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

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. Για το λόγο αυτό στο συγκεκριμένο εργαστήριο θα παρουσιαστούν οι υλοποιήσεις γραμμικών λιστών αλλά και οι ενσωματωμένες δυνατότητες της C++ μέσω της STL.

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];
17 }
18
```

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

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
```

2.2 Συνδεδεμένες γραμμικές λίστες

```
1  #include <iostream>
2
3  using namespace std;
4
5  template <class T> struct node {
6     T data;
7     struct node<T> *next = NULL;
8  };
9
10 template <class T> struct linked_list {
11    struct node<T> *head = NULL;
12    int size = 0;
13 };
14
15 // get node item at position i
16 template <class T>
17 struct node<T> *access_node(linked_list<T> &linked_list, int i) {
18    if (i < 0 || i >= linked_list.size)
19       throw out_of_range("the index is out of range");
20    struct node<T> *current = linked_list.head;
21    for (int k = 0; k < i; k++)
22       current = current->next;
23    return current;
24 }
```

```
25
26 // get node item at position i
27 template <class T>
28 T access(linked_list<T> &linked_list, int i) {
29   struct node<T> *item = access_node(linked_list, i);
30   return item->data;
31 }
32
33 // get the position of item x
34 template <class T> int search(linked_list<T> &linked_list, T x) {
35   struct node<T> *current = linked_list.head;
36   int i = 0;
37   while (current != NULL) {
38     if (current->data == x)
39       return i;
40     i++;
41     current = current->next;
42   }
43   return -1;
44 }
45
46 // append item x at the end of the list
47 template <class T> void push_back(linked_list<T> &l, T x) {
48   struct node<T> *new_node, *current;
49   new_node = new node<T>();
50   new_node->data = x;
51   new_node->next = NULL;
52   current = l.head;
53   if (current == NULL) {
54     l.head = new_node;
55     l.size++;
56   } else {
57     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)

```
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
```

2.3 Γραμμικές λίστες της STL

2.3.1 list

2.3.2 forwardlist

2.3.3 vector

3 Παραδείγματα

3.1 Παράδειγμα 1

Έστω μια υποθετική τράπεζα. Για κάθε πελάτη έστω ότι η τράπεζα διατηρεί σε ένα αρχείο το ονοματεπώνυμο του και το υπόλοιπο του λογαριασμού του. Για τις ανάγκες της άσκησης θα πρέπει να δημιουργηθούν τυχαίοι πελάτες ως εξής: το όνομα κάθε πελάτη να αποτελείται από 10 γράμματα που θα επιλέγονται με τυχαίο τρόπο από τα γράμματα της αγγλικής αλφαβήτου και το δε υπόλοιπο κάθε πελάτη να είναι ένας τυχαίος αριθμός από το 0 μέχρι το 5.000. Θα παρουσιαστούν τέσσερις εκδόσεις του ίδιου προγράμματος. Η μεν πρώτη θα υλοποιείται με στατική λίστα, η δεύτερη με συνδεδεμένη λίστα η τρίτη με τη στατική γραμμική λίστα της C++, std::vector και η τέταρτη με τη συνδεδεμένη λίστα της C++, std::list. Και στις τέσσερις περιπτώσεις το πρόγραμμα θα πραγματοποιεί τις ακόλουθες λειτουργίες:

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

```
1 #include "linked_list.cpp"
2 #include "static_list.cpp"
3 #include <algorithm>
4 #include <chrono>
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);
15 uniform_int_distribution<int> uni2(0, 25);
16
17 struct customer {
18   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 // #### START STATIC LIST
34 void generate_data_static_list(static_list<customer> &static_list, int N) {
35   for (int i = 0; i < N; i++) {
36     customer c;
37     c.name = generate_random_name(10);
38     c.balance = uni1(*mt);
39     push_back(static_list, c);
40   }
41 }
42
43 void print_customers_static_list(static_list<customer> &static_list, int k) {
44   for (int i = 0; i < k; i++) {
45     customer cu = access(static_list, i);
46     cout << cu.name << " – " << cu.balance << endl;
47   }
48   cout << "SIZE " << static_list.size << endl;
49 }
50
51 void total_balance_static_list(static_list<customer> &static_list, char c) {
52   int sum = 0;
53   for (int i = 0; i < static_list.size; i++) {
54     customer cu = access(static_list, i);
55     if (cu.name.at(0) == c)
56       sum += cu.balance;
57   }
58   cout << "Total balance for customers having name starting with character "
59        << c << " is " << sum << endl;
60 }
61
62 void add_extra_customers_static_list(static_list<customer> &static_list,
63                                      char c) {
64   int i = 0;
65   while (i < static_list.size) {
66     customer cu = access(static_list, i);
67     if (cu.name.at(0) == c) {
68       customer ncu;
69       ncu.name = cu.name;
70       reverse(ncu.name.begin(), ncu.name.end());
```

