ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ

https://helios.ntua.gr/course/view.php?id=869

Διδάσκοντες: Γιώργος Γκούμας
Άρης Παγουρτζής
Γιώργος Στάμου
Βασίλης Βεσκούκης
Θάνος Βουλόδημος
Γιώργος Αλεξανδρίδης
Γιώργος Σιόλας
Παρασκευή Τζούβελη

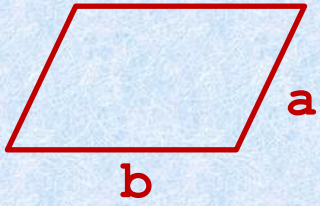
progtech@cslab.ece.ntua.gr

Διαφάνειες παρουσιάσεων

- Η γλώσσα προγραμματισμού C++
- Αρχές αντικειμενοστρεφούς προγραμματισμού
- Δομές δεδομένων

Πολλαπλή κληρονομικότητα

- Η κλάση Rectangle

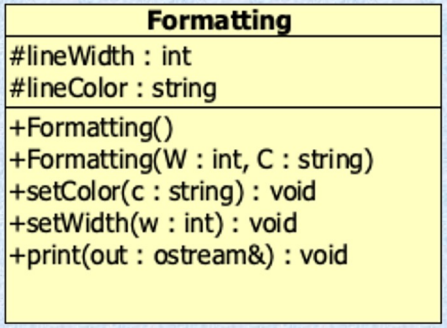


Rectangle
#a : double #b : double
+Rectangle() +Rectangle(A : double, B : double) +A() : double +B() : double +setA(A : double) : void +setB(B : double) : void +perimeter() : double +area() : double +print(out : ostream&) : void

```
class Rectangle {
protected:
    double a, b;
public:
    Rectangle;
    Rectangle(double A, double B);

    double A();
    double B();
    void setA(double A);
    void setB(double B);

    double perimeter();
    double area();
    void print(ostream &out);

    friend ostream& operator<<(ostream &out, Rectangle r
};
```

Πολλαπλή κληρονομικότητα

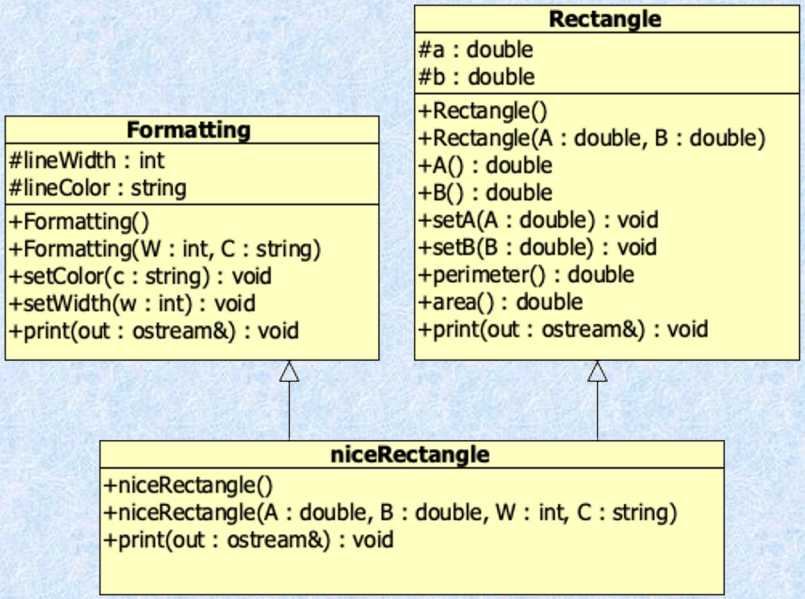
- Η κλάση Formatting




```
class Formatting {
protected:
  int lineWidth;
  string lineColor;
public:
  Formatting();
  Formatting(int W, string C);

  void setColor(string c);
  void setWidth(int w);
  friend ostream& operator<<(ostream &out, Formatting d);
};
```

Πολλαπλή κληρονομικότητα

- Μια κλάση κληρονομεί από περισσότερες



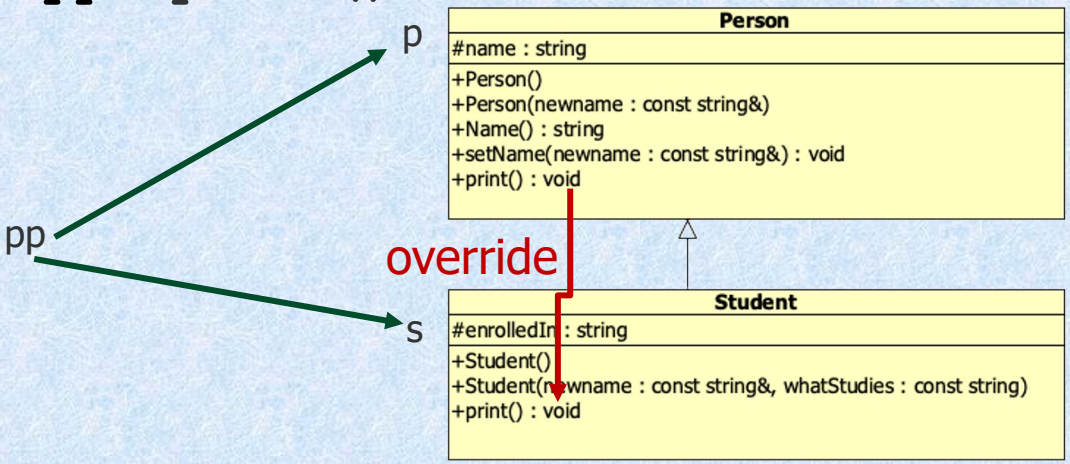

```
class niceRectangle : public Formatting, public Rectangle {
public:
  niceRectangle();
  niceRectangle(double, double, int, string);
  void print(ostream &out );
  friend ostream& operator <<(ostream&, niceRectangle); };
```

