Output of the Algorithm:

```
Enter Cashe Size: 4

Reading...
 12345 SABAN DEMIRHAN 1993 istanbul
Insert to  cashe.

---Hash Table---
 htable[0]: /
 htable[1]: /
 htable[2]: /
 htable[3]: /
 id: 12345 htable[4]: 0
 htable[5]: /
 htable[6]: /
---Cashe---
12345       | SABAN     | DEMIRHAN   | 1993 | istanbul

Reading...
 32145 TEYFIK ALBEN 1984 ankara
Insert to  cashe.

---Hash Table---
 htable[0]: /
 id: 32145 htable[1]: 0
 htable[2]: /
 htable[3]: /
 id: 12345 htable[4]: 1
 htable[5]: /
 htable[6]: /
---Cashe---
32145       | TEYFIK    | ALBEN     | 1984 | ankara
12345       | SABAN     | DEMIRHAN   | 1993 | istanbul

Reading...
 43213 AYSU SATIROGLU 1991 istanbul
Insert to  cashe.

---Hash Table---
 htable[0]: /
 id: 32145 htable[1]: 1
```

```
 id: 43213 htable[2]: 0
 htable[3]: /
 id: 12345 htable[4]: 2
 htable[5]: /
 htable[6]: /
---Cashe---
43213       | AYSU      | SATIROGLU  | 1991 | istanbul
32145       | TEYFIK    | ALBEN     | 1984 | ankara
12345       | SABAN     | DEMIRHAN   | 1993 | istanbul

Reading...
 45543 EMIRCAN GOKMEN 2003 ankara
Insert to  cashe.

---Hash Table---
 htable[0]: /
 id: 32145 htable[1]: 2
 id: 43213 htable[2]: 1
 htable[3]: /
 id: 12345 htable[4]: 3
 id: 45543 htable[5]: 0
 htable[6]: /
---Cashe---
45543       | EMIRCAN   | GOKMEN    | 2003 | ankara
43213       | AYSU      | SATIROGLU  | 1991 | istanbul
32145       | TEYFIK    | ALBEN     | 1984 | ankara
12345       | SABAN     | DEMIRHAN   | 1993 | istanbul

Reading...
 43321 ZUBEYDE HARMANBASI 2001 izmir
Cashe is full. deleting last node in cahe...

---Hash Table---
 id: 43321 htable[0]: 0
 id: 32145 htable[1]: 3
 id: 43213 htable[2]: 2
 htable[3]: /
 htable[4]: /
 id: 45543 htable[5]: 1
 htable[6]: /
---Cashe---
43321       | ZUBEYDE   | HARMANBASI | 2001 | izmir
45543       | EMIRCAN   | GOKMEN    | 2003 | ankara
```

```
43213       | AYSU      | SATIROGLU  | 1991 | istanbul
32145       | TEYFIK    | ALBEN     | 1984 | ankara

Reading...
 54213 AYSEL OZBEK 2000 trabzon
Cashe is full. deleting last node in cahe...

---Hash Table---
 id: 43321 htable[0]: 1
 htable[1]: /
 id: 43213 htable[2]: 3
 htable[3]: /
 htable[4]: /
 id: 45543 htable[5]: 2
 id: 54213 htable[6]: 0
---Cashe---
54213       | AYSEL     | OZBEK     | 2000 | trabzon
43321       | ZUBEYDE   | HARMANBASI | 2001 | izmir
45543       | EMIRCAN   | GOKMEN    | 2003 | ankara
43213       | AYSU      | SATIROGLU  | 1991 | istanbul

Reading...
 43321 ZUBEYDE HARMANBASI 2001 izmir
Already found in the cashe. Updating...

---Hash Table---
 id: 43321 htable[0]: 0
 htable[1]: /
 id: 43213 htable[2]: 3
 htable[3]: /
 htable[4]: /
 id: 45543 htable[5]: 2
 id: 54213 htable[6]: 1
---Cashe---
43321       | ZUBEYDE   | HARMANBASI | 2001 | izmir
54213       | AYSEL     | OZBEK     | 2000 | trabzon
45543       | EMIRCAN   | GOKMEN    | 2003 | ankara
43213       | AYSU      | SATIROGLU  | 1991 | istanbul

Reading...
 33445 ACELYA SENLIK 1990 adana
Cashe is full. deleting last node in cahe...
