Δομές Δεδομένων και Αλγόριθμοι - Εργαστήριο 4
Γραμμικές λίστες (στατικές λίστες και συνδεδεμένες λίστες) και λίστες της STL

Τ.Ε.Ι. Ηπείρου, Τμήμα Μηχανικών Πληροφορικής Τ.Ε.
Χρήστος Γκόγκος - Αναπληρωτής Καθηγητής

1 Εισαγωγή

Οι γραμμικές λίστες είναι δομές δεδομένων που επιτρέπουν την αποθήκευση και την προσπέλαση στοιχείων έτσι ώστε τα στοιχεία να βρίσκονται σε μια σειρά με σαφώς ορισμένη την έννοια της θέσης καθώς και το ποιο στοιχείο προηγείται και ποιο έπεται καθενός. Σε χαμηλού επιπέδου γλώσσες προγραμματισμού όπως η C η υλοποίηση γραμμικών λιστών είναι ευθύνη του προγραμματιστή. Από την άλλη μεριά, γλώσσες υψηλού επιπέδου όπως η C++, η Java, η Python κ.α. προσφέρουν έτοιμες υλοποιήσεις γραμμικών λιστών. Ωστόσο, η γνώση υλοποίησης των συγκεκριμένων δομών (όπως και άλλων) αποτελεί βασική ικανότητα η οποία αποκτά ιδιαίτερη χρησιμότητα όταν ζητούνται εξειδικευμένες υλοποιήσεις. Στο συγκεκριμένο εργαστήριο θα παρουσιαστούν οι υλοποιήσεις γραμμικών λιστών αλλά και οι ενσωματωμένες δυνατότητες της C++ μέσω containers της STL όπως το vector και το list.

2 Γραμμικές λίστες

Υπάρχουν δύο βασικοί τρόποι αναπαράστασης γραμμικών λιστών, η στατική αναπαράσταση η οποία γίνεται με τη χρήση πινάκων και η αναπαράσταση με συνδεδεμένη λίστα η οποία γίνεται με τη χρήση δεικτών.

2.1 Στατικές γραμμικές λίστες

Στη στατική γραμμική λίστα τα δεδομένα αποθηκεύονται σε ένα πίνακα. Κάθε στοιχείο της στατικής λίστας μπορεί να προσπελαστεί με βάση τη θέση του στον ίδιο σταθερό χρόνο με όλα τα άλλα στοιχεία άσχετα με τη θέση στην οποία βρίσκεται (τυχαία προσπέλαση). Ο κώδικας υλοποίησης μιας στατικής λίστας με μέγιστη χωρητικότητα 50.000 στοιχείων παρουσιάζεται στη συνέχεια.

```
1  #include <iostream>
2
3  using namespace std;
4
5  const int MAX = 50000;
6  template <class T> struct static_list {
7     T elements[MAX];
8     int size = 0;
9  };
10
11 // get item at position i
12 template <class T> T access(static_list<T> &static_list, int i) {
13    if (i < 0 || i >= static_list.size)
14       throw out_of_range("the index is out of range");
15    else
16       return static_list.elements[i];
```

```
}

// get the position of item x
template <class T> int search(static_list<T> &static_list, T x) {
   for (int i = 0; i < static_list.size; i++)
      if (static_list.elements[i] == x)
         return i;
   return −1;
}

// append item x at the end of the list
template <class T> void push_back(static_list<T> &static_list, T x) {
   if (static_list.size == MAX)
      throw ”full list exception”;
   static_list.elements[static_list.size] = x;
   static_list.size++;
}

// append item x at position i, shift the rest to the right
template <class T> void insert(static_list<T> &static_list, int i, T x) {
   if (static_list.size == MAX)
      throw ”full list exception”;
   if (i < 0 || i >= static_list.size)
      throw out_of_range(”the index is out of range”);
   for (int k = static_list.size; k > i; k−−)
      static_list.elements[k] = static_list.elements[k − 1];
   static_list.elements[i] = x;
   static_list.size++;
}

// delete item at position i, shift the rest to the left
template <class T> void delete_item(static_list<T> &static_list, int i) {
   if (i < 0 || i >= static_list.size)
      throw out_of_range(”the index is out of range”);
   for (int k = i; k < static_list.size; k++)
      static_list.elements[k] = static_list.elements[k + 1];
   static_list.size−−;
}

template <class T> void print_list(static_list<T> &static_list) {
   cout << ”List: ”;
   for (int i = 0; i < static_list.size; i++)
      cout << static_list.elements[i] << ” ”;
   cout << endl;
}
```

