Δομές Δεδομένων και Αλγόριθμοι
ΜΕΡΟΣ Β’
Εργαστήριο (C++)
Τ.Ε.Ι. Ηπείρου - Τμήμα Μηχανικών Πληροφορικής Τ.Ε.

Χρήστος Γκόγκος

Άρτα - 2017

Εργαστήριο 4

Γραμμικές λίστες, λίστες της STL

4.1 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.

4.2 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.

4.2.1 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  #include <stdexcept>
3
4  using namespace std;
5
6  const int MAX = 50000;
7  template <class T> struct static_list {
8    T elements[MAX];
9    int size = 0;
10 };
11
12 // get item at position i
13 template <class T> T access(static_list<T> &static_list, int i) {
14   if (i < 0 || i >= static_list.size)
```

```
15      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 4.1: 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 4.2: 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 4.2) 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.

4.2.2 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.

```
1  #include <iostream>
2  #include <stdexcept>
3
4  using namespace std;
5
6  template <class T> struct node {
7    T data;
8    struct node<T> *next = NULL;
9  };
10
11 template <class T> struct linked_list {
12   struct node<T> *head = NULL;
13   int size = 0;
14 };
15
16 // get node item at position i
17 template <class T>
18 struct node<T> *access_node(linked_list<T> &linked_list, int i) {
19   if (i < 0 || i >= linked_list.size)
20     throw out_of_range("the index is out of range");
21   struct node<T> *current = linked_list.head;
22   for (int k = 0; k < i; k++)
23     current = current->next;
24   return current;
25 }
26
27 // get node item at position i
28 template <class T>
29 T access(linked_list<T> &linked_list, int i) {
30   struct node<T> *item = access_node(linked_list, i);
31   return item->data;
32 }
33
34 // get the position of item x
35 template <class T> int search(linked_list<T> &linked_list, T x) {
36   struct node<T> *current = linked_list.head;
37   int i = 0;
38   while (current != NULL) {
39     if (current->data == x)
40       return i;
41     i++;
42     current = current->next;
43   }
44   return -1;
45 }
46
47 // append item x at the end of the list
48 template <class T> void push_back(linked_list<T> &l, T x) {
49   struct node<T> *new_node, *current;
50   new_node = new node<T>();
```

```
51    new_node->data = x;
52    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 4.3: 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 4.4: 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.

4.2.3 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.

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 4.5: 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
```

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>

using namespace std;

int main() {
   deque<int> de;
   for (int i = 1; i <= 20; i++)
      if (i % 2 == 0)
         de.push_front(i);
      else
         de.push_back(i);

   for (int x : de)
      cout << x << " ";
   cout << endl;
}
```

Κώδικας 4.6: Παράδειγμα με deque (deque.cpp)

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

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.

```
#include <algorithm>
#include <array>
#include <iostream>

using namespace std;

int main() {
   array<double, 5> a{6.5, 2.1, 7.2, 8.1, 1.9};
   sort(a.begin(), a.end());
   for (double x : a)
      cout << x << " ";
   cout << endl;
}
```

Κώδικας 4.7: Παράδειγμα με array (array.cpp)

```
1.9 2.1 6.5 7.2 8.1
```

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.

```
#include <iostream>
#include <list>

```

```
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 4.8: 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
```

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 4.9: Παράδειγμα με forward_list (forward_list.cpp)

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

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.

```
#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 4.10: 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
```

4.3 Παραδείγματα

4.3.1 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)

```
#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"
           << "\n";
      cout << "2.Insert item (back)"
           << "\n";
      cout << "3.Insert item (at position)"
           << "\n";
      cout << "4.Delete item (from position)"
           << "\n";
```

```
18      cout << "5.Delete all items having value"
19           << "-";
20      cout << "6.Exit" << endl;
21      cout << "Enter choice:";
22      cin >> choice;
23      if (choice < 1 || choice > 6) {
24        cerr << "Choice should be 1 to 6" << endl;
25        continue;
26      }
27      try {
28        switch (choice) {
29        case 1:
30          print_list(alist);
31          break;
32        case 2:
33          cout << "Enter value:";
34          cin >> value;
35          push_back(alist, value);
36          break;
37        case 3:
38          cout << "Enter position and value:";
39          cin >> position >> value;
40          insert(alist, position, value);
41          break;
42        case 4:
43          cout << "Enter position:";
44          cin >> position;
45          delete_item(alist, position);
46          break;
47        case 5:
48          cout << "Enter value:";
49          cin >> value;
50          int i = 0;
51          while (i < alist.size)
52            if (access(alist, i) == value)
53              delete_item(alist, i);
54            else
55              i++;
56        }
57      } catch (out_of_range oor) {
58        cerr << "Out of range, try again" << endl;
59      }
60    } while (choice != 6);
61  }
```

Κώδικας 4.11: 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
```

```
Enter choice:4
Enter position:0
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: 15 20 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:5
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:1
List: 15
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
```

4.3.2 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 4.3 καθώς και άλλων συναρτήσεων που επιτρέπουν την αποδοτικότερη υλοποίηση των παραπάνω ερωτημάτων.

```
#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), uni2(0, 25);

struct customer {
  string code;
  int balance;
  bool operator<(customer other) { return code < other.code; }
};

string generate_random_code(int k) {
  string code{};
  string letters_en(”ABCDEFGHIJKLMNOPQRSTUVWXYZ”);
  for (int j = 0; j < k; j++) {
    char c{letters_en[uni2(*mt)]};
    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 4.12: Λίστα πελατών για το ίδιο πρόβλημα (lab04_ex2.cpp)

```
Testing linked list
#########################################################
LIST SIZE=40000: GGFSICRZWW – 2722 UBKZNBPWLH – 4019 UPIHSBIIBS – 3896 JRQVGHLTNM – 395 LUWYTFTNFJ – 784
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 4.3 χρησιμοποιηθεί η στατική λίστα του κώδικα 4.1 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 4.1. 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 4.1: Χρόνοι εκτέλεσης σε δευτερόλεπτα των ερωτημάτων του παραδείγματος 2 ανάλογα με τον τρόπο υλοποίησης της λίστας

4.4 Ασκήσεις

1. Έστω η συνδεδεμένη λίστα που παρουσιάστηκε στον κώδικα 4.3. 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 4.3. Προσθέστε μια συνάρτηση που να αντιστρέφει τη λίστα.

3. Υλοποιήστε τη στατική λίστα (κώδικας 4.1) και τη συνδεδεμένη λίστα (κώδικας 4.3) 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.

Βιβλιογραφία

[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/

Εργαστήριο 5

Στοίβες και ουρές, οι δομές στοίβα και ουρά στην STL

5.1 Εισαγωγή

Οι στοίβες και οι ουρές αποτελούν απλές δομές δεδομένων που είναι ιδιαίτερα χρήσιμες στην επίλυση αλγοριθμικών προβλημάτων. Η στοίβα είναι μια λίστα στοιχείων στην οποία τα νέα στοιχεία τοποθετούνται στην κορυφή και όταν πρόκειται να αφαιρεθεί ένα στοιχείο αυτό πάλι συμβαίνει από την κορυφή των στοιχείων της στοίβας. Από την άλλη μεριά, η ουρά είναι επίσης μια λίστα στοιχείων στην οποία όμως οι εισαγωγές γίνονται στο πίσω άκρο της ουράς ενώ οι εξαγωγές πραγματοποιούνται από το εμπρός άκρο της ουράς. Στο εργαστήριο αυτό θα παρουσιαστούν υλοποιήσεις της στοίβας και της ουράς. Επιπλέον, θα παρουσιαστούν οι δομές της STL std::stack και std::queue. Ο κώδικας όλων των παραδειγμάτων βρίσκεται στο https://github.com/chgogos/ceteiep_dsa.

5.2 Στοίβα

Η στοίβα (stack) είναι μια ειδική περίπτωση γραμμικής λίστας στην οποία οι εισαγωγές και οι διαγραφές επιτρέπονται μόνο από το ένα άκρο. Συνήθως αυτό το άκρο λέγεται κορυφή (top). Πρόκειται για μια δομή στην οποία οι εισαγωγές και οι εξαγωγές γίνονται σύμφωνα με τη μέθοδο τελευταίο μέσα πρώτο έξω (LIFO=Last In First Out).

