# SE 2S03 — Assignment 3

Ned Nedialkov

30 October 2019

**Due date:** 13 November

- **Avenue will be open until 18 Nov 12:20pm.**

  **No penalty if submitted between 13th and 18th, 12:20pm.**

- **No hardcopy will be accepted after 12:20pm on 18 Nov.**

  **NO EXCEPTIONS**

  **The above captures the 5-day extension for MSAFs, so no need to submit such.**

- You will need to implement seven C functions to build and process a simple binary tree. This is a short assignment, but requires deep understanding.
- You can read about binary trees at e.g. `https://en.wikipedia.org/wiki/Binary_tree`.
- Note: If the text that follows does not forbid explicitly some feature, function, etc., then you can use it.

- **Store your implementation in a file with name `expr.c`**
- **Submit `expr.c` on Avenue.**
- **Submit a hard-copy of `expr.c`.**
- **If one of these two submissions is missing, the grade will be zero.**

**Problem 1** (20 points) You are given the file `expr.h` containing

```
#ifndef EXPR_H
#define EXPR_H
typedef enum {
    addop = 0,    /* encodes addition */
    subop,        /* subtraction */
    mulop,        /* multipllicaiton */
    divop,        /* division */
} Operation;
struct node {
    Operation operation;
    double value;
    char* expr_string;
    int num_parents;
    struct node *left, *right;
};
typedef struct node Node;
char* makeString(char* s1, char* s2, char* s3);
Node* createNode(char* s, double val);
Node* binop(Operation op, Node* a, Node* b);
double evalTree(Node* root);
void freeTree(Node* root);
Node* duplicateTree(Node* root);
void printTree(Node* root);
#endif
```

Implement the functions in this file as follows. For each function returning a pointer, if the corresponding memory cannot be allocated, i.e. `malloc` fails, the `NULL` pointer must be returned.

```
char* makeString(char* s1, char* s2, char* s3);
```

Concatenates the strings at `s1`, `s2`, and `s3` and returns a pointer to the concatenated string. For example, `printf("%s\n", makeString("ab", "cd", "ef"));` should output `abcdef`.

```
Node* createNode(char* s, double val);
```

Creates a node, copies the string at `s` to `expr_string`, sets `left` and `right` to `NULL`, `num_parents` to 0, and `value` to `val`, and returns a pointer to the node.

Note: if `s == NULL`, nothing is copied.

Do Not Share this document

```
Node* binop(Operation op, Node* a, Node* b);
```

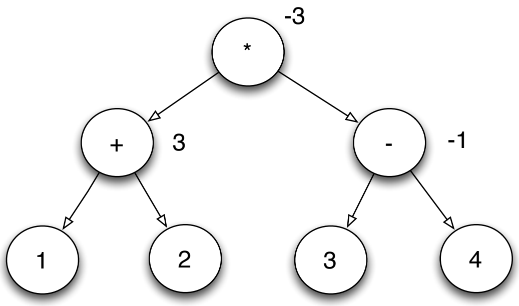
Creates a node, as explained below, and returns a pointer to it.

If the number of parents in a or b is 1, this function returns NULL. Otherwise, it creates a node and sets in this node: left=a, right=b, operation=op, increments the number of parents in a and b by 1, and creates an expression string as follows.

Suppose the expression strings associated with a->exp_string and b->expr_string are ''a'' and ''b'', respectively. Then if the operation is addop or subop, this function creates ''a+b'' or ''a-b'', respectively. If the operation is mulop or divop, it creates ''(a)*(b)'' or ''(a)/(b)'', respectively. This string is stored at expr_string.

```
double evalTree(Node* root);
```

It evaluates the tree at root by applying the operation at each node as illustrated below and returns the value at the root. The value of each binop (non-leaf) node must be set to the result of the expression it contains when this function is executed.



```
void freeTree(Node* root);
```

Deallocates the memory associated with the tree at root.

```
Node* duplicateTree(Node* root);
```

Creates a copy of the tree at root and returns a pointer to the root of the new tree. At this root, num_parents must be 0.

```
void printTree(Node* root);
```

Prints on the standard output the contents of the nodes of the tree at root.

