Δομές Δεδομένων και Αλγόριθμοι - Εργαστήριο 2
Εισαγωγή στα templates, στην STL και στα lambdas - σύντομη παρουσίαση του Test Driven Development

Τ.Ε.Ι. Ηπείρου, Τμήμα Μηχανικών Πληροφορικής Τ.Ε.
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. Τέλος, γίνεται μια σύντομη αναφορά στην τεχνική ανάπτυξης προγραμμάτων TDD η οποία εξασφαλίζει σε κάποιο βαθμό την ανάπτυξη προγραμμάτων με ορθή λειτουργία εξαναγκάζοντας τους προγραμματιστές να ενσωματώσουν τη δημιουργία ελέγχων στον κώδικα που παράγουν καθημερινά. Επιπλέον υλικό για την STL βρίσκεται στις αναφορές [1], [2], [3], [4], [5].

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 υποστηρίζει 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(5, 10, 7) << endl;
24    cout << "The minimum among 3 real numbers is " << min(3.1, 0.7, 2.5) << endl;
25 }
```

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(5, 10, 7) << endl;
15    cout << "The minimum among 3 real numbers is " << min(3.1, 0.7, 2.5) << endl;
16 }
```

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
```

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
```

```
v5: 16 3 6 1 9 10
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 (ουρά 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:

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

```cpp
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main(int argc, char *argv[]) {
   map<string, string> phone_book;
   phone_book.insert(make_pair("nikos", "1234567890"));
   phone_book.insert(make_pair("maria", "2345678901"));
   phone_book.insert(make_pair("petros", "3456789012"));
   phone_book.insert(make_pair("kostas", "4567890123"));

   string name;
   cout << "Enter name: ";
   cin >> name;
   if (phone_book.find(name) == phone_book.end())
      cout << "No such name found ";
   else
      cout << "The phone is " << phone_book[name] << endl;
}
```

Κώδικας 5: Παράδειγμα με map (container3.cpp)

```
Enter name: nikos
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 πέντε 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.

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

using namespace std;

int main(int argc, char **argv)
{
   vector<int> v = {23, 13, 31, 17, 56};
   cout << "iteration using index: ";
   for (int i = 0; i < v.size(); i++)
      cout << v[i] << " ";
   cout << endl;

   cout << "iteration using ranged based for: ";
   for (int x : v)
      cout << x << " ";
   cout << endl;

   cout << "forward iteration using iterator: ";
   vector<int>::iterator iter;
   for (iter = v.begin(); iter != v.end(); iter++)
      cout << *iter << " ";
   cout << endl;

   cout << "backward iteration using iterator: ";
   vector<int>::reverse_iterator riter;
   for (riter = v.rbegin(); riter != v.rend(); riter++)
      cout << *riter << " ";
   cout << endl;
}
```

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

```
iteration using index: 23 13 31 17 56
iteration using ranged based for: 23 13 31 17 56
forward iteration using iterator: 23 13 31 17 56
backward iteration using iterator: 56 17 31 13 23
```

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.

```
#include <algorithm>
#include <iostream>
#include <random>
```

```cpp
#include <vector>

using namespace std;

int main(int argc, char **argv) {
   cout << "### STL Sort Example ###" << endl;
   int a[] = {45, 32, 16, 11, 7, 18, 21, 16, 11, 15};
   cout << "BEFORE (static array example): ";
   for (int i = 0; i < 10; i++)
      cout << a[i] << " ";
   cout << endl;
   sort(a, a + 10);
   cout << "AFTER: ";
   for (int i = 0; i < 10; i++)
      cout << a[i] << " ";
   cout << endl;

   cout << "BEFORE (vector example 1): ";
   vector<int> va(a, a + 10);
   auto rng = default_random_engine{};
   shuffle(va.begin(), va.end(), rng);
   for (auto it = va.begin(); it < va.end(); it++)
      cout << *it << " ";
   cout << endl;
   sort(va.begin(), va.end());
   cout << "AFTER: ";
   for (auto it = va.begin(); it < va.end(); it++)
      cout << *it << " ";
   cout << endl;

