1(a) **(10 marks) Using the `<if-statement>` as a basis, extend this to include the `<while-statement>`. Enclose the stuff inside the loop with braces (` { } `). For now, you may assume that `<boolean-expression>` has been defined.**

```
<while-statement> ::= "while" "("<boolean-expression>"){"
          <statements>
"}"
```

1(b) (10 marks) Using `and`, `or`, and `not` as the terminals for the logical primitives define the BNF for `<boolean-expression>`. You only need to implement logical inclusive 'or' and 'and'. i.e. implement only the equivalent of &&, ||, and ! from C. You may assume <boolean> ::= "True" | "False". You may also assume that for this language, 'and' and 'or' have the same precedence.

```
<boolean-expression> ::=
          <boolean>
          |"not" <boolean-expression>
          | <boolean-AND-expression>
          | <boolean-OR-expression>


<boolean-AND-expression> ::=  <boolean-expression> “and” <boolean-expression>
<boolean-OR-expression> ::= <boolean-expression> “or” <boolean-expression>
```

1(c). (10 marks) Your boss is an oldschool `LISP` programmer and a huge fan of the Polish notation. He wants this new language to use the Polish notation for arithmetic expressions (much to your chagrin). Extend the definition for `<arithmeticexpression>`

(`+,,*,/`) with Polish notation in BNF. (see <https://en.wikipedia.org/wiki/Polish_notation> for more information about Polish notation). You may assume that all arithmetic operations are binary and, for this question, assume that variables are not used in arithmetic expressions. i.e. an arithmetic expression should only have numbers, arithmetic operations.

Also note that you may have to define additional non terminals. As an example, this is a statement produced using <statement> ::= <id> "=" <arithmetic-expression>

n = + 4 + 5 * 2 3;
Its equivalent in C would be int n = 2 * 3 + 5 + 4;

```
<arithmetic-operator> ::= "+" |"-"  |"*" |"/"
<arithmetic-expression> ::=
           <arithmetic-operator>  <arithmetic-expression> <arithmetic-expression>
          |<arithmetic-operator>  <number> <arithmetic-expression>
          |<arithmetic-operator>  <arithmetic-expression> <number>
          |<arithmetic-operator>  <number> <number>
```

1d. extend the BNF so that "<arithmetic-expression>" can use predefined variables.
Vince:

```
<arithmetic-operator> ::= "+" |"-"  |"*" |"/"
<arithmetic-expression> ::=
           <arithmetic-operator>  <arithmetic-expression> <arithmetic-expression>
          |<arithmetic-operator>  <val> <arithmetic-expression>
          |<arithmetic-operator>  <arithmetic-expression> <val>
          |<arithmetic-operator>  <val> <val>
<val> ::= <id> | <number>
```

1e.As a proof of concept, your boss wants you to write a program in this new language to print out the numbers between 1 and 1000 (inclusive) that are divisible by 7 or 13 but not both. Assume ==, > etc are implemented but not %

```
start
          i = 1;
          nextFactor7 = 7;
          nextFactor13 = 13;
          while (i <= 1000){
                    if(  i == nextFactor7 ) then
                              nextFactor7 = + nextFactor7 7;
                              if( i == nextFactor13) then
                                        nextFactor13 = + nextFactor13 13;
                              else
                                        print i;
                              fi
                    else
                              if( i == nextFactor13) then
                                        print i;
                                        nextFactor13 = + nextFactor13 13;
```

```
                        else
                                void;
                        fi

                fi
                i = + i 1;
        }
end
```

Question 2 - 4

see Question2_4.pl