

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 << "). Εναπομείναντα: "
        << 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 << "). Εναπομείναντα: " << 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",
"MTAIKEIVSRNKRRYQEDGFDLDTYIYPNIIAMGFPA");

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

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

Το 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 << "). Εναπομείναντα: "
        << 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 πρωτεϊνών ---
vector<Protein> proteins;

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

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

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

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

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

◆ Επιπλέον πρακτικά παραδείγματα

Παράδειγμα 1 – Λίστα ονομάτων πρωτεϊνών:

```
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 βοηθητικές μεθόδους,
- η μνήμη **αποδεσμεύεται αυτόματα** όταν ο `vector` τελειώσει.