   cout << "BEFORE (vector example 2): ";
   shuffle(va.begin(), va.end(), rng);
   for (auto it = va.begin(); it < va.end(); it++)
      cout << *it << " ";
   cout << endl;
   // descending
   sort(va.begin(), va.end(), greater<int>());
   cout << "AFTER: ";
   for (auto it = va.begin(); it < va.end(); it++)
      cout << *it << " ";
   cout << endl;
   return 0;
}
```

Κώδικας 7: Ταξινόμηση με τη συνάρτηση sort της STL (sort1.cpp)

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

Η συνάρτηση sort() εφαρμόζεται σε sequence containers πλην των list και forward_list στα οποία δεν μπορεί να γίνει απευθείας πρόσβαση σε κάποιο στοιχείο με τη χρήση δείκτη. 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 ταξινόμησή ορθογωνίων παρουσιάζονται τρεις διαφορετικοί τρόποι που παράγουν το ίδιο αποτέλεσμα.

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

using namespace std;

class Rectangle {
public:
  Rectangle(double w, double h) : width(w), height(h){};
  double area() const { return width * height; } // must be const
  void print_info() {
    cout << "Width:" << width << " Height:" << height << " Area "
         << this->area() << endl;
  }
  bool operator<(const Rectangle &other) { return this->area() < other.area(); }

private:
  double width;
  double height;
};

int main(int argc, char *argv[]) {
  list<Rectangle> rectangles;
  rectangles.push_back(Rectangle(5, 6));
  rectangles.push_back(Rectangle(3, 3));
  rectangles.push_back(Rectangle(5, 2));
  rectangles.push_back(Rectangle(6, 1));

  rectangles.sort();
  for (auto r : rectangles)
    r.print_info();
}
```

Κώδικας 8: Ταξινόμηση λίστας με αντικείμενα - α' τρόπος (sort2.cpp)

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

using namespace std;

class Rectangle {
public:
  Rectangle(double w, double h) : width(w), height(h){};
  double area() const { return width * height; }
  void print_info() {
    cout << "Width:" << width << " Height:" << height << " Area "
         << this->area() << endl;
  }

private:
  double width;
  double height;
};

bool operator<(const Rectangle &r1, const Rectangle &r2) {
  return r1.area() < r2.area();
}

int main(int argc, char *argv[]) {
  list<Rectangle> rectangles = {{5,6},{3,3},{5,2},{6,1}};

  rectangles.sort();
```

```
28    for (auto r : rectangles)
29       r.print_info();
30 }
```

Κώδικας 9: 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 << " Area "
12            << this->area() << endl;
13    }
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 10: 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<(Rectangle other) {
10       return width * height < other.width * other.height;
11    }
12 };
13
```

```
int main(int argc, char *argv[]) {
   list<Rectangle> rectangles = {{5, 6}, {3, 3}, {5, 2}, {6, 1}};

   rectangles.sort();
   for (auto r : rectangles)
      cout << "Width:" << r.width << " Height:" << r.height
           << " Area: " << r.width * r.height << endl;
}
```

Κώδικας 11: Ταξινόμηση λίστας με εγγραφές (sort5.cpp)

Αντίστοιχα, για να γίνει αναζήτηση ενός στοιχείου σε έναν ταξινομημένο πίνακα μπορούν να χρησιμοποιηθούν διάφορες συναρτήσεις της STL όπως η συνάρτηση binary_search, ή η συνάρτηση upper_bound. Ένα παράδειγμα χρήσης των συναρτήσεων αυτών δίνεται στον ακόλουθο κώδικα.

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

using namespace std;

int main(int argc, char **argv) {
   int a[] = {45, 32, 16, 11, 7, 18, 21, 16, 11, 15};
   int N = sizeof(a) / sizeof(int);
   cout << "The size of the array is " << N << endl;
   int key;
   sort(a, a + N);
   for (int i = 0; i < N; i++)
      cout << a[i] << " ";
   cout << endl;
   cout << "Enter a value to be searched for: ";
   cin >> key;
   if (binary_search(a, a + N, key))
      cout << "found" << endl;
   else
      cout << "not found" << endl;

   auto it = lower_bound(a, a + N, key);
   if (*it == key)
      cout << "found at position " << it - a << endl;
   else
      cout << "not found" << endl;
}
```

Κώδικας 12: Αναζήτηση σε ταξινομημένο πίνακα (search.cpp)

```
The size of the array is 10
7 11 11 15 16 16 18 21 32 45
Enter a value to be searched for: 11
found
found at position 1
```

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:

```
[capture list] (parameter list) -> return type
```

```
{
function body
}
```

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.

```
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:

```
auto area = [](double x, double y)
{
    return x * y;
};
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.

