

## Lab2

Ωραία Έχουμε λοιπόν το **εργαστήριο 2**, που βασίζεται στο προηγούμενο (το οποίο ήδη μου έδωσες), και σύμφωνα με το PDF

OO-lab\_02-instructions

, ζητάει να επεκτείνεις τις κλάσεις **Protein** και **Gene** ώστε να:

- χρησιμοποιούν **std::string** αντί για **char\***,
- έχουν **constructors** και **destructors** που εκτυπώνουν μηνύματα,
- χρησιμοποιούν **λίστες** αρχικοποίησης,
- (προαιρετικά) να υπάρχει **μετρητής ενεργών αντικειμένων**,
- να παρουσιαστεί η **διαφορά** μεταξύ αυτόματης και δυναμικής **δημιουργίας** αντικειμένων (`new / delete`).

---

Παρακάτω είναι η ολοκληρωμένη, σωστή και πλήρως σχολιασμένη λύση:

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

using namespace std;

// -----
// Καθολικοί μετρητές ενεργών αντικειμένων (όχι static μέλη ακόμα)
// -----

int protein_count = 0;
int gene_count = 0;

// -----
// Κλάση Protein
// -----
```

```
class Protein {  
private:  
    string id;  
    string name;  
    string sequence;  
  
public:  
    // --- Constructors ---  
    // Default constructor  
    Protein()  
        : id("N/A"), name("Unnamed"), sequence("") // λίστα αρχικοποίησης  
    {  
        protein_count++;  
        cout << "[Protein] Δημιουργήθηκε αντικείμενο (default). Σύνολο: "  
            << protein_count << endl;  
    }  
  
    // Constructor με παραμέτρους  
    Protein(const string& in_id, const string& in_name, const string& in_seq)  
        : id(in_id), name(in_name), sequence(in_seq)  
    {  
        protein_count++;  
        cout << "[Protein] Δημιουργήθηκε αντικείμενο με δεδομένα: "  
            << name << " (" << id << "). Σύνολο: " << protein_count << endl;  
    }  
  
    // --- Destructor ---
```

```

~Protein() {

    protein_count--;

    cout << "[Protein] Καταστράφηκε αντικείμενο: "
        << name << " (" << id << "). Εναπομέίναντα: "
        << protein_count << endl;

}

// --- Setters ---

void setID(const string& in_id) { id = in_id; }

void setName(const string& in_name) { name = in_name; }

void setSequence(const string& in_seq) { sequence = in_seq; }

// --- Getters ---

string getID() const { return id; }

string getName() const { return name; }

string getSequence() const { return sequence; }

// --- Άλλες μέθοδοι ---

void describe() const {

    cout << "Protein " << name << " (" << id << "): " << sequence << endl;
}

int length() const {

    return sequence.length();
}

};


```

```

// -----
// Κλάση Gene

// -----
class Gene {

private:

    const string organism; // σταθερό μέλος (πρέπει να αρχικοποιηθεί στη λίστα)

    string id;
    string name;
    string chrom;
    int start;
    int end;
    char strand;

public:

    // --- Constructors ---

    // Default constructor
    Gene()
        : organism("Homo sapiens"), id("N/A"), name("Unnamed"),
          chrom("chr?"), start(0), end(0), strand('+')

    {
        gene_count++;

        cout << "[Gene] Δημιουργήθηκε αντικείμενο (default). Σύνολο: "
            << gene_count << endl;
    }

    // Constructor με παραμέτρους

```

```

Gene(const string& in_id, const string& in_name, const string& in_chrom,
      int in_start, int in_end, char in_strand,
      const string& in_organism = "Homo sapiens")
: organism(in_organism), id(in_id), name(in_name),
  chrom(in_chrom), start(in_start), end(in_end), strand(in_strand)

{
    gene_count++;

    cout << "[Gene] Δημιουργήθηκε αντικείμενο: " << name
        << " (" << id << "). Σύνολο: " << gene_count << endl;
}

