ASSIGNMENT IV

Subject : Trees
Advisor : Res. Assist. Selim YILMAZ
Due Date : 19.01.2019 — 23:59:59

Backus-Naur Form (BNF)

1 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.

2 The Task: Generation of a BNF-Tree

In this assignment you are highly expected to i) generate a BNF-Tree considering the given symbols and production rule, and ii) print the BNF-Tree in a *C*-style. Please read the following explanations carefully to achieve the goals.

The production rule for BNF-Tree is given in Fig. 1 so that you are able to generate the tree. In the figure; `<alg.>`, `<cond>`, `<expr>`, `<op>`, `<pre-op>`, `<rel-op>`, `<set-op>`, and `<var>` are the non-terminal symbols which are to be expanded; whereas OP, PRE_OP, REL_OP, SET_OP, and VAR indicate the names of the files, which are shared with you, containing the terminal items of interest. It is worth mentioning here that even if the production rule given

in the figure will not be modified, but the content and the size of the terminal items may vary in this assignment.

```
(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.

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 (refer to Fig. 2).

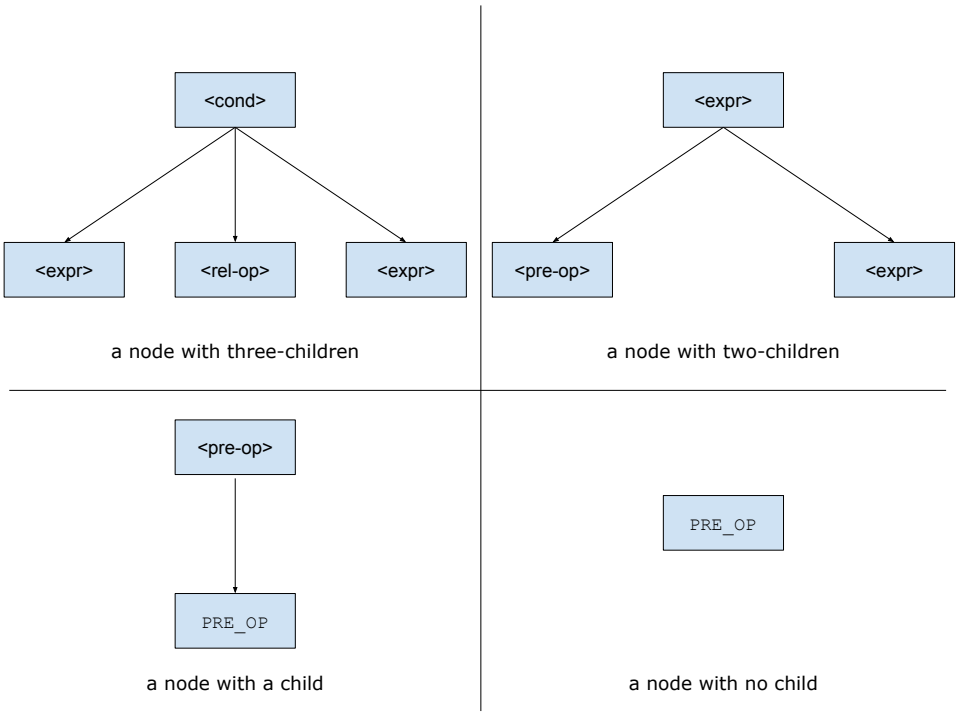

Figure 2: Node types

An exemplar to a BNF-Tree that can be generated upon the production rule above is given in Fig. 3.

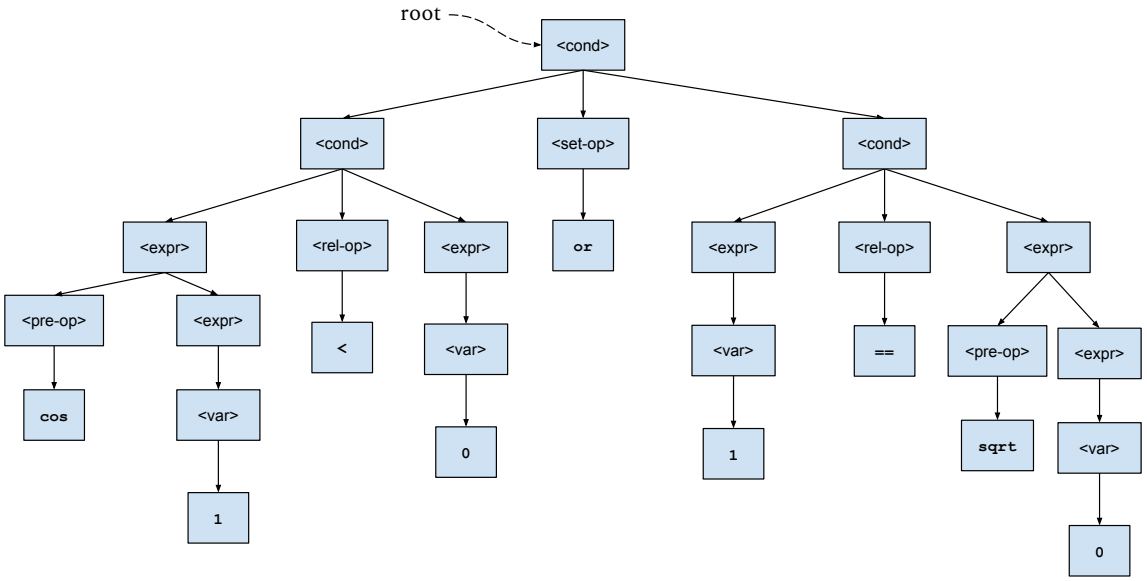

Figure 3: An example BNF-Tree

Printing of BNF-Tree given in Fig. 3 is given below. It is worth mentioning here that printing should be a separate process in your implementation, which points out that you are not allowed to print BNF-Tree while constructing it. In addition to that parenthesis marks are not necessarily be stored in BNF-Tree but should be placed appropriately while printing.

