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

Χρήστος Γκόγκος

2017

Περιεχόμενα

Εργαστήριο 1

Βασικές έννοιες στη C και στη C++

1.1 Εισαγωγή

Στο πρώτο αυτό εργαστήριο θα επιχειρηθεί η παρουσίαση των βασικών γνώσεων που απαιτούνται έτσι ώστε να είναι δυνατή η κατανόηση της ύλης που ακολουθεί. Ειδικότερα, θα γίνει αναφορά σε δείκτες, στη δυναμική δέσμευση και αποδέσμευση μνήμης, στο πέρασμα παραμέτρων σε συναρτήσεις (με τιμή και με αναφορά), στην παραγωγή τυχαίων τιμών, στους πίνακες (μονοδιάστατους, δισδιάστατους, πολυδιάστατους, δυναμικούς), στις δομές (struct), στις κλάσεις και στα αντικείμενα και τέλος στην ανάγνωση από αρχεία και στην εγγραφή σε αυτά. Θα παρουσιαστούν λυμένα παραδείγματα καθώς και εκφωνήσεις ασκήσεων προς επίλυση. Ο κώδικας όλων των παραδειγμάτων βρίσκεται στο https://github.com/chgogos/ceteiep_dsa. Αναλυτικότερη παρουσίαση των ανωτέρω θεμάτων γίνεται στα ελεύθερα διαθέσιμα βιβλία [1], [2], [3], [4] που παρατίθενται ως αναφορές στο τέλος του κειμένου του εργαστηρίου.

1.2 Δείκτες

Κάθε θέση μνήμης στην οποία μπορούν να αποθηκευτούν δεδομένα βρίσκεται σε μια διεύθυνση μνήμης. Η δε μνήμη του υπολογιστή αποτελείται από ένα συνεχόμενο χώρο διευθύνσεων. Αν μια μεταβλητή δηλωθεί ως τύπου int * (δείκτης σε ακέραιο) τότε η τιμή που θα λάβει ερμηνεύεται ως μια διεύθυνση που δείχνει σε μια θέση μνήμης η οποία περιέχει έναν ακέραιο. Από την άλλη μεριά το σύμβολο & επιτρέπει τη λήψη της διεύθυνσης μιας μεταβλητής. Στον ακόλουθο κώδικα δηλώνονται 2 ακέραιες μεταβλητές (a και b) και ένας δείκτης σε ακέραια τιμή (p). Ο δείκτης p λαμβάνει ως τιμή τη διεύθυνση της μεταβλητής a. Στη συνέχεια, οι μεταβλητές a και b λαμβάνουν τιμές μέσω του δείκτη p. Για να συμβεί αυτό γίνεται έμμεση αναφορά ή αλλιώς αποαναφορά (dereference) του δείκτη με το *p. Συνεπώς, το *p αντιστοιχεί στο περιεχόμενο της διεύθυνσης μνήμης που έχει ο δείκτης p.

```
1  #include <cstdio>
2
3  int main(int argc, char **argv) {
4    int a, b;
5    int *p;
6    p = &a;
7    *p = 5;
8    b = *p + 1;
9    printf("variable a=%d address=%p\n", a, &a);
10   printf("variable b=%d address=%p\n", b, &b);
11   printf("pointer p=%p *p=%d\n", p, *p);
12   return 0;
13 }
```

Κώδικας 1.1: Παράδειγμα με δείκτες (pointers1.cpp)

Χρησιμοποιώντας τον compiler g++, η μεταγλώττιση του κώδικα 1.1 γίνεται με την ακόλουθη εντολή:

```
g++ pointers1.cpp -o pointers1
```

Δημιουργείται το εκτελέσιμο pointers1 το οποίο όταν εκτελεστεί παράγει ως έξοδο το:

```
variable a=5 address=000000000022fe44
variable b=6 address=000000000022fe40
pointer p=000000000022fe44 *p=5
```

Ένα συνηθισμένο λάθος με δείκτες παρουσιάζεται όταν γίνεται dereference ενός δείκτη (δηλαδή, δεδομένου ενός δείκτη p όταν χρησιμοποιείται το *p) χωρίς ο δείκτης να έχει αρχικοποιηθεί πρώτα δείχνοντας σε μια έγκυρη θέση μνήμης. Σε αυτή την περίπτωση το πρόγραμμα καταρρέει.

```
#include <iostream>

using namespace std;

int main(int argc, char **argv) {
  int *p;
  *p = 2;
  cout << *p << endl;
  return 0;
}
```

Κώδικας 1.2: Λανθασμένη χρήση δείκτη (pointers2.cpp)

```
segmentation fault
```

Αν η μεταγλώττιση του κώδικα γίνει με το compiler flag **-Wall** τότε θα εμφανιστεί μήνυμα που θα προειδοποιεί για τη λάθος χρήση του δείκτη.

```
g++ -Wall pointers2.cpp -o pointers2
pointers2.cpp: In function 'int main(int, char**)':
pointers2.cpp:8:11: warning: 'p' is used uninitialized in this function [-Wuninitialized]
     *p = 2;
          ^
```

1.3 Κλήση με τιμή και κλήση με αναφορά

Οι δείκτες μπορούν να χρησιμοποιηθούν έτσι ώστε να επιτευχθεί, εφόσον απαιτείται, κλήση με αναφορά (call by reference ή pass by reference) στις παραμέτρους μιας συνάρτησης και όχι κλήση με τιμή (call by value ή pass by value) που είναι ο προκαθορισμένος τρόπος κλήσης συναρτήσεων. Όταν γίνεται κλήση με τιμή τα δεδομένα αντιγράφονται από τη συνάρτηση που καλεί προς τη συνάρτηση που καλείται. Συνεπώς. αν στη συνάρτηση που καλείται τα δεδομένα αλλάξουν η τιμή τους παρουσιάζεται αλλαγμένη μόνο μέσα σε αυτή τη συνάρτηση και όχι στη συνάρτηση που την κάλεσε. Στην περίπτωση της κλήσης με αναφορά ένας δείκτης προς τα δεδομένα αντιγράφεται αντί για τα ίδια τα δεδομένα. Οι αλλαγές που γίνονται στη συνάρτηση που καλείται εφόσον αφορούν τα δεδομένα στα οποία δείχνει ο δείκτης αφορούν τα δεδομένα και της συνάρτησης από την οποία έγινε η κλήση. Συνεπώς, πρόκειται για τα ίδια δεδομένα και οποιαδήποτε αλλαγή σε αυτά μέσα από τη συνάρτηση που καλείται αντικατοπτρίζεται και στη συνάρτηση που καλεί.

Στο παράδειγμα που ακολουθεί η συνάρτηση swap (σε αντίθεση με τη συνάρτηση swap_not_ok) επιτυγχάνει την αντιμετάθεση των δύο μεταβλητών που δέχεται ως ορίσματα καθώς χρησιμοποιεί δείκτες που αναφέρονται στις ίδιες τις μεταβλητές του κυρίου προγράμματος και όχι σε αντίγραφά τους.

```
#include <cstdio>

void swap_not_ok(int x, int y) {
  int temp = x;
```

```
5    x = y;
6    y = temp;
7  }
8
9  void swap(int *x, int *y) {
10   int temp = *x;
11   *x = *y;
12   *y = temp;
13 }
14
15 int main(int argc, char **argv) {
16   int a = 5, b = 7;
17   printf("BEFORE a=%d b=%d\n", a, b);
18   swap_not_ok(a, b);
19   printf("AFTER a=%d b=%d\n", a, b);
20   printf("BEFORE a=%d b=%d\n", a, b);
21   swap(&a, &b);
22   printf("AFTER a=%d b=%d\n", a, b);
23   return 0;
24 }
```

Κώδικας 1.3: Αντιμετάθεση μεταβλητών με δείκτες (swap1.cpp)

```
1 BEFORE a=5 b=7
2 AFTER a=5 b=7
3 BEFORE a=5 b=7
4 AFTER a=7 b=5
```

Η γλώσσα C++ προκειμένου να απλοποιήσει την κλήση με αναφορά εισήγαγε την έννοια των ψευδωνύμων (aliases). Τοποθετώντας στη δήλωση μιας παραμέτρου συνάρτησης το σύμβολο **&** η παράμετρος λειτουργεί ως ψευδώνυμο για τη μεταβλητή που περνά στην αντίστοιχη θέση. Η συγκεκριμένη συμπεριφορά παρουσιάζεται στον ακόλουθο κώδικα.

```
1  #include <cstdio>
2
3  void swap_cpp(int &x, int &y) {
4    int temp = x;
5    x = y;
6    y = temp;
7  }
8
9  int main(int argc, char **argv) {
10   int a = 5, b = 7;
11   printf("BEFORE a=%d b=%d\n", a, b);
12   swap_cpp(a, b);
13   printf("AFTER a=%d b=%d\n", a, b);
14   return 0;
15 }
```

Κώδικας 1.4: Αντιμετάθεση μεταβλητών με αναφορές (swap2.cpp)

```
1 BEFORE a=5 b=7
2 AFTER a=7 b=5
```

1.4 Πίνακες

Ένας πίνακας είναι μια συλλογή από στοιχεία του ίδιου τύπου καθένα από τα οποία μπορεί να αναγνωριστεί από την τιμή ενός ακεραίου δείκτη (index). Το γεγονός αυτό επιτρέπει την τυχαία προσπέλαση (random access) στα στοιχεία του πίνακα. Οι δείκτες των πινάκων ξεκινούν από το μηδέν.

1.4.1 Μονοδιάστατοι πίνακες

Οι μονοδιάστατοι πίνακες είναι η πλέον απλή δομή δεδομένων. Η αναφορά στα στοιχεία του πίνακα γίνεται συνήθως με μια δομή επανάληψης (π.χ. for). Στο ακόλουθο παράδειγμα δύο μονοδιάστατοι πίνακες αρχικοποιούνται κατά τη δήλωσή τους και εν συνεχεία υπολογίζεται το εσωτερικό γινόμενό τους δηλαδή το άθροισμα των γινομένων των στοιχείων των πινάκων που βρίσκονται στην ίδια θέση.

```
#include <cstdio>

int main(int argc, char **argv) {
   double a[] = {3.2, 4.1, 7.3};
   double b[] = {6.0, .1, −5.3};
   double sum = .0;
   for (int i = 0; i < 3; i++)
      sum += a[i] * b[i];
   printf("inner product %.2f\n", sum);
   return 0;
}
```

Κώδικας 1.5: Υπολογισμός εσωτερικού γινομένου δύο πινάκων (arrays1.cpp)

```
inner product −19.08
```

1.4.2 Δυναμικοί πίνακες

Δυναμικοί πίνακες χρησιμοποιούνται όταν το μέγεθος του πίνακα πρέπει να αλλάζει κατά τη διάρκεια εκτέλεσης του προγράμματος και συνεπώς δεν μπορεί να ορισθεί κατά τη μεταγλώττιση. Πριν χρησιμοποιηθεί ένας δυναμικός πίνακας θα πρέπει δεσμευτούν οι απαιτούμενες θέσεις μνήμης. Επίσης, θα πρέπει να απελευθερωθεί ο χώρος που καταλαμβάνει όταν πλέον δεν χρησιμοποιείται. Στο ακόλουθο παράδειγμα ο χρήστης εισάγει το μέγεθος ενός μονοδιάστατου πίνακα και ο απαιτούμενος χώρος δεσμεύεται κατά την εκτέλεση του κώδικα. Στη συνέχεια ο πίνακας γεμίζει με τυχαίες ακέραιες τιμές στο διάστημα [1,100]. Παρουσιάζονται δύο εκδόσεις του κώδικα, μια που χρησιμοποιεί τις συναρτήσεις malloc και free της γλώσσας C και μια που χρησιμοποιεί τις εντολές new και delete της C++ για τη δέσμευση και την αποδέσμευση μνήμης. Επιπλέον, χρησιμοποιείται διαφορετικός τρόπος για τη δημιουργία των τυχαίων τιμών στα δύο προγράμματα.

```
#include <cstdio>
#include <cstdlib>
#include <ctime>

int main(int argc, char **argv) {
   int n;
   printf("Enter the size of the vector: ");
   fflush(stdout);
   scanf("%d", &n);
   int *a = (int *)malloc(sizeof(int) * n);
   // srand(time(NULL));
   srand(1821);

   for (int i = 0; i < n; i++)
      a[i] = (rand() % 100) + 1;

   for (int i = 0; i < n; i++)
      printf("%d ", a[i]);

   free(a);
   return 0;
}
```