Κώδικας 1: Υλοποίηση στατικής γραμμικής λίστας (static_list.cpp)

```
#include ”static_list.cpp”
#include <iostream>

using namespace std;

int main(void) {
   static_list<int> alist;
   cout << ”#1. Add items 10, 20 and 30” << endl;
   push_back(alist, 10);
   push_back(alist, 20);
   push_back(alist, 30);
   print_list(alist);
   cout << ”#2. Insert at position 1 item 15” << endl;
```

```
14    insert(alist, 1, 15);
15    print_list(alist);
16    cout << "#3. Delete item at position 0" << endl;
17    delete_item(alist, 0);
18    print_list(alist);
19    cout << "#4. Item at position 2: " << access(alist, 2) << endl;
20    try {
21       cout << "#5. Item at position -1" << access(alist, -1) << endl;
22    } catch (out_of_range oor) {
23       cerr << "Exception: " << oor.what() << endl;
24    }
25    cout << "#6. Search for item 20: " << search(alist, 20) << endl;
26    cout << "#7. Search for item 21: " << search(alist, 21) << endl;
27    cout << "#8. Append item 99 until full list exception occurs" << endl;
28    try {
29       while (true)
30          push_back(alist, 99);
31    } catch (const char *msg) {
32       cerr << "Exception: " << msg << endl;
33    }
34 }
```

Κώδικας 2: Παράδειγμα με στατική γραμμική λίστα (list1.cpp)

Στους κώδικες που προηγήθηκαν καθώς και σε επόμενους γίνεται χρήση εξαιρέσεων (exceptions) για να σηματοδοτηθούν γεγονότα τα οποία αφορούν έκτακτες καταστάσεις που το πρόγραμμα θα πρέπει να διαχειρίζεται. Για παράδειγμα, όταν επιχειρηθεί η προσπέλαση ενός στοιχείου σε μια θέση εκτός των ορίων της λίστας (π.χ. ενέργεια 5 στον κώδικα 2) τότε γίνεται throw ένα exception out_of_range το οποίο θα πρέπει να συλληφθεί (να γίνει catch) από τον κώδικα που καλεί τη συνάρτηση που προκάλεσε το throw exception. Περισσότερες πληροφορίες για τα exceptions και τον χειρισμό τους μπορούν να αναζητηθούν στην αναφορά [1].

```
1  #1. Add items 10, 20 and 30
2  List: 10 20 30
3  #2. Insert at position 1 item 15
4  List: 10 15 20 30
5  #3. Delete item at position 0
6  List: 15 20 30
7  #4. Item at position 2: 30
8  Exception: the index is out of range
9  #6. Search for item 20: 1
10 #7. Search for item 21: -1
11 #8. Append item 99 until full list exception occurs
12 Exception: full list exception
```

Οι στατικές γραμμικές λίστες έχουν τα ακόλουθα πλεονεκτήματα:

- Εύκολη υλοποίηση.
- Σταθερός χρόνος $O(1)$ εντοπισμού στοιχείου με βάση τη θέση του.
- Γραμμικός χρόνος $O(n)$ για αναζήτηση ενός στοιχείου ή λογαριθμικός χρόνος $O(log(n))$ αν τα στοιχεία είναι ταξινομημένα.

Ωστόσο, οι στατικές γραμμικές λίστες έχουν και μειονεκτήματα τα οποία παρατίθενται στη συνέχεια:

- Δέσμευση μεγάλου τμήματος μνήμης ακόμη και όταν η λίστα είναι άδεια ή περιέχει λίγα στοιχεία.
- Επιβολή άνω ορίου στα δεδομένα τα οποία μπορεί να δεχθεί (ο περιορισμός αυτός μπορεί να ξεπεραστεί με συνθετότερη υλοποίηση που αυξομειώνει το μέγεθος του πίνακα υποδοχής όταν αυτό απαιτείται).
- Γραμμικός χρόνος $O(n)$ για εισαγωγή και διαγραφή στοιχείων του πίνακα.

