*Ho Chi Minh University of Technology*

*Faculty of Computer Science an Engineering - Computer Science Major*

*Course: Discrete Structure for Computing*

--------------------o0o--------------------



**HK212**

**Assignment Report  
Co- Lecturer**: Nguyễn Tiến Thịnh - Nguyễn Văn Minh Mẫn

**Class**: CC01 - **Group**: 3

|  |
| --- |
| Team member |

|  |  |  |
| --- | --- | --- |
| **No.** | **Name** | **Student ID** |
| 1 | Vũ Châu Duy Quang | 2153730 |
| 2 | La Cẩm Huy | 2153379 |
| 3 | Đỗ Lâm Ngọc Thức | 2153143 |
| 4 | Nguyễn Vĩnh Huy | 2152597 |

Catalog

[I. Theories Recall 3](#_Toc23310)

[1. Mathematical Theories: 3](#_Toc12098)

[2. Computing Theories: 3](#_Toc6574)

[II. Idea and execution 4](#_Toc10428)

[1. Problems: 4](#_Toc17999)

[2. Idea: 4](#_Toc11347)

[3. Execution: 5](#_Toc8506)

[4. Explanation: 6](#_Toc25629)

[III. Result 10](#_Toc13150)

[1. P1a 10](#_Toc18383)

[2. P1b 10](#_Toc26130)

[3. P1c 11](#_Toc4593)

[4. P2a 11](#_Toc28169)

[5. P2b 11](#_Toc8085)

[6. P2c 11](#_Toc19112)

[IV. Conclusion 12](#_Toc5814)

[1. Advantage: 12](#_Toc25959)

[2. Disadvantage 12](#_Toc20659)

[3. Final conclusion 12](#_Toc30948)

# Theories Recall

## Mathematical Theories:

### Prefix notation:

A parenthesis-free notation for forming mathematical expressions in which each operator precedes operands.

Ex: 6+5 can be written in Prefix notation: + 6 5

### Postfix notation:

A parenthesis-free notation for forming mathematical expressions in which each operator follows its operands.

Ex: 7\*2 can be written in Post fix notation: 7 2 \*

* **Infix notation:**

A parenthesis-free notation for forming mathematical expressions in which each operator is in between its operands. This is also know as the way we normally use to express functions or calculations.

## Computing Theories:

* **Graph theory:**

Graph theory is the study of graphs, which are mathematical structures used to model

pairwise relations between objects.

* **Tree - Binary Tree:**

Tree structure or a tree is an undirected graph in which any two vertices are connected by exactly one path

A binary tree is know as a tree data structure in which each node has at most two children,

which are often called as the left child and the right child.

# **Idea and execution**

1. **Problems:**

P1: Write a function receive a constant string which is:

1. An infix arithmetic notation expression. Return another string which is it’s prefix notation
2. An infix arithmetic notation expression. Return another string which is it’s postfix notation
3. A prefix or postfix arithmetic notation expression. Return the value of the expression.

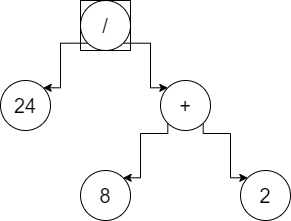
P1: Write a function receive a constant string which is:

1. An infix logical notation expression. Return another string which is it’s prefix notation
2. An infix logical notation expression. Return another string which is it’s postfix notation
3. A prefix or postfix logical notation expression. Return the value of the expression.
4. **Idea:**

The main idea to solve the problems listed above is to use a binary tree structure to store the expression. Then use the algorithm provided in the problem (PreOrder Tree Traversal, PostOrder Tree Traversal and Postfix Evaluation) to transform or evaluate the expression given which is converted to binary tree above.

Recall that an operator only have 1 or 2 operands (this is true for both arithmetic and logical expression) which is perfectly fit with the binary tree definition (The operator will be parents and its operands or smaller expression will be the child node).

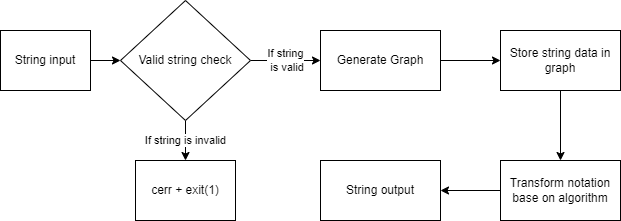
Ex: 24/(8+2) can be store in a binary tree like this:



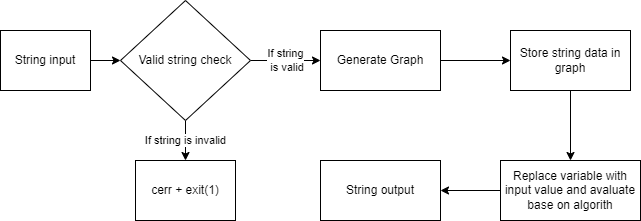
1. **Execution:**

Knowing the main idea to solve the problems, we come up with a simple yet efficient way to handle it.

