Programming 3
Assignment Part 1
Report

Tama Waddell S2626286

Date: Fri May 13

A virtual heap used to allocate and deallocate memory. The aim of my program was simplicity. It is rather short and easy to read with detailed commenting and complete testing.

My program uses a linked list of every node in the heap. Each node contains the size allocated to it, a boolean value to specify if it is used or not and a pointer to the next node. The nodes are sorted by location which occurs when they are created. Deleted nodes are then linked up with the previous or next node in the list. I have also included a tail node which points to the last node on the heap for easy calculations.

When the program starts it creates the first node which contains the size of the entire heap minus itself. The tail node will always contain the left over size if no other empty node can fulfil the request of the user. The tail is continual updated and serves as a permanent locator to add new nodes to the available space at the end of the heap.

The user can call the function vnew which will then check if any existing nodes can be reused, if not it will create a new node an update the tail. To delete the node the user can call the vdelete function which will locate any free nodes behind and in front of it and merge them into a large node. The function vcnew performs the same as vnew except zeros the values using another function named memset.

Other functions include vsizeof and printHeapState. Vsizeof reports the size of the memory block by simply casting back the size of the node and returning the size member. PrintHeapState displays the status of the heap by traversing through the list of nodes and outputting a character for each byte. 'I' for the size of the node and then outputting 'D' if it is used and 'F' if it isn't.

The complications of this program included merging with updating the tail and maintaining the size when reusing existing nodes. I had to make sure when deleting nodes at the end and merging with the tail, that the size of the next free space is reported correctly. When reusing nodes that contain a bigger size than requested I had to retain the larger value so we don't lose any bytes.

Pointer arithmetic could of been used everywhere to calculate the size but I chose to put the actual size in the node even if it's not the size requested to make the code easier to read.

With the testing I have used assert statements to test if the memory holds the correct values and I have given lots of examples for testing the allocating, deallocating, merging of the nodes with different sizes and clear print statements about what's happening for each test.

By Tama Waddell