```
#include "stdio.h"
#include "stdlib.h"
#include "malloc.h"
#include "string.h"
#include "stdbool.h"

typedef struct tree {
	char key;
	struct tree *left;
	struct tree *right;
	bool first;
} node;

int prior(char c) {
	switch (c) {
	case '+': case '-': return 1;
	case '*': case '/': return 2;
	case '^': return 3;
	}
	return 10;
};

node *maketree(char expr[], int first, int last) {
	int minpr, i, k, prt, sk = 0;
	node *t = NULL;
	t = (node*)malloc(sizeof(node));
	if (first == last) {
		t->key = expr[first];
		t->left = NULL;
		t->right = NULL;
		return t;
	}
	minpr = 10;
	for (i = first; i <= last; i++) {
		if (expr[i] == '(') { sk++; continue; }
		if (expr[i] == ')') { sk--; continue; }
		if (sk > 0) continue;
		prt = prior(expr[i]);
		if (prt <= minpr) {
			minpr = prt;
			k = i;
		}
	}
	if (minpr == 10 && expr[first] == '('&&expr[last] == ')') {
		free(t);
		return maketree(expr, first + 1, last - 1);
	}
	t->key = expr[k];
	t->left = maketree(expr, first, k - 1);
	t->right = maketree(expr, k + 1, last);
	return t;
}

void print_tree(node *root, int h) {
	if (root != NULL) {
		print_tree(root->right, h + 3);
		printf("-%*c%c\n", h, ' ', root->key);
		print_tree(root->left, h + 3);
	}
}

void print_expression(node *root) {
	int p;
```

```
        if (!root) return;
        p = 0; if (root->left != NULL && prior(root->key) > prior(root->left->key)) {
                p = 1; printf("(");
        }
        print_expression(root->left); if (p == 1) printf(")");
        printf("%c", root->key);
        p = 0; if (root->right != NULL && (prior(root->key) > prior(root->right->key) || (prior(root->key) ==
prior(root->right->key) && root->key == '/'))) {
                p = 1; printf("(");
        }
        print_expression(root->right); if (p == 1) printf(")");
}

void task(node *root) {
        node *tmp = NULL;

        if ((root->first == true) && (root->key == '*') && ((root->left->key == '1') || (root->right->key == '1'))) {
                if (root->left->key == '1') {
                        free(root->left);
                        root->right->first = true;
                        free(root);
                }
        }
        if ((root->first == true) && (root->key == '*') && ((root->left->key == '1') || (root->right->key == '1'))) {
                if (root->right->key == '1') {
                        free(root->right);
                        root->left->first = true;
                        free(root);
                }
        }
        if ((root->left->key == '*') && ((root->left->left->key == '1') || (root->left->right->key == '1'))) {
                if (root->left->left->key == '1' ) {
                        free(root->left->left);
                        root->left = root->left->right;
                }

        }
        if ((root->left->key == '*') && ((root->left->left->key == '1') || (root->left->right->key == '1'))) {
                if (root->left->right->key == '1') {
                        free(root->left->right);
                        root->left = root->left->left;
                }
        }

        if ((root->right->key == '*') && ((root->right->left->key == '1') || (root->right->right->key == '1'))) {
                if (root->right->left->key == '1') {
                        free(root->right->left);
                        root->right = root->right->right;
                }

        }
        if ((root->right->key == '*') && ((root->right->left->key == '1') || (root->right->right->key == '1'))) {
                if (root->right->right->key == '1') {
                        free(root->right->right);
                        root->right = root->right->left;
                }
        }

        if (root->left != NULL && root->left->left != NULL && root->left->right != NULL)
                task(root->left);
        if (root->right != NULL && root->right->left != NULL && root->right->right != NULL)
                task(root->right);
}
```

```
void menu() {
	printf("============================\n");
	printf("||   1-Enter expression                ||\n");
	printf("||   2-Print expression                ||\n");
	printf("||   3-Print tree                      ||\n");
	printf("||   4-Curry to task                   ||\n");
	printf("||   5-Menu                            ||\n");
	printf("||   0-End                             ||\n");
	printf("============================\n");
	printf("\n");
}

int main() {
	node *t = NULL;
	int ch = 10, k, x = 0, l = 0;
	char data[100];
	menu();
	while (ch != 0) {
		printf("=> ");
		scanf("%d", &ch);
		switch (ch) {
		case 1: printf("Enter expression: ");
			scanf("%s", data);
			k = strlen(data);
			t = maketree(data, 0, k - 1);
			break;
		case 2: if (t != NULL) {
			print_expression(t);
			printf("\n");
		}
				else printf("Expression not enter\n");
			break;
		case 3: if (t != NULL) print_tree(t, 0);
				else printf("Expression not enter\n");
			break;
		case 4:
			t->first = true;
			task(t);
			break;
		case 5:
			menu();
			break;
		}
	}
	return 0;
}
```

```
Admin@LAPTOP-Q5U6S2UH:/mnt/c/Users/Admin/Desktop/Все для вуза$ ./a.out
=============================
||   1-Enter expression           ||
||   2-Print expression           ||
||   3-Print tree                 ||
||   4-Curry to task              ||
||   5-Menu                       ||
||   0-End                        ||
=============================

=> 1
Enter expression: 9*4+4*1
=> 3
-     1
-   *
-     4
- +
-     4
-   *
-     9
=> 4
=> 3
-   4
- +
-     4
-   *
-     9
=> 2
9*4+4
=> 1
Enter expression: (9+4)*2*1*3*4
=> 3
-   4
- *
-     3
-   *
-      1
-    *
-        2
-      *
-          4
-        +
-          9
=> 4
=> 2
(9+4)*2*3*4
=> 1
Enter expression: 1*(9-5)-3*1+(9+4)-2
=> 3
-  2
- -
-      4
-    +
-      9
-  +
-        1
-      *
-        3
-    -
-          5
-        -
-          9
-      *
-        1
=> 4
=> 2
9-5-3+9+4-2
```