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*) που αποτελείται από δείκτες σε εγγραφές. Οι πίνακες κατακερματισμού χρησιμοποιούνται για τη γρήγορη ένθεση, αναζήτηση και διαγραφή στοιχείων. Το μειονέκτημα των πινάκων κατακερματισμού είναι ότι κατά κανόνα χρησιμοποιούν περισσότερο χώρο από όσο πραγματικά χρειάζεται, πράγμα που τους καθιστά ακατάλληλους για μεγάλο όγκο δεδομένων.

Ένθεση στον πίνακα κατακερματισμού

Η θέση ένθεσης ενός στοιχείου στον πίνακα κατακερματισμού δίνεται από τη συνάρτηση τοποθετήσεως, η οποία για την παρούσα εργασία είναι:

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

δηλαδή $h_0(x) = h(x)$, $h_1(x) = h(x+1)$, $h_2(x) = h(x+2)$ κ.ο.κ., όπου **x** ένας ακέραιος που επιστρέφεται από την παρακάτω συνάρτηση μετασχηματισμού του αλφαριθμητικού σε ακέραιο. Αντιγράψτε τον παρακάτω κώδικα, μην κάνετε copy-paste.

```
unsigned long getHashCode(const char *str) {
    unsigned long hash = 97;
    int c;

    while ((c = *(str++)) != '\0')
        hash = ((hash << 5) + hash) + c; /* hash * 33 + c */

    return hash;
}
```

Αρχικά η συνάρτηση τοποθετήσεως επιχειρεί να εισάγει το νέο στοιχείο στη θέση που προκύπτει από την παραπάνω σχέση για **n=0**. Εάν η θέση είναι διαθέσιμη τότε το στοιχείο εισάγεται στη θέση αυτή. Εάν η θέση είναι κατειλημμένη τότε προχωρά στην εξέταση της επόμενης θέσης προς ένθεση που δίνει η συνάρτηση για **n=1**. Η διαδικασία επαναλαμβάνεται μέχρι να εντοπίσουμε διαθέσιμη θέση ή μέχρι να επιστρέψουμε στην αρχική θέση από την οποία ξεκινήσαμε την αναζήτηση για **n=0**.

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 ο δείκτης στην αντίστοιχη θέση είναι NULL.
2. Εάν έχει αποθηκευτεί κάποιο στοιχείο κατά το παρελθόν και στη συνέχεια έχει διαγραφεί, τότε ο δείκτης στην αντίστοιχη θέση έχει μία τιμή της επιλογής σας (π.χ. 0x9999 ή 0xFFFF ή 0x1000).

Σημείωση: Κατά τη διαγραφή είναι απαραίτητο να αποθηκεύσουμε μία τιμή που να δηλώνει ότι στη συγκεκριμένη θέση υπήρχε στοιχείο που διαγράφηκε. Η προσθήκη αυτή είναι σημαντική κατά την αναζήτηση, ώστε η αναζήτηση να συνεχίζεται αν βρει μία θέση που έχει διαγραφεί, αλλά να μη συνεχίζεται αν βρεθεί μία θέση που είναι κενή (περιέχει την τιμή NULL).

Αναζήτηση στον πίνακα κατακερματισμού

Η αναζήτηση στον πίνακα κατακερματισμού χρησιμοποιεί τη συνάρτηση τοποθετήσεως $h_n(x)$ 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 δε βρεθεί η ζητούμενη λέξη. Η αναζήτηση επαναλαμβάνεται μόνο όταν εντοπίζονται θέσεις που έχουν περιεχόμενο διαφορετικό του επιθυμητού ή θέσεις έχουν διαγραφεί κατά το παρελθόν. Η αναζήτηση σταματά επιτυχώς εάν βρεθεί το επιθυμητό περιεχόμενο και ανεπιτυχώς εάν βρεθεί κενή θέση ή εάν επιστρέψουμε στη θέση του πίνακα από την οποία ξεκινήσαμε την αναζήτηση για **n=0**.