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

Design Notes & Constraints

- **Playing in a tricky way:** You are NOT allowed to develop a 'poor' solution such as the scenario where only a type of node with four children is defined and used in the tree, which is illustrated in Fig. 4.

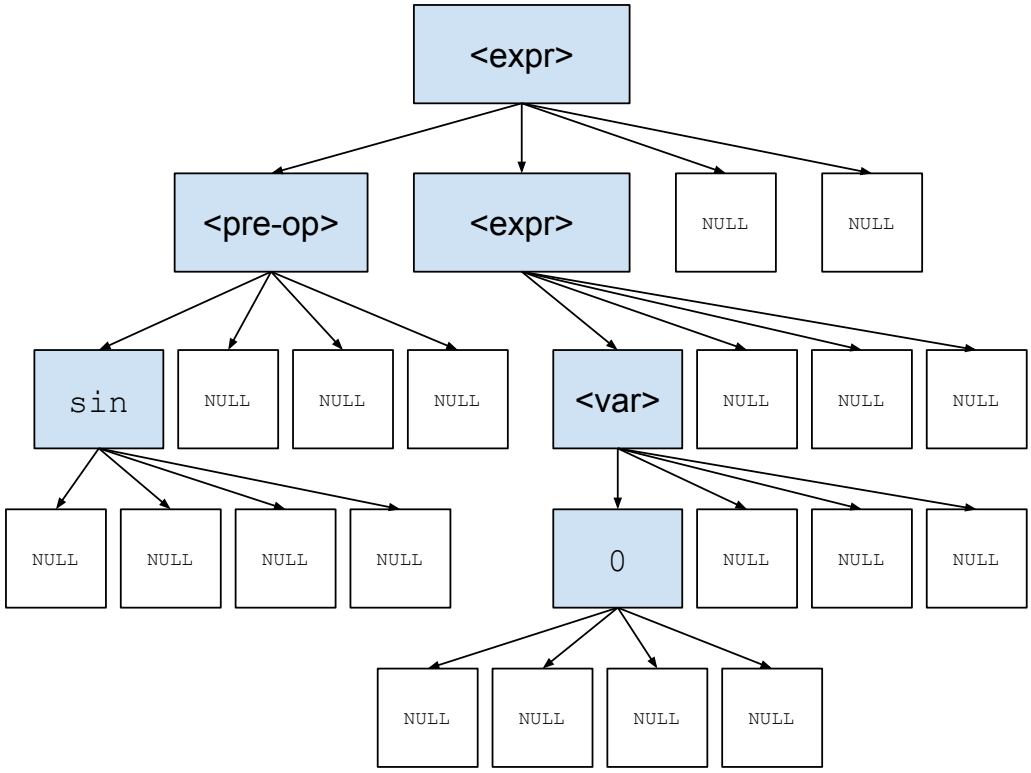

Figure 4: A 'poor' design.

- **Item size in terminal nodes:** Feel free to suppose that total number of items in OP, PRE_OP, REL_OP, SET_OP, and VAR cannot exceed the value of 10.
- **Linked-list representation:** You must implement a LINKED-LIST to construct BNF-Tree, which should point only the root node of the tree.
- **Recursion:** You should print BNF-Tree in RECURSIVE manner, which is easier way to do that than iterative approach.
- **Writing into a file:** DO NOT write anything into any file; instead print BNF-Tree on the SCREEN.
- **A moderate response time:** Your implementation is expected to respond in an admissible time.

- **Dynamic implementation:** As stated earlier, even if the production rule will not be modified while evaluating your works, it is not the case for the terminal items. Therefore your implementation should adopt any change in these items. In addition, do not further improve the production rule on your own; however, you are highly encouraged to do so after submission by introducing, for instance, new non-terminal and terminal items like the one given below, which enables new type expressions such as `mod(3,5)`.

