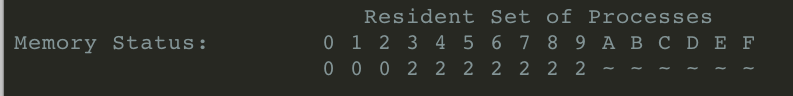
# Memory Manager

Author: Zachary Clute

## Functions

* allocate(pcb\*)
  + Inputs: pcb\*
  + Outputs: void
  + Purpose: To assign memory for a new process
* deallocate(pcb\*)
  + Inputs: pcb\*
  + Outputs: void
  + Purpose: To remove memory from a given process
* getFreeAmount()
  + Inputs: void
  + Outputs: int
  + Purpose: Returns how many frames of memory are free for use.
* print()
  + Inputs: void
  + Outputs: void
  + Purpose: Displays the current memory layout looks like in terms of what frame is assigned to what process
* debug()
  + Input: void
  + Output: void
  + Purpose: prints debug information to the screen

## Data Structures Used

* pcb\* memory[16]
  + Purpose: an array of size 16 where each cell holds a pcb pointer. Each cell is essentially one frame of memory where each frame is 1MB
* int memorySize
  + Purpose: memorySize is a constant which holds the total system memory (in terms of megabytes) and is always 16
* Int freeMem
  + Purpose: freeMem is an integer which holds the amount of free memory in megabytes. It changes when a process gets allocated or deallocated.

## High-Level Description

The purpose of this class is to hold and manipulate a simulation of memory for our simulated operating system. Every process which moves from new to ready must first pass through the memory manger and be allocated where as every process which is blocked or exited must be passed through the memory manager and deallocated. This ensures that memory is accurately simulated to how a real operating system would frame memory.

## Low-Level Description

A MemoryManager object is initially created with its constructor which will initialize the memory layout. The constructor begins by looping through the memory[16] array and setting each cell to NULL which will identify that that cell as not assigned to any process.

Once initialization is complete the object can be used to manipulate its internal memory array by passing in pointers of type pcb\* to the allocate method. The object will first check if there’s enough memory for allocation by comparing its freeMem int variable with \*pcb->ramNeeded. If freeRam is less than pcb\*->ramNeeded then allocation is allowed the the function will loop through the memory array and assign however many needed free cells to the pcb\* by placing the pcb pointer in the cell. Once all needed frames are assigned the function returns true to let whoever called the function know the operation completed successfully.

For deallocation a pcb pointer is passed to the deallocation method. The deallocation method will loop through the memory array and check each cell to see if that cell contains a pointer to the same pcb object. If it does, that cell is set to NULL and freeMem is incremented. This is done until the entire array is looped through. If the function at any point frees a cell of a pcb pointer then the function will return true to let the caller know that the process was freed successfully.