Κώδικας 1.6: Δημιουργία δυναμικού πίνακα με συναρτήσεις της C (arrays2.cpp)

```
Enter the size of the vector: 10
86 72 71 16 32 49 75 35 1 70
```

```
#include <iostream>
#include <random>
using namespace std;
int main(int argc, char **argv) {
   mt19937 mt(1821);
   uniform_int_distribution<int> dist(1, 100);
   int n;
   cout << "Enter the size of the vector: ";
   cin >> n;
   int *a = new int[n];
   for (int i = 0; i < n; i++)
      a[i] = dist(mt);
   for (int i = 0; i < n; i++)
      cout << a[i] << " ";
   delete[] a;
   return 0;
}
```

Κώδικας 1.7: Δημιουργία δυναμικού πίνακα με συναρτήσεις της C++ (arrays3.cpp)

```
Enter the size of the vector: 10
73 44 67 66 1 98 52 85 19 24
```

Για να γίνει η μεταγλώττιση του κώδικα 1.7 θα πρέπει να χρησιμοποιηθεί το flag **-std=c++11** όπως φαίνεται στην ακόλουθη εντολή.

```
g++ arrays3.cpp -o arrays3 -std=c++11
```

1.4.3 Πίνακας ως παράμετρος συνάρτησης και επιστροφή πολλών αποτελεσμάτων

Ένας πίνακας μπορεί να περάσει ως παράμετρος σε μια συνάρτηση. Συχνά χρειάζεται να περάσουν ως παράμετροι και οι διαστάσεις του πίνακα. Στον ακόλουθο κώδικα η συνάρτηση simple_stats δέχεται ως παράμετρο έναν μονοδιάστατο πίνακα ακεραίων και το πλήθος των στοιχείων του και επιστρέφει μέσω κλήσεων με αναφορά το μέσο όρο, το ελάχιστο και το μέγιστο από όλα τα στοιχεία του πίνακα.

```
#include <cstdio>

void simple_stats(int a[], int N, double *avg, int *min, int *max) {
   int sum;
   sum = *min = *max = a[0];
   for (int i = 1; i < N; i++) {
      sum += a[i];
      if (*max < a[i])
         *max = a[i];
      if (*min > a[i])
         *min = a[i];
   }
   *avg = (double)sum / (double)N;
}

int main(int argc, char **argv) {
   int a[] = {3, 2, 9, 5, 1};
   double avg;
   int min, max;
   simple_stats(a, 5, &avg, &min, &max);
   printf("Average= %.2f min=%d max=%d\n", avg, min, max);
```

```
22   return 0;
23 }
```

Κώδικας 1.8: Δυναμικός πίνακας ως παράμετρος συνάρτησης και χρήση δεικτών για επιστροφή τιμών (arrays4.cpp)

```
1 Average= 4.00 min=9 max=1
```

Το ίδιο αποτέλεσμα με τον παραπάνω κώδικα μπορεί να επιτευχθεί απλούστερα χρησιμοποιώντας το μηχανισμό των ψευδωνύμων της C++. Σε αυτή την περίπτωση ο κώδικας θα είναι ο ακόλουθος.

```
1  #include <cstdio>
2
3  void simple_stats(int a[], int N, double &avg, int &min, int &max) {
4    int sum;
5    sum = min = max = a[0];
6    for (int i = 1; i < N; i++) {
7      sum += a[i];
8      if (max < a[i])
9        max = a[i];
10     if (min > a[i])
11       min = a[i];
12   }
13   avg = (double)sum / (double)N;
14 }
15
16 int main(int argc, char **argv) {
17   int a[] = {3, 2, 9, 5, 1};
18   double avg;
19   int min, max;
20   simple_stats(a, 5, avg, min, max);
21   printf("Average= %.2f min=%d max=%d\n", avg, min, max);
22   return 0;
23 }
```

Κώδικας 1.9: Δυναμικός πίνακας ως παράμετρος συνάρτησης και χρήση ψευδωνύμων για επιστροφή τιμών (arrays5.cpp)

1.4.4 Δισδιάστατοι πίνακες

Ένας δισδιάστατος πίνακας αποτελείται από γραμμές και στήλες και η αναφορά στα στοιχεία του γίνεται με δύο δείκτες από τους οποίους ο πρώτος δείκτης υποδηλώνει τη γραμμή και ο δεύτερος υποδηλώνει τη στήλη του πίνακα. Οι πίνακες είναι ιδιαίτερα σημαντικοί για την εκτέλεση μαθηματικών υπολογισμών (π.χ. πολλαπλασιασμό πινάκων, επίλυση συστημάτων γραμμικών εξισώσεων κ.α.). Στον ακόλουθο κώδικα δίνεται ένα παράδειγμα δήλωσης ενός δισδιάστατου πίνακα 5 x 4 ο οποίος περνά ως παράμετρος στη συνάρτηση sums_row_wise. Η δε συνάρτηση επιστρέφει το άθροισμα κάθε γραμμής του πίνακα.

```
1  #include <cstdio>
2
3  void sums_row_wise(int a[][4], int M, int N, int *row) {
4    for (int i = 0; i < M; i++) {
5      row[i] = 0;
6      for (int j = 0; j < N; j++)
7        row[i] += a[i][j];
8    }
9  }
10
11 int main(int argc, char **argv) {
12   int a[5][4] = {{5, 4, 0, −1},
```

```
                    {1, 5, 42, 2},
                    {−3, 7, 8, 2},
                    {7, 312, −56, 6},
                    {19, 45, 6, 5}};
  int row[5];
  sums_row_wise(a, 5, 4, row);
  for (int i = 0; i < 5; i++)
    printf("sum of row %d is %d\n", i, row[i]);
  return 0;
}
```

Κώδικας 1.10: Δισδιάστατος πίνακας ως παράμετρος συνάρτησης (arrays6.cpp)

```
sum of row 0 is 8
sum of row 1 is 50
sum of row 2 is 14
sum of row 3 is 269
sum of row 4 is 75
```

1.4.5 Πολυδιάστατοι πίνακες

Αν και οι μονοδιάστατοι και οι δισδιάστατοι πίνακες χρησιμοποιούνται συχνότερα, υποστηρίζονται και πίνακες μεγαλύτερων διαστάσεων. Στη συνέχεια δίνεται ένα παράδειγμα δήλωσης και αρχικοποίησης ενός τρισδιάστατου πίνακα 3x3x2 και ενός τετραδιάστατου πίνακα 3x3x3x2.

```
int main() {
  int d[3][3][2];
  int e[3][3][2][2];
  for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
      for (int k = 0; k < 2; k++) {
        d[i][j][k] = 1;
        for (int l = 0; l < 2; l++)
          e[i][j][k][l] = 1;
      }
}
```

Κώδικας 1.11: Δήλωση και αρχικοποίηση τρισδιάστατου και τετραδιάστατου πίνακα (arrays7.cpp)

1.4.6 Πριονωτοί πίνακες

Εφόσον ένας πολυδιάστατος πίνακας δημιουργείται δυναμικά μπορεί να οριστεί με τέτοιο τρόπο έτσι ώστε η κάθε γραμμή του να μην έχει τον ίδιο αριθμό στοιχείων. Στον ακόλουθο κώδικα δημιουργείται ένας δισδιάστατος πίνακας 5 γραμμών με την πρώτη γραμμή να έχει 1 στοιχείο και κάθε επόμενη γραμμή ένα περισσότερο στοιχείο από την προηγούμενη της.

```
#include <iostream>
using namespace std;
int main(int argc, char **argv) {
  int *a[5];
  for (int i = 0; i < 5; i++) {
    a[i] = new int[i + 1];
    for (int j = 0; j < i + 1; j++)
      a[i][j] = i + j;
  }
  for (int i = 0; i < 5; i++) {
    for (int j = 0; j < i + 1; j++)
      cout << a[i][j] << " ";
    cout << endl;
```

```
14    }
15    for (int i = 0; i < 5; i++)
16       delete[] a[i];
17    return 0;
18 }
```

Κώδικας 1.12: Παράδειγμα πριονωτού πίνακα με 5 γραμμές (arrays8.cpp)

```
1 0
2 1 2
3 2 3 4
4 3 4 5 6
5 4 5 6 7 8
```

1.5 Δομές

Οι δομές χρησιμοποιούνται όταν απαιτούνται σύνθετοι τύποι δεδομένων οι οποίοι αποτελούνται από επιμέρους στοιχεία. Στο παράδειγμα που ακολουθεί ορίζεται η δομή Book με 3 πεδία. Στη συνέχεια δημιουργούνται 3 μεταβλητές που πρόκειται να αποθηκεύσουν πληροφορίες για ένα βιβλίο η κάθε μια. Η τρίτη μεταβλητή είναι δείκτης προς τη δομή Book και προκειμένου να χρησιμοποιηθεί θα πρέπει πρώτα να δεσμευθεί μνήμη (new) ενώ με τον τερματισμό του προγράμματος θα πρέπει η μνήμη αυτή να επιστραφεί στο σύστημα (delete).

```
1  #include <iostream>
2
3  using namespace std;
4
5  typedef struct {
6     string title;
7     int price;
8     bool isHardpack;
9  } Book;
10
11 // struct Book
12 // {
13 // string title;
14 // int price;
15 // bool isHardpack;
16 // };
17
18 void print_book(Book b) {
19    cout << "Title: " << b.title << " Price: " << b.price / 100.0
20         << " Hardcover: " << (b.isHardpack ? "YES" : "NO") << endl;
21 }
22
23 int main(int argc, char **argv) {
24    Book b1, b2, *pb;
25    b1.title = "The SIMPSONS and their mathematical secrets";
26    b1.price = 1899;
27    b1.isHardpack = false;
28    b2 = b1;
29    b2.isHardpack = true;
30    b2.price = 2199;
31    pb = new Book;
32    pb->title = "Bad Science";
33    pb->price = 999;
34    pb->isHardpack = false;
35    print_book(b1);
36    print_book(b2);
37    print_book(*pb);
```

```
38   delete pb;
39   return 0;
40 }
```

Κώδικας 1.13: Μεταβλητές τύπου δομής Book (structs1.cpp)

```
1 Title: The SIMPSONS and their mathematical secrets Price: 18.99 Hardcover: NO
2 Title: The SIMPSONS and their mathematical secrets Price: 21.99 Hardcover: YES
3 Title: Bad Science Price: 9.99 Hardcover: NO
```

1.6 Κλάσεις - Αντικείμενα

Ο αντικειμενοστρεφής προγραμματισμός εντοπίζει τα αντικείμενα που απαρτίζουν την εφαρμογή και τα συνδυάζει προκειμένου να επιτευχθεί η απαιτούμενη λειτουργικότητα. Για κάθε αντικείμενο γράφεται μια κλάση η οποία είναι υπεύθυνη για τη δημιουργία των επιμέρους στιγμιοτύπων (object instances). Κάθε αντικείμενο έχει μεταβλητές και συναρτήσεις οι οποίες μπορεί να είναι είτε ιδιωτικές (private) είτε δημόσιες (public) (είτε προστατευμένες-protected). Τα ιδιωτικά μέλη χρησιμοποιούνται εντός της κλάσης που ορίζει το αντικείμενο ενώ τα δημόσια μπορούν να χρησιμοποιηθούν και από κώδικα εκτός της κλάσης. Στο ακόλουθο παράδειγμα ορίζεται η κλάση Box η οποία έχει 3 ιδιωτικά μέλη (length, width, height) και 1 δημόσιο μέλος, τη συνάρτηση volume. Επιπλέον η κλάση Box διαθέτει έναν κατασκευαστή (constructor) που δέχεται τρεις παραμέτρους και μπορεί να χρησιμοποιηθεί για τη δημιουργία νέων αντικειμένων. Στη main δημιουργούνται με τη βοήθεια του κατασκευαστή δύο αντικείμενα (στιγμιότυπα) της κλάσης Box και καλείται για καθένα από αυτά η δημόσια συνάρτηση μέλος της Box, volume. Θα πρέπει να σημειωθεί ότι η πρόσβαση στα μέλη δεδομένων (length, width, height) δεν είναι εφικτή από συναρτήσεις εκτός της κλάσης καθώς τα μέλη αυτά είναι ιδιωτικά. Ωστόσο, αν η πρόσβαση στα ιδιωτικά μέλη ήταν επιθυμητή τότε θα έπρεπε να κατασκευαστούν δημόσιες συναρτήσεις μέλη (getters και setters) έτσι ώστε να δοθεί έμμεση πρόσβαση μέσω αυτών στα ιδιωτικά μέλη της κλάσης.

