BBM203 PROGRAMMING LAB. - REPORT
Backus-Naur Form

Yusuf Emre Genç

21803663

1 Problem Definition

In this assignment, I am expected to implement BackusNaur Form that is a notation technique in computer science. The main idea behind my program is to implement tree structure. I will focus on implementation and design of my program in this report.

2 Methods and Solution

I am expected to use four structs for each possible type of branching. Therefore, my solution is not a poor design. You can see these types in Figure 1. Also you can see the structs that I defined in Figure 2, 3, 4 and 5.

The "option" variable denotes the option of branching our non-terminal node.
The "symbol" variable denotes the type of node (Cond, Expr...).
I stored children in a two dimensional void pointer which is denoted by "children" variable.
Lastly the "branchingOptionOfChildren" variable stores the option of branching of children of each child.

My program creates a tree that represents "if statements" that is randomly fed from "op", "pre op", "rel op", "set op", "var" files.

I also used recursion while printing the tree structure. My code is also dynamic so that you can change the content of files and my program will generate new statements with new content. Also there will not be any possible leaks in my program.

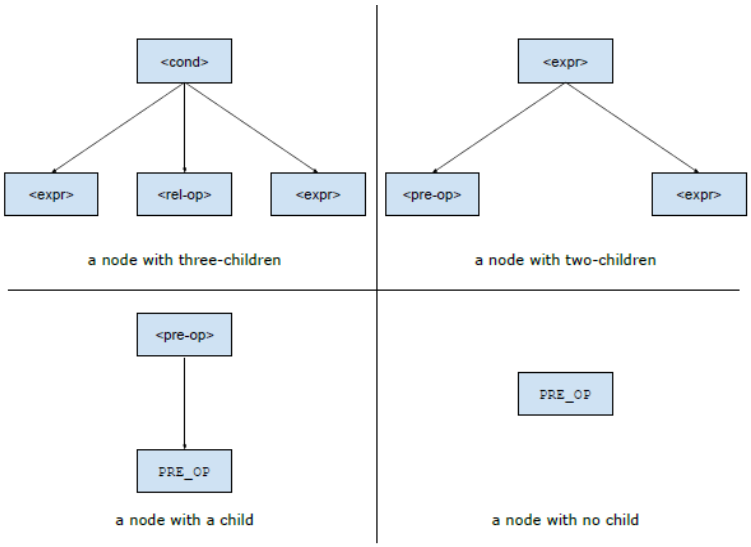

Figure 1: Node Types

```
struct Node0{
    Symbol symbol;
    char *data;
};
```

Figure 2: Caption

```
struct Node1{
    int option;
    Symbol symbol;
    void **child;
    int* branchingOptionOfChildren;
};
```

Figure 3: Caption

```
struct Node2{
    int option;
    Symbol symbol;
    void **children;
    int* branchingOptionOfChildren;
};
```

Figure 4: Caption

```
struct Node3{
    int option;
    Symbol symbol;
    void **children;
    int* branchingOptionOfChildren;
};
```

Figure 5: Caption

3 Functions

There are three main functions in my program.

3.1 branching

This function creates our tree recursively and randomly. It needs parent node as parameter. The base case of recursive function is terminal nodes. It generates random numbers for non-terminal nodes and creates children of parent node according to generated random option. For terminal nodes, it does not create any children, but it stores the data to parent node and terminates recursive function.

3.2 print

This is a recursive function that prints the tree. Again, its base case terminal nodes. When root parameter is a non-terminal root, it puts necessary parentheses and calls itself. But otherwise, the function prints the data that is stored in root and terminates the recursion.

3.3 deallocateTree

This is also a recursive function and it deallocates the buffer for all nodes.