1806ICT
PROGRAMMING FUNDAMENTALS

ASSIGNMENT – DOCUMENTATION

Alison Butcher

s5113170

alison.butcher@griffithuni.edu.au

TABLE OF CONTENTS

DATA STRUCTURES

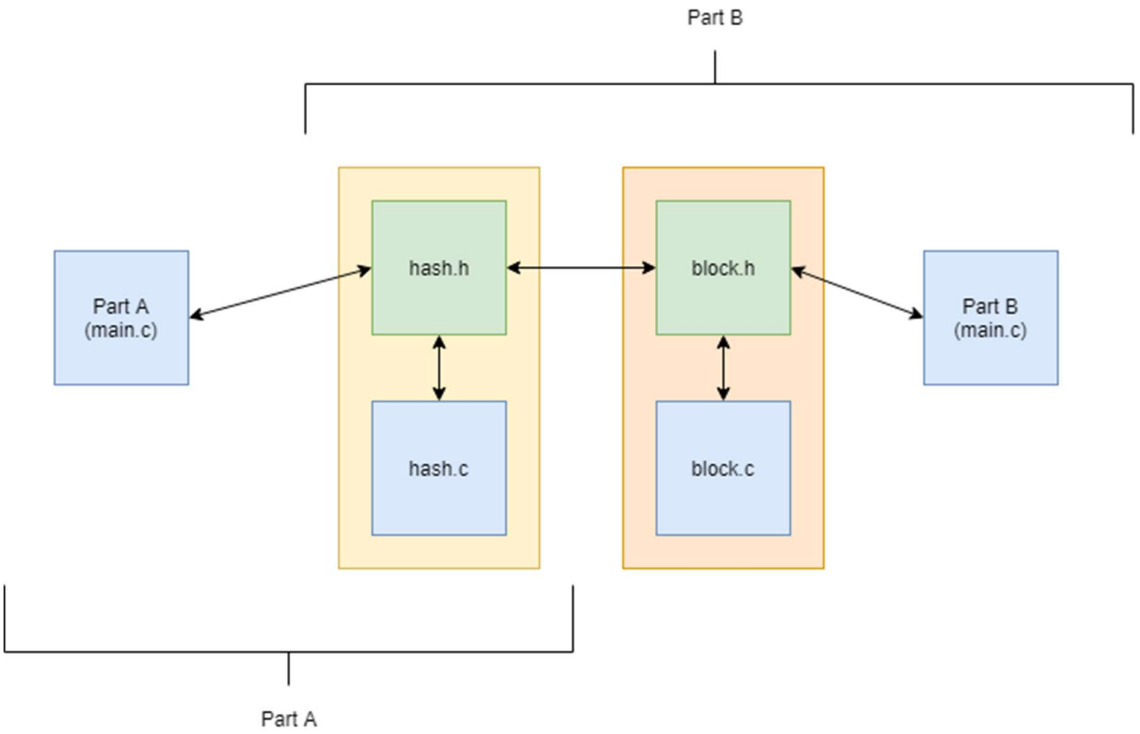

Figure 1 - File Structure

Central to Part A is a 1024 long char array (transactionString) which is used to process each transaction string.

Part B is centred around a linked list using the following struct:

```
typedef struct TransactionNode {
    char transactionString[100];
    int priority;
    int blockId;
    unsigned short int hash;
    struct TransactionNode *next;
} TNODE;


TNODE * head = NULL;                    // Global Head for linked list
```

The struct contains the transaction string as received by the input file, a priority value which is calculated by the transaction fee divided by the length of the transaction string and multiplied by 1000 to avoid zero values, a blockID which is used for designating which block the transaction has been assigned to, a hash value as calculated in Part A, and of course the next pointer essential for a linked list.

A global pointer (head) which points to the location of the head node in the linked list is used throughout several of the functions in block.c

FUNCTIONS PART A

MAIN FUNCTION (PART A)

In Part A the main function handles reading and writing the input and output files. While iterating through each line of the input file it passes the transaction string to the hash function which then returns a hash (proof of work value). The hash value is appended to the transaction string for each line and written to an output file. After all transactions are processed, a total of all the hash values is written to the last line of the output file.