Flow chart for transform infix notation to others notation: (P1a, 1b,2a, 2b)



Flow chart for calculate the value of notation:

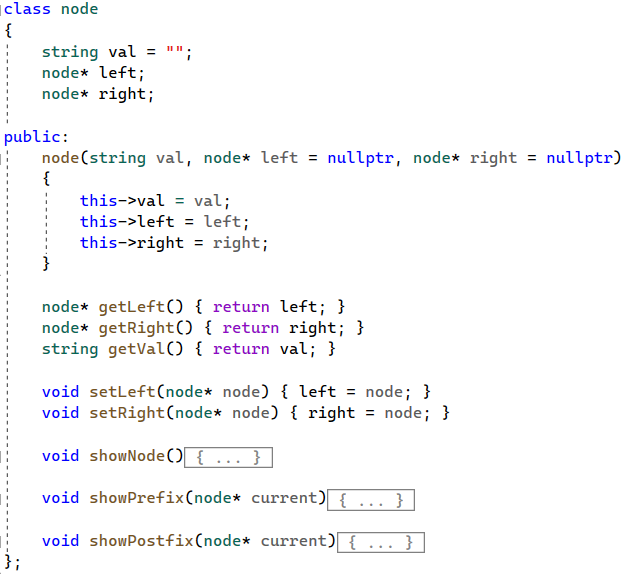


\*\*Note: Since logical variable only have 2 type of input value (either 1 for true and 0 for false), we must build a function in other to check for valid input of these variable in order to evaluate.

Looking at the flowcharts, it can be seen that building function base on usage would cost less time, especially function that can be reused in all of the 6 problems , such as graph generate function or valid input check function. So we decided to start working that way in order to save time and reduce the amount of work we need to do.

Also, with the current COVID-19 situation in Sai Gon, we decided that it would be best to work and communicate over GitHub instead of direct meeting, and it turn out to be a great idea.

1. **Explanation:**

We will first build a class named "node" that represents each node of the tree. Includes 1 "val" containing the value of that node (Can be a sign or number) and 2 nodes pointing to the left and right nodes "left" and "right" as well as the number of functions to access the next node and some functions to show the tree.

We will first build a class named "node" that represents each node of the tree. Includes 1 "val" containing the value of that node (Can be a sign or number) and 2 nodes pointing to the left and right nodes "left" and "right" as well as the number of functions to access the next node and some functions to show the tree

Next I will build a function "FindingChar" with the parameters "subleft", "subright", "charlist", "precharList" and "expression" to find the last operator appearing in the string according to the input parameter. .

- The operator we need to find will be stored in "charList" and "precharList" is used to pass in the signs before "charList" because we will use this function to build a tree for both logical expression and mathematical expression

- First, we need to ignore the operators in the parathesis because the operators will be calculated first, so they will be built at the bottom of the tree, so we will consider the last.

- After finding an operator that meets the conditions, we will build it into a node of the tree and return to use it

- Then split the string string into 2 substrings, left substring and right substring and stored in "subleft", "subright"

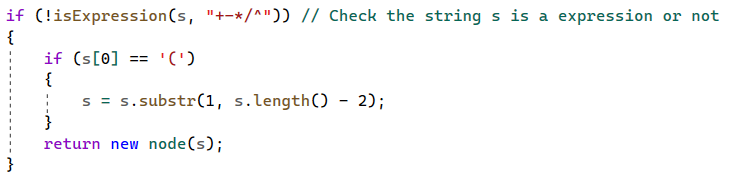
IMG_257

Next we will build a new function "Analysis\_V1" to build a complete tree for a given expression.

