Lab Work 6
Due Date: May 16, 2025 (EoD)
Groups of 2.

Task 1: You have already implemented the grammar for arithmetic using Flex and Bison. Your task is to **modify the parser to build a parse tree** instead of directly evaluating the expressions.

You will:
- Update your grammar and bison files to have the = (assignment operator) in it.
- Define a tree node structure.
- Replace the arithmetic actions with parse tree construction using `mkTree()`.
- Print the constructed tree after parsing.

To Do:
1. **Base Files**: You have `calcgrammar.l` and `calcgrammar.y` as a base. You can use your own extended ones also, but be careful with the errors.
2. Do NOT evaluate expressions directly (no `$1 + $3`). Instead, use `mkTree()` to create nodes.
3. Print the final parse tree in an indented format. Start from root, then traverse the braches.

What to submit:
ONE Zip file (not RAR). Name of the file: **ID1_ID2.zip**
At the root of the zip file (NO sub-folders), place your **.l .y** and **makefile**
Assume that you have your input in math.txt as a **single-line single-instruction** format.

Heling Code: You can use the following helping code. It is up to you to understand, modify and place it appropriately. Be careful with the operators, unary or binary, or more than that.

```
typedef struct TreeNode {
    char *label;
    struct TreeNode *left;
    struct TreeNode *right;
} TreeNode;

TreeNode* mkTree(const char *label, TreeNode *left, TreeNode *right) {
    TreeNode *node = malloc(sizeof(TreeNode));
    node->label = strdup(label);
    node->left = left;
    node->right = right;
    return node;
}
```

```
%union {
    double dval;
    TreeNode* node;
}

%token <dval> VALf
%type <node> expr
```

```
expr:
    expr '+' expr  { $$ = mkTree("+", $1, $3); }
  | expr '-' expr  { $$ = mkTree("-", $1, $3); }
  | expr '*' expr  { $$ = mkTree("*", $1, $3); }
  | expr '/' expr  { $$ = mkTree("/", $1, $3); }
  | '(' expr ')'   { $$ = $2; }
  | VALf           {
        char buf[32];
        snprintf(buf, sizeof(buf), "%g", $1);
        $$ = mkTree(strdup(buf), NULL, NULL);
    }
;
```

```
void printTree(TreeNode *node, int indent) {
    if (!node) return;
    for (int i = 0; i < indent; i++) printf("  ");
    printf("%s\\n", node->label);
    printTree(node->left, indent + 1);
    printTree(node->right, indent + 1);
}
```

```
TreeNode *root = result_of_top_level_expr;
printTree(root, 0);
```