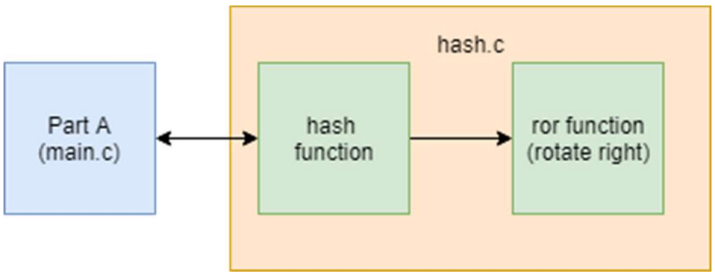

Figure 2 - Function Interaction Part A

ROTATE RIGHT FUNCTION

The Rotate Right function is designed to provide a right rotate across 16 bits of a word (data) by a specified number of times provided by the calling function (numRotates). Before proceeding with the rotate a modulus 16 of the number of rotates required reduces the number of rotates to 16 or below. A FOR loop controls the amount of times to rotate the data word. In each iteration of the loop a temporary variable (bit) is used to store the LSB and at the same time shift it to the MSB. Next a right shift is applied to the word followed by OR'ing the previously stored bit variable resulting in a 16 bit right shift for each iteration of the loop.

```
for each rotate amount
      save LSB of into temp word from data
      shift left temp word 15 times (LSB is now MSB 16 bits)
      shift data right 1
      OR the data with temp
Return data
```

HASHING FUNCTION

The Hash function iterates over the transaction string two bytes at a time (16 bit based on my student ID) and packs the char data into an integer word. It then calls the rotate right function to rotate the data a specified number of times based on its position within the transaction string. The returned rotated data is then XOR'ed with the current hash value. The function loops around continuously but incrementing a count each time until a hash value of zero is reached. The count (number of times the loop iterated) is the proof of work value (or hash value) from this function.

```
while end of transaction string not reached

      pack 2 bytes from transaction string into integer

      rotate right n times based on position in string

      xor with hash value

      increment count

if hash is 0

      return count

else

      loop again
```

FUNCTIONS PART B

MAIN FUNCTION (PART B)

In Part B the Main function sequentially calls several functions from block.c. The diagram below provides a good illustration of how each function is called.