```
(3)    <expr>     ::= (<expr><op><expr>) | <pre-op>(<expr>) | <var>
                     | <pre-op>(<expr>) | <pre-op-2>(<expr>,<expr>)
                     | <var>
(4)    <op>       ::= OP
(5)    <pre-op>   ::= PRE_OP
(5)    <pre-op-2>::= min | max | mod | pow
(6)    <rel-op>   ::= REL_OP
(7)    <set-op>   ::= SET_OP
(8)    <var>      ::= VAR
```

- **Different outcome:** Your implementation is able to produce DIFFERENT ALGORITHM every time it is run.

3 Grading Policy

Task	Score^{+}
BNF-Tree generation through linked list*	**20**
Node types in BNF-Tree*	**20**
Printing in *C*-style*	**10**
Printing through recursion*	**10**
Random output	**10**
Adaptability to changing environment	**5**
Response time	**5**
Code readability	**5**
Report	**15**

* it is a must task without which your work is excluded from evaluation!

$$^{+}Final\ Score = Exec.\ Score + \left(Report\ Score \times \frac{Exec.\ Score}{85} \right)$$

Notes

- Do not miss the deadline.
- Save all your work **until the assignment is graded**.
- The assignment must be **original**, **individual work**. Duplicate or very similar assignments are both going to be considered as cheating.
- Regardless of the length, use **understandable** names to your variables, functions, and etc.
- Should you have a question, feel free to ask on Piazza. Be aware that the question has not been asked/discussed before.
- Do not submit your works via e-mail.
- You will submit your work from https://submit.cs.hacettepe.edu.tr/index.php with the file hierarchy as below:

This file hierarchy must be zipped before submission (Not .rar, only .zip files are supported by the system)

→ **<student id>.zip**
  → **src.zip**[1]
    → *.c
    → *.h
  → Makefile
  → report.pdf

[1] may contain subfolder(s)