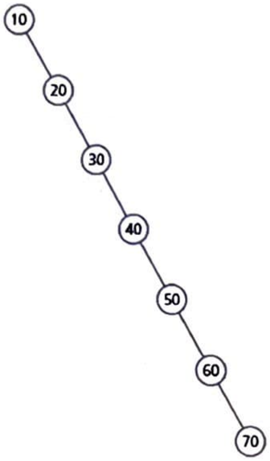
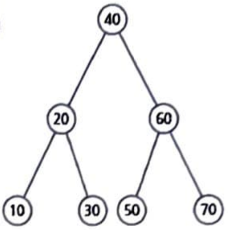
1. If the two binary search trees shown below are storing data, which do you suppose would provide for quicker and more efficient usage (to, for example, find out if a specific number is or is not stored in the data structure). Explain why, in detail.

(a)



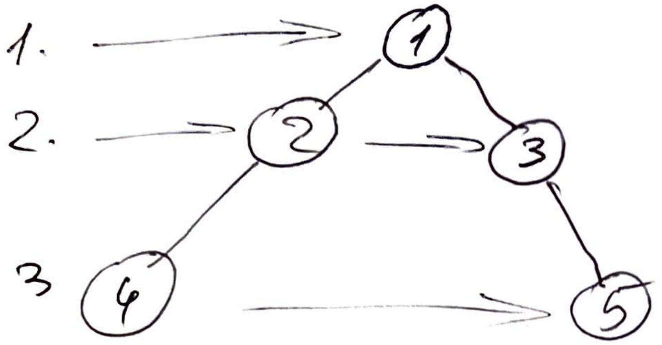
(b)



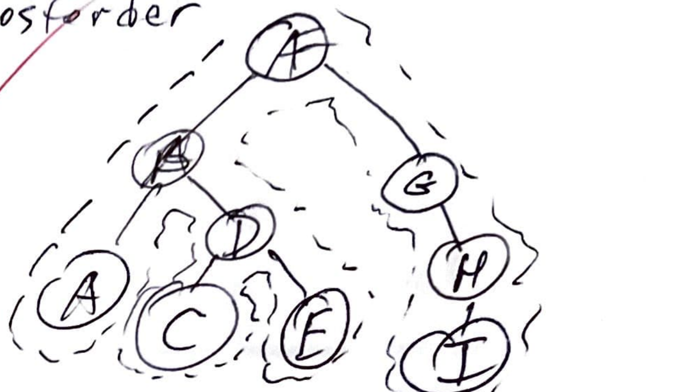
It is answer b, because it has smaller height then a, plus it Binary search tree and it use binary search, unlike a use linear search

2 Explain what "**Depth-First**" traversal of Tree data structure means, and what "**Breadth-First**" traversal means. **Use a diagram of a tree** of your own making to show your point. **For extra points also include a code fragment that gives the general idea of what steps will be taken in the code of each traversal type.**

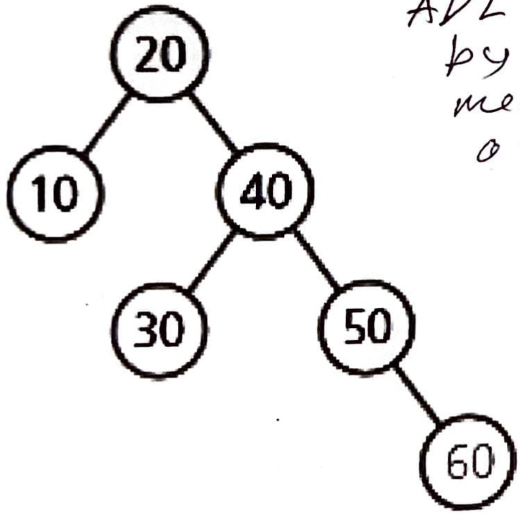
Breadth-First use level approach (it traverse throug all nodes in same level and then go to one below.