Στον κώδικα 5.1 παρουσιάζεται μια υλοποίηση στοίβας που χρησιμοποιεί για την αποθήκευση των στοιχείων της έναν πίνακα. Εναλλακτικά, στη θέση του πίνακα μπορεί να χρησιμοποιηθεί συνδεδεμένη λίστα. Μια υλοποίηση στη γλώσσα C μπορεί να βρεθεί στην αναφορά [2], ενώ στην εργασία [1] παρουσιάζονται 16(!) διαφορετικοί τρόποι υλοποίησης της στοίβας στην C++.

Στο παράδειγμα που ακολουθεί ωθούνται σε μια στοίβα τα γράμματα της αγγλικής αλφαβήτου (A-Z) και στη συνέχεια απωθούνται ένα προς ένα και μέχρι η στοίβα να αδειάσει.

```
1  #include <iostream>
2
3  using namespace std;
4
5  template <typename T> class my_stack {
6  private:
7    T *data;
8    int top, capacity;
9
10 public:
11   // constructor
12   my_stack(int c) {
```

```
13      top = -1;
14      capacity = c;
15      data = new T[capacity];
16    }
17
18    // destructor
19    ~my_stack() { delete[] data; }
20
21    bool empty() { return (top == -1); }
22
23    void push(T elem) {
24      if (top == (capacity - 1))
25        throw "The stack is full";
26      else {
27        top++;
28        data[top] = elem;
29      }
30    }
31
32    T pop() {
33      if (top == -1)
34        throw "the stack is empty";
35      top--;
36      return data[top + 1];
37    }
38
39    void print() {
40      for (int i = 0; i <= top; i++)
41        cout << data[i] << " ";
42      cout << endl;
43    }
44  };
45
46  int main() {
47    cout << "Custom stack implementation" << endl;
48    my_stack<char> astack(100);
49    for (char c = 65; c < 65 + 26; c++)
50      astack.push(c);
51    astack.print();
52    while (!astack.empty())
53      cout << astack.pop() << " ";
54    cout << endl;
55  }
```

Κώδικας 5.1: Υλοποίηση στοίβας (stack_oo.cpp)

```
1  Custom stack impementation
2  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
3  Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

5.3 Ουρά

Η ουρά (queue) είναι μια ειδική περίπτωση γραμμικής λίστας στην οποία επιτρέπονται εισαγωγές στο πίσω άκρο της και εξαγωγές από το εμπρός άκρο της μόνο. Τα δύο αυτά άκρα συνήθως αναφέρονται ως πίσω (rear) και εμπρός (front) αντίστοιχα. Η ουρά είναι μια δομή στην οποία οι εισαγωγές και οι εξαγωγές γίνονται σύμφωνα με τη μέθοδο πρώτο μέσα πρώτο έξω (FIFO=First In First Out).

Στη συνέχεια παρουσιάζεται μια υλοποίηση ουράς στην οποία τα δεδομένα της τοποθετούνται σε έναν πίνακα (εναλλακτικά θα μπορούσε να είχε χρησιμοποιηθεί μια άλλη δομή όπως για παράδειγμα η συνδεδεμένη

λίστα). Ο πίνακας λειτουργεί κυκλικά, δηλαδή όταν συμπληρωθεί και εφόσον υπάρχουν διαθέσιμες κενές θέσεις στην αρχή του πίνακα, τα νέα στοιχεία που πρόκειται να εισαχθούν στην ουρά τοποθετούνται στις πρώτες διαθέσιμες, ξεκινώντας από την αρχή του πίνακα, θέσεις.

```
#include <iostream>

using namespace std;

template <typename T> class my_queue {
private:
  T *data;
  int front, rear, capacity, size;

public:
  // constructor
  my_queue(int c) {
    front = 0;
    rear = -1;
    size = 0;
    capacity = c;
    data = new T[capacity];
  }

  // destructor
  ~my_queue() { delete[] data; }

  bool empty() { return (size == 0); }

  void enqueue(T elem) {
    if (size == capacity)
      throw "The queue is full";
    else {
      rear++;
      rear %= capacity;
      data[rear] = elem;
      size++;
    }
  }

  T dequeue() {
    if (size == 0)
      throw "the queue is empty";
    T x = data[front];
    front++;
    front %= capacity;
    size--;
    return x;
  }


  void print(bool internal = true) {
    for (int i = front; i < front + size; i++)
      cout << data[i % capacity] << " ";
    cout << endl;
    if (internal){
      for (int i = 0; i < capacity; i++)
        if (front <= rear && i >= front && i <= rear)
          cout << "[" << i << "]->" << data[i] << " ";
        else if (front >= rear && (i >= front || i <= rear))
          cout << "[" << i << "]->" << data[i] << " ";
        else
```

```
58              cout << "[" << i << "]->X ";
59      }
60      cout << " (front:" << front << " rear:" << rear << ")" << endl;
61    }
62 };
63
64 int main() {
65    cout << "Custom queue implementation" << endl;
66    my_queue<int> aqueue(10);
67    cout << "1. Enqueue 10 items" << endl;
68    for (int i = 51; i <= 60; i++)
69       aqueue.enqueue(i);
70    aqueue.print();
71    cout << "2. Dequeue 5 items" << endl;
72    for (int i = 0; i < 5; i++)
73       aqueue.dequeue();
74    aqueue.print();
75    cout << "3. Enqueue 3 items" << endl;
76    for (int i = 61; i <= 63; i++)
77       aqueue.enqueue(i);
78    aqueue.print();
79 }
```

Κώδικας 5.2: Υλοποίηση ουράς (queue_oo.cpp)

```
1  Custom queue implementation
2  1. Enqueue 10 items
3  51 52 53 54 55 56 57 58 59 60
4  [0]->51 [1]->52 [2]->53 [3]->54 [4]->55 [5]->56 [6]->57 [7]->58 [8]->59 [9]->60 (front:0 rear:9)
5  2. Dequeue 5 items
6  56 57 58 59 60
7  [0]->X [1]->X [2]->X [3]->X [4]->X [5]->56 [6]->57 [7]->58 [8]->59 [9]->60 (front:5 rear:9)
8  3. Enqueue 3 items
9  56 57 58 59 60 61 62 63
10 [0]->61 [1]->62 [2]->63 [3]->X [4]->X [5]->56 [6]->57 [7]->58 [8]->59 [9]->60 (front:5 rear:2)
```

5.4 Οι δομές στοίβα και ουρά στην STL

Οι δομές std::stack και std::queue έχουν υλοποιηθεί στην STL ως container adaptors δηλαδή κλάσεις που χρησιμοποιούν εσωτερικά ένα άλλο container και παρέχουν ένα συγκεκριμένο σύνολο από λειτουργίες που επιτρέπουν την προσπέλαση και την τροποποίηση των στοιχείων τους.

5.4.1 std::stack

Το προκαθορισμένο εσωτερικό container που χρησιμοποιεί η std::stack είναι το std::deque. Ωστόσο, μπορούν να χρησιμοποιηθούν και τα std::vector και std::list καθώς και τα τρία αυτά containers παρέχουν τις λειτουργίες empty(), size(), push_back(), pop_back() και back() που απαιτούνται για να υλοποιηθεί το stack interface [3]. Τυπικές λειτουργίες που παρέχει η std::stack είναι οι ακόλουθες:

- empty(), ελέγχει αν η στοίβα είναι άδεια.
- size(), επιστρέφει το μέγεθος της στοίβας.
- top(), προσπελαύνει το στοιχείο που βρίσκεται στη κορυφή της στοίβας (χωρίς να το αφαιρεί).
- push(), ωθεί ένα στοιχείο στη κορυφή της στοίβας
- pop(), αφαιρεί το στοιχείο που βρίσκεται στη κορυφή της στοίβας.

Ένα παράδειγμα χρήσης της std::stack παρουσιάζεται στη συνέχεια.

