## Classes

### **DProcess***(double hashing)*

| | | |
|---|---|---|
| -m_PID: int | -> | PID of the process |
| -m_page_index_start: int | -> | Index of physical memory array where the page starts |
| +getPID(): int | -> | Returns the value of the member variable “m_PID” |
| +getPhysicalIndex(ADDR: int): int | -> | Returns index of physical memory index of address |
| +Node(PID: int) | -> | initialize m_PID as PID and set member pointers as nullptr |

*Destructor is not required*

### **CProcess**(Chaining)

| | | |
|---|---|---|
| -m_PID: int | -> | PID of the process |
| -m_page_index_start: int | -> | Index of physical memory array where the page starts |
| -M_p_next: CProcess* | -> | Pointer of the next CProcess in the doubly linked list |
| -M_p_prev: CProcess* | -> | Pointer of the previous CProcess in the doubly linked list |
| +getPID(): int | -> | Returns the value of the member variable “m_PID” |
| +getPhysicalIndex(ADDR: int): int | -> | Returns index of physical memory index of address |
| +getPageIndexStart(): int | -> | Returns index of physical memory index of start of the page |
| +setNext(next: CProcess*): void | -> | Sets the the next CProcess in the doubly linked list |
| +setPrev(prev: CProcess*): void | -> | Sets the the previous CProcess in the doubly linked list |
| +setPhysicalIndex(page_index_start: int): void | -> | sets the m_page_index_start |
| +Node(PID: int) | -> | initialize m_PID as PID and set member pointers as nullptr |

*Destructor is not required*

### **ChainHashTable**

| | | |
|---|---|---|
| -m_p_memory_array: int* | -> | Array that conceptualizes physical memory |
| -m_p_process_array: CProcess** | -> | Hashmap as array of CProcess array using PID |
| -m_page_size: int | -> | Size of each page |
| -m_hash_size: int | -> | Size of hashmap |
| -m_p_process_page: int* | -> | Array that shows which page is assigned to which PID |

+checkPID(size: int): bool

-Checks if the PID already exists in the hashmap and if there are pages left to assign

+getFreePage(): int

-returns free page that has not been assigned yet

-Runtime: O(n): do a linear search to check if there is free page

+getProcess(PID: int): CProcess*

-returns CProcess that has the corresponding PID

-Runtime: O(n): use the hashing function and iterate through the doubly linked list until given PID is found

+createMemoryArray(memory_size: int, page_size: int): void

-allocate m_p_process_array, m_p_memory_array, m_p_process_page to the appropriate size

-assigns member variables appropriate value

-Runtime: O(1):

+search(PID: int): void

-finds index of the CProcess with the given PID in the hashmap

-Runtime: O(1): call getProcess() function which has runtime of O(1)

+insert(PID: int): void

-Allocate new CProcess with the given PID and add to the hashmap

-When allocated to the same hash key, add the CProcess to the doubly linked list in descending order of PID

-Runtime: O(1): if uniform hashing, simply put it through the hashing function and allocate CProcess

+write(PID: int, ADDR: int, x: int): void

-Write x into the physical memory index(m_p_memory_array)

-Runtime: O(1): if uniform hashing, there is only one node in the chain, so write the data in to the physical memory

+read(PID: int, ADDR: int): void

-Reads the integer stored in the corresponding PID and address

-Runtime: O(1): call getProcess() which has runtime of O(1)

+print(m: int): void

-prints the chain of PID stored in the m(index of the m_p_process_array) in descending order

-Runtime: O(1): CProcess is already stored in the chain in descending order, so simply traverse through them and print

*Constructor*: not required

*Destructor*: ~SimpleCalculator()

-Deallocate every CProcess that exists

-Deallocate m_p_process_array, m_p_memory_array, and m_p_process_page

## DoubleHashTable

| | | |
|---|---|---|
| -m_p_memory_array: int* | -> | Array that conceptualizes physical memory |
| -m_p_process_array: CProcess** | -> | Hashmap as array of CProcess array using PID |
| -m_page_size: int | -> | Size of each page |
| -m_hash_size: int | -> | Size of hashmap |
| -hashingFunc(PID: int, iter_index: int): int | | -> return hash value of the given PID and iter_index |

-getHashIndex(PID: int): int

-returns the hash_index if DProcess with the PID is found and returns -1 if not found in the hash map

-Runtime: O(1): if uniform hashing, the hash function with iter_index=0 will work right away

+createMemoryArray(memory_size: int, page_size: int): void

-allocate m_p_process_array, m_p_memory_array to the appropriate size

-assigns member variables appropriate value

-Runtime: O(1)

+search(PID: int): void

-finds index of the CProcess with the given PID in the hashmap

-Runtime: O(1): call getHashIndex() function which has runtime of O(1)

+insert(PID: int): void

-Allocate new CProcess with the given PID and add to the hashmap

-When collision occurs, use double hashing to resolve.

