# Programming Assignment 3

- Objective: to have hands-on experience of building a paging-based memory management system
  - It will be used as part of Programming Assignment 4
  - Assignment 4: students will implement page-replacement algorithms, compare the performance of algorithms
- Program description:
  - Implement functions
    - Initializing memory space
    - Allocating/deallocating memory space to process
    - Reading/writing data to process memory space
    - Printing out the memory contents of a process.
  - Define two lists: freeFrameList and processList
    - freeFrameList keeps track of free frames: simply a list of the frame numbers of free frames
    - processList keeps track of active processes (that are loaded in memory) and their sizes in terms of the number of allocated pages

- Program Requirements:
  - Implement the following functions to develop a prototype paging-based memory management scheme
  - Use C/C++ on POSIX-compliant operating systems
    - Alternatively: Python/C#/Java: please contact me

List of Functions:

- void memoryManager(int memSize, int frameSize):
  - Simulates creation and initialization of a physical memory space consisting of frames
  - Int memSize: specifies the size of the created memory space
  - Int frameSize: specifies the frame size of the created memory space in bytes. (For simplicity, fix the frame size to 1)
  - Initialize each frame with a value of 0
  - For example:
    - memoryManager(10, 1) will create memory space consisting of ten free fames. Each frame will be initialized with 0
  - These free frames will be allocated to processes
  - Create and initialize a freeFrameList that maintains and keeps track of free frames.
  - For example, if you created 10 frames with this function, the freeFrameList will have 10 entries where each entry is simply an array index to the array of free frames that you created

List of Functions:

- Int allocate(int allocSize, int pid):
  - Allocates a chunk of memory space of ‘allocSize’ bytes to a process with process ID pid.
  - When we allocate a memory space to a process, a set of pages are allocated to a process which are then mapped to available free frames.
  - Use the default size of 1 byte for simplicity
  - For example, a function call, allocate(10, 2) will allocate 10 pages to a process with process id 2.
  - These allocated pages are then mapped to free frames.

List of Functions:

- Int allocate(int allocSize, int pid):
  - Use the freeFrameList to find available free frames.
  - Part of these free frames are mapped with pages of a prcess
  - For the mapping between pages and free frames, use the random mapping method
    - A page is mapped to a randomly selected free frame from the list of available free frames
  - Create page table that records and maintains the mapping between the allocated pages
  - Refer to course slides to get an idea of how a page table is structured
  - The return value of this function is 1 if a requested memory space has been successfully allocated. If not, this function will return -1. (if there is not enough free frames)

List of Functions:

- Int allocate(int allocSize, int pid):
  - We assume that the page tables are not stored in the physical memory space we created (an array of free frames)
  - Let's assume that page tables just exist without taking a memory space for simplicity
  - Update freeFrameList and processList after allocating free frames to a process
  - For example, a function call allocate(2, 1) will allocate two pages to a process with pid 1. These two pages are randomly mapped to two free frames if available in the created memory space (see function memoryManager() ), and the two lists will be updates since part of the free frames are now allocated to a process, which in turn becomes an active process

- Int deallocate(int pid)
  - This function deallocates a memory space from a process with process ID pid.
  - Similar to the allocate() function, the freeFrameList and processList must be updated after deallocating a memory space from a process
  - This function returns 1 if successful, -1 if not successful

- Int write( int pid, int logical_address)
  - This function is used to write a value of ‘1’ to the memory space of a process with pid.
  - More specifically, a value of ‘1’ is written at the memory address specified by the parameter ‘logical_address’
  - Here the logical address is simply a page number
  - For example, a function call write(1,2) will write a value of ‘1’ at the memory location specified by the page number ‘2’ of a process with id ‘1’
  - A page number must be converted to the corresponding frame number in order to write a value of ‘1’
  - To find the corresponding frame number based on the provided page number, you need to use the page table of the process with pid
  - This function returns 1 if successful, and -1 if not successful

- Int read(int pid, int logical_address):
  - Similar to the write() function. The only difference is that this function is used to read a value from the memory location specified by the page number ‘logical_address’
  - Make sure to translate the page number into the corresponding frame number using the page table to read a value from the memory space
- Void printMemory(void)
  - This function prints out the physical memory space
    - freeFrameList
      - 1 2 6 (this indicates that frames 1, 2, and 6 are free frames in this example).
    - processList
      - 1 7 (this indicates that a value of ‘1’ is the pid, and ‘7’ is the process size, i.e., the number of pages allocated to this process)

- User Input:
  - Must be able to receive and handle the following user input
  - The input format is:
    - <Command><space><parameter1><space><parameter2>

M memorySice frameSize // create and initialize a physical meory sace

A size pid // allocate pages to a process with pid

W pid address1 // write a value of '1' to a process with pid at the location specified by 'address1'

R pid address1 // read from the memory space of a process with pid at the location specified by 'address1'

D pid // deallocate a memory space from a process with pid

P // print out the physical address space, the free frame list, and the process list

- Project Management Tool Requirement
  - Use your project management tool to keep track of changes that you make in all your files for this assignment
  - Screenshot of your project management tool that shows your file revision history
- What to turn in
  - Submit a zip file (filename: firstname_lastname.zip) containing the following files:
    - Your source and header files (e.g. c/c++ files)
    - All other files that are needed to run your program
    - A report in which you have to include: (1) a description of how to run your code, (2) a ‘brief’ description of your program design, and (3) a screenshot of your project management tool.
    - Submit your zip file to the dropbox on D2L.

- Evaluation Criteria
  - Documentation 10%
  - Compilation 15%
  - Project Management Tool 5%
  - Correctness 65%