```
#include <deque>
#include <iostream>
#include <list>
#include <stack>
#include <vector>

using namespace std;
int main(void) {
  cout << "std::stack example" << endl;
  stack<char> items; // adaptor over a deque container
  // stack<char, deque<char>> items; // adaptor over a deque container
  // stack<char, vector<char>> items; // adaptor over a vector container
  // stack<char, list<char>> items; // adaptor over a list container

  for (char c = 65; c < 65 + 26; c++) {
    cout << c << " ";
    items.push(c);
  }
  cout << endl;

  while (!items.empty()) {
    cout << items.top() << " ";
    items.pop();
  }
  cout << endl;
}
```

Κώδικας 5.3: Παράδειγμα χρήσης της std::stack (stl_stack_example.cpp)

```
std::stack example
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

5.4.2 std::queue

Στην περίπτωση του std::queue το εσωτερικό container μπορεί να είναι κάποιο από τα containers std::dequeue, std::list (προκαθορισμένη επιλογή) ή οποιοδήποτε container που υποστηρίζει τις λειτουργίες empty(), size(), front(), back(), push_back() και pop_front() [4]. Τυπικές λειτουργίες που παρέχει η std::queue είναι οι ακόλουθες:

- empty(), ελέγχει αν η ουρά είναι άδεια.
- size(), επιστρέφει το μέγεθος της ουράς.
- front(), προσπελαύνει το στοιχείο που βρίσκεται στο εμπρός άκρο της ουράς (χωρίς να το αφαιρεί).
- back(), προσπελαύνει το στοιχείο που βρίσκεται στο πίσω άκρο της ουράς (χωρίς να το αφαιρεί).
- push(), εισάγει ένα στοιχείο στο πίσω άκρο της ουράς
- pop(), εξάγει το στοιχείο που βρίσκεται στο εμπρός άκρο της ουράς.

Ένα παράδειγμα χρήσης της std::queue παρουσιάζεται στη συνέχεια.

```
#include <iostream>
#include <queue>
#include <list>

using namespace std;

int main() {
  cout << "std::queue" << endl;
  queue<int> aqueue; // adaptor over a deque container
```

```
10    // queue<int, deque<int>> aqueue; // adaptor over a deque container
11    // queue<int, list<int>> aqueue; // adaptor over a list container
12
13    cout << "1. Enqueue 10 items" << endl;
14    for (int i = 51; i < 60; i++) {
15       cout << i << " ";
16       aqueue.push(i);
17    }
18    cout << endl << "2. Dequeue 5 items" << endl;
19    for (int i = 0; i < 5; i++) {
20       cout << aqueue.front() << " ";
21       aqueue.pop();
22    }
23    cout << endl << "3. Enqueue 3 items" << endl;
24    for (int i = 61; i <= 63; i++) {
25       cout << i << " ";
26       aqueue.push(i);
27    }
28    while (!aqueue.empty()) {
29       cout << aqueue.front() << " ";
30       aqueue.pop();
31    }
32    cout << endl;
33 }
```

Κώδικας 5.4: Παράδειγμα χρήσης της std::queue (stl_queue_example.cpp)

```
1 std::queue
2 1. Enqueue 10 items
3 51 52 53 54 55 56 57 58 59
4 2. Dequeue 5 items
5 51 52 53 54 55
6 3. Enqueue 3 items
7 61 62 63 56 57 58 59 61 62 63
```

5.5 Παραδείγματα

5.5.1 Παράδειγμα 1

Να γραφεί πρόγραμμα που να δέχεται μια φράση ως παράμετρο γραμμής εντολών (command line argument) και να εμφανίζει το εάν είναι παλινδρομική ή όχι. Μια φράση είναι παλινδρομική όταν διαβάζεται η ίδια από αριστερά προς τα δεξιά και από δεξιά προς τα αριστερά.

```
1  #include <iostream>
2  #include <stack>
3
4  using namespace std;
5  // examples of palindromic sentences:
6  // SOFOS, A MAN A PLAN A CANAL PANAMA, AMORE ROMA, LIVE NOT ON EVIL
7  int main(int argc, char **argv) {
8     if (argc != 2) {
9        cerr << "Usage examples: " << endl;
10       cerr << "\t\t " << argv[0] << " SOFOS" << endl;
11       cerr << "\t\t " << argv[0] << " \"A MAN A PLAN A CANAL PANAMA\"" << endl;
12       exit(-1);
13    }
14    string str = argv[1];
15    stack<char> astack;
16    string str1;
17    for (char c : str)
```

```
18      if (c != ' ') {
19          str1 += c;
20          astack.push(c);
21      }
22    string str2;
23    while (!astack.empty()) {
24      str2 += astack.top();
25      astack.pop();
26    }
27    if (str1 == str2)
28      cout << "The sentence " << str << " is palindromic." << endl;
29    else
30      cout << "The string " << str << " is not palindromic." << endl;
31  }
```

Κώδικας 5.5: Έλεγχος παλινδρομικής φράσης (lab05_ex1.cpp)

```
1  $ ./lab05_ex1
2  Usage examples:
3                      ./lab05_ex1 SOFOS
4                      ./lab05_ex1 "A MAN A PLAN A CANAL PANAMA"
5
6  $ ./lab05_ex1 "A MAN A PLAN A A CANAL PANAMA"
7  The string A MAN A PLAN A A CANAL PANAMA is not palindromic.
8
9  $ ./lab05_ex1 "A MAN A PLAN A A CANAL PANAM"
10 The string A MAN A PLAN A A CANAL PANAM is not palindromic.
```

5.5.2 Παράδειγμα 2

Να γραφεί πρόγραμμα που να δέχεται ένα δυαδικό αριθμό ως λεκτικό και να εμφανίζει την ισοδύναμη δεκαδική του μορφή.

```
1  #include <iostream>
2  #include <stack>
3  #include <string> // stoi
4
5  using namespace std;
6  int main() {
7    string bs;
8    stack<int> astack;
9    cout << "Enter a binary number: ";
10   cin >> bs;
11   for (char c : bs) {
12     if (c != '0' && c != '1') {
13       cerr << "use only digits 0 and 1" << endl;
14       exit(-1);
15     }
16     astack.push(c - '0');
17   }
18
19   int sum = 0, x = 1;
20   while (!astack.empty()) {
21     sum += astack.top() * x;
22     astack.pop();
23     x *= 2;
24   }
25   cout << "Decimal: " << sum << endl;
26
27   cout << "Decimal: " << stoi(bs, nullptr, 2) << endl; // one line solution :)
28 }
```

Κώδικας 5.6: Μετατροπή δυαδικού σε δεκαδικό (lab05_ex2.cpp)

```
1 Enter a binary number: 10101010101010101111111100111
2 Decimal: 178958311
3 Decimal: 178958311
```

5.6 Ασκήσεις

1. Να υλοποιηθεί η δομή της ουράς χρησιμοποιώντας αντικείμενα στοίβας (std::stack) και τις λειτουργίες που επιτρέπονται σε αυτά. Υλοποιήστε τις λειτουργίες της ουράς empty(), size(), enqueue(), dequeue() και front().

2. Να υλοποιηθεί η δομή της στοίβας χρησιμοποιώντας αντικείμενα ουράς (std::queue) και τις λειτουργίες που επιτρέπονται σε αυτά. Υλοποιήστε τις λειτουργίες της στοίβας empty(), size(), push(), pop() και top().

Βιβλιογραφία

[1] Sixteen Ways To Stack a Cat, by Bjarne Stroustrup http://www.stroustrup.com/stack_cat.pdf

[2] Tech Crash Course, C Program to Implement a Stack using Singly Linked List, http://www.techcrashcourse.com/2016/06/c-program-implement-stack-using-linked-list.html

[3] C++ Reference Material by Porter Scobey, The STL stack Container Adaptor http://cs.stmarys.ca/ porter/csc/ref/stl/cont_stack.html

[4] C++ Reference Material by Porter Scobey, The STL queue Container Adaptor http://cs.stmarys.ca/ porter/csc/ref/stl/cont_queue.html

Εργαστήριο 6

Σωροί μεγίστων και σωροί ελαχίστων, η ταξινόμηση heapsort, ουρές προτεραιότητας στην STL

6.1 Εισαγωγή

Οι σωροί επιτρέπουν την οργάνωση των δεδομένων με τέτοιο τρόπο έτσι ώστε το μεγαλύτερο στοιχείο να είναι συνεχώς προσπελάσιμο σε σταθερό χρόνο. Η δε λειτουργίες της εισαγωγής νέων τιμών στη δομή και της διαγραφή της μεγαλύτερης τιμής πραγματοποιούνται ταχύτατα. Σε αυτό το εργαστήριο θα παρουσιαστεί η υλοποίηση ενός σωρού μεγίστων και ο σχετικός με τη δομή αυτή αλγόριθμος ταξινόμησης, heapsort. Επιπλέον, θα παρουσιαστεί η δομή std::priority_queue που υλοποιεί στην STL της C++ τους σωρούς μεγίστων και ελαχίστων. Ο κώδικας όλων των παραδειγμάτων βρίσκεται στο https://github.com/chgogos/ceteiep_dsa.