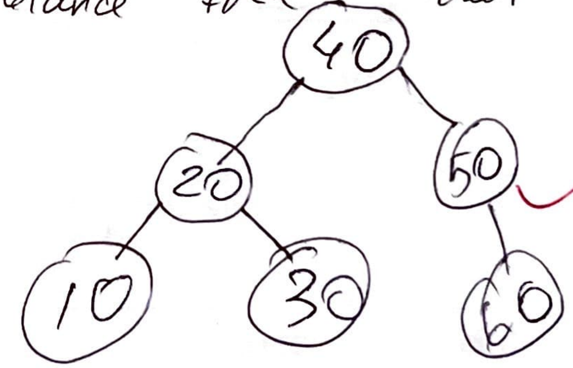
Deapth-First use approuch of going all the way to bottom of the tree by 3 traversal: inorder, preorde posterder



3. Explain how an AVL tree's "balancing" would "fix" the tree below by "rotation" by showing what the new version of the tree would look like (Hint: with its new root).



AVL tree balancing was founded by people, who's acrongm was named after. It based on principle of balance tree, that left subtre cannot be more than 1 height then right



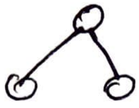
4. Explain why/how the new representation of the Binary Tree Data structure that you just drew on the right, above (hopefully!) . . . is "better".

Now it is balanced tree left subtree comply that is no more than 1 height difference than right subtree. This tree is balance (it has some height) It faster way to find specified value. It has same height

5. Define and explain in words or examples - your choice (or both words & pictures)

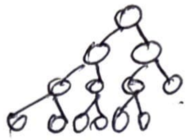
"Full binary tree"

Root has 2 child (left and right child)
no missing nodes



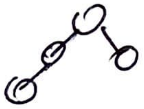
"Complete binary tree"

is always smaller then full binary tree, because it is n-1 and it starts to fill from left to right



"Balanced binary tree"

- it has to be no more than 1 height difference between left subtree and right subtree



```
// FUNCTION A
MysteryTraversal(binTree: BinaryTree): void
if (binTree is not empty)
{
 MysteryTraversal(Left subtree of binTree's root)
 Visit the root of binTree
 MysteryTraversal(Right subtree of binTree's root)
}
```

```
// FUNCTION B
MysteryTraversal(binTree: BinaryTree): void
if (binTree is not empty)
{
 MysteryTraversal(Left subtree of binTree's root)
 MysteryTraversal(Right subtree of binTree's root)
 Visit the root of binTree
}
```

6. Which algorithm above (Function A or Function B) represents code for:

a **POSTORDER** traversal of a binary tree of type BinaryTree?

Function B

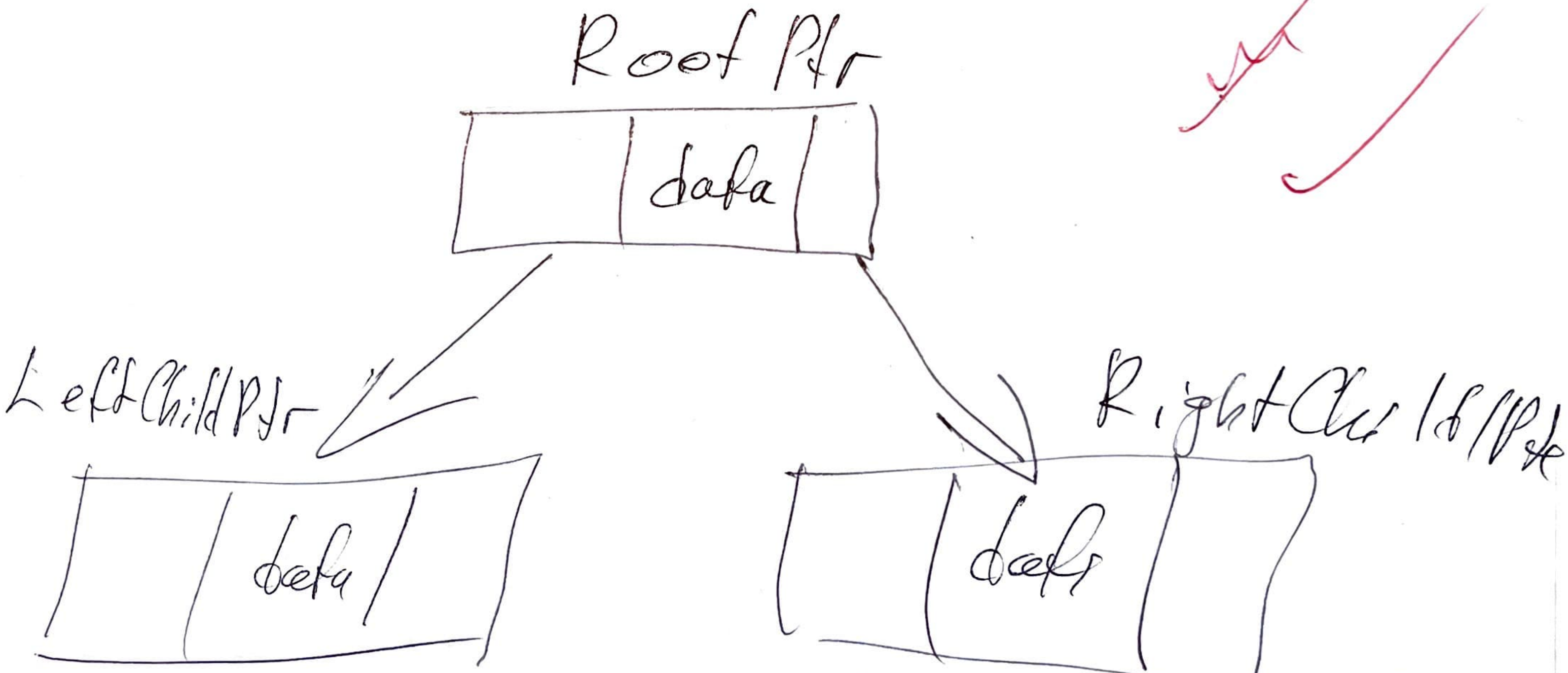
an **INORDER** traversal of a binary tree of type BinaryTree?

