Homework 4 (100 points) **Due date:** November 17, 2023 11:59 PM

Before you start…

Keep in mind these tips from your instructor:

- Watch the lectures in Module 6.2 before you start. Revise your notes and implementations completed in Module 5
- Read carefully the information presented in the instructions. You will be using custom classes that need to interact with each other and with Python's data types
- All methods that output a string must **return** the string, not **print** it. Code will not receive credit if you use print to display the output
- CacheList is a simple DoublyLinkedList that behaves according to the specifications given in the assignment. This means, methods that add values to the list must create an instance of the Node class first
- Eviction methods modify the list by adding or removing nodes from the list, revise the implementation of the LinkedList class in Module 5 if you need a starting point
- Ask questions using our Homework 4 channel in Microsoft Teams

REMINDER: As you work on your coding assignments, it is important to remember that passing the examples provided does not guarantee full credit. While these examples can serve as a helpful starting point, it is ultimately your responsibility to thoroughly test your code and ensure that it is functioning correctly and meets all the requirements and specifications outlined in the assignment instructions. Failure to thoroughly test your code can result in incomplete or incorrect outputs, which will lead to deduction in points for each failed case.

60% > Passing all test cases 40% > Clarity and design of your code

Section 1: What is a Cache? What are we making?

A computer cache is a part of your computer that holds data. It can be accessed very quickly but cannot hold as much information as your main memory.

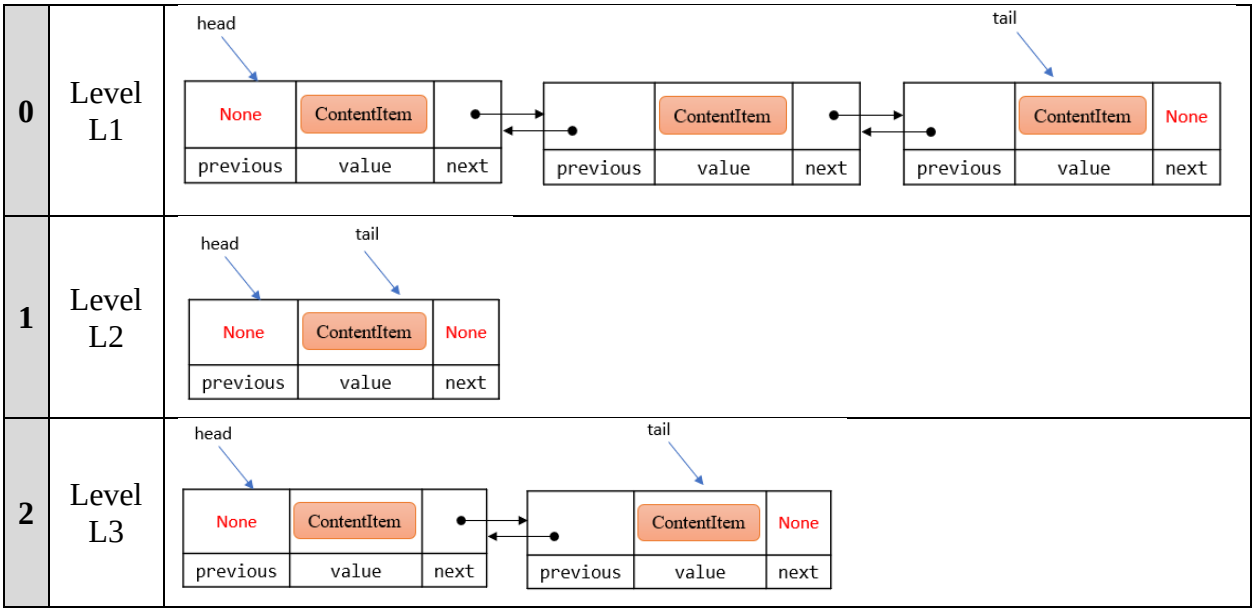
Most caches on a computer are split into different levels. Your computer will typically look for data starting at the top level (L1) and work its way down the different levels. The most frequently used information is kept in L1 cache because it's the fastest and the first level it will check.

The implementation of the cache for this assignment will be very simplified and would ignore some of the features of a cache you might find online – so make sure to read this assignment document carefully rather than searching for some "inspiration" online. Note that in this assignment we are not interested in the details of how a Cache works, instead, we are interested on using the data structures we have discussed to emulate a similar configuration to manipulate and search data.

In our implementation, we will have three different levels of cache (L1, L2, L3). The overall cache structure will be implemented as a hash table using separate chaining for collision resolution, with each individual level implemented as a doubly linked list. Content is put into different levels based on one of the content's attributes.

Visualization of the Cache, CacheLists, and ContentItems:

Cache Hash Table



Note that each bucket contains a Doubly Linked List. To receive credit for your code, each list must be properly linked backwards and forwards. The provided doctest is not checking for the integrity of the list from tail to head, it is your responsibility to test for that.

Section 2: The ContentItem and Node classes

The ContentItem class holds a piece of content and is described below. All methods have been implemented for you except __hash__. Do not modify the given code.