```

```
---Hash Table---
 id: 43321 htable[0]: 1
 id: 33445 htable[1]: 0
 htable[2]: /
 htable[3]: /
 htable[4]: /
 id: 45543 htable[5]: 3
 id: 54213 htable[6]: 2
---Cashe---
33445       | ACELYA    | SENLIK    | 1990 | adana
43321       | ZUBEYDE   | HARMANBASI | 2001 | izmir
54213       | AYSEL     | OZBEK     | 2000 | trabzon
45543       | EMIRCAN   | GOKMEN    | 2003 | ankara

Reading...
 12345 SABAN DEMIRHAN 1993 istanbul
Cashe is full. deleting last node in cahe...

---Hash Table---
 id: 43321 htable[0]: 2
 id: 33445 htable[1]: 1
 htable[2]: /
 htable[3]: /
 id: 12345 htable[4]: 0
 htable[5]: /
 id: 54213 htable[6]: 3
---Cashe---
12345       | SABAN     | DEMIRHAN   | 1993 | istanbul
33445       | ACELYA    | SENLIK    | 1990 | adana
43321       | ZUBEYDE   | HARMANBASI | 2001 | izmir
54213       | AYSEL     | OZBEK     | 2000 | trabzon

Reading...
 33445 ACELYA SENLIK 1990 adana
Already found in the cashe. Updating...

---Hash Table---
 id: 43321 htable[0]: 2
 id: 33445 htable[1]: 0
 htable[2]: /
 htable[3]: /
 id: 12345 htable[4]: 1
 htable[5]: /
```

```
 id: 54213 htable[6]: 3
---Cashe---
33445       | ACELYA    | SENLIK    | 1990 | adana
12345       | SABAN     | DEMIRHAN   | 1993 | istanbul
43321       | ZUBEYDE   | HARMANBASI | 2001 | izmir
54213       | AYSEL     | OZBEK     | 2000 | trabzon

---Final State---
---Cashe---
33445       | ACELYA    | SENLIK    | 1990 | adana
12345       | SABAN     | DEMIRHAN   | 1993 | istanbul
43321       | ZUBEYDE   | HARMANBASI | 2001 | izmir
54213       | AYSEL     | OZBEK     | 2000 | trabzon

---Hash Table---
 id: 43321 htable[0]: 2
 id: 33445 htable[1]: 0
 htable[2]: /
 htable[3]: /
 id: 12345 htable[4]: 1
 htable[5]: /
 id: 54213 htable[6]: 3
```

Kaynak kod:

```
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
#include<string.h>

#define STRING_SIZE 30
#define ID_SIZE 5
// #define CACHE_LENGTH 4
#define FILE_NAME "test.txt"
#define NULL_ -1

typedef struct{
   char id[STRING_SIZE];
   int loc;
}Hashtable;

typedef struct{
   char id[STRING_SIZE];
   char fname[STRING_SIZE];
   char lname[STRING_SIZE];
   int byear;
   char address[STRING_SIZE];
}Person;

typedef struct _Node{
   Person data;
   struct _Node *next;
```

```
}Node;



// hashing functions
//functions to return key based on the id
int hornerMethod(char *id);
// hash function
int hash(int key, int i, int cacheSize);
// function to retun hash table size
int hTableSize(int cacheSize);
int isPrime(int n);

// Hash table Functions
Hashtable* initHtable();
//functions to insert data to hash table
void insertToHtable(Hashtable *htable,char *id, int loc, int cacheSize);
// Update the locations of the old nodes in the cashe when inserting new one
void updateHtableInsert(Hashtable *htable, int cacheSize);
// Remove element in hash table based on ots id
void remove_element(Hashtable *htable, char *id, int cacheSize);
// Check if the id is alredy exists in the table
int checkNodeTable(Hashtable *htable, char *id, int cacheSize);
// Update the locations of the old nodes in the cashe when reinserting old one
void updateHtableModify(Hashtable *htable, char *id, int cacheSize);
//function to display hash table
void display(Hashtable *htable, int cacheSize);
// returns the location of the node in cash from the hash table based on the id
int getLoc(Hashtable *htable,char *id, int cacheSize);

//Cashe Functions
// insert node to cashe
void insertNode( Node** head_ref, Person new_data);
// remove last node in the cashe
void deleteLastNode(Node** head_ref);
// returns last node in the cashe
Node* getLastNode(Node** head_ref);
// prints the node in the cashe
void printList( Node *node);
// deletes node from the cashe based on its position
void deleteNode(Node** head_ref, int position);

int main(int argc, char const *argv[])
```

```
{
   int loc, cacheSize;
   printf("Enter Cashe Size: ");
   scanf("%d", &cacheSize);
   Hashtable *htable = initHtable(cacheSize);