6.2 Σωροί

Ο σωρός είναι μια μερικά ταξινομημένη δομή δεδομένων. Υπάρχουν δύο βασικά είδη σωρών: ο σωρός μεγίστων (MAXHEAP) και ο σωρός ελαχίστων (MINHEAP). Οι ιδιότητες των σωρών που θα περιγραφούν στη συνέχεια αφορούν τους σωρούς μεγίστων αλλά αντίστοιχες ιδιότητες ισχύουν και για τους σωρούς ελαχίστων. Ειδικότερα, ένας σωρός μεγίστων υποστηρίζει ταχύτατα τις ακόλουθες λειτουργίες:

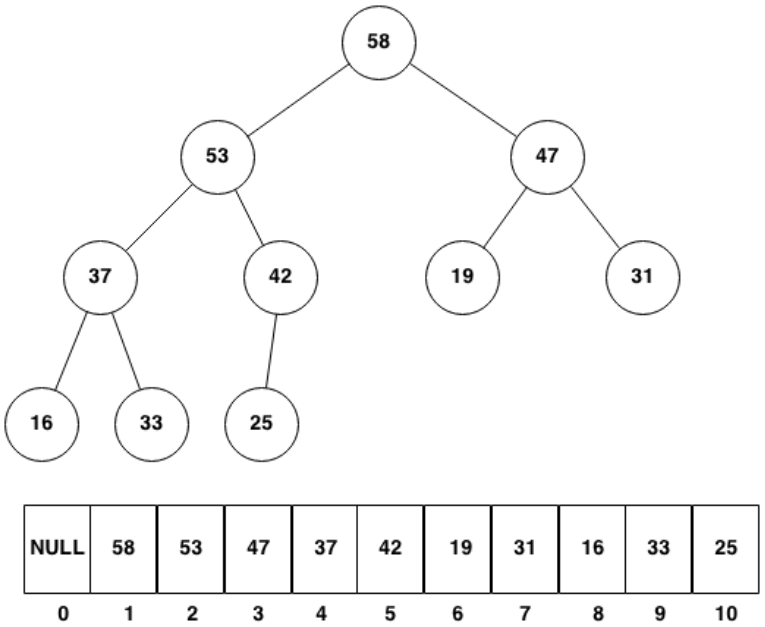
- Εύρεση του στοιχείου με τη μεγαλύτερη τιμή κλειδιού.
- Διαγραφή του στοιχείου με τη μεγαλύτερη τιμή κλειδιού.
- Εισαγωγή νέου κλειδιού στη δομή.

Ένας σωρός μπορεί να θεωρηθεί ως ένα δυαδικό δένδρο για το οποίο ισχύουν οι ακόλουθοι δύο περιορισμοί:

- *Πληρότητα*: το δυαδικό δένδρο είναι συμπληρωμένο, δηλαδή όλα τα επίπεδά του είναι πλήρως συμπληρωμένα εκτός πιθανά από το τελευταίο (χαμηλότερο) επίπεδο στο οποίο μπορούν να λείπουν μόνο κάποια από τα δεξιότερα φύλλα.
- *Κυριαρχία γονέα*: το κλειδί σε κάθε κορυφή είναι μεγαλύτερο ή ίσο από τα κλειδιά των παιδιών (σε MAXHEAP).

Ένας σωρός μπορεί να υλοποιηθεί με ένα πίνακα καταγράφοντας στον πίνακα στη σειρά τα στοιχεία του δυαδικού δένδρου από αριστερά προς τα δεξιά και από πάνω προς τα κάτω (σχήμα 6.1). Μερικές σημαντικές ιδιότητες οι οποίες προκύπτουν εφόσον τηρηθεί ο παραπάνω τρόπος αντιστοίχισης των στοιχείων του δένδρου στα στοιχεία του πίνακα είναι οι ακόλουθες:

- Στον πίνακα, τα κελιά γονείς βρίσκονται στις πρώτες $\lfloor \frac{n}{2} \rfloor$ θέσεις ενώ τα φύλλα καταλαμβάνουν τις υπόλοιπες θέσεις.
- Στον πίνακα, τα παιδιά για κάθε κλειδί στις θέσεις i από 1 μέχρι και $\lfloor \frac{n}{2} \rfloor$ βρίσκονται στις θέσεις $2 * i$ και $2 * i + 1$.
- Στον πίνακα, ο γονέας για κάθε κλειδί στις θέσεις i από 2 μέχρι και n βρίσκεται στη θέση $\lfloor \frac{i}{2} \rfloor$.



Σχήμα 6.1: Αναπαράσταση ενός σωρού μεγίστων ως πίνακα

Για το παράδειγμα του σχήματος ισχύουν τα ακόλουθα:

- Οι κόμβοι που είναι γονείς (έχουν τουλάχιστον ένα παιδί) βρίσκονται στις θέσεις από 1 μέχρι και 5.
- Οι κόμβοι που είναι φύλλα βρίσκονται στις θέσεις από 6 μέχρι και 10.
- Ο γονέας στη θέση 1 (η τιμή 58) έχει παιδιά στις θέσεις $2 * 1 = 2$ (τιμή 53) και $2 * 1 + 1 = 3$ (τιμή 47).
- Ο γονέας στη θέση 2 (η τιμή 53) έχει παιδιά στις θέσεις $2 * 2 = 4$ (τιμή 37) και $2 * 2 + 1 = 5$ (τιμή 42).
- Ο γονέας στη θέση 3 (η τιμή 47) έχει παιδιά στις θέσεις $2 * 3 = 6$ (τιμή 19) και $2 * 3 + 1 = 7$ (τιμή 31).
- Ο γονέας στη θέση 4 (η τιμή 37) έχει παιδιά στις θέσεις $2 * 4 = 8$ (τιμή 16) και $2 * 4 + 1 = 9$ (τιμή 33).
- Ο γονέας στη θέση 5 (η τιμή 42) έχει παιδιά στις θέσεις $2 * 5 = 10$ (τιμή 25).
- Ο κόμβος παιδί στη θέση 2 (η τιμή 53) έχει γονέα στη θέση $\lfloor \frac{2}{2} \rfloor = 1$ (τιμή 58).
- Ο κόμβος παιδί στη θέση 3 (η τιμή 47) έχει γονέα στη θέση $\lfloor \frac{3}{2} \rfloor = 1$ (τιμή 58).
- Ο κόμβος παιδί στη θέση 4 (η τιμή 37) έχει γονέα στη θέση $\lfloor \frac{4}{2} \rfloor = 2$ (τιμή 53).
- Ο κόμβος παιδί στη θέση 5 (η τιμή 42) έχει γονέα στη θέση $\lfloor \frac{5}{2} \rfloor = 2$ (τιμή 53).
- Ο κόμβος παιδί στη θέση 6 (η τιμή 19) έχει γονέα στη θέση $\lfloor \frac{6}{2} \rfloor = 3$ (τιμή 47).
- Ο κόμβος παιδί στη θέση 7 (η τιμή 31) έχει γονέα στη θέση $\lfloor \frac{7}{2} \rfloor = 3$ (τιμή 47).

- Ο κόμβος παιδί στη θέση 8 (η τιμή 16) έχει γονέα στη θέση $\lfloor \frac{8}{2} \rfloor = 4$ (τιμή 37).
- Ο κόμβος παιδί στη θέση 9 (η τιμή 33) έχει γονέα στη θέση $\lfloor \frac{9}{2} \rfloor = 4$ (τιμή 37).
- Ο κόμβος παιδί στη θέση 10 (η τιμή 25) έχει γονέα στη θέση $\lfloor \frac{10}{2} \rfloor = 5$ (τιμή 42).

