



SCHOOL OF COMPUTER SCIENCE

LLM internship documentation

And Even A Fancy Subtitle

Oscar and Lee

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree
of Bachelor of Science in the Faculty of Engineering.

Wednesday 26th June, 2024

Contents

1	Query	8
2	Examples	10
2.1	Example 1	10
2.2	Example 2	10
2.3	Example 3	11
2.4	Example 4	13
3	LLM 1 (GPT 4)	14
3.1	Example 1	14
3.2	Example 2	14
3.3	Example 3	15
3.4	Example 4	15
4	LLM 2 (Gemini Pro)	17
4.1	Example 1	17
4.2	Example 2	17
4.3	Example 3	18
4.4	Example 4	18
5	Critical Evaluation	20
6	Conclusion	21
A	Appendix A: AI Prompts	23
B	An Example Appendix	24

List of Figures

List of Tables

Summary of Changes

Supporting Technologies

Notation and Acronyms

Chapter 1

Query

```
1 # Preamble
2 You are given a C program. We need to create a proof harness function.
3
4 # Code generation example
5 Q: Write a method "void proof_harness_withdraw()" that tests method withdraw below for
   all possible inputs.
6
7 // Define the Account structure
8 struct Account {
9     unsigned short bal;
10 };
11
12 // Function to withdraw an amount from an account
13 void withdraw(struct Account *account, unsigned short amount) {
14     unsigned short de = account->bal;
15     account->bal = de - amount;
16 }
17
18 A:
19 struct Account {
20     unsigned short bal;
21 };
22
23
24 void withdraw(struct Account *account, unsigned short amount) {
25     unsigned short de = account->bal;
26     account->bal = de - amount;
27 }
28
29 void proof_harness_withdraw() {
30     struct Account *account = (struct Account *)malloc(sizeof(struct Account));
31     __CPROVER_assume(account != NULL); // Ensure account is not NULL
32
33     unsigned short amount;
34
35     __CPROVER_assume(account->bal >= 0);
36     __CPROVER_assume(amount > 0);
37     __CPROVER_assume(account->bal >= amount);
38
39     unsigned short initial_balance = account->bal;
40
41     withdraw(account, amount);
42     assert(account->bal == initial_balance - amount);
43
44     free(account);
45 }
46
47
48 # Instruction
49 Give me a proof harness code of the below C code.
50
51 # Query
52 Q: Write method "void proof_harness()" that tests the method (\\input method) below for
   all possible inputs.
```

```
53
54 //Input Code
55
56
57 # Constraints
58 Here are some constraints that you should respect:
59 - Give me only the translated code, dont add explanations or anything else.
60 - Use only safe C.
61 - Do not use custom generics. # fuzzer limitation
62 - You're only allowed to modify the proof harness function.
```

Listing 1.1: Preamble and Example

Due to the fact that the example from the original prompt had pointer errors, it has been replaced with the updated response. The language models used are ChatGPT-4 and Gemini Pro, with code examples sourced from Kaggle.

Chapter 2

Examples

2.1 Example 1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct node{
5     struct node *leftNode;
6     int data;
7     struct node *rightNode;
8 };
9
10 struct node *newNode(int data){
11     struct node *node = (struct node *)malloc(sizeof(struct node));
12
13     node->leftNode = NULL;
14     node->data = data;
15     node->rightNode = NULL;
16
17     return node;
18 }
19
20 int main(void){
21     return 0;
22 }
```

Listing 2.1: Code for prompt 1

Example 1 is a code for creating a new node.

2.2 Example 2

```
1 #include <math.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <time.h>
5
6 void calculateMedian(int *array, int size, float *median);
7
8 int main()
9 {
10     int n = 10; // Fixed number of random numbers
11     int i;
12     float median;
13
14     int *a = (int *)malloc(n * sizeof(int));
15
16     if (a == NULL) {
17         fprintf(stderr, "Memory allocation failed\n");
18         return 1;
19     }
20
21     srand(time(NULL));
```

