

Δομές Δεδομένων και Αλγόριθμοι - Εργαστήριο 2

Εισαγωγή στα templates, στην STL και στα lambdas - σύντομη παρουσίαση του Test Driven Development

Πανεπιστήμιο Ιωαννίνων, Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Χρήστος Γκόγκος - Αναπληρωτής Καθηγητής

1 Εισαγωγή

Στο εργαστήριο αυτό παρουσιάζεται ο μηχανισμός των templates, τα lambdas και οι βασικές δυνατότητες της βιβλιοθήκης STL (Standard Template Library) της C++. Τα templates επιτρέπουν την κατασκευή γενικού κώδικα επιτρέποντας την αποτύπωση της λογικής μιας συνάρτησης ανεξάρτητα από τον τύπο των ορισμάτων που δέχεται. Από την άλλη μεριά, η βιβλιοθήκη STL, στην οποία γίνεται εκτεταμένη χρήση των templates παρέχει στον προγραμματιστή έτοιμη λειτουργικότητα για πολλές ενέργειες που συχνά συναντώνται κατά την ανάπτυξη εφαρμογών. Επιπλέον υλικό για την STL βρίσκεται στις αναφορές [1], [2], [3], [4].

2 Templates

Τα templates (πρότυπα) είναι ένας μηχανισμός της C++ ο οποίος μπορεί να διευκολύνει τον προγραμματισμό. Η γλώσσα C++ είναι statically typed το οποίο σημαίνει ότι οι τύποι δεδομένων των μεταβλητών και σταθερών ελέγχονται κατά τη μεταγλώττιση. Το γεγονός αυτό μπορεί να οδηγήσει στην ανάγκη υλοποίησης διαφορετικών εκδόσεων μιας συνάρτησης έτσι ώστε να υποστηριχθεί η ίδια λογική για διαφορετικούς τύπους δεδομένων. Για παράδειγμα, η εύρεση της ελάχιστης τιμής ανάμεσα σε τρεις τιμές θα έπρεπε να υλοποιηθεί με δύο συναρτήσεις έτσι ώστε να υποστηρίξει τόσο τους ακεραίους όσο και τους πραγματικούς αριθμούς, όπως φαίνεται στον κώδικα που ακολουθεί.

```
1 #include <iostream>
2 using namespace std;
3
4 int min(int a, int b, int c) {
5     int m = a;
6     if (b < m)
7         m = b;
8     if (c < m)
9         m = c;
10    return m;
11 }
12
13 double min(double a, double b, double c) {
14     double m = a;
15     if (b < m)
16         m = b;
17     if (c < m)
18         m = c;
19    return m;
20 }
21
22 int main(int argc, char *argv[]) {
23     cout << "The minimum among 3 integer numbers is " << min
24         (5, 10, 7) << endl;
25     cout << "The minimum among 3 real numbers is " << min(3.1,
26         0.7, 2.5) << endl;
27 }
```

Κώδικας 1: Επανάληψη λογικής κώδικα (minmax.cpp)

1 The minimum among 3 integer numbers is 5
2 The minimum among 3 real numbers is 0.7

Με τη χρήση των templates μπορεί να γραφεί κώδικας που να υποστηρίζει ταυτόχρονα πολλούς τύπους δεδομένων. Ειδικότερα, χρησιμοποιείται, η δεσμευμένη λέξη template και εντός των συμβόλων < και > τοποθετείται η λίστα των παραμέτρων του template. Ο μεταγλωττιστής αναλαμβάνει να δημιουργήσει όλες τις απαιτούμενες παραλλαγές των συναρτήσεων που θα χρειαστούν στον κώδικα που μεταγλωττίζει.

```

1 #include <iostream>
2 using namespace std;
3
4 template <typename T> T min(T a, T b, T c) {
5     T m = a;
6     if (b < m)
7         m = b;
8     if (c < m)
9         m = c;
10    return m;
11 }
12
13 int main(int argc, char *argv[]) {
14     cout << "The minimum among 3 integer numbers is " << min
15         (5, 10, 7) << endl;
16     cout << "The minimum among 3 real numbers is " << min(3.1,
17         0.7, 2.5) << endl;
18 }

```

Κώδικας 2: Χρήση template για αποφυγή επανάληψης λογικής κώδικα (minmaxt.cpp)

```

1 The minimum among 3 integer numbers is 5
2 The minimum among 3 real numbers is 0.7

```

θα πρέπει να σημειωθεί ότι στη θέση της δεσμευμένης λέξης typename μπορεί εναλλακτικά να χρησιμοποιηθεί η δεσμευμένη λέξη class.

3 Η βιβλιοθήκη STL

Η βιβλιοθήκη STL (Standard Template Library) της C++ προσφέρει έτοιμη λειτουργικότητα για πολλά θέματα τα οποία ανακύπτουν συχνά στον προγραμματισμό εφαρμογών. Πρόκειται για μια generic βιβλιοθήκη, δηλαδή κάνει εκτεταμένη χρήση των templates. Βασικά τμήματα της STL είναι οι containers (υποδοχείς), οι iterators (επαναλήπτες) και οι αλγόριθμοι.