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Box {
6  public:
7    // constructor declaration with default values
8    Box(double l = 1, double w = 1, double h = 1) {
9      length = l;
10     width = w;
11     height = h;
12   };
13   // member function
14   double volume() { return length * width * height; }
15
16 private:
17   // member data
18   double length;
19   double width;
20   double height;
21 };
22
23 int main(int argc, char **argv) {
24   Box b1(10, 5, 10);
25   cout << "The volume is " << b1.volume() << endl;
26   Box *pb = new Box();
27   cout << "The volume is " << pb->volume() << endl;
28   delete pb;
```

```
29   return 0;
30 }
```

Κώδικας 1.14: Παράδειγμα κλάσης Box (objects1.cpp)

```
1 The volume is 500
2 The volume is 1
```

Εναλλακτικά, ο κώδικας του προηγούμενου παραδείγματος μπορεί να γραφτεί όπως παρακάτω, πραγματοποιώντας τη συγγραφή του σώματος των συναρτήσεων της κλάσης Box μετά τη δήλωση της κλάσης και χρησιμοποιώντας λίστα αρχικοποίησης τιμών (initializer list) στον κατασκευαστή της κλάσης.

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Box {
6  public:
7    // constructor declaration with default values
8    Box(double l = 1, double w = 1, double h = 1);
9    // member function
10   double volume();
11
12 private:
13   // member data
14   double length;
15   double width;
16   double height;
17 };
18
19 // constructor using initializer list
20 Box::Box(double l, double w, double h) : length(l), width(w), height(h) {}
21
22 double Box::volume() { return length * width * height; }
23
24 int main(int argc, char **argv) {
25   Box b1(10, 5, 10);
26   cout << "The volume is " << b1.volume() << endl;
27   Box *pb = new Box();
28   cout << "The volume is " << pb->volume() << endl;
29   delete pb;
30   return 0;
31 }
```

Κώδικας 1.15: Παράδειγμα κλάσης Box (objects2.cpp)

1.7 Αρχεία

Συχνά χρειάζεται να αποθηκεύσουμε δεδομένα σε αρχεία ή να επεξεργαστούμε δεδομένα τα οποία βρίσκονται σε αρχεία. Ο ακόλουθος κώδικας πρώτα δημιουργεί έναν αρχείο με 100 τυχαίους ακεραίους στον τρέχοντα κατάλογο και στη συνέχεια ανοίγει το αρχείο και εμφανίζει τα στοιχεία του.

1.7.1 Εγγραφή και ανάγνωση δεδομένων από αρχείο με συναρτήσεις της C

```
1 #include <cstdio>
2 #include <cstdlib>
3 #include <ctime>
4
```

```
int main(int argc, char **argv) {
  FILE *fp = fopen("data_int_100.txt", "w");
  srand(time(NULL));
  for (int i = 0; i < 100; i++)
    fprintf(fp, "%d ", rand() % 1000 + 1);
  fclose(fp);

  fp = fopen("data_int_100.txt", "r");
  if (fp == NULL) {
    printf("error opening file");
    exit(-1);
  }
  int x;
  while (!feof(fp)) {
    fscanf(fp, "%d", &x);
    printf("%d ", x);
  }
  fclose(fp);
  return 0;
}
```

Κώδικας 1.16: Εγγραφή 100 ακέραιων αριθμητικών δεδομένων σε αρχείο και ανάγνωση τους από το ίδιο αρχείο (files1.cpp)

```
811 718 632 412 529 957 359 735 498 302 855 265 749 756 336 625 489 870 120 177 ...
```

1.7.2 Εγγραφή και ανάγνωση δεδομένων από αρχείο με συναρτήσεις της C++

Η C++ έχει προσθέσει νέους τρόπους με τους οποίους μπορεί να γίνει η αλληλεπίδραση με τα αρχεία. Ακολουθεί ένα παράδειγμα εγγραφής και ανάγνωσης δεδομένων από αρχείο με τη χρήση των fstream και sstream.

```
#include <fstream>
#include <iomanip>
#include <iostream>
#include <sstream>

using namespace std;

int main(int argc, char **argv) {
  int constexpr N = 10;
  fstream filestr;
  string buffer;
  int i = 0;
  string names[N] = {"nikos", "maria", "petros", "sofia", "kostas",
                     "dimitra", "giorgos", "christos", "anna", "apostolis"};
  int grades[N] = {55, 30, 70, 80, 10, 25, 75, 90, 100, 30};
  filestr.open("data_student_struct10.txt", ios::out);
  if (!filestr.is_open()) {
    cerr << "file not found" << std::endl;
    exit(-1);
  }
  for (i = 0; i < N; i++)
    filestr << names[i] << "\t" << grades[i] << endl;
  filestr.close();

  filestr.open("data_student_struct10.txt");
  if (!filestr.is_open()) {
    cerr << "file not found" << std::endl;
    exit(-1);
  }
```

```
30    string name;
31    int grade;
32    while (getline(filestr, buffer)) {
33       stringstream ss(buffer);
34       ss >> name;
35       ss >> grade;
36       cout << name << " " << setprecision(1) << fixed
37             << static_cast<double>(grade) / 10.0 << endl;
38    }
39    filestr.close();
40    return 0;
41 }
```

Κώδικας 1.17: Εγγραφή και ανάγνωση αλφαριθμητικών και ακεραίων από αρχείο (files2.cpp)

```
1  nikos 5.5
2  maria 3.0
3  petros 7.0
4  sofia 8.0
5  kostas 1.0
6  dimitra 2.5
7  giorgos 7.5
8  christos 9.0
9  anna 10.0
10 apostolis 3.0
```

1.8 Παραδείγματα

1.8.1 Παράδειγμα 1

Γράψτε κώδικα που να δημιουργεί μια δομή με όνομα Point και να έχει ως πεδία 2 double αριθμούς (x και y) που υποδηλώνουν τις συντεταγμένες ενός σημείου στο καρτεσιανό επίπεδο. Δημιουργήστε έναν πίνακα με όνομα points με 5 σημεία με απευθείας εισαγωγή τιμών για τα ακόλουθα σημεία: (4, 17), (10, 21), (5, 32), (-1, 16), (-4, 7). Γράψτε τον κώδικα που εμφανίζει τα 2 πλησιέστερα σημεία. Ποια είναι τα πλησιέστερα σημεία και ποια η απόσταση μεταξύ τους;

```
1  #include <climits>
2  #include <cmath>
3  #include <cstdio>
4
5  struct Point {
6     double x;
7     double y;
8  };
9
10 int main(int argc, char **argv) {
11    struct Point points[5] = {{4, 17}, {10, 21}, {5, 32}, {−1, 16}, {−4, 7}};
12
13    double min = INT_MAX;
14    Point a, b;
15    for (int i = 0; i < 5; i++)
16       for (int j = i + 1; j < 5; j++) {
17          Point p1 = points[i];
18          Point p2 = points[j];
19          double distance = sqrt(pow(p2.x − p1.x, 2.0) + pow(p2.y − p1.y, 2.0));
20          if (distance < min) {
21             min = distance;
22             a = p1;
23             b = p2;
24          }
```

```
25      }
26    printf("Min %.4f\n", min);
27    printf("Point A: (%.4f, %.4f)\n", a.x, a.y);
28    printf("Point B: (%.4f, %.4f)\n", b.x, b.y);
29    return 0;
30  }
```

Κώδικας 1.18: Λύση παραδείγματος 1 (lab01_ex1.cpp)

```
1  Min 5.0990
2  Point A: (4.0000, 17.0000)
3  Point B: (−1.0000, 16.0000)
```

1.8.2 Παράδειγμα 2

Με τη γεννήτρια τυχαίων αριθμών mt19937 δημιουργήστε 10000 τυχαίες ακέραιες τιμές στο διάστημα 0 έως 10000 με seed την τιμή 1729. Τοποθετήστε τις τιμές σε ένα δισδιάστατο πίνακα 100 x 100 έτσι ώστε να συμπληρώνονται οι τιμές στον πίνακα κατά σειρές από πάνω προς τα κάτω και από αριστερά προς τα δεξιά. Να υπολογιστεί το άθροισμα της κάθε γραμμής του πίνακα. Ποιος είναι ο αριθμός της γραμμής με το μεγαλύτερο άθροισμα και ποιο είναι αυτό;

```
1  #include <iostream>
2  #include <random>
3
4  using namespace std;
5
6  int main(int argc, char **argv) {
7    int seed = 1729;
8    int a[100][100];
9    mt19937 mt(seed);
10   uniform_int_distribution<int> dist(0, 10000);
11   for (int i = 0; i < 100; i++)
12     for (int j = 0; j < 100; j++)
13       a[i][j] = dist(mt);
14   int b[100];
15   int max = 0;
16   for (int i = 0; i < 100; i++) {
17     int sum = 0;
18     for (int j = 0; j < 100; j++)
19       sum += a[i][j];
20     b[i] = sum;
21     if (sum > max)
22       max = sum;
23   }
24   cout << "Maximum row sum: " << max << endl;
25   for (int i = 0; i < 100; i++)
26     if (b[i] == max)
27       cout << "Occurs at row: " << i << endl;
28   return 0;
29 }
```

Κώδικας 1.19: Λύση παραδείγματος 2 (lab01_ex2.cpp)

```
1  Maximum row sum: 586609
2  Occurs at row: 40
```

1.8.3 Παράδειγμα 3

Γράψτε 10000 τυχαίες ακέραιες τιμές στο διάστημα [1,10000] στο αρχείο data_int_10000.txt χρησιμοποιώντας τις συναρτήσεις rand και srand και seed την τιμή 1729. Διαβάστε τις τιμές από το αρχείο. Εντοπίστε τη μεγαλύτερη τιμή στα δεδομένα. Ποιες είναι οι τιμές που εμφανίζονται τις περισσότερες φορές στα δεδομένα;

```cpp
#include <cstdio>
#include <cstdlib>
#include <iostream>


using namespace std;

int main(int argc, char **argv) {
   const char *fn = "data_int_10000.txt";
   int N = 10000;
   srand(1729);

   FILE *fp = fopen(fn, "w");
   if (fp == NULL) {
      printf("error opening file");
      exit(-1);
   }
   for (int i = 0; i < N; i++)
      fprintf(fp, "%d ", rand() % 10000 + 1);
   fclose(fp);

   fp = fopen(fn, "r");
   if (fp == NULL) {
      printf("error opening file");
      exit(-1);
   }
   int *data = new int[N];
   int i = 0;
   while (!feof(fp)) {
      fscanf(fp, "%d", &data[i]);
      i++;
   }
   fclose(fp);
   int max = data[0];
   for (i = 1; i < N; i++)
      if (data[i] > max)
         max = data[i];
   printf("Maximum value %d\n", max);
   int *freq = new int[max + 1];
   for (i = 0; i < max + 1; i++)
      freq[i] = 0;
   for (i = 0; i < N; i++)
      freq[data[i]]++;
   int max2 = 0;
   for (i = 0; i < max + 1; i++)
      if (freq[i] > max2)
         max2 = freq[i];
   for (i = 0; i < max + 1; i++)
      if (freq[i] == max2)
         printf("Value %d exists %d times\n", i, max2);
   delete[] freq;
   return 0;
}
```