```

22
23     for (i = 0; i < n; i++)
24     {
25         a[i] = rand() % 100;
26     }
27
28     printf("Random Numbers Generated are:\n");
29     for (i = 0; i < n; i++)
30     {
31         printf("%d ", a[i]);
32     }
33     printf("\n");
34
35     calculateMedian(a, n, &median);
36
37     printf("Sorted Data:\n");
38     for (i = 0; i < n; i++)
39     {
40         printf("%d ", a[i]);
41     }
42     printf("\n");
43
44     printf("Median is: %f\n", median);
45
46     free(a);
47     return 0;
48 }
49
50 void calculateMedian(int *array, int size, float *median)
51 {
52     int i, j, temp;
53
54     // Sort the array
55     for (i = 0; i < size; i++)
56     {
57         for (j = i + 1; j < size; j++)
58         {
59             if (array[i] > array[j])
60             {
61                 temp = array[i];
62                 array[i] = array[j];
63                 array[j] = temp;
64             }
65         }
66     }
67
68     // Calculate the median
69     if (size % 2 == 0)
70     {
71         *median = (array[size / 2] + array[(size / 2) - 1]) / 2.0;
72     }
73     else
74     {
75         *median = array[size / 2];
76     }
77 }

```

Listing 2.2: Code fed as a prompt 2

Example 2 is a code for taking the median of 10 random numbers between 0 and 99

2.3 Example 3

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct Node{
5     int data;
6     struct Node *llink;
7     struct Node *rlink;
8 } node;
9
10 node *create_node(int data){
11     node *ptr = (node *)malloc(sizeof(node)
12 );
13     ptr->rlink = ptr->llink = NULL;
14     ptr->data = data;
15     return ptr;
16 }
17 void insert_bt(node **root, int data){

```

```

18 node *new_node = create_node(data);
19 node *temp;
20 node *prev;
21 if (*root == NULL){
22     *root = new_node;
23 }
24 else{
25     temp = *root;
26     prev = NULL;
27     while (temp != NULL){
28         if (new_node->data > temp->data
29 ){
30     prev = temp;
31     temp = temp->rlink;
32 }
33     else if (new_node->data < temp
34 ->data){
35     prev = temp;
36     temp = temp->llink;
37 }
38     else{
39     return;
40 }
41 }
42 if (new_node->data > prev->data){
43     prev->rlink = new_node;
44 }
45 else{
46     prev->llink = new_node;
47 }
48 }
49
50 void search(node *root, int ele){
51     node *temp = root;
52     while (temp != NULL){
53         if (temp->data == ele){
54             break;
55         }
56         else if (ele > temp->data){
57             temp = temp->rlink;
58         }
59         else{
60             temp = temp->llink;
61         }
62     }
63
64     if (temp == NULL){
65         printf("%s\n", "Element not found.");
66     }
67     else printf("%s\n", "Element found.");
68 }
69
70 void inorder_display(node *curr){
71     if (curr != NULL){
72         inorder_display(curr->llink);
73         printf("%d\t", curr->data);
74         inorder_display(curr->rlink);
75     }
76 }
77
78 void postorder_display(node *curr){
79     if (curr != NULL){
80         postorder_display(curr->llink);
81         postorder_display(curr->rlink);
82         printf("%d\t", curr->data);
83     }
84 }
85
86 void preorder_display(node *curr){
87     if (curr != NULL){
88         printf("%d\t", curr->data);
89         preorder_display(curr->llink);
90         preorder_display(curr->rlink);
91     }
92 }
93
94 void delete_bt(node **root, int ele){
95     node *temp;
96     node *prev;
97     if (*root == NULL)
98         return;
99     else{
100         temp = *root;
101         prev = NULL;
102         while (temp != NULL){
103             if (temp->data == ele){
104                 break;
105             }
106             else if (ele > temp->data){
107                 prev = temp;
108                 temp = temp->rlink;
109             }
110             else{
111                 prev = temp;
112                 temp = temp->llink;
113             }
114         }
115
116         if (temp == NULL)
117             return;
118         else
119         {
120             node *replacement;
121             node *t;
122             if (temp->llink == NULL && temp->
123 rlink == NULL){
124                 replacement = NULL;
125             }
126             else if (temp->llink == NULL &&
127 temp->rlink != NULL){
128                 replacement = temp->rlink;
129             }
130             else if (temp->llink != NULL &&
131 temp->rlink == NULL){
132                 replacement = temp->llink;
133             }
134             else{
135                 replacement = temp->rlink;
136                 t = replacement;
137                 while (t->llink != NULL){
138                     t = t->llink;
139                 }
140                 t->llink = temp->llink;
141             }
142
143             if (temp == *root){
144                 free(*root);
145                 *root = replacement;
146             }
147             else if (prev->llink == temp){
148                 free(prev->llink);
149                 prev->llink = replacement;
150             }
151             else if (prev->rlink == temp){
152                 free(prev->rlink);
153                 prev->rlink = replacement;
154             }
155         }
156     }
157 }
158
159 int main(){
160     node *root = NULL;