```
71          ncu.balance = cu.balance;
72          insert(static_list, i + 1, ncu);
73          i++;
74        }
75        i++;
76      }
77    }
78
79    void remove_customers_static_list(static_list<customer> &static_list, char c) {
80      int i = 0;
81      while (i < static_list.size) {
82        customer cu = access(static_list, i);
83        if (cu.name.at(0) == c)
84          delete_item(static_list, i);
85        else
86          i++;
87      }
88    }
89
90    void test_static_list() {
91      cout << "Testing static list" << endl;
92      cout << "############################################################" << endl;
93      auto t1 = high_resolution_clock::now();
94      struct static_list<customer> static_list;
95      generate_data_static_list(static_list, 40000);
96      auto t2 = high_resolution_clock::now();
97      print_customers_static_list(static_list, 5);
98      auto duration = duration_cast<microseconds>(t2 - t1).count();
99      cout << "Time elapsed: " << duration << " microseconds" << endl;
100
101     t1 = high_resolution_clock::now();
102     total_balance_static_list(static_list, 'A');
103     t2 = high_resolution_clock::now();
104     duration = duration_cast<microseconds>(t2 - t1).count();
105     cout << "Time elapsed: " << duration << " microseconds" << endl;
106
107     t1 = high_resolution_clock::now();
108     add_extra_customers_static_list(static_list, 'A');
109     t2 = high_resolution_clock::now();
110     print_customers_static_list(static_list, 5);
111     duration = duration_cast<microseconds>(t2 - t1).count();
112     cout << "Time elapsed: " << duration << " microseconds" << endl;
113
114     t1 = high_resolution_clock::now();
115     remove_customers_static_list(static_list, 'B');
116     t2 = high_resolution_clock::now();
117     print_customers_static_list(static_list, 5);
118     duration = duration_cast<microseconds>(t2 - t1).count();
119     cout << "Time elapsed: " << duration << " microseconds" << endl;
120     cout << "############################################################" << endl;
121   }
122   // #### END STATIC LIST
123
124   // #### START LINKED LIST
125   void generate_data_linked_list(linked_list<customer> &linked_list, int N) {
126     for (int i = 0; i < N; i++) {
127       customer c;
128       c.name = generate_random_name(10);
129       c.balance = uni1(*mt);
130       push_back(linked_list, c);
131     }
```

```
132 }
133
134 void print_customers_linked_list(linked_list<customer> &linked_list, int k) {
135   for (int i = 0; i < k; i++) {
136     customer cu = access(linked_list, i);
137     cout << cu.name << " – " << cu.balance << endl;
138   }
139   cout << "SIZE " << linked_list.size << endl;
140 }
141
142 void total_balance_linked_list(linked_list<customer> &linked_list, char c) {
143   struct node<customer> *ptr;
144   ptr = linked_list.head;
145   int i = 0;
146   int sum = 0;
147   while (ptr != NULL) {
148     customer cu = ptr->data;
149     if (cu.name.at(0) == c)
150       sum += cu.balance;
151     ptr = ptr->next;
152     i++;
153   }
154   cout << "Total balance for customers having name starting with character "
155        << c << " is " << sum << endl;
156 }
157
158 void add_extra_customers_linked_list(linked_list<customer> &linked_list,
159                                      char c) {
160   struct node<customer> *ptr = linked_list.head;
161   while (ptr != NULL) {
162     customer cu = ptr->data;
163     if (cu.name.at(0) == c) {
164       customer ncu;
165       ncu.name = cu.name;
166       reverse(ncu.name.begin(), ncu.name.end());
167       ncu.balance = cu.balance;
168       struct node<customer> *new_node = new node<customer>();
169       new_node->data = ncu;
170       new_node->next = ptr->next;
171       ptr->next = new_node;
172       linked_list.size++;
173       ptr = new_node->next;
174     } else
175       ptr = ptr->next;
176   }
177 }
178
179 void remove_customers_linked_list(linked_list<customer> &linked_list, char c) {
180   int i = 0;
181   while (i < linked_list.size) {
182     struct customer cu = access(linked_list, i);
183     if (cu.name.at(0) == c)
184       delete_item(linked_list, i);
185     else
186       i++;
187   }
188 }
189
190 void test_linked_list() {
191   cout << "Testing linked list" << endl;
192   cout << "############################################################" << endl;
```