Do Not Shar ocument

For example, my main program

```
#include <stdio.h>
#include "expr.h"
#define N 4
int main()
{
    Node* a[4];
    a[0] = createNode("a", .1);
    a[1] = createNode("b", .2);
    a[2] = createNode("c", .3);
    a[3] = createNode("d", .4);
    Node* t1 = binop(mulop, a[0], a[1]); /* a*b */
    Node* t2 = binop(addop, a[2], t1);   /* c+a*b */
    Node* t3 = binop(mulop, a[3], t2);   /* d*(c+a*b) */
    Node* t4 = duplicateTree(t3);        /* d*(c+a*b) */
    Node* t5 = binop(subop, t4, t3);
    Node* f = t5;
    printf("--- Tree at t4 \n");
    printTree(t4);
    double val = evalTree(f);
    printf("\n--- Value at root of f  %g\n", val);
    printf("--- Evaluated Tree at f \n");
    printTree(f);
    freeTree(f);
    return 0;
}
```

produces

```
--- Tree at t4
 Node
     expr_string = (d)*(c+(a)*(b))
     value       = 0
     num_parents = 1
 Node
     expr_string = d
     value       = 0.4
     num_parents = 1
 Node
     expr_string = c+(a)*(b)
     value       = 0
     num_parents = 1
 Node
     expr_string = c
     value       = 0.3
     num_parents = 1
 Node
     expr_string = (a)*(b)
     value       = 0
```

Do Not Share this document

```
    num_parents = 1
 Node
    expr_string = a
    value       = 0.1
    num_parents = 1
 Node
    expr_string = b
    value       = 0.2
    num_parents = 1

--- Value at root of f  0
--- Evaluated Tree at t5
 Node
    expr_string = (d)*(c+(a)*(b))-(d)*(c+(a)*(b))
    value       = 0
    num_parents = 0
 Node
    expr_string = (d)*(c+(a)*(b))
    value       = 0.128
    num_parents = 1
 Node
    expr_string = d
    value       = 0.4
    num_parents = 1
 Node
    expr_string = c+(a)*(b)
    value       = 0.32
    num_parents = 1
 Node
    expr_string = c
    value       = 0.3
    num_parents = 1
 Node
    expr_string = (a)*(b)
    value       = 0.02
    num_parents = 1
 Node
    expr_string = a
    value       = 0.1
    num_parents = 1
 Node
    expr_string = b
    value       = 0.2
    num_parents = 1
 Node
    expr_string = (d)*(c+(a)*(b))
    value       = 0.128
    num_parents = 1
 Node
    expr_string = d
    value       = 0.4
    num_parents = 1
 Node
    expr_string = c+(a)*(b)
```

Do Not Share this document

```
    value       = 0.32
    num_parents = 1
Node
    expr_string = c
    value       = 0.3
    num_parents = 1
Node
    expr_string = (a)*(b)
    value       = 0.02
    num_parents = 1
Node
    expr_string = a
    value       = 0.1
    num_parents = 1
Node
    expr_string = b
    value       = 0.2
    num_parents = 1
```

Use the following makefile

```
CFLAGS= -Wall -Werror -ansi -g
CC=gcc
main: main.o expr.o
clean:
        rm *~ *.o main
```

**Problem 2** (3 points) Before submitting your code, format it with `clang-format`; 0 points if not formatted.

After you format it, run `cloc expr.c` (`http://cloc.sourceforge.net/`) and submit the output with your hardcopy. This is my output

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                                1             14              0            121
-------------------------------------------------------------------------------
```

**Problem 3** ($-X$ points) On mills.mcmaster.ca, run

```
valgrind ./main
```

If `valgrind` reports $X$ memory errors when we test your functions, $X$ is subtracted from the grade in Problem 1. Your output should look like

```
==22366== HEAP SUMMARY:
==22366==     in use at exit: 22,648 bytes in 164 blocks
==22366==   total heap usage: 214 allocs, 50 frees, 29,668 bytes allocated
==22366==
```

Do Not Share this document

```
==22366== LEAK SUMMARY:
==22366==    definitely lost: 0 bytes in 0 blocks
==22366==    indirectly lost: 0 bytes in 0 blocks
==22366==      possibly lost: 72 bytes in 3 blocks
==22366==    still reachable: 348 bytes in 9 blocks
==22366==         suppressed: 22,228 bytes in 152 blocks
==22366== Rerun with --leak-check=full to see details of leaked memory
==22366==
==22366== For counts of detected and suppressed errors, rerun with: -v
==22366== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
```

For example, if `ERROR SUMMARY: 3 errors`, we subtract 3 from your grade.

**Problem 4** (5 points)   If `valgrind` reports

```
definitely lost: 0 bytes in 0 blocks
indirectly lost: 0 bytes in 0 blocks
```

**Bonus**   (5 points) Run

```
clang-format expr.c > tmp.c ; cloc tmp.c
```

The student with the smallest number of code lines receives 5 points, the second smallest 4 and so on.

To make this fun and competitive, you can enter your results at `https://docs.google.com/spreadsheets/d/16zRM-3EAfvIxK2FlNxDAT5ZuJUI-7qwFVTkkrNkbnEM/edit?usp=sharing`

Do Not Share this document