Overview
In this project, you will design and implement a cache manager for a hierarchical storage system that stores and retrieves variable sized objects. The target system consists of two storage hierarchies: DRAM and magnetic disk. The objects are variable sized and reside on the magnetic disk drive permanently. When an object is referenced, your cache manager decides whether to cache a copy of it in DRAM. You may implement the GreedyDual algorithm detailed in [1] to decide whether an object should become cache resident and to select victims when the available cache space has been exhausted.

The objectives of this project are two folds. First, to demonstrate algorithms that consider size when deciding which data items to cache in main memory. Second, to introduce you to off-the-shelf storage managers for both research and development. This way you’ll become rich and famous architects sending us a note to let us know how valuable this project was to your careers.

Logistics and Deliverables
In this project, you have a choice of either working by yourself or with a teammate. We strongly encourage you to form a team because designs can become quite complex and you will benefit from participation of a team member to discuss alternatives.

To keep the project on track, we have come up with the following timetable for your deliverables:
2. April 9th: First implementation of V based on simple tests.
3. April 16th: Second implementation of V based on traces.
4. April 23rd: A competition to choose the fastest implementation.

A competition on April 23rd will make two awards. The first award is for an implementation that achieves the highest cache hit and byte hit ratios (each is weighted equally). The second award goes the implementation that processes the highest number of requests per second, i.e., highest throughput. Details of the competition will become available as we proceed forward.

Functionality
The V cache manager implements simple interfaces that resemble Berkeley DB. It supports the concept of multiple data-zone name containing (key, value) pairs. You may conceptualize a data-zone as either a relation or a set of (key, value) pairs. V supports a data-zone named “_V_Universal”. This data-zone name is used when the calling operation does not specify the name of a data-zone, see description of Insert method. The calling application may not create a data-zone named “_V_Universal”.
The Vdt data structure is the basis of the V cache manager. It consists of two properties:
1. Data is a character pointer to the specified byte-array,
2. Size denotes the size of the byte-array.

The specification for Vdt is as follows:
```cpp
class Vdt {
    char *data;
    int size;

public:
    Vdt() { data = NULL; size = 0; }

    void *get_data() const { return data; }
    void set_data(char *value) { data = value; }

    int get_size() const { return size; }
    void set_size(int value) { size = value; }
};
```

The data is a character pointer to the specified data with a pre-specified size stored in the “size” property.

All methods supported by V return an integer value. A negative return value indicates an error. A zero or a positive return value indicates successful execution of the method. The V method that you design and implement are:

**int Initialize(int MemSizeInGBytes, int MemSizeInBytes)**
This initializes V, invoking all your initialization methods. You may assume an application invokes this method prior to performing any other operations. This method may check to see if “_V_Universal” data-zone exists and create it if it does not exist. It assumes the size of DRAM is MemSizeInGBytes + MemSizeInBytes. For example, Initialize (1, 512*1024*1024) assumes the amount of available memory is 1.5 Gigabytes.

**int Shutdown()**
This method shuts-down V, performing all necessary graceful steps to ensure consistency of the database on the disk. For example, this method may write all dirty data (if any) from DRAM to disk prior to terminating.

**int Create(const Vdt& dzname)**
Creates a data-zone with the specified name. Assuming your implementation employs an instance of BDB to manage the content of the disk, you may maintain a database to keep track of all the data-zone names in the V storage manager. The name specified for a data-zone, i.e., dzname.get_data() return value, may not equal “_V_Universal” because this name is reserved as a default data-zone when the application does not specify a data-zone name.

**int GetNumberOfDataZones()**
Retrieves the number of data-zones registered with V. When the system first starts, an invocation of this method returns “1” because the “_V_Universal” data-zone exists.
int GetDataZoneNames(Vdt *dzNameArr, int *NumberOfDataZones)
Retrieves the name of all data-zones registered with V. The user provides an array for
this method to populate. The size of this array is specified in NumberOfDataZones. This
method populates the array elements with the names of all data-zones. If
NumberOfDataZones is less than the number of data zones registered with V then this
method returns an error code and stores the current number of data zones in
NumberOfDataZones. The application program may subsequently provide a larger array.
Note: This method should be called after invoking GetNumberOfDataZones(). One
reason why this method may fail is because of race conditions where a reader invokes
GetNumberOfDataZones and races with a Create method call by another thread.

int Insert(const Vdt *dzname, const Vdt& Key, const Vdt& Value)
Inserts the specified (Key, Value) pair into the specified data-zone name, dzname. If the
specified dzname is NULL, this method inserts the specified (Key, Value) pair into
"_V_Universal" data-zone. The Vdt data structure is used for dzname, Key, and Value.

int Delete(const Vdt *dzname, const Vdt& key);
Deletes the specified key from the target dzname. If the specified dzname is NULL, this
method deleted the specified key from the "_V_Universal" data-zone.

int Get(const Vdt *dzname, const Vdt& key, Vdt *Value)
Given a data-zone name, dzname, and a key, this method retrieves the corresponding
value. If no value exists then the data field of the Value is set to NULL and its size to
zero. If dzname is NULL, the specified key is retrieved from the "_V_Universal" data-
zone.

int DeleteAllKeys(const Vdt *dzname)
This method deletes all keys in the specified data-zone name, dzname. If dzname is
NULL, this method deletes all keys from the "_V_Universal" data-zone.

int DeleteInAllDataZones(const Vdt *KeyArray, const int size)
Given an array of keys, this method deletes each specified key from every data-zone
name registered with V including the "_V_Universal" data-zone.

References