// --- Destructor ---

~Gene() {
    gene_count--;
    cout << "[Gene] Καταστράφηκε αντικείμενο: " << name
        << " (" << id << "). Εναπομέίναντα: " << gene_count << endl;
}

// --- Setters ---

void setID(const string& in_id) { id = in_id; }

void setName(const string& in_name) { name = in_name; }

void setChrom(const string& in_chrom) { chrom = in_chrom; }

void setStart(int in_start) { start = in_start; }

void setEnd(int in_end) { end = in_end; }

void setStrand(char in_strand) { strand = in_strand; }

// --- Getters ---

```

```

string getID() const { return id; }

string getName() const { return name; }

string getChrom() const { return chrom; }

int getStart() const { return start; }

int getEnd() const { return end; }

char getStrand() const { return strand; }

string getOrganism() const { return organism; }

// --- Άλλες μέθοδοι ---

void describe() const {

    cout << "Gene " << name << " (" << id << ":" <<

        "Organism: " << organism

        << ", Chromosome: " << chrom

        << ", Start: " << start

        << ", End: " << end

        << ", Strand: " << strand << endl;

}

};

// -----
// Συνάρτηση που επιστρέφει αντικείμενο Protein
// -----

Protein makeProtein() {

    Protein temp("P999", "TEMP_PROT", "MSSSSSSSS");

    cout << "Η συνάρτηση makeProtein() τελειώνει.\n";

    return temp;

}

```

```

// -----
// main
// -----
int main() {

    cout << "==== Δημιουργία αυτόματων αντικειμένων ===" << endl;

    Protein p1("P001", "PTEN",
    "MTAIKEIVSRNKRRYQEDGFDLDLTYIYPNIIAMGFPA");

    Protein p2("P002", "BRCA1_Protein", "MSSSQDNRNLQPQKAK");

    cout << "\n==== Πληροφορίες Πρωτεϊνών ===" << endl;
    p1.describe();
    cout << "Length: " << p1.length() << endl;
    p2.describe();
    cout << "Length: " << p2.length() << endl;

    cout << "\n==== Δημιουργία Γονιδίων ===" << endl;
    Gene g1("G001", "BRCA1", "chr17", 43044295, 43170245, '+');
    Gene g2("G002", "TP53", "chr17", 7668402, 7687550, '-');

    g1.describe();
    g2.describe();

    cout << "\n==== Δυναμική δημιουργία ===" << endl;
    Protein* p3 = new Protein("P003", "MYC", "MTENEVHVEEQK");
    p3->describe();
    delete p3; // Εδώ καλείται ο destructor
}

```

```

cout << "\n==== Δοκιμή προσωρινού αντικειμένου από συνάρτηση ===" <<
endl;

Protein temp = makeProtein(); // εδώ θα δεις constructors/destructors

temp.describe();

cout << "\n==== Τέλος main() ===" << endl;

