# CSc 352

# Binary Tree with Structs

Benjamin Dicken

# Linked List

Previously build linked list with structs

```
typedef struct ListNode {
  int value;
  struct ListNode* next;
} ListNode;
```

What would a Binary Tree node look like?

# Binary Tree

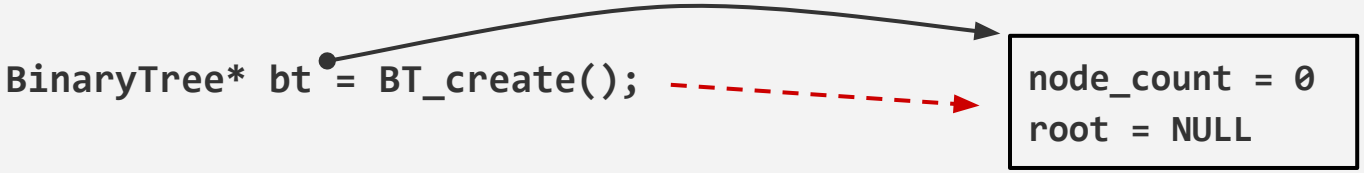
Previously build linked list with structs

```c
typedef struct BinaryTreeNode {
  char* value;
  struct BinaryTreeNode* left;
  struct BinaryTreeNode* right;
} BinaryTreeNode;
```

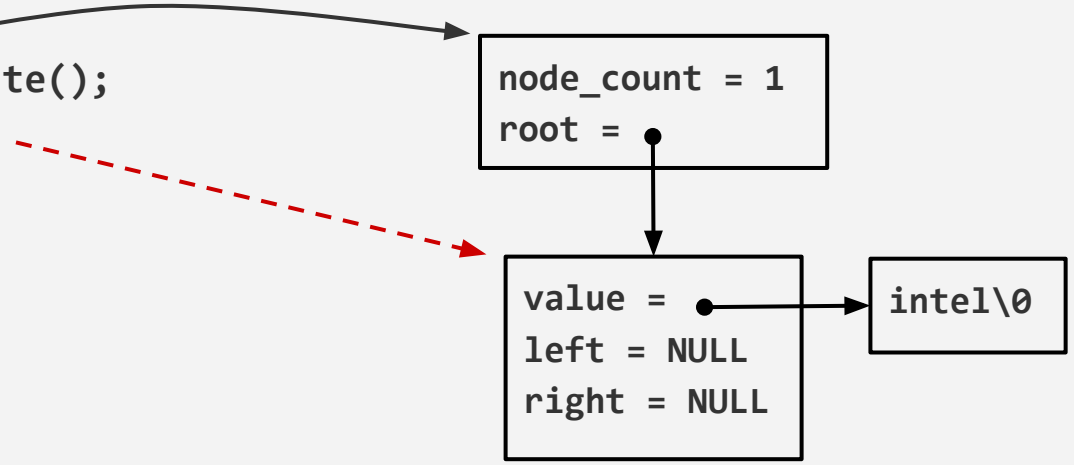
```c
typedef struct BinaryTree {
  int node_count;
  struct BinaryTreeNode* root;
} BinaryTree;
```

