# Simplified Basic-C Compiler

The project developed parses and calculates statements written in a C-like fashion. It can initialize variables of type bool and int, compute the 4 basic arithmetic operations plus the power operation, and perform comparison of variables. Lastly, the parser does type-checking on its primitive data-types.

Although, scoping and control-flow structures have not been fully implemented (a stack structure is necessary in order to compute the correct execution branch at the right moment) it can parse them correctly.

Instead of declaring precedence rules we have developed our grammar in an unambiguous fashion, so that the operations with a higher precedence are computed closer to the leaf level in the parse tree. By doing so we have that a multiplication is computed before the addition but after a unary operation such as the negative value of a variable.

## Program

This is how a program in our Basic-C Compiler can start.

```
$user:flex Analyzer.l
$user:yacc -vd Parser.y
$user:gcc y.tab.c -ly -ll -o compiler.out
$user:./compiler<input.txt
```

- Or if make is installed

```
$user:make
$user:./compiler<input.txt
```

- It can also receive input interactively like so:

```
$user:./compiler
```

### Valid input

```
int x : 10;
int y : 20;

y: y + 1; // prints 21;

int k : y + x;
print k; // prints 31;

bool t : true;
bool f : false;
print f; // prints 0

if(t){print f;}

if(x > y) {return true;} else { return y;} // prints "This is the else branch executed;"
while(t)do{print f;} // prints "The while loop should execute here"
return;
```

### Invalid input

- Invalid variable declaration and initialization

```
int k = true;
int x : 10;
bool t : true;
```

- The following are invalid operations

```
x : 5 + t;
bool result : (5 or 5);
int res : (true + true);
print (12 and true;)
```

- Missing return statement

```
int x : 2;
int y : 3;

print (x + y);
```

## The grammar of Basic-C

```
program -> program stmt | ε

stmt -> varDeclInit ;
        | simpleExp ;
        | if (simpleExp) {stmt}
        | if (simpleExp) {stmt} else {stmt}
        | while (simpleExp) do  {stmt}
        | return ;
        | return simpleExp ;
        | print simpleExp ;

varDeclInit -> typeSpec varDeclId simpleExp | varDeclId : simpleExp

varDeclId -> ID

typeSpec -> int | bool

simpleExp -> simpleExp or andExp | andExp

andExp -> andExp and unaryRelExp | unaryRelExp

unaryRelExp -> not unaryRelExp | relExp

relExp -> sumExp relOp sumExp | sumExp

relOp -> <= | < | >| >= | == | !=

sumExp ->  sumExp sumOp mulExp | mulExp

sumOp -> + | -

mulExp -> mulExp mulOp unaryExp | unaryExp

mulOp -> * | / | **

unaryExp -> - unaryExp | NUM | FALSE | TRUE | ID | (simpleExp)
```