6.3 Υλοποίηση ενός σωρού

Στη συνέχεια παρουσιάζεται η υλοποίηση ενός σωρού μεγίστων που περιέχει ακέραιες τιμές-κλειδιά.

```
1  #include <iostream>
2  using namespace std;
3
4  // MAXHEAP
5  const int static HEAP_SIZE_LIMIT = 100000;
6  int heap[HEAP_SIZE_LIMIT + 1];
7  int heap_size = 0;
8
9  void clear_heap() {
10   for (int i = 0; i < HEAP_SIZE_LIMIT + 1; i++)
11     heap[i] = 0;
12   heap_size = 0;
13 }
14
15 void print_heap(bool newline = true) {
16   cout << "HEAP(" << heap_size << ") [";
17   for (int i = 1; i <= heap_size; i++)
18     if (i == heap_size)
19       cout << heap[i];
20     else
21       cout << heap[i] << " ";
22   cout << "]";
23   if (newline)
24     cout << endl;
25 }
26
27 void heapify(int k) {
28   int v = heap[k];
29   bool flag = false;
30   while (!flag && 2 * k <= heap_size) {
31     int j = 2 * k;
32     if (j < heap_size)
33       if (heap[j] < heap[j + 1])
34         j++;
35     if (v >= heap[j])
36       flag = true;
37     else {
38       heap[k] = heap[j];
39       k = j;
40     }
41   }
42   heap[k] = v;
43 }
44
45 void heap_bottom_up(int *a, int N, bool verbose = false) {
46   heap_size = N;
47   for (int i = 0; i < N; i++)
48     heap[i + 1] = a[i];
49   for (int i = heap_size / 2; i >= 1; i--) {
50     if (verbose)
```

```
51          cout << "heapify " << heap[i] << " ";
52        heapify(i);
53        if (verbose)
54          print_heap();
55      }
56  }
57
58  bool empty() {
59    return (heap_size==0);
60  }
61
62  int top() { return heap[1]; }
63
64  void push(int key) {
65    heap_size++;
66    heap[heap_size] = key;
67    int pos = heap_size;
68    while (pos != 1 && heap[pos / 2] < heap[pos]) {
69      swap(heap[pos / 2], heap[pos]);
70      pos = pos / 2;
71    }
72  }
73
74  void pop() {
75    swap(heap[1], heap[heap_size]);
76    heap_size−−;
77    heapify(1);
78  }
```

Κώδικας 6.1: Σωρός μεγίστων με κλειδιά ακέραιες τιμές (max_heap.cpp)

Οι συναρτήσεις δημιουργίας σωρού από πίνακα, (heap_bottom_up() και heapify()

Ένας πίνακας μπορεί να μετασχηματιστεί ταχύτατα σε σωρό. Η διαδικασία ξεκινά από τον τελευταίο κόμβο γονέα του δένδρου (που βρίσκεται στη θέση $\lfloor \frac{n}{2} \rfloor$) και σταδιακά εφαρμόζεται μέχρι να φτάσει στον κόμβο στη θέση 1. Για καθένα από αυτούς τους κόμβους εξετάζεται από πάνω προς τα κάτω αν ισχύει η κυριαρχία γονέα και αν δεν ισχύει τότε γίνεται αντιμετάθεση με το μεγαλύτερο από τα παιδιά του επαναληπτικά.

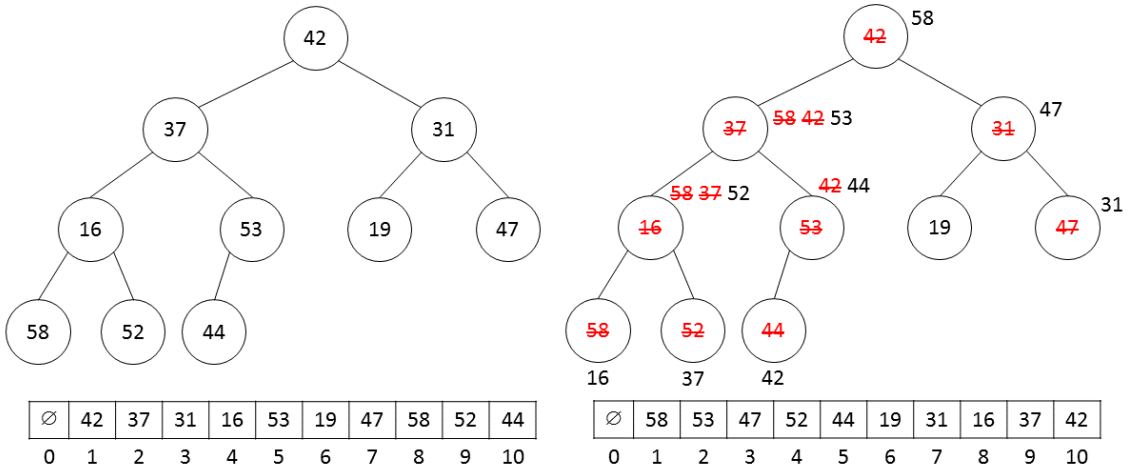
Ο ακόλουθος κώδικας χρησιμοποιεί τη συνάρτηση heap_bottom_up() και μέσω αυτής τη συνάρτηση heapify() προκειμένου να μετασχηματίσει έναν πίνακα ακεραίων σε σωρό μεγίστων.

```
1  #include "max_heap.cpp"
2
3  int main(void) {
4    cout << "#### Test heap construction with heapify ####" << endl;
5    int a[10] = {42, 37, 31, 16, 53, 19, 47, 58, 52, 44};
6    heap_bottom_up(a, 10, true);
7    print_heap();
8  }
```

Κώδικας 6.2: Δημιουργία σωρού από πίνακα με heapify (heap1.cpp)

```
1  #### Test heap construction with heapify ####
2  heapify 53 HEAP(10) [42 37 31 16 53 19 47 58 52 44]
3  heapify 16 HEAP(10) [42 37 31 58 53 19 47 16 52 44]
4  heapify 31 HEAP(10) [42 37 47 58 53 19 31 16 52 44]
5  heapify 37 HEAP(10) [42 58 47 52 53 19 31 16 37 44]
6  heapify 42 HEAP(10) [58 53 47 52 44 19 31 16 37 42]
7  HEAP(10) [58 53 47 52 44 19 31 16 37 42]
```

Στο σχήμα 6.2 παρουσιάζονται οι τιμές που έλαβε κάθε κόμβος του δένδρου προκειμένου να μετασχηματιστεί τελικά σε σωρό μεγίστων.



Σχήμα 6.2: Δημιουργία σωρού από πίνακα (heapify)

Η συνάρτηση ελέγχου του εάν ο σωρός είναι άδειος, empty()

Η συνάρτηση empty εξετάζει το μέγεθος του σωρού μέσω της μεταβλητής heap_size. Αν η μεταβλητή heap_size είναι μηδέν τότε επιστρέφει true, αλλιώς επιστρέφει false.