```
193   struct linked_list<customer> linked_list;
194   auto t1 = high_resolution_clock::now();
195   generate_data_linked_list(linked_list, 40000);
196   auto t2 = high_resolution_clock::now();
197   print_customers_linked_list(linked_list, 5);
198   auto duration = duration_cast<microseconds>(t2 − t1).count();
199   cout << ”Time elapsed: ” << duration << ” microseconds” << endl;
200
201   t1 = high_resolution_clock::now();
202   total_balance_linked_list(linked_list, ’A’);
203   t2 = high_resolution_clock::now();
204   duration = duration_cast<microseconds>(t2 − t1).count();
205   cout << ”Time elapsed: ” << duration << ” microseconds” << endl;
206
207   t1 = high_resolution_clock::now();
208   add_extra_customers_linked_list(linked_list, ’A’);
209   t2 = high_resolution_clock::now();
210   print_customers_linked_list(linked_list, 5);
211   duration = duration_cast<microseconds>(t2 − t1).count();
212   cout << ”Time elapsed: ” << duration << ” microseconds” << endl;
213
214   t1 = high_resolution_clock::now();
215   remove_customers_linked_list(linked_list, ’B’);
216   // remove_customers_linked_list_alt(linked_list, ’B’);
217   t2 = high_resolution_clock::now();
218   print_customers_linked_list(linked_list, 5);
219   duration = duration_cast<microseconds>(t2 − t1).count();
220   cout << ”Time elapsed: ” << duration << ” microseconds” << endl;
221   cout << ”############################################################” << endl;
222 }
223 // #### END LINKED LIST
224
225 // #### START VECTOR
226 void generate_data_stl_vector(vector<customer> &stl_vector, int N) {
227   for (int i = 0; i < N; i++) {
228     customer c;
229     c.name = generate_random_name(10);
230     c.balance = uni1(*mt);
231     stl_vector.push_back(c);
232   }
233 }
234
235 void print_customers_stl_vector(vector<customer> &stl_vector, int k) {
236   int c = 0;
237   for (customer cu : stl_vector) {
238     cout << cu.name << ” – ” << cu.balance << endl;
239     if (++c == k)
240       break;
241   }
242   cout << ”SIZE ” << stl_vector.size() << endl;
243 }
244
245 void total_balance_stl_vector(vector<customer> &stl_vector, char c) {
246   int sum = 0;
247   for (customer cu : stl_vector)
248     if (cu.name.at(0) == c)
249       sum += cu.balance;
250   cout << ”Total balance for customers having name starting with character ”
251        << c << ” is ” << sum << endl;
252 }
253
```

```
254 void add_extra_customers_stl_vector(vector<customer> &stl_vector, char c) {
255   auto i = stl_vector.begin();
256   while (i != stl_vector.end()) {
257     customer cu = *i;
258     if (cu.name.at(0) == c) {
259       customer ncu;
260       ncu.name = cu.name;
261       reverse(ncu.name.begin(), ncu.name.end());
262       ncu.balance = cu.balance;
263       i++;
264       stl_vector.insert(i, ncu);
265     }
266     i++;
267   }
268 }
269
270 void remove_customers_stl_vector(vector<customer> &stl_vector, char c) {
271   auto i = stl_vector.begin();
272   while (i != stl_vector.end()) {
273     customer cu = *i;
274     if (cu.name.at(0) == c) {
275       i = stl_vector.erase(i);
276     } else
277       i++;
278   }
279 }
280
281 void test_stl_vector() {
282   cout << "Testing stl vector" << endl;
283   cout << "############################################################" << endl;
284   auto t1 = high_resolution_clock::now();
285   vector<customer> stl_vector;
286   generate_data_stl_vector(stl_vector, 40000);
287   auto t2 = high_resolution_clock::now();
288   print_customers_stl_vector(stl_vector, 5);
289   auto duration = duration_cast<microseconds>(t2 - t1).count();
290   cout << "Time elapsed: " << duration << " microseconds" << endl;
291
292   t1 = high_resolution_clock::now();
293   total_balance_stl_vector(stl_vector, 'A');
294   t2 = high_resolution_clock::now();
295   duration = duration_cast<microseconds>(t2 - t1).count();
296   cout << "Time elapsed: " << duration << " microseconds" << endl;
297
298   t1 = high_resolution_clock::now();
299   add_extra_customers_stl_vector(stl_vector, 'A');
300   t2 = high_resolution_clock::now();
301   print_customers_stl_vector(stl_vector, 5);
302   duration = duration_cast<microseconds>(t2 - t1).count();
303   cout << "Time elapsed: " << duration << " microseconds" << endl;
304
305   t1 = high_resolution_clock::now();
306   remove_customers_stl_vector(stl_vector, 'B');
307   t2 = high_resolution_clock::now();
308   print_customers_stl_vector(stl_vector, 5);
309   duration = duration_cast<microseconds>(t2 - t1).count();
310   cout << "Time elapsed: " << duration << " microseconds" << endl;
311   cout << "############################################################" << endl;
312 }
313 // #### END VECTOR
314
```