2.2 Συνδεδεμένες γραμμικές λίστες

Η συνδεδεμένη γραμμική λίστα αποτελείται από μηδέν ή περισσότερους κόμβους. Κάθε κόμβος περιέχει δεδομένα και έναν ή περισσότερους δείκτες σε άλλους κόμβους της συνδεδεμένης λίστας. Συχνά χρησιμο-

ποιείται ένας πρόσθετος κόμβος με όνομα head (κόμβος κεφαλής) που δείχνει στο πρώτο στοιχείο της λίστας και μπορεί να περιέχει επιπλέον πληροφορίες όπως το μήκος της. Στη συνέχεια παρουσιάζεται ο κώδικας που υλοποιεί μια απλά συνδεδεμένη λίστα.

```
#include <iostream>

using namespace std;

template <class T> struct node {
   T data;
   struct node<T> *next = NULL;
};

template <class T> struct linked_list {
   struct node<T> *head = NULL;
   int size = 0;
};

// get node item at position i
template <class T>
struct node<T> *access_node(linked_list<T> &linked_list, int i) {
   if (i < 0 || i >= linked_list.size)
      throw out_of_range("the index is out of range");
   struct node<T> *current = linked_list.head;
   for (int k = 0; k < i; k++)
      current = current->next;
   return current;
}

// get node item at position i
template <class T>
T access(linked_list<T> &linked_list, int i) {
   struct node<T> *item = access_node(linked_list, i);
   return item->data;
}

// get the position of item x
template <class T> int search(linked_list<T> &linked_list, T x) {
   struct node<T> *current = linked_list.head;
   int i = 0;
   while (current != NULL) {
      if (current->data == x)
         return i;
      i++;
      current = current->next;
   }
   return -1;
}

// append item x at the end of the list
template <class T> void push_back(linked_list<T> &l, T x) {
   struct node<T> *new_node, *current;
   new_node = new node<T>();
   new_node->data = x;
   new_node->next = NULL;
   current = l.head;
   if (current == NULL) {
      l.head = new_node;
      l.size++;
   } else {
      while (current->next != NULL)
```

```
58          current = current->next;
59        current->next = new_node;
60        l.size++;
61      }
62  }
63
64  // append item x after position i
65  template <class T> void insert_after(linked_list<T> &linked_list, int i, T x) {
66      if (i < 0 || i >= linked_list.size)
67          throw out_of_range("the index is out of range");
68      struct node<T> *ptr = access_node(linked_list, i);
69      struct node<T> *new_node = new node<T>();
70      new_node->data = x;
71      new_node->next = ptr->next;
72      ptr->next = new_node;
73      linked_list.size++;
74  }
75
76  // append item at the head
77  template <class T> void insert_head(linked_list<T> &linked_list, T x) {
78      struct node<T> *new_node = new node<T>();
79      new_node->data = x;
80      new_node->next = linked_list.head;
81      linked_list.head = new_node;
82      linked_list.size++;
83  }
84
85  // append item x at position i
86  template <class T> void insert(linked_list<T> &linked_list, int i, T x) {
87      if (i == 0)
88          insert_head(linked_list, x);
89      else
90          insert_after(linked_list, i - 1, x);
91  }
92
93  // delete item at position i
94  template <class T> void delete_item(linked_list<T> &l, int i) {
95      if (i < 0 || i >= l.size)
96          throw out_of_range("the index is out of range");
97      if (i == 0) {
98          struct node<T> *ptr = l.head;
99          l.head = ptr->next;
100         delete ptr;
101     } else {
102         struct node<T> *ptr = access_node(l, i - 1);
103         struct node<T> *to_be_deleted = ptr->next;
104         ptr->next = to_be_deleted->next;
105         delete to_be_deleted;
106     }
107     l.size--;
108 }
109
110 template <class T> void print_list(linked_list<T> &l) {
111     cout << "List: ";
112     struct node<T> *current = l.head;
113     while (current != NULL) {
114         cout << current->data << " ";
115         current = current->next;
116     }
117     cout << endl;
118 }
```

