BILKENT UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

# CS 315
# PROJECT 1
# REPORT

**Mehmet Ali Altuntaş - 21401004**
**Damla Eda Bıçakcı - 21402130**
**İrem Yurdakul - 21400299**

**CONTENTS**

Our language's name is CSL, and in this report, we explain CSL's syntax and its descriptions. Lex file and example programs are also included.

## Part A - Language Design

**Language Name:** CSL

**Language Design in BNF form:**

**Non-terminals:**

<prog>-> BEGIN <stat_list> END
<stat_list> -> <stmt> | <stat_list> <stmt>
<stmt> -> <if_expr> |<stmt>
<stmt_e> -> <assign> | <loops> | <fnc> | <in_out_stmt> | <expr>

Truth Values:
<vals> -> <true>|<false>
<true> -> true | TRUE
<false>->false |FALSE

Constants:
<const_type> -> const

Variables:
<char> -> [A-Za-z]
<numb> -> [-+]?[0-9]+
<id> -> {char}+[0-9]*
<ids> -> <ids><comma><id>
|<id>
<type> -> "int" | "bool" | "char" | "string" | "void"
<var> -> <type> <id>
|<id>
<vars>-><vars><comma><var>
|<var>
<fcall> -> f_

Assignment Operator:
<assing_op> -> =
<assign> -> <var> <assing_op> <numb>
|<var> <assing_op> <vals>
|<var> <assing_op> <expr>
|<var> <assing_op> <id>

Connectives:
<expr> -><log_expr> | <mat_expr>
<log_expr> -> <log_expr> && < log_term>

```
            | < log_expr > | | < log_term >
            | < log_expr >  ->  < log_term >
            | < log_expr >  <->  < log_term >
            | < log_term>
            |~(<log_term>)
<log_term > -> (<log_expr>)
            | <id>
            |~<id>
<mat_expr> -> < mat_expr > ++< mat_term >
             |< mat_expr > --< mat_term >
            |<mat_term>
<mat_term> -> <mat_term>**<math_factor>
            |<mat_term> //<math_factor>
             |<math_factor>
<math_factor>-> (<mat_expr>)
         |<id>
```

<u>Predicates - Predicate Instantiations:</u>

```
<fnc>-> <predicate>| <predicate_instantiations>
<predicate> - > <type><fcall><id>(<vars>)
<predicate_instantiations> -> <fcall ><id>(<id>)
```

<u>Selection Statements:</u>

```
<if_stmt> -> <matched>|<unmatched>
<matched>-> if<logic_expr>then <matched>else <matched>
             | if<mat_expr>then <matched>else <matched>
             |<stmt_e>
<unmatched> -> if<logic_expr>then <stmt>
                    | if<mat_expr>then<stmt>
                    | if<logic_expr>then <matched>else <unmatched>
                   | if<mat_expr>then <matched>else <unmatched>
```

<u>Loops:</u>

```
<loops> -> <while_stat>
<while_stat> -> while <expr> <stmt>
```

<u>Input/Output Statements:</u>

```
<in_out_stmt> -> <inp> | <out>
<inp> -> inp <id>
       | inp <expr>
       | inp <numb>
       |inp<var>
<out> -> out<id>
       |out<expr>
       |out<numb>
       |out<var>
```

**Terminals:**

( : Left parenthesis
) :Right parenthesis.
|| :“Or” logic term.
&&:“And ” logic term.
-> : “Implication” logic term
<-> : “Equivalence” logic term
; : “Semicolon” sign
:: “Colon” sign
++: “Plus” sign
--: “Minus” sign
**: “Multiplication” sign
//: “Division” sign
,: “Comma” sign
_:”Underscore” operator
~: “Negation operator
==: “Equal”operator

**BNF Descriptions:**

**<prog> :** It checks the program if it starts or ends with “BEGIN” and “END” keywords.
**<stat_list> :** It makes possible to write multiple code segments many times.
**<stmt> :** It determines statements of the program among assign loop function if and expression opeartions.
**<assign> :** It works for assign a value to another value**.**
**<assign_op> :** It determines “=” symbol
**<loops> :** It determines loops in our language.
**<while_stat> :** It determines while loop.
**<fnc> :** It determines predicate functions in our language which are going to be used for calling a function and defining a function.
**<in_out_stmt> :** It determines the both input and out statement in the language.
**<expr> :** It determines <log_expr> and <mat_expr> as an expression.
**<log_expr> :** It determines logical expressions in the language.
**<log_term> :** It determines the logical values in the language.
**<id> :** It determines the single terminals in our language, like A,b, ab1 etc.
**<mat_expr> :** It determines mathematical operations in the language.
**<mat_term> :** It defines the terms mathematical values in the language.
**<math_factor> :** It defines the precedence of mathematical operations both “**” and “//” .
**<type>** : It defines the type of the variable which are “int”, “string”, “char”, “bool” and “void”.
**<pred> :** It defines how the header of a function is. The input format is:
Return_Type f_functionName ( parameter, parameter..)
e.g. int f_multiplier ( int a, int b)
**<predInst> :** it defines how a function is called. The input format is:
f_functionName ( varName, varName...)
e.g. f_multiplier ( a, b)

**<if_stmt> :** It determines the both mathced and unmatched if statements in the language.
**<in_out_stmt> :** It defines the input and out statements.
**<inp> :** It defines the inputs in the language.
**<out> :** It defines the outputs in the language.

