Εργασία 4η - Πίνακας Κατακερματισμού (HashTable)

Σε αυτή την εργασία θα υλοποιήσετε την **αποθήκευση λέξεων** (*δηλ. αλφαριθμητικών που δεν περιέχουν κενά*) σε ένα πίνακα κατακερματισμού (*HashTable*). Οι πίνακες κατακερματισμού χρησιμοποιούνται για την γρήγορη ένθεση, αναζήτηση και διαγραφή στοιχείων μέσα σε αυτούς. Το μειονέκτημα των πινάκων κατακερματισμού είναι ότι κατά κανόνα χρησιμοποιούν περισσότερο χώρο από όσο πραγματικά χρειάζονται, πράγμα που τους καθιστά ακατάλληλους για μεγάλο όγκο δεδομένων.

Ένθεση στον πίνακα κατακερματισμού

Η θέση ένθεσης ενός στοιχείου στον πίνακα κατακερματισμού δίνεται από την συνάρτηση κατακερματισμού η οποία για την εργασία συνοψίζεται στα εξής:

$$h_n(x) = (x+n) \text{ modulo } TABLE\_SIZE$$

δηλαδή $h_0(x) = h(x)$, $h_1(x) = h(x+1)$, $h_2(x) = h(x+2)$ κ.ο.κ., όπου **x** είναι το άθροισμα των ASCII κωδικών των χαρακτήρων κάθε περιεχόμενης λέξης. Για παράδειγμα το **x** για την λέξη **apple** είναι

Χαρακτήρας	Κωδικός ASCII
`'a'`	97
`'p'`	112
`'p'`	112
`'l'`	108
`'e'`	101
Άθροισμα	**530**

Αρχικά η συνάρτηση κατακερματισμού επιχειρεί να εισάγει το νέο στοιχείο στην θέση που προκύπτει από την παραπάνω σχέση για **n=0**. Εάν η θέση είναι διαθέσιμη τότε το στοιχείο εισάγεται στην θέση αυτή. Εάν η θέση δεν είναι διαθέσιμη τότε προχωρά στην εξέταση της επόμενης θέσης προς ένθεση που δίνει η συνάρτηση κατακερματισμού για **n=1**. Εάν η νέα θέση είναι διαθέσιμη τότε το στοιχείο εισάγεται στην θέση αυτή, ενώ εάν δεν είναι διαθέσιμη προχωρά στην επόμενη θέση. Η διαδικασία συνεχίζεται μέχρι να βρούμε διαθέσιμη θέση ή μέχρι να επιστρέψουμε στην αρχική θέση που μας επέστρεψε η συνάρτηση κατακερματισμού για **n=0**.

Με βάση την συνάρτηση κατακερματισμού αναζητούμε την 1η διαθέσιμη προς ένθεση θέση. Μία διαθέσιμη θέση ορίζεται ως μία θέση που είτε δεν έχει αποθηκευτεί κανένα στοιχείο κατά το παρελθόν, είτε έχει αποθηκευτεί κάποιο στοιχείο, αλλά πλέον έχει διαγραφεί.

1. Εάν δεν έχει αποθηκευτεί κανένα στοιχείο κατά το παρελθόν τότε αντικείμενο τύπου ***string*** στην αντίστοιχη θέση είναι κενό.
2. Εάν έχει αποθηκευτεί κάποιο στοιχείο κατά το παρελθόν και στη συνέχεια έχει διαγραφεί, τότε το αντικείμενο ***string*** στην αντίστοιχη θέση έχει το περιεχόμενο `"##tomb##"` (*από τα αρχικά της λέξης tombstone*).