Η συνάρτηση λήψης της μεγαλύτερης τιμής από το σωρό, top()

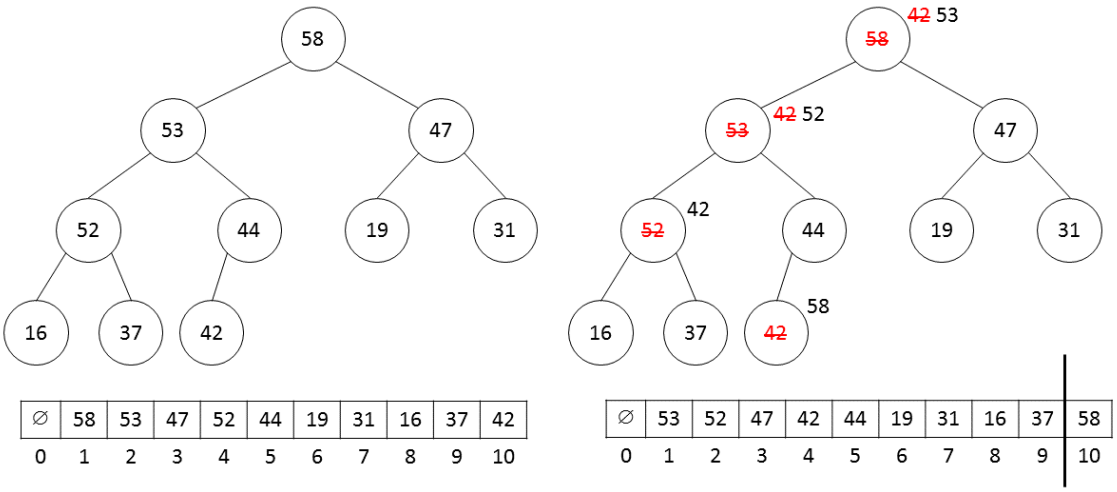
Καθώς η μεγαλύτερη τιμή βρίσκεται πάντα στη θέση 1 του πίνακα που διατηρεί τα δεδομένα του σωρού η συνάρτηση top απλά επιστρέφει την τιμή αυτή.

Η συνάρτηση εξαγωγής της μεγαλύτερης τιμής από το σωρό, pop()

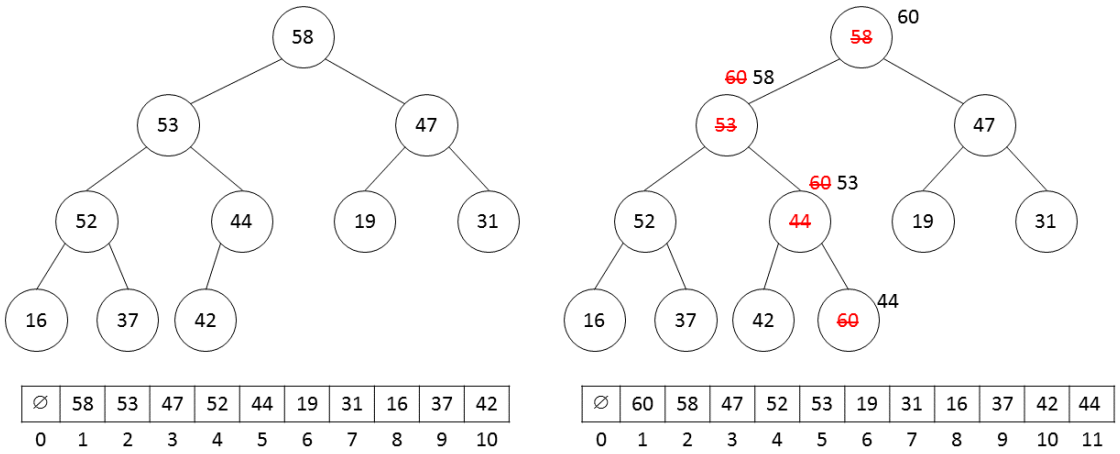
Η εξαγωγή της μεγαλύτερης τιμής γίνεται ως εξής. Το στοιχείο που βρίσκεται στην κορυφή του σωρού αντιμετατίθεται με το τελευταίο στοιχείο του σωρού. Στη συνέχεια το στοιχείο που έχει βρεθεί στην κορυφή του σωρού κατεβαίνει προς τα κάτω αν έχει παιδί που είναι μεγαλύτερό του πραγματοποιώντας αντιμετάθεση με το μεγαλύτερο στοιχείο από τα παιδιά του. Η διαδικασία επαναλαμβάνεται για τη νέα θέση του στοιχείου που αρχικά είχε μεταφερθεί στη κορυφή και μέχρι να ισχύσει ότι είναι μεγαλύτερο και από τα δύο παιδιά του. Στο σχήμα 6.3 παρουσιάζεται η εξαγωγή της κορυφαίας τιμής του σωρού.

Η συνάρτηση εισαγωγής νέας τιμής στο σωρό, push()

Η εισαγωγή ενός στοιχείου γίνεται ως φύλλο στη πρώτη διαθέσιμη θέση από πάνω προς τα κάτω και από δεξιά προς τα αριστερά. Το στοιχείο αυτό συγκρίνεται με το γονέα του και αν είναι μεγαλύτερο αντιμετατίθεται με αυτόν. Η διαδικασία συνεχίζεται μέχρι είτε να βρεθεί το νέο στοιχείο στην κορυφή είτε να ισχύει η κυριαρχία γονέα. Στο σχήμα 6.4 παρουσιάζεται η εισαγωγή της τιμής 60 σε έναν σωρό μεγίστων.



Σχήμα 6.3: Εξαγωγή της μεγαλύτερης τιμής του σωρού (pop)



Σχήμα 6.4: Εισαγωγή της τιμής 60 στο σωρό (push)

Παράδειγμα χρήσης των συναρτήσεων push() και pop()

Ο ακόλουθος κώδικας δημιουργεί σταδιακά έναν σωρό εισάγοντας δέκα τιμές με τη συνάρτηση push(). Στη συνέχεια πραγματοποιούνται εξαγωγές τιμών με τη συνάρτηση pop() μέχρι ο σωρός να αδειάσει.

```
1  #include ”max_heap.cpp”
2
3  int main(void) {
4    int a[10] = {42, 37, 31, 16, 53, 19, 47, 58, 33, 25};
5    for (int i = 0; i < 10; i++) {
6      print_heap(false);
7      cout << ” ==> push key ” << a[i] << ” ==> ”;
8      push(a[i]);
9      print_heap();
10   }
11   while (heap_size > 0) {
12     print_heap(false);
13     cout << ” ==> pop ==> key=” << heap[1] << ”, ”;
14     pop();
```

```
15      print_heap();
16    }
17  }
```

Κώδικας 6.3: Δημιουργία σωρού με εισαγωγές τιμών και εν συνεχεία άδειασμα του σωρού με διαδοχικές διαγραφές της μέγιστης τιμής (heap2.cpp)

```
1  HEAP(0) [] ==> push key 42 ==> HEAP(1) [42]
2  HEAP(1) [42] ==> push key 37 ==> HEAP(2) [42 37]
3  HEAP(2) [42 37] ==> push key 31 ==> HEAP(3) [42 37 31]
4  HEAP(3) [42 37 31] ==> push key 16 ==> HEAP(4) [42 37 31 16]
5  HEAP(4) [42 37 31 16] ==> push key 53 ==> HEAP(5) [53 42 31 16 37]
6  HEAP(5) [53 42 31 16 37] ==> push key 19 ==> HEAP(6) [53 42 31 16 37 19]
7  HEAP(6) [53 42 31 16 37 19] ==> push key 47 ==> HEAP(7) [53 42 47 16 37 19 31]
8  HEAP(7) [53 42 47 16 37 19 31] ==> push key 58 ==> HEAP(8) [58 53 47 42 37 19 31 16]
9  HEAP(8) [58 53 47 42 37 19 31 16] ==> push key 33 ==> HEAP(9) [58 53 47 42 37 19 31 16 33]
10 HEAP(9) [58 53 47 42 37 19 31 16 33] ==> push key 25 ==> HEAP(10) [58 53 47 42 37 19 31 16 33 25]
11 HEAP(10) [58 53 47 42 37 19 31 16 33 25] ==> pop ==> key=58, HEAP(9) [53 42 47 33 37 19 31 16 25]
12 HEAP(9) [53 42 47 33 37 19 31 16 25] ==> pop ==> key=53, HEAP(8) [47 42 31 33 37 19 25 16]
13 HEAP(8) [47 42 31 33 37 19 25 16] ==> pop ==> key=47, HEAP(7) [42 37 31 33 16 19 25]
14 HEAP(7) [42 37 31 33 16 19 25] ==> pop ==> key=42, HEAP(6) [37 33 31 25 16 19]
15 HEAP(6) [37 33 31 25 16 19] ==> pop ==> key=37, HEAP(5) [33 25 31 19 16]
16 HEAP(5) [33 25 31 19 16] ==> pop ==> key=33, HEAP(4) [31 25 16 19]
17 HEAP(4) [31 25 16 19] ==> pop ==> key=31, HEAP(3) [25 19 16]
18 HEAP(3) [25 19 16] ==> pop ==> key=25, HEAP(2) [19 16]
19 HEAP(2) [19 16] ==> pop ==> key=19, HEAP(1) [16]
20 HEAP(1) [16] ==> pop ==> key=16, HEAP(0) []
```

6.4 Ταξινόμηση Heapsort

Ο αλγόριθμος Heapsort προτάθηκε από τον J.W.J.Williams το 1964 [1] και αποτελείται από 2 στάδια:

- Δημιουργία σωρού με τα n στοιχεία ενός πίνακα που ζητείται να ταξινομηθούν.
- Εφαρμογή της διαγραφής της ρίζας n -1 φορές.