Κώδικας 1.20: Λύση παραδείγματος 3 (lab01_ex3.cpp)

```
Maximum value 9999
Value 885 exists 6 times
Value 1038 exists 6 times
Value 3393 exists 6 times
Value 4771 exists 6 times
Value 5482 exists 6 times
Value 8722 exists 6 times
Value 9501 exists 6 times
```

1.8.4 Παράδειγμα 4

Γράψτε κώδικα που να δημιουργεί μια δομή με όνομα student (σπουδαστής) και να έχει ως πεδία το name (όνομα) τύπου string και το grade (βαθμός) τύπου int. Διαβάστε τα περιεχόμενα του αρχείου που έχει δημιουργηθεί με τον κώδικα 1.17 (data_student_struct10.txt) και τοποθετήστε τα σε κατάλληλο πίνακα. Βρείτε τα ονόματα και το μέσο όρο βαθμολογίας των σπουδαστών με βαθμό άνω του μέσου όρου όλων των σπουδαστών. Θεωρείστε ότι οι βαθμοί έχουν αποθηκευτεί στο αρχείο data_student_struct10.txt ως ακέραιοι αριθμοί από το 0 μέχρι και το 100, αλλά η εμφάνισή τους θα πρέπει να γίνεται εφόσον πρώτα διαιρεθούν με το 10. Δηλαδή, ο βαθμός 55 αντιστοιχεί στο βαθμό 5.5.

```
#include <fstream>
#include <iostream>
#include <sstream>

using namespace std;

struct student {
   string name;
   int grade;
};

int main(int argc, char **argv) {
   constexpr int N = 10;
   int i = 0;
   student students[N];
   const char *fn = "data_student_struct10.txt";
   fstream filestr;
   string buffer;
   filestr.open(fn);
   if (!filestr.is_open()) {
      cerr << "File not found" << std::endl;
      exit(-1);
   }
   while (getline(filestr, buffer)) {
      stringstream ss(buffer);
      ss >> students[i].name;
      ss >> students[i].grade;
      i++;
   }
   filestr.close();
   double sum = 0.0;
   for (i = 0; i < N; i++)
      sum += students[i].grade;
   double avg = sum / N;
   cout << "Average grade =" << avg / 10.0 << endl;

   double sum2 = 0.0;
   int c = 0;
   for (i = 0; i < N; i++)
      if (students[i].grade > avg) {
```

```
41          cout << students[i].name << " grade = " << students[i].grade / 10.0
42                  << endl;
43          sum2 += students[i].grade;
44          c++;
45        }
46    double avg2 = sum2 / c;
47    cout << "Average grade for students having grade above the average grade = "
48           << avg2 / 10.0 << endl;
49    return 0;
50  }
```

Κώδικας 1.21: Λύση παραδείγματος 4 (lab01_ex4.cpp)

```
1  Average grade =5.65
2  petros grade = 7
3  sofia grade = 8
4  giorgos grade = 7.5
5  christos grade = 9
6  anna grade = 10
7  Average grade for students having grade above the average grade = 8.3
```

1.9 Ασκήσεις

1. Γράψτε μια συνάρτηση που να δέχεται έναν πίνακα ακεραίων και το μέγεθός του και να επιστρέφει το μέσο όρο των τιμών καθώς και το πλήθος των τιμών που απέχουν το πολύ 10% από το μέσο όρο. Δοκιμάστε την κλήση της συνάρτησης για έναν πίνακα 100 θέσεων με τυχαίες ακέραιες τιμές στο διάστημα [1,100] οι οποίες θα δημιουργηθούν με τη χρήση των συναρτήσεων srand() και rand() της C. Χρησιμοποιήστε ως seed για την αρχικοποίηση των τυχαίων τιμών την τιμή 12345.

2. Γράψτε πρόγραμμα που να διαβάζει τα στοιχεία υπαλλήλων (όνομα, μισθό και έτη προϋπηρεσίας) από το αρχείο data_ypallhlos_struct20.txt και να εμφανίζει τα στοιχεία του κάθε υπαλλήλου μέσω μιας συνάρτησης που θα δέχεται ως παράμετρο μια μεταβλητή τύπου δομής υπαλλήλου. Στη συνέχεια να υπολογίζει και να εμφανίζει το ποσό που θα συγκεντρωθεί αν για κάθε υπάλληλο με περισσότερα από 5 έτη προϋπηρεσίας παρακρατηθεί το 5% του μισθού του ενώ για τους υπόλοιπους υπαλλήλους παρακρατηθεί το 7% του μισθού τους.

3. Γράψτε το προηγούμενο πρόγραμμα ξανά χρησιμοποιώντας κλάση στη θέση της δομής. Επιπλέον ορίστε constructor και getters/setters για τα μέλη δεδομένων του αντικειμένου υπάλληλος.

4. Γράψτε ένα πρόγραμμα που να γεμίζει έναν πίνακα με όνομα a, 5 γραμμών και 5 στηλών, με τυχαίες ακέραιες τιμές στο διάστημα 1 έως και 1000 (χρησιμοποιήστε ως seed την τιμή 12345). Γράψτε μια συνάρτηση που να δέχεται ως παράμετρο τον πίνακα a και να επιστρέφει σε μονοδιάστατο πίνακα με όνομα col το άθροισμα των τιμών κάθε στήλης του πίνακα. Οι τιμές που επιστρέφονται να εμφανίζονται στο κύριο πρόγραμμα το οποίο να εμφανίζει επιπλέον και τον αριθμό στήλης με το μεγαλύτερο άθροισμα.

Βιβλιογραφία

[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] Allen B. Downey. How to think like a computer scientist, C++ version, 2012, http://www.greenteapress.com/thinkcpp/.

[3] Juan Soulié. C++ Language Tutorial. cplusplus.com, 2007, http://www.cplusplus.com/files/tutorial.pdf.

[4] Brian Hall. Beej's Guide to C Programming, 2007, http://beej.us/guide/bgc/.

Εργαστήριο 2

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

2.1 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].

2.2 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 2.1: 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 2.2: 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.

2.3 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.

2.3.1 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.

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

using namespace std;

int main(int argc, char *argv[]) {
   int x;
   cout << "Enter the 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 2.3: Παράδειγμα με προσθήκη στοιχείων σε vector (container1.cpp)

```
Enter size of vector: 10
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.

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

using namespace std;

void print_vector(const string &name, const vector<int> &v) {
   cout << name << ": ";
   for (int i = 0; i < v.size(); i++)
      cout << v[i] << " ";
   cout << endl;
}

int main(int argc, char *argv[]) {
   vector<int> v1;
   v1.push_back(5);
   v1.push_back(16);
   print_vector("v1", v1);

   vector<int> v2 = {16, 3, 6, 1, 9, 10};
   print_vector("v2", v2);

   vector<int> v3{5, 2, 10, 1, 8};
   print_vector("v3", v3);

   vector<int> v4(5, 10);
   print_vector("v4", v4);

   vector<int> v5(v2);
   print_vector("v5", v5);
```

```
30
31    vector<int> v6(v2.begin() + 1, v2.end() − 1);
32    print_vector(”v6”, v6);
33  }
```

Κώδικας 2.4: 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 (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:

- 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 2.5: Παράδειγμα με map (container3.cpp)

```
1  Enter name: nikos
2  The phone is 1234567890
```

2.3.2 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 2.6: 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.

```
#include <iostream>
#include <map>

using namespace std;
int main(int argc, char *argv[]) {
   map<string, int> cities_population_2011 = {{"arta", 21895},
                                              {"ioannina", 65574},
                                              {"preveza", 19042},
                                              {"igoumenitsa", 9145}};

   cout << "Cities of Epirus using range for:" << endl;
   for (auto kv : cities_population_2011)
      cout << kv.first << " " << kv.second << endl;

   cout << "Cities of Epirus using iterator:" << endl;
   for (map<string, int>::iterator iter = cities_population_2011.begin();
         iter != cities_population_2011.end(); iter++)
      cout << iter->first << " " << iter->second << endl;

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

Κώδικας 2.7: Τρεις διαφορετικοί τρόποι προσπέλασης των στοιχείων ενός map (iterator2.cpp)

```
Cities of Epirus using range for:
arta 21895
igoumenitsa 9145
ioannina 65574
preveza 19042
Cities of Epirus using iterator:
arta 21895
igoumenitsa 9145
ioannina 65574
preveza 19042
Cities of Epirus using reverse iterator:
preveza 19042
ioannina 65574
igoumenitsa 9145
arta 21895
```

2.3.3 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>
#include <vector>

using namespace std;

int main(int argc, char **argv) {
   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  for (auto it = va.begin(); it < va.end(); it++)
26     cout << *it << " ";
27  cout << endl;
28  sort(va.begin(), va.end());
29  cout << "AFTER: ";
30  for (auto it = va.begin(); it < va.end(); it++)
31     cout << *it << " ";
32  cout << endl;
33
34  cout << "BEFORE (vector example 2): ";
35  shuffle(va.begin(), va.end(), rng);
36  for (auto it = va.begin(); it < va.end(); it++)
37     cout << *it << " ";
38  cout << endl;
39  // descending
40  sort(va.begin(), va.end(), greater<int>());
41  cout << "AFTER: ";
42  for (auto it = va.begin(); it < va.end(); it++)
43     cout << *it << " ";
44  cout << endl;
45  return 0;
46 }
```

Κώδικας 2.8: 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 κώδικά 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>
```

```
#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) const {
    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 2.9: 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.

```
#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[]) {
```

```
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 2.10: 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 << " 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 2.11: 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<(Rectangle other) {
10     return width * height < other.width * other.height;
```

```
  }
};

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 2.12: Ταξινόμηση λίστας με εγγραφές (sort5.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 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.

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

using namespace std;

int main(int argc, char **argv) {
  int a[] = {45, 11, 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 using binary_search" << endl;
  else
    cout << "Not found (binary_search)" << endl;

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

Κώδικας 2.13: Αναζήτηση σε ταξινομημένο πίνακα (search1.cpp)

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

```
Found using binary_search
Found at positions 1 up to 3
```

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

using namespace std;

int main(int argc, char **argv) {
   vector<int> a = {45, 11, 16, 11, 7, 18, 21, 16, 11, 15};
   cout << "The size of the vector is " << a.size() << endl;
   int key;
   sort(a.begin(), a.end());
   for (int i = 0; i < a.size(); i++)
      cout << a[i] << " ";
   cout << endl;
   cout << "Enter a value to be searched for: ";
   cin >> key;
   if (binary_search(a.begin(), a.end(), key))
      cout << "Found using binary_search" << endl;
   else
      cout << "Not found (binary_search)" << endl;

   auto it1 = lower_bound(a.begin(), a.end(), key);
   auto it2 = upper_bound(a.begin(), a.end(), key);
   if (*it1 == key) {
      cout << "Found at positions " << it1 - a.begin() << " up to "
           << it2 - a.begin() - 1 << endl;
   } else
      cout << "Not found (lower_bound and upper_bound)" << endl;
}
```

Κώδικας 2.14: Αναζήτηση σε ταξινομημένο διάνυσμα (search2.cpp)

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

2.4 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 a 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;
  });
  for (Rectangle r : rectangles_list)
    cout << ”Width:” << r.width << ” Height:” << r.height
         << ” Area: ” << r.width * r.height << endl;

  cout << ”Example6: sort vector of rectangles by area (descending)” << endl;
  vector<Rectangle> rectangles_vector = {{5, 6}, {3, 3}, {5, 2}, {6, 1}};
  sort(rectangles_vector.begin(), rectangles_vector.end(),
       [](Rectangle &r1, Rectangle &r2) {
         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 2.15: 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)" << 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 2.16: 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
```

2.5 TDD (Test Driven Development)

Τα τελευταία χρόνια η οδηγούμενη από ελέγχους ανάπτυξη (Test Driven Development) έχει αναγνωριστεί ως μια αποδοτική τεχνική ανάπτυξης εφαρμογών. 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 [5]. 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. Το δε παραγοντικό του μηδενός είναι εξ ορισμού η μονάδα. Η πρώτη υλοποίηση είναι λανθασμένη καθώς δεν υπολογίζει σωστά το παραγοντικό του μηδενός αποδίδοντάς του την τιμή μηδέν.

