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++.

1. **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> ::=

1. **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 variables that are passed to a function. It is required in order to declare a function, and in order to use the created 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. Expression is any statement that returns a value.

* **conditional\_expression**

Conditional expressions are equations that result in boolean values.

* **assignment\_expression**
* **low\_precedence\_arithmetic\_expression**

def is a

* **high\_precedence\_arithmetic\_expression**

def is a

* **highest\_precedence\_arithmetic\_expression**

def is a

* **low\_precedence\_operaton**

def is a

* **high\_precedence\_operator**

def is a

* **highest\_precedence\_operator**

def is a

* **value**

A value is anything that is returned

* **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.

1. **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.

* **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, which determines what type of data a variable holds.

* **TYPE**

A type defines the data a variable holds, 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”…

* **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 for loops and functions.

* **Operators (LESSER, LARGER, LESSER\_EQ, LARGER\_EQ, EQUALS, NOT\_EQUALS, AND, OR, XOR)**

Standard operators are defined as such.