   Node* head = ( Node*) malloc(sizeof( Node));
   head = NULL;
   Person *people;
   int i, key;
   people = malloc(100 * sizeof(Person));
   if(htable == NULL) {
  /*Memory allocation failed */
   }
   FILE *fileptr;

   fileptr = fopen(FILE_NAME, "r");
   if (!fileptr) {
       printf("Couldn't read %s", FILE_NAME);
       return 0;
   }
   int size = 0;
   i = 0;
   while (fscanf(fileptr, " %s %s %s %d %s", people[i].id, people[i].fname,
people[i].lname, &(people[i].byear), people[i].address) == 5)
   {
       printf("\nReading...\n");
       printf(" %s %s %s %d %s\n", people[i].id, people[i].fname, people[i].lname,
(people[i].byear), people[i].address);

       //if the node already exists in the cashe put the node in the front of the
cashe and update its location in the hash table also update the other nodes'
       //locations if needed
       if (checkNodeTable(htable, people[i].id, cacheSize))
       {
           printf("Already found in the cashe. Updating...\n");
           loc = getLoc(htable, people[i].id, cacheSize);
           updateHtableModify(htable, people[i].id, cacheSize);
           deleteNode(&head, loc);
           insertNode(&head, people[i]);
           printf("\n---Hash Table---");
           display(htable, cacheSize);
           printf("\n---Cashe---");
```

```
            printList(head);

        }
        //insert new node to the cahe and update the loations of the old ones
        else if (size < cacheSize)
        {
            printf("Insert to  cashe.\n");
            insertNode(&head, people[i]);
            updateHtableInsert(htable, cacheSize);
            insertToHtable(htable, people[i].id, 0, cacheSize);
            printf("\n---Hash Table---");
            display(htable, cacheSize);
            printf("\n---Cashe---");
            printList(head);
            size++;
        }
        // if the the cashe is full delete the last node in the cahe and insert the new
one
        else{
            printf("Cashe is full. deleting last node in cahe...\n");
            remove_element(htable, getLastNode(&head)->data.id, cacheSize);
            deleteLastNode(&head);
            updateHtableInsert(htable, cacheSize);
            insertNode(&head, people[i]);
            insertToHtable(htable, people[i].id, 0, cacheSize);
            printf("\n---Hash Table---");
            display(htable, cacheSize);
            printf("\n---Cashe---");
            printList(head);
        }

        i++;
    }
    printf("\n---Final State---");
    printf("\n---Cashe---");
    printList(head);
    printf("\n---Hash Table---");
    display(htable, cacheSize);

    free(htable);
    free(people);
    free(head);
```

```
    return 0;
}


int hornerMethod(char *id){
    size_t i;
    int assciVal, num;
    int hashValue = 0;

    for ( i = 0; i < ID_SIZE; i++)
    {
        assciVal = id[i];
        num = assciVal - '0';

        hashValue += pow(31, ID_SIZE - (i +1)) * num;
    }
    return hashValue;
}

int hash(int key, int i, int cacheSize){
    int h, h1, h2, m = hTableSize(cacheSize);

    h1 = key % m;
    h2 = 1 + (key % (m-1));
    h = (h1 + i*h2) % m;
    return h;
}

int hTableSize(int cacheSize){

    int m = cacheSize / 0.6;
    if (m < 3)
        return 2;
    if (m % 2 == 0)
        m++;
    while (!isPrime(m))
        m += 2;
    return m;
}
```

```
Hashtable* initHtable(int cacheSize)
{
   int size = hTableSize(cacheSize), i;
   Hashtable *ptr;
   ptr = malloc(size * sizeof(Hashtable));
   if (!ptr)
       {
           printf("Cannot allocate memory\n");

       }
   for (i = 0; i < size; i++)
   {
       strcpy(ptr[i].id, "NULL");
       ptr[i].loc = NULL_;
   }
   return ptr;
}

int isPrime(int n) {
   int i = 5;
   if (n <= 1)
       return 0;
   if (n <= 3)
       return 1;

   if (!(n % 2) || !(n % 3))
       return 0;

   while (i * i <= n && (n % i || n % (i + 2)))
       i += 6;

   if (i * i > n)
       return 1;

   return 0;
}