```

```

158     int choice, n;
159     do{
160         scanf("%d", &choice);
161
162         switch (choice){
163             case 1:
164                 printf("%s\n", "Enter a no:");
165                 scanf("%d", &n);
166                 insert_bt(&root, n);
167                 break;
168             case 2:
169                 inorder_display(root);
170                 printf("\n");
171                 break;
172             case 3:
173                 preorder_display(root);
174                 printf("\n");
175                 break;
176             case 4:
177                 postorder_display(root);
178                 printf("\n");
179                 break;
180             case 5:
181                 printf("%s\n", "Enter a no:");
182                 scanf("%d", &n);
183                 delete_bt(&root, n);
184                 break;
185             case 6:
186                 printf("%s\n", "Enter a no:");
187                 scanf("%d", &n);
188                 search(root, n);
189                 break;
190         }
191     } while (choice != 0);
192     return 0;
193 }

```

Listing 2.3: Code for prompt 3

Example 3 is a code for manipulating binary trees. It involves searching, deleting, ordering, inserting and creating new nodes.

2.4 Example 4

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <assert.h>
5
6
7
8  void calculateMean(int *array, int size);
9
10 int main()
11 {
12     int n = 10, i;
13     int *a;
14
15     // Seed the random number generator with the current time
16     srand(time(NULL));
17
18     a = (int *)malloc(n * sizeof(int));
19
20     printf("Random Numbers Generated are: ");
21     for (i = 0; i < n; i++)
22     {
23         a[i] = rand() % 100;
24         printf("%2d, ", a[i]);
25     }
26     putchar('\n');
27
28     calculateMean(a, n);
29
30     free(a);
31     return 0;
32 }
33
34 void calculateMean(int *array, int size)
35 {
36     int sum = 0;
37     float mean;
38     for (int i = 0; i < size; i++) sum += array[i];
39     mean = sum / (float)size;
40     printf("\nMean: %f\n", mean);
41 }

```

Listing 2.4: Code for prompt 4

Example 4 is a code which calculates the mean for a random set of 10 numbers all between 0-99.

Chapter 3

LLM 1 (GPT 4)

The following are the proof harnesses generated from Chat GPT 4 from their corresponding example C codes.

3.1 Example 1

Query:

```
1 Q: Write method "void proof_harness()" that tests the method newNode below for all possible inputs.
```

Returned proof harness function:

```
1 void proof_harness() {
2     int data;
3
4     struct node *n = newNode(data);
5
6     // Ensure that the returned node is not NULL
7     __CPROVER_assert(n != NULL, "Node allocation failed");
8
9     // Ensure that the node's data is correctly assigned
10    __CPROVER_assert(n->data == data, "Node data assignment failed");
11
12    // Ensure that left and right pointers are NULL
13    __CPROVER_assert(n->leftNode == NULL, "Node leftNode is not NULL");
14    __CPROVER_assert(n->rightNode == NULL, "Node rightNode is not NULL");
15
16    free(n);
17 }
```

Listing 3.1: Generated proof harness function for example 1

Call to CBMC:

```
1 cbmc --function proof_harness newNode.c
```

Output: VERIFICATION SUCCESSFUL

3.2 Example 2

Query:

```
1 Q: Write method "void proof_harness()" that tests the method calculateMedian below for all possible inputs.
```

Returned proof harness function:

```
1 void proof_harness() {
2     int size;
3     __CPROVER_assume(size > 0 && size <= 100); // Assume a reasonable size for the array
4
5     int *array = (int *)malloc(size * sizeof(int));
6     __CPROVER_assume(array != NULL); // Ensure array is not NULL
```

```

7
8     for (int i = 0; i < size; i++) {
9         __CPROVER_assume(array[i] >= -1000 && array[i] <= 1000); // Assume reasonable
10        bounds for array values
11    }
12
13    float median;
14    calculateMedian(array, size, &median);
15
16    // Add assertions to verify post-conditions if needed
17    // Example: Check if median falls within the range of the array
18    __CPROVER_assert(median >= array[0] && median <= array[size - 1], "Median is within
19    the range of the array");
20
21    free(array);
22 }

```

Listing 3.2: Generated proof harness function

Call to CBMC:

```
1 cbmc median.c --function proof_harness --unwind 5
```

VERIFICATION SUCCESSFUL.

Note: I also tried the unwind function for 10 and 15 but the calculation didn't stop for over 5 minutes.

3.3 Example 3

Query:

```
1 Q: Write method "void proof_harness_bst()" that tests the methods insert_bt, search,
inorder_display, postorder_display, preorder_display and delete_bt below for all
possible inputs.
```

Listing 3.3: Prompt Query

```

1 void proof_harness_bst() {
2     node *root = NULL;
3
4     int data;
5     __CPROVER_assume(data >= 0); // Assume a non-negative integer for data
6
7     insert_bt(&root, data);
8     search(root, data);
9     inorder_display(root);
10    preorder_display(root);
11    postorder_display(root);
12    delete_bt(&root, data);
13
14    // Additional assertions to check tree properties can be added here
15    // For example, we can check if the root is NULL after deletion of the only node
16    assert(root == NULL || root->data != data);
17 }

```

Listing 3.4: Generated Proof Harness Function

Call to CBMC:

```
1 cbmc BST.c --function proof_harness_bst --unwind 10
```

VERIFICATION SUCCESSFUL

3.4 Example 4

Query:

```
1 Q: Write method "void proof_harness_mean()" that tests the method calculateMean below for
all possible inputs.
```

```
1 void proof_harness_mean() {
2     int size;
3     __CPROVER_assume(size > 0 && size < 100); // Assume a reasonable size for the array
4
5     int *array = (int *)malloc(size * sizeof(int));
6     __CPROVER_assume(array != NULL); // Ensure array is not NULL
7
8     for (int i = 0; i < size; i++) {
9         __CPROVER_assume(array[i] >= 0 && array[i] <= 100); // Assume elements are within
10         a reasonable range
11     }
12
13     calculateMean(array, size);
14     free(array);
15 }
```

Listing 3.5: Generated Proof Harness Function

Call to CBMC:

```
1 cbmc mean.c --function proof_harness_mean --unwind 10
```

VERIFICATION SUCCESSFUL

Chapter 4

LLM 2 (Gemini Pro)

The following are the proof harness functions generated by Gemini Pro.

4.1 Example 1

Query:

```
1 Q : Write method " void proof_harness () " that tests the method newNode below for all
2 possible inputs .

1 void proof_harness_newNode() {
2     int data;
3     __CPROVER_assume(data >= 0 && data <= 2147483647); // Assuming int range
4
5     struct node *node = newNode(data);
6
7     assert(node->data == data);
8     assert(node->leftNode == NULL);
9     assert(node->rightNode == NULL);
10
11     free(node);
12 }
```

Listing 4.1: Generated Proof Harness Function: Verification Successful

CBMC passed with a successful verification under the following function call.