```
315 // #### START LIST
316 void generate_data_stl_list(list<customer> &stl_list, int N) {
317   for (int i = 0; i < N; i++) {
318     customer c;
319     c.name = generate_random_name(10);
320     c.balance = uni1(*mt);
321     stl_list.push_back(c);
322   }
323 }
324
325 void print_customers_stl_list(list<customer> &stl_list, int k) {
326   int c = 0;
327   for (customer cu : stl_list) {
328     cout << cu.name << " – " << cu.balance << endl;
329     if (++c == k)
330       break;
331   }
332   cout << "SIZE " << stl_list.size() << endl;
333 }
334
335 void total_balance_stl_list(list<customer> &stl_list, char c) {
336   int sum = 0;
337   for (customer cu : stl_list)
338     if (cu.name.at(0) == c)
339       sum += cu.balance;
340   cout << "Total balance for customers having name starting with character "
341        << c << " is " << sum << endl;
342 }
343
344 void add_extra_customers_stl_list(list<customer> &stl_list, char c) {
345   auto i = stl_list.begin();
346   while (i != stl_list.end()) {
347     customer cu = *i;
348     if (cu.name.at(0) == c) {
349       customer ncu;
350       ncu.name = cu.name;
351       reverse(ncu.name.begin(), ncu.name.end());
352       ncu.balance = cu.balance;
353       i++;
354       stl_list.insert(i, ncu);
355     } else
356       i++;
357   }
358 }
359
360 void remove_customers_stl_list(list<customer> &stl_list, char c) {
361   auto i = stl_list.begin();
362   while (i != stl_list.end()) {
363     customer cu = *i;
364     if (cu.name.at(0) == c) {
365       i = stl_list.erase(i);
366     } else
367       i++;
368   }
369 }
370
371 void test_stl_list() {
372   cout << "Testing stl list" << endl;
373   cout << "############################################################" << endl;
374   auto t1 = high_resolution_clock::now();
375   list<customer> stl_list;
```

```
376   generate_data_stl_list(stl_list, 40000);
377   auto t2 = high_resolution_clock::now();
378   print_customers_stl_list(stl_list, 5);
379   auto duration = duration_cast<microseconds>(t2 − t1).count();
380   cout << "Time elapsed: " << duration << " microseconds" << endl;
381
382   t1 = high_resolution_clock::now();
383   total_balance_stl_list(stl_list, 'A');
384   t2 = high_resolution_clock::now();
385   duration = duration_cast<microseconds>(t2 − t1).count();
386   cout << "Time elapsed: " << duration << " microseconds" << endl;
387
388   t1 = high_resolution_clock::now();
389   add_extra_customers_stl_list(stl_list, 'A');
390   t2 = high_resolution_clock::now();
391   print_customers_stl_list(stl_list, 5);
392   duration = duration_cast<microseconds>(t2 − t1).count();
393   cout << "Time elapsed: " << duration << " microseconds" << endl;
394
395   t1 = high_resolution_clock::now();
396   remove_customers_stl_list(stl_list, 'B');
397   t2 = high_resolution_clock::now();
398   print_customers_stl_list(stl_list, 5);
399   duration = duration_cast<microseconds>(t2 − t1).count();
400   cout << "Time elapsed: " << duration << " microseconds" << endl;
401   cout << "############################################################" << endl;
402 }
403 // #### END LIST
404
405 int main(int argc, char **argv) {
406   mt = new mt19937(1940);
407   test_static_list();
408   delete mt;
409   mt = new mt19937(1940);
410   test_linked_list();
411   delete mt;
412   mt = new mt19937(1940);
413   test_stl_vector();
414   delete mt;
415   mt = new mt19937(1940);
416   test_stl_list();
417   delete mt;
418 }
```

Κώδικας 5: Παράδειγμα με συνδεδεμένη γραμμική λίστα (lab04_ex1.cpp)