Function A

7. Your job is to specify the ADT (Abstract Data Type) for a Binary Tree Data Structure. Show what you would specify (describe the data, and name and describe at least 8 functions you might include). Also, show what its UML might look like.

UML

+isEmpty(); ← check if binary tree is Empt
+getHeightOFTree(); ← get height of B.T.
+getNumberOfNodes(); – get all the nodes
+ getRoot(): – get root of B.T.
+ setRoot(); – set data to root.
+add(); – add node to the tree
+remove(); – remove node from tree,
+inorder(); – travers in order;
+preorder(); – travers preorder
+ postorder(); – travers postorder.



Binary Tree consist of nodes, root, edges. Node consist of 3 parts: left cell point to address to left child, 2 cell is data, and 3rd is pointing to right child

8. Explain and demonstrate with code, diagrams and tables, how one could implement a representation of a Binary Tree of data **in an array**.

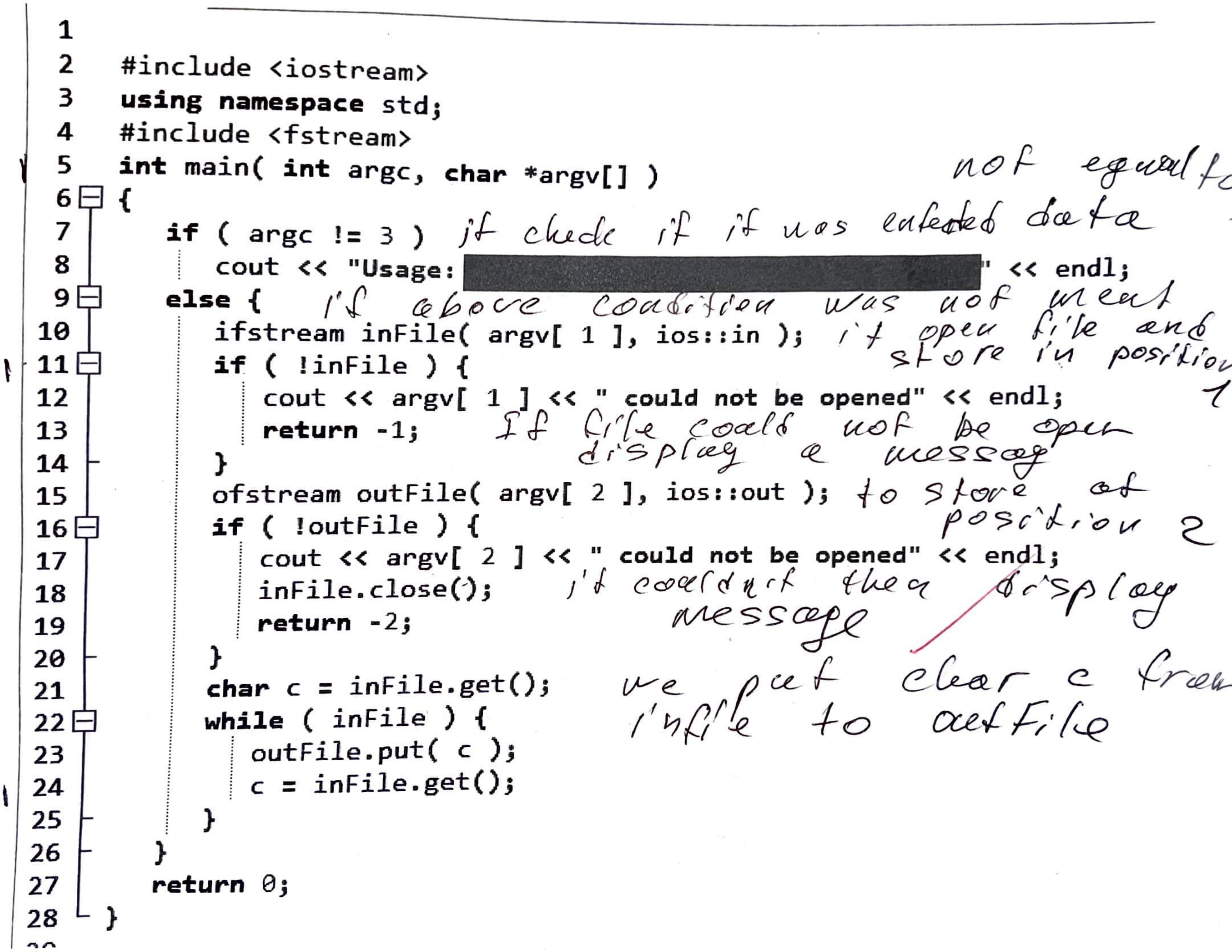
C ++ code

-3

9. For the code below:
In line 5, what are "**argc**" and "**argv**" and what are they used for?

argc - counts how many was entered
argv - stores what been entered in argc in ar

Then, write some comments to the right of the code, explaining what this program is doing at various lines of code, to describe what is going on in particular sections and lines.



