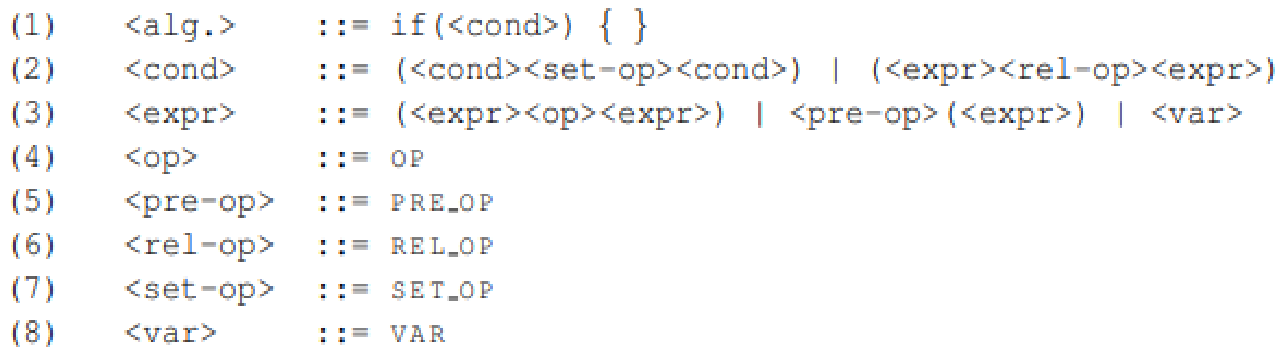
ASSIGNMENT 4 REPORT

PROBLEM DEFINITION

In computer science, BackusNaur Form, or BNF, is one of the main notation techniques often used to describe the syntax of the languages, such as computer programming languages, document formats, instruction sets, communication protocols, and the like. A BNF consists of:

- a set of terminal symbols,
- a set of non-terminal symbols,
- a set of production rules.

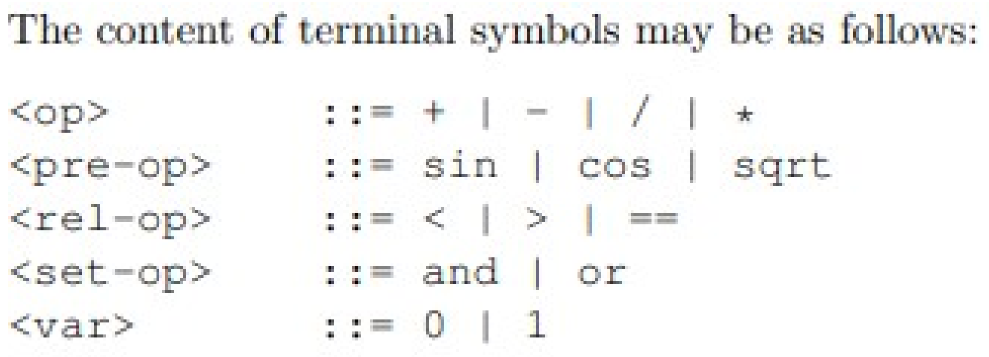
In this assignment firstly generate a BNF-Tree considering the given symbols and production rule, and print the BNF-Tree in a C -style. The production rule for BNF-Tree is given below;



```
(1)   <alg.>     ::= if(<cond>) { }
(2)   <cond>     ::= (<cond><set-op><cond>) | (<expr><rel-op><expr>)
(3)   <expr>     ::= (<expr><op><expr>) | <pre-op>(<expr>) | <var>
(4)   <op>       ::= OP
(5)   <pre-op>   ::= PRE_OP
(6)   <rel-op>   ::= REL_OP
(7)   <set-op>   ::= SET_OP
(8)   <var>      ::= VAR
```

Figure 1: Production rule for BNF-Tree.

