Can Tücer	22203239	Section 1
Artun Berke Gül	22203316	Section 2
Orhun Güder	22202471	Section 2

1. Name of The Programming Language

We have decided to name our language as ç++.

2. BNF Description

```
<program> ::= <statement_list> <program> | <comment_list> <program> | <comment_list> | <statement_list>

<statement_list> ::= <statement> <SEMICOLON> <statement_list> | <statement> <SEMICOLON>
<statement> ::= <void_statement> | <expression>

<comment_list> ::= <COMMENT> <comment_list> | <COMMENT>

<void_statement> ::= <if_statement> | <loop> | <function_def> | <type_def> | <BREAK> | <CONTINUE>

<if_statement> ::= <IF> <expression> <DO> <LBRACE> <statement_list> <RBRACE> | <IF> <conditional_expression> <DO> <LBRACE>
<statement_list> <RBRACE> <ELSE> <DO> <LBRACE> <statement_list> <RBRACE>

<loop> ::= <while_loop> | <for_loop>
<while_loop> ::= <WHILE> <expression> <DO> <LBRACE> <statement_list> <RBRACE>
<for_loop> ::= <FOR> <LP> <assignment_expression> <SEMICOLON> <expression> <SEMICOLON> <assignment_expression> <RP>
<DO> <LBRACE> <statement_list> <RBRACE>

<function_def> ::= <DEF> <VARIABLE> <LP> <variable_list> <RP> <DO> <LBRACE> <statement_list> <RETURN> <LP> <expression> <RP>
<RBRACE>
<variable_list> ::= <VARIABLE> <COMMA> <variable_list> | <VARIABLE> | <EMPTY>

<type_def> ::= <TYPE> <VARIABLE>

<expression> ::= <conditional_expression> | <assignment_expression>

<conditional_expression> ::= <conditional_expression> <condition_operator> <low_precedence_arithmetic_expression> | <LP>
<conditional_expression> <RP> | <BOOLEAN>

<condition_operator> ::= <LESSER> | <LARGER> | <LESSER_EQ> | <LARGER_EQ> | <EQUALS> | <NOT_EQUALS> | <AND> | <OR> |
<XOR>

<assignment_expression> ::= <type_def> <ASSIGNMENT> <expression> | <VARIABLE> <ASSIGNMENT> <expression> |
<low_precedence_arithmetic_expression>

<low_precedence_arithmetic_expression> ::=  <low_precedence_arithmetic_expression> <low_precedence_operator>
<high_precedence_arithmetic_expression> | <high_precedence_arithmetic_expression>
<high_precedence_arithmetic_expression> ::= <high_precedence_arithmetic_expression> <high_precedence_operator>
<low_precedence_arithmetic_expression> | <highest_precedence_arithmetic_expression>
<highest_precedence_arithmetic_expression> ::= <highest_precedence_operator> <highest_precedence_arithmetic_expression> |
<LP> <low_precedence_arithmetic_expression> <RP> | <value>

<low_precedence_operator> ::= <MINUS> | <PLUS>
<high_precedence_operator> ::= <DIVISION> | <MULTIPLICATION> | <MODULUS>
<highest_precedence_operator> ::= <NOT>

<value> ::= <VARIABLE> | <INTEGER> | <FLOAT> | <STRING> | <BOOLEAN> | <function_call>

<function_name> ::= <VARIABLE> | <PRIMITIVE_FUNCTION>

<function_call> ::= <function_name> <LP> <expression_list> <RP>
<expression_list> ::= <expression> <COMMA> <expression_list> | <expression> | <EMPTY>

<EMPTY> ::=
```

3. Construct Explanations

- **program**

This is the starting token for BNF description. It represents the whole file and it is described as the combination of functional statements and documentation comments used within code. Uses right recursion for combinations.

- **statement_list**

A list of functional statements inside the program.

- **statement**

A statement is the largest functional building element. They can consist of single lines or multiple lines. They are divided into two categories depending on their return value. They should always be closed using a semicolon.

- **comment_list**

A list of nonfunctional documentation comments inside the program.

- **void_statement**

A void statement is a statement that has no return value. It is a set of if statements, for and while loops, function definitions, variable type definitions and some reserved keywords. Those statements can't be used in places that require a value return.

- **if_statement**

An if statement is a void statement that uses decision making to run or not run another statement. They take value statements as input to decide whether the given statement will be run or not. They are always initiated by the if keyword. The end of the condition and start of the conditioned statement is initiated by the do keyword. Usage of else keyword is optional, in cases user would like to run a statement when the given value statement is negative. Conditional statements can be nested. The condition is accepted as positive (first given statement is run, else statement is not) when the value statement given as connection is anything but "false" (of Boolean type), 0 (of int type) or 0.0 (of float type).

- **loop**

Loops are used to run statements multiple times. There are two types of loops depending on their decision-making processes. Every kind of loop can be nested.