-Runtime: O(1): if uniform hashing, simply put it through the hashing function and allocate DProcess

+write(PID: int, ADDR: int, x: int): void

-Write x into the physical memory index(m_p_memory_array)

-Runtime: O(1): if uniform hashing, there is only one node in the chain, so write the data in to the physical memory

+read(PID: int, ADDR: int): void

-Reads the integer stored in the corresponding PID and address

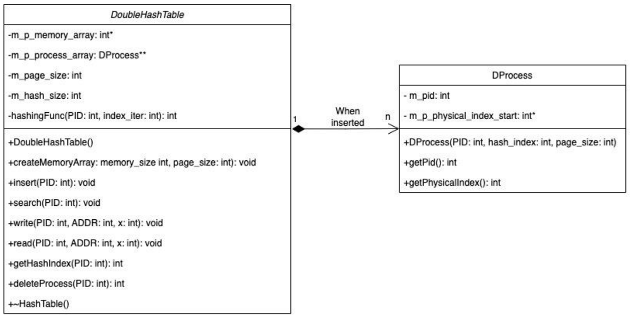
-Runtime: O(1): call getHashIndex() which has runtime of O(1)

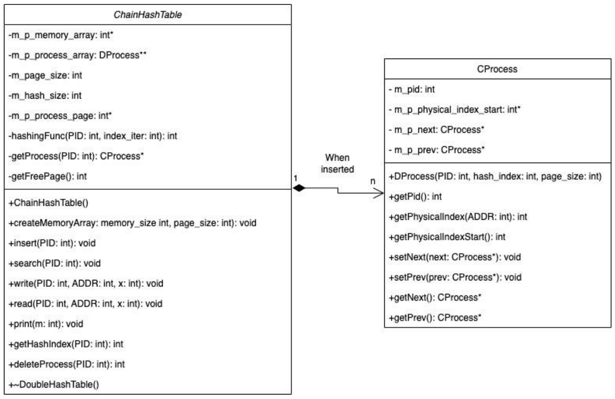
*Constructor*: not required

*Destructor*: ~SimpleCalculator()

-Deallocate every DProcess that exists

-Deallocate m_p_process_array and m_p_memory_array


DoubleHashTable
-m_p_memory_array: int*
-m_p_process_array: DProcess**
-m_page_size: int
-m_hash_size: int
-hashingFunc(PID: int, index_iter: int): int
+DoubleHashTable()
+createMemoryArray: memory_size int, page_size: int): void
+insert(PID: int): void
+search(PID: int): void
+write(PID: int, ADDR: int, x: int): void
+read(PID: int, ADDR: int, x: int): void
+getHashIndex(PID: int): int
+deleteProcess(PID: int): int
+~HashTable()
1
When inserted
n
DProcess
- m_pid: int
- m_p_physical_index_start: int*
+DProcess(PID: int, hash_index: int, page_size: int)
+getPid(): int
+getPhysicalIndex(): int


ChainHashTable
-m_p_memory_array: int*
-m_p_process_array: DProcess**
-m_page_size: int
-m_hash_size: int
-m_p_process_page: int*
-hashingFunc(PID: int, index_iter: int): int
-getProcess(PID: int): CProcess*
-getFreePage(): int
+ChainHashTable()
+createMemoryArray: memory_size int, page_size: int): void
+insert(PID: int): void
+search(PID: int): void
+write(PID: int, ADDR: int, x: int): void
+read(PID: int, ADDR: int, x: int): void
+print(m: int): void
+getHashIndex(PID: int): int
+deleteProcess(PID: int): int
+~DoubleHashTable()
1
When inserted
n
CProcess
- m_pid: int
- m_p_physical_index_start: int*
- m_p_next: CProcess*
- m_p_prev: CProcess*
+DProcess(PID: int, hash_index: int, page_size: int)
+getPid(): int
+getPhysicalIndex(ADDR: int): int
+getPhysicalIndexStart(): int
+setNext(next: CProcess*): void
+setPrev(prev: CProcess*): void
+getNext(): CProcess*
+getPrev(): CProcess*