- This function will perform expression processing recursively:

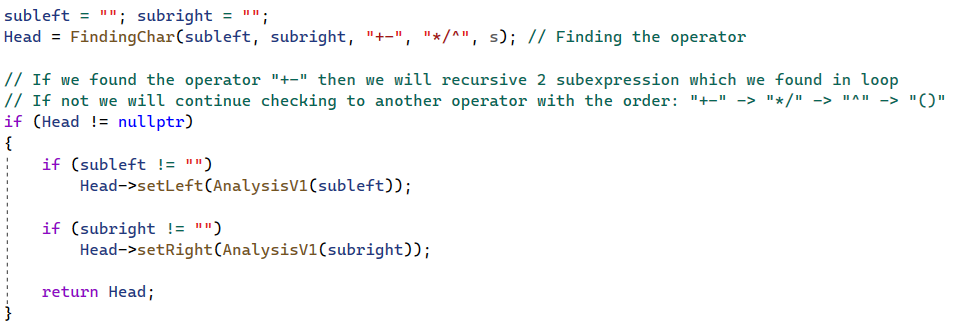
+ We will first check if an expression passed is an expression (Contains both operant and operator) or contains only operant

+ If it is only operant, it will no longer be an expression and will be added as a leaf of the tree

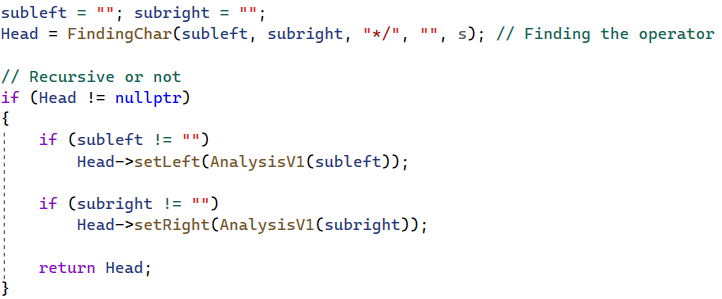


+ If it is still an expression, we will continue to split and split the function according to the following rule:

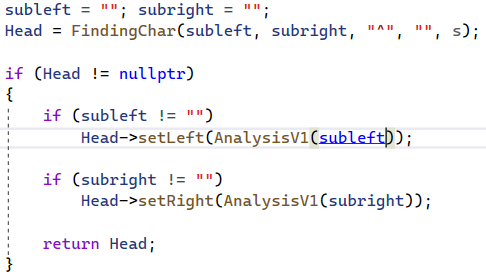
- First look for the +- signs. If found, recursively the 2 substrings that have just been split



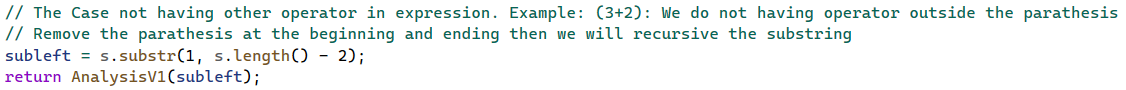
- If we can't find it, we will go to the next step to search for \*/. If found, recursively the 2 substrings that have just been split



- If you can't find it again, the next step is to find the ^ sign. If found, recursively the 2 substrings that have just been split

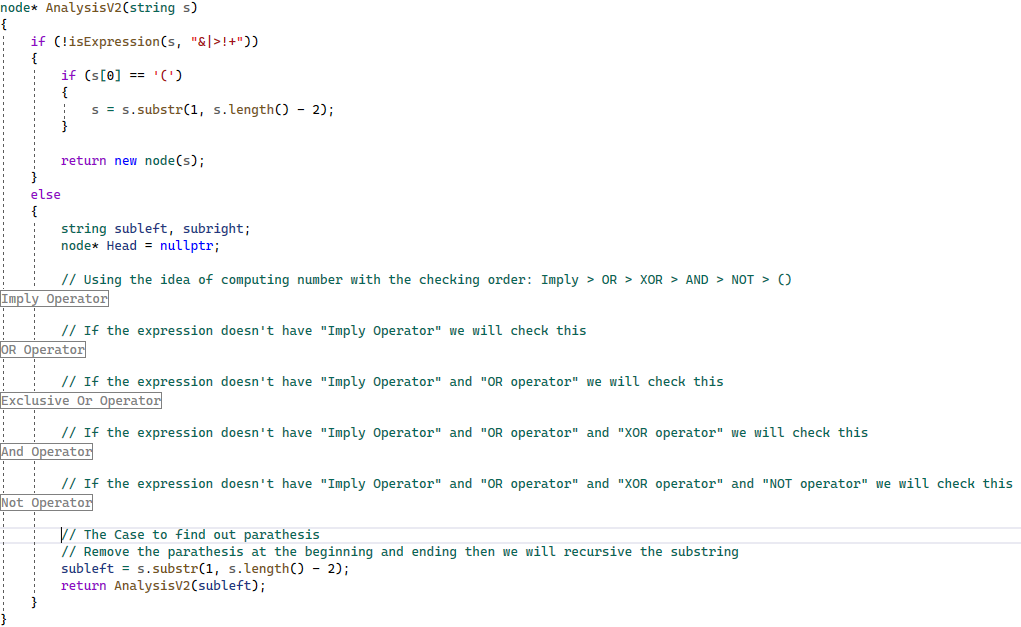


- Otherwise, we only have the case "(expression)" (Ex: (3+5)). So we will remove 2 parathesis signs and recursively substring that



- When we finish we will return the root of the tree.