Op, pre-op, rel-op, set-op and var are the non-terminal symbols which are shared with us. This files contain some operands and operators ;



The content of terminal symbols may be as follows:

```
<op>          ::= + | - | / | *
<pre-op>      ::= sin | cos | sqrt
<rel-op>      ::= < | > | ==
<set-op>      ::= and | or
<var>         ::= 0 | 1
```

By using information in above, tree may will be like that;

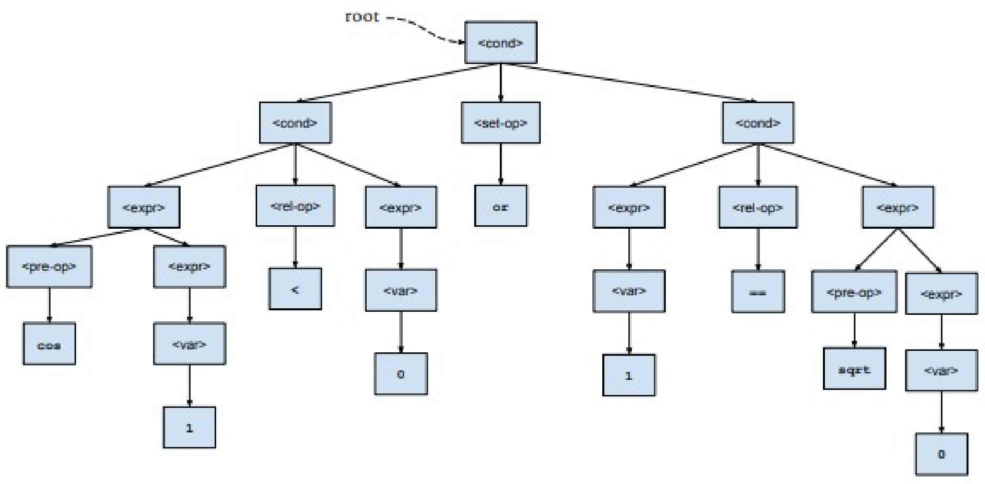


Figure 3: An example BNF-Tree

Finally, we must print contain of non-terminal symbols of tree. For example;

if (((cos(1))<(0))or((1)==(sqrt(0)))) {}

SOLUTIONS AND FUNCTIONS

At first I describe struct of all node type; zero_child, one_child, two_child, three_child. This structs contain name of structs e.g expr, cond, op . . . etc. Then child's or children's void pointers and integer variable named kids_number which this pointer how many child have below.

char *readFileContent(char *filename)

This function returns random word from given file.

random_number(int max)

This function take one parameter which max number to give random number among zero to max.

void *cond_eval(struct three_child *cond)

This function take three child struct which named cond and create child structs of that.

void *three_child_expr_eval(struct three_child *expr)

This function take three child struct which named expr. And create that three children.

void *two_child_expr_eval(struct two_child *expr)

This function take two child struct which named expr. And create that two children.

void *one_child_expr_eval(struct one_child *expr)

This function take one child struct which named expr. And create that child.

void *op_eval(struct one_child *op)

This function take one child struct which named op. Then create that child type of zero_child which has random contain of op.

void *pre_op_eval(struct one_child *pre_op)

This function take one child struct which named pre_op. Then create that child type of zero_child which has random contain of pre_op.

void *rel_op_eval(struct one_child *rel_op)

This function take one child struct which named rel_op. Then create that child type of zero_child which has random contain of rel_op.

void *set_op_eval(struct one_child *set_op)

This function take one child struct which named set_op. Then create that child type of zero_child which has random contain of set_op.

void *var_eval(struct one_child *var)

This function take one child struct which named var. Then create that child type of zero_child which has random contain of var.

void *three_child_print_tree(struct three_child *cond)

This function take three child struct to reach child structs.

void *two_child_print_tree(struct two_child *cond)

This function take two child struct to reach child structs.

void *one_child_print_tree(struct one_child *cond)

This function take one child struct to reach zero child struct to print contain of that.