Κώδικας 3: Υλοποίηση συνδεδεμένης γραμμικής λίστας (linked_list.cpp)

```
#include "linked_list.cpp"
#include <iostream>

using namespace std;

int main(int argc, char *argv[]) {
   linked_list<int> alist;
   cout << "#1. Add items 10, 20 and 30" << endl;
   push_back(alist, 10);
   push_back(alist, 20);
   push_back(alist, 30);
   print_list(alist);
   cout << "#2. Insert at position 1 item 15" << endl;
   insert(alist, 1, 15);
   print_list(alist);
   cout << "#3. Delete item at position 0" << endl;
   delete_item(alist, 0);
   print_list(alist);
   cout << "#4. Item at position 2: " << access(alist, 2) << endl;
   try {
      cout << "#5. Item at position -1" << access(alist, -1) << endl;
   } catch (out_of_range oor) {
      cerr << "Exception: " << oor.what() << endl;
   }
   cout << "#6. Search for item 20: " << search(alist, 20) << endl;
   cout << "#7. Search for item 21: " << search(alist, 21) << endl;
   cout << "#8. Delete allocated memory " << endl;
   for (int i = 0; i < alist.size; i++)
      delete_item(alist, i);
}
```

Κώδικας 4: Παράδειγμα με συνδεδεμένη γραμμική λίστα (list2.cpp)

```
#1. Add items 10, 20 and 30
List: 10 20 30
#2. Insert at position 1 item 15
List: 10 15 20 30
#3. Delete item at position 0
List: 15 20 30
#4. Item at position 2: 30
Exception: the index is out of range
#6. Search for item 20: 1
#7. Search for item 21: -1
#8. Delete allocated memory
```

Οι συνδεδεμένες γραμμικές λίστες έχουν τα ακόλουθα πλεονεκτήματα:

- Καλή χρήση του αποθηκευτικού χώρου (αν και απαιτείται περισσότερος χώρος για την αποθήκευση κάθε κόμβου λόγω των δεικτών).
- Σταθερός χρόνος $O(1)$ για την εισαγωγή και διαγραφή στοιχείων.

Από την άλλη μεριά τα μειονεκτήματα των συνδεδεμένων λιστών είναι τα ακόλουθα:

- Συνθετότερη υλοποίηση.
- Δεν επιτρέπουν την απευθείας μετάβαση σε κάποιο στοιχείο με βάση τη θέση του.

2.3 Γραμμικές λίστες της STL

Τα containers της STL που μπορούν να λειτουργήσουν ως διατεταγμένες συλλογές (ordered collections) είναι τα ακόλουθα: vector, deque, arrays, list και forward_list.

2.3.1 Vectors

Τα vectors αλλάζουν αυτόματα μέγεθος καθώς προστίθενται ή αφαιρούνται στοιχεία σε αυτά. Τα δεδομένα τους τοποθετούνται σε συνεχόμενες θέσεις μνήμης. Περισσότερες πληροφορίες για τα vectors μπορούν να βρεθούν στην αναφορά [2].

2.3.2 Deques

Τα deques (double ended queues = ουρές με δύο άκρα) είναι παρόμοια με τα vectors αλλά μπορούν να προστεθούν ή να διαγραφούν στοιχεία τόσο από την αρχή όσο και από το τέλος τους. Περισσότερες πληροφορίες για τα deques μπορούν να βρεθούν στην αναφορά [3].

2.3.3 Arrays

Τα arrays εισήχθησαν στη C++11 με στόχο να αντικαταστήσουν τους απλούς πίνακες της C. Κατά τη δήλωση ενός array προσδιορίζεται και το μέγεθός του. Περισσότερες πληροφορίες για τα arrays μπορούν να βρεθούν στην αναφορά [4].

2.3.4 Lists

Οι lists είναι διπλά συνδεδεμένες λίστες. Δηλαδή κάθε κόμβος της λίστας διαθέτει έναν δείκτη προς το επόμενο και έναν δείκτη προς το προηγούμενο στοιχείο στη λίστα. Περισσότερες πληροφορίες για τις lists μπορούν να βρεθούν στην αναφορά [5].

2.3.5 Forward Lists

Οι forward lists είναι απλά συνδεδεμένες λίστες με κάθε κόμβο να διαθέτει έναν δείκτη προς το επόμενο στοιχείο της λίστας. Περισσότερες πληροφορίες για τις forward lists μπορούν να βρεθούν στις αναφορές [6], [7].

3 Παραδείγματα

3.1 Παράδειγμα 1

Γράψτε ένα πρόγραμμα που να ελέγχεται από το ακόλουθο μενού και να πραγματοποιεί τις λειτουργίες που περιγράφονται σε μια απλά συνδεδεμένη λίστα με ακεραίους.