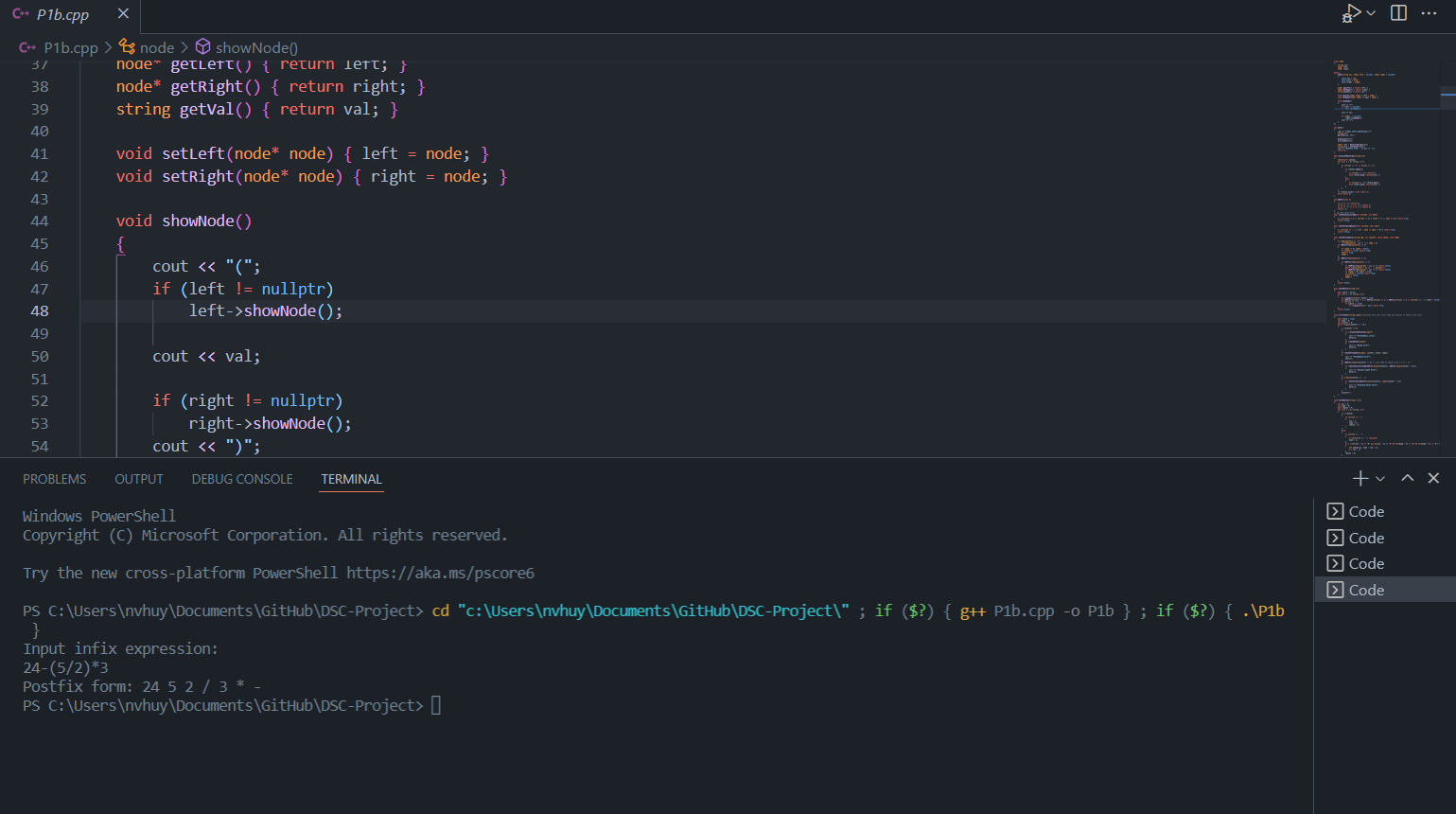
\* For expression that is a logical expression, we still follow the method of mathematical expression with the order of consideration being Imply > OR > Exclusive Or > And > Not > Parathesis



# Result

## P1a

## P1b

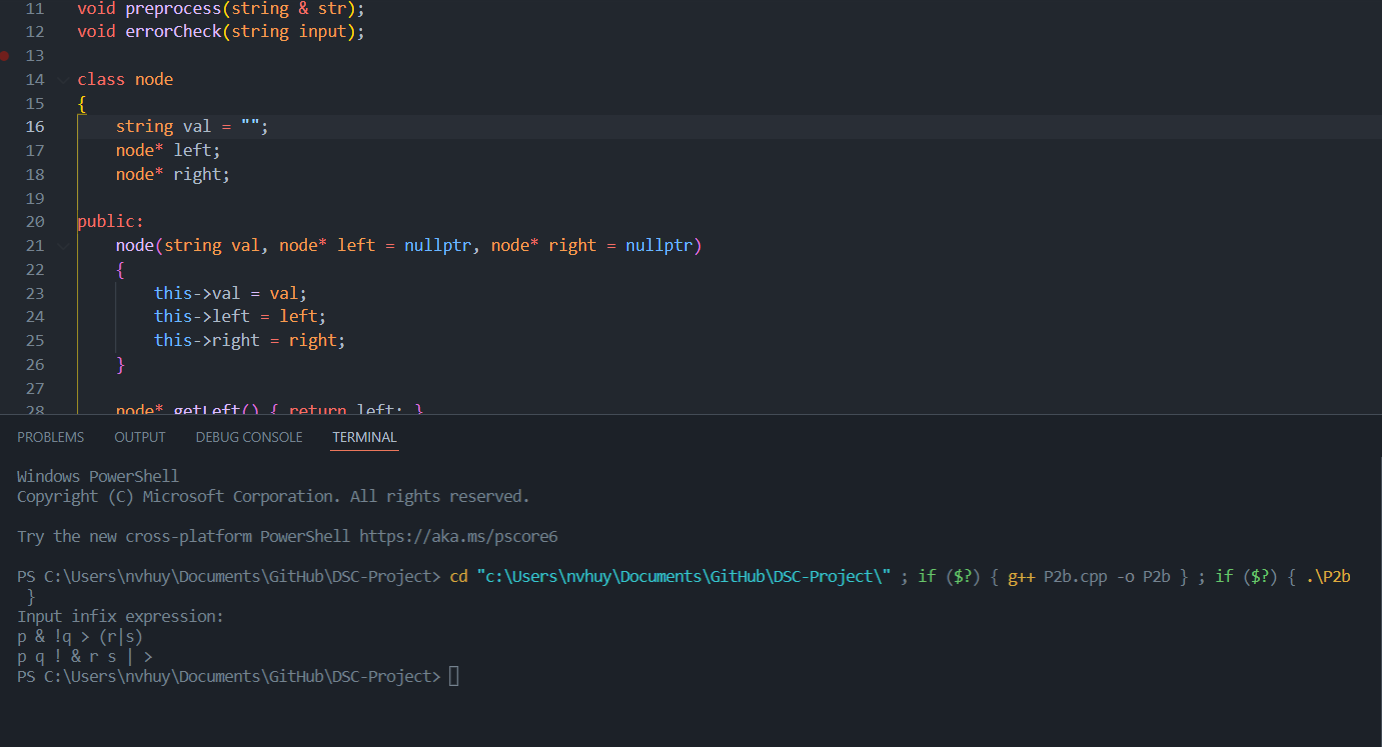


## P1c

## IMG_256IMG_256P2a

## 

## P2b



## IMG_256P2c (\*\*Note: we declare “p” for true value (1) and “q” for false value (0))

# **Conclusion**

Finalizing the solution, we have a look back in the process and see that our work have some advantage and disadvantage we want to share about

## **Advantage:**

* Building function base on usage and reusability, which can reduce work and time spent
* Remote working and communication via GitHub
* Decent use of graph and tree, especially binary tree in solution

## **Disadvantage**

* Program is not fully debugged due to lack of time and tight schedules (We need to work for other assignments and for final exam as well)
* Lack direct communication (most of our communication is via facebook and GitHub), but we think that this work just fine)

## **Final conclusion**

Although tree and graph are strong data structure and can be use to solve complex problems, it is indeed hard and time-consuming trying to fully understand and apply these structure in real life programming, especially with the lack of time. However, we been working and researching about the problem for a while, and we think that we have a decent work and gain a great amount knowledge about this.