Σημείωση: Κατά την διαγραφή είναι απαραίτητο να βάλουμε ένα χαρακτηριστικό που να δηλώνει ότι στη συγκεκριμένη θέση υπήρχε στοιχείο που διαγράφηκε. Η προσθήκη αυτή είναι σημαντική κατά την αναζήτηση διότι ενημερώνει, ότι στη συγκεκριμένη θέση υπήρχε στοιχείο το οποίο πλέον έχει διαγραφεί

Αναζήτηση στον πίνακα κατακερματισμού

Η αναζήτηση στον πίνακα κατακερματισμού χρησιμοποιεί τη συνάρτηση κατακερματισμού $\mathbf{h_n(x)}$ για να εντοπίσει εάν υπάρχει το στοιχείο προς αναζήτηση στον πίνακα ή όχι. Η αναζήτηση ξεκινά για **n=0** και επαναλαμβάνεται για **n=1,2,3** κλπ εφόσον δεν βρεθεί η ζητούμενη λέξη. Η αναζήτηση επαναλαμβάνεται μόνο όταν εντοπίζονται θέσεις που έχουν περιεχόμενο διαφορετικό του επιθυμητού ή θέσεις έχουν διαγραφεί κατά το παρελθόν και περιέχουν τη δεσμευμένη λέξη **"##tomb##"**. Η αναζήτηση σταματά επιτυχώς εάν βρεθεί το επιθυμητό περιεχόμενο και ανεπιτυχώς εάν βρεθεί κενή θέση ή εάν επιστρέψουμε στην θέση του πίνακα από την οποία εκκινήσαμε την αναζήτηση για **n=0**.

Διαγραφή από τον πίνακα κατακερματισμού

Η διαγραφή προϋποθέτει ότι έχει γίνει επιτυχής αναζήτηση του στοιχείου προς διαγραφή. Κατά την διαγραφή είναι απαραίτητο να βάλουμε ένα χαρακτηριστικό που να δηλώνει ότι στη συγκεκριμένη θέση υπήρχε στοιχείο που διαγράφηκε. Κατά σύμβαση στην παρούσα εργασία τοποθετούμε την λέξη **"##tomb##"** .

Υλοποίηση εργασίας

Η κλάση HashTableException

Η κλάση **HashTableException** είναι απόγονος της s**td::exception**. **Exceptions** της συγκεκριμένης κλάσης συνδέονται με την αδυναμία των αντικειμένων της κλάση **HashTable** να ενθέσουν νέα στοιχεία στον πίνακα κατακερματισμού. Η κλάση σας δίνεται έτοιμη παρακάτω.

```
#ifndef __H_TABLE_EXCEPTION_H__
#define __H_TABLE_EXCEPTION_H__

class HashTableException : public std::exception {

public:
  virtual const char* what() const noexcept {
    return " ----- HashTableException -----\n";
  }
};

#endif
```

Μέρος 1ο - Πίνακας κατακερματισμού

Προκειμένου να υλοποιήσετε την λειτουργία του πίνακα κατακερματισμού φτιάξτε την κλάση ***HashTable*** η οποία περιγράφεται παρακάτω:

Η κλάση *HashTable*

```
class HashTable {

protected:
  int size;
  int capacity;
  string *table;
  static int getHashCode(const char *str);

  bool isEmpty(int pos) const;
  bool isTomb(int pos) const;
  bool isAvailable(int pos) const;

public:
  HashTable(int capacity=8);
  HashTable(const HashTable &ht);
  int getSize() const;
  int getCapacity() const;

  bool contains(const string &s) const;
```

```
bool contains(const char *s) const;
string print() const;

virtual bool add(const string &s);
virtual bool add(const char *s);
virtual bool remove(const string &s);
virtual bool remove(const char *s);

HashTable& operator=(const HashTable &ht);

bool operator += (const string &str);
bool operator += (const char* s);
bool operator -= (const string &str);
bool operator -= (const char *s);

HashTable operator + (const string &str) const;
HashTable operator + (const char* s) const;
HashTable operator - (const string &str) const;
HashTable operator - (const char *s) const;

HashTable operator+(const HashTable &t) const;
HashTable &operator+=(const HashTable &t);

friend std::ostream& operator<<(std::ostream &stream, HashTable &t);

Iterator begin() const;
Iterator end() const;
```

Οι κατασκευαστές και οι μέθοδοι της κλάσης *HashTable* είναι οι εξής:

Μέθοδος	Περιγραφή
`HashTable(int capacity=8);`	Κατασκευαστής που λαμβάνει το μέγεθος του **HashTable**. Εάν δοθεί αρνητικό όρισμα ή δεν μπορεί να δεσμεύσει την απαραίτητη μνήμη πετάει το *exception* ***std::bad_alloc***.
`HashTable(const HashTable &ht);`	*Copy constructor* (κατασκευαστής αντιγραφέας). Εάν δεν μπορεί να δεσμεύσει την απαραίτητη μνήμη πετάει το *exception* ***std::bad_alloc***.
`int getSize() const;`	Επιστρέφει τον αριθμό των αποθηκευμένων στοιχείων στον πίνακα.
`int getCapacity() const;`	Επιστρέφει τη μέγιστη χωρητικότητα του πίνακα.
`bool isEmpty(int pos) const;`	Επιστρέφει ***true*** εάν η συγκεκριμένη θέση του πίνακα είναι άδεια, διαφορετικά ***false***. Εάν **`pos >= capacity`** επιστρέφει ***false***.
`bool isTomb(int pos) const;`	Επιστρέφει ***true*** εάν η συγκεκριμένη θέση του πίνακα περιέχει τη λέξη "**`##tomb##`**" *(σηματοδοτώντας ότι έχει διαγραφεί*

	κάποια λέξη κατά το παρελθόν) διαφορετικά ***false***. Εάν **`pos >= capacity`** επιστρέφει ***false***.
`bool isAvailable(int pos) const;`	Επιστρέφει ***true*** εάν επιστρέφει ***true*** μία από τις συναρτήσεις **`isTomb`** και **`isEmpty`**.
`virtual bool add(const string &s);`	Ενθέτει το αλφαριθμητικό στον πίνακα, αφού προηγουμένως βεβαιωθεί ότι δεν υπάρχει σε αυτόν. Επιστρέφει ***true*** εάν η ένθεση είναι επιτυχής, διαφορετικά επιστρέφει ***false***. Εάν το αλφαριθμητικό υπάρχει ήδη ή δεν υπάρχουν διαθέσιμες θέσεις αποτυγχάνει. Εάν δεν υπάρχουν διαθέσιμες θέσεις πετάει **HashtableException**.
`virtual bool add(const char *s);`	Ενθέτει το αλφαριθμητικό στον πίνακα, αφού προηγουμένως βεβαιωθεί ότι δεν υπάρχει σε αυτόν. Επιστρέφει ***true*** εάν η ένθεση είναι επιτυχής, διαφορετικά επιστρέφει ***false***. Εάν το αλφαριθμητικό υπάρχει ήδη αποτυγχάνει. Εάν δεν υπάρχουν διαθέσιμες θέσεις πετάει **HashtableException**.
`virtual bool remove(const string &s);`	Διαγράφει το αλφαριθμητικό από τον πίνακα εφόσον αυτό υπάρχει. Επιστρέφει ***true*** εάν η διαγραφή είναι επιτυχής, διαφορετικά επιστρέφει ***false***.
`virtual bool remove(const char *s);`	Διαγράφει το αλφαριθμητικό από τον πίνακα εφόσον αυτό υπάρχει. Επιστρέφει ***true*** εάν η διαγραφή είναι επιτυχής, διαφορετικά επιστρέφει ***false***.
`bool contains(const string &s) const;`	Επιστρέφει ***true*** εάν το αλφαριθμητικό περιέχεται στον πίνακα.
`bool contains(const char *s) const;`	Επιστρέφει ***true*** εάν το αλφαριθμητικό περιέχεται στον πίνακα.
`string print() const;`	Δίνεται έτοιμη
`bool operator += (const string &str);`	Λειτουργικότητα όμοια με τη συνάρτηση **`bool add(const string &s)`** (χρησιμοποιεί την παραπάνω συνάρτηση εσωτερικά)
`bool operator += (const char* s);`	Λειτουργικότητα όμοια με τη συνάρτηση **`bool add(const char *s);`** (χρησιμοποιεί την παραπάνω συνάρτηση εσωτερικά)
`bool operator -= (const string &str);`	Λειτουργικότητα όμοια με τη συνάρτηση **`bool remove(const string &s);`** (χρησιμοποιεί την παραπάνω συνάρτηση εσωτερικά)
`bool operator -= (const char *s);`	Λειτουργικότητα όμοια με τη συνάρτηση **`bool remove(const char *s);`** (χρησιμοποιεί την παραπάνω συνάρτηση εσωτερικά)
`HashTable operator+(const string &str) const;`	Παράγει ένα νέο ***HashTable*** που προκύπτει από την ένθεση του ***str*** στο υφιστάμενο ***HashTable***. Εάν το αλφαριθμητικό

