This is bomb 86 for lab comp40fall23.

It is owned by the following student(s):
nadjek01
aharpe02

Overview
    Names: Ayah Harper, Nana Adjekum
    Bomb: 86
    Help Received: TA's & Piazza
    Hours: ~20

Defuse
    phase_1: Midnight has been canceled.
    phase_2: 1 2 6 24 120 720
    phase_3: 1 -1726
    phase_4: 5
    phase_5: rucdka
    phase_6: 667
    secret_phase: 36

Descriptions
Phase 1:
    Phase 1 takes in a string from user input and compares
    it to the string already in the program.
    If the string is not equal to this, the program blows up.
        Defuse Statement: "Midnight has been canceled."

Phase 2:
    Phase 2 takes in a string containing 6 numbers and reads it.
    It then uses each of these numbers and calculates the
factorial
    of the index + 1.

    (n) = (i) * (n - 1) --> where i starts at 1 and increments
from 1 to 6

        Defuse Statement: 1 2 6 24 120 720

Phase 3:
    Phase 3 implements 7 switch case statements. That each
    calculate and return a different integer result. The
    function takes in the switch case to start at and the
    result of that case. In our case we used 1 and -1726.
        Defuse Statement: 1 -1726

Phase 4:
    Phase 4 implements a factorial function (fun4) and requires the user to provide the number that will result in 120 (5!).
        Defuse Statement: 5

Phase 5:
    We have the code for phase 5 in phase_5.c. But this function takes in a string and does a key look-up in an array based on a mod16.
        Defuse Statement: rucdka

Phase 6:
    We have the code for phase 6 in phase_6.c. But this function takes in the second largest number in a linked list and compares
    it to the sorted linked list. that the program builds out.
        Defuse Statement: 667
Secret_phase:
     To get to the secret phase you have to pass in two arguments to phase 4—the correct value (“5”) to defuse the bomb followed by a string that will get the executable code to flow into the secret phase function call (“Igor_Straminsky”). The secret_phase reading in a string converting it to an integer via strtol and checking the range of said converted integer. It then calls fun7, if the function is within the range. Fun7 seems to be doing some sort of list traversal, where if the base case is 0 the function returns 0, and the conditional jumps are dependent on a comparison case of the user provided value and the pre-determined value (node value). If the comparison results in the user provided number being larger than the node value and is not equal to the node value it moves on to the next node. We passed in 36 which we were able to attain from printing out the first value passed into rdi register (node value for comparison).

Code
    PHASE 5:
    /* phase_5.c
    *
    * by Ayah Harper and Nana Adjekum, 11/5/23
    *
    * This file incldes our interpretation of the C code that may correspond
    * to the assembly instructions in phase 5 of the binary bomb.

```
    */
    #include <stdbool.h>
    #include <stdio.h>
    #include <string.h>
    #include <stdlib.h>

    // extern bool strings_not_equal(char *string1, char
*string2);
    void explode_bomb();
    char *is_string_ravens(char *input_string);

    int main(int argc, char *argv[])
    {
            // call function
            char *test = is_string_ravens((char *)argv[1]);

            if (test == NULL) {
                    exit(EXIT_FAILURE);
            }

            exit(EXIT_SUCCESS);
    }

    char *is_string_ravens(char *input_string)
    {
            if (strlen(input_string) != 6) {
                    explode_bomb();
                    return(NULL);
            }

            char *ravens = "ravens";
            char *letters = "isrveawhobpnutfg";
            char *results = "";

            for (int i = 0; i < strlen(input_string); i++) {

                    int index = input_string[i] %
strlen(letters);

                    results += letters[index];
            }

            if (strcmp(ravens, results) == 1) {
                    explode_bomb();
                    return NULL;
            }
```

```
        return ravens;

}

void explode_bomb()
{
        printf("BOOM!\n");
}

PHASE 6:
/* phase_6.c
*
* by Ayah Harper and Nana Adjekum, 11/5/23
*
* This file includes our interpretation of the C code that
may correspond
* to the assembly instructions in phase 5 of the binary
bomb.
*/

#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>


typedef struct Node {
        int value;
        struct Node *next;
} Node;

Node *sort_and_return_second(Node *list);
Node *bubbleSort(Node* head);


int main(int argc, char *argv[])
{
        /* build a list of 9 nodes */
        struct Node node1;
        struct Node node2;
        struct Node node3;
        struct Node node4;
        struct Node node5;
        struct Node node6;
        struct Node node7;
```

```
        struct Node node8;
        struct Node node9;

        node1.value = 127;
        node1.next = &node2;
        node2.value = 101;
        node2.next = &node3;
        node3.value = 406;
        node3.next = &node4;
        node4.value = 667;
        node4.next = &node5;
        node5.value = 157;
        node5.next = &node6;
        node6.value = 901;
        node6.next = &node7;
        node7.value = 531;
        node7.next = &node8;
        node8.value = 256;
        node8.next = &node9;
        node9.value = 629;
        node9.next = NULL;

        /* answer points to the second largest node! */
        Node *answer = sort_and_return_second(&node1);

}

/* bubble sorts a list in descending order and returns
*  the head of the sorted list */
Node *sort_and_return_second(Node *list)
{
        Node *sorted = bubbleSort(list);

        return (sorted)->next;
}

/* bubble sorting helper function */
Node *bubbleSort(Node* head)
{
        Node *current;
        Node *nextNode;
        Node *prev = NULL;
        bool swaps_made;

        /* return empty list */
        if (head == NULL)
```

```
                return head;

        /* bubble sort the list, swapping adjacent nodes
        *  that are out of order */
        do {
                swaps_made = false;;
                current = head;
                nextNode = (head)->next;

                while (nextNode != prev) {
                        if ((current)->value < (nextNode)-
>value) {
                                if (prev != NULL) {
                                        (prev)->next =
nextNode;
                                } else {
                                        (head) = nextNode;
                                }
                                (current)->next =
(nextNode)->next;
                                (nextNode)->next = current;
                                swaps_made = true;
                                current = nextNode;
                        }
                        prev = current;
                        current = (current)->next;
                        nextNode = (current)->next;
                }

        } while (swaps_made);

        /* return the head of the sorted list */
        return head;
    }
```