Δομές Δεδομένων και Αλγόριθμοι - Εργαστήριο 4
Γραμμικές λίστες (στατικές λίστες και συνδεδεμένες λίστες), 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 δύο πιθανές υλοποιήσεις γραμμικών λιστών (στατικής λίστας και απλά συνδεδεμένης λίστας) καθώς και οι ενσωματωμένες δυνατότητες της C++ μέσω containers της STL όπως το vector, το list και άλλα. Ο κώδικας όλων των παραδειγμάτων βρίσκεται στο https://github.com/chgogos/ceteiep_dsa.

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.

```
#include <iostream>
#include <stdexcept>

using namespace std;

const int MAX = 50000;
template <class T> struct static_list {
  T elements[MAX];
  int size = 0;
};

// get item at position i
template <class T> T access(static_list<T> &static_list, int i) {
  if (i < 0 || i >= static_list.size)
    throw out_of_range("the index is out of range");
```

```
16    else
17       return static_list.elements[i];
18 }
19
20 // get the position of item x
21 template <class T> int search(static_list<T> &static_list, T x) {
22    for (int i = 0; i < static_list.size; i++)
23       if (static_list.elements[i] == x)
24          return i;
25    return -1;
26 }
27
28 // append item x at the end of the list
29 template <class T> void push_back(static_list<T> &static_list, T x) {
30    if (static_list.size == MAX)
31       throw "full list exception";
32    static_list.elements[static_list.size] = x;
33    static_list.size++;
34 }
35
36 // append item x at position i, shift the rest to the right
37 template <class T> void insert(static_list<T> &static_list, int i, T x) {
38    if (static_list.size == MAX)
39       throw "full list exception";
40    if (i < 0 || i >= static_list.size)
41       throw out_of_range("the index is out of range");
42    for (int k = static_list.size; k > i; k--)
43       static_list.elements[k] = static_list.elements[k - 1];
44    static_list.elements[i] = x;
45    static_list.size++;
46 }
47
48 // delete item at position i, shift the rest to the left
49 template <class T> void delete_item(static_list<T> &static_list, int i) {
50    if (i < 0 || i >= static_list.size)
51       throw out_of_range("the index is out of range");
52    for (int k = i; k < static_list.size; k++)
53       static_list.elements[k] = static_list.elements[k + 1];
54    static_list.size--;
55 }
56
57 template <class T> void print_list(static_list<T> &static_list) {
58    cout << "List: ";
59    for (int i = 0; i < static_list.size; i++)
60       cout << static_list.elements[i] << " ";
61    cout << endl;
62 }
```

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

```
1  #include "static_list.cpp"
2  #include <iostream>
3
4  using namespace std;
5
6  int main(void) {
7     static_list<int> alist;
8     cout << "#1. Add items 10, 20 and 30" << endl;
9     push_back(alist, 10);
10    push_back(alist, 20);
11    push_back(alist, 30);
```

```
12    print_list(alist);
13    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)

```
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
```

Εξαιρέσεις στη C++ 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].

Σχετικά με τις στατικές γραμμικές λίστες ισχύει ότι 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 χρησιμοποιείται 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>
#include <stdexcept>

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;
```

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

```
114   while (current != NULL) {
115     cout << current->data << " ";
116     current = current->next;
117   }
118   cout << endl;
119 }
```

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

```
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    cout << "#1. Add items 10, 20 and 30" << endl;
9    push_back(alist, 10);
10   push_back(alist, 20);
11   push_back(alist, 30);
12   print_list(alist);
13   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. Delete allocated memory " << endl;
28   for (int i = 0; i < alist.size; i++)
29     delete_item(alist, i);
30 }
```

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

```
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. 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.

Οι αναφορές [2] και [3] παρέχουν χρήσιμες πληροφορίες και ασκήσεις σχετικά με τις συνδεδεμένες λίστες και το ρόλο των δεικτών στην υλοποίησή τους.

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, forward_list και bitset.

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 στις αναφορές [4] και [5]. Στο ακόλουθο παράδειγμα παρουσιάζονται 4 διαφορετικοί τρόποι με τους οποίους μπορεί να προσπελαστεί το πρώτο και το τελευταίο στοιχείο του διανύσματος καθώς και η δυνατότητα ελέγχου με τον τελεστή της ισότητας σχετικά με το αν δύο διανύσματα είναι ίσα.