- **while_loop**

This kind of loop is used when a statement should run continuously as long as a given condition is met. When the statement is done and condition is positive, it will be run again from the top. Similarly to conditional statements, they take value statements as input to decide whether the given statement will be run or not. They are always initiated by the while keyword. The end of the condition and start of the conditioned statement is initiated by the do keyword. The condition is accepted as positive (given statement is run) when the value statement given as connection is anything but "false" (of Boolean type), 0 (of int type) or 0.0 (of float type).

- **for_loop**

This kind of loop is used when a more advanced control is needed upon the running conditions. Within for loop, a new variable can be defined, a running condition can be specified and a statement that will run after each iteration the inside statement finishes running can be implemented. Everything this statement does can be done using while loops, however for loops provide a more readable and organized structure for specific needs. They are always initiated by the for keyword. The end of the condition and start of the conditioned statement is initiated by the do keyword. The condition is accepted as positive (given statement is run) when the value statement given as connection is anything but "false" (of Boolean type), 0 (of int type) or 0.0 (of float type).

- **function_def**

A function definition is a void statement that assigns a variable name to a block of code so that it can be used multiple times throughout the program without repeatedly writing the same thing over and over. A function definition is initiated by the keyword "def", followed by the name of the function,

variable_list in parenthesis, the do keyword, followed by the function itself in curly braces. Within the function definition, the “return” keyword is used to return a value to the function caller.

- **variable_list**

  A variable list is the list of value names that should be passed to a function. It is required in order to declare a function.

- **variable_list**

  An expression list is the list of expressions that are passed to a function. It is required in order to use the previously declared function somewhere else in the program.

- **type_def**

  A type def is a definition of a variable and its type. It is written by a type in front of a variable name.

- **expression**

  Expression is the counterpart and the reverse of void statements. It is any statement that has a return value.

- **conditional_expression**

  Conditional expressions are equations that result in boolean values. They are used to compare the values, to check their equity, and for binary operations.

- **assignment_expression**

  Assignment expressions are used to assign values to variables. A variable should either have a previously declared type, or it needs to have the type declaration in the assignment expression. Assignment expressions return the assignment value. Left hand side has to be a variable while right hand side can be any expression.

- **low_precedence_arithmetic_expression**

  A low_precedence_arithmetic_expression is one of 2 things: either an arithmetic operation between another low precedence arithmetic expression and a high precedence arithmetic expression using a low precedence operator or it is a high precedence arithmetic expression. It is designed to result in left associativity following real life conventions.

- **high_precedence_arithmetic_expression**

  A high_precedence_arithmetic_expression is one of 2 things: either an arithmetic operation between another high precedence arithmetic expression and a highest precedence arithmetic expression using a high precedence operator or it is a highest precedence arithmetic expression. It is designed to result in left associativity following real life conventions.

- **highest_precedence_arithmetic_expression**

  A highest precedence arithmetic expression can be a highest precedence arithmetic expression acted on by an operator, a low precedence arithmetic expression in paranthesis, or a value.

- **low_precedence_operaton**

  A low precedence operator is a group of operators that have lower precedence in arithmetic, like addition and substraction operators.

- **high_precedence_operator**

  A high precedence operator is a group of operators that have higher precedence than the low precedence operators, like multiplication and division operators.

- **highest_precedence_operator**

  A highest precedence operator is a group of operators that have precedence over all other defined operators.

- **value**

  A value is anything that is returnable. It can be a literal, variable or a function call.

- **function_call**

  A function call is the format to invoke an already defined function. It is the combination of, first a function name and then an expression list filled with expressions matching the pre-defined parameter list.

- **function_name**

A function name is either a variable that was used in the definition of a function before or a primitive function which is a pre-defined function by the language. It has to be alphanumerical.

4. Token Definitions

- **COMMENT**

A comment is a terminal. They are non-functional and are meant to be used for documentation / explanation purposes of the code. Comments can only be initiated by the // symbol and they can't be longer than a line. They can consist of any character.

- **VARIABLE**

A variable is a name that stores data values. The information it holds can be changed or updated. They are declared with a type such as int, string, float. Type determines what type of data a variable hold. They must be alphanumerical. Reserved keywords or operator names can't be used as variable names.

- **TYPE**

A type defines the data set a variable can hold, and how the operations on that variable are performed. Examples: int, string, float…

- **INTEGER**

Integer is a set of values that represents all integers, e.g. 5, -7, 2 , 0…

- **FLOAT**

Float is a set of values that represents all floating point numbers, e.g. 3.1, -9.7 , 1.12335664…

- **STRING**

String is a set that represents a combination of characters and are enclosed within "", e.g. "egg", "apple", "happiness" … They can consist of any character.

- **BOOLEAN**

Boolean is a value representing either true or false.

- **Reserved Keywords (IF, ELSE, WHILE, FOR, DO, DEF, RETURN, BREAK, CONTINUE)**

These keywords are used special statements. Else can only be used after an If. Return can be only used inside a function. Break and Continue are only usable inside loops.

- **Operators (LESSER, LARGER, LESSER_EQ, LARGER_EQ, EQUALS, NOT_EQUALS, AND, OR, XOR)**

Standard operators are defined as such.