	υπάρχει στο αρχικό ***HashTable*** ή δεν υπάρχουν διαθέσιμες θέσεις, το προκύπτον ***HashTable*** είναι όμοιο με το αρχικό. Εάν δεν υπάρχουν διαθέσιμες θέσεις πετάει **HashtableException**.
`HashTable operator+(const char* s) const;`	Παράγει ένα νέο ***HashTable*** που προκύπτει από την ένθεση του ***s*** στο υφιστάμενο ***HashTable***. Εάν το αλφαριθμητικό υπάρχει στο αρχικό ***HashTable***, το προκύπτον ***HashTable*** είναι όμοιο με το αρχικό. Εάν δεν υπάρχουν διαθέσιμες θέσεις πετάει **HashtableException**.
`HashTable operator-(const string &str) const;`	Παράγει ένα νέο ***HashTable*** που προκύπτει από τη διαγραφή του ***str*** από το υφιστάμενο ***HashTable***. Εάν το αλφαριθμητικό δεν υπάρχει στο αρχικό ***HashTable***, το προκύπτον ***HashTable*** είναι όμοιο με το αρχικό.
`HashTable operator-(const char *s const);`	Παράγει ένα νέο ***HashTable*** που προκύπτει από τη διαγραφή του ***str*** από το υφιστάμενο ***HashTable***. Εάν το αλφαριθμητικό δεν υπάρχει στο αρχικό ***HashTable***, το προκύπτον ***HashTable*** είναι όμοιο με το αρχικό.
`HashTable operator+(HashTable &t) const;`	Παράγει ένα νέο ***HashTable*** που προκύπτει από την ένωση του ***t*** με το υφιστάμενο ***HashTable***. Η χωρητικότητα του νέου ***HashTable*** είναι το άθροισμα των χωρητικοτήτων των επιμέρους ***HashTable*** που μετέχουν στην πράξη.
`HashTable &operator+=(HashTable &t);`	Ενθέτει τα στοιχεία του **t** στο υφιστάμενο **HashTable**, ~~χωρίς να αλλάξει την χωρητικότητα του~~ αλλάζοντας την χωρητικότητα του. Η χωρητικότητα του αριστερού τελεστέου (τρέχον **HashTable**) μεταβάλλεται και είναι ίση με το άθροισμα των χωρητικοτήτων των επιμέρους **HashTable** που μετέχουν στην πράξη. Επιστρέφει μία αναφορά στο υφιστάμενο **HashTable**.
`friend std::ostream& operator<<(std::ostream &stream, HashTable &t);`	Υπερφορτώνει τον τελεστή **<<**, ώστε το string που επιστρέφει η μέθοδος **print** να εκτυπωθεί στο `ostream`.

Μέρος 2ο - Iterator

Σας ζητείται να κατασκευάσετε την κλάση ***Iterator*** που είναι τύπου **ForwardIterator** ως εσωτερική κλάση (`HashTable::Iterator`) της κλάσης ***HashTable***, η οποία επιτρέπει την διάτρεξη των στοιχείων του πίνακα, τη σύγκριση δύο θέσεων μεταξύ τους ως προς την ισότητα/ανισότητα, τη λήψη του περιεχομένου μιας θέσης του πίνακα κ.ο.κ. Για τον λόγο αυτό η κλάση **HashTable** περιέχει τις μεθόδους **begin** και **end** οι οποίες κάνουν τα εξής:

Μέθοδος	Περιγραφή
`Iterator begin();`	Επιστρέφει ένα αντικείμενο της κλάσης ***Iterator*** που δείχνει ~~στην αρχή~~ στην 1η μη κενή θέση του πίνακα.
`Iterator end();`	Επιστρέφει ένα αντικείμενο της κλάσης ***Iterator*** που δείχνει στη διεύθυνση αμέσως μετά το τέλος του πίνακα.