```
#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 2.17: Πρώτη έκδοση της συνάρτησης παραγοντικού και έλεγχοι (tdd1.cpp)

Η μεταγλώττιση και η εκτέλεση του κώδικα γίνεται ως εξής:

```
g++ tdd1.cpp −o tdd1
./tdd1
```

Τα δε αποτελέσματα της εκτέλεσης είναι τα ακόλουθα:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
tdd1 is a Catch v1.10.0 host application.
Run with −? for options

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
Factorials are computed
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
tdd1.cpp:9
...............................................................................

tdd1.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 2.18: Δεύτερη έκδοση της συνάρτησης παραγοντικού και έλεγχοι (tdd2.cpp)

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

2.6 Παραδείγματα

2.6.1 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.

```
#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 2.19: Λύση παραδείγματος 1 (lab02_ex1.cpp)

```
Result 9644
```

2.6.2 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.

```cpp
#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 2.20: Λύση παραδείγματος 2 (lab02_ex2.cpp)

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

2.6.3 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.

```cpp
#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 2.21: Λύση παραδείγματος 3 (lab02_ex3.cpp)

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

2.6.4 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 2.22: 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
```

2.7 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 v ενώ για κάθε αρνητική τιμή που εισάγεται να αναζητείται η απόλυτη τιμή της στο διάνυσμα v. 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).

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 ακολουθώντας τις αρχές του TDD.

Βιβλιογραφία

[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/

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

Εργαστήριο 3

Θεωρητική μελέτη αλγορίθμων, χρονομέτρηση κώδικα, αλγόριθμοι ταξινόμησης και αλγόριθμοι αναζήτησης

3.1 Εισαγωγή

Στο εργαστήριο αυτό παρουσιάζονται ορισμένοι αλγόριθμοι ταξινόμησης και ορισμένοι αλγόριθμοι αναζήτησης. Πρόκειται για μερικούς από τους σημαντικότερους αλγορίθμους στην επιστήμη των υπολογιστών. Σε πρακτικό επίπεδο ένα σημαντικό ποσοστό της επεξεργαστικής ισχύος των υπολογιστών δαπανάται στην ταξινόμηση δεδομένων η οποία διευκολύνει τις αναζητήσεις που ακολουθούν. Επιπλέον, γίνεται αναφορά στη θεωρητική εκτίμηση της απόδοσης ενός αλγορίθμου και στη μέτρηση χρόνου εκτέλεσης κώδικα.

3.2 Θεωρητική και εμπειρική εκτίμηση της απόδοσης αλγορίθμων

Συχνά χρειάζεται να εκτιμηθεί η καταλληλότητα ενός αλγορίθμου για την επίλυση ενός προβλήματος. Ποιοτικά χαρακτηριστικά όπως ο χρόνος εκτέλεσης και ο χώρος που απαιτεί στη μνήμη και στο δίσκο μπορεί να τον καθιστούν υποδεέστερο άλλων αλγορίθμων ή ακόμα και ακατάλληλο για επίλυση του προβλήματος. Γενικά, η απόδοση ενός αλγορίθμου μπορεί να εκτιμηθεί θεωρητικά και εμπειρικά.

3.2.1 Θεωρητική μελέτη αλγορίθμων

Η θεωρητική μελέτη προσδιορίζει την ασυμπτωτική συμπεριφορά του αλγορίθμου, δηλαδή πως θα συμπεριφέρεται ο αλγόριθμος καθώς τα δεδομένα εισόδου αυξάνονται σε μέγεθος προσεγγίζοντας μεγάλες τιμές. Μελετώντας θεωρητικά διαφορετικούς αλγορίθμους που επιτελούν το ίδιο έργο μπορεί να πραγματοποιηθεί σύγκριση μεταξύ τους ακόμα και χωρίς να γραφεί κώδικας που να τους υλοποιεί σε μια συγκεκριμένη γλώσσα προγραμματισμού. Η μέθοδος που έχει επικρατήσει για τη θεωρητική μελέτη αλγορίθμων είναι η ασυμπτωτική ανάλυση και ιδιαίτερα ο συμβολισμός του μεγάλου Ο (Big O notation). Ο συμβολισμός του μεγάλου Ο περιγράφει τη χειρότερη περίπτωση εκτέλεσης και συνήθως αφορά το χρόνο εκτέλεσης ή σπανιότερα το χώρο που απαιτείται από τον αλγόριθμο. Στη συνέχεια θα επιχειρηθεί μια πρακτική παρουσίαση του εν λόγω συμβολισμού μέσω παραδειγμάτων [1].

$O(1)$

Ο συμβολισμός $O(1)$ περιγράφει αλγορίθμους που πάντα εκτελούνται απαιτώντας τον ίδιο χρόνο (ή χώρο), άσχετα με το μέγεθος των δεδομένων με τα οποία τροφοδοτούνται. Ο ακόλουθος κώδικας που είναι ένα παρά-

δειγμα αλγορίθμου με ασυμπτωτική πολυπλοκότητα $O(1)$ επιστρέφει true αν το πρώτο στοιχείο ενός πίνακα ακεραίων είναι άρτιο, αλλιώς επιστρέφει false.

```
bool is_first_element_even(int a[]) {
    return a[0] % 2 == 0;
}
```

$O(n)$

Ο συμβολισμός $O(n)$ περιγράφει αλγορίθμους που ο χρόνος εκτέλεσής τους μεγαλώνει γραμμικά και σε ευθεία αναλογία με το μέγεθος της εισόδου. Ο κώδικας που ακολουθεί επιστρέφει true αν το στοιχείο key υπάρχει στον πίνακα a, N θέσεων. Η ασυμπτωτική του πολυπλοκότητα είναι $O(n)$.

```
bool exists(int a[], int N, int key) {
  for (int i = 0; i < N; i++)
    if (a[i] == key)
      return true;
  return false;
}
```

$O(n^2)$

Ο συμβολισμός $O(n^2)$ περιγράφει αλγορίθμους που ο χρόνος εκτέλεσής τους μεγαλώνει ανάλογα με το τετράγωνο του μεγέθους της εισόδου. Αυτό τυπικά συμβαίνει όταν ο κώδικας περιέχει δύο εντολές επανάληψης την μια μέσα στην άλλη. Ο ακόλουθος κώδικας εξετάζει αν ένας πίνακας έχει διπλότυπα και έχει ασυμπτωτική πολυπλοκότητα $O(n^2)$.

```
bool has_duplicates(int a[], int N) {
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++) {
      if (i == j)
        continue;
      if (a[i] == a[j])
        return true;
    }
  return false;
}
```

$O(2^n)$

Το $O(2^n)$ αφορά αλγορίθμους που ο χρόνος εκτέλεσής τους διπλασιάζεται για κάθε μονάδα αύξησης των δεδομένων εισόδου. Η αύξηση είναι εξαιρετικά απότομη καθιστώντας τον αλγόριθμο μη χρησιμοποιήσιμο παρά μόνο για μικρές τιμές του n. Ένα παράδειγμα αλγορίθμου με ασυμπτωτική πολυπλοκότητα $O(2^n)$ είναι ο αναδρομικός υπολογισμός των αριθμών Fibonacci. Η ακολουθία Fibonacci είναι η ακόλουθη: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... Εξ ορισμού οι δύο πρώτοι όροι είναι το μηδέν και το ένα, ενώ κάθε επόμενος όρος είναι το άθροισμα των δύο προηγούμενων του.

```
int fibo(int n) {
  if (n <= 1)
    return n;
  else
    return fibo(n − 2) + fibo(n − 1);
}
```

$O(log(n))$

Το $O(log(n))$ περιγράφει αλγορίθμους στους οποίους σε κάθε βήμα τους το μέγεθος των δεδομένων που μένει να εξετάσει ο αλγόριθμος μειώνεται στο μισό. Ένα παράδειγμα αλγορίθμου με ασυμπτωτική πολυπλοκότητα $O(log(n))$ είναι ο ακόλουθος κώδικας που επιστρέφει λογική τιμή σχετικά με το εάν υπάρχει το στοιχείο key στον ταξινομημένο πίνακα a, N θέσεων.

```
bool exists_in_sorted(int a[], int N, int key) {
  int left = 0, right = N − 1, m;
  while (left <= right) {
    m = (left + right) / 2;
    if (a[m] == key)
      return true;
    else if (a[m] > key)
      right = m − 1;
    else
      left = m + 1;
  }
  return false;
}
```

Στη συνέχεια παρατίθενται ασυμπτωτικές πολυπλοκότητες αλγορίθμων από τις ταχύτερες προς τις βραδύτερες: $O(1)$, $O(log(n))$, $O(n)$,$O(nlog(n))$, $O(n^2)$, $O(n^3)$, $O(n^4)$, $O(2^n)$, $O(n!)$. Με εξαιρέσεις, επιθυμητές πολυπλοκότητες αλγορίθμων είναι μέχρι και $O(n^2)$.

3.2.2 Εμπειρική μελέτη αλγορίθμων

Η εμπειρική εκτίμηση της απόδοσης ενός προγράμματος έχει να κάνει με τη χρονομέτρησή του για διάφορες περιπτώσεις δεδομένων εισόδου και τη σύγκρισή του με εναλλακτικές υλοποιήσεις προγραμμάτων. Στη συνέχεια θα παρουσιαστούν δύο τρόποι μέτρησης χρόνου εκτέλεσης κώδικα που μπορούν να εφαρμοστούν στη C++.

Μέτρηση χρόνου εκτέλεσης κώδικα με τη συνάρτηση clock()

Ο ακόλουθος κώδικας μετράει το χρόνο που απαιτεί ο υπολογισμός του αθροίσματος των τετραγωνικών ριζών 10.000.000 τυχαίων ακέραιων αριθμών με τιμές στο διάστημα από 0 έως 10.000. Η μέτρηση του χρόνου πραγματοποιείται με τη συνάρτηση clock() η οποία επιστρέφει τον αριθμό από clock ticks που έχουν περάσει από τη στιγμή που το πρόγραμμα ξεκίνησε την εκτέλεση του. Ο αριθμός των δευτερολέπτων που έχουν περάσει προκύπτει διαιρώντας τον αριθμό των clock ticks με τη σταθερά CLOCKS_PER_SEC. Αυτός ο τρόπος υπολογισμού του χρόνου εκτέλεσης έχει "κληρονομηθεί" στη C++ από τη C.

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>

using namespace std;

int main(int argc, char **argv) {
  clock_t t1, t2;
  t1 = clock();
  srand(1821);
  double sum = 0.0;
  for (int i = 1; i <= 10000000; i++) {
    int x = rand() % 10000 + 1;
    sum += sqrt(x);
  }
```

```
17    cout << "The sum is: " << sum << endl;
18
19    t2 = clock();
20    double elapsed_time = (double)(t2 - t1) / CLOCKS_PER_SEC;
21    cout << "Elapsed time " << elapsed_time << " seconds" << endl;
22 }
```

Κώδικας 3.1: Μέτρηση χρόνου εκτέλεσης κώδικα(timing1.cpp)

```
1 The sum is: 6.39952e+008
2 Elapsed time 0.213
```

Μέτρηση χρόνου εκτέλεσης κώδικα με τη χρήση του high_resolution_clock::time_point

Η C++ έχει προσθέσει νέους τρόπους μέτρησης του χρόνου εκτέλεσης προγραμμάτων. Στον ακόλουθο κώδικα παρουσιάζεται ένα παράδειγμα με χρήση time_points.