## Part B - Lexical Analysis

```
%option main
numb [+-]?[0-9]+
char [A-Za-z]
id {char}+[0-9]*
type "int"|"char"|"string"|"void"|"bool"
const_type "const"|"CONST"
ids {id}{spc}*{comma}{spc}*
assign "="
spc " "
var {type}{spc}*{id}
vars {var}{spc}*{comma}{spc}*
lp "("
rp ")"
fcall "f"{alt}
alt "_"
comma ","
tru "true"|"TRUE"
fal "false"|"FALSE"
plus "++"
minus "--"
mul "**"
div "//"
and "&&"
or "||"
implication "->"
equivalence "<->"
neg "~"
equal "=="
colon ":"
semicolon ";"
greater ">"
smaller "<"
gr_eq ">="
sm_eq "<="
if "if"
then "then"
else "else"
while "while"
inp "inp"
out "out"
pred {type}{spc}*{fcall}{id}{spc}*{lp}{spc}*{vars}*{var}{spc}*{rp}
```

```
predInst {fcall}{id}{spc}*{lp}{spc}*{ids}*{id}{spc}*{rp}
%%
BEGIN printf(“BEGIN”);
END printf(“END”);
{lp} printf("LEFT_PARA");
{rp} printf("RIGHT_PARA");
{inp} printf("INP_STRM");
{out} printf("OUT_STRM");
{if} printf("IF");
{else} printf("ELSE");
{then} printf("THEN");
{while} printf("WHILE");
{const_type} printf("CONST_TYPE");
{tru} printf("TRUE");
{fal} printf("FALSE");
{numb} printf("NUMBER");
{var} printf("VARIABLE");
{pred} printf("PREDIFIER_FUNC");
{predInst} printf ("PREDIFIER_INST_FCALL");
{id} printf("ID");
{assign} printf("ASSIGN_OP");
{comma} printf("COMMA");
{colon} printf("COLON");
{semicolon} printf(" SEMICOLON");
{greater} printf("GREATER_OP");
{smaller} printf("SMALLER_OP");
{gr_eq} printf("GREATER_EQUAL");
{sm_eq} printf("SMALLER_EQUAL");
{plus} printf("PLUS");
{minus} printf("MINUS");
{mul} printf("MULTIPLIER");
{div} printf("DIVISION");
{and} printf("AND");
{or} printf("OR");
{equivalence} printf("EQUIVALENCE");
{implication} printf("IMPLICATION");
{equal} printf("IS_EQUAL");
{neg} printf("NEGATION");
```

## Part C - Example Programs

**TEST 1**

```
BEGIN
const bool a = true;
int k = 5;
if (y == false) then y = y ** 5 else y = 0;
while (j == 5) then f = 5 // 4
```

```
d =( a || b )&& c
t = a + s
qwe -> dwqe
r = ~(e <-> tr)
asd = ~asd
void f_AD(char a, string r, int k)
f_AD(j,y);
inp 56
out var
END
```

**Output:**

```
BEGIN
CONST_TYPE VARIABLE ASSIGN_OP TRUE SEMICOLON
VARIABLE ASSIGN_OP NUMBER SEMICOLON
IF LEFT_PARAID IS_EQUAL FALSERIGHT_PARA THEN ID ASSIGN_OP ID MULTIPLIER
NUMBER ELSE ID ASSIGN_OP NUMBER SEMICOLON
WHILE LEFT_PARAID IS_EQUAL NUMBERRIGHT_PARA THEN ID ASSIGN_OP NUMBER
DIVISION NUMBER
ID ASSIGN_OPLEFT_PARA ID OR ID RIGHT_PARAAND ID
ID ASSIGN_OP ID + ID
ID IMPLICATION ID
ID ASSIGN_OP NEGATIONLEFT_PARAID EQUIVALENCE IDRIGHT_PARA
ID ASSIGN_OP NEGATIONID
PREDIFIER_FUNC
PREDIFIER_INST_FCALL SEMICOLON
INP_STRM NUMBER
OUT_STRM ID
END
```

**TEST 2**

```
BEGIN
bool A1 = 0;
bool B1 = 1;
int num1 = 5;
const digit num2 = 6;
int result = num1 + num2;
int f_foo (string s, digit k);
while ( A1 && B2 == FALSE);
if (result >= 10);
A1 = ~A1;
int f_foo (sum,num2);
END
```

**Output:**

```
BEGIN
VARIABLE ASSIGN_OP NUMBER SEMICOLON
VARIABLE ASSIGN_OP NUMBER SEMICOLON
VARIABLE ASSIGN_OP NUMBER SEMICOLON
CONST_TYPE ID ID ASSIGN_OP NUMBER SEMICOLON
VARIABLE ASSIGN_OP ID + ID SEMICOLON
VARIABLE_ID LEFT_PARAVARIABLECOMMA ID IDRIGHT_PARA SEMICOLON
WHILE LEFT_PARA ID AND ID IS_EQUAL FALSERIGHT_PARA SEMICOLON
IF LEFT_PARAID GREATER_EQUAL NUMBERRIGHT_PARA SEMICOLON
ID ASSIGN_OP NEGATIONID SEMICOLON
VARIABLE_ID LEFT_PARAIDCOMMAIDRIGHT_PARA SEMICOLON
END
```

## Conclusion

In our language, our main goal is being easy to usage of the language. Our language allows user, flexibility and writability. User can create and assign logical and mathematical operations easily, he/she can determine values with their types and proper names. In the usage of mathematical expressions, predicate functionality for multiplication and division operation is also user friendly functionalities. By using function calls, loops and selection expression, useful programs can be created. When the user determines a value, language forces user to determine a proper type. Therefore, it allows our language reliability. However, while readability is increasing to keep the format certain we strict the predicate declarations and this cause the decrease in flexibility. And lastly, spaces during the writing of a code is can cause contradictions. That's why the “spc” definition help us on that point. It is beneficial while declaring function prototypes, function declarations, multiple variable declarations and “id” initializations.