```
1 cbmc --function proof_harness_newNode --bounds-check --pointer-check --memory-leak-check
   --unsigned-overflow-check newNode.c
```

Listing 4.2: Function Call for Ex 1

4.2 Example 2

Query:

```
1 Q: Write method "void proof_harness_median()" that tests if the code calculates the
   median for any of the possible random inputs.

1 void proof_harness_median() {
2     int size;
3     __CPROVER_assume(size > 0 && size < 1000); // Assuming a reasonable bound for size
4     int *array = (int *)malloc(size * sizeof(int));
5     __CPROVER_assume(array != NULL);
6     float median;
7
8     for (int i = 0; i < size; i++) {
9         array[i] = nondet_int();
10        __CPROVER_assume(array[i] >= 0 && array[i] <= 100); // Assuming values are within
11        0-100
12    }
13    calculateMedian(array, size, &median);
}
```

```

14
15 // Add assertions to check properties of the median
16 // For example:
17 // 1. Check if the median is within the expected range
18 // 2. Check if the median is calculated correctly for both even and odd sizes
19
20 free(array);
21 }

```

Listing 4.3: Generated Proof Harness Function: Example 2

CBMC passed with a successful verification under the following function call:

```

1 cbmc median.c --unwind 15 --function proof_harness

```

Listing 4.4: Function Call for Ex 2

4.3 Example 3

```

1 Q: Write method "void proof_harness_bst()" that tests the methods insert_bt, search,
inorder_display, postorder_display, preorder_display and delete_bt below for all
possible inputs.

```

Listing 4.5: Prompt query

```

1 void proof_harness_bst() {
2     node *root = NULL;
3     int data = nondet_int();
4     __CPROVER_assume(data > 0 && data <= 100);
5
6     // Test insertion
7     insert_bt(&root, data);
8     // Assuming a maximum tree depth to avoid infinite loops
9     // Adjust the bound if needed
10    for (int i = 0; i < 5; ++i) {
11        int newData = nondet_int();
12        __CPROVER_assume(newData > 0 && newData <= 100);
13        insert_bt(&root, newData);
14    }
15    // Add assertions to check if the tree structure is valid
16    // and the inserted node can be found using search
17
18    // Test search
19    search(root, data);
20
21    // Add assertions for inorder_display, postorder_display, preorder_display
22
23    // Test delete
24    delete_bt(&root, data);
25
26    // Add assertions to check if the tree structure is still valid
27    // and the deleted node is not present
28 }

```

Listing 4.6: Generated Proof Harness Function

Under the following call to CBMC

```

1 cbmc BST.c --unwind 15 --function proof_harness_bst

```

Listing 4.7: CBMC call for Ex 3

All tests were passed successful

4.4 Example 4

Query:

```

1 Q: Write method "void proof_harness_mean()" that tests if the code calculates the mean
for any of the possible random inputs.

```

```

1 void proof_harness_mean() {
2     int size;
3     __CPROVER_assume(size > 0);
4     int *array = (int *)malloc(size * sizeof(int));
5     __CPROVER_assume(array != NULL);
6
7     for (int i = 0; i < size; i++) {
8         array[i] = nondet_int(); // Assign a non-deterministic value
9         __CPROVER_assume(array[i] >= 0 && array[i] <= 100); // Now assume the range
10    }
11
12    int sum = 0;
13    for (int i = 0; i < size; i++) {
14        sum += array[i];
15    }
16    float expected_mean = (float)sum / size;
17
18    calculateMean(array, size);
19
20    free(array);
21 }

```

Listing 4.8: Generated Proof Harness Function

Only one iteration was needed to generate the proof harness. CBMC passed with a successful verification under the following function call:

```

1 cbmc mean.c --unwind 10 --function proof_harness_mean

```

Listing 4.9: CBMC Call for Ex 4

Chapter 5

Critical Evaluation

Chapter 6

Conclusion

Bibliography

Appendix A

Appendix A: AI Prompts

Appendix B

An Example Appendix