Διαγραφή από τον πίνακα κατακερματισμού

Η διαγραφή προϋποθέτει ότι έχει γίνει επιτυχής αναζήτηση του στοιχείου προς διαγραφή. Κατά τη διαγραφή είναι απαραίτητο να αποθηκεύουμε μία επιλεγμένη χαρακτηριστική τιμή που να δηλώνει ότι στη συγκεκριμένη θέση υπήρχε στοιχείο που διαγράφηκε. Για παράδειγμα, μπορείτε να τοποθετήσετε μία από τις τιμές 0x9999 ή 0xFFFF ή 0x1000.

Υλοποίηση εργασίας

Η κλάση HashTableException

Η κλάση **HashTableException** είναι απόγονος της **std::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 unsigned long getHashCode(const char *str) const;

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

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

  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);

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

  friend std::ostream& operator<<(std::ostream &stream, const 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 παράγει το *exception* ***std::bad_alloc***.
`HashTable(const HashTable &ht);`	*Copy constructor* (κατασκευαστής αντιγραφέας). Εάν δεν μπορεί να δεσμεύσει την απαραίτητη μνήμη παράγει το *exception* ***std::bad_alloc***.
`int getSize() const;`	Επιστρέφει τον αριθμό των αποθηκευμένων στοιχείων στον πίνακα.
`int getCapacity() const;`	Επιστρέφει τη χωρητικότητα του πίνακα.
`bool isEmpty(int pos) const;`	Επιστρέφει ***true*** εάν η συγκεκριμένη θέση του πίνακα είναι άδεια (δεν είχε πληρωθεί ποτέ), διαφορετικά ***false***. Εάν `pos >= capacity` επιστρέφει ***false***.
`bool isTomb(int pos) const;`	Επιστρέφει ***true*** εάν η συγκεκριμένη θέση του πίνακα περιέχει την ειδική τιμή της επιλογής σας που *σηματοδοτεί ότι έχει διαγραφεί κάποια λέξη κατά το παρελθόν*, διαφορετικά επιστρέφει ***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 ήδη σε αυτόν. Επιστρέφει ***true*** εάν η ένθεση είναι επιτυχής, διαφορετικά επιστρέφει ***false***. Εάν το αλφαριθμητικό δεν υπάρχει, αλλά δεν υπάρχουν διαθέσιμες θέσεις παράγει **HashtableException**.
`virtual bool add(const char *s);`	Λειτουργικότητα όμοια με τη συνάρτηση `bool add(const string &s)`.
`virtual bool remove(const string &s);`	Διαγράφει το αλφαριθμητικό από τον πίνακα εφόσον αυτό υπάρχει. Επιστρέφει ***true*** εάν η διαγραφή ήταν επιτυχής, διαφορετικά επιστρέφει ***false***.
`virtual bool remove(const char *s);`	Λειτουργικότητα όμοια με τη συνάρτηση `bool remove(const string &s)`.
`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.
`HashTable &operator= (const HashTable`	Υπερφόρτωση του τελεστή '**=**'. Στον αριστερό τελεστέο

