for testing purposes.