```
1  #include <chrono>
2  #include <cmath>
3  #include <iostream>
4  #include <random>
5
6  using namespace std;
7  using namespace std::chrono;
8
9  int main() {
10    high_resolution_clock::time_point t1 = high_resolution_clock::now();
11    mt19937 mt(1821);
12    uniform_int_distribution<int> dist(0, 10000);
13    double sum = 0.0;
14    for (int i = 1; i <= 10000000; i++) {
15       int x = dist(mt);
16       sum += sqrt(x);
17    }
18    cout << "The sum is: " << sum << endl;
19    high_resolution_clock::time_point t2 = high_resolution_clock::now();
20    auto duration = duration_cast<microseconds>(t2 - t1).count();
21    cout << "Time elapsed: " << duration << " microseconds "
22         << duration / 1000000.0 << " seconds" << endl;
23 }
```

Κώδικας 3.2: Μέτρηση χρόνου εκτέλεσης κώδικα (timing2.cpp)

```
1 The sum is: 6.6666e+008
2 Time elapsed: 537030 microseconds 0.53703 seconds
```

Θα πρέπει να σημειωθεί ότι κατά τη μεταγλώττιση είναι δυνατό να δοθεί οδηγία προς το μεταγλωττιστή έτσι ώστε να προχωρήσει σε βελτιστοποιήσεις του κώδικα που παράγει που θα οδηγήσουν σε ταχύτερη εκτέλεση. Το flag που χρησιμοποιείται είναι το -O και οι πιθανές τιμές που μπορεί να λάβει είναι: -O0, -O1, -O2, -O3. Καθώς αυξάνεται ο αριθμός δεξιά του -O που χρησιμοποιείται στη μεταγλώττιση ενεργοποιούνται περισσότερες βελτιστοποιήσεις σε βάρος του χρόνου μεταγλώττισης. Στη συνέχεια παρουσιάζονται οι εντολές μεταγλώττισης και ο χρόνος εκτέλεσης για κάθε μια από τις 4 περιπτώσεις του κώδικα .

```
1 g++ timing2.cpp -o timing2a -O0 -std=c++11
2 ./timing2a
3 The sum is: 6.6666e+008
4 Time elapsed: 537030 microseconds 0.53703 seconds
5
6 g++ timing2.cpp -o timing2b -O1 -std=c++11
7 ./timing2b
```

```
The sum is: 6.6666e+008
Time elapsed: 125007 microseconds 0.125007 seconds

g++ timing2.cpp -o timing2c -O2 -std=c++11
./timing2c
The sum is: 6.6666e+008
Time elapsed: 127007 microseconds 0.127007 seconds

g++ timing2.cpp -o timing2d -O3 -std=c++11
./timing2d
The sum is: 6.6666e+008
Time elapsed: 114006 microseconds 0.114006 seconds
```

3.3 Αλγόριθμοι ταξινόμησης

3.3.1 Ταξινόμηση με εισαγωγή

Η ταξινόμηση με εισαγωγή (insertion-sort) λειτουργεί δημιουργώντας μια ταξινομημένη λίστα στο αριστερό άκρο των δεδομένων και επαναληπτικά τοποθετεί το στοιχείο το οποίο βρίσκεται δεξιά της ταξινομημένης λίστας στη σωστή θέση σε σχέση με τα ήδη ταξινομημένα στοιχεία. Ο αλγόριθμος ταξινόμησης με εισαγωγή καθώς και η κλήση του από κύριο πρόγραμμα για την αύξουσα ταξινόμηση ενός πίνακα 10 θέσεων παρουσιάζεται στον κώδικα που ακολουθεί.

```cpp
template <class T> void insertion_sort(T a[], int n) {
  for (int i = 1; i < n; i++) {
    T key = a[i];
    int j = i - 1;
    while ((j >= 0) && (key < a[j])) {
      a[j + 1] = a[j];
      j--;
    }
    a[j + 1] = key;
  }
}
```

Κώδικας 3.3: Ο αλγόριθμος ταξινόμησης με εισαγωγή (insertion_sort.cpp)

```cpp
#include "insertion_sort.cpp"
#include <iostream>

using namespace std;

int main(int argc, char **argv) {
  int a[] = {45, 32, 16, 11, 7, 18, 21, 16, 11, 15};
  cout << "Sort using insertion sort" << endl;
  insertion_sort(a, 10);
  for (int i = 0; i < 10; i++)
    cout << a[i] << " ";
}
```

Κώδικας 3.4: Κλήση της συνάρτησης insertion_sort (sort1.cpp)

```
Sort using insertion sort
7 11 11 15 16 16 18 21 32 45
```

3.3.2 Ταξινόμηση με συγχώνευση

Η ταξινόμηση με συγχώνευση (merge-sort) είναι αναδρομικός αλγόριθμος και στηρίζεται στη συγχώνευση ταξινομημένων υποακολουθιών έτσι ώστε να δημιουργούνται νέες ταξινομημένες υποακολουθίες. Μια υλοποί-

ηση του κώδικα ταξινόμησης με συγχώνευση παρουσιάζεται στη συνέχεια.

```
template <class T> void merge(T a[], int l, int m, int r) {
  T la[m − l + 1];
  T ra[r − m];
  for (int i = 0; i < m − l + 1; i++)
    la[i] = a[l + i];
  for (int i = 0; i < r − m; i++)
    ra[i] = a[m + 1 + i];
  int i = 0, j = 0, k = l;
  while ((i < m − l + 1) && (j < r − m)) {
    if (la[i] < ra[j]) {
      a[k] = la[i];
      i++;
    } else {
      a[k] = ra[j];
      j++;
    }
    k++;
  }
  if (i == m − l + 1) {
    while (j < r − m) {
      a[k] = ra[j];
      j++;
      k++;
    }
  } else {
    while (i < m − l + 1) {
      a[k] = la[i];
      i++;
      k++;
    }
  }
}

template <class T> void merge_sort(T a[], int l, int r) {
  if (l < r) {
    int m = (l + r) / 2;
    merge_sort(a, l, m);
    merge_sort(a, m + 1, r);
    merge(a, l, m, r);
  }
}

template <class T> void merge_sort(T a[], int N) { merge_sort(a, 0, N − 1); }
```

Κώδικας 3.5: Ο αλγόριθμος ταξινόμησης με συγχώνευση (merge_sort.cpp)

```
#include ”merge_sort.cpp”
#include <iostream>

using namespace std;

int main(int argc, char **argv) {
  int a[] = {45, 32, 16, 11, 7, 18, 21, 16, 11, 15};
  cout << ”Sort using merge sort” << endl;
  merge_sort(a, 10);
  for (int i = 0; i < 10; i++)
    cout << a[i] << ” ”;
}
```

Κώδικας 3.6: Κλήση της συνάρτησης merge_sort (sort2.cpp)

```
Sort using merge sort
7 11 11 15 16 16 18 21 32 45
```

3.3.3 Γρήγορη ταξινόμηση

Ο κώδικας της γρήγορης ταξινόμησης παρουσιάζεται στη συνέχεια. Πρόκειται για κώδικα ο οποίος καλείται αναδρομικά σε υποακολουθίες των δεδομένων και σε κάθε κλήση επιλέγει ένα στοιχείο (pivot) και διαχωρίζει τα υπόλοιπα στοιχεία έτσι ώστε αριστερά να είναι τα στοιχεία που είναι μικρότερα του pivot και δεξιά αυτά τα οποία είναι μεγαλύτερα.

```cpp
#include <utility> // std::swap

template <class T> int partition(T a[], int l, int r) {
  int p = l;
  int i = l + 1;
  for (int j = l + 1; j <= r; j++) {
    if (a[j] < a[p]) {
      std::swap(a[j], a[i]);
      i++;
    }
  }
  std::swap(a[p], a[i − 1]);
  return i − 1;
}

template <class T> void quick_sort(T a[], int l, int r) {
  if (l >= r)
    return;
  else {
    int p = partition(a, l, r);
    quick_sort(a, l, p − 1);
    quick_sort(a, p + 1, r);
  }
}

template <class T> void quick_sort(T a[], int N) { quick_sort(a, 0, N − 1); }
```

Κώδικας 3.7: Ο αλγόριθμος γρήγορης ταξινόμησης (quick_sort.cpp)

```cpp
#include ”quick_sort.cpp”
#include <iostream>

using namespace std;

int main(int argc, char **argv) {
  int a[] = {45, 32, 16, 11, 7, 18, 21, 16, 11, 15};
  cout << ”Sort using quick sort” << endl;
  quick_sort(a, 10);
  for (int i = 0; i < 10; i++)
    cout << a[i] << ” ”;
}
```

Κώδικας 3.8: Κλήση της συνάρτησης quick_sort (sort3.cpp)

```
Sort using quick sort
7 11 11 15 16 16 18 21 32 45
```

3.3.4 Ταξινόμηση κατάταξης

ο αλγόριθμος ταξινόμησης κατάταξης (rank-sort) λειτουργεί ως εξής: Για κάθε στοιχείο του δεδομένου πίνακα a που επιθυμούμε να ταξινομήσουμε υπολογίζεται μια τιμή κατάταξης (rank). Η τιμή κατάταξης ενός στοιχείου του πίνακα είναι το πλήθος των μικρότερων από αυτό στοιχείων συν το πλήθος των ίσων με αυτό στοιχείων που έχουν μικρότερο δείκτη σε σχέση με αυτό το στοιχείο (δηλαδή βρίσκονται αριστερά του). Δηλαδή ισχύει ότι η τιμή κατάταξης ενός στοιχείου x του πίνακα είναι ίση με το άθροισμα 2 όρων: του πλήθους των μικρότερων στοιχείων του x από όλο τον πίνακα και του πλήθους των ίσων με το x στοιχείων που έχουν μικρότερο δείκτη σε σχέση με το x. Για παράδειγμα στην ακολουθία τιμών a=[44, 21, 78, 16, 56, 21] θα πρέπει να δημιουργηθεί ένας νέος πίνακας r = [3, 1, 5, 0, 4, 2]. Έχοντας υπολογίσει τον πίνακα r θα πρέπει τα στοιχεία του a να αντιγραφούν σε ένα νέο βοηθητικό πίνακα temp έτσι ώστε κάθε τιμή που υπάρχει στον πίνακα r να λειτουργεί ως δείκτης για το που πρέπει να τοποθετηθεί το αντίστοιχο στοιχείο του a στον πίνακα temp. Τέλος θα πρέπει να αντιγραφεί ο πίνακας temp στον πίνακα a. Στη συνέχεια παρουσιάζεται ο κώδικας του αλγορίθμου rank-sort. Παρουσιάζονται δύο υλοποιήσεις. Η πρώτη υλοποίηση (rank_sort) αφορά τον αλγόριθμο όπως έχει περιγραφεί παραπάνω ενώ η δεύτερη (rank_sort_in_place) είναι από το βιβλίο “Δομές Δεδομένων, Αλγόριθμοι και Εφαρμογές στη C++ του Sartaj Sahnii” [2] και δεν απαιτεί τη χρήση του βοηθητικού πίνακα temp, συνεπώς είναι αποδοτικότερος.