`&t);`	ανατίθεται το περιεχόμενο του δεξιού τελεστέου. Το περιεχόμενο του αριστερού τελεστέου διαγράφεται πριν από την ανάθεση.
`HashTable& operator += (const string &str);`	Ενθέτει το αλφαριθμητικό στον πίνακα, εφόσον το αλφαριθμητικό δεν υπάρχει σε αυτόν και επιστρέφει μία αναφορά στο ανανεωμένο HashTable. Εάν δεν υπάρχει επαρκής χώρος για την ένθεση παράγει **HashtableException**.
`HashTable& operator += (const char* s);`	Λειτουργικότητα όμοια με τη συνάρτηση `HashTable& operator += (const string &str)`
`HashTable& operator -= (const string &str);`	Διαγράφει το αλφαριθμητικό από τον πίνακα εφόσον αυτό υπάρχει και επιστρέφει μία αναφορά στο ανανεωμένο HashTable.
`HashTable& operator -= (const char *s);`	Λειτουργικότητα όμοια με τη συνάρτηση `HashTable& operator -= (const string &str)`
`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*** το προκύπτον ***HashTable*** είναι όμοιο με το αρχικό. Εάν δεν υπάρχουν διαθέσιμες θέσεις παράγει **HashtableException**.
`HashTable operator+(const char* s) const;`	Λειτουργικότητα όμοια με τη συνάρτηση `HashTable operator+(const char* s) const`
`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);`	Λειτουργικότητα όμοια με τη συνάρτηση `HashTable operator-(const char* s) const`
`friend std::ostream& operator<<(std::ostream &stream, const HashTable &t);`	Υπερφορτώνει τον τελεστή **<<**, ώστε το string που επιστρέφει η μέθοδος **print** να εκτυπωθεί στο `ostream`.

Η συνάρτηση print δίνεται έτοιμη παρακάτω (μην κάνετε copy-paste, αλλά αντιγράψτε τον κώδικα):

```
string HashTable::print() const {
  string str;
  char buf[128];

 for(int i=0; i<capacity; i++) {
    if( !isAvailable(i) ) {
      sprintf(buf, "%2d. -%s-\n", i, (*table[i]).c_str());
      str.append(buf);
    }
  }

  sprintf(buf, " --- CAPACITY: %d, SIZE: %d ---\n", capacity, size);
  str.append(buf);
  return str;
}
```

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 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;
  int position;  // Θέση του δείκτη curr στον πίνακα.
                 // Ξεκινάμε την αρίθμηση από το 0.

public:
  Iterator(const HashTable *t);
  Iterator(const HashTable *t, bool start 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->();
  int pos() const;
};
```

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

Μέθοδος	Περιγραφή
`Iterator(const HashTable *t, bool start = true);`	Κατασκευαστής που λαμβάνει ως όρισμα ένα δείκτη στη δομή. Το πεδίο **curr** αρχικοποιείται με βάση την τιμή της 2ης μεταβλητής ως εξής: • Εάν η παράμετρος start έχει την τιμή `true` (default

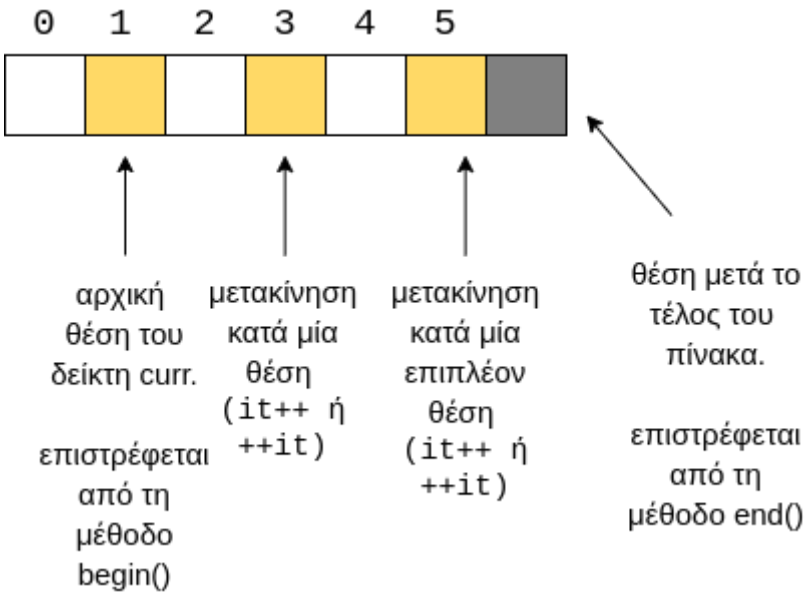
	τιμή) τότε το πεδίο curr δείχνει στο πρώτο μη κενό στοιχείο του πίνακα. • Εάν η παράμετρος start έχει την τιμή `false` τότε το πεδίο curr δείχνει μετά το τελευταίο στοιχείο του πίνακα.
`Iterator(const Iterator &it);`	*Copy constructor*
`Iterator& operator=(const Iterator &it);`	Υπερφόρτωση του τελεστή '**=**'. Στον αριστερό τελεστέο ανατίθεται το περιεχόμενο του δεξιού τελεστέου.
`Iterator operator++();`	Τελεστής μεταβολής της θέσης του τρέχοντος *Iterator* κατά μία θέση. Επιστρέφει ένα αντικείμενο τύπου ***Iterator*** που περιέχει την ανανεωμένη θέση του δείκτη **curr.**
`Iterator operator++(int a);`	Τελεστής μεταβολής της θέσης του τρέχοντος Iterator κατά μία θέση. Επιστρέφει ένα αντικείμενο τύπου ***Iterator*** που περιέχει την αρχική θέση του δείκτη **curr** πριν τη μεταβολή.
`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 μία αναφορά στο string που είναι αποθηκευμένο στη διεύθυνση που δείχνει ο δείκτης ***curr***.
`string* operator->();`	Επιστρέφει τη διεύθυνση του string που είναι αποθηκευμένο στη διεύθυνση που δείχνει ο δείκτης ***curr***.
`int pos() const;`	Επιστρέφεται η θέση του πίνακα στην οποία δείχνει ο δείκτης curr. Η αρίθμηση ξεκινά από την θέση 0.

Επεξήγηση

Στο παρακάτω σχήμα δίνεται η μεταβολή της θέσης του δείκτη curr του Iterator κατά τη διάτρεξη του πίνακα. Ένας τυπικός κώδικας διάτρεξης μέσω Iterator είναι ο παρακάτω:

```
for(HashTable::Iterator it = begin(); it!=end(); it++) {
    sprintf(buf, "%2d. -%s-\n", it.pos(), it->c_str());
    str.append(buf);
}
```

Ας υποθέσουμε ότι ο παραπάνω κώδικας διατρέχει τον πίνακα του παρακάτω σχήματος μεγέθους 6 θέσεων. Οι λευκές θέσεις είναι κενές ή κενωθείσες ενώ οι κίτρινες θέσεις έχουν περιεχόμενο. Η θέση με σκούρο γκρι χρώμα σηματοδοτεί τη θέση ΜΕΤΑ την τελευταία θέση του πίνακα.



- Αρχικά η μέθοδος begin() επιστρέφει ένα δείκτη στη θέση 1, γιατί η θέση 0 δεν έχει περιεχόμενο.
- Στην επόμενη επανάληψη ο δείκτης curr μετακινείται στη θέση 3 (προσπερνάει τη 2 που επίσης δεν έχει περιεχόμενο).
- Ομοίως, στην επόμενη επανάληψη ο δείκτης curr μετακινείται στη θέση 5 (προσπερνάει την 4).
- Μετά τη θέση 5, ο δείκτης καταλήγει μετά το τέλος του πίνακα. Σε αυτό το σημείο τερματίζει η επανάληψη.

Εάν σας εξυπηρετεί μπορείτε να δεσμεύσετε μία επιπλέον θέση για τον πίνακα, για τη θέση μετά το τέλος αυτού. Αυτή τη θέση θα την έχετε μόνο ως τερματική θέση και και όχι ως περιεχόμενο του πίνακα.

Παρατηρήσεις:

1. Κατά την διάτρεξη με τη βοήθεια του ***Iterator***, αυτός επιστρέφει μόνο θέσεις στις οποίες υπάρχουν λέξεις και όχι κενές ή διαγραμμένες θέσεις.
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 την κλάση ***ExtHashTable (ExtensibleHashTable)*** η οποία αποτελεί επέκταση της κλάσης HashTable και έχει τα εξής χαρακτηριστικά:

1. Πριν από την επιτυχή ένθεση ενός στοιχείου εξετάζει εάν ο παράγων φόρτου ***size/capacity*** είναι μεγαλύτερος από το προκαθορισμένο ποσοστό ***upper_bound_ratio***. Εάν ναι, διπλασιάζει τη χωρητικότητα του πίνακα.
2. Μετά τη διαγραφή ενός στοιχείου εξετάζει εάν ο παράγων φόρτου ***size/capacity*** είναι μικρότερος από το προκαθορισμένο ποσοστό ***lower_bound_ratio***. Εάν ναι, υπο-διπλασιάζει τη χωρητικότητα του 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;

  ExtHashTable& operator += (const string &str);
  ExtHashTable& operator += (const char* s);
  ExtHashTable& operator -= (const string &str);
  ExtHashTable& 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=8);`	Κατασκευαστής.
`ExtHashTable(ExtHashTable &t);`	*Copy constructor*
`void rehash();`	Εάν ο αριθμός των αποθηκευμένων στοιχείων του πίνακα είναι μηδέν (0), η μέθοδος επιστρέφει άμεσα χωρίς να μεταβάλλει το μέγεθος του πίνακα. Εξετάζει εάν ο παράγων φόρτου **size/capacity** είναι μεγαλύτερος από το προκαθορισμένο ποσοστό

	upper_bound_ratio. Εάν είναι μεγαλύτερος, διπλασιάζει τη χωρητικότητα του πίνακα. Εξετάζει εάν ο παράγων φόρτου **size/capacity** είναι μικρότερος από το προκαθορισμένο ποσοστό **lower_bound_ratio**. Εάν είναι μικρότερος, υποδιπλασιάζει τη χωρητικότητα του πίνακα. Κάθε φορά που αλλάζει το μέγεθος του πίνακα εκτυπώνει στην καθιερωμένη είσοδο ένα μήνυμα της μορφής: **--> Size: X, New capacity: Y** όπου **X** αριθμός των αποθηκευμένων στοιχείων του πίνακα και **Y** η χωρητικότητα του.
