Program Pseudo Code

```
// Include the necessary header files (stdio, stdlib, string)
// Set up definitions for specific pointer increments

typedef struct Transaction{
        // 3 variables to store Transaction Id, Customer ID and the amount of money spent
}transaction;

typedef struct Nodetype{ // Linked list data structure
        // Pointer Variable to store the memory location of a transaction
        // Pointer Variable to store the memory location of a node
}node;

// Function declarations

Int main(int argc, char **argv){
        // Initialize pointer variable for the message payload - (initMessage)
        // Initialize pointer variable for a new node (Start of the linked list) - (transactions)
        // Prompt user to enter message payload and dynamically allocate memory based on
        // size of string. Assign initMessage to point to this newly allocated memory.
        // Call function parseMessage(&transaction,initMessage)
        // Call function printMessageAll(transactions)
        // Call function freeData(transactions, initMessage)
        // END PROGRAM
}

void createTransaction(unsigned int transactionId, unsigned int customerId, float amount,
transaction **curTransaction){
        // Allocate memory for *curTransaction based on size of the Transaction structure.
        //Check if there was a memory allocation error and exit the program if there was.
        //Assign the transaction variables to the newly allocated *curTransaction.
}

void createNode(node **newNode, transaction *data){
        //Allocate memory for *newNode based on size of the NodeType structure.
        //Check if there was a memory allocation error and exit the program if there was/
        // Assign the data memory location to the newly allocated *newNode.
}

void parseMessage(node **transactions, char *message){
        // Initialize temporary transaction variables, a node pointer and a transaction pointer.
        while(1==1){
                if(current character at message pointer == '|'){
                        //Increment message pointer to where 0's start in transaction id
```

```
                        //(using definitions set above)

                        //Using the message pointer grab the entire transaction id string and
                        //convert it to an integer and store it in the temporary transaction variable.
                        //Repeat above steps but for customer id and amount.

                        //Call function createTransaction() with temp variables as input
                        //Call function createNode() with temp node and newly created
                        //transaction as input.

                        //Set the newly created node's next pointer to the memory location of the
                        //previously created node.
                        //Set the head node pointer to the newly created node.
                }
                else{
                        return
                }
        }
}

void printMessageSingle(node *transaction){
        //initialize a variable to store a string representation of the transactions variables
        //initialize a variable to store the length of the string representation
        //Convert the transaction id integer to a string and then store the strings length in
        //the newly initialized variable.
        //Based on the length of the string print out a certain amount of 0's (padding)
        //Print out the transaction Id integer and repeat the above steps with the customer id,
        //then print the amount.
}

void printMessageAll(node *transaction){
        //Initialize variable to keep track of the number of transactions
        //Initialize variable to keep track of the total amount of money across all transactions
        //Check if there are any transactions in the passed node. If not then return
        while(Next transaction is not empty){
                printMessageSingle(transaction)
                //Increment total number of transactions and total amount
                transaction = Next transaction
        }
        printMessageSingle(transaction)
        //Increment total number of transactions and total amount
        //Print total number of transactions and print average amount by dividing the total
        //amount by the number of transactions.
}
```

```
void freeData(node *transaction, char *message){
        //Initialize a new pointer variable and set it to the memory location of transaction.
        while(transaction is not empty){
                //Set the pointer variable to the memory location of the Next Transaction
                //Free the data and memory stored in transaction.
                //Set transaction = to the pointer variables stored memory location
                //Or better known as the Next Transaction.
        }
        //Free the message strings memory
}
```

END OF PSEUDO CODE

I. The message payload will be parsed by first sending the character pointer that points to the start of the message to a function. This function will then increment the message pointer by defined amounts based on where the 0's start for the transaction and customer id as well as the amount. Another function will then be called that grabs each of the required strings from the message payload and eventually convert them to ints or floats. These ints and floats are then passed into another function that initializes a new transaction structure and sets it's inner variables to these values.

II. A linked list data structure can be used for this assignment as it's total memory can grow and shrink dynamically based on how many transactions are entered as user input. The malloc function can be used to dynamically allocate memory for the linked list.

III. Dynamic Allocations:
    A. Linked list nodes.
    B. Each node's data / each transaction.
    C. Message payload string.

IV. Static Allocations:
    A. Temporary variables used to store parsed data
    B. Temporary pointers used to store node / transaction memory locations