University of athens

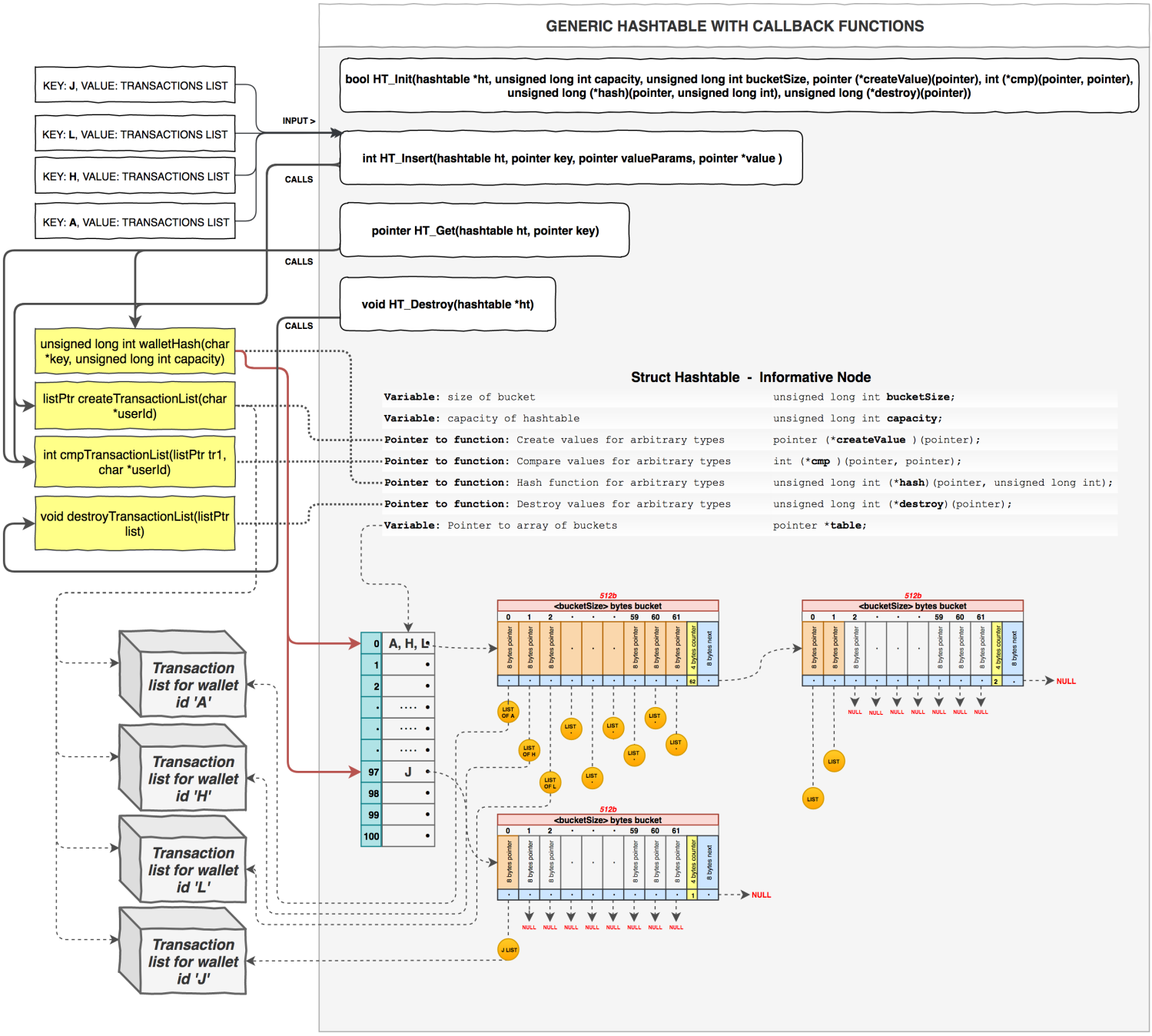
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

K24 - UNIX SYSTEM PROGRAMMING

1ST ASSIGNMENT

-TRANSACTIONS SIMULATION WITH BITCOINS-

R.N.: 1115201300202 – Konstantinos Chatzopoulos



Representation of the basic structures that has implemented for the needs of project.

Assumptions

Generic Hashtable

For the needs of the project, I've develop a general hash table where it undertakes to hash and store pointers of any type of data for its quick retrieve. To make the hash table that works with every different type of data, the client/programmer is necessary to define in some place of their code, their own hash function and then pass it in HT with the help of the **HT_Init** function.

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

To perform a hashtable initialize, just call HT_Init function 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 doesn't change during runtime. For that reason, it is inevitable many collisions that has re result of full buckets. So, if a particular bucket is filled, an overflow bucket should be created as shown in the figure above.

As a contract, a bucket it always holds a pointer at the end where it initially has the **NULL** value and serves to point to possible overflow bucket, as well as a count of the occupied positions. In addition, each slot of bucket, holds a **size (void *)** pointer that indicating the actual data of the corresponding data type.

Thus, the hashtable is capable of being used for each 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;
```