Το αποτέλεσμα είναι ότι τα στοιχεία αφαιρούνται από το σωρό σε φθίνουσα σειρά (για έναν σωρό μεγίστων). Καθώς κατά την αφαίρεσή του κάθε στοιχείου, αυτό τοποθετείται στο τέλος του σωρού, τελικά ο σωρός περιέχει τα αρχικά δεδομένα σε αύξουσα σειρά.

Στη συνέχεια παρουσιάζεται η υλοποίηση του αλγορίθμου Heapsort. Επιπλέον ο κώδικας ταξινομεί πίνακες μεγέθους 10.000, 20.000, 40.000 80.000 και 100.000 που περιέχουν τυχαίες ακέραιες τιμές και πραγματοποιείται σύγκριση με τους χρόνους εκτέλεσης που επιτυγχάνει η std::sort().

```
1  #include "max_heap.cpp"
2  #include <algorithm>
3  #include <chrono>
4  #include <random>
5
6  using namespace std::chrono;
7
8  void heapsort() {
9    while (!empty())
10     pop();
11 }
12
13 int main(void) {
14   high_resolution_clock::time_point t1, t2;
15   mt19937 mt(1940);
16   uniform_int_distribution<int> uni(0, 200000);
17   int problem_sizes[] = {10000, 20000, 40000, 80000, 100000};
18   for (int i = 0; i < 5; i++) {
```

```
19      clear_heap();
20      int N = problem_sizes[i];
21      int *a = new int[N];
22      for (int i = 0; i < N; i++)
23        a[i] = uni(mt);
24      heap_bottom_up(a, N);
25      t1 = high_resolution_clock::now();
26      heapsort();
27      t2 = high_resolution_clock::now();
28      duration<double, std::milli> duration1 = t2 − t1;
29      for (int i = 0; i < N; i++)
30        a[i] = uni(mt);
31      t1 = high_resolution_clock::now();
32      sort(a, a + N);
33      t2 = high_resolution_clock::now();
34      duration<double, std::milli> duration2 = t2 − t1;
35      cout << ”SIZE ” << N << ” heap sort ” << duration1.count()
36           << ”ms std::sort ” << duration2.count() << ”ms” << endl;
37      delete[] a;
38    }
39  }
```

Κώδικας 6.4: Ο αλγόριθμος heapsort (heapsort.cpp)

```
1  SIZE 10000 heap sort 4.0003ms std::sort 4.0003ms
2  SIZE 20000 heap sort 5.0003ms std::sort 4.0002ms
3  SIZE 40000 heap sort 10.0006ms std::sort 10.0006ms
4  SIZE 80000 heap sort 19.0011ms std::sort 18.001ms
5  SIZE 100000 heap sort 24.0014ms std::sort 22.0013ms
```

Περισσότερες πληροφορίες για την ταξινόμηση heapsort μπορούν να βρεθούν στην αναφορά [2].

6.5 Η δομή priority_queue της STL

Η STL της C++ περιέχει υλοποίηση της δομής std::priority_queue (ουρά προτεραιότητας) η οποία είναι ένας σωρός μεγίστων. Κάθε στοιχείο που εισέρχεται στην ουρά προτεραιότητας έχει μια προτεραιότητα που συνδέεται με αυτό και το στοιχείο με τη μεγαλύτερη προτεραιότητα βρίσκεται πάντα στην αρχή της ουράς. Οι κυριότερες λειτουργίες που υποστηρίζονται από την std::priority_queue είναι οι ακόλουθες:

- push(), εισαγωγή ενός στοιχείου στη δομή.
- top(), επιστροφή χωρίς εξαγωγή του στοιχείου με τη μεγαλύτερη προτεραιότητα.
- pop(), απώθηση του στοιχείου με τη μεγαλύτερη προτεραιότητα.
- size(), πλήθος των στοιχείων που υπάρχουν στη δομή.
- empty(), επιστρέφει true αν η δομή είναι άδεια αλλιώς επιστρέφει false.

Η std::priority_queue είναι ένας container adaptor που χρησιμοποιεί ως εσωτερικό container ένα std::vector. Εναλλακτικά μπορεί να χρησιμοποιηθεί std::deque το οποίο όπως και το std::vector παρέχει τις λειτουργίες empty(), size(), push_back(), pop_back() και front() που απαιτούνται.

Ένα παράδειγμα χρήσης της std::priority_queue ως σωρού μεγίστων αλλά και ως σωρού ελαχίστων παρουσιάζεται στη συνέχεια.

```
1  #include <algorithm>
2  #include <iostream>
3  #include <queue>
4
5  using namespace std;
6
7  int main(void) {
```

```
8    int a[10] = {15, 16, 13, 23, 45, 67, 11, 22, 37, 10};
9    cout << "priority queue (MAXHEAP): ";
10   priority_queue<int> pq1(a, a + 10);
11   while (!pq1.empty()) {
12      int x = pq1.top();
13      pq1.pop();
14      cout << x << " ";
15   }
16   cout << endl;
17
18   cout << "priority queue (MINHEAP): ";
19   priority_queue<int, std::vector<int>, std::greater<int>> pq2(a, a + 10);
20   while (!pq2.empty()) {
21      int x = pq2.top();
22      pq2.pop();
23      cout << x << " ";
24   }
25   cout << endl;
26 }
```

Κώδικας 6.5: Παράδειγμα με priority_queue της STL (stl_priority_queue.cpp)

```
1 priority queue (MAXHEAP): 67 45 37 23 22 16 15 13 11 10
2 priority queue (MINHEAP): 10 11 13 15 16 22 23 37 45 67
```

Περισσότερες πληροφορίες για τη δομή std::priority_queue μπορούν να βρεθούν στις αναφορές [3] και [4].

6.6 Παραδείγματα

6.6.1 Παράδειγμα 1

Χρησιμοποιώντας τον κώδικα 1, να γραφεί πρόγραμμα που να εισάγει 100.000 τυχαίες ακέραιες τιμές (στο διάστημα [-1.000.000,1.000.000]) σε έναν σωρό μεγίστων με τη συνάρτηση heap_bottom_up() καθώς και με διαδοχικές κλήσεις της συνάρτησης push(). Χρονομετρείστε τον κώδικα και στις δύο περιπτώσεις δημιουργίας του σωρού και εμφανίστε το κορυφαίο στοιχείο του σωρού. Επαναλάβετε τη διαδικασία χρησιμοποιώντας την std::priority_queue.

```
1  #include "max_heap.cpp"
2  #include <chrono>
3  #include <queue>
4  #include <random>
5
6
7  using namespace std::chrono;
8
9  int main(void) {
10   constexpr int N = 100000;
11   mt19937 mt(1821);
12   int a[N];
13   uniform_int_distribution<int> dist(−1000000, 1000000);
14   for (int i = 0; i < N; i++)
15      a[i] = dist(mt);
16
17   auto t1 = high_resolution_clock::now();
18   heap_bottom_up(a, N, false);
19   auto t2 = high_resolution_clock::now();
20   std::chrono::duration<double, std::micro> duration_micro_sec = t2 − t1;
21   cout << "A. Top item: " << top() << endl;
22   cout << "Time elapsed (heap_bottom_up): " << duration_micro_sec.count()
```

```
23          << " microseconds " << endl;
24
25    clear_heap();
26
27    t1 = high_resolution_clock::now();
28    for (int i = 0; i < N; i++)
29       push(a[i]);
30    t2 = high_resolution_clock::now();
31    duration_micro_sec = t2 - t1;
32    cout << "B. Top item: " << top() << endl;
33    cout << "Time elapsed (push): " << duration_micro_sec.count()
34          << " microseconds " << endl;
35
36    t1 = high_resolution_clock::now();
37    priority_queue<int> pq(a, a + N);
38    t2 = high_resolution_clock::now();
39    duration_micro_sec = t2 - t1;
40    cout << "C. Top item: " << pq.top() << endl;
41    cout << "Time elapsed (push): " << duration_micro_sec.count()
42          << " microseconds " << endl;
43 }
```

