# Storing Data in a Binary Tree

## Main Function

As the program has executed a frame is added to the stack when the main function is called. Within this frame contains one local variable called ‘root’, representing the root node of the binary tree. When the root node is instantiated, it is stored on the heap and the reference to the object on the heap is stored in the local variable ‘root’ in the main function.

The root node’s pointer is passed (line 198) to the function ‘display\_tree’ and as a result, the function ‘display\_tree’ is added to the stack with the pointer of the object stored as a local variable with reference to the object on the heap. This is process is repeated when the function ‘inOrder’ is called.

## Node Structure

The node is stored as a struct containing each attribute – module number, module year, module title, module semester, and the pointers for the left and right node. The struct’s instance variables are stored in contiguous memory locations in the order they are declared in.

For the root node, the address is as follows.

|  |  |
| --- | --- |
| modTitle | 0x7feba1401910 |
| modYear | 0x7feba1401920 |
| modSem | 0x7feba1401930 |
| modNumber | 0x7feba1401940 |
| left | 0x0 |
| right | 0x0 |

Shown above, each variable is contiguously assigned to memory with 10 bytes (8 bytes data + 2 bytes null character). However, the module number is assigned 4 bytes and the left and right pointer are assigned to a null pointer address.

The struct itself is stored on the heap when the object is instantiated, and the left and right pointers are references to other nodes stored on the heap.

## Read Data

Stored on the stack, the function ‘readData’ (lines 122-147) reads the data file and parses each line, separating the data. Line 125 dynamically allocates memory for each line using the malloc function and following this the file is opened and the root node is then set to null.

Next, line 127 through to 143 reads each line in the file until it reaches the 400th line or the end of the file is reached. Each line is then tokenized by commas and the pointer of each token is stored in an array of pointers. The advantage of this is optimisation, where the program does not have to create duplicate data and instead stores a reference to the memory location. However, this does not necessarily work for integers which are 4 bytes in size compared to the pointers which are 8 bytes (64-bit system). Therefore, pointers should only be applied to certain data types.

Following this, the pointer of the array is passed to the function ‘insertNode’. Once the program has inserted the last node, this function returns the pointer of the root node and is deallocated from the stack as the function exits on line 151.

## Insert a Node

The function ‘insertNode’ is the core of the program, as it determines the position of the new node by traversing the tree and comparing the current node to the new node. Line 49 tests the root node and if it is null, a new objected is instantiated to create the root node; otherwise, the tree is traversed to find the correct child node.

When the current node is visited, it is compared to the new value and if the new value is lower, the current node is changed to the current left pointer. If the new value is higher, the current node is changed to the current right pointer.

As the new position is found, the loop breaks, and the pointer of the current node is set to the newly created node. To create the new node, the pointer of the data array is passed to the function ‘createNode’.

## Create a Node

Memory is dynamically allocated for the new node and this is stored on the heap. This node is referenced through the parent nodes’ pointer. The new node is then tested for a null value if so, the program is out of memory and the program exits.

Following this, each attribute is assigned their corresponding value/address from the array of pointers. This new node’s pointer is then returned to the previous function.

## Display Tree

Within the main sequence, the tree is displayed by passing the reference of the root node to the function ‘display\_tree’. This function visits all nodes and outputs the data in the following format.

Key (Left:Key)(Right:Key)

One by one, each node’s pointer is recursively passed to the function ‘display\_tree’ until each node has been visited. As the function is called, a new frame is allocated to the stack storing the reference to the currently visited node. With the given data, the stack will have an additional eighteen frames (18 nodes) until they are deallocated when the function exits.

Within C and any other language, there is a chance of a stack overflow error; however, for such a small program this would require a vast number of nodes. The program itself is optimised for this where the pointers store a reference instead of the actual value. This reduces the amount of memory contained in the frame.

## Traverse and Write Data

The last two functions ‘inOrder’ and ‘writeData’ work one after the other to output the data in sorted order to a file.

In-order traversal is used to sort the data and as the function recursively calls itself, a frame is allocated to the stack with the reference of the current node stored as a local variable. In this algorithm, the left subtree is visited, then the root node, followed by the right subtree.

Before the data is appended to a file, the data is formatted. The module number is cast to an array of characters and the data is concatenated to a character array with allocated memory space. This is then written to the last line of the file.

## Memory Overview

### Stacks

Frames are allocated to the stack as the methods are called and are deallocated when the method exits. Stacks are much faster but are limited in size and threadsafe.

### Heap

Allocated when objects are created and deallocated when a garbage collector recognises no link to the object. Heaps are much slower but are not limited in size and are not threadsafe.

### Malloc Function

An important note, when the malloc function is used to allocate memory, it allocates it to the heap and not the stack.

# Bibliography

Guest, A. (2019) 01a - *Memory* [Lecture]. 2CB101 Programming 03. Cordukes 102, York St John University, York [23rd September].

Guest, A. (2019) 01c - *C* [Lecture]. 2CB101 Programming 03. Cordukes 102, York St John University, York [23rd September].

Guest, A. (2019) 02a - *Memory* [Lecture]. 2CB101 Programming 03. Cordukes 102, York St John University, York [30rd September].

Guest, A. (2019) *C Pointers* [Lecture]. 2CB101 Programming 03. Cordukes 102, York St John University, York [21st October].

Guest, A. (2019) *C Pointers 2* [Lecture]. 2CB101 Programming 03. Cordukes 102, York St John University, York [28th October].