Virtual Memory- 2

In this lab, we will be completing our implementation of a virtual memory. For this purpose, you can use **the programming language of your choice**.

Step 2 – Virtual Memory

« Step 1 » refers to the previous lab.

At this point, we will be using « Step 1 » to build a full-fledged virtual memory using a one-level paging scheme.

We set the page size to 128 words/integers.

The size of the physical memory/RAM is 2^16 words.

The previous implementation is fromnow on the process logical space (mem is virtual) ; therefore we make the following changes to the implementation of « Step 1 » :

1. Change the definition of « mem_t » to

   ```
   typedef struct {
     // add here anything needed to manage v_mem
     // add the page table
   } mem_t;
   ```

   As you can see, « mem » has been removed as it bcomes now a VIRTUAL space

2. « myContWrite » and « myContRead » are irrelevant now and should be commented out
3. The new code that you should be able to run is the following :

```c
int main() {
  mem_t *mem = initContMem(); // create the memory space and intialize properly its management area
  address_t adr1 = myAlloc(mem, 5); // allocate 5 words from mem
  address_t adr2 = myAlloc(mem, 10); // allocate 10 words from mem
  address_t adr3 = myAlloc(mem, 100); // allocate 100 words from mem

  myFree(mem, adr2, 10); // release the 10 words allocated in line 14
  myFree(mem, adr3, 5); // release the 5 words allocated in line 13
  myWrite(mem, adr3, 543);  // write on the 1st word of the memory allocated in line 15
  myWrite(mem, adr3+9, 34); // write on the 10th word of the memory allocated in line 15

  int val1 = myRead(mem, adr3); // read the 1st word
  int val2 = myRead(mem, adr3+9); // read the 10th word
}
```

4. Create the physical memory/RAM as a global/static array of 2^16 words
5. Create any necessary data structures to manage the RAM ; this will become clear once you start implementing the function « myAlloc »

myAlloc(mem, nb_of_words)

1. Find « nb_words_to_allocate » of free/available contiguous words inside mem and declare them used
2. For each page used in 1
    a. find a free frame in physical memory a
    b. update the free list of physical frames accordingly
    c. update the mapping table (page table) to link the page to the frame
3. Return the base address of « 1 »

« 1 » and « 3 » in the above algorithm are exactly what « myContAlloc » does, so call it ! The only new part to write is « 2 ».

myWrite(mem, address, value)

1. Translate the virtual address « address » into a physical one using the page table of « mem »
2. Write « value » to the corresponding physical address

myRead(mem, address)

1. Translate the virtual address « address » into a physical one using the page table of « mem »
2. Read the content at index « address » from the RAM and return it

myFree(mem, address, nb)

That's the last function to write ; once you have written the previous functions, you will figure it out on your own !

Step 3 – Testing your code (Bonus)

To test your code, you could write multiple threads and each thread would be a simulation of a process with its own virtual memory, and then do some allocations, reads, writes, … and prove that no collisions whatsoever could occur between all these processes (ask me for hints).