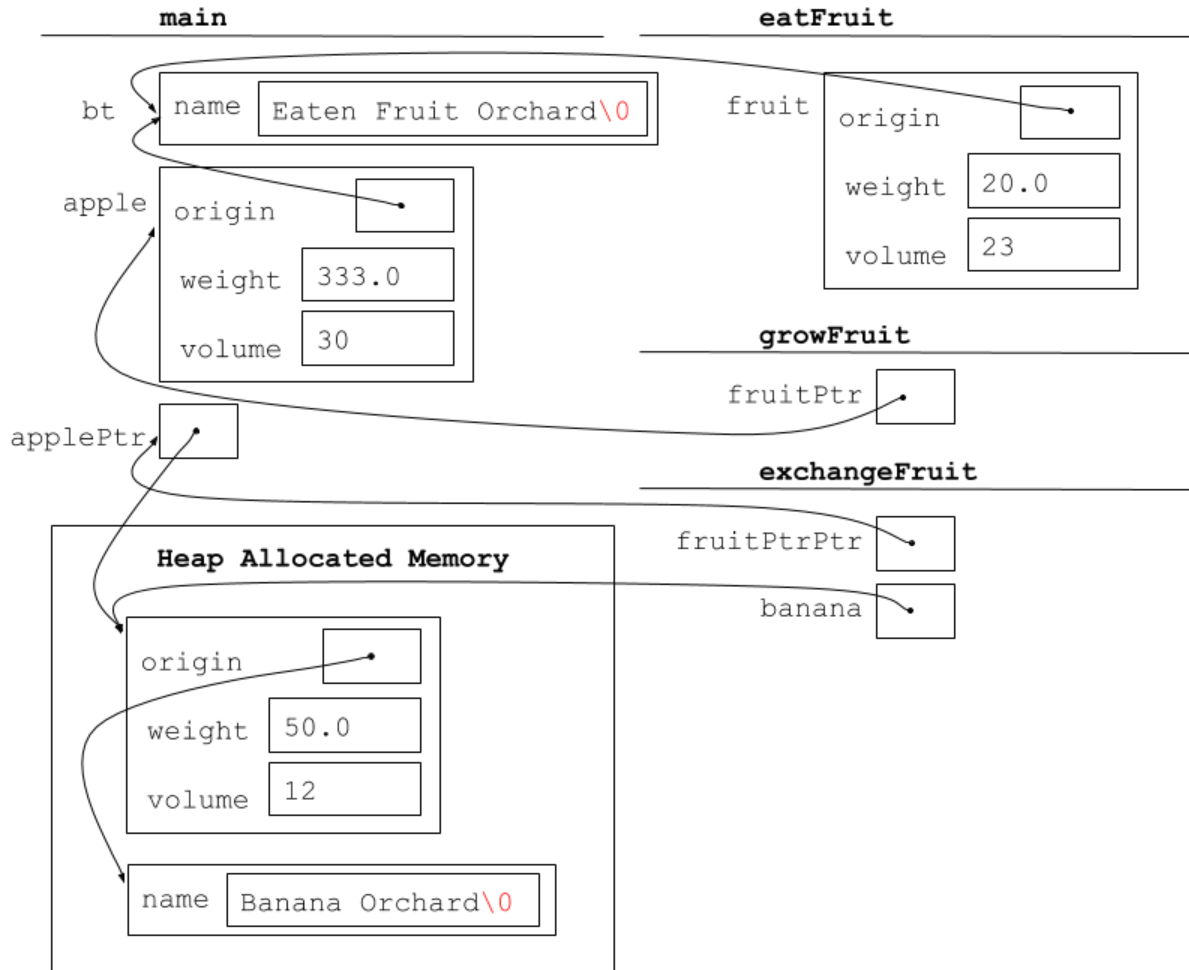


CSE 333 – Section 2: Structs, Debugging, Memory Management, and Valgrind

SOLUTIONS

1. Structs and Pointers

Final memory diagram (Note: eatFruit, growFruit, and exchangeFruit contexts would be cleaned up and reused during program execution).



Output:

- | | | |
|----|----------------------------------|-----------------------------------|
| 1. | "20.5, 33, Apple Orchard" | Initial values that were assigned |
| 2. | "20.5, 23, Eaten Fruit Orchard" | Struct is passed by value |
| 3. | "333.0, 30, Eaten Fruit Orchard" | Struct passed by "reference" |
| 4. | "50.0, 12, Banana Orchard" | Struct is completely reassigned |

2. Reverse a Linked List [Extra Practice]

```
struct Node* reverse(struct Node* head) {
    struct Node *prev = NULL, *next = NULL;
    struct Node *current = head;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }

    return prev;
}
```

3. Sorted Array To Binary Search Tree [Extra Practice].

```
struct TreeNode *sortedArrayToBST(int[] arr, int low, int high) {
    if (low > high) {
        return NULL;
    }

    // Make the middle element the root of this subtree.
    int mid = (low + high) / 2;
    struct TreeNode *root = (struct TreeNode*)malloc(sizeof(TreeNode));
    root->value = arr[mid];

    // Construct the left subtree and assign it to be the left child.
    root->left = sortedArrayToBST(arr, low, mid - 1);

    // Construct the right subtree and assign it to be the right child.
    root->right = sortedArrayToBST(arr, mid + 1, high);

    return root;
}
```

4. Leaky Code and Valgrind

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

// Returns an array containing [n, n+1, ... , m-1, m]. If n>m, then the
// array returned is []. If an error occurs, NULL is returned.
int* rangeArray(int n, int m) {
    int length = m - n + 1;

    // Heap allocate the array needed to return
    int *array = (int*) malloc(sizeof(int) * length);

    // Initialize the elements
    // By using <=, we are writing to length + 1 ints instead of length ints
    // Change <= to < to fix this off-by-one error
    for (int i = 0; i < length; i++) {
        array[i] = i + n;
    }

    return array;
}

// Accepts two integers as arguments
int main(int argc, char *argv[]) {
    if (argc != 3) return EXIT_FAILURE;

    int n = atoi(argv[1]), m = atoi(argv[2]); // Parse cmd-line args
    int *nums = rangeArray(n, m);

    // Print the resulting array
    // We're allocating space for 10 ints, but we access 11
    // ints with i <= instead of i <
    for (int i = 0; i < (m - n + 1); i++) {
        printf("%d", nums[i]);
    }

    // We need to free the array of integers malloced in RangeArray.
    free(nums);

    // Append newline char to our output
    puts("");

    return EXIT_SUCCESS;
}
```