```
1  Testing static list
2  ############################################################
3  GGFSICRZWW − 2722
4  UBKZNBPWLH − 4019
5  UPIHSBIIBS − 3896
6  JRQVGHLTNM − 395
7  LUWYTFTNFJ − 784
8  SIZE 40000
9  Time elapsed: 36002 microseconds
10 Total balance for customers having name starting with character A is 3871562
11 Time elapsed: 1000 microseconds
12 GGFSICRZWW − 2722
13 UBKZNBPWLH − 4019
14 UPIHSBIIBS − 3896
15 JRQVGHLTNM − 395
16 LUWYTFTNFJ − 784
17 SIZE 41552
18 Time elapsed: 657037 microseconds
```

```
GGFSICRZWW – 2722
UBKZNBPWLH – 4019
UPIHSBIIBS – 3896
JRQVGHLTNM – 395
LUWYTFTNFJ – 784
SIZE 39921
Time elapsed: 680038 microseconds
########################################################
Testing linked list
########################################################
GGFSICRZWW – 2722
UBKZNBPWLH – 4019
UPIHSBIIBS – 3896
JRQVGHLTNM – 395
LUWYTFTNFJ – 784
SIZE 40000
Time elapsed: 3941225 microseconds
Total balance for customers having name starting with character A is 3871562
Time elapsed: 1000 microseconds
GGFSICRZWW – 2722
UBKZNBPWLH – 4019
UPIHSBIIBS – 3896
JRQVGHLTNM – 395
LUWYTFTNFJ – 784
SIZE 41552
Time elapsed: 1000 microseconds
GGFSICRZWW – 2722
UBKZNBPWLH – 4019
UPIHSBIIBS – 3896
JRQVGHLTNM – 395
LUWYTFTNFJ – 784
SIZE 39921
Time elapsed: 4398251 microseconds
########################################################
Testing stl vector
########################################################
GGFSICRZWW – 2722
UBKZNBPWLH – 4019
UPIHSBIIBS – 3896
JRQVGHLTNM – 395
LUWYTFTNFJ – 784
SIZE 40000
Time elapsed: 32001 microseconds
Total balance for customers having name starting with character A is 3871562
Time elapsed: 1000 microseconds
GGFSICRZWW – 2722
UBKZNBPWLH – 4019
UPIHSBIIBS – 3896
JRQVGHLTNM – 395
LUWYTFTNFJ – 784
SIZE 41552
Time elapsed: 561032 microseconds
GGFSICRZWW – 2722
UBKZNBPWLH – 4019
UPIHSBIIBS – 3896
JRQVGHLTNM – 395
LUWYTFTNFJ – 784
SIZE 39921
Time elapsed: 529030 microseconds
########################################################
Testing stl list
########################################################
GGFSICRZWW – 2722
UBKZNBPWLH – 4019
UPIHSBIIBS – 3896
JRQVGHLTNM – 395
LUWYTFTNFJ – 784
SIZE 40000
Time elapsed: 33001 microseconds
Total balance for customers having name starting with character A is 3871562
```

```
Time elapsed: 1000 microseconds
GGFSICRZWW -- 2722
UBKZNBPWLH -- 4019
UPIHSBIIBS -- 3896
JRQVGHLTNM -- 395
LUWYTFTNFJ -- 784
SIZE 41552
Time elapsed: 2000 microseconds
GGFSICRZWW -- 2722
UBKZNBPWLH -- 4019
UPIHSBIIBS -- 3896
JRQVGHLTNM -- 395
LUWYTFTNFJ -- 784
SIZE 39921
Time elapsed: 2000 microseconds
##########################################################
```

3.2 Παράδειγμα 2

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

4 Ασκήσεις

1. Έστω η συνδεδεμένη λίστα που παρουσιάστηκε στον κώδικα XX. Προσθέστε μια συνάρτηση που μια λίστα ακεραίων στην οποία τα στοιχεία της είναι ταξινομημένα από το μικρότερο στο μεγαλύτερο να προσθέτει ένα ακόμα στοιχείο στην κατάλληλη θέση έτσι ώστε η λίστα να παραμένει ταξινομημένη.

2. Υλοποιήστε τον κώδικα της στατικής και της συνδεδεμένης λίστας με κλάσεις. Γράψτε ένα πρόγραμμα που ανάλογα με την επιθυμία του χρήστη θα χρησιμοποιεί είτε τη στατική είτε τη συνδεδεμένη λίστα προκειμένου να εκτελέσει τις ίδιες λειτουργίες πάνω σε μια λίστα.

3. 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.

4. Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Αναφορές

[1] Stable Sorting, https://hackernoon.com/stable-sorting-677453884792