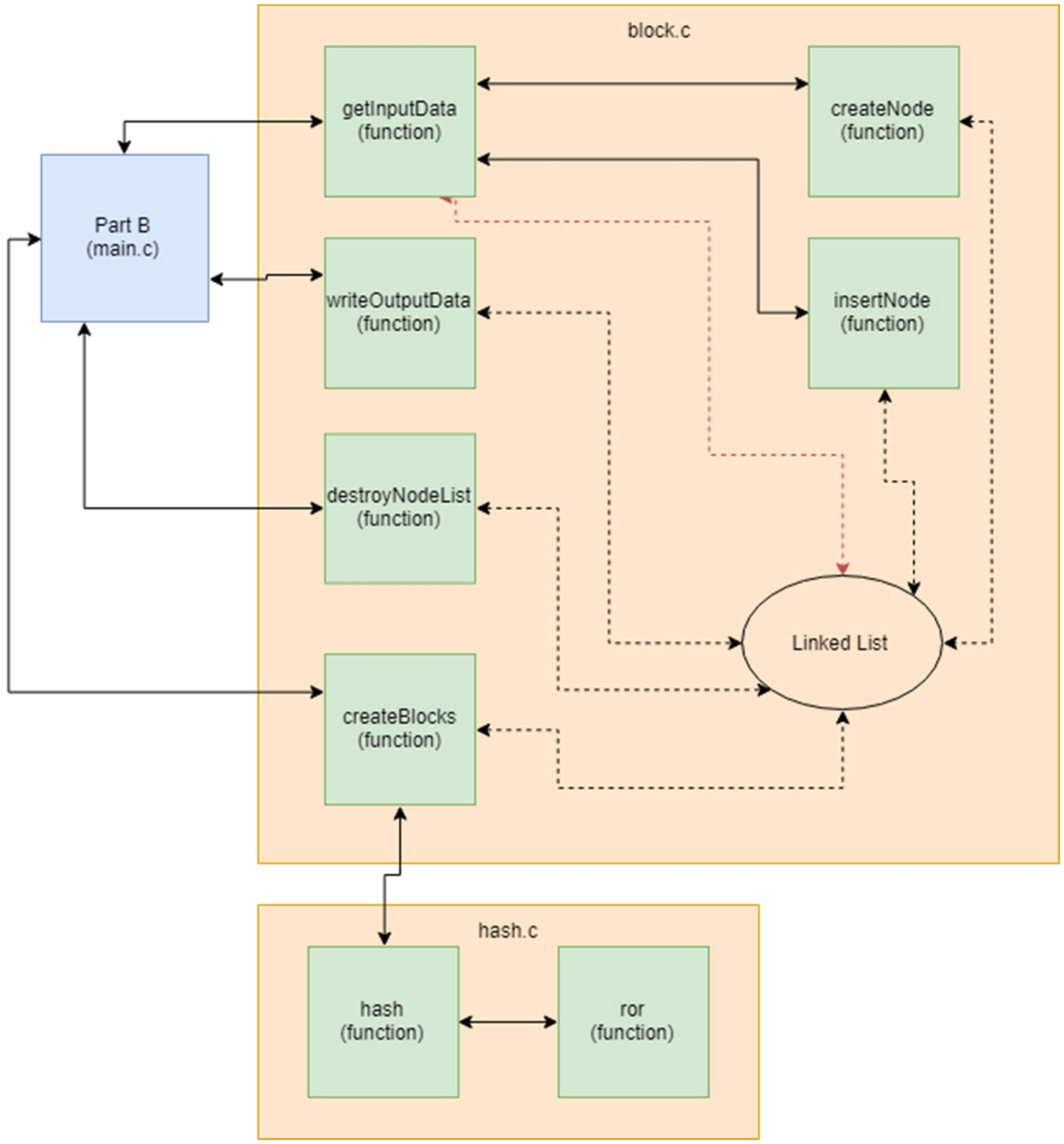

Figure 3 - Function Interactions Part B

GETINPUTDATA FUNCTION

This function, given a filename processes the input file containing the transactions list.

WRITEOUTPUTDATA FUNCTION

The write output function iterates over the linked list and writes each transaction and its hash value found in the linked list that is a member of the current block to an output file specified as an input to the function.

DESTROYNODELIST FUNCTION

The destroyNodeList function is responsible for de allocating the memory used by each node. It iterates through each node in the linked list and frees the memory allocated to it.

CREATEBLOCKS FUNCTION

The createBlocks function iterates through the already sorted linked list and allocates a block ID to nodes that will fit in the current block based on their priority value and if the the size of the transaction string will fit into the space left in the current block.

```
While not looped linked list without processing any nodes
    While current node next not null
        If node fits in block
            Set node->blockID to active blockID
            Get Hash of current node trans string
            Increment node processed count
        set current to next node
        calc block space remaining
```

CREATENODE FUNCTION

The createNode function allocates memory for each new node and extracts information such as priority and ensures the blockID is set to zero.

INSERTNODE FUNCTION

The insertNode function takes as a parameter a pointer to a newly created node and inserts it into the linked list based on its priority. This function is the key to the descending order of priority in the linked list.

ALGORITHMS

See above, included with each relevant function.

TESTING

Testing was carried out by using some extra functions that removed the need to input data each time the project was run. In addition each individual function was tested on its own before inclusion in the finished program.

For some parts of the project it was necessary to output debug files, print extra information to the console and of course use the debugger in the IDE.