bool add(const string &str);	Όμοια με την συνάρτηση της γονικής κλάσης, με τη διαφορά ότι καλεί τη συνάρτηση **rehash(void)** και δεν παράγει **HashtableException**.
bool add(const char *s);	Λειτουργικότητα όμοια με τη συνάρτηση **bool add(const string &str);**
bool remove(const string &str);	Όμοια με την συνάρτηση της γονικής κλάσης, με τη διαφορά ότι καλεί τη συνάρτηση **rehash(void)** μετά την επιτυχή διαγραφή ~~και δεν παράγει **HashtableException**~~.
bool remove(const char *s);	Λειτουργικότητα όμοια με τη συνάρτηση **bool remove(const string &str);**
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***.
ExtHashTable operator+(const char *s) const;	Λειτουργικότητα όμοια με τη συνάρτηση **ExtHashTable operator+(const string &str) const;**
ExtHashTable operator-(const string &str) const;	Παράγει ένα νέο ***ExtHashTable*** που προκύπτει από τη διαγραφή του ***str*** από το υφιστάμενο ***ExtHashTable***.
ExtHashTable operator-(const char *s) const;	Λειτουργικότητα όμοια με τη συνάρτηση **ExtHashTable operator-(const string &str) const;**
ExtHashTable& operator += (const string &str);	Ενθέτει το αλφαριθμητικό **str** στο τρέχον αντικείμενο και επιστρέφει μία αναφορά σε αυτό.
ExtHashTable& operator += (const char *);	Λειτουργικότητα όμοια με τη συνάρτηση **ExtHashTable& operator += (const string &str)**
ExtHashTable& operator -= (const string &str);	Διαγράφει το αλφαριθμητικό **str** από το τρέχον αντικείμενο και επιστρέφει μία αναφορά σε αυτό.
ExtHashTable& operator -= (const char *);	Λειτουργικότητα όμοια με τη συνάρτηση **bool operator -= (const string &str);**
ExtHashTable operator+(const ExtHashTable &t) const;	Παράγει ένα νέο ***ExtHashTable*** που προκύπτει από την ένωση του ***t*** με το υφιστάμενο ***ExtHashTable***. Η διαδικασία

	ένωσης έχει ως εξής: 1. Δημιουργείται ένα νέο HashTable που είναι αντίγραφο του αριστερού τελεστέου. 2. Διατρέχουμε το `t` από την αρχή έως το τέλος και προσθέτουμε κάθε στοιχείο του στο αντίγραφο HashTable. 3. Επιστρέφουμε το αντίγραφο.
`ExtHashTable &operator+=(const ExtHashTable &t);`	Ενθέτει το **t** στο υφιστάμενο ***ExtHashTable*** και επιστρέφει μία αναφορά στο υφιστάμενο ***ExtHashTable***. Κατά την ένθεση διατρέχουμε το `t` από την αρχή έως το τέλος και προσθέτουμε κάθε στοιχείο του στο τρέχον ***ExtHashTable***.

Γενικές Οδηγίες

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

Τα αρχεία που πρέπει να περιέχει η εργασία σας είναι τα εξής:

- `HashTable.hpp`: Header file για τις κλάσεις του `HashTable` και `HashTable::Iterator`.
- `HashTable.cpp`: Αρχείο που περιέχει τις υλοποιήσεις για την κλάση `HashTable`.
- `HashTableIterator.cpp`: Περιέχει τις υλοποιήσεις για την κλάση `HashTable::Iterator`. Το αρχείο μπορεί να είναι κενό (πρέπει όμως να υπάρχει) και τις υλοποιήσεις του Iterator να τις έχετε στο αρχείο `HashTable.cpp`.
- `ExtHashTable.hpp`: Header file για την κλάση `ExtHashTable`.
- `ExtHashTable.cpp`: Υλοποίηση για την κλάση `ExtHashTable`.

Μέρος του Makefile που χρησιμοποιείται για τη μεταγλώττιση δίνεται παρακάτω:

```
CFLAGS=-Wall -g -std=c++11 -fsanitize=address

HashTable.o: HashTable.cpp
	g++ ${CFLAGS} HashTable.cpp -c

HashTableIterator.o: HashTableIterator.cpp
	g++ ${CFLAGS} HashTableIterator.cpp -c

ExtHashTable.o: ExtHashTable.cpp
	g++ ${CFLAGS} ExtHashTable.cpp -c

libHashTable: HashTable.o HashTableIterator.o ExtHashTable.o
	ar cr  HashTable.o HashTableIterator.o ExtHashTable.o
```

Τρόπος Αποστολής

Η αποστολή της εργασίας θα γίνει μέσω της πλατφόρμας autolab. Για την υποβολή ακολουθήστε τα εξής βήματα:

Συμπιέστε σε μορφή zip, από το project σας τα περιεχόμενα του καταλόγου που περιέχει τον πηγαίο κώδικα της εργασίας σας, δηλαδή τα αρχεία `HashTable.hpp, HashTable.cpp, HashTableException.hpp, HashTableIterator.cpp, ExtHashTable.hpp, ExtHashTable.cpp`.

Το αρχείο που προκύπτει πρέπει να έχει όνομα **`hw4.zip`**.

Συνδέεστε στο autolab (μέσω VPN) και επιλέγετε το μάθημα **ECE326_2022 (S22)** και από αυτό την εργασία **HW4**.

Για να υποβάλετε την εργασία σας κάνετε click στην επιλογή “I affirm that I have compiled with this course academic integrity policy…” και πατάτε submit. Στη συνέχεια επιλέγετε το αρχείο hw1.zip που δημιουργήσατε παραπάνω.