Attributes

Type	Name	Description
int	cid	Stores the content id.
int	size	Stores the size of the content as a nonnegative integer.
str	header	Information stored by the ContentItem (used for hash function later)
str	content	Information stored by the ContentItem

Special methods

Type	Name	Description
None	__init__(self, cid, size, header, content)	Creates a ContentItem from parameters
int	__hash__(self)	Returns the hash value for this ContentItem
str	__str__(self), __repr__(self)	String representation of this object
bool	__eq__(self, other)	Checks for equality with another object

__hash__(self) **(5 pts)**

Returns the hash value for this ContentItem (used by the Cache class). For this assignment, let the hash value be equal to the sum of every ASCII value in the header, modulo 3. This is the special method for the built-in method hash(*object*), for example hash('hello'). Hint: the ord(*c*) method could be helpful here.

Output	
int	An integer between 0 and 2 (inclusive), based on the hash function described above

The Node class has been implemented for you and is described below. Do not modify the given code.

Attributes

Type	Name	Description
ContentItem	value	Stores the value of this Node (always a ContentItem in this case)
Node	previous	Points to the preceding Node in the linked list (defaults to None)
Node	next	Points to the following Node in the linked list (defaults to None)

Special methods

Type	Name	Description
None	__init__(self, content)	Creates a new Node that holds the given ContentItem
str	__str__(self), __repr__(self)	String representation of this object

Section 3: The CacheList class

The CacheList class describes a single cache level in our hierarchy, implemented as a doubly linked list with references to the first and last node in the list (head and tail). Items are moved to the front every time they are added or used, creating an order in the list from most recently used to least recently used. **READ** the outline for all the methods in this class first, the *put* method should be the last one to be implemented in this class since it relies on the correctness of the other methods. **A portion of your grade in this class comes from your ability to reuse code by calling other methods.**

Attributes

Type	Name	Description
Node	head	Points to the first node in the linked list (defaults to None)
Node	tail	Points to the last node in the linked list (defaults to None)
int	maxSize	Maximum size that the CacheList can store
int	remainingSpace	Remaining size that the CacheList can store
int	numItems	The number of items currently in the CacheList

Methods

Type	Name	Description
str	put(self, content, evictionPolicy)	Adds Nodes at the beginning of the list
str	update(self, cid, content)	Updates the content in the list
None	mruEvict(self), lruEvict(self)	Removes the first/last item of the list
str	clear(self)	Removes all items from the list

Special methods

Type	Name	Description
None	__init__(self, size)	Creates a new CacheList with a given maximum size
str	__str__(self), __repr__(self)	String representation of this object
int	__len__(self)	The number of items in the CacheList
bool	__contains__(self, cid)	Determines if a content with cid is in the list

put(self, content, evictionPolicy) **(15 pts)**

5 pts for addition at the beginning of the list,
10 pts for complete functionality from the HashTable class and evictions

Adds nodes at the beginning of the list and evicts items as necessary to free up space. If the content is larger than the maximum size, do not evict anything. Otherwise, if the content id exists in the list prior the insertion, new content is not added into the list, but the existing content is moved to the beginning of the list. If there is currently not enough space for the content, evict items according to the eviction policy.

Input		
ContentItem	content	The content item to add to the list
str	evictionPolicy	The desired eviction policy (either ‘lru’ or ‘mru’)

Output	
str	‘INSERTED: *contentItem*’ if insertion was successful
str	‘Insertion not allowed’ if content size > maximum size
str	‘Content {*id*} already in cache, insertion not allowed’ if *id* is already present in the list

__contains__(self, cid) **(15 pts)**

Finds a ContentItem from the list by id, moving the ContentItem to the front of the list if found. This is the special method for the **in** operator to allow the syntax *cid* **in** *object*.

Input		
int	cid	The id to search for in the CacheList

Output	
bool	True if the matching ContentItem is in the list, False otherwise

update(self, cid, content) **(10 pts)**

Updates a ContentItem with a given id in the list. If a match is found, it is moved to the beginning of the list and the old ContentItem is entirely replaced with the new ContentItem. You cannot assume the size of the new content will be the same as the content in the list, thus, you must check that there is enough available space to perform the update. The update is not completed if the change results on exceeding the maxSize of the list, but the match is moved at the beginning of the list.

Input		
int	cid	The id to search for in the CacheList
ContentItem	content	The values to update the existing ContentItem with

Output	
str	‘UPDATED: *contentItem*’ if update was successful
str	‘Cache miss!’ is returned if no match is found or the update exceeds the maxSize

lruEvict(self) / mruEvict(self) **(10 pts each)**

Removes the last (least recently used) or the first (most recently used) item of the list

Output	
None	This function returns nothing.

clear(self) **(5 pts)**

Removes all items from the list. This method must unlink all nodes from the list

Output	
str	'Cleared cache!'

Section 4: The Cache class

The Cache class describes the overall cache, implemented as a hash table. It contains three CacheLists which actually store the ContentItems. Hash values of 0 correspond with the first CacheList (L1), 1 with L2, and 2 with L3.