return 0;
}

```

---

### Επεξήγηση – Τι ζητούσε και πώς το υλοποιήσαμε:

Θέμα	Περιγραφή
<b>Χρήση string</b>	Όλα τα char* αντικαταστάθηκαν από std::string, χωρίς ανάγκη για new, delete, strcpy.
<b>Constructors</b>	Δημιουργήθηκαν default και parametric constructors που εκτυπώνουν μήνυμα όταν καλούνται.
<b>Destructors</b>	Εκτυπώνουν μήνυμα καταστροφής ώστε να φανεί πότε τα αντικείμενα βγαίνουν από το scope ή διαγράφονται.
<b>Λίστες αρχικοποίησης</b>	Χρησιμοποιήθηκαν : id(...), name(...), ... για σωστή αρχικοποίηση, ειδικά για το const string organism.
<b>Μετρητές αντικειμένων</b>	Καθολικές μεταβλητές protein_count, gene_count αυξάνονται/μειώνονται στους constructors/destructors.
<b>Δυναμική δημιουργία</b>	Χρήση new και delete ώστε να φανεί πότε καταστρέφονται τα αντικείμενα του heap.
<b>Συνάρτηση αντικείμενο επιστροφής</b>	με Η makeProtein() δείχνει πώς δημιουργούνται/καταστρέφονται προσωρινά αντικείμενα.

---

Αν θέλεις, μπορώ να σου φτιάξω και **εκδοχή με std::vector**, ώστε να αποθηκεύεις δυναμικά λίστες πρωτεϊνών/γονιδίων (π.χ. `vector<Protein> proteins;`  
`proteins.push_back(p1);`).  
Θες να σου την προσθέσω;

Πηγές

**Είπες:**

ναι και αυτό και ύστερα δώσε μου όλες τις περιπτώσεις χρήσεις του std::vector

**To ChatGPT είπε:**

Τέλεια!



Ας επεκτείνουμε το πρόγραμμα ώστε να αξιοποιεί **std::vector**, όπως ζητάει και το εργαστήριο 2 (οδηγίες από το PDF). Στη συνέχεια θα σου δώσω **αναλυτικά όλες τις βασικές περιπτώσεις χρήσης του std::vector**, με σχόλια και παραδείγματα.

---

### ◆ Επεκταμένο πρόγραμμα με χρήση std::vector

Αυτός είναι ο **βελτιωμένος κώδικας**, όπου πλέον δημιουργούμε **δυναμικές λίστες πρωτεϊνών και γονιδίων** μέσω `vector`, και εκτελούμε πράξεις όπως προσθήκη, αφαίρεση, εμφάνιση όλων των στοιχείων κ.λπ.

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

// -----
// Καθολικοί μετρητές ενεργών αντικειμένων
// -----
int protein_count = 0;
int gene_count = 0;

// -----
```

```
// Κλάση Protein

// -----
class Protein {

private:
    string id;
    string name;
    string sequence;

public:
    // --- Constructors ---

    Protein()
        : id("N/A"), name("Unnamed"), sequence("") {
            protein_count++;
            cout << "[Protein] Δημιουργήθηκε αντικείμενο (default). Σύνολο: "
                << protein_count << endl;
    }

    Protein(const string& in_id, const string& in_name, const string& in_seq)
        : id(in_id), name(in_name), sequence(in_seq) {
            protein_count++;
            cout << "[Protein] Δημιουργήθηκε αντικείμενο με δεδομένα: "
                << name << " (" << id << "). Σύνολο: " << protein_count << endl;
    }

    // --- Destructor ---
```

```
~Protein() {

    protein_count--;

    cout << "[Protein] Καταστράφηκε αντικείμενο: "
        << name << " (" << id << "). Εναπομέίναντα: "
        << protein_count << endl;

}

// --- Setters / Getters ---

void setID(const string& in_id) { id = in_id; }

void setName(const string& in_name) { name = in_name; }

void setSequence(const string& in_seq) { sequence = in_seq; }

string getID() const { return id; }

string getName() const { return name; }

string getSequence() const { return sequence; }

// --- Άλλες μέθοδοι ---

void describe() const {

    cout << "Protein " << name << " (" << id << "): " << sequence << endl;

}

int length() const {

    return sequence.length();

}

};

// -----
```

```
// Κλάση Gene
// -----
class Gene {
private:
    const string organism;
    string id;
    string name;
    string chrom;
    int start;
    int end;
    char strand;

public:
    // --- Constructors ---
    Gene()
        : organism("Homo sapiens"), id("N/A"), name("Unnamed"),
          chrom("chr?"), start(0), end(0), strand('+')
    {
        gene_count++;
        cout << "[Gene] Δημιουργήθηκε αντικείμενο (default). Σύνολο: "
            << gene_count << endl;
    }

    Gene(const string& in_id, const string& in_name, const string& in_chrom,
         int in_start, int in_end, char in_strand,
         const string& in_organism = "Homo sapiens")
        : organism(in_organism), id(in_id), name(in_name),
```

```

chrom(in_chrom), start(in_start), end(in_end), strand(in_strand)

{

    gene_count++;

    cout << "[Gene] Δημιουργήθηκε αντικείμενο: " << name
        << " (" << id << "). Σύνολο: " << gene_count << endl;

}

// --- Destructor ---

~Gene() {

    gene_count--;

    cout << "[Gene] Καταστράφηκε αντικείμενο: " << name
        << " (" << id << "). Εναπομείναντα: " << gene_count << endl;

}

// --- Setters / Getters ---

void setID(const string& in_id) { id = in_id; }

void setName(const string& in_name) { name = in_name; }

void setChrom(const string& in_chrom) { chrom = in_chrom; }

void setStart(int in_start) { start = in_start; }

void setEnd(int in_end) { end = in_end; }

void setStrand(char in_strand) { strand = in_strand; }

string getID() const { return id; }

string getName() const { return name; }

string getChrom() const { return chrom; }

int getStart() const { return start; }

int getEnd() const { return end; }