Generic value pointer

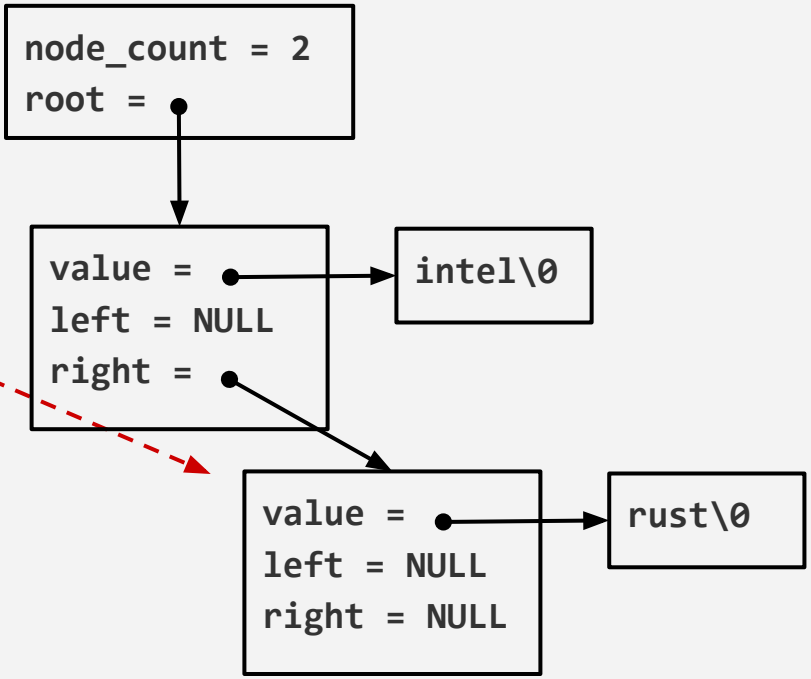
```
BinaryTree* bt = BT_create();
```


BinaryTree* bt = BT_create();
node_count = 0
root = NULL

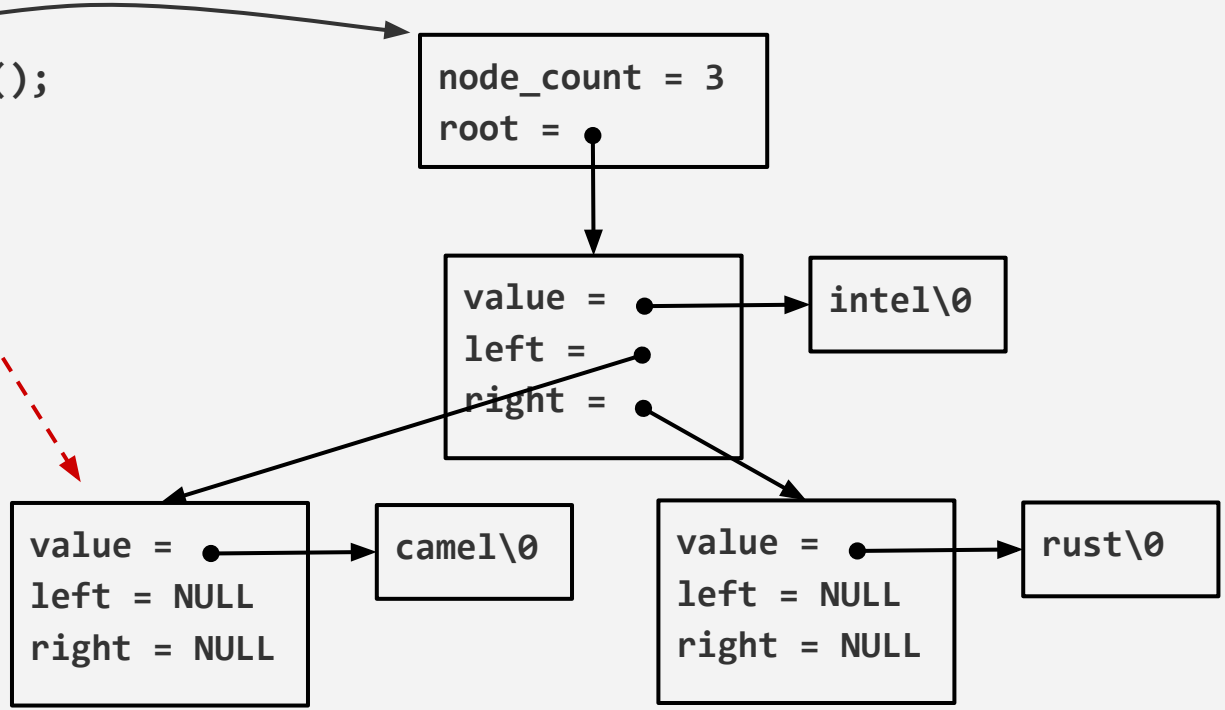
```
BinaryTree* bt = BT_create();
BT_insert(bt, "intel");
```