```
#include <iostream>
#include <utility> // swap

using namespace std;

template <class T> void rank_sort(T a[], int n) {
   int r[n] = {0};
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         if (a[j] < a[i] || (a[j] == a[i] && j < i))
            r[i]++;
   int temp[n];
   for (int i = 0; i < n; i++)
      temp[r[i]] = a[i];
   for (int i = 0; i < n; i++)
      a[i] = temp[i];
}

template <class T> void rank_sort_in_place(T a[], int n) {
   int r[n] = {0};
   for (int i = 0; i < n; i++)
      for (int j = 0; j < i; j++)
         if (a[j] <= a[i])
            r[i]++;
         else
            r[j]++;
   for (int i = 0; i < n; i++)
      while (r[i] != i) {
         int t = r[i];
         swap(a[i], a[t]);
         swap(r[i], r[t]);
      }
}

int main(int argc, char **argv) {
   int a[] = {45, 32, 16, 11, 7, 18, 21, 16, 11, 15};
   cout << ”Sort using rank sort” << endl;
   rank_sort(a, 10);
   for (int i = 0; i < 10; i++)
      cout << a[i] << ” ”;
```

```
41    cout << endl << "Sort using rank sort (in place)" << endl;
42    int b[] = {45, 32, 16, 11, 7, 18, 21, 16, 11, 15};
43    rank_sort_in_place(b, 10);
44    for (int i = 0; i < 10; i++)
45       cout << b[i] << " ";
46 }
```

Κώδικας 3.9: Ο αλγόριθμος ταξινόμησης κατάταξης (sort4.cpp)

```
1 Sort using rank sort
2 7 11 11 15 16 16 18 21 32 45
3 Sort using rank sort (in place)
4 7 11 11 15 16 16 18 21 32 45
```

3.3.5 Σταθερή ταξινόμηση (stable sorting)

Ένας αλγόριθμος ταξινόμησης είναι stable αν τα στοιχεία με την ίδια τιμή εμφανίζονται με την ίδια σειρά με την οποία βρισκόταν στην αρχική λίστα και στην ταξινομημένη λίστα [3]. Για παράδειγμα, αν σε μια λίστα εγγραφών φοιτητών/φοιτητριών (όνομα - βαθμός) πραγματοποιηθεί σταθερή ταξινόμηση με βάση το βαθμό, θα πρέπει οι φοιτητές με τον ίδιο βαθμό να μην αλλάξουν σειρά μεταξύ τους σε σχέση με τη σειρά που είχαν στην αρχική λίστα. Ο ακόλουθος κώδικας διαβάζει τα στοιχεία υποθετικών ατόμων από το αρχείο students20.txt και εφαρμόζοντας αλγορίθμους ταξινόμησης που παρουσιάστηκαν νωρίτερα καθώς και αλγορίθμους ταξινόμησης της STL παρουσιάζει τα δεδομένα ταξινομημένα.

```
1  #include "insertion_sort.cpp"
2  #include "merge_sort.cpp"
3  #include "quick_sort.cpp"
4  #include <algorithm>
5  #include <fstream>
6  #include <iostream>
7  #include <sstream>
8  #include <vector>
9
10 using namespace std;
11
12 struct student {
13    string name;
14    int grade;
15    bool operator<(student other) const { return grade > other.grade; }
16 };
17
18 void print_students(student a[], int N) {
19    for (int i = 0; i < N; i++)
20       cout << a[i].name << "-" << a[i].grade << " ";
21    cout << endl;
22 }
23
24 int main(void) {
25    vector<student> v;
26    fstream filestr;
27    string buffer;
28
29    filestr.open("students20.txt");
30    if (!filestr.is_open()) {
31       cerr << "File not found" << endl;
32       exit(-1);
33    }
34    while (getline(filestr, buffer)) {
35       stringstream ss(buffer);
```

```
36      student st;
37      ss >> st.name;
38      ss >> st.grade;
39      v.push_back(st);
40    }
41    filestr.close();
42
43    int N = v.size();
44    cout << "ORIGINAL LIST:" << endl;
45    for (int i = 0; i < N; i++)
46      cout << v[i].name << "-" << v[i].grade << " ";
47    cout << endl;
48
49    cout << "INSERTION SORT: (STABLE)" << endl;
50    student *a = new student[N];
51    for (int i = 0; i < N; i++)
52      a[i] = v[i];
53    insertion_sort(a, N);
54    print_students(a, N);
55    delete[] a;
56
57    cout << "MERGE SORT: (NON STABLE)" << endl;
58    a = new student[N];
59    for (int i = 0; i < N; i++)
60      a[i] = v[i];
61    merge_sort(a, N);
62    print_students(a, N);
63    delete[] a;
64
65    cout << "QUICK SORT: (NON STABLE)" << endl;
66    a = new student[N];
67    for (int i = 0; i < N; i++)
68      a[i] = v[i];
69    quick_sort(a, N);
70    print_students(a, N);
71    delete[] a;
72
73    cout << "STL SORT: (NON STABLE)" << endl;
74    a = new student[N];
75    for (int i = 0; i < N; i++)
76      a[i] = v[i];
77    sort(a, a + N);
78    print_students(a, N);
79    delete[] a;
80
81    cout << "STL STABLESORT: (STABLE)" << endl;
82    a = new student[N];
83    for (int i = 0; i < N; i++)
84      a[i] = v[i];
85    stable_sort(a, a + N);
86    print_students(a, N);
87    delete[] a;
88  }
```

Κώδικας 3.10: Σταθερή ταξινόμηση (stable.cpp)

```
1  ORIGINAL LIST:
2  apostolis-5 nikos-5 petros-7 maria-2 kostas-5 giannis-5 sofia-1 dimitra-10 kiki-5 aristea-9 christos-7 niki-4 katerina-9 giorgos-8
       lefteris-4 efthymios-10 iordanis-9 alexis-5 anna-5 georgia-4
3  INSERTION SORT:
4  dimitra-10 efthymios-10 aristea-9 katerina-9 iordanis-9 giorgos-8 petros-7 christos-7 apostolis-5 nikos-5 kostas-5 giannis-5 kiki-5
       alexis-5 anna-5 niki-4 lefteris-4 georgia-4 maria-2 sofia-1
```

```
MERGE SORT:
efthymios-10 dimitra-10 iordanis-9 katerina-9 aristea-9 giorgos-8 christos-7 petros-7 anna-5 alexis-5 kiki-5 giannis-5 kostas-5 nikos-5 apostolis-5 georgia-4 lefteris-4 niki-4 maria-2 sofia-1
QUICK SORT:
efthymios-10 dimitra-10 iordanis-9 aristea-9 katerina-9 giorgos-8 christos-7 petros-7 apostolis-5 anna-5 kostas-5 giannis-5 nikos-5 kiki-5 alexis-5 georgia-4 niki-4 lefteris-4 maria-2 sofia-1
STL SORT:
efthymios-10 dimitra-10 iordanis-9 katerina-9 aristea-9 giorgos-8 petros-7 christos-7 nikos-5 anna-5 alexis-5 kiki-5 giannis-5 kostas-5 apostolis-5 niki-4 lefteris-4 georgia-4 maria-2 sofia-1
STL STABLESORT:
dimitra-10 efthymios-10 aristea-9 katerina-9 iordanis-9 giorgos-8 petros-7 christos-7 apostolis-5 nikos-5 kostas-5 giannis-5 kiki-5 alexis-5 anna-5 niki-4 lefteris-4 georgia-4 maria-2 sofia-1
```

3.4 Αλγόριθμοι αναζήτησης

3.4.1 Σειριακή αναζήτηση

Η σειριακή αναζήτηση είναι ο απλούστερος αλγόριθμος αναζήτησης. Εξετάζει τα στοιχεία ένα προς ένα στη σειρά μέχρι να βρει το στοιχείο που αναζητείται. Το πλεονέκτημα του αλγορίθμου είναι ότι μπορεί να εφαρμοστεί σε μη ταξινομημένους πίνακες.

```cpp
#include <iostream>

using namespace std;

template <class T> int sequential_search(T a[], int n, T key) {
  for (int i = 0; i < n; i++)
    if (a[i] == key)
      return i;
  return -1;
}

int main(int argc, char **argv) {
  int a[] = {5, 11, 45, 23, 10, 17, 32, 8, 9, 4};
  int key;
  cout << "Search for: ";
  cin >> key;
  int pos = sequential_search(a, 10, key);
  if (pos == -1)
    cout << "Not found" << endl;
  else
    cout << "Found at position " << pos << endl;
}
```

Κώδικας 3.11: Ο αλγόριθμος σειριακής αναζήτησης (search1.cpp)

```
Search for: 45
Found at position 2
```

3.4.2 Δυαδική αναζήτηση

Η δυαδική αναζήτηση μπορεί να εφαρμοστεί μόνο σε ταξινομημένα δεδομένα. Διαιρεί επαναληπτικά την ακολουθία σε 2 υποακολουθίες και απορρίπτει την ακολουθία στην οποία συμπεραίνει ότι δεν μπορεί να βρεθεί το στοιχείο.

```cpp
// non-recursive implementation
template <class T> int binary_search(T a[], int l, int r, T key) {
  while (l <= r) {
    int m = (l + r) / 2;
```

```
    if (a[m] == key)
      return m;
    else if (a[m] < key)
      l = m + 1;
    else
      r = m - 1;
  }
  return -1;
}

template <class T> int binary_search(T a[], int n, T key) {
  return binary_search(a, 0, n - 1, key);
}

// recursive implementation
template <class T> int binary_search_r(T a[], int l, int r, T key) {
  int m = (l + r) / 2;
  if (l > r)
    return -1;
  else if (a[m] == key)
    return m;
  else if (key < a[m])
    return binary_search_r(a, l, m - 1, key);
  else
    return binary_search_r(a, m + 1, r, key);
}

template <class T> int binary_search_r(T a[], int n, T key) {
  return binary_search_r(a, 0, n - 1, key);
}
```

Κώδικας 3.12: Ο αλγόριθμος δυαδικής αναζήτησης σε μη αναδρομική και σε αναδρομική έκδοση (binary_search.cpp)

```
#include "binary_search.cpp"
#include <iostream>

using namespace std;

int main(int argc, char **argv) {
    int key, a[] = {11, 45, 53, 60, 67, 72, 88, 91, 94, 98}, N = sizeof(a) / sizeof(int);
    for (int i = 0; i < N; i++)
        cout << "a[" << i << "]=" << a[i] << " ";
    cout << endl;
    cout << "Search for: ";
    cin >> key;
    cout << "Using non recursive algorithm (binary search)" << endl;
    int pos = binary_search(a, N, key);
    if (pos == -1)
        cout << "Not found" << endl;
    else
        cout << "Found at position " << pos << endl;

    cout << "Using recursive algorithm (binary search)" << endl;
    pos = binary_search_r(a, N, key);
    if (pos == -1)
        cout << "Not found" << endl;
    else
        cout << "Found at position " << pos << endl;
}
```

Κώδικας 3.13: Κλήση της συνάρτησης binary_search (search2.cpp)

```
a[0]=11 a[1]=45 a[2]=53 a[3]=60 a[4]=67 a[5]=72 a[6]=88 a[7]=91 a[8]=94 a[9]=98
Search for: 88
Using non recursive algorithm (binary search)
Found at position 6
Using recursive algorithm (binary search)
Found at position 6
```

3.4.3 Αναζήτηση με παρεμβολή

Η αναζήτηση με παρεμβολή (interpolation-search) είναι μια παραλλαγή της δυαδικής αναζήτησης και μπορεί να εφαρμοστεί μόνο σε ταξινομημένα δεδομένα. Αντί να χρησιμοποιηθεί η τιμή 50% για να διαχωριστούν τα δεδομένα σε 2 ισομεγέθεις λίστες (όπως συμβαίνει στη δυαδική αναζήτηση) υπολογίζεται μια τιμή η οποία εκτιμάται ότι θα οδηγήσει πλησιέστερα στο στοιχείο που αναζητείται. Αν l είναι ο δείκτης του αριστερότερου στοιχείου της ακολουθίας και r ο δείκτης του δεξιότερου στοιχείου της ακολουθίας τότε υπολογίζεται ο συντελεστής c = (key−a[l])/(a[r]−a[l]) όπου key είναι το στοιχείο προς αναζήτηση και a είναι η ακολουθία τιμών στην οποία αναζητείται το key. Η ακολουθία των δεδομένων διαχωρίζεται με βάση τον συντελεστή c σε δύο υποακολουθίες. Η διαδικασία επαναλαμβάνεται ανάλογα με τη δυαδική αναζήτηση. Στη συνέχεια παρουσιάζεται ο κώδικας της αναζήτησης με παρεμβολή.

