I use a memory block type of structure implemented through linked lists to manage the memory. Each block contains the start address, size, process ID, and pointer to the next block.

createBlock function is a helper function that allocates memory and creates a new block. It dynamically allocates memory for a new block, sets all the provided values, initializes the next pointer to NULL, and returns the new block.

Status function goes over the linked list of memory blocks and prints out the current state of memory allocation. For each block, it displays the address range and whether it's allocated to a process or is a hole. I also keep running totals of allocated and free memory, which it displays at the end.

Allocate function implements the three memory allocation strategies - [F B W]. It first searches through the memory blocks to find an appropriate hole based on the specified strategy - first-fit takes the first suitable hole, best-fit takes the smallest suitable hole, and worst-fit takes the largest suitable hole.

Deallocate finds the specified process in memory and converts it back to a hole. After marking the block as a hole, I perform hole merging - if there are adjacent holes I combine them into a single larger hole by updating the size of one block and removing the other, ensuring memory doesn't become unnecessarily fragmented.

Compact reorganizes memory by moving all processes to the beginning of the memory space and combining all holes into one large hole at the end. First calculate new addresses for all processes, then create a new list with all processes moved to their new locations, then add a single hole at the end that combines all the previous holes. Basically, defragmenting.

I updated main to call all of these.

First we need to gcc the starter-code, then the object code needs to be called with the total memory size.