void insertToHtable(Hashtable *htable,char *id, int loc, int cacheSize)
{

   int i;
   int key = hornerMethod(id);
```

```
    i = 0;
    int index = hash(key, i, cacheSize);
    while (htable[index].loc != NULL_)
    {
        i++;
        index = hash(key, i, cacheSize);
    }

    strcpy(htable[index].id, id);
    htable[index].loc = loc;

}

void remove_element(Hashtable *htable, char *id, int cacheSize)
{
    int key = hornerMethod(id);
    int i = 0;

    int size = hTableSize(cacheSize);
    int index = hash(key, i, cacheSize);

    while (strcmp(htable[index].id, id) && i < size)
    {
        i++;
        index = hash(key, i, cacheSize);
    }

    strcpy(htable[index].id, "NULL");
    htable[index].loc = NULL_;

}

int getLoc(Hashtable *htable,char *id, int cacheSize)
{

    int i;
    int key = hornerMethod(id);

    i = 0;
    int index = hash(key, i, cacheSize);
    while (strcmp(htable[index].id, id))
```

```
    {
        i++;
        index = hash(key, i, cacheSize);
    }

    return htable[index].loc ;
}

int checkNodeTable(Hashtable *htable, char *id, int cacheSize){
    int size = hTableSize(cacheSize), i;
    for (i = 0; i < size; i++){
        if (!strcmp(htable[i].id, id))
        {
            return 1;
        }
    }
    return 0;
}

void updateHtableModify(Hashtable *htable, char *id, int cacheSize){
    int size = hTableSize(cacheSize), i, loc, index;

    for (i = 0; i < size; i++){
        if (!strcmp(htable[i].id, id))
        {
            loc = htable[i].loc;
            index = i;
        }
    }

    for ( i = 0; i < size; i++)
    {
        if (htable[i].loc != NULL_ && htable[i].loc < loc)
        {
            htable[i].loc += 1;
         }
    }

    htable[index].loc = 0;

}
```

```
void display(Hashtable *htable, int cacheSize)
{
   int size = hTableSize(cacheSize), i;

   for (i = 0; i < size; i++)
       {
       if (htable[i].loc == NULL_)
       {
           printf("\n htable[%d]: / ", i);
       }
       else
       {
           printf("\n id: %s htable[%d]: %d \t", htable[i].id, i, htable[i].loc);
       }
   }

}

void insertNode( Node** head_ref, Person new_data)
{
   Node* new_node = ( Node*) malloc(sizeof( Node));
   new_node->data  = new_data;
   new_node->next = *head_ref;
   *head_ref = new_node;
}

Node* getLastNode(Node** head_ref) {

   if(*head_ref != NULL) {
       if((*head_ref)->next == NULL) {
           *head_ref = NULL;
       } else {


       Node* temp = *head_ref;
       while(temp->next->next != NULL)
           temp = temp->next;

       Node* lastNode = temp->next;
       return lastNode;
       }
   }
```

```
   return NULL;
}

void deleteLastNode(Node** head_ref) {

   if(*head_ref != NULL) {
       if((*head_ref)->next == NULL) {
           *head_ref = NULL;
       } else {


       Node* temp = *head_ref;
       while(temp->next->next != NULL)
           temp = temp->next;

       Node* lastNode = temp->next;
       temp->next = NULL;
       free(lastNode);
       }
   }
}

void printList( Node *node)
{
   printf("\n");
   while (node != NULL)
   {
       printf("%-10s | %-10s | %-10s | %-4d | %-10s\n",node->data.id,
node->data.fname, node->data.lname, node->data.byear, node->data.address);
       node = node->next;
   }
}

void updateHtableInsert(Hashtable *htable, int cacheSize){
   int size = hTableSize(cacheSize);
   int i;
   for ( i = 0; i < size; i++)
   {
       if (htable[i].loc != NULL_)
       {
           htable[i].loc += 1;
       }
```

```
    }

}

void deleteNode(Node** head_ref, int position)
{
    if (*head_ref == NULL)
        return;

    Node* temp = *head_ref;
    if (position == 0) {
        *head_ref = temp->next;
        free(temp);
        return;
    }
    for (int i = 0; temp != NULL && i < position - 1; i++)
        temp = temp->next;

    if (temp == NULL || temp->next == NULL)
        return;

    Node* next = temp->next->next;
    free(temp->next);
    temp->next = next;
}
```