1. Εμφάνιση στοιχείων λίστας. (Show list)
2. Εισαγωγή στοιχείου στο πίσω άκρο της λίστας. (Insert item (back))
3. Εισαγωγή στοιχείου σε συγκεκριμένη θέση. (Insert item (at position))
4. Διαγραφή στοιχείου σε συγκεκριμένη θέση. (Delete item (from position))
5. Διαγραφή όλων των στοιχείων που έχουν την τιμή. (Delete all items having value)
6. Έξοδος. (Exit)

```
1  #include ”linked_list.cpp”
2  #include <iostream>
3
4  using namespace std;
5
6  int main(int argc, char **argv) {
7    linked_list<int> alist;
8    int choice, position, value;
9    do {
10     cout << ”1. Show list” << endl;
11     cout << ”2. Insert item (back)” << endl;
```

```
    cout << "3. Insert item (at position)" << endl;
    cout << "4. Delete item (from position)" << endl;
    cout << "5. Delete all items having value" << endl;
    cout << "6. Exit" << endl;
    cout << "Enter choice: ";
    cin >> choice;
    if (choice < 1 || choice > 6) {
      cerr << "Choice should be 1 to 5" << endl;
      continue;
    }
    try {
      switch (choice) {
      case 1:
        print_list(alist);
        break;
      case 2:
        cout << "Enter value:";
        cin >> value;
        push_back(alist, value);
        break;
      case 3:
        cout << "Enter position and value:";
        cin >> position >> value;
        insert(alist, position, value);
        break;
      case 4:
        cout << "Enter position:";
        cin >> position;
        delete_item(alist, position);
        break;
      case 5:
        cout << "Enter value:";
        cin >> value;
        int i = 0;
        while (i < alist.size)
          if (access(alist, i) == value)
            delete_item(alist, i);
          else
            i++;
      }
    } catch (out_of_range oor) {
      cerr << "Out of range, try again" << endl;
    }
  } while (choice != 6);
}
```

Κώδικας 5: Έλεγχος συνδεδεμένης λίστας ακεραίων μέσω μενού (lab04_ex1.cpp)

```
1. Show list
2. Insert item (back)
3. Insert item (at position)
4. Delete item (from position)
5. Delete all items having value
6. Exit
Enter choice: 2
Enter value:10
1. Show list
2. Insert item (back)
3. Insert item (at position)
4. Delete item (from position)
5. Delete all items having value
6. Exit
Enter choice: 2
```

```
Enter value:20
1. Show list
2. Insert item (back)
3. Insert item (at position)
4. Delete item (from position)
5. Delete all items having value
6. Exit
Enter choice: 3
Enter position and value:1 30
1. Show list
2. Insert item (back)
3. Insert item (at position)
4. Delete item (from position)
5. Delete all items having value
6. Exit
Enter choice: 1
List: 10 30 20
1. Show list
2. Insert item (back)
3. Insert item (at position)
4. Delete item (from position)
5. Delete all items having value
6. Exit
Enter choice: 6
```

3.2 Παράδειγμα 2

Έστω μια τράπεζα που διατηρεί για κάθε πελάτη της το ονοματεπώνυμό του και το υπόλοιπο του λογαριασμού του. Για τις ανάγκες του παραδείγματος θα δημιουργηθούν τυχαίοι πελάτες ως εξής: το όνομα κάθε πελάτη θα αποτελείται από 10 γράμματα που θα επιλέγονται με τυχαίο τρόπο από τα γράμματα της αγγλικής αλφαβήτου και το δε υπόλοιπο κάθε πελάτη θα είναι ένας τυχαίος ακέραιος αριθμός από το 0 μέχρι το 5.000. Το πρόγραμμα θα πραγματοποιεί τις ακόλουθες λειτουργίες:

Α΄ Θα δημιουργεί μια λίστα με 40.000 τυχαίους πελάτες.
Β΄ Θα υπολογίζει το άθροισμα των υπολοίπων από όλους τους πελάτες που το όνομά τους ξεκινά με το χαρακτήρα Α.
Γ΄ Θα προσθέτει για κάθε πελάτη που το όνομά του ξεκινά με το χαρακτήρα G στην αμέσως επόμενη θέση έναν πελάτη με όνομα το αντίστροφο όνομα του πελάτη και το ίδιο υπόλοιπο λογαριασμού.
Δ΄ Θα διαγράφει όλους τους πελάτες που το όνομά τους ξεκινά με το χαρακτήρα Β.