```

```
char getStrand() const { return strand; }

string getOrganism() const { return organism; }

// --- Άλλες μέθοδοι ---

void describe() const {

    cout << "Gene " << name << " (" << id << "): "
        << "Organism: " << organism
        << ", Chromosome: " << chrom
        << ", Start: " << start
        << ", End: " << end
        << ", Strand: " << strand << endl;
}

};

// -----
// Συνάρτηση για δημιουργία προσωρινού Protein
// -----

Protein makeProtein() {

    Protein temp("P999", "TEMP_PROT", "MSSSSSSSS");
    cout << "Η συνάρτηση makeProtein() τελειώνει.\n";
    return temp;
}

// -----
// main
// -----

int main() {
```

```

cout << "==== Δημιουργία λιστών με std::vector ===" << endl;

// --- Δημιουργία vector πρωτεΐνων ---

vector<Protein> proteins;

proteins.push_back(Protein("P001", "PTEN",
"MTAIKEIVSRNKRRYQEDGFDLDLTYIYPNIIAMGFPA"));

proteins.push_back(Protein("P002", "BRCA1_Protein",
"MSSSQDNRNLPQKAK"));

proteins.emplace_back("P003", "MYC", "MTENEVHVEEQK"); // emplace_back: κατασκευή επί τόπου

cout << "\n--- Λίστα πρωτεΐνων ---" << endl;

for (const auto& p : proteins) {

    p.describe();

    cout << "Length: " << p.length() << endl;
}

// --- Διαγραφή τελευταίας πρωτεΐνης ---

cout << "\nΔιαγραφή τελευταίας πρωτεΐνης..." << endl;

proteins.pop_back();

// --- Δημιουργία vector γονιδίων ---

vector<Gene> genes;

genes.emplace_back("G001", "BRCA1", "chr17", 43044295, 43170245, '+');

genes.emplace_back("G002", "TP53", "chr17", 7668402, 7687550, '-');

cout << "\n--- Λίστα γονιδίων ---" << endl;

for (const Gene& g : genes) {

```

```
g.describe();  
}  
  
cout << "\nΠλήθος ενεργών πρωτεϊνών: " << protein_count << endl;  
cout << "Πλήθος ενεργών γονιδίων: " << gene_count << endl;  
  
cout << "\n==== Δυναμικό αντικείμενο ===" << endl;  
Protein* dynamicProt = new Protein("P010", "Dynamic_PROT",  
"MVVVVLLLQQ");  
dynamicProt->describe();  
delete dynamicProt; // destructor καλείται τώρα  
  
cout << "\n==== Συνάρτηση makeProtein() ===" << endl;  
Protein temp = makeProtein();  
temp.describe();  
  
cout << "\n==== Τέλος main() ===" << endl;  
return 0;  
}
```

---

## ◆ Περιπτώσεις χρήσης του std::vector

Το std::vector είναι **δυναμικός πίνακας**: μπορεί να μεγαλώνει ή να μικραίνει σε μέγεθος κατά την εκτέλεση. Παρακάτω είναι όλες οι βασικές λειτουργίες — με παραδείγματα και σχόλια:

Εντολή	Περιγραφή / Παράδειγμα
Δήλωση	vector<int> v; — δημιουργεί έναν κενό vector ακεραίων.
Αρχικοποίηση με τιμές	vector<string> names = {"Alice", "Bob", "Carol"};
Προσθήκη στοιχείου	v.push_back(42); — προσθέτει στο τέλος.
Κατασκευή επί τόπου	v.emplace_back(1, 2); — αποδοτικότερο για αντικείμενα (δημιουργεί απευθείας στο vector).
Αφαίρεση τελευταίου στοιχείου	v.pop_back();
Πρόσβαση με δείκτη θέσης	cout << v[0]; ή v.at(1); (το .at() ελέγχει όρια).
Πρώτο και τελευταίο στοιχείο	v.front() και v.back()
Μέγεθος	v.size() — πλήθος στοιχείων.
Έλεγχος αν είναι άδειος	v.empty() — true/false.
Εκκαθάριση όλων των στοιχείων	v.clear();
Επανάληψη (loop)	for (auto &x : v) cout << x;
Διαγραφή συγκεκριμένου στοιχείου	v.erase(v.begin() + 2); — αφαιρεί το 3ο στοιχείο.
Εισαγωγή συγκεκριμένη θέση σε	v.insert(v.begin() + 1, 99);
Αντιγραφή	vector<int> copy = v;
Ανταλλαγή	v.swap(v2);
Προκαθορισμένο	vector<double> data(5, 0.0); — 5 στοιχεία με τιμή

<b>Εντολή</b>	<b>Περιγραφή / Παράδειγμα</b>
<b>μέγεθος</b>	0.0
<b>Δυνατότητα ταξινόμησης</b>	sort(v.begin(), v.end()); (απαιτεί <algorithm>)
<b>Εύρεση στοιχείου</b>	auto it = find(v.begin(), v.end(), 5);
<b>Αποδέσμευση μνήμης</b>	Η μνήμη απελευθερώνεται αυτόματα όταν ο vector βγει εκτός scope.
<b>Διπλάσιο (capacity)</b>	<b>μέγεθος</b> Ο vector αυξάνει εσωτερικά το capacity αυτόματα. Μπορείς να το ελέγξεις με v.capacity().

---

### ◆ Επιπλέον πρακτικά παραδείγματα

#### Παράδειγμα 1 – Λίστα ονομάτων πρωτεΐνων:

```
vector<string> names;
names.push_back("PTEN");
names.push_back("BRCA1");
names.push_back("TP53");
```

```
for (const auto& n : names)
    cout << n << endl;
```

#### Παράδειγμα 2 – Λίστα αντικειμένων Protein:

```
vector<Protein> proteins;
proteins.emplace_back("P001", "PTEN", "MTAIKEIV...");
proteins.emplace_back("P002", "BRCA1", "MSSSQD...");
```

```
for (const Protein& p : proteins)
    p.describe();
```

### **Παράδειγμα 3 – Διαγραφή στοιχείου:**

```
proteins.erase(proteins.begin() + 1); // διαγράφει τη 2η πρωτεΐνη
```

### **Παράδειγμα 4 – Αναζήτηση με std::find\_if:**

```
#include <algorithm>

auto it = find_if(proteins.begin(), proteins.end(),
    [] (const Protein& p){ return p.getName() == "PTEN"; });

if (it != proteins.end()) it->describe();
```

---

### **Συνοψίζοντας**

Με το std::vector:

- **έχεις δυναμική διαχείριση μνήμης χωρίς new/delete,**
- **μπορείς να αποθηκεύεις αντικείμενα οποιασδήποτε κλάσης,**
- **παρέχει ασφάλεια και ευκολία με πολλές βιοηθητικές μεθόδους,**
- **η μνήμη αποδεσμεύεται αυτόματα** όταν ο vector τελειώσει.