3.1 Containers

Η STL υποστηρίζει έναν αριθμό από containers στους οποίους μπορούν να αποθηκευτούν δεδομένα. Ένα από τα containers είναι το vector (διάνυσμα). Στον ακόλουθο κώδικα φαίνεται πως η χρήση του vector διευκολύνει τον προγραμματισμό καθώς δεν απαιτούνται εντολές διαχείρισης μνήμης ενώ η δομή είναι δυναμική, δηλαδή το μέγεθος της μπορεί να μεταβάλλεται κατά τη διάρκεια εκτέλεσης του προγράμματος.

```

1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main(int argc, char *argv[]) {
7     int x;
8     cout << "Enter the size of the vector: ";
9     cin >> x;
10    vector<int> v(x);
11    for (int i = 0; i < x; i++)
12        v[i] = i;
13    v.push_back(99);
14    for (int i = 0; i < v.size(); i++)
15        cout << v[i] << " ";
16 }

```

Κώδικας 3: Παράδειγμα με προσθήκη στοιχείων σε vector (container1.cpp)

```

1 Enter size of vector: 10
2 0 1 2 3 4 5 6 7 8 9 99

```

Ένα container τύπου vector μπορεί να λάβει τιμές με πολλούς τρόπους. Στον ακόλουθο κώδικα παρουσιάζονται έξι διαφορετικοί τρόποι με τους οποίους μπορεί να γίνει αυτό.

```

1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 void print_vector(const string &name, const vector<int> &v) {
7     cout << name << ": ";
8     for (int i = 0; i < v.size(); i++)
9         cout << v[i] << " ";
10    cout << endl;
11 }
12
13 int main(int argc, char *argv[]) {
14     vector<int> v1;
15     v1.push_back(5);
16     v1.push_back(16);
17     print_vector("v1", v1);
18
19     vector<int> v2 = {16, 3, 6, 1, 9, 10};
20     print_vector("v2", v2);
21
22     vector<int> v3{5, 2, 10, 1, 8};
23     print_vector("v3", v3);
24
25     vector<int> v4(5, 10);
26     print_vector("v4", v4);
27
28     vector<int> v5(v2);
29     print_vector("v5", v5);
30
31     vector<int> v6(v2.begin() + 1, v2.end() - 1);
32     print_vector("v6", v6);
33 }

```

Κώδικας 4: Αρχικοποίηση vectors (container2.cpp)

```

1 v1: 5 16
2 v2: 16 3 6 1 9 10
3 v3: 5 2 10 1 8
4 v4: 10 10 10 10 10
5 v5: 16 3 6 1 9 10
6 v6: 3 6 1 9

```

Τα containers χωρίζονται σε σειριακά (sequence containers) και συσχετιστικά (associative containers). Τα σειριακά containers είναι συλλογές ομοειδών στοιχείων στις οποίες κάθε στοιχείο έχει συγκεκριμένη θέση μέσω της οποίας μπορούμε να αναφερθούμε σε αυτό. Τα σειριακά containers είναι τα εξής:

- array (πίνακας)
- deque (συρά με δύο άκρα)
- forward_list (λίστα διανυόμενη προς τα εμπρός)
- list (λίστα)
- vector (διάνυσμα)

Τα συσχετιστικά containers παρουσιάζουν το πλεονέκτημα της γρήγορης προσπέλασης. Συσχετιστικά containers της STL είναι τα εξής:

- map (λεξικό)
- unordered_map (λεξικό χωρίς σειρά)
- multimap (πολλαπλό λεξικό)
- unordered_multimap. (πολλαπλό λεξικό χωρίς σειρά)
- set (σύνολο)
- unordered_set (σύνολο χωρίς σειρά)
- multiset (πολλαπλό σύνολο)
- unordered_multiset (πολλαπλό σύνολο χωρίς σειρά)

Στη συνέχεια παρουσιάζεται ένα παράδειγμα χρήσης του συσχετιστικού container map. Δημιουργείται ένας τηλεφωνικός κατάλογος που περιέχει πληροφορίες της μορφής όνομα - τηλέφωνο και ο οποίος δίνει τη δυνατότητα να αναζητηθεί ένα τηλέφωνο με βάση ένα όνομα.

```

1 #include <iostream>
2 #include <map>
3 #include <string>
4 using namespace std;
5
6 int main(int argc, char *argv[]) {
7     map<string, string> phone_book;
8     phone_book.insert(make_pair("nikos", "1234567890"));

```

```

9  phone_book.insert(make_pair("maria", "2345678901"));
10 phone_book.insert(make_pair("petros", "3456789012"));
11 phone_book.insert(make_pair("kostas", "4567890123"));
12
13 string name;
14 cout << "Enter name: ";
15 cin >> name;

```

```

16 if (phone_book.find(name) == phone_book.end())
17     cout << "No such name found ";
18 else
19     cout << "The phone is " << phone_book[name] << endl;
20 }

```

Κώδικας 5: Παράδειγμα με map (container3.cpp)

```

1 Enter name: nikos
2 The phone is 1234567890

```

3.2 Iterators

Οι iterators αποτελούν γενικεύσεις των δεικτών και επιτρέπουν την πλοήγηση στα στοιχεία ενός container με τέτοιο τρόπο έτσι ώστε να μπορούν να χρησιμοποιηθούν οι ίδιοι αλγόριθμοι σε περισσότερα του ενός containers. Στον ακόλουθο κώδικα παρουσιάζεται το πέρασμα από τα στοιχεία ενός vector με τέσσερις διαφορετικούς τρόπους. Καθώς το container είναι τύπου vector παρουσιάζεται αρχικά το πέρασμα από τις τιμές του με τη χρήση δεικτοδότησης τύπου πίνακα. Στη συνέχεια χρησιμοποιείται η πρόσβαση στα στοιχεία του container μέσω του range for. Ακολούθως, χρησιμοποιείται ένας iterator για πέρασμα από την αρχή προς το τέλος και ένας reverse_iterator για πέρασμα από το τέλος προς την αρχή.

```

1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main(int argc, char **argv)
7 {
8     vector<int> v = {23, 13, 31, 17, 56};
9     cout << "iteration using index: ";
10    for (int i = 0; i < v.size(); i++)
11        cout << v[i] << " ";
12    cout << endl;
13
14    cout << "iteration using ranged based for: ";
15    for (int x : v)
16        cout << x << " ";
17    cout << endl;