```
1
2   #include <iostream>
3   using namespace std;
4   #include <fstream>
5   int main( int argc, char *argv[] )                          not equal to
6   {
7       if ( argc != 3 )   it check if it was entered data
8           cout << "Usage: [redacted]" << endl;
9       else {    if above condition was not meet
10          ifstream inFile( argv[ 1 ], ios::in );   it open file and
                                                       store in position 1
11          if ( !inFile ) {
12              cout << argv[ 1 ] << " could not be opened" << endl;
13              return -1;        If file could not be open
14          }                         display a message
15          ofstream outFile( argv[ 2 ], ios::out );  to store at
16          if ( !outFile ) {                          position 2
17              cout << argv[ 2 ] << " could not be opened" << endl;
18              inFile.close();      if couldn't then display
19              return -2;               message
20          }
21          char c = inFile.get();      we put clear c from
22          while ( inFile ) {           infile to outFile
23              outFile.put( c );
24              c = inFile.get();
25          }
26      }
27      return 0;
28  }
```

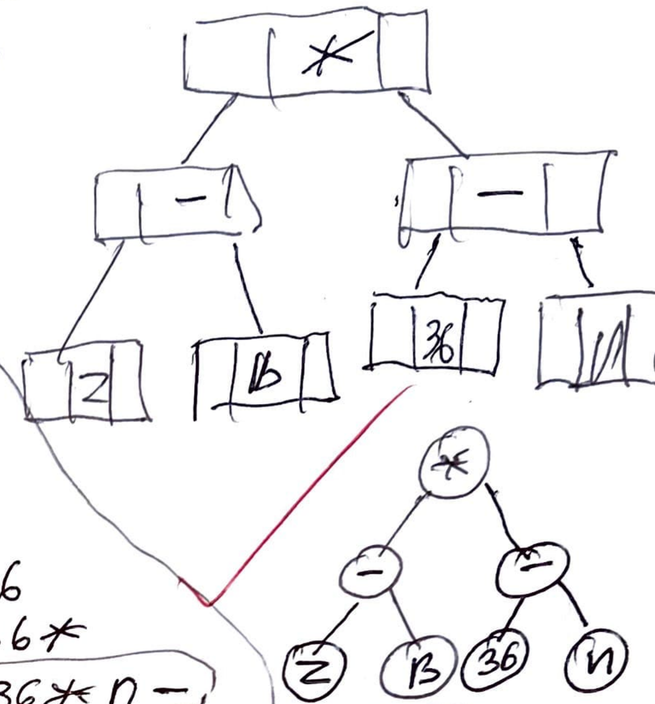
10a]. Convert from INFIX to POSTFIX: num * (Z - B) * 36 – n

10b]. Illustrate by (tree) diagram how the expression above can be stored in a tree data structure.

Show all nodes and pointers in your diagram

a)

	Stack	output
num		num
*	*	
(	*(	
Z		num, Z
–	*(–	
B		num Z B
)	*	num Z B –
*	*	num Z B – *
36		num Z B – * 36
–	–	num Z B – * 36*
n		num Z B – * 36* n –



11. (Generally) what are the phases/steps involved in the COMPILER taking Source Code and producing Executable Object Code?

Source Code → Prepposecor → Compiler → Assembler – Loaber → Machin Code

Analyzing Phase | Generating Phase

Source program | Lexi'call Analyse | syntex Analyse | Semantic Analyso | Intermediate code Generator | code opti-mi-zer | code ge-ne-ra-tor | targ Machine code

12a]. You have a C++ compiler for IBM Hardware and you want to adapt it to Apple Hardware - What part of the Compiler needs most of the work? backend

12b]. You have C++ compiler for IBM Hardware and you would like to write a JAVA compiler for the same Hardware - What part of the Compiler needs most of the work? front end

13. The algorithm at the top of this page is implemented in the code at the bottom of the page. Write a short comment next to key lines from 20 to 40 that indicate which step of the algorithm is done in that line of code.

Converting Expression from Infix to Postfix using STACK

To convert an expression from infix to postfix, we are going to use a stack.

Algorithm

1) Examine the next element in the input.
2) If it is an operand, output it.
3) If it is opening parenthesis, push it on stack.
4) If it is an operator, then
i) If stack is empty, push operator on stack.
ii) If the top of the stack is opening parenthesis, push operator on stack.
iii) If it has higher priority than the top of stack, push operator on stack.
iv) Else pop the operator from the stack and output it, repeat step 4.
5) If it is a closing parenthesis, pop operators from the stack and output them until an opening parenthesis is encountered. pop and discard the opening parenthesis.
6) If there is more input go to step 1
7) If there is no more input, unstack the remaining operators to output.

```
1   #include <iostream>
2   #include <sstream>
3   #include <stack>
4   #include <limits>
5   #include <string>
6   using namespace std;
7   int priority(char a) {
8     int temp;
9     if (a == '^') temp = 1;  else  if (a == '*' || a == '/') temp = 2;
10    else  if (a == '+' || a == '-') temp = 3;
11    return temp;      }
12  int main() {
13    bool truth=false;
14    while(!truth){
15    string infix;
16    cout << "Enter an arithmetic expression: " << endl;
17    getline(cin, infix);
18    stack<char> operator_stack;
19    stringstream output;
20    int x=0, y=0;
21     for (unsigned i = 0; i < infix.length(); i++) { 1
22      if (infix[i]=='+'||infix[i]=='-'||infix[i]=='*'||infix[i]=='/'||infix[i]=='^') {      4
23       while(!operator_stack.empty()&&priority(operator_stack.top())<= priority(infix[i])){
24           output << operator_stack.top(); operator_stack.pop();          }
25       operator_stack.push(infix[i]);
26       } else if (infix[i] == '(') {  x++;  3
27       operator_stack.push(infix[i]);
28       } else if (infix[i] == ')') { y++;   5
29       if(y>x) break;
30       while (operator_stack.top() != '(') {
31           output << operator_stack.top();
32           operator_stack.pop();          }
33           operator_stack.pop();
34         } else { output << infix[i]; }        }
35      while (!operator_stack.empty()) {
36          output << operator_stack.top();
37          operator_stack.pop();        }
38      truth = (x==y);
39      if(truth)cout << output.str() << endl;
40      else cout<<"unbalanced parenthesis\n";        }
41      return 0;
42  }
```