UNIVERSITY OF ATHENS

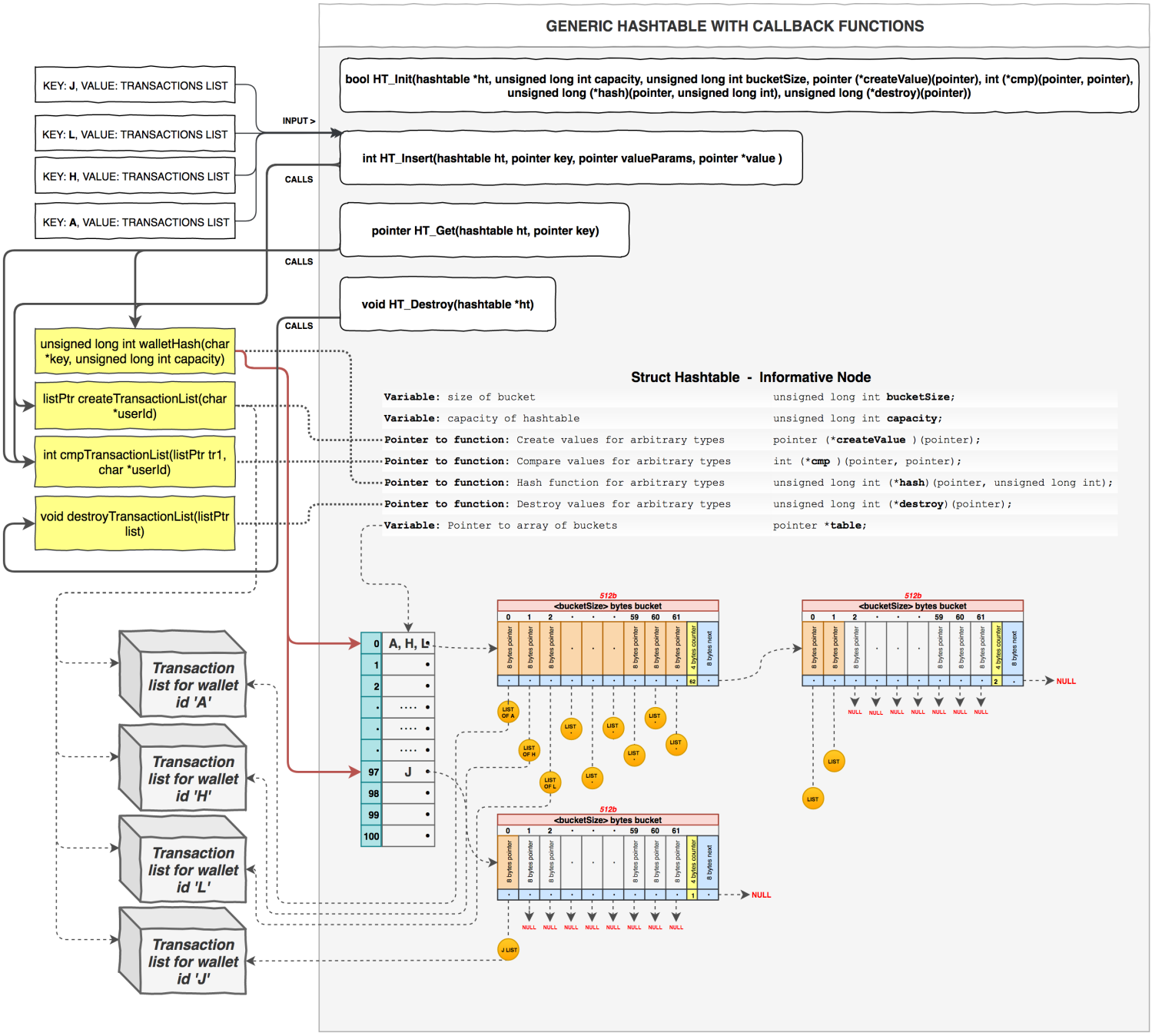
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

K24 - UNIX SYSTEM PROGRAMMING

1ST ASSIGNMENT

-TRANSACTIONS SIMULATION WITH BITCOINS-

R.N.: 1115201300202 – Konstantinos Chatzopoulos



Representation of the basic structures that have been implemented for the needs of the project.

Assumptions

Generic Hashtable

For the needs of the project, I've developed a generic hash table which allows to hash and store pointers of any type of data for quick access. To make the hash table work for every different type of data, the client/programmer is necessary to define in some places of their code, their own hash function and then pass it in HT with the help of the **HT_Init** function.

Every time that the hash table needs to call the **hash** function, it calls the client's/programmer's defined hash function with **key** & **capacity** parameters. Furthermore, it's necessary to pass extra function pointers that undertake to **create, compare** & **delete** the items that were inserted in the hash table.

To initialize the hashtable, the HT_Init function is being called with the following arguments:

bool HT_Init(Hashtable ***ht**, unsigned long **capacity**, unsigned long int **bucketSize**, void *(***createValue**)(void *), int (***cmp**)(void *, void *), unsigned long (***hash**)(void *, unsigned long int), unsigned long (***destroy**)(void *));

Functionality & Structure:

The HT consists of a dynamic index table where each pointer leads to buckets of dynamic size. The sizes are defined in the initialization (HT_Init) and do not change during run time. For that reason, collisions are inevitable due to full buckets. So, if a particular bucket is filled, an overflow bucket will be created as shown in the figure above.

As a contract, a bucket always holds a counter for the occupied slots as well as a pointer at the end which initially has the **NULL** value and serves to point to a possible overflow bucket.

In addition, each slot of the bucket, holds a **size (void *)** pointer that indicates the actual data of the corresponding data type.

Thus, the hash table is capable of handling every type of data. For the needs of the project, the used data types are the following:

```
typedef struct Wallet {
    char *userId;
    List bitcoins;
    unsigned long int
balance;
} *Wallet;
```

```
struct BitCoin {
    unsigned long int bid;
    bcNode root;
    unsigned long int nodes;
};
```

```
struct List {
    pointer identifier;
    nodePtr start;
    nodePtr current;
};
```

```
typedef struct Transaction{
    char *transactionId;
    char *senderWalletId;
    char *receiverWalletId;
    unsigned long int value;
    time_t timestamp;
} *Transaction;
```