Κώδικας 6.6: Χρόνος δημιουργίας MAXHEAP: A) με την heap_bottom_up() B) με σταδιακές εισαγωγές (push) τιμών στο σωρό και C) με την std::priority_queue (lab06_ex1.cpp)

```
1 A. Top item: 999994
2 Time elapsed (heap_bottom_up): 3000.2 microseconds
3 B. Top item: 999994
4 Time elapsed (push): 6000.4 microseconds
5 C. Top item: 999994
6 Time elapsed (push): 11000.6 microseconds
```

6.6.2 Παράδειγμα 2

Έστω ένα παιχνίδι στο οποίο οι παίκτες έχουν όνομα (name) και επίδοση (score). Να γράψετε πρόγραμμα στο οποίο να εισέρχονται στο παιχνίδι 10 παίκτες στη σειρά (player1, player2, ...), πετυχαίνοντας κάποια επίδοση ο καθένας (τυχαίος ακέραιος από το 0 μέχρι το 50.000). Να εμφανίζεται μετά την εισαγωγή του κάθε παίκτη ο παίκτης που προηγείται και η επίδοση του. Τέλος, να εμφανίζονται τα ονόματα των παικτών με τις 3 υψηλότερες επιδόσεις.

```
1  #include <iostream>
2  #include <queue>
3  #include <random>
4  #include <string>
5  #define N 10
6  #define TOP 3
7
8  using namespace std;
9  struct player {
10    string name;
11    int score;
12    bool operator<(const player &other) const { return score < other.score; }
13 };
14
15 int main() {
16    mt19937 mt(1821);
17    uniform_int_distribution<int> dist(0, 50000);
18    priority_queue<player> pq;
19    int best_score = -1;
20    for (int i = 0; i < N; i++) {
```

```
21      player p;
22      p.name = "player" + to_string(i + 1);
23      p.score = dist(mt);
24      pq.push(p);
25      player top_player = pq.top();
26      if (top_player.score != best_score)
27        best_score = top_player.score;
28      cout << "New player: " << p.name << " with score " << p.score << " best["
29           << top_player.name << " score " << top_player.score << "]" << endl;
30    }
31    cout << "Top " << TOP << " players:" << endl;
32    for (int i = 0; i < TOP; i++) {
33      player p = pq.top();
34      cout << i + 1 << " " << p.name << " " << p.score << endl;
35      pq.pop();
36    }
37  }
```

Κώδικας 6.7: Διατήρηση επιδόσεων σε σωρό (lab06_ex2.cpp)

```
1  New player: player1 with score 36323 best[player1 score 36323]
2  New player: player2 with score 21613 best[player1 score 36323]
3  New player: player3 with score 33218 best[player1 score 36323]
4  New player: player4 with score 32634 best[player1 score 36323]
5  New player: player5 with score 454 best[player1 score 36323]
6  New player: player6 with score 48987 best[player6 score 48987]
7  New player: player7 with score 25627 best[player6 score 48987]
8  New player: player8 with score 42239 best[player6 score 48987]
9  New player: player9 with score 9284 best[player6 score 48987]
10 New player: player10 with score 11639 best[player6 score 48987]
11 Top 3 players:
12 1 player6 48987
13 2 player8 42239
14 3 player1 36323
```

6.6.3 Παράδειγμα 3

Διάμεσος ενός δείγματος N παρατηρήσεων οι οποίες έχουν διαταχθεί σε αύξουσα σειρά ορίζεται ως η μεσαία παρατήρηση, όταν το N είναι περιττός αριθμός, ή ο μέσος όρος (ημιάθροισμα) των δύο μεσαίων παρατηρήσεων όταν το N είναι άρτιος αριθμός. Έστω ότι για διάφορες τιμές που παράγονται με κάποιον τρόπο ζητείται ο υπολογισμός της διάμεσης τιμής καθώς παράγεται κάθε νέα τιμή και για όλες τις τιμές που έχουν προηγηθεί μαζί με την τρέχουσα τιμή όπως φαίνεται στο επόμενο παράδειγμα:
5 ⇒ διάμεσος 5
5, 7 ⇒ διάμεσος 6
5, 7, 13 ⇒ διάμεσος 7
5, 7, 13, 12 ⇒ 5, 7, 12, 13 ⇒ διάμεσος 9.5
5, 7, 13, 12, 2 ⇒ 2, 5, 7, 12, 13 ⇒ διάμεσος 7

```
1  #include <chrono>
2  #include <iomanip>
3  #include <iostream>
4  #include <queue>
5  #include <random>
6
7  using namespace std;
8  using namespace std::chrono;
9
10 double medians(int a[], int N) {
11   priority_queue<int, std::vector<int>, std::less<int>> pq1;
12   priority_queue<int, std::vector<int>, std::greater<int>> pq2;
```

```
13    int first = a[0];
14    int second = a[1];
15    if (first < second) {
16       pq1.push(first);
17       pq2.push(second);
18    } else {
19       pq2.push(first);
20       pq1.push(second);
21    }
22    double sum = first + (first + second) / 2.0;
23    for (int i = 2; i < N; i++) {
24       int x = a[i];
25       if (x <= pq1.top())
26          pq1.push(x);
27       else
28          pq2.push(x);
29       if (pq1.size() < pq2.size()) {
30          pq1.push(pq2.top());
31          pq2.pop();
32       }
33       double median;
34       if (pq1.size() == pq2.size())
35          median = (pq1.top() + pq2.top()) / 2.0;
36       else
37          median = pq1.top();
38       sum += median;
39    }
40    return sum;
41 }
42
43 int main(int argc, char **argv) {
44    high_resolution_clock::time_point t1, t2;
45    t1 = high_resolution_clock::now();
46    mt19937 mt(1940);
47    uniform_int_distribution<int> uni(0, 200000);
48    int N = 500000;
49    int *a = new int[N];
50    for (int i = 0; i < N; i++)
51       a[i] = uni(mt);
52    double sum = medians(a, N);
53    delete[] a;
54    t2 = high_resolution_clock::now();
55    duration<double, std::milli> duration = t2 − t1;
56    cout.precision(2);
57    cout << ”Moving medians sum = ” << std::fixed << sum << ” elapsed time ”
58         << duration.count() << ”ms” << endl;
59 }
```

Κώδικας 6.8: Υπολογισμός διαμέσου σε μια ροή τιμών (lab06_ex3.cpp)

```
1 Moving medians sum = 54441518145.50 elapsed time 132.52ms
```

6.7 Ασκήσεις

1. Να υλοποιηθεί ο σωρός μεγίστων που παρουσιάστηκε στον κώδικα 1 ως κλάση. Προσθέστε εξαιρέσεις έτσι ώστε να χειρίζονται περιπτώσεις όπως όταν ο σωρός είναι άδειος και ζητείται εξαγωγή της μεγαλύτερης τιμής ή όταν ο σωρός είναι γεμάτος και επιχειρείται εισαγωγή νέας τιμής.

2. Να γραφεί συνάρτηση που να δέχεται ως παράμετρο έναν πίνακα ακεραίων και έναν ακέραιο αριθμό κ και να επιστρέφει το κ-οστό μεγαλύτερο στοιχείο του πίνακα.

Βιβλιογραφία

[1] NIST, heapsort, https://xlinux.nist.gov/dads/HTML/heapSort.html

[2] PROGRAMIZ, Heap Sort Algorithm, https://www.programiz.com/dsa/heap-sort

[3] Geeks for Geeks, Priority Queue in C++ Standard Template Library (STL), http://www.geeksforgeeks.org/priority-queue-in-cpp-stl/

[4] Cppreference.com, std::priority_queue, http://en.cppreference.com/w/cpp/container/priority_queue