```
#include <algorithm>
#include <iostream>
#include <list>
#include <vector>

using namespace std;

int main() {

  cout << ”Example1: call lambda function” << endl;
  cout << ”Area = ” << [](double x, double y) { return x * y; }(3.0, 4.5)
       << endl;

  cout << ”Example2: assign lambda function to variable” << endl;
  auto area = [](double x, double y) { return x * y; };
  cout << ”Area = ” << area(3.0, 4.5) << endl;
  cout << ”Area = ” << area(7.0, 5.5) << endl;

  vector<int> v{5, 1, 3, 2, 8, 7, 4, 5};
  // find_if
  cout << ”Example3: find the first even number in the vector” << endl;
  vector<int>::iterator iter =
      find_if(v.begin(), v.end(), [](int x) { return x % 2 == 0; });
  cout << *iter << endl;

  // count_if
  cout << ”Example4: count the number of even numbers in the vector” << endl;
  int c = count_if(v.begin(), v.end(), [](int x) { return x % 2 == 0; });
  cout << c << endl;

  // sort
  cout << ”Example5: sort list of rectangles by area (ascending)” << endl;
  struct Rectangle {
    double width;
    double height;
  };
  list<Rectangle> rectangles_list = {{5, 6}, {3, 3}, {5, 2}, {6, 1}};
  rectangles_list.sort([](Rectangle &r1, Rectangle &r2) {
    return r1.width * r1.height < r2.width * r2.height;
```

```
40    });
41    for (Rectangle r : rectangles_list)
42       cout << "Width:" << r.width << " Height:" << r.height
43            << " Area: " << r.width * r.height << endl;
44
45    cout << "Example6: sort vector of rectangles by area (descending)" << endl;
46    vector<Rectangle> rectangles_vector = {{5, 6}, {3, 3}, {5, 2}, {6, 1}};
47    sort(rectangles_vector.begin(), rectangles_vector.end(),
48         [](Rectangle &r1, Rectangle &r2) {
49           return r1.width * r1.height > r2.width * r2.height;
50         });
51    for (Rectangle r : rectangles_vector)
52       cout << "Width:" << r.width << " Height:" << r.height
53            << " Area: " << r.width * r.height << endl;
54
55    // for_each
56    cout << "Example7: for_each" << endl;
57    for_each(v.begin(), v.end(), [](int i) { cout << i << " "; });
58    cout << endl;
59    for_each(v.begin(), v.end(), [](int i) { cout << i * i << " "; });
60    cout << endl;
61 }
```

Κώδικας 13: 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)" << endl;
37   lambda1(1);
38   cout << "Example2: capture by reference (all external variables)" << endl;
39   lambda2(2);
40   cout << "Example3: mixed capture (v1 by value, v2 by reference)" << endl;
41   lambda3(1);
42
43   cout << "v1: ";
44   for (int x : v1)
45     cout << x << " ";
46   cout << endl;
47
48   cout << "v2: ";
49   for (int x : v2)
50     cout << x << " ";
51   cout << endl;
52 }
```

Κώδικας 14: 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 TDD (Test Driven Development)

Τα τελευταία χρόνια έχει αναγνωριστεί ως μια αποδοτική τεχνική ανάπτυξης εφαρμογών η οδηγούμενη από ελέγχους ανάπτυξη (Test Driven Development). Αν και το θέμα είναι αρκετά σύνθετο, η βασική ιδέα είναι ότι ο προγραμματιστής πρώτα γράφει κώδικα που ελέγχει αν η ζητούμενη λειτουργικότητα ικανοποιείται και στη συνέχεια προσθέτει τον κώδικα που θα υλοποιεί αυτή τη λειτουργικότητα [6]. Ανά πάσα στιγμή υπάρχει ένα σύνολο από συσσωρευμένους ελέγχους οι οποίοι για κάθε αλλαγή που γίνεται στον κώδικα είναι σε θέση να εκτελεστούν άμεσα και να δώσουν εμπιστοσύνη μέχρι ένα βαθμό στο ότι το υπό κατασκευή ή υπό τροποποίηση λογισμικό λειτουργεί ορθά. Γλώσσες όπως η Java και η Python διαθέτουν εύχρηστες βιβλιοθήκες που υποστηρίζουν την ανάπτυξη TDD (junit και pytest αντίστοιχα). Στην περίπτωση της C++ επίσης υπάρχουν διάφορες βιβλιοθήκες που μπορούν να υποστηρίξουν την ανάπτυξη TDD. Στα πλαίσια του εργαστηρίου θα χρησιμοποιηθεί η βιβλιοθήκη Catch για το σκοπό της επίδειξης του TDD.

