ASSIGNMENT4

Subject: **Tree Implementation**

Advisor: ***Res. Assist. Selim YILMAZ***

Due Date: ***19.01.2020 – 23:59:59***

Student: ***M ucahit Veli CUMART -- 2160589***

INTRODUCTION

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.*

Left hand side of a production rule represents a non-terminal symbols where right-hand side is a sequence of symbols including terminal or non-terminal. A BNF grammar example for a phone number like (123)456-7890 (or like 123-4567) is given below:

<phone-number> -> <area-code><7-dig>|<7-dig>

<area-code> -> "("<digit><digit><digit>")"

<7-dig> -> <digit><digit><digit>"-"<digit><digit><digit><digit>

<digit> -> "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"

where all items between angle brackets are the non-terminal symbols and others stated within

“.” are the terminals representing the end items. Note that all non-terminal symbols should be expanded until they are parsed into the terminal symbols.

BNF-TREE

(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> ::= O P

(5) <pre-op> ::= P R E O P

(6) <rel-op> ::= R E L O P

(7) <set-op> ::= S E T O P

(8) <var> ::= V A R

The content of terminal symbols may be as follows:

<op> ::= + | - | / | *

<pre-op> ::= sin | cos | sqrt

<rel-op> ::= < | > | ==

<set-op> ::= and | or

<var> ::= 0 | 1

As you can deduce from the production rule that BNF-Tree must always be rooted at <cond> and that <expr>, for instance, can only be generated by <cond>. In addition to that, the rule points out that a node in the BNF-Tree can have zero, one, two, or three offspring.

In this assignment, there are four structs for operation,one child,two childs,three childs

STRUCTURES of THIS ASSIGNMENT

```
typedef struct{
    bool var;
    char *operator;
}Node;
typedef struct{
    char type[15];
    void *ptr;
}Node1;
typedef struct{
    struct {
        char type1[15];
        char type2[15];
    }Data_type;
    void *ptr1;
    void *ptr2;
}Node2;
typedef struct{
    void *ptr1;
    void *ptr2;
    void *ptr3;
    struct {
        char type1[15];
        char type2[15];
        char type3[15];
    }Data_type;

}Node3;
```

FUNCTIONS of THIS ASSIGNMENT

```
Node1 *generate_op(char *type_of,char *filename)
```

This function creates that the nodes which has one child and assigns their operation names.

```
void display_Tree(void *root,char *root_type)
```

This function prints the tree's elements with recursion.

```
void expression_genarate(void **expression,char *type_of_expression)
```

This function generates childs of the tree randomly with recursion.

```
void condition_generate(void **cond,char *type_of_expression)
```

This function generates the childs of the 'Expressions' with recursion.