Δομές Δεδομένων και Αλγόριθμοι - Εργαστήριο 2
Εισαγωγή στα templates και στην STL, TDD

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 ακέραιους 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 (lab02_01.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 (lab02_02.cpp)

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

3 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.

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

Στη συνέχεια παρουσιάζονται παραδείγματα στα οποία χρησιμοποιούνται lambda συναρτήσεις σε συνδυασμό με τις συναρτήσεις της STL: find_if, count_if, unique και accumulate.

```
to be completed
```

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 στην b με αναφορά

```
to be completed
```

4 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.

4.1 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.

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

using namespace std;

int main(int argc, char *argv[]) {
   int x;
   cout << ”Enter size of the vector: ”;
   cin >> x;
   vector<int> v(x);
   for (int i = 0; i < x; i++)
      v[i] = i;
   v.push_back(99);
   for (int i = 0; i < v.size(); i++)
      cout << v[i] << ” ”;
}
```

Κώδικας 3: Παράδειγμα με προσθήκη στοιχείων σε vector (lab02_03.cpp)

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

Ειδικότερα, 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)

Παραδείγματα χρήσης των containers θα παρουσιαστούν στο παρόν και στα επόμενα εργαστήρια.

4.2 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. Τέλος χρησιμοποιείται η for_each σε συνδυασμό με μια lambda συνάρτηση που απλά εμφανίζει τα στοιχεία του vector.

```cpp
#include <algorithm> // for_each
#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;
```

```
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
31    cout << "iteration using for_each and lambda expression: ";
32    for_each(v.begin(), v.end(), [](int i) { cout << i << " "; });
33 }
```

Κώδικας 4: Πέντε διαφορετικοί τρόποι προσπέλασης των στοιχείων ενός vector (lab02_04.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
5 iteration using for_each and lambda expression: 23 13 31 17 56
```

4.3 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 ενός στατικού πίνακα και ενός vector.

```
1  #include <algorithm>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(int argc, char **argv) {
7     cout << "### STL Sort Example ###" << endl;
8     int a[] = {45, 32, 16, 11, 7, 18, 21, 16, 11, 15};
9     cout << "BEFORE (static array example): ";
10    for (int i = 0; i < 10; i++)
11       cout << a[i] << " ";
12    cout << endl;
13    sort(a, a + 10);
14    cout << "AFTER: ";
15    for (int i = 0; i < 10; i++)
16       cout << a[i] << " ";
17    cout << endl;
18
19    cout << "BEFORE (vector example): ";
20    vector<int> va(a, a + 10);
21    auto rng = std::default_random_engine{};
22    shuffle(va.begin(), va.end(), rng);
23    for (auto it = va.begin(); it < va.end(); it++)
24       cout << *it << " ";
25    cout << endl;
26    sort(va.begin(), va.end());
27    cout << "AFTER: ";
28    for (auto it = va.begin(); it < va.end(); it++)
```

```
29      cout << *it << " ";
30    cout << endl;
31    return 0;
32 }
```

Κώδικας 5: Ταξινόμηση με τη συνάρτηση sort της STL (lab02_05.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): 21 18 16 16 11 15 45 11 7 32
5 AFTER: 7 11 11 15 16 16 18 21 32 45
```

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 συνάρτηση μέλος sort που επιτρέπει την ταξινόμησή τους. Στον ακόλουθο κώδικα δημιουργείται μια λίστα με αντικείμενα ορθογώνιων παραλληλογράμμων τα οποία ταξινομούνται με βάση το εμβαδόν τους σε φθίνουσα σειρά. Για την ταξινόμησή ορθογωνίων παρουσιάζονται τέσσερις 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 << " Area "
12          << this->area() << endl;
13   }
14   bool operator<(const Rectangle &other) { return this->area() < other.area(); }
15
16 private:
17   double width;
18   double height;
19 };
20
21 int main() {
22   list<Rectangle> rectangles;
23   rectangles.push_back(Rectangle(5, 6));
24   rectangles.push_back(Rectangle(3, 3));
25   rectangles.push_back(Rectangle(5, 2));
26   rectangles.push_back(Rectangle(6, 1));
27
28   rectangles.sort();
29   for (auto r : rectangles)
30     r.print_info();
31 }
```

Κώδικας 6: Ταξινόμηση λίστας με αντικείμενα - α' τρόπος (lab02_06a.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 bool operator<(const Rectangle &r1, const Rectangle &r2) {
21  return r1.area() < r2.area();
22 }
23
24 int main() {
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 7: Ταξινόμηση λίστας με αντικείμενα - β' τρόπος (lab02_06b.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() {
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 8: Ταξινόμηση λίστας με αντικείμενα - γ' τρόπος (lab02_06c.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;
};

int main() {
  list<Rectangle> rectangles = {{5,6},{3,3},{5,2},{6,1}};

  rectangles.sort([](const Rectangle &r1, const Rectangle &r2) {
    return r1.area() < r2.area();
  });

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

Κώδικας 9: Ταξινόμηση λίστας με αντικείμενα - δ' τρόπος (lab02_06d.cpp)

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

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.

```cpp
#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 10: Αναζήτηση σε ταξινομημένο πίνακα (lab02_07.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
```

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.

```
#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 11: Πρώτη έκδοση της συνάρτησης παραγοντικού και έλεγχοι (lab02_08.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 12: Δεύτερη έκδοση της συνάρτησης παραγοντικού και έλεγχοι (lab02_09.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() {
  mt19937 mt(1821);
  uniform_int_distribution<int> dist(1, 10000);
  constexpr int N = 1000;
  constexpr int M = 100000;
  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 13: Λύση παραδείγματος 1 (lab02_10.cpp)

```
1 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.

```
1  #include <array>
2  #include <iostream>
3  #include <numeric>
4  #include <vector>
5
6  using namespace std;
7
8  int main() {
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 14: Λύση παραδείγματος 2 (lab02_11.cpp)

```
1 Sum over vector using accumulate: 77
2 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.

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

```
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 15: Λύση παραδείγματος 3 (lab02_12.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
```

6.4 Παράδειγμα 4

Κατασκευάστε μια συνάρτηση που να επιστρέφει την απόσταση Hamming ανάμεσα σε δύο σειρές χαρακτήρων που θα δέχεται ως παράμετρο (η απόσταση Hamming είναι το πλήθος των χαρακτήρων που είναι διαφορετικοί στις ίδιες θέσεις ανάμεσα στις δύο σειρές). Δημιουργήστε μια λίστα με 100 τυχαίες σειρές μήκους 100 χρησιμοποιώντας μόνο τους χαρακτήρες G,A,T,C και βρείτε τις αποστάσεις Hamming ανάμεσα σε κάθε πιθανό ζεύγος στοιχείων της λίστας.

7 Ασκήσεις

1. a
2. a
3. a
4. a

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