BBM203 Programming Assignment 4

Ayşe Balcı

January, 2020

1-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. I am expected to generate a BNF-Tree considering the given symbols and production rule and print the BNF-Tree in a C -style. The problems I faced while implementing the system are putting together op, pre_op, set_op, rel_op, var information and apply tree structure.

2-Solution Approach: I created tree, I used 4 different nodes for this.

3-C language features used and functions:

Structs for tree;

```
struct node0{
    char *data;
    int countChild;
};

struct node1{
    char *data;
    int countChild;
    void *child1;
};
```

```
struct node2{
    char *data;
    int countChild;
    void *child1;
    void *child2;
};

struct node3{
    char *data;
    int countChild;
    void *child1;
    void *child2;
    void *child3;
};
```

Functions:

```
int findLengthOfFile(char *file);   //Function for calculating length of file

int findLineCount(char *fileString, char *file, int size);  //Function for
calculating count of file

char** allocateAndFillArray(char*stringArray, int lineCount, int size);
//Function for allocating memory

node0* createNode0(char *value);  //function creates new node with 0 child by
given char* value

node1* createNode1(char *value, int op_count, char **op_array, int pre_op_count,
char **pre_op_array, int rel_op_count, char **rel_op_array, int set_op_count,
char **set_op_array, int var_count, char **var_array);  //function creates new
node with 1 child by given char* value

node2* createNode2(char *value, int op_count, char **op_array, int pre_op_count,
char **pre_op_array, int rel_op_count, char **rel_op_array, int set_op_count,
char **set_op_array, int var_count, char **var_array);   //function creates new
node with 2 child by given char* value
```

```
node3* createNode3(char *value, int op_count, char **op_array, int pre_op_count,
char **pre_op_array, int rel_op_count, char **rel_op_array, int set_op_count,
char **set_op_array, int var_count, char **var_array);   //function creates new
node with 3 child by given char* value

int randomNumber(int lower, int upper,int count);  //function returns integer
number between lower count and upper count

void createCondition(node3* root, int op_count, char **op_array, int
pre_op_count, char **pre_op_array, int rel_op_count, char **rel_op_array, int
set_op_count, char **set_op_array, int var_count, char **var_array);  //if root
<cond>, it calculate random number 0 or 1. if number is 0: Childs: <cond><set-
op><cond>, if number is 1: Childs:<expr><rel-op><expr>

void call3ChildNodes(node3* root, int op_count, char **op_array, int
pre_op_count, char **pre_op_array, int rel_op_count, char **rel_op_array, int
set_op_count, char **set_op_array, int var_count, char **var_array);  //if root
is <cond> return this function

void createExpressionNode3(node3* root, int op_count, char **op_array, int
pre_op_count, char **pre_op_array, int rel_op_count, char **rel_op_array, int
set_op_count, char **set_op_array, int var_count, char **var_array); //if
expression node is 3, expr childs are <expr><op><expr>

void createExpressionNode2(node2* root, int op_count, char **op_array, int
pre_op_count, char **pre_op_array, int rel_op_count, char **rel_op_array, int
set_op_count, char **set_op_array, int var_count, char **var_array);  //if
expression node is 2, expr childs are <pre-op>(<expr>

void createExpressionNode1(node1* root, int op_count, char **op_array, int
pre_op_count, char **pre_op_array, int rel_op_count, char **rel_op_array, int
set_op_count, char **set_op_array, int var_count, char **var_array);  //if
expression node is 1, expr child is <var>

void print1(node3* root);  //function take node3 root from main function, print2
or print3, then if child's type node3, return yourself, if child's type node2
return print2, if child's type node1 return print1

void print2(node2* root);  //function take node3 root from print2 or print3, then
if child's type node3, return print2, if child's type node2 return yourself, if
child's type node1 return print1

void print3(node1* root);  //function send own child to print4

void print4(node0* root);  //function writes data

int findCountChildOfNode3(node3* root);  //function return node3 root'child type

int findCountChildOfNode2(node2* root);  //function return node2 root'child type

int findCountChildOfNode1(node1* root);  //function return node1 root'child type
```