```cpp
template <class T> int interpolation_search(T a[], int l, int r, T key) {
  int m;
  if (l > r)
    return −1;
  else if (l == r)
    m = l;
  else {
    double c = (double)(key − a[l]) / (double)(a[r] − a[l]);
    if ((c < 0) || (c > 1))
      return −1;
    m = (int)(l + (r − l) * c);
  }
  if (a[m] == key)
    return m;
  else if (key < a[m])
    return interpolation_search(a, l, m − 1, key);
  else
    return interpolation_search(a, m + 1, r, key);
}

template <class T> int interpolation_search(T a[], int n, T key) {
  return interpolation_search(a, 0, n − 1, key);
}
```

Κώδικας 3.14: Ο αλγόριθμος αναζήτησης με παρεμβολή (interpolation_search.cpp)

```cpp
#include ”interpolation_search.cpp”
#include <iostream>

using namespace std;

int main(int argc, char **argv) {
  int key, a[] = {11, 45, 53, 60, 67, 72, 88, 91, 94, 98}, N = sizeof(a) / sizeof(int);
  for (int i = 0; i < N; i++)
      cout << ”a[” << i << ”]=” << a[i] << ” ”;
   cout << endl;
```

```
11    cout << "Search for: ";
12    cin >> key;
13    int pos = interpolation_search(a, 10, key);
14    if (pos == -1)
15       cout << "Not found" << endl;
16    else
17       cout << "Found at position " << pos << endl;
18 }
```

Κώδικας 3.15: Κλήση της συνάρτησης interpolation_search search3.cpp

```
1 a[0]=11 a[1]=45 a[2]=53 a[3]=60 a[4]=67 a[5]=72 a[6]=88 a[7]=91 a[8]=94 a[9]=98
2 Search for: 91
3 Found at position 7
```

3.5 Παραδείγματα

3.5.1 Παράδειγμα 1

Γράψτε πρόγραμμα που να συγκρίνει τους χρόνους εκτέλεσης των αλγορίθμων ταξινόμησης insertion-sort, merge-sort, quick-sort καθώς και του αλγορίθμου ταξινόμησης της βιβλιοθήκης STL (συνάρτηση sort). Η σύγκριση να αφορά τυχαία δεδομένα τύπου float με τιμές στο διάστημα από -1.000 έως 1.000. Τα μεγέθη των πινάκων που θα ταξινομηθούν να είναι 5.000, 10.000, 20.0000, 40.000, 80.000, 160.0000 και 320.000 αριθμών.

```
1  #include "insertion_sort.cpp"
2  #include "merge_sort.cpp"
3  #include "quick_sort.cpp"
4  #include <algorithm>
5  #include <chrono>
6  #include <iomanip>
7  #include <iostream>
8  #include <random>
9
10 using namespace std;
11 using namespace std::chrono;
12
13 void benchmark_sort_algorithm(string alg) {
14    mt19937 mt(1821);
15    uniform_real_distribution<float> dist(-1000, 1000);
16
17    int sizes[] = {5000, 10000, 20000, 40000, 80000, 160000, 320000};
18    for (int i = 0; i < 7; i++) {
19       int N = sizes[i];
20       float *a = new float[N];
21       for (int i = 0; i < N; i++)
22          a[i] = dist(mt);
23
24       auto t1 = high_resolution_clock::now();
25       if (alg.compare("merge-sort") == 0)
26          merge_sort(a, N);
27       else if (alg.compare("quick-sort") == 0)
28          quick_sort(a, N);
29       else if (alg.compare("STL-sort") == 0)
30          sort(a, a + N);
31       else if (alg.compare("insertion-sort") == 0)
32          insertion_sort(a, N);
33       auto t2 = high_resolution_clock::now();
34
```

```
35      auto elapsed_time = duration_cast<milliseconds>(t2 − t1).count();
36      cout << fixed << setprecision(3);
37      cout << "Elapsed time " << alg << "\t" << sizes[i] << "\t" << elapsed_time
38              << " milliseconds" << endl;
39      delete[] a;
40    }
41  }
42
43  int main(int argc, char **argv) {
44    benchmark_sort_algorithm("insertion−sort");
45    cout << "####################################" << endl;
46    benchmark_sort_algorithm("merge−sort");
47    cout << "####################################" << endl;
48    benchmark_sort_algorithm("quick−sort");
49    cout << "####################################" << endl;
50    benchmark_sort_algorithm("STL−sort");
51    cout << "####################################" << endl;
52  }
```

Κώδικας 3.16: Σύγκριση χρόνου εκτέλεσης αλγορίθμων ταξινόμησης (lab03_ex1.cpp)

```
1  Elapsed time insertion−sort 5000 46 milliseconds
2  Elapsed time insertion−sort 10000 167 milliseconds
3  Elapsed time insertion−sort 20000 658 milliseconds
4  Elapsed time insertion−sort 40000 2595 milliseconds
5  Elapsed time insertion−sort 80000 10377 milliseconds
6  Elapsed time insertion−sort 160000 41441 milliseconds
7  Elapsed time insertion−sort 320000 167593 milliseconds
8  ####################################
9  Elapsed time merge−sort 5000 1 milliseconds
10 Elapsed time merge−sort 10000 3 milliseconds
11 Elapsed time merge−sort 20000 7 milliseconds
12 Elapsed time merge−sort 40000 14 milliseconds
13 Elapsed time merge−sort 80000 32 milliseconds
14 Elapsed time merge−sort 160000 60 milliseconds
15 Elapsed time merge−sort 320000 125 milliseconds
16 ####################################
17 Elapsed time quick−sort 5000 1 milliseconds
18 Elapsed time quick−sort 10000 3 milliseconds
19 Elapsed time quick−sort 20000 5 milliseconds
20 Elapsed time quick−sort 40000 10 milliseconds
21 Elapsed time quick−sort 80000 24 milliseconds
22 Elapsed time quick−sort 160000 47 milliseconds
23 Elapsed time quick−sort 320000 105 milliseconds
24 ####################################
25 Elapsed time STL−sort 5000 1 milliseconds
26 Elapsed time STL−sort 10000 2 milliseconds
27 Elapsed time STL−sort 20000 4 milliseconds
28 Elapsed time STL−sort 40000 8 milliseconds
29 Elapsed time STL−sort 80000 17 milliseconds
30 Elapsed time STL−sort 160000 37 milliseconds
31 Elapsed time STL−sort 320000 79 milliseconds
32 ####################################
```

3.5.2 Παράδειγμα 2

Γράψτε πρόγραμμα που να συγκρίνει τους χρόνους εκτέλεσης των αλγορίθμων αναζήτησης binary-search, interpolation-search και του αλγορίθμου αναζήτησης της βιβλιοθήκης STL binary_search για ταξινομημένα ακέραια δεδομένα με τιμές στο διάστημα από 0 έως 10.000.000. Η σύγκριση να εξετάζει τα ακόλουθα μεγέθη πινάκων 5.000, 10.000, 20.000, 40.000, 80.000, 160.000 και 320.000 αριθμών. Οι χρόνοι εκτέλεσης να αφορούν τους συνολικούς χρόνους που απαιτούνται έτσι ώστε να αναζητηθούν 100.000 τυχαίες τιμές με καθένα από τους αλγορίθμους.

```cpp
#include "binary_search.cpp"
#include "interpolation_search.cpp"
#include <algorithm>
#include <ctime>
#include <iomanip>
#include <iostream>
#include <random>
#include <chrono>

using namespace std;
using namespace std::chrono;

void benchmark_search_algorithm(string alg) {
  clock_t t1, t2;
  mt19937 mt(1729);
  uniform_int_distribution<int> dist(0, 1000000);
  int M = 100000;
  int keys[M];
  for (int i = 0; i < M; i++) {
    keys[i] = dist(mt);
  }
  int sizes[] = {5000, 10000, 20000, 40000, 80000, 160000, 320000};
  for (int i = 0; i < 7; i++) {
    int N = sizes[i];
    int *a = new int[N];
    for (int i = 0; i < N; i++)
      a[i] = dist(mt);
    sort(a, a + N);

    auto t1 = high_resolution_clock::now();
    int c = 0;
    if (alg.compare("binary-search") == 0) {
      for (int j = 0; j < M; j++)
        if (binary_search(a, 0, N - 1, keys[j]) != -1)
          c++;
    } else if (alg.compare("interpolation-search") == 0) {
      for (int j = 0; j < M; j++)
        if (interpolation_search(a, 0, N - 1, keys[j]) != -1)
          c++;
    } else if (alg.compare("STL-binary-search") == 0)
      for (int j = 0; j < M; j++)
        if (binary_search(a, a + N, keys[j]))
          c++;
    auto t2 = high_resolution_clock::now();

    auto elapsed_time = duration_cast<milliseconds>(t2 - t1).count();
    cout << fixed << setprecision(3);
    cout << "Elapsed time " << alg << "\t" << sizes[i] << "\t" << elapsed_time
         << " milliseconds" << endl;
    delete[] a;
  }
}

int main(int argc, char **argv) {
  benchmark_search_algorithm("binary-search");
  cout << "#####################################" << endl;
  benchmark_search_algorithm("interpolation-search");
  cout << "#####################################" << endl;
  benchmark_search_algorithm("STL-binary-search");
  cout << "#####################################" << endl;
```

```
61 }
```

Κώδικας 3.17: Σύγκριση χρόνου εκτέλεσης αλγορίθμων ταξινόμησης (lab03_ex2.cpp)

```
1  Elapsed time binary−search 5000 20 milliseconds
2  Elapsed time binary−search 10000 20 milliseconds
3  Elapsed time binary−search 20000 22 milliseconds
4  Elapsed time binary−search 40000 24 milliseconds
5  Elapsed time binary−search 80000 31 milliseconds
6  Elapsed time binary−search 160000 30 milliseconds
7  Elapsed time binary−search 320000 35 milliseconds
8  ####################################
9  Elapsed time interpolation−search 5000 13 milliseconds
10 Elapsed time interpolation−search 10000 14 milliseconds
11 Elapsed time interpolation−search 20000 14 milliseconds
12 Elapsed time interpolation−search 40000 15 milliseconds
13 Elapsed time interpolation−search 80000 15 milliseconds
14 Elapsed time interpolation−search 160000 17 milliseconds
15 Elapsed time interpolation−search 320000 19 milliseconds
16 ####################################
17 Elapsed time STL−binary−search 5000 30 milliseconds
18 Elapsed time STL−binary−search 10000 33 milliseconds
19 Elapsed time STL−binary−search 20000 35 milliseconds
20 Elapsed time STL−binary−search 40000 39 milliseconds
21 Elapsed time STL−binary−search 80000 41 milliseconds
22 Elapsed time STL−binary−search 160000 43 milliseconds
23 Elapsed time STL−binary−search 320000 51 milliseconds
24 ####################################
```

3.6 Ασκήσεις

1. Ο αλγόριθμος bogosort αναδιατάσσει τυχαία τις τιμές ενός πίνακα μέχρι να προκύψει μια ταξινομημένη διάταξη. Γράψτε ένα πρόγραμμα που να υλοποιεί τον αλγόριθμο bogosort για την ταξινόμηση ενός πίνακα ακεραίων τιμών. Χρησιμοποιήστε τη συνάρτηση shuffle.

2. Να υλοποιηθεί ο αλγόριθμος ταξινόμησης με επιλογή (selection sort) και να εφαρμοστεί για τη ταξινόμηση ενός πίνακα πραγματικών τιμών, ενός πίνακα ακεραίων και ενός πίνακα με λεκτικά (δηλαδή να γίνουν τρεις κλήσεις του αλγορίθμου). Ο αλγόριθμος ταξινόμησης με επιλογή ξεκινά εντοπίζοντας το μικρότερο στοιχείο και το τοποθετεί στη πρώτη θέση. Συνεχίζει, ακολουθώντας την ίδια διαδικασία χρησιμοποιώντας το τμήμα του πίνακα που δεν έχει ταξινομηθεί ακόμα.

3. Γράψτε μια αναδρομική έκδοση του κώδικα για την ταξινόμηση με επιλογή (selection sort).

4. Υλοποιήστε τον αλγόριθμο radix sort (https://en.wikipedia.org/wiki/Radix_sort) σε C++ και χρησιμοποιήστε τον για την ταξινόμηση ενός μεγάλου πίνακα ακεραίων.

Βιβλιογραφία

[1] A beginners guide to Big O notation, https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation

[2] Sartaj Sahnii. Δομές Δεδομένων, Αλγόριθμοι και Εφαρμογές στη C++. Εκδόσεις Τζιόλα, 2004.

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