```

```

18
19    cout << "forward iteration using iterator: ";
20    vector<int>::iterator iter;
21    for (iter = v.begin(); iter != v.end(); iter++)
22        cout << *iter << " ";
23    cout << endl;
24
25    cout << "backward iteration using iterator: ";
26    vector<int>::reverse_iterator riter;
27    for (riter = v.rbegin(); riter != v.rend(); riter++)
28        cout << *riter << " ";
29    cout << endl;
30 }

```

Κώδικας 6: Τέσσερις διαφορετικοί τρόποι προσπέλασης των στοιχείων ενός vector (iterator1.cpp)

```

1 iteration using index: 23 13 31 17 56
2 iteration using ranged based for: 23 13 31 17 56
3 forward iteration using iterator: 23 13 31 17 56
4 backward iteration using iterator: 56 17 31 13 23

```

Ακολουθεί κώδικας στον οποίο παρουσιάζεται το πέρασμα από όλα τα στοιχεία ενός map με τρεις διαφορετικούς τρόπους. Ο πρώτος τρόπος χρησιμοποιεί range for. Ο δεύτερος έναν iterator και ο τρίτος έναν reverse iterator.

```

1 #include <iostream>
2 #include <map>
3
4 using namespace std;
5 int main(int argc, char *argv[]) {
6     map<string, int> cities_population_2011 = {{"arta", 21895},

```

```

{"ioannina",
 65574},
{"preveza",
 19042},
{"igoumenitsa",
 9145}};

```

```

10 cout << "Cities of Epirus using range for:" << endl;
11 for (auto kv : cities_population_2011)
12     cout << kv.first << " " << kv.second << endl;
13
14 cout << "Cities of Epirus using iterator:" << endl;
15 for (map<string, int>::iterator iter = cities_population_2011.
16     begin();
17     iter != cities_population_2011.end(); iter++)
18     cout << iter->first << " " << iter->second << endl;

```

```

19 cout << "Cities of Epirus using reverse iterator:" << endl;
20 for (map<string, int>::reverse_iterator iter =
21     cities_population_2011.rbegin();
22     iter != cities_population_2011.rend(); iter++)
23     cout << iter->first << " " << iter->second << endl;
24 }

```

Κώδικας 7: Τρεις διαφορετικοί τρόποι προσπέλασης των στοιχείων ενός map (iterator2.cpp)

```

1 Cities of Epirus using range for:
2 arta 21895
3 igoumenitsa 9145
4 ioannina 65574
5 preveza 19042
6 Cities of Epirus using iterator:
7 arta 21895

```

```

8 igoumenitsa 9145
9 ioannina 65574
10 preveza 19042
11 Cities of Epirus using reverse iterator:
12 preveza 19042
13 ioannina 65574
14 igoumenitsa 9145
15 arta 21895

```

3.3 Αλγόριθμοι

Η STL διαθέτει πληθώρα αλγορίθμων που μπορούν να εφαρμοστούν σε διάφορα προβλήματα. Για παράδειγμα, προκειμένου να ταξινομηθούν δεδομένα μπορεί να χρησιμοποιηθεί η συνάρτηση sort της STL η οποία υλοποιεί τον αλγόριθμο Introspective Sort. Στον ακόλουθο κώδικα πραγματοποιείται η ταξινόμηση αρχικά ενός στατικού πίνακα και στη συνέχεια εφόσον πρώτα οι τιμές του πίνακα μεταφερθούν σε ένα vector και ανακατευτούν τυχαία, πρώτα ταξινομούνται σε αύξουσα και μετά σε φθίνουσα σειρά.

```

1 #include <algorithm>
2 #include <iostream>
3 #include <random>
4 #include <vector>
5
6 using namespace std;
7
8 int main(int argc, char **argv) {
9     cout << "### STL Sort Example ###" << endl;
10    int a[] = {45, 32, 16, 11, 7, 18, 21, 16, 11, 15};
11    cout << "BEFORE (static array example): ";
12    for (int i = 0; i < 10; i++)
13        cout << a[i] << " ";
14    cout << endl;
15    sort(a, a + 10);
16    cout << "AFTER: ";
17    for (int i = 0; i < 10; i++)
18        cout << a[i] << " ";
19    cout << endl;
20
21    cout << "BEFORE (vector example 1): ";
22    vector<int> va(a, a + 10);
23    auto rng = default_random_engine{};
24    shuffle(va.begin(), va.end(), rng);
25
26    for (auto it = va.begin(); it < va.end(); it++)
27        cout << *it << " ";
28    cout << endl;
29    sort(va.begin(), va.end());
30    cout << "AFTER: ";
31    for (auto it = va.begin(); it < va.end(); it++)
32        cout << *it << " ";
33    cout << endl;
34
35    cout << "BEFORE (vector example 2): ";
36    shuffle(va.begin(), va.end(), rng);
37    for (auto it = va.begin(); it < va.end(); it++)
38        cout << *it << " ";
39    cout << endl;
40    // descending
41    sort(va.begin(), va.end(), greater<int>());
42    cout << "AFTER: ";
43    for (auto it = va.begin(); it < va.end(); it++)
44        cout << *it << " ";
45    cout << endl;
46    return 0;
}

```

Κώδικας 8: Ταξινόμηση με τη συνάρτηση sort της STL (sort1.cpp)

```

1 ### STL Sort Example ###
2 BEFORE (static array example): 45 32 16 11 7 18 21 16 11 15
3 AFTER: 7 11 11 15 16 16 18 21 32 45
4 BEFORE (vector example 1): 21 18 16 16 11 15 45 11 7 32
5 AFTER: 7 11 11 15 16 16 18 21 32 45

```

6 BEFORE (vector example 2): 45 11 15 11 21 7 32 16 16 18
 7 AFTER: 45 32 21 18 16 16 15 11 11 7

Στον παραπάνω κώδικα έγινε χρήση της δεσμευμένης λέξης `auto` στη δήλωση μεταβλητών. Η λέξη `auto` μπορεί να χρησιμοποιηθεί στη θέση ενός τύπου δεδομένων όταν γίνεται ταυτόχρονα δήλωση και ανάθεση τιμής σε μια μεταβλητή. Σε αυτή την περίπτωση ο μεταγλωττιστής της C++ είναι σε θέση να αναγνωρίσει τον πραγματικό τύπο της μεταβλητής από την τιμή που της εκχωρείται.

Η συνάρτηση `sort()` εφαρμόζεται σε `sequence containers` πλην των `list` και `forward_list` στα οποία δεν μπορεί να γίνει απευθείας πρόσβαση σε στοιχεία τους χρησιμοποιώντας ακεραίους αριθμούς για τον προσδιορισμό της θέσης τους. Ειδικά για αυτά τα `containers` υπάρχει η συνάρτηση μέλος `sort` που επιτρέπει την ταξινόμησή τους. Στον ακόλουθο κώδικα δημιουργείται μια λίστα με αντικείμενα ορθογωνίων παραλληλογράμμων τα οποία ταξινομούνται με βάση το εμβαδόν τους σε αύξουσα σειρά. Για την ταξινόμηση των αντικειμένων παρουσιάζονται τρεις διαφορετικοί τρόποι που παράγουν το ίδιο αποτέλεσμα.

```

1 #include <iostream>
2 #include <list>
3
4 using namespace std;
5
6 class Rectangle {
7 public:
8     Rectangle(double w, double h) : width(w), height(h){};
9     double area() const { return width * height; } // must be const
10    void print_info() {
11        cout << "Width:" << width << " Height:" << height << "
12            Area "
13            << this->area() << endl;
14    }
15    bool operator<(const Rectangle &other) const {
16        return this->area() < other.area();
17    }
18 private:
19     double width;
20     double height;
21 };
22
23 int main(int argc, char *argv[]) {
24     list<Rectangle> rectangles;
25     rectangles.push_back(Rectangle(5, 6));
26     rectangles.push_back(Rectangle(3, 3));
27     rectangles.push_back(Rectangle(5, 2));
28     rectangles.push_back(Rectangle(6, 1));
29
30     rectangles.sort();
31     for (auto r : rectangles)
32         r.print_info();
33 }
```

Κώδικας 9: Ταξινόμηση λίστας με αντικείμενα - α' τρόπος (sort2.cpp)

Θα πρέπει να σημειωθεί ότι η δεσμευμένη λέξη `this` στον κώδικα μιας κλάσης αναφέρεται σε έναν δείκτη προς το ίδιο το αντικείμενο για το οποίο καλούνται οι συναρτήσεις μέλη.

```

1 #include <iostream>
2 #include <list>
3
4 using namespace std;
5
6 class Rectangle {
7 public:
8     Rectangle(double w, double h) : width(w), height(h){};
9     double area() const { return width * height; }
10    void print_info() {
11        cout << "Width:" << width << " Height:" << height << "
12            Area "
13            << this->area() << endl;
14    }
15 private:
16     double width;
17     double height;
18 };
19
20 bool operator<(const Rectangle &r1, const Rectangle &r2) {
21     return r1.area() < r2.area();
22 }
23
24 int main(int argc, char *argv[]) {
25     list<Rectangle> rectangles = {{5,6},{3,3},{5,2},{6,1}};
26
27     rectangles.sort();
28     for (auto r : rectangles)
29         r.print_info();
30 }
```

Κώδικας 10: Ταξινόμηση λίστας με αντικείμενα - β' τρόπος (sort3.cpp)

```

1 #include <iostream>
2 #include <list>
3
4 using namespace std;
5
6 class Rectangle {
7 public:
8     Rectangle(double w, double h) : width(w), height(h){};
9     double area() const { return width * height; }
10    void print_info() {
11        cout << "Width:" << width << " Height:" << height << "
12            Area "
13            << this->area() << endl;
14    }
15 private:
16    double width;
17    double height;
18 };
19
20 int main(int argc, char *argv[]) {
21     list<Rectangle> rectangles = {{5,6},{3,3},{5,2},{6,1}};
22
23     struct CompareRectangle {
24         bool operator()(Rectangle lhs, Rectangle rhs) {
25             return lhs.area() < rhs.area();
26         }
27     };
28     rectangles.sort(CompareRectangle());
29
30     for (auto r : rectangles)
31         r.print_info();
32 }

```

Κώδικας 11: Ταξινόμηση λίστας με αντικείμενα - γ' τρόπος (sort4.cpp)

```

1 Width:6 Height:1 Area 6
2 Width:3 Height:3 Area 9
3 Width:5 Height:2 Area 10
4 Width:5 Height:6 Area 30