Τα δεδομένα θα αποθηκεύονται σε μια συνδεδεμένη λίστα πραγματοποιώντας χρήση του κώδικα 3 καθώς και άλλων συναρτήσεων που επιτρέπουν την αποδοτικότερη υλοποίηση των παραπάνω ερωτημάτων.

```cpp
#include "linked_list.cpp"
#include <algorithm>
#include <chrono>
#include <iomanip>
#include <iostream>
#include <list>
#include <random>
#include <string>

using namespace std;
using namespace std::chrono;

mt19937 *mt;
uniform_int_distribution<int> uni1(0, 5000);
uniform_int_distribution<int> uni2(0, 25);

struct customer {
  string name;
```

```
19    int balance;
20    bool operator<(customer other) { return name < other.name; }
21  };
22
23  string generate_random_name(int k) {
24    string name{};
25    string letters_en("ABCDEFGHIJKLMNOPQRSTUVWXYZ");
26    for (int j = 0; j < k; j++) {
27      char c{letters_en[uni2(*mt)]};
28      name += c;
29    }
30    return name;
31  }
32
33  void generate_data_linked_list(linked_list<customer> &linked_list, int N) {
34    struct node<customer> *current, *next_customer;
35    current = new node<customer>();
36    current->data.name = generate_random_name(10);
37    current->data.balance = uni1(*mt);
38    current->next = NULL;
39    linked_list.head = current;
40    linked_list.size++;
41    for (int i = 1; i < N; i++) {
42      next_customer = new node<customer>();
43      next_customer->data.name = generate_random_name(10);
44      next_customer->data.balance = uni1(*mt);
45      next_customer->next = NULL;
46      current->next = next_customer;
47      current = next_customer;
48      linked_list.size++;
49    }
50  }
51
52  // slower alternative for data generation
53  void generate_data_linked_list_alt(linked_list<customer> &linked_list, int N) {
54    for (int i = 0; i < N; i++) {
55      customer c;
56      c.name = generate_random_name(10);
57      c.balance = uni1(*mt);
58      push_back(linked_list, c);
59    }
60  }
61
62  void print_customers_linked_list(linked_list<customer> &linked_list, int k) {
63    cout << "LIST SIZE=" << linked_list.size << ": ";
64    for (int i = 0; i < k; i++) {
65      customer cu = access(linked_list, i);
66      cout << cu.name << " - " << cu.balance << " ";
67    }
68    cout << endl;
69  }
70
71  void total_balance_linked_list(linked_list<customer> &linked_list, char c) {
72    struct node<customer> *ptr;
73    ptr = linked_list.head;
74    int i = 0;
75    int sum = 0;
76    while (ptr != NULL) {
77      customer cu = ptr->data;
78      if (cu.name.at(0) == c)
79        sum += cu.balance;
```

```
80      ptr = ptr->next;
81      i++;
82    }
83    cout << "Total balance for customers having name starting with character "
84         << c << " is " << sum << endl;
85  }
86
87  void add_extra_customers_linked_list(linked_list<customer> &linked_list,
88                                       char c) {
89    struct node<customer> *ptr = linked_list.head;
90    while (ptr != NULL) {
91      customer cu = ptr->data;
92      if (cu.name.at(0) == c) {
93        customer ncu;
94        ncu.name = cu.name;
95        reverse(ncu.name.begin(), ncu.name.end());
96        ncu.balance = cu.balance;
97        struct node<customer> *new_node = new node<customer>();
98        new_node->data = ncu;
99        new_node->next = ptr->next;
100       ptr->next = new_node;
101       linked_list.size++;
102       ptr = new_node->next;
103     } else
104       ptr = ptr->next;
105   }
106 }
107