```
#include <iostream>
#include <vector>

using namespace std;
int main() {
   vector<int> v1{10, 20, 30, 40};
   cout << "1. The first element is " << v1.front() << endl;
   cout << "2. The first element is " << v1[0] << endl;
   cout << "3. The first element is " << v1.at(0) << endl;
   cout << "4. The first element is " << *(v1.begin()) << endl;
   cout << "1. The last element is " << v1.back() << endl;
   cout << "2. The last element is " << v1[3] << endl;
   cout << "3. The last element is " << v1.at(3) << endl;
   cout << "4. The last element is " << *(v1.end() - 1) << endl;

   vector<int> v2{10, 20, 30, 40};
   if (v1 == v2)
      cout << "equal vectors" << endl;
}
```

Κώδικας 5: Παράδειγμα με vectors (vector.cpp)

```
1. The first element is 10
2. The first element is 10
3. The first element is 10
4. The first element is 10
1. The last element is 40
2. The last element is 40
3. The last element is 40
4. The last element is 40
equal vectors
```

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 [6]. Στο παράδειγμα που ακολουθεί εισάγονται σε ένα deque εναλλάξ στο αριστερό και στο δεξί άκρο οι άρτιοι και οι περιττοί ακέραιοι αριθμοί στο διάστημα [1,20].

```
#include <deque>
#include <iostream>

```

```
4  using namespace std;
5
6  int main() {
7    deque<int> de;
8    for (int i = 1; i <= 20; i++)
9      if (i % 2 == 0)
10       de.push_front(i);
11     else
12       de.push_back(i);
13
14   for (int x : de)
15     cout << x << " ";
16   cout << endl;
17 }
```

Κώδικας 6: Παράδειγμα με deque (deque.cpp)

```
1  20 18 16 14 12 10 8 6 4 2 1 3 5 7 9 11 13 15 17 19
```

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 [7]. Στο ακόλουθο παράδειγμα δημιουργείται ένα array με 5 πραγματικές τιμές, ταξινομείται και εμφανίζεται.

```
1  #include <algorithm>
2  #include <array>
3  #include <iostream>
4
5  using namespace std;
6
7  int main() {
8    array<double, 5> a{6.5, 2.1, 7.2, 8.1, 1.9};
9    sort(a.begin(), a.end());
10   for (double x : a)
11     cout << x << " ";
12   cout << endl;
13 }
```

Κώδικας 7: Παράδειγμα με array (array.cpp)

```
1  1.9 2.1 6.5 7.2 8.1
```

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 [8]. Στο παράδειγμα που ακολουθεί μια διπλά συνδεδεμένη λίστα διανύεται από δεξιά προς τα αριστερά και από αριστερά προς τα δεξιά στην ίδια επανάληψη.

```
1  #include <iostream>
2  #include <list>
3
4  using namespace std;
5
6  int main() {
7    list<int> alist{10, 20, 30, 40};
```

```
8   list<int>::iterator it = alist.begin();
9   list<int>::reverse_iterator rit = alist.rbegin();
10
11  while (it != alist.end()) {
12    cout << ”Forwards:” << *it << endl;
13    cout << ”Backwards:” << *rit << endl;
14    it++;
15    rit++;
16  }
17 }
```

Κώδικας 8: Παράδειγμα με list (forward_list.cpp)

```
1 Forwards:10
2 Backwards:40
3 Forwards:20
4 Backwards:30
5 Forwards:30
6 Backwards:20
7 Forwards:40
8 Backwards:10
```

2.3.5 Forward Lists

Οι forward lists (λίστες προς τα εμπρός) 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 [9] και [10]. Ακολουθεί ένα παράδειγμα που αντιστρέφει μια απλά συνδεδεμένη λίστα στην οποία έχουν πριν προστεθεί στοιχεία.

