CSPB 2270 - Summer 2023 Jones - Data Structures & Algorithms

Dashboard / My courses / 2234:CSPB 2270 / 3 July - 9 July / Exam1 - Programming Part (Remotely Proctored)

Started on	Friday, 7 July 2023, 2:43 PM
State	Finished
Completed on	Friday, 7 July 2023, 4:38 PM
Time taken	1 hour 54 mins
Grade	**44.40** out of 50.00 (**89**%)

Question **1**

Correct

Mark 2.00 out of 2.00

Following function *Foo()* implements a linked list of size 4 and return a pointer to the top of it. if you press the "check" button below the code window you will see that it doesn't pass some of the tests.

fix the code such that it passes all the tests. (you can fix it by adding one line of code). For this, you need to read the given code in the test to figure what the test expects.

Note: press Precheck to check your answer against test cases and eventually press Check to submit your answer to moodle.

Answer: (penalty regime: 0 %)

Reset answer

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
struct list_node{
    int value;
    shared_ptr<list_node> next;
};

shared_ptr<list_node> Foo(){
    // create heap variables
    shared_ptr<list_node> top(new list_node);
    shared_ptr<list_node> two(new list_node);
    shared_ptr<list_node> six(new list_node);
    shared_ptr<list_node> nine(new list_node);

    // assign values
    top->value = 1;
    two->value = 2;
    six->value = 6;
    nine->value = 9;
```

	Test	Expected	Got	

✓	`shared_ptr<list_node> t = Foo();`
	`cout << t->value << endl;`
		1	1	✓

✓	`shared_ptr<list_node> t = Foo();`
	`cout << t->next->value << endl;`
		2	2	✓

✓	`shared_ptr<list_node> t = Foo();`
	`cout << t->next->next->value << endl;`
		6	6	✓

✓	`shared_ptr<list_node> t = Foo();`
	`cout << t->next->next->next->value << endl;`
		9	9	✓

✓	`shared_ptr<list_node> t = Foo();`
	`if(!t->next->next->next->next)`
	`  cout << "Tail is NULL" << endl;`
	`else`
	`  cout << "Failed" << endl;`
		Tail is NULL	Tail is NULL	✓

Passed all tests! ✓

Correct

Marks for this submission: 2.00/2.00.

Question **2**

Correct

Mark 6.00 out of 6.00

- The following is a recursive `Size` function for Binary Search trees. It contains two bugs that cause it to return the wrong number.
- Identify what the bugs are and fix the bugs so that it returns the correct tree size.
- Test cases for this question are hidden on purpose and you will be informed only whether function passes or fails the tests.
- Don't forget about corner cases

Note: some tests are hidden on purpose.

Answer: (penalty regime: 0 %)

Reset answer

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
// binary tree node
struct bst_node {
    int value;
    shared_ptr<bst_node> left;
    shared_ptr<bst_node> right;
};

int Size(shared_ptr<bst_node> t) {
    if(t == nullptr) { // means if(t==NULL)
        return 0;
    } else {
        return 1 + Size(t->left) + Size(t->right);
    }
}
```

	Test	Expected	Got	

✓	`shared_ptr<bst_node> root(new bst_node);`
	`shared_ptr<bst_node> five(new bst_node);`
	`shared_ptr<bst_node> two(new bst_node);`
	`root->value = 1;`
	`five->value = 5;`
	`two->value = 2;`
	`root->right = five;`
	`five->left = two;`
	`// other leaves by default would be NULL`
	`int ret = Size(root);`
	`if(ret == 3){`
	`  cout << "PASS" << endl;`
	`} else {`
	`  cout << "FAIL" << endl;`
	`}`
		PASS	PASS	✓

Passed all tests! ✓

Correct

Marks for this submission: 6.00/6.00.

Question **3**

Partially correct

Mark 2.40 out of 4.00

Following recursive function should return the number of odd integers that are smaller than the input integer.

There is a bug that prevents the function from stopping correctly for certain inputs,

Fix the bug by modifying the lines marked **A and B** in the code

Note: (v%2) calculates modulus of variable v to 2 (remainder of v/2). for example

10%2 = 0

5%2 = 1

1%2 = 1

5%3 = 2

Note : All tests for this question are hidden on purpose.

Answer: (penalty regime: 0 %)

Reset answer

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
int num_odds_below(int v) {
  if (v == 0 || v % 2 == 0) { // A
    return 0;
  }
  if (v % 2 == 1 && v > 0) { // B
    return 1 + num_odds_below(v-1);
  }
  return num_odds_below(v-1);
}
```

	Test	Expected	Got	

Your code failed one or more hidden tests.

Partially correct

Marks for this submission: 1.60/4.00.

Question **4**

Correct

Mark 12.00 out of 12.00

Write a **Recursive Version** of the linked list `Size` function, using the provided function signature. Consider the stopping conditions, and be sure to cover the full range of valid inputs.

Answer: (penalty regime: 0 %)

Reset answer

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
struct list_node{
    int value;
    shared_ptr<list_node> next;
};

int Size(shared_ptr<list_node> n){
    // your code here
    if (n == nullptr) {
        return 0;
    }
    else {
        return 1 + Size(n->next);
    }
}
```

	Test	Expected	Got	

✓	`shared_ptr<list_node> top;`
	`top.reset();   //set it to NULL`
	`cout << Size(top) << endl;`
		0	0	✓

✓	`shared_ptr<list_node> top(new list_node);`
	`top->value = 10;`
	`top->next.reset();`
	`cout << Size(top) << endl;`
		1	1	✓

✓	`shared_ptr<list_node> top(new list_node);`
	`shared_ptr<list_node> three(new list_node);`
	`shared_ptr<list_node> five(new list_node);`
	`top->value = 1;`
	`three->value = 3;`
	`five->value = 5;`
	`top->next = three;`
	`three->next = five;`
	`five->next.reset(); // set it to null`
	`cout << Size(top) << endl;`
		3	3	✓

Passed all tests! ✓

Correct

Marks for this submission: 12.00/12.00.

Question **5**

Correct

Mark 16.00 out of 16.00

Write a function called `CheckInvariant` that returns true if an input binary tree satisfies the invariant of an unbalanced binary search tree, and false if there is a violation of the binary search tree invariant.

Answer: (penalty regime: 0 %)

Reset answer

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
struct bst_node{
    int value;
    shared_ptr<bst_node> left;
    shared_ptr<bst_node> right;
};

bool CheckInvariant(shared_ptr<bst_node> n){
    bool result = true;
    //your code here

    if (n != nullptr) {
        if (n->left != nullptr && n->right != nullptr) {
            if (n->left->value > n->right->value) {
                result = false;
            }
        }
        else if (n->left == nullptr && n->right != nullptr) {
            if (n->right->value > n->right->left->value) {
```

	Test	Expected	Got	

✓	`shared_ptr<bst_node> root;`
	`cout << CheckInvariant(root) << endl;`
		1	1	✓

✓	`shared_ptr<bst_node> root(new bst_node);`
	`root->value = 1;`
	`root->left = shared_ptr<bst_node>(NULL);`
	`root->right = shared_ptr<bst_node>(NULL);`
	`cout << CheckInvariant(root) << endl;`
		1	1	✓

✓	`shared_ptr<bst_node> root(new bst_node);`
	`shared_ptr<bst_node> four(new bst_node);`
	`shared_ptr<bst_node> ten(new bst_node);`
	`root->value = 1;`
	`four->value = 4;`
	`ten->value = 10;`
	`root->right = four;`
	`root->left = ten;`
	`// leaves will be NULL automatically`
	`cout << CheckInvariant(root) << endl;`
		0	0	✓

	Test	Expected	Got	

✓	`shared_ptr<bst_node> root(new bst_node);`
	`shared_ptr<bst_node> four(new bst_node);`
	`shared_ptr<bst_node> ten(new bst_node);`
	`root->value = 1;`
	`four->value = 4;`
	`ten->value = 10;`
	`root->right = ten;`
	`ten->left = four;`
	`// leaves will be NULL automatically`
	`cout << CheckInvariant(root) << endl;`
		1	1	✓

✓	`shared_ptr<bst_node> root(new bst_node);`
	`shared_ptr<bst_node> four(new bst_node);`
	`shared_ptr<bst_node> ten(new bst_node);`
	`root->value = 1;`
	`four->value = 4;`
	`ten->value = 10;`
	`root->right = four;`
	`four->left = ten;`
	`// leaves will be NULL automatically`
	`cout << CheckInvariant(root) << endl;`
		0	0	✓

✓	`shared_ptr<bst_node> root(new bst_node);`
	`shared_ptr<bst_node> four(new bst_node);`
	`shared_ptr<bst_node> ten(new bst_node);`
	`root->value = 1;`
	`four->value = 4;`
	`ten->value = 10;`
	`root->left = ten;`
	`ten->left = four;`
	`// leaves will be NULL automatically`
	`cout << CheckInvariant(root) << endl;`
		0	0	✓

Passed all tests! ✓

Correct

Marks for this submission: 16.00/16.00.

Question **6**

Partially correct

Mark 6.00 out of 10.00

- Complete the function `ToTree` bellow that reads data from a linked-list and stores them into a binary search tree.
- The function signature is given in the code `shared_ptr<bst_node> ToTree(shared_ptr<list_node> top)` which gets a pointer to top of the linked list and eventually returns a pointer to root of the BST
- On exit, the returned binary tree must contain all items that the input linked list contains.
- You may define helper functions if you like.
- Make sure you obey the invariants of BST.

Answer: (penalty regime: 0 %)

Reset answer

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
void my_custom_func(shared_ptr<bst_node> a_node,int an_int,float a_float){

}

// Edit and complete ToTree function bellow. Feel free to add additional functions if you need.
shared_ptr<bst_node> ToTree(shared_ptr<list_node> top){
    shared_ptr<bst_node> bstroot(new bst_node);

    // dynamically create nodes for BST and insert data from linked-list here
    shared_ptr<list_node> current = top;
    if (top == nullptr) {
        bstroot = shared_ptr<bst_node>(NULL);
    }
    else {
        shared_ptr<list_node> previous(new list_node);
        shared_ptr<list_node> newTop(new list_node);
        while (current != nullptr) {
            if (bstroot->value == 0) {
                newTop->value = current->value;
                bstroot->value = newTop->value;
            }
            else {
                if (current->value < newTop->value) {
                    shared_ptr<bst_node> newLeft(new bst_node);
                    newLeft->value = current->value;
                    bstroot->left = newLeft;
                }
                else {
                    shared_ptr<bst_node> newRight(new bst_node);
                    newRight->value = current->value;
                    bstroot->right == newRight;
                }
            }
            current = current->next;
        }
    }

    // finally return root of the tree
    return bstroot;
}
```

	Test	Expected	Got	

✓	`shared_ptr<list_node> top(new list_node);`
	`top = shared_ptr<list_node>(NULL);`
	`shared_ptr<bst_node> ret = ToTree(top);`
	`if(ret){`
	`  cout << "NOT NULL" << endl;`
	`} else {`
	`  cout << "NULL" << endl;`
	`}`
		NULL	NULL	✓

	Test	Expected	Got	

✓	shared_ptr<list_node> top(new list_node);
	top->value = 12;
	top->next = shared_ptr<list_node>(NULL);
	shared_ptr<bst_node> ret = ToTree(top);
	if((ret)&&(!ret->left)&&(!ret->right)&&(ret->value==12))
	{
	  cout << "PASS" << endl;
	} else {
	  cout << "FAIL" << endl;
	}
		PASS	PASS	✓

✗	shared_ptr<list_node> top(new list_node);
	shared_ptr<list_node> one(new list_node);
	shared_ptr<list_node> three(new list_node);
	top->value = 2;
	one->value = 1;
	three->value = 3;
	top->next = one;
	one->next = three;
	three->next = shared_ptr<list_node>(NULL);
	shared_ptr<bst_node> ret = ToTree(top);
	if((ret->value == 2)&&
	 (ret->left->value == 1)&&
	 (ret->right->value == 3)){
	  cout << "PASS" << endl;
	} else {
	  cout << "FAIL" << endl;
	}
		PASS	***Error***
			Segmentation fault (core dumped)
				✗

Testing was aborted due to error.

Show differences

Partially correct

Marks for this submission: 4.00/10.00.