0	**1**	**2**
Level L1	Level L2	Level L3
CacheList(size)	CacheList(size)	CacheList(size)

All methods in the Cache class will call a corresponding method in the CacheList class. For example, the *insert* method calls the *put* method from the CacheList class.

Do not change the initialization of the CacheList objects in the starter code. **You are not allowed to add any other methods in this class. Your code will not receive credit if you add other methods.**

Attributes

Type	Name	Description
list	hierarchy	List with 3 CacheList objects of size (lst_capacity)
int	size	Number of levels in our hierarchy (always set to 3)

Methods

Type	Name	Description
str	insert(self, content, evictionPolicy)	Adds an item into the proper cache list
str	clear(self)	Clears all CacheLists in the hierarchy.

Special methods

Type	Name	Description
None	__init__(self, lst_capacity)	Creates a new Cache with (3) CacheLists of size lst_capacity
str	__str__(self), __repr__(self)	String representation of this object
(various)	__getitem__(self, content)	Gets a Node from the proper cache list
None	__setitem__(self, cid, content)	Updates an item from the proper cache list

30 pts for correct status of the Hash Table after mixed calls do insert, getitem and setitem (no partial credit for this set of points)

insert(self, content, evictionPolicy)

Inserts a ContentItem into the proper CacheList. After using the hash function to determine which CacheList the content should go into, call that CacheList's put method to add the content.

Input		
ContentItem	content	The content item to add to the list
str	evictionPolicy	The desired eviction policy (either 'lru' or 'mru')

Output	
str	(Return the output from the put method call)

__getitem__(self, content)

Returns a reference to a Node that stores a ContentItem in a CacheList. After using the hash function to determine which CacheList the content should exist in, you may use the CacheList's **in** operator to find the proper node. This is the special method to support the syntax *object***[***content***]**

Input		
ContentItem	content	The content item to retrieve

Output	
Node	Reference to Node object with value that matches the ContentItem (item was found)
str	'Cache miss!' is returned if it's a cache miss (item was not found)

setitem(self, cid, content)

Updates a ContentItem. After using the hash function to determine which CacheList the content would be in, call that CacheList's update method to update the content.

Input		
ContentItem	content	The content item to update

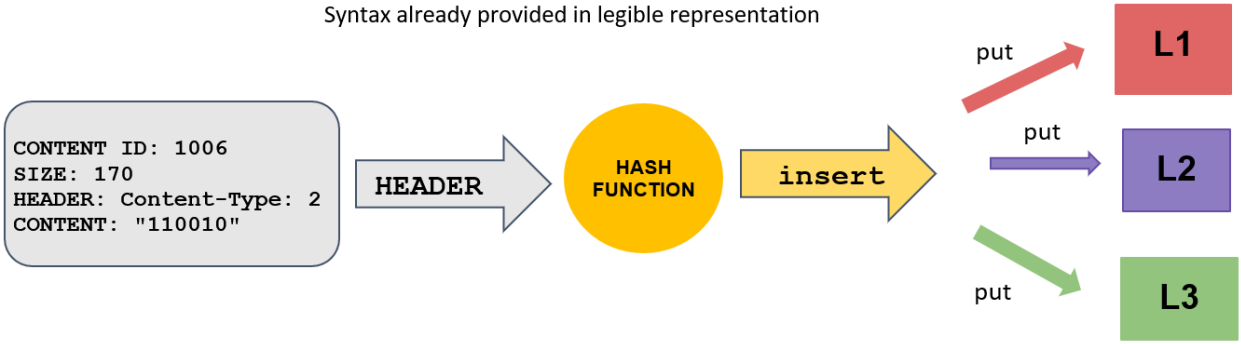
clear(self)

Clears all the lists in the hierarchy. This method is already implemented for you.

Output	
str	'Cache cleared!'

```
>>> content1 = ContentItem(1006, 170, "Content-Type: 2", "110010")
>>> cache.insert(content1, 'mru')
'INSERTED: CONTENT ID: 1006 SIZE: 170 HEADER: Content-Type: 2 CONTENT: 110010'
```

Syntax already provided in legible representation




```
>>> content1 = ContentItem(1006, 170, "Content-Type: 2", "110010")
>>> cache.insert(content1, 'mru')
'INSERTED: CONTENT ID: 1006 SIZE: 170 HEADER: Content-Type: 2 CONTENT: 110010'
>>> cache[content1]
CONTENT ID: 1006 SIZE: 170 HEADER: Content-Type: 2 CONTENT: 110010
```

.value

Syntax already provided in legible representation, just return the contentItem object


```
>>> content1 = ContentItem(1006, 170, "Content-Type: 2", "110010")
>>> cache.insert(content1, 'mru')
'INSERTED: CONTENT ID: 1006 SIZE: 170 HEADER: Content-Type: 2 CONTENT: 110010'
>>> content5 = ContentItem(1006, 170, "Content-Type: 2", "11111111111111")
>>> cache.updateContent(content5)
'UPDATED: CONTENT ID: 1006 SIZE: 170 HEADER: Content-Type: 2 CONTENT: 11111111111111'
```