```
1  #include <forward_list>
2  #include <iostream>
3
4  using namespace std;
5  int main() {
6    forward_list<int> fl{10, 20, 30, 40, 50};
7    for (int x : fl)
8      cout << x << ” ”;
9    cout << endl;
10   fl.reverse();
11   for (int x : fl)
12     cout << x << ” ”;
13   cout << endl;
14 }
```

Κώδικας 9: Παράδειγμα με forward_list (forward_list.cpp)

```
1 10 20 30 40 50
2 50 40 30 20 10
```

2.3.6 Bitset

Τα bitsets είναι πίνακες με λογικές τιμές τις οποίες αποθηκεύουν με αποδοτικό τρόπο καθώς για κάθε λογική τιμή απαιτείται μόνο 1 bit. Το μέγεθος ενός bitset πρέπει να είναι γνωστό κατά τη μεταγλώττιση. Μια ιδιαιτερότητά του είναι ότι οι δείκτες θέσης που χρησιμοποιούνται για την αναφορά στα στοιχεία του ξεκινούν την αρίθμησή τους με το μηδέν από δεξιά και αυξάνονται προς τα αριστερά. Για παράδειγμα ένα bitset με τιμές 101011 έχει την τιμή 1 στις θέσεις 0,1,3,5 και 0 στις θέσεις 2 και 4. Περισσότερες πληροφορίες για τα bitsets μπορούν να βρεθούν στις αναφορές [11] και [12]. Ακολουθεί ένα παράδειγμα που εμφανίζει χρησιμοποιώντας 5 δυαδικά ψηφία τους ακέραιους αριθμούς από το 0 μέχρι το 7.

```cpp
#include <bitset>
#include <iostream>

using namespace std;

int main() {
  for (int x = 0; x < 8; x++) {
    bitset<5> b(x);
    cout << x << " ==> " << b << " bits set " << b.count() << endl;
  }
}
```

Κώδικας 10: Παράδειγμα με bitset (bitset.cpp)

```
0 ==> 00000 bits set 0
1 ==> 00001 bits set 1
2 ==> 00010 bits set 1
3 ==> 00011 bits set 2
4 ==> 00100 bits set 1
5 ==> 00101 bits set 2
6 ==> 00110 bits set 2
7 ==> 00111 bits set 3
```

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)

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

using namespace std;

int main(int argc, char **argv) {
  linked_list<int> alist;
  int choice, position, value;
  do {
    cout << "1.Show list" << "–";
    cout << "2.Insert item (back)" << "–";
    cout << "3.Insert item (at position)" << "–";
    cout << "4.Delete item (from position)" << "–";
    cout << "5.Delete all items having value" << "–";
    cout << "6.Exit" << endl;
    cout << "Enter choice:";
    cin >> choice;
    if (choice < 1 || choice > 6) {
      cerr << "Choice should be 1 to 6" << endl;
      continue;
    }
    try {
```

```
23          switch (choice) {
24          case 1:
25            print_list(alist);
26            break;
27          case 2:
28            cout << "Enter value:";
29            cin >> value;
30            push_back(alist, value);
31            break;
32          case 3:
33            cout << "Enter position and value:";
34            cin >> position >> value;
35            insert(alist, position, value);
36            break;
37          case 4:
38            cout << "Enter position:";
39            cin >> position;
40            delete_item(alist, position);
41            break;
42          case 5:
43            cout << "Enter value:";
44            cin >> value;
45            int i = 0;
46            while (i < alist.size)
47              if (access(alist, i) == value)
48                delete_item(alist, i);
49              else
50                i++;
51          }
52        } catch (out_of_range oor) {
53          cerr << "Out of range, try again" << endl;
54        }
55      } while (choice != 6);
56  }
```

Κώδικας 11: Έλεγχος συνδεδεμένης λίστας ακεραίων μέσω μενού (lab04_ex1.cpp)

```
1  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
2  Enter choice:2
3  Enter value:10
4  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
5  Enter choice:2
6  Enter value:20
7  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
8  Enter choice:2
9  Enter value:20
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
11 Enter choice:3
12 Enter position and value:1 15
13 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
14 Enter choice:1
15 List: 10 15 20 20
16 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
17 Enter choice:4
18 Enter position:0
19 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
20 Enter choice:1
21 List: 15 20 20
22 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
23 Enter choice:5
24 Enter value:20
25 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
26 Enter choice:1
27 List: 15
28 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
```

```
29 Enter choice:6
```