108 void remove_customers_linked_list(linked_list<customer> &linked_list, char c) {
109   struct node<customer> *ptr1;
110   while (linked_list.size > 0) {
111     customer cu = linked_list.head->data;
112     if (cu.name.at(0) == c) {
113       ptr1 = linked_list.head;
114       linked_list.head = ptr1->next;
115       delete ptr1;
116       linked_list.size--;
117     } else
118       break;
119   }
120   if (linked_list.size == 0)
121     return;
122   ptr1 = linked_list.head;
123   struct node<customer> *ptr2 = ptr1->next;
124   while (ptr2 != NULL) {
125     customer cu = ptr2->data;
126     if (cu.name.at(0) == c) {
127       ptr1->next = ptr2->next;
128       delete (ptr2);
129       ptr2 = ptr1->next;
130       linked_list.size--;
131     } else {
132       ptr1 = ptr2;
133       ptr2 = ptr2->next;
134     }
135   }
136 }
137
138 // slower alternative for customer removal
139 void remove_customers_linked_list_alt(linked_list<customer> &linked_list,
140                                       char c) {
```

```
141   int i = 0;
142   while (i < linked_list.size) {
143     struct customer cu = access(linked_list, i);
144     if (cu.name.at(0) == c)
145       delete_item(linked_list, i);
146     else
147       i++;
148   }
149 }
150
151 int main(int argc, char **argv) {
152   long seed = 1940;
153   mt = new mt19937(seed);
154   cout << "Testing linked list" << endl;
155   cout << "############################################################" << endl;
156   struct linked_list<customer> linked_list;
157   auto t1 = high_resolution_clock::now();
158   generate_data_linked_list(linked_list, 40000);
159   // generate_data_linked_list_alt(linked_list, 40000);
160   auto t2 = high_resolution_clock::now();
161   print_customers_linked_list(linked_list, 5);
162   auto duration = duration_cast<microseconds>(t2 - t1).count();
163   cout << "A(random customers generation). Time elapsed: " << duration
164        << " microseconds " << setprecision(3) << duration / 1000000.0
165        << " seconds" << endl;
166
167   t1 = high_resolution_clock::now();
168   total_balance_linked_list(linked_list, 'A');
169   t2 = high_resolution_clock::now();
170   duration = duration_cast<microseconds>(t2 - t1).count();
171   cout << "B(total balance for customers having name starting with A). Time "
172           "elapsed: "
173        << duration << " microseconds " << setprecision(3)
174        << duration / 1000000.0 << " seconds" << endl;
175
176   t1 = high_resolution_clock::now();
177   add_extra_customers_linked_list(linked_list, 'G');
178   t2 = high_resolution_clock::now();
179   print_customers_linked_list(linked_list, 5);
180   duration = duration_cast<microseconds>(t2 - t1).count();
181   cout << "C(insert new customers). Time elapsed: " << duration
182        << " microseconds " << setprecision(3) << duration / 1000000.0
183        << " seconds" << endl;
184
185   t1 = high_resolution_clock::now();
186   remove_customers_linked_list(linked_list, 'B');
187   // remove_customers_linked_list_alt(linked_list, 'B');
188   t2 = high_resolution_clock::now();
189   print_customers_linked_list(linked_list, 5);
190   duration = duration_cast<microseconds>(t2 - t1).count();
191   cout << "D(remove customers). Time elapsed: " << duration << " microseconds "
192        << setprecision(3) << duration / 1000000.0 << " seconds" << endl;
193   cout << "############################################################" << endl;
194 }
```