Η κλάση ***Iterator*** περιγράφεται ως εξής:

Η κλάση *HashTable::Iterator*

```
class HashTable::Iterator {
  string *curr;
  const HashTable *ht;

public:
  Iterator(const HashTable *t);
  Iterator(const HashTable *t, bool end);
  Iterator(const Iterator &it);
  Iterator& operator=(const Iterator &it);
  Iterator operator++();
  Iterator operator++(int a);
  bool operator==(const Iterator &it) const ;
  bool operator!=(const Iterator &it) const;
  const string& operator*();
  const string* operator->();
};
```

Οι κατασκευαστές και οι μέθοδοι της κλάσης ***Iterator*** περιγράφονται ως εξής:

Μέθοδος	Περιγραφή
`Iterator(const HashTable *t);`	Κατασκευαστής που λαμβάνει ως όρισμα ένα δείκτη στη δομή. Το πεδίο **curr** αρχικοποιείται ώστε να δείχνει στο 1ο στοιχείο του πίνακα.
`Iterator(const HashTable *t, bool start);`	Κατασκευαστής που λαμβάνει ως όρισμα ένα δείκτη στη δομή. Το πεδίο **curr** αρχικοποιείται ώστε να δείχνει μετά το τελευταίο στοιχείο του πίνακα εάν το 2ο όρισμα είναι **false**. Εάν το 2ο όρισμα είναι **true**, το πεδίο **curr** αρχικοποιείται ώστε να δείχνει στο 1ο στοιχείο του πίνακα.
`Iterator(const Iterator &it);`	*Copy constructor*
`Iterator operator++();`	Τελεστής αύξησης της θέσης του ***Iterator*** κατά 1. Επιστρέφει έναν αντικείμενο τύπου ***Iterator*** που περιέχει την ανανεωμένη θέση του δείκτη **curr.**.
`Iterator operator++(int a);`	Τελεστής αύξησης της θέσης του Iterator κατά 1. Επιστρέφει έναν αντικείμενο τύπου ***Iterator*** που περιέχει την παλιά θέση του δείκτη **curr**.
`bool operator==(Iterator &it);`	Ελέγχει την ισότητα μεταξύ του τρέχοντος αντικειμένου και της παραμέτρου ***it***. Επιστρέφει ***true*** εάν τα αντικείμενα είναι ίδια, διαφορετικά ***false***.
`bool operator!=(Iterator it);`	Ελέγχει την ισότητα μεταξύ του τρέχοντος αντικειμένου και της παραμέτρου ***it***. Επιστρέφει ***true*** εάν τα αντικείμενα ΔΕΝ είναι ίδια, διαφορετικά ***false***.

`string &operator*();`	Επιστρέφει το περιεχόμενο της διεύθυνσης που δείχνει ***curr*** ο δείκτης του ***Iterator***.
`string* operator->();`	Επιστρέφει τη διεύθυνση που δείχνει ο δείκτης ***curr*** του ***Iterator***.

Παρατηρήσεις:

1. Κατά την διάτρεξη με τη βοήθεια του ***Iterator***, αυτός επιστρέφει μόνο μη κενές ή μη διαγραμμένες θέσεις, δηλαδή επιστρέφει θέσεις στις οποίες έχουμε λέξεις.
2. Κατά την διάτρεξη με τη βοήθεια του ***Iterator*** μπορείτε να υποθέσετε ότι ο πίνακας κατακερματισμού παραμένει αμετάβλητος.

Μέρος 3ο - Ανακατακερματισμός

Σας ζητείται να κατασκευάσετε τη κλάση ***ExtHashTable (ExtensibleHashTable)*** η οποία αποτελεί επέκταση της κλάσης HashTable και έχει τα εξής χαρακτηριστικά:

1. Κατά την ένθεση ενός στοιχείου εξετάζει εάν ο λόγος ***size/capacity*** είναι μεγαλύτερος από ένα προκαθορισμένο ποσοστό (***upper_bound_ratio***). Εάν ναι, διπλασιάζει τη χωρητικότητα του πίνακα, ενθέτοντας ξανά τα στοιχεία στον νέο πίνακα.
2. Κατά τη διαγραφή ενός στοιχείου εξετάζει εάν ο λόγος ***size/capacity*** είναι μικρότερος από ένα προκαθορισμένο ποσοστό (***lower_bound_ratio***). Εάν ναι, υπο-διπλασιάζει τη χωρητικότητα του πίνακα, ενθέτοντας ξανά τα στοιχεία στον νέο πίνακα.