3.2 Παράδειγμα 2

Έστω μια τράπεζα που διατηρεί για κάθε πελάτη της τον κωδικό του και το υπόλοιπο του λογαριασμού του. Για τις ανάγκες του παραδείγματος θα δημιουργηθούν τυχαίοι πελάτες ως εξής: ο κωδικός κάθε πελάτη θα αποτελείται από 10 σύμβολα που θα επιλέγονται με τυχαίο τρόπο από τα γράμματα της αγγλικής αλφαβήτου και το 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 ο κωδικός τους ξεκινά με το χαρακτήρα Α.
Γ΄ Θα προσθέτει για κάθε πελάτη που ο κωδικός του ξεκινά με το χαρακτήρα G στην αμέσως επόμενη θέση έναν πελάτη με κωδικό το αντίστροφο κωδικό του πελάτη και το ίδιο υπόλοιπο λογαριασμού.
Δ΄ Θα διαγράφει όλους τους πελάτες που ο κωδικός 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.

```
1  #include ”linked_list.cpp”
2  #include <algorithm>
3  #include <chrono>
4  #include <iomanip>
5  #include <iostream>
6  #include <list>
7  #include <random>
8  #include <string>
9
10 using namespace std;
11 using namespace std::chrono;
12
13 mt19937 *mt;
14 uniform_int_distribution<int> uni1(0, 5000), uni2(0, 25);
15
16 struct customer {
17    string code;
18    int balance;
19    bool operator<(customer other) { return code < other.code; }
20 };
21
22 string generate_random_code(int k) {
23    string code{};
24    string letters_en(”ABCDEFGHIJKLMNOPQRSTUVWXYZ”);
25    for (int j = 0; j < k; j++) {
26       char c{letters_en[uni2(*mt)]};
27       code += c;
28    }
29    return code;
30 }
31
32 void generate_data_linked_list(linked_list<customer> &linked_list, int N) {
33    struct node<customer> *current, *next_customer;
34    current = new node<customer>();
35    current−>data.code = generate_random_code(10);
36    current−>data.balance = uni1(*mt);
37    current−>next = NULL;
38    linked_list.head = current;
```

```
39    linked_list.size++;
40    for (int i = 1; i < N; i++) {
41      next_customer = new node<customer>();
42      next_customer->data.code = generate_random_code(10);
43      next_customer->data.balance = uni1(*mt);
44      next_customer->next = NULL;
45      current->next = next_customer;
46      current = next_customer;
47      linked_list.size++;
48    }
49  }
50
51  void print_customers_linked_list(linked_list<customer> &linked_list, int k) {
52    cout << "LIST SIZE=" << linked_list.size << ": ";
53    for (int i = 0; i < k; i++) {
54      customer cu = access(linked_list, i);
55      cout << cu.code << " - " << cu.balance << " ";
56    }
57    cout << endl;
58  }
59
60  void total_balance_linked_list(linked_list<customer> &linked_list, char c) {
61    struct node<customer> *ptr;
62    ptr = linked_list.head;
63    int i = 0;
64    int sum = 0;
65    while (ptr != NULL) {
66      customer cu = ptr->data;
67      if (cu.code.at(0) == c)
68        sum += cu.balance;
69      ptr = ptr->next;
70      i++;
71    }
72    cout << "Total balance for customers having code starting with character "
73         << c << " is " << sum << endl;
74  }
75
76  void add_extra_customers_linked_list(linked_list<customer> &linked_list,
77                                       char c) {
78    struct node<customer> *ptr = linked_list.head;
79    while (ptr != NULL) {
80      customer cu = ptr->data;
81      if (cu.code.at(0) == c) {
82        customer ncu;
83        ncu.code = cu.code;
84        reverse(ncu.code.begin(), ncu.code.end());
85        ncu.balance = cu.balance;
86        struct node<customer> *new_node = new node<customer>();
87        new_node->data = ncu;
88        new_node->next = ptr->next;
89        ptr->next = new_node;
90        linked_list.size++;
91        ptr = new_node->next;
92      } else
93        ptr = ptr->next;
94    }
95  }
96
97  void remove_customers_linked_list(linked_list<customer> &linked_list, char c) {
98    struct node<customer> *ptr1;
99    while (linked_list.size > 0) {
```

```
100     customer cu = linked_list.head->data;
101     if (cu.code.at(0) == c) {
102       ptr1 = linked_list.head;
103       linked_list.head = ptr1->next;
104       delete ptr1;
105       linked_list.size--;
106     } else
107       break;
108   }
109   if (linked_list.size == 0)
110     return;
111   ptr1 = linked_list.head;
112   struct node<customer> *ptr2 = ptr1->next;
113   while (ptr2 != NULL) {
114     customer cu = ptr2->data;
115     if (cu.code.at(0) == c) {
116       ptr1->next = ptr2->next;
117       delete (ptr2);
118       ptr2 = ptr1->next;
119       linked_list.size--;
120     } else {
121       ptr1 = ptr2;
122       ptr2 = ptr2->next;
123     }
124   }
125 }
126
127 int main(int argc, char **argv) {
128   long seed = 1940;
129   mt = new mt19937(seed);
130   cout << "Testing linked list" << endl;
131   struct linked_list<customer> linked_list;
132   string msgs[] = {"A(random customers generation)",
133                    "B(total balance for customers having code starting with A)",
134                    "C(insert new customers)", "D(remove customers)"};
135   for (int i = 0; i < 4; i++) {
136     cout << "############################################################" << endl;
137     auto t1 = high_resolution_clock::now();
138     if (i == 0) {
139       generate_data_linked_list(linked_list, 40000);
140     } else if (i == 1)
141       total_balance_linked_list(linked_list, 'A');
142     else if (i == 2) {
143       add_extra_customers_linked_list(linked_list, 'G');
144     } else if (i == 3) {
145       remove_customers_linked_list(linked_list, 'B');
146     }
147     auto t2 = high_resolution_clock::now();
148     auto duration = duration_cast<microseconds>(t2 - t1).count();
149     print_customers_linked_list(linked_list, 5);
150     cout << msgs[i] << ". Time elapsed: " << duration << " microseconds "
151          << setprecision(3) << duration / 1000000.0 << " seconds" << endl;
152   }
153   delete mt;
154 }
```