```

Αν αντί για αντικείμενα το container περιέχει εγγραφές τύπου struct Rectangle τότε ένας τρόπος με το οποίο μπορεί να επιτευχθεί η ταξινόμηση των εγγραφών ορθογωνίων σε αύξουσα σειρά εμβαδού είναι ο ακόλουθος.

```

1 #include <iostream>
2 #include <list>
3
4 using namespace std;
5
6 struct Rectangle {
7     double width;
8     double height;
9     bool operator<(const Rectangle &other) const {
10         return width * height < other.width * other.height;
11     }
12 };
13
14 int main(int argc, char *argv[]) {
15     list<Rectangle> rectangles = {{5, 6}, {3, 3}, {5, 2}, {6, 1}};
16
17     rectangles.sort();
18     for (auto r : rectangles)
19         cout << "Width:" << r.width << " Height:" << r.height
20             << " Area: " << r.width * r.height << endl;
21 }

```

Κώδικας 12: Ταξινόμηση λίστας με εγγραφές (sort5.cpp)

```

1 Width:6 Height:1 Area: 6
2 Width:3 Height:3 Area: 9
3 Width:5 Height:2 Area: 10
4 Width:5 Height:6 Area: 30

```

Αντίστοιχα, για να γίνει αναζήτηση ενός στοιχείου σε έναν ταξινομημένο πίνακα μπορούν να χρησιμοποιηθούν συναρτήσεις της STL όπως η συνάρτηση `binary_search`, η συνάρτηση `lower_bound` και η συνάρτηση `upper_bound`. Η `binary_search` επιστρέφει `true` αν το στοιχείο υπάρχει στον πίνακα αλλιώς επιστρέφει `false`. Οι `lower_bound` και `upper_bound` εντοπίζουν την χαμηλότερη και την υψηλότερη θέση στην οποία μπορεί να εισαχθεί το στοιχείο χωρίς να διαταραχθεί η ταξινομημένη σειρά των υπόλοιπων στοιχείων. Ένα παράδειγμα χρήσης των συναρτήσεων αυτών δίνεται στον ακόλουθο κώδικα.

```

1 #include <algorithm>
2 #include <iostream>
3
4 using namespace std;
5
6 int main(int argc, char **argv) {
7     int a[] = {45, 11, 16, 11, 7, 18, 21, 16, 11, 15};
8     int N = sizeof(a) / sizeof(int);
9     cout << "The size of the array is " << N << endl;

```



```

10 int key;
11 sort(a, a + N);
12 for (int i = 0; i < N; i++)
13     cout << a[i] << " ";
14 cout << endl;
15 cout << "Enter a value to be searched for: ";
16 cin >> key;
17 if (binary_search(a, a + N, key))
18     cout << "Found using binary_search" << endl;
19 else
20     cout << "Not found (binary_search)" << endl;
21
22 auto it1 = lower_bound(a, a + N, key);

```

```

23 auto it2 = upper_bound(a, a + N, key);
24 if (*it1 == key) {
25     cout << "Found at positions " << it1 - a << " up to " << it2
26         - a - 1
27         << endl;
28 } else
29     cout << "Not found (lower_bound and upper_bound)" <<
        endl;

```

Κώδικας 13: Αναζήτηση σε ταξινομημένο πίνακα (search1.cpp)

```

1 The size of the array is 10
2 7 11 11 11 15 16 16 18 21 45
3 Enter a value to be searched for: 11
4 Found using binary_search
5 Found at positions 1 up to 3

```

```

1 #include <algorithm>
2 #include <iostream>
3 #include <vector>
4
5 using namespace std;
6
7 int main(int argc, char **argv) {
8     vector<int> a = {45, 11, 16, 11, 7, 18, 21, 16, 11, 15};
9     cout << "The size of the vector is " << a.size() << endl;
10    int key;
11    sort(a.begin(), a.end());
12    for (int i = 0; i < a.size(); i++)
13        cout << a[i] << " ";
14    cout << endl;
15    cout << "Enter a value to be searched for: ";
16    cin >> key;

```

```

17 if (binary_search(a.begin(), a.end(), key))
18     cout << "Found using binary_search" << endl;
19 else
20     cout << "Not found (binary_search)" << endl;
21
22 auto it1 = lower_bound(a.begin(), a.end(), key);
23 auto it2 = upper_bound(a.begin(), a.end(), key);
24 if (*it1 == key) {
25     cout << "Found at positions " << it1 - a.begin() << " up to "
26         << it2 - a.begin() - 1 << endl;
27 } else
28     cout << "Not found (lower_bound and upper_bound)" <<
        endl;
29 }

```

Κώδικας 14: Αναζήτηση σε ταξινομημένο διάνυσμα (search2.cpp)

```

1 The size of the vector is 10
2 7 11 11 11 15 16 16 18 21 45
3 Enter a value to be searched for: 16
4 Found using binary_search
5 Found at positions 5 up to 6

```

4 Lambdas

Η δυνατότητα lambdas έχει ενσωματωθεί στη C++ από την έκδοση 11 και μετά και επιτρέπει τη συγγραφή ανώνυμων συναρτήσεων στο σημείο που χρειάζονται, διευκολύνοντας με αυτό τον τρόπο τη συγγραφή προγραμμάτων. Ο όρος lambda ιστορικά έχει προέλθει από τη συναρτησιακή γλώσσα προγραμματισμού LISP. Μια lambda έκφραση στη C++ έχει την ακόλουθη μορφή:

```

1 [capture list] (parameter list) -> return type
2 {
3     function body
4 }

```


Συνήθως το τμήμα `-> return type` παραλείπεται καθώς ο μεταγλωττιστής είναι σε θέση να εκτιμήσει ο ίδιος τον τύπο επιστροφής της συνάρτησης. Στον επόμενο κώδικα παρουσιάζεται μια απλή συνάρτηση lambda η οποία δέχεται δύο `double` παραμέτρους και επιστρέφει το γινόμενό τους.

```
1 cout << "Area = " << [](double x, double y){return x * y;}(3.0, 4.5) << endl;
```