Η κλάση **ExtHashTable** περιγράφεται ως εξής:

Η κλάση *ExtHashTable*

```
class ExtHashTable: public HashTable {
private:
  double upper_bound_ratio, lower_bound_ratio;
  void rehash();

public:
  ExtHashTable( double upper_bound_ratio=0.5,
                double lower_bound_ratio=0.125,
                int size=8);
  ExtHashTable(const ExtHashTable &t);
  bool add(const string &str);
  bool add(const char *s);
  bool remove(const string &str);
  bool remove(const char *s);

  ExtHashTable &operator=(const ExtHashTable &t);

  ExtHashTable operator+(const string &str) const;
```

```
  ExtHashTable operator+(const char* s) const;
  ExtHashTable operator-(const string &str) const;
  ExtHashTable operator-(const char *s) const;

  bool operator += (const string &str);
  bool operator += (const char* s);
  bool operator -= (const string &str);
  bool operator -= (const char *s);

  ExtHashTable operator+(const ExtHashTable &t) const;
  ExtHashTable &operator+=(const ExtHashTable &t);
};
```

Οι μέθοδοι της κλάσης περιγράφονται ως εξής:

Μέθοδος	Περιγραφή

`ExtHashTable(`
`  double  upper_bound_ratio=0.5,`
`  double lower_bound_ratio=0.125,`
`  int size=12);`
	Κατασκευαστής.

`ExtHashTable(ExtHashTable &t);`	*Copy constructor*
`bool add(const string &str);`	Όμοια με την συνάρτηση της γονικής κλάσης, με τη διαφορά ότι καλεί εσωτερικά τη συνάρτηση **`rehash(void)`** και δεν πετάει **HashtableException** (γιατί;).
`bool add(const char *s);`	Όμοια με την συνάρτηση της γονικής κλάσης, με τη διαφορά ότι καλεί εσωτερικά τη συνάρτηση **`rehash(void)`** και δεν πετάει **HashtableException** (γιατί;).
`bool remove(const string &str);`	Όμοια με την συνάρτηση της γονικής κλάσης, με τη διαφορά ότι καλεί εσωτερικά τη συνάρτηση **`rehash(void)`** και δεν πετάει **HashtableException** (γιατί;).
`bool remove(const char *s);`	Όμοια με την συνάρτηση της γονικής κλάσης, με τη διαφορά ότι καλεί εσωτερικά τη συνάρτηση **`rehash(void)`** και δεν πετάει **HashtableException** (γιατί;).
`void rehash();`	Ελέγχει εάν θα πρέπει να γίνει ***rehash*** και υλοποιεί τη λειτουργία ***rehashing*** εφόσον απαιτείται. Κάθε φορά που αλλάζει το μέγεθος του πίνακα εκτυπώνει ένα μήνυμα της μορφής: **`--> Size: X, New capacity: Y`** όπου **`X`** αριθμός των αποθηκευμένων στοιχείων του πίνακα και **`Y`** η χωρητικότητα του.
`ExtHashTable &operator= (const ExtHashTable &t);`	Υπερφόρτωση του τελεστή '**=**'.
`ExtHashTable operator+(const string &str) const;`	Παράγει ένα νέο ***ExtHashTable*** που προκύπτει από την ένθεση του ***str*** στο υφιστάμενο ***ExtHashTable***. Εάν το αλφαριθμητικό υπάρχει στο αρχικό ***ExtHashTable***, το προκύπτον ***ExtHashTable*** είναι όμοιο με το αρχικό.

`ExtHashTable operator+(const char *s) const;`	Παράγει ένα νέο ***ExtHashTable*** που προκύπτει από την ένθεση του ***s*** στο ***ExtHashTable ExtHashTable***. Εάν το αλφαριθμητικό υπάρχει στο αρχικό ***ExtHashTable***, το προκύπτον ***ExtHashTable*** είναι όμοιο με το αρχικό.
`ExtHashTable operator-(const string &str) const;`	Παράγει ένα νέο ***ExtHashTable*** που προκύπτει από τη διαγραφή του ***str*** από το υφιστάμενο ***ExtHashTable***. Εάν το αλφαριθμητικό δεν υπάρχει στο αρχικό ***ExtHashTable***, το προκύπτον ***ExtHashTable*** είναι όμοιο με το αρχικό.
`ExtHashTable operator-(const char *s) const;`	Παράγει ένα νέο ***ExtHashTable*** που προκύπτει από τη διαγραφή του ***s*** από το υφιστάμενο ***ExtHashTable***. Εάν το αλφαριθμητικό δεν υπάρχει στο αρχικό ***ExtHashTable***,
`bool operator += (const string &str);`	Λειτουργικότητα όμοια με τη συνάρτηση `bool add(const string &s)` (χρησιμοποιεί την παραπάνω συνάρτηση εσωτερικά)
`bool operator += (const char *);`	Λειτουργικότητα όμοια με τη συνάρτηση `bool add(const char *s);` (χρησιμοποιεί την παραπάνω συνάρτηση εσωτερικά)
`bool operator -= (const string &str);`	Λειτουργικότητα όμοια με τη συνάρτηση `bool remove(const string &s)` (χρησιμοποιεί την παραπάνω συνάρτηση εσωτερικά)
`bool operator -= (const char *);`	Λειτουργικότητα όμοια με τη συνάρτηση `bool remove(const char *s);` (χρησιμοποιεί την παραπάνω συνάρτηση εσωτερικά)
`ExtHashTable operator+(const ExtHashTable &t) const;`	Παράγει ένα νέο ***ExtHashTable*** που προκύπτει από την ένωση του ***t*** με το υφιστάμενο ***ExtHashTable***.~~Η χωρητικότητα του νέου ***ExtHashTable*** είναι το άθροισμα των χωρητικοτήτων των επιμέρους ***ExtHashTables*** που μετέχουν στην πράξη.~~ **Η χωρητικότητα του νέου ExtHashTable προκύπτει από την διαδικασία του ανακατακερματισμού.** **Για παράδειγμα εάν τα δύο ExtHashTable T1, T2 που μετέχουν στην πράξη έχουν T1(capacity: 16, size: 7) T2(capacity:32, size: 15), το τελικό HashTable θα έχει capacity: 64, size: 22).**
`ExtHashTable &operator+=(const ExtHashTable &t);`	Ενθέτει το **t** στο υφιστάμενο ***ExtHashTable***. Επιστρέφει μία αναφορά στο υφιστάμενο ***ExtHashTable***.

Γενικές Οδηγίες

Μπορείτε να ορίσετε επιπλέον κατασκευαστές, συναρτήσεις μέλη της κλάσης ή φιλικές συναρτήσεις εφόσον το κρίνετε απαραίτητο.

Τρόπος Αποστολής

Σε ένα φάκελο με όνομα το όνομα και το ΑΕΜ κάθε μέλους της ομάδας σας, αποθηκεύστε ΜΟΝΟ τα αρχεία πηγαίου κώδικα C++. Παράδειγμα φακέλου είναι το εξής: *GiorgosThanos_1234_PeterGordon_1235.*

Αφού συμπιέσετε το φάκελο σε ένα αρχείο ZIP με το ίδιο όνομα και κατάληξη ZIP ή TGZ ή TAR.GZ (**όχι RAR**), αποστείλετε την δουλειά σας με e-mail στην διεύθυνση **ce325.course@gmail.com** ως εξής:

Τίτλος (subject): CE325 hw04

Συνημμένο: Το παραπάνω συμπιεσμένο αρχείο.

Σώμα μηνύματος: Τα ονόματα και τα ΑΕΜ της ομάδας σας.

Εργασίες που δεν είναι συνεπείς με τους παραπάνω περιορισμούς δεν αξιολογούνται.