Στο παράδειγμα που ακολουθεί παρουσιάζεται η υλοποίηση της συνάρτησης παραγοντικό. Το παραγοντικό συμβολίζεται με το θαυμαστικό (!), ορίζεται μόνο για τους μη αρνητικούς ακεραίους αριθμούς και είναι το γινόμενο όλων των θετικών ακεραίων που είναι μικρότεροι ή ίσοι του αριθμού για τον οποίο ζητείται το παραγοντικό. Η πρώτη υλοποίηση είναι λανθασμένη καθώς δεν υπολογίζει σωστά το παραγοντικό του μηδενός που πρέπει να είναι μονάδα.

```cpp
#define CATCH_CONFIG_MAIN // This tells Catch to provide a main() - only do this
                          // in one cpp file
#include "catch.hpp"

unsigned int Factorial(unsigned int number) {
  return number <= 1 ? number : Factorial(number - 1) * number;
}

TEST_CASE("Factorials are computed", "[factorial]") {
  REQUIRE(Factorial(0) == 1);
  REQUIRE(Factorial(1) == 1);
  REQUIRE(Factorial(2) == 2);
  REQUIRE(Factorial(3) == 6);
  REQUIRE(Factorial(10) == 3628800);
}
```

Κώδικας 15: Πρώτη έκδοση της συνάρτησης παραγοντικού και έλεγχοι (tdd1.cpp)

```

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
lab02_08 is a Catch v1.10.0 host application.
Run with -? for options

-------------------------------------------------------------------------------
Factorials are computed
-------------------------------------------------------------------------------
lab02_08.cpp:9
...............................................................................

lab02_08.cpp:10: FAILED:
  REQUIRE( Factorial(0) == 1 )
with expansion:
  0 == 1

===============================================================================
test cases: 1 | 1 failed
assertions: 1 | 1 failed
```

Η δεύτερη υλοποίηση είναι σωστή. Τα μηνύματα που εμφανίζονται σε κάθε περίπτωση υποδεικνύουν το σημείο στο οποίο βρίσκεται το πρόβλημα και ότι πλέον αυτό λύθηκε.

```
#define CATCH_CONFIG_MAIN // This tells Catch to provide a main() - only do this
                          // in one cpp file
#include "catch.hpp"

unsigned int Factorial(unsigned int number) {
  return number > 1 ? Factorial(number - 1) * number : 1;
}

TEST_CASE("Factorials are computed", "[factorial]") {
  REQUIRE(Factorial(0) == 1);
  REQUIRE(Factorial(1) == 1);
  REQUIRE(Factorial(2) == 2);
  REQUIRE(Factorial(3) == 6);
  REQUIRE(Factorial(10) == 3628800);
}
```

Κώδικας 16: Δεύτερη έκδοση της συνάρτησης παραγοντικού και έλεγχοι (tdd2.cpp)

```
===============================================================================
All tests passed (5 assertions in 1 test case)
```

6 Παραδείγματα

6.1 Παράδειγμα 1

Να γράψετε πρόγραμμα που να δημιουργεί πίνακα Α με 1.000 τυχαίες ακέραιες τιμές στο διάστημα [1, 10.000] και πίνακα Β 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 Β υπάρχουν στον πίνακα Α.

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

using namespace std;

int main(int argc, char *argv[]) {
  mt19937 mt(1821);
  uniform_int_distribution<int> dist(1, 10000);
  constexpr int N = 1000;
  constexpr int M = 100000;
  int a[N];
  int b[M];
  for (int i = 0; i < N; i++)
    a[i] = dist(mt);
  for (int i = 0; i < M; i++)
    b[i] = dist(mt);
  sort(a, a + N);
  int c = 0;
  for (int i = 0; i < M; i++)
    if (binary_search(a, a + N, b[i]))
      c++;
  cout << "Result " << c << endl;
  return 0;
}
```

Κώδικας 17: Λύση παραδείγματος 1 (lab02_ex1.cpp)

```
Result 9644
```

6.2 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.