Κώδικας 6: Λίστα πελατών για το ίδιο πρόβλημα (lab04_ex2.cpp)

```
1 Testing linked list
2 ############################################################
3 LIST SIZE=100000: GGFSICRZWW -- 2722 UBKZNBPWLH -- 4019 UPIHSBIIBS -- 3896 JRQVGHLTNM -- 395 LUWYTFTNFJ -- 784
4 A(random customers generation). Time elapsed: 88005 microseconds 0.088 seconds
5 Total balance for customers having name starting with character A is 9788898
```

```
B(total balance for customers having name starting with A). Time elapsed: 3000 microseconds 0.003 seconds
LIST SIZE=103904: GGFSICRZWW − 2722 WWZRCISFGG − 2722 UBKZNBPWLH − 4019 UPIHSBIIBS − 3896 JRQVGHLTNM − 395
C(insert new customers). Time elapsed: 3000 microseconds 0.003 seconds
LIST SIZE=99874: GGFSICRZWW − 2722 WWZRCISFGG − 2722 UBKZNBPWLH − 4019 UPIHSBIIBS − 3896 JRQVGHLTNM − 395
D(remove customers). Time elapsed: 2000 microseconds 0.002 seconds
##########################################################
```

Αν στη θέση της συνδεδεμένης λίστας του κώδικα 3 χρησιμοποιηθεί η στατική λίστα του κώδικα 1 ή ένα vector ή ένα list της STL τα αποτελέσματα θα είναι τα ίδια αλλά η απόδοση στα επιμέρους ερωτήματα του παραδείγματος θα διαφέρει όπως φαίνεται στον πίνακα 1. Ο κώδικας που παράγει τα αποτελέσματα βρίσκεται στο αρχείο lab04/lab04_ex2_x4.cpp.

	Ερώτημα Α	Ερώτημα Β	Ερώτημα Γ	Ερώτημα Δ
Συνδεδεμένη λίστα	0.030	0.001	0.002	0.001
Στατική λίστα	0.034	0.003	0.642	0.671
vector (STL)	0.033	0.002	0.543	0.519
list (STL)	0.033	0.002	0.002	0.001

Πίνακας 1: Χρόνοι εκτέλεσης σε δευτερόλεπτα των ερωτημάτων του παραδείγματος 2 ανάλογα με τον τρόπο υλοποίησης της λίστας

4 Ασκήσεις

1. Έστω η συνδεδεμένη λίστα που παρουσιάστηκε στον κώδικα 3. Προσθέστε μια συνάρτηση έτσι ώστε για μια λίστα ταξινομημένων στοιχείων από το μικρότερο προς το μεγαλύτερο, να προσθέτει ένα ακόμα στοιχείο στην κατάλληλη θέση έτσι ώστε η λίστα να παραμένει ταξινομημένη.

2. Έστω η συνδεδεμένη λίστα που παρουσιάστηκε στον κώδικα 3. Προσθέστε μια συνάρτηση που να αντιστρέφει τη λίστα.

3. Υλοποιήστε τη στατική λίστα (κώδικας 1) και τη συνδεδεμένη λίστα (κώδικας 3) με κλάσεις. Τροποποιήστε το παράδειγμα 1 έτσι ώστε να δίνεται επιλογή στο χρήστη να χρησιμοποιήσει είτε τη στατική είτε τη συνδεδεμένη λίστα προκειμένου να εκτελέσει τις ίδιες λειτουργίες πάνω σε μια λίστα.

4. Υλοποιήστε μια κυκλικά συνδεδεμένη λίστα. Η κυκλική λίστα είναι μια απλά συνδεδεμένη λίστα στην οποία το τελευταίο στοιχείο της λίστας δείχνει στο πρώτο στοιχείο της λίστας. Η υλοποίηση θα πρέπει να συμπεριλαμβάνει και δύο δείκτες, έναν που να δείχνει στο πρώτο στοιχείο της λίστας και έναν που να δείχνει στο τελευταίο στοιχείο της λίστας. Προσθέστε τις απαιτούμενες λειτουργίες έτσι ώστε η λίστα να παρέχονται οι ακόλουθες λειτουργίες: εμφάνιση λίστας, εισαγωγή στοιχείου, διαγραφή στοιχείου, εμφάνιση πλήθους στοιχείων, εύρεση στοιχείου. Γράψτε πρόγραμμα που να δοκιμάζει τις λειτουργίες της λίστας.

Αναφορές

[1] C++ Tutorial-exceptions-2017 by K. Hong, http://www.bogotobogo.com/cplusplus/exceptions.php

[2] Geeks for Geeks, Vector in C++ STL, http://www.geeksforgeeks.org/vector-in-cpp-stl/.

[3] Geeks for Geeks, Deque in C++ STL, http://www.geeksforgeeks.org/deque-cpp-stl/.

[4] Geeks for Geeks, Array class in C++ STL http://www.geeksforgeeks.org/array-class-c/.

[5] Geeks for Geeks, List in C++ STL http://www.geeksforgeeks.org/list-cpp-stl/

[6] Geeks for Geeks, Forward List in C++ (Set 1) http://www.geeksforgeeks.org/forward-list-c-set-1-introduction-important-functions/

[7] Geeks for Geeks, Forward List in C++ (Set 2) http://www.geeksforgeeks.org/forward-list-c-set-2-manipulating-functions/