```
#include "expr.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *makeString(char *s1, char *s2, char *s3) {
  char *s =
      (char *)malloc((strlen(s1) + strlen(s2) + strlen(s3) + 1) * sizeof(char));
  strcpy(s, s1);
  strcat(s, s2);
  strcat(s, s3);
  return s;
}

Node *createNode(char *s, double val) {
  Node *node = (Node *)malloc(sizeof(Node));
  node->expr_string = (char *)malloc((strlen(s) + 1) * sizeof(char));
  strcpy(node->expr_string, s);
  node->left = NULL;
  node->right = NULL;
  node->num_parents = 0;
  node->value = val;
  return node;
}

Node *binop(Operation op, Node *a, Node *b) {
  if ((a->num_parents == 1) || (b->num_parents == 1)) {
    return NULL;
  }
  Node *n = NULL;
  char *s = NULL, *s1 = NULL, *s2 = NULL;
  switch (op) {
  case addop:
    s = makeString(a->expr_string, "+", b->expr_string);
    n = createNode(s, 0);
    free(s);
    break;
  case subop:
    s = makeString(a->expr_string, "-", b->expr_string);
    n = createNode(s, 0);
    free(s);
    break;
  case mulop:
    s1 = makeString("(", a->expr_string, ")");
```

```
    s2 = makeString("(", b->expr_string, ")");
    s = makeString(s1, "*", s2);
    n = createNode(s, 0);
    free(s1);
    free(s2);
    free(s);
    break;
  case divop:
    s1 = makeString("(", a->expr_string, ")");
    s2 = makeString("(", b->expr_string, ")");
    s = makeString(s1, "*", s2);
    n = createNode(s, 0);
    free(s1);
    free(s2);
    free(s);
    break;
  }
  n->left = a;
  n->right = b;
  n->operation = op;
  a->num_parents += 1;
  b->num_parents += 1;
  return n;
}

double evalTree(Node *root) {
  if (root == NULL) {
    return 0;
  }
  if ((root->left == NULL) && (root->right == NULL)) {
    return root->value;
  }
  root->left->value = evalTree(root->left);
  root->right->value = evalTree(root->right);
  switch (root->operation) {
  case addop:
    return root->left->value + root->right->value;
  case subop:
    return root->left->value - root->right->value;
  case mulop:
    return root->left->value * root->right->value;
  case divop:
    return root->left->value / root->right->value;
  default:
```

```
    break;
  }
}

void freeTree(Node *root) {
  if (!root)
    return;
  freeTree(root->left);
  freeTree(root->right);
  free(root->expr_string);
  free(root);
}

Node *duplicateTree(Node *root) {
  if (!root)
    return NULL;
  Node *n = createNode(root->expr_string, root->value);
  n->operation = root->operation;
  n->num_parents = root->num_parents;
  n->left = duplicateTree(root->left);
  n->right = duplicateTree(root->right);
  return n;
}

void printTree(Node *root) {
  if (!root)
    return;
  printf("Node\n\texpr_string = %s\n\tvalue = %g\n\tnum_parents = %d\n",
         root->expr_string, root->value, root->num_parents);
  printTree(root->left);
  printTree(root->right);
}

==767==
==767== HEAP SUMMARY:
==767==     in use at exit: 0 bytes in 0 blocks
==767==   total heap usage: 39 allocs, 39 frees, 1,438 bytes allocated
==767==
==767== All heap blocks were freed -- no leaks are possible
==767==
==767== For counts of detected and suppressed errors, rerun with: -v
==767== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```