```
#include <array>
#include <iostream>
#include <numeric>
#include <vector>

using namespace std;

int main(int argc, char *argv[]) {
   vector<int> v{5, 15, 20, 17, 11, 9};
   int sum = accumulate(v.begin(), v.end(), 0);
   cout << "Sum over vector using accumulate: " << sum << endl;

   array<int, 6> a{5, 15, 20, 17, 11, 9};
   sum = accumulate(a.begin(), a.end(), 0);
   cout << "Sum over array using accumulate: " << sum << endl;
}
```

Κώδικας 18: Λύση παραδείγματος 2 (lab02_ex2.cpp)

```
Sum over vector using accumulate: 77
Sum over array using accumulate: 77
```

6.3 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.

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

using namespace std;

int main(int argc, char *argv[]) {
   vector<string> v{"petros", "anna", "nikos"};
   sort(v.begin(), v.end());
   do {
      for (string x : v)
         cout << x << " ";
      cout << endl;
   } while (next_permutation(v.begin(), v.end()));
}
```

Κώδικας 19: Λύση παραδείγματος 3 (lab02_ex3.cpp)

```
anna nikos petros
anna petros nikos
nikos anna petros
nikos petros anna
petros anna nikos
petros nikos anna
```

6.4 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.

```cpp
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int hamming(string x, string y) {
  int c = 0;
  int length = x.size() > y.size() ? x.size() : y.size();
  for (int i = 0; i < length; i++)
    if (x.at(i) != y.at(i))
      c++;
  return c;
}

int main(int argc, char *argv[]) {
  constexpr int N = 100;
  constexpr int L = 20;
  mt19937 mt(1821);
  uniform_int_distribution<int> dist(0, 3);
  char gact[] = {'A', 'G', 'C', 'T'};
  vector<string> sequences(N);
  for (int i = 0; i < N; i++)
    for (int j = 0; j < L; j++)
      sequences[i] += gact[dist(mt)];

  int c = 0;
  for (int i = 0; i < N; i++) {
    cout << "Checking sequence: " << sequences[i] << "..." << endl;
    for (int j = 0; j < N; j++) {
      if (i == j)
        continue;
      int hd = hamming(sequences[i], sequences[j]);
      cout << sequences[i] << " " << sequences[j]
           << " ==> hamming distance=" << hd << endl;
      if (hd <= 10) {
        c++;
        break;
      }
    }
    cout << endl;
  }
  cout << "Result=" << c << endl;
}
```

Κώδικας 20: Λύση παραδείγματος 4 (lab02_ex4.cpp)

```
Checking sequence: CGCCATCTAAGGACTCCCCA
CGCCATCTAAGGACTCCCCA CACATTCAAACTGTGGGCCA ==> hamming distance=11
...
CGCCATCTAAGGACTCCCCA CCCCATCTCGGCCCACGCTG ==> hamming distance=9

```

```
Checking sequence: CACATTCAAACTGTGGGCCA
CACATTCAAACTGTGGGCCA CGCCATCTAAGGACTCCCCA ==> hamming distance=11
...
CACATTCAAACTGTGGGCCA CATATAACAACAGCATGCGA ==> hamming distance=9

...

Checking sequence: TAGGTGCCATAAGAATCACT
TAGGTGCCATAAGAATCACT CGCCATCTAAGGACTCCCCA ==> hamming distance=16
...
TAGGTGCCATAAGAATCACT AGCTGATTACGACAGCCTTC ==> hamming distance=16

Result=71
```

7 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 ν ενώ για κάθε αρνητική τιμή που εισάγεται να αναζητείται η απόλυτη τιμή της στο διάνυσμα ν. 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.

5. Υλοποιήστε μια κλάση με όνομα BankAccount (λογαριασμός τράπεζας). Για κάθε αντικείμενο της κλάσης να τηρούνται τα στοιχεία όνομα δικαιούχου και υπόλοιπο λογαριασμού. Οι δε λειτουργίες που θα υποστηρίζει να είναι κατ' ελάχιστον η κατάθεση και η ανάληψη χρηματικού ποσού. Η υλοποίηση της κλάσης να γίνει χρησιμοποιώντας τις αρχές του TDD.

Αναφορές

[1] Σταμάτης Σταματιάδης. Εισαγωγή στη γλώσσα προγραμματισμού C++11. Τμήμα Επιστήμης και Τεχνολογίας Υλικών, Πανεπιστήμιο Κρήτης, 2017, https://www.materials.uoc.gr/el/undergrad/courses/ETY215/notes.pdf.

[2] http://www.geeksforgeeks.org/cpp-stl-tutorial/.

[3] https://www.topcoder.com/community/data-science/data-science-tutorials/power-up-c-with-the-standard-template-library-part-1/.

[4] https://www.topcoder.com/community/data-science/data-science-tutorials/power-up-c-with-the-standard-template-library-part-2/.

[5] https://www.hackerearth.com/practice/notes/standard-template-library/

[6] http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd