tree.h

```
#ifndef _TREE_H_
#define _TREE_H_

#include <stdlib.h>
#include "lexer.h"

typedef struct tree_node *Tree;
struct tree_node {
   Token node;
   Tree left;
   Tree right;
};

Tree tree_create(Token tokens[], int idx_left, int idx_right);
void tree_print(Tree t, size_t depth);
void tree_infix(Tree t);

#endif
```

tree.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>

#include "tree.h"

int get_priority(char c)
{
   switch (c) {
      case '+': case '-': return 1;
```

```
        case '*': case '/': return 2;
        case '^': return 3;
    }
    return 100;
}

Tree tree_create(Token tokens[], int idx_left, int idx_right)
{
    Tree t = (Tree) malloc(sizeof(struct tree_node));
    if (idx_left > idx_right) {
        return NULL;
    }

    if (idx_left == idx_right) {
        t->node  = tokens[idx_left];
        t->left  = NULL;
        t->right = NULL;
        return t;
    }

    int priority;
    int priority_min = get_priority('a');
    int brackets = 0;
    int op_pos;

    for (int i = idx_left; i <= idx_right; ++i) {
        if ((tokens[i].type == BRACKET) && (tokens[i].data.is_left_bracket)) {
            ++brackets;
            continue;
        }
        if ((tokens[i].type == BRACKET) && !(tokens[i].data.is_left_bracket)) {
```

```
            --brackets;
            continue;
        }
transform.h
#ifndef __TRANSFORM_H__
#define __TRANSFORM_H__

#include "tree.h"

void tree_transform(Tree *t);

#endif
transform.c
#include "tree.h"
#include "transform.h"

int conditional_test(Tree *t)
{
    if  (((*t) == NULL) || ((*t)->node.type != OPERATOR)
        || ((*t)->node.data.operator_name != '/')) {
        return 0;
    }

    if  ((((*t)->right->node.type == INTEGER) && ((*t)->right->node.data.value_int == 1)) ||
        (((*t)->right->node.type == FLOATING) && ((*t)->right->node.data.value_float == 1.0))) {
        return 1;
    }

    return 0;
}
```

```
void transform_power(Tree *t)
{
   Tree tmp;



  if (conditional_test(t) == 1) {
      tmp = (*t)->left;
      (*t)->left = NULL;
   }
   //tree_delete(t);
   *t = tmp;
}

void tree_transform(Tree *t)
{
   if ((*t) != NULL) {
      tree_transform(&((*t)->left));
      tree_transform(&((*t)->right));
      if (conditional_test(t)) {
         transform_power(t);
      }
   }
}
```

main.c

```
#include <stdio.h>
#include <stdlib.h>

#include "lexer.h"
#include "tree.h"
#include "transform.h"
```

```
int main(void)
{
   printf("Enter exprassion and \'!\' ,then progremme will print tree and will print transformed tree
\n");
   printf("^D - for exit\n");

   size_t tokens_qty = 0;
   Token tokens_1[256];
   Token token;
      Token tokens[256];
      token_next(&token);
   while (token.type != FINAL) {
      tokens[tokens_qty++] = token;
      token_next(&token);
   }

   Tree tree = tree_create(tokens, 0, tokens_qty - 1);

   printf("\nExpression tree:\n");
   tree_print(tree, 0);

   tree_transform(&tree);

   printf("\nSemitransformed expression tree:\n");
   tree_print(tree, 0);

   printf("\nTree's infix linearization:\n");
   tree_infix(tree);
   printf("\n");

   return 0;
```

```
}
C:\Users\Бондский\Desktop\24 лаба она  моя> gcc main.c transform.c tree.c lexer.c

C:\Users\Бондский\Desktop\24 лаба она  моя> a.exe
Enter exprassion and '!' ,then progremme will print tree and will print transformed tree
^D - for exit
(6/1*3)!

Expression tree:
      3
*
          1
      /
          6

Semitransformed expression tree:
      3
*
      6

Tree's infix linearization:
(6*3)
```