Μια lambda συνάρτηση μπορεί να αποθηκευτεί σε μια μεταβλητή και στη συνέχεια να κληθεί μέσω της μεταβλητής αυτής όπως στο ακόλουθο παράδειγμα:

```
1 auto area = [](double x, double y)
2 {
3     return x * y;
4 };
5 cout << "Area = " << area(3.0, 4.5) << endl;
```

Στη συνέχεια παρουσιάζονται διάφορα παραδείγματα lambda συναρτήσεων καθώς και παραδείγματα στα οποία χρησιμοποιούνται lambda συναρτήσεις σε συνδυασμό με τις συναρτήσεις της STL: `find_if`, `count_if`, `sort` (σε `list` και σε `vector`) και `for_each`.

```
1 #include <algorithm>
2 #include <iostream>
3 #include <list>
4 #include <vector>
5
6 using namespace std;
7
8 int main() {
9
10     cout << "Example1: call lambda function" << endl;
11     cout << "Area = " << [](double x, double y) { return x * y; }
12         (3.0, 4.5)
13         << endl;
14
15     cout << "Example2: assign lambda function to a variable" <<
16         endl;
17     auto area = [](double x, double y) { return x * y; };
18     cout << "Area = " << area(3.0, 4.5) << endl;
19     cout << "Area = " << area(7.0, 5.5) << endl;
20
21     vector<int> v{5, 1, 3, 2, 8, 7, 4, 5};
22     // find_if
23     cout << "Example3: find the first even number in the vector"
24         << endl;
25     vector<int>::iterator iter =
26         find_if(v.begin(), v.end(), [](int x) { return x % 2 == 0; });
27     cout << *iter << endl;
28
29     // count_if
30     cout << "Example4: count the number of even numbers in the
31         vector" << endl;
32     int c = count_if(v.begin(), v.end(), [](int x) { return x % 2 ==
33         0; });
34     cout << c << endl;
35
36     // sort
37
38     cout << "Example5: sort list of rectangles by area (ascending)"
39         << endl;
40     struct Rectangle {
41         double width;
42         double height;
43     };
44     list<Rectangle> rectangles_list = {{5, 6}, {3, 3}, {5, 2}, {6,
45         1}};
46     rectangles_list.sort([](Rectangle &r1, Rectangle &r2) {
47         return r1.width * r1.height < r2.width * r2.height;
48     });
49     for (Rectangle r : rectangles_list)
50         cout << "Width:" << r.width << " Height:" << r.height
51             << " Area: " << r.width * r.height << endl;
52
53     cout << "Example6: sort vector of rectangles by area (
54         descending)" << endl;
55     vector<Rectangle> rectangles_vector = {{5, 6}, {3, 3}, {5, 2},
56         {6, 1}};
57     sort(rectangles_vector.begin(), rectangles_vector.end(),
58         [](Rectangle &r1, Rectangle &r2) {
59             return r1.width * r1.height > r2.width * r2.height;
60         });
61     for (Rectangle r : rectangles_vector)
62         cout << "Width:" << r.width << " Height:" << r.height
63             << " Area: " << r.width * r.height << endl;
64
65     // for_each
66     cout << "Example7: for_each" << endl;
67     for_each(v.begin(), v.end(), [](int i) { cout << i << " "; });
68     cout << endl;
69     for_each(v.begin(), v.end(), [](int i) { cout << i * i << " "; });
70     cout << endl;
71 }
```

Κώδικας 15: Παραδείγματα με lambdas (lambda1.cpp)

```
1 Example1: call lambda function
2 Area = 13.5
3 Example2: assign lambda function to variable
```

```
4 Area = 13.5
5 Area = 38.5
6 Example3: find the first even number in the vector
7 2
```

```

8 Example4: count the number of even numbers in the vector
9 3
10 Example5: sort list of rectangles by area (ascending)
11 Width:6 Height:1 Area: 6
12 Width:3 Height:3 Area: 9
13 Width:5 Height:2 Area: 10
14 Width:5 Height:6 Area: 30
15 Example6: sort vector of rectangles by area (descending)
16 Width:5 Height:6 Area: 30
17 Width:5 Height:2 Area: 10
18 Width:3 Height:3 Area: 9
19 Width:6 Height:1 Area: 6
20 Example7: for_each
21 5 1 3 2 8 7 4 5
22 25 1 9 4 64 49 16 25

```

Μια lambda έκφραση μπορεί να έχει πρόσβαση σε μεταβλητές που βρίσκονται στην εμβέλεια που περικλείει την ίδια τη lambda έκφραση. Ειδικότερα, η πρόσβαση (capture) στις εξωτερικές μεταβλητές μπορεί να γίνει είτε με αναφορά (capture by reference), είτε με τιμή (capture by value) είτε να γίνει μικτή πρόσβαση (mixed capture). Το δε συντακτικό που χρησιμοποιείται για να υποδηλώσει το είδος της πρόσβασης είναι:

- `[]`: καμία πρόσβαση σε εξωτερικές της lambda συνάρτησης μεταβλητές
- `&`: πρόσβαση σε όλες τις εξωτερικές μεταβλητές με αναφορά
- `=`: πρόσβαση σε όλες τις εξωτερικές μεταβλητές με τιμή
- `[a, &b]`: πρόσβαση στην a με τιμή και πρόσβαση στη b με αναφορά

```

1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main(int argc, char *argv[]) {
7     vector<int> v1 {1, 2, 3, 4, 5, 6};
8     vector<int> v2(6, 1);
9
10    // capture by value
11    auto lambda1 = [=](int x) {
12        cout << "v1: ";
13        for (auto p = v1.begin(); p != v1.end(); p++)
14            if (*p != x)
15                cout << *p << " ";
16        cout << endl;
17    };
18
19    // capture by reference
20    auto lambda2 = [&](int x) {
21        for (auto p = v2.begin(); p != v2.end(); p++)
22            (*p) += x;
23    };
24
25    // mixed capture
26    auto lambda3 = [v1, &v2](int x) {
27        cout << "v1: ";
28        for (auto p = v1.begin(); p != v1.end(); p++)
29            if (*p != x)
30                cout << *p << " ";
31        cout << endl;
32        for (auto p = v2.begin(); p != v2.end(); p++)
33            (*p) += x;
34    };
35
36    cout << "Example1: capture by value (all external variables)"
37        << endl;
38    lambda1(1);
39    cout << "Example2: capture by reference (all external
40        variables)" << endl;
41    lambda2(2);
42    cout << "Example3: mixed capture (v1 by value, v2 by
43        reference)" << endl;
44    lambda3(1);
45
46    cout << "v1: ";
47    for (int x : v1)
48        cout << x << " ";
49    cout << endl;
50
51    cout << "v2: ";
52    for (int x : v2)
53        cout << x << " ";
54    cout << endl;
55 }

```

Κώδικας 16: Παραδείγματα με πρόσβαση σε εξωτερικές μεταβλητές σε lambdas (lambda2.cpp)

```

1 Example1: capture by value (all external variables)
2 v1: 2 3 4 5 6
3 Example2: capture by reference (all external variables)
4 Example3: mixed capture (v1 by value, v2 by reference)
5 v1: 2 3 4 5 6
6 v1: 1 2 3 4 5 6
7 v2: 4 4 4 4 4 4

```

5 Παραδείγματα

5.1 Παράδειγμα 1

Να γράψετε πρόγραμμα που να δημιουργεί πίνακα A με 1.000 τυχαίες ακέραιες τιμές στο διάστημα [1, 10.000] και πίνακα B με 100.000 τυχαίες ακέραιες τιμές στο ίδιο διάστημα τιμών. Η παραγωγή των τυχαίων τιμών να γίνει με τη γεννήτρια τυχαίων αριθμών mt19937 και με seed την τιμή 1821. Χρησιμοποιώντας τη συνάρτηση `binary_search` της STL να βρεθεί πόσες από τις τιμές του B υπάρχουν στον πίνακα A.

```

1 #include <algorithm>
2 #include <iostream>
3
4 using namespace std;
5
6 int main(int argc, char *argv[]) {
7     mt19937 mt(1821);
8     uniform_int_distribution<int> dist(1, 10000);
9     constexpr int N = 1000;
10    constexpr int M = 100000;
11    int a[N];
12    int b[M];
13    for (int i = 0; i < N; i++)
14        a[i] = dist(mt);
15    for (int i = 0; i < M; i++)
16        b[i] = dist(mt);
17    sort(a, a + N);
18    int c = 0;
19    for (int i = 0; i < M; i++)
20        if (binary_search(a, a + N, b[i]))
21            c++;
22    cout << "Result " << c << endl;
23    return 0;
24 }
```

Κώδικας 17: Λύση παραδείγματος 1 (lab02_ex1.cpp)

1 Result 9644

5.2 Παράδειγμα 2

Η συνάρτηση `accumulate()` της STL επιτρέπει τον υπολογισμό αθροισμάτων στα στοιχεία ενός container. Δημιουργήστε ένα vector με διάφορες ακέραιες τιμές της επιλογής σας και υπολογίστε το άθροισμα των τιμών με τη χρήση της συνάρτησης `accumulate`. Επαναλάβετε τη διαδικασία για ένα container τύπου array.

```

1 #include <array>
2 #include <iostream>
3 #include <numeric>
4 #include <vector>
5
6 using namespace std;
7
8 int main(int argc, char *argv[]) {
9     vector<int> v{5, 15, 20, 17, 11, 9};
10    int sum = accumulate(v.begin(), v.end(), 0);
11    cout << "Sum over vector using accumulate: " << sum << endl;
12
13    array<int, 6> a{5, 15, 20, 17, 11, 9};
14    sum = accumulate(a.begin(), a.end(), 0);
15    cout << "Sum over array using accumulate: " << sum << endl;
16 }
```

Κώδικας 18: Λύση παραδείγματος 2 (lab02_ex2.cpp)

1 Sum over vector using accumulate: 77
2 Sum over array using accumulate: 77

5.3 Παράδειγμα 3

Δημιουργήστε ένα vector που να περιέχει ονόματα. Χρησιμοποιώντας τη συνάρτηση `next_permutation()` εμφανίστε όλες τις διαφορετικές διατάξεις των ονομάτων που περιέχει το vector.

```

1 #include <algorithm>
2 #include <iostream>
3 #include <vector>
4
5 using namespace std;
6
```

```

7 int main(int argc, char *argv[]) {
8     vector<string> v{"petros", "anna", "nikos"};
9     sort(v.begin(), v.end());
10    do {
11        for (string x : v)
12            cout << x << " ";
13        cout << endl;
14    } while (next_permutation(v.begin(), v.end()));
15 }

```

Κώδικας 19: Λύση παραδείγματος 3 (lab02_ex3.cpp)

```

1 anna nikos petros
2 anna petros nikos
3 nikos anna petros
4 nikos petros anna
5 petros anna nikos
6 petros nikos anna