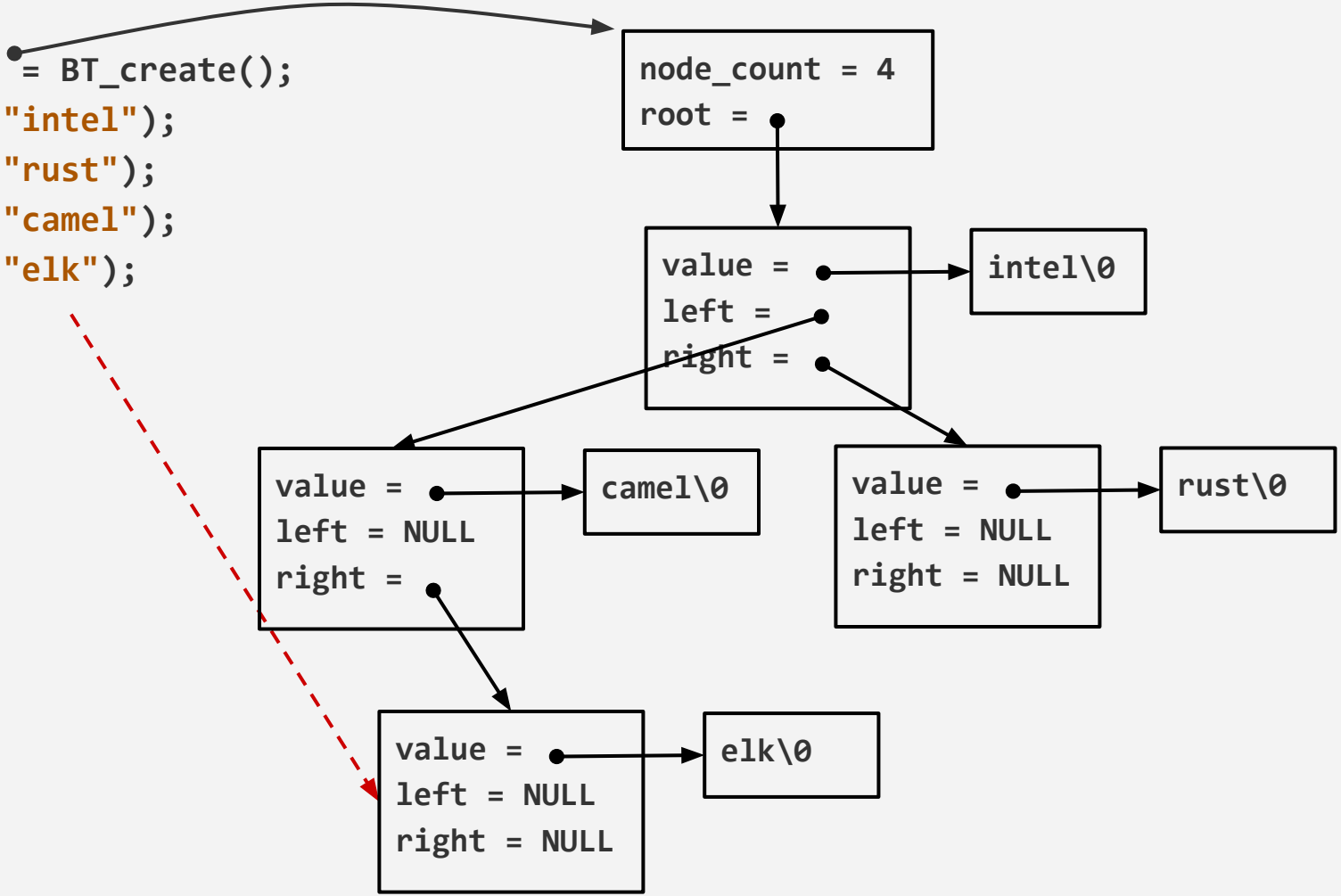
```
BinaryTree* bt = BT_create();
BT_insert(bt, "intel");
BT_insert(bt, "rust");
```



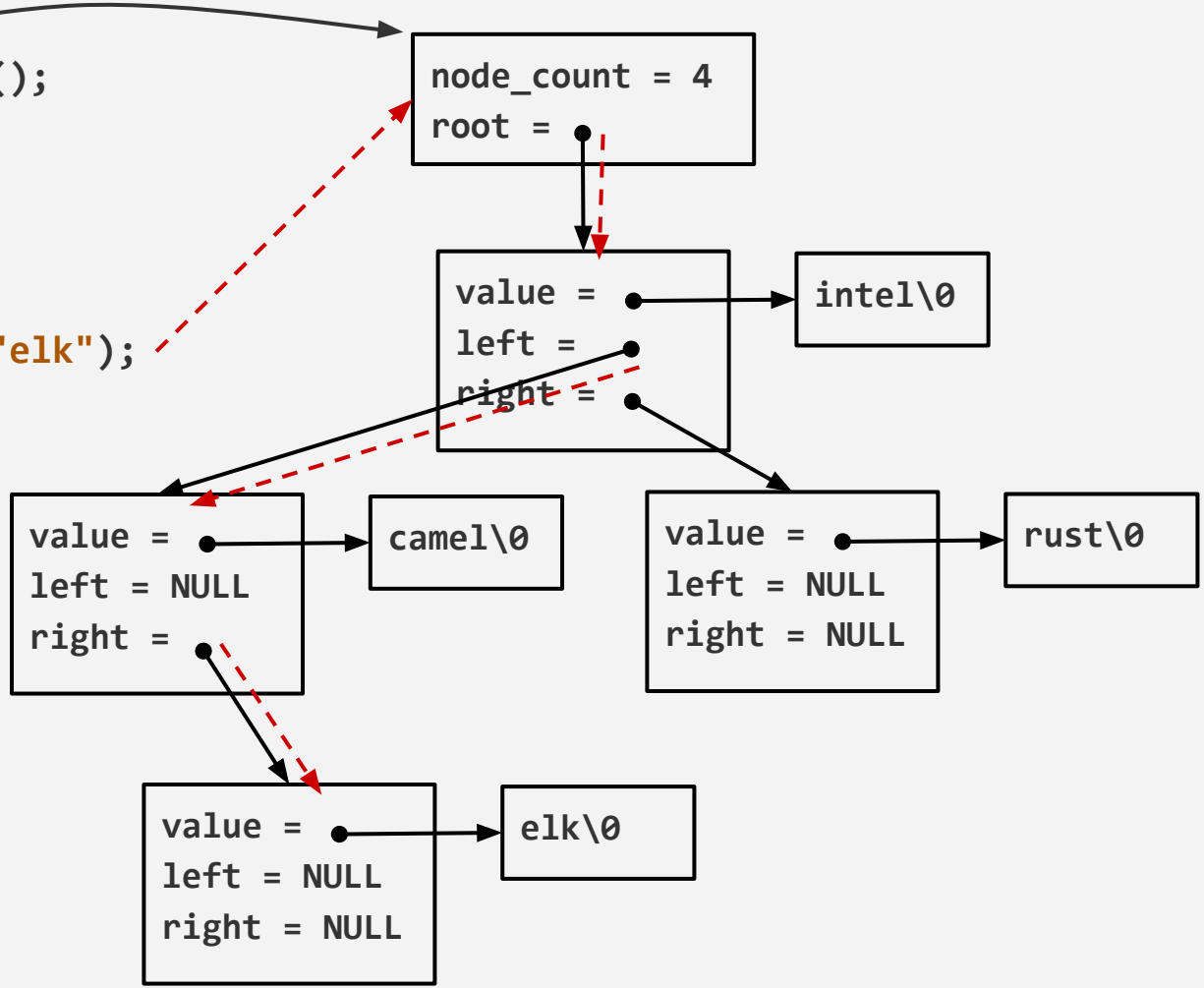
```
BinaryTree* bt = BT_create();
BT_insert(bt, "intel");
BT_insert(bt, "rust");
BT_insert(bt, "camel");
```



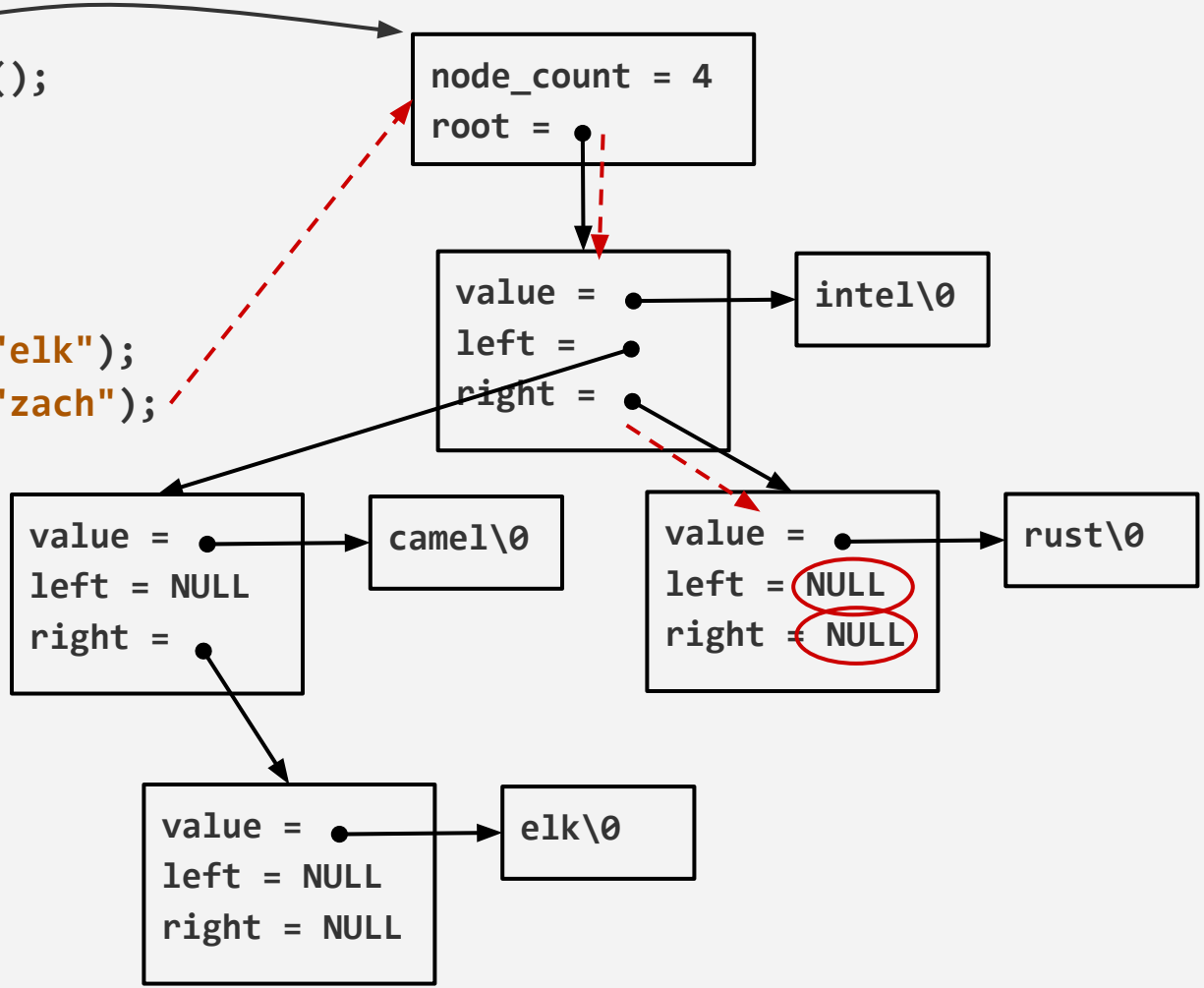
```
BinaryTree* bt = BT_create();
BT_insert(bt, "intel");
BT_insert(bt, "rust");
BT_insert(bt, "camel");
BT_insert(bt, "elk");
```



```
BinaryTree* bt = BT_create();
BT_insert(bt, "intel");
BT_insert(bt, "rust");
BT_insert(bt, "camel");
BT_insert(bt, "elk");
bool c1 = BT_contains(bt, "elk");
```



```
BinaryTree* bt = BT_create();
BT_insert(bt, "intel");
BT_insert(bt, "rust");
BT_insert(bt, "camel");
BT_insert(bt, "elk");
bool c1 = BT_contains(bt, "elk");
bool c2 = BT_contains(bt, "zach");
```

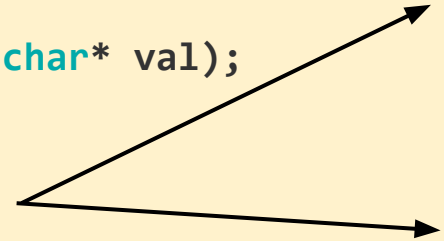


# Implement the Binary Tree

Implement:

```
BinaryTree* BT_create();
void BT_insert(
    BinaryTree* bt, char* val);
bool BT_contains(
    BinaryTree* bt, char* val);
```

Use these structs



```
typedef struct BinaryTreeNode {
  char* value;
  struct BinaryTreeNode* left;
  struct BinaryTreeNode* right;
} BinaryTreeNode;

typedef struct BinaryTree {
  int node_count;
  struct BinaryTreeNode* root;
} BinaryTree;
```

# Implement `BT_create`

```
BinaryTree* BT_create() {
    // What goes in here?
}
```

# Implement BT_create

```
BinaryTree* BT_create() {
  BinaryTree* bt = calloc(sizeof(BinaryTree), 1);
  return bt;
}
```

Activity

# Implement BT_insert

```
void BT_insert(BinaryTree* bt, char* val);
  // What goes in here?
}
```

# Implement BT_insert

```
void BT_insert(BinaryTree* bt, char* val) {
  if (bt->root == NULL) {
    node->root = BT_create_node(val);
    return;
  }
  BT_insert_helper(bt->root, val);
}
```

# Implement BT_insert

```
void BT_insert_helper(BinaryTreeNode* node, char* val) {
  int r = strcmp(val, node->value);
  if (r < 0) {
    if (node->left != NULL) {
      BT_insert_helper(node->left, val);
    } else {
      node->left = BT_create_node(val);
      return;
    }
  } else if (r > 0) {
    if (node->right != NULL) {
      BT_insert_helper(node->right, val);
    } else {
      node->right = BT_create_node(val);
      return;
    }
  }
}
```

# Implement BT_insert

```
BinaryTreeNode* BT_create_node(char* val) {
  BinaryTreeNode* n = calloc(sizeof(BinaryTreeNode), 1);
  n->value = malloc(strlen(val)+1);
  strcpy(n->value, val);
  return n;
}
```

Activity

# Implement `BT_contains`

```
bool BT_contains(BinaryTree* bt, char* val) {
    // What goes in here?
}
```

# Implement BT_contains

```
bool BT_contains(BinaryTree* bt, char* val) {
  if (bt->root == NULL) {
    return false;
  }
  return BT_contains_helper(bt->root, val);
}
```

# Implement BT_contains

```
bool BT_contains_helper(BinaryTreeNode* node, char* val) {
  int r = strcmp(val, node->value);
  if (r < 0) {
    if (node->left != NULL) {
      return BT_contains_helper(node->left, val);
    } else {
      return false;
    }
  } else if (r > 0) {
    if (node->right != NULL) {
      return BT_contains_helper(node->right, val);
    } else {
      return false;
    }
  } else {
    return true;
  }
}
```

# Implement the Binary Tree

Implement:

```
BinaryTree* BT_create();
void BT_insert(
    BinaryTree* bt, char* val);
bool BT_contains(
    BinaryTree* bt, char* val);
bool BT_free(BinaryTree* bt);
```

Add free!

```
typedef struct BinaryTreeNode {
  char* value;
  struct BinaryTreeNode* left;
  struct BinaryTreeNode* right;
} BinaryTreeNode;

typedef struct BinaryTree {
  int node_count;
  struct BinaryTreeNode* root;
} BinaryTree;
```