Κώδικας 12: Λίστα πελατών για το ίδιο πρόβλημα (lab04_ex2.cpp)

```
1 Testing linked list
2 ############################################################
3 LIST SIZE=40000: GGFSICRZWW -- 2722 UBKZNBPWLH -- 4019 UPIHSBIIBS -- 3896 JRQVGHLTNM -- 395 LUWYTFTNFJ -- 784
4 A(random customers generation). Time elapsed: 39002 microseconds 0.039 seconds
```

```
##########################################################
Total balance for customers having code starting with character A is 3871562
LIST SIZE=40000: GGFSICRZWW − 2722 UBKZNBPWLH − 4019 UPIHSBIIBS − 3896 JRQVGHLTNM − 395 LUWYTFTNFJ − 784
B(total balance for customers having code starting with A). Time elapsed: 1000 microseconds 0.001 seconds
##########################################################
LIST SIZE=41548: GGFSICRZWW − 2722 WWZRCISFGG − 2722 UBKZNBPWLH − 4019 UPIHSBIIBS − 3896 JRQVGHLTNM − 395
C(insert new customers). Time elapsed: 2000 microseconds 0.002 seconds
##########################################################
LIST SIZE=39928: GGFSICRZWW − 2722 WWZRCISFGG − 2722 UBKZNBPWLH − 4019 UPIHSBIIBS − 3896 JRQVGHLTNM − 395
D(remove customers). Time elapsed: 1000 microseconds 0.001 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
std::vector	0.033	0.002	0.543	0.519
std::list	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 παρέχει τις 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] Linked List Basics by N. Parlante, http://cslibrary.stanford.edu/103/.

[3] Linked List Problems by N. Parlante, http://cslibrary.stanford.edu/105/.

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

[5] Codecogs, Vector, a random access dynamic container, http://www.codecogs.com/library/computing.

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

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

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

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

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

[11] Geeks for Geeks, C++ bitset and its application, http://www.geeksforgeeks.org/c-bitset-and-its-application/

[12] Geeks for Geeks, C++ bitset interesting facts, http://www.geeksforgeeks.org/c-bitset-interesting-facts/