```

5.4 Παράδειγμα 4

Κατασκευάστε μια συνάρτηση που να επιστρέφει την απόσταση Hamming ανάμεσα σε δύο σειρές χαρακτήρων (η απόσταση Hamming είναι το πλήθος των χαρακτήρων που είναι διαφορετικοί στις ίδιες θέσεις ανάμεσα στις δύο σειρές). Δημιουργήστε ένα διάνυσμα με 100 τυχαίες σειρές μήκους 20 χαρακτήρων η κάθε μια χρησιμοποιώντας μόνο τους χαρακτήρες G,A,T,C. Εμφανίστε το πλήθος από τις σειρές για τις οποίες υπάρχει τουλάχιστον μια άλλη σειρά χαρακτήρων με απόσταση Hamming μικρότερη ή ίση του 10.

```

1 #include <algorithm>
2 #include <iostream>
3 #include <string>
4 #include <vector>
5
6 using namespace std;
7
8 int hamming(string x, string y) {
9     int c = 0;
10    int length = x.size() > y.size() ? x.size() : y.size();
11    for (int i = 0; i < length; i++)
12        if (x.at(i) != y.at(i))
13            c++;
14    return c;
15 }
16
17 int main(int argc, char *argv[]) {
18     constexpr int N = 100;
19     constexpr int L = 20;
20     mt19937 mt(1821);
21     uniform_int_distribution<int> dist(0, 3);
22     char gact[] = {'A', 'G', 'C', 'T'};
23     vector<string> sequences(N);
24     for (int i = 0; i < N; i++)
25         for (int j = 0; j < L; j++)
26             sequences[i] += gact[dist(mt)];
27
28     int c = 0;
29     for (int i = 0; i < N; i++) {
30         cout << "Checking sequence: " << sequences[i] << "..." << endl;
31         for (int j = 0; j < N; j++) {
32             if (i == j)
33                 continue;
34             int hd = hamming(sequences[i], sequences[j]);
35             cout << sequences[i] << " " << sequences[j]
36                 << " ==> hamming distance=" << hd << endl;
37             if (hd <= 10) {
38                 c++;
39                 break;
40             }
41         }
42         cout << endl;
43     }
44     cout << "Result=" << c << endl;
45 }

```

Κώδικας 20: Λύση παραδείγματος 4 (lab02_ex4.cpp)

```

1 Checking sequence: CGCCATCTAAGGACTCCCCA
2 CGCCATCTAAGGACTCCCCA CACATTCAAAGTGTGGGCCA ==> hamming distance=11
3 ...
4 CGCCATCTAAGGACTCCCCA CCCCATCTCGGCCACGCTG ==> hamming distance=9
5
6 Checking sequence: CACATTCAAAGTGTGGGCCA
7 CACATTCAAAGTGTGGGCCA CGCCATCTAAGGACTCCCCA ==> hamming distance=11
8 ...
9 CACATTCAAAGTGTGGGCCA CATATAACAACAGCATGCGA ==> hamming distance=9
10
11 ...
12

```

```
13 Checking sequence: TAGGTGCCATAAGAATCACT
14 TAGGTGCCATAAGAATCACT CGCCATCTAAGGACTCCCCA ==> hamming distance=16
15 ...
16 TAGGTGCCATAAGAATCACT AGCTGATTACGACAGCCTTC ==> hamming distance=16
17
18 Result=71
```

6 Ασκήσεις

1. Γράψτε ένα πρόγραμμα που να δέχεται τιμές από το χρήστη και για κάθε τιμή που θα δίνει ο χρήστης να εμφανίζει όλες τις τιμές που έχουν εισαχθεί μέχρι εκείνο το σημείο ταξινομημένες σε φθίνουσα σειρά.
2. Γράψτε ένα πρόγραμμα που να γεμίζει ένα διάνυσμα 1.000 θέσεων με τυχαίες πραγματικές τιμές στο διάστημα -100 έως και 100 διασφαλίζοντας ότι γειτονικές τιμές απέχουν το πολύ 10% η μια από την άλλη. Στη συνέχεια υπολογίστε την επτάδα συνεχόμενων τιμών με το μεγαλύτερο άθροισμα σε όλο το διάνυσμα.
3. Γράψτε ένα πρόγραμμα που να δέχεται τιμές από το χρήστη. Οι θετικές τιμές να εισάγονται σε ένα διάνυσμα n ενώ για κάθε αρνητική τιμή που εισάγεται να αναζητείται η απόλυτη τιμή της στο διάνυσμα n . Καθώς εισάγονται οι τιμές να εμφανίζονται στατιστικά για το πλήθος των τιμών που περιέχει το διάνυσμα, πόσες επιτυχίες και πόσες αποτυχίες αναζήτησης υπήρξαν.
4. Γράψτε ένα πρόγραμμα που να διαβάζει όλες τις λέξεις ενός αρχείου κειμένου και να εμφανίζει πόσες φορές υπάρχει η κάθε λέξη στο κείμενο σε αύξουσα σειρά συχνότητας. Χρησιμοποιήστε ως είσοδο το κείμενο του βιβλίου 1984 του George Orwell (<http://gutenberg.net.au/ebooks01/0100021.txt>).

Αναφορές

- [1] <http://www.geeksforgeeks.org/cpp-stl-tutorial/>.
- [2] <https://www.topcoder.com/community/data-science/data-science-tutorials/power-up-c-with-the-standard-template-library-part-1/>.
- [3] <https://www.topcoder.com/community/data-science/data-science-tutorials/power-up-c-with-the-standard-template-library-part-2/>.
- [4] <https://www.hackerearth.com/practice/notes/standard-template-library/>