Εικονικές μέθοδοι

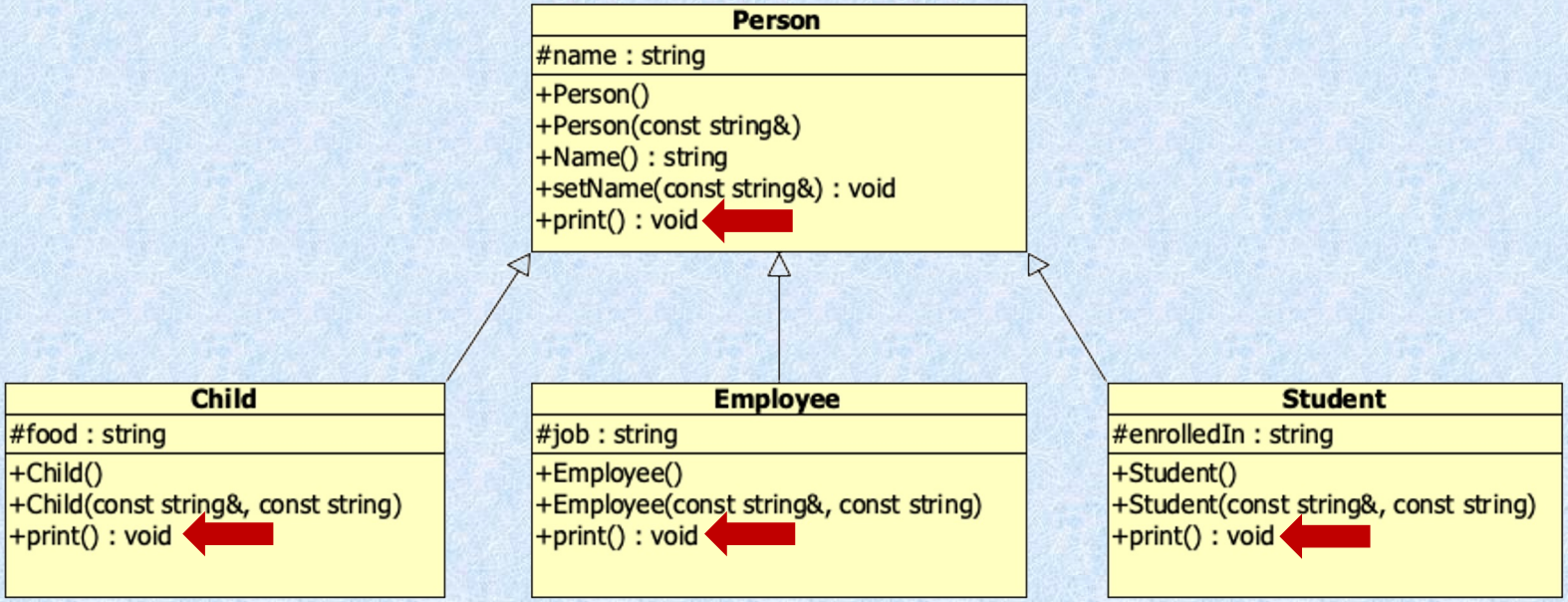
```
class Person {
protected:
  string name;
public:
  ...
  virtual void print() const {
     cout << "person " << name << " sleeps\n";}
};

class Student : public Person {
private:
  string enrolledIn;
public:
  void print() const override{ cout << "student "
    << name << " reads about " << enrolledIn << "\n"; }
};
```

Εικονικές μέθοδοι

```
Person p, *pp;
Student s;
p.print();  // person John Doe sleeps
s.print();  // student John Doe reads about maths

pp = &p;
pp->print(); // person John Doe sleeps

pp = &s;
pp->print(); // student John Doe reads about maths
```



Πολυμορφισμός



Πολυμορφισμός

```
class Person {
  ...
  virtual void print() const { cout <<"person "<<name
                             <<" sleeps\n";} };

class Student : public Person {
  ...
  void print() const override { cout<<"student "<< name
                        <<" studies "<<enrolledIn<<"\n";} };

class Child : public Person {
  ...
  void print() const override { cout <<"child "<<name
                             <<" eats "<<food<<"\n"; } };

class Employee : public Person {
  ...
  void print() const override { cout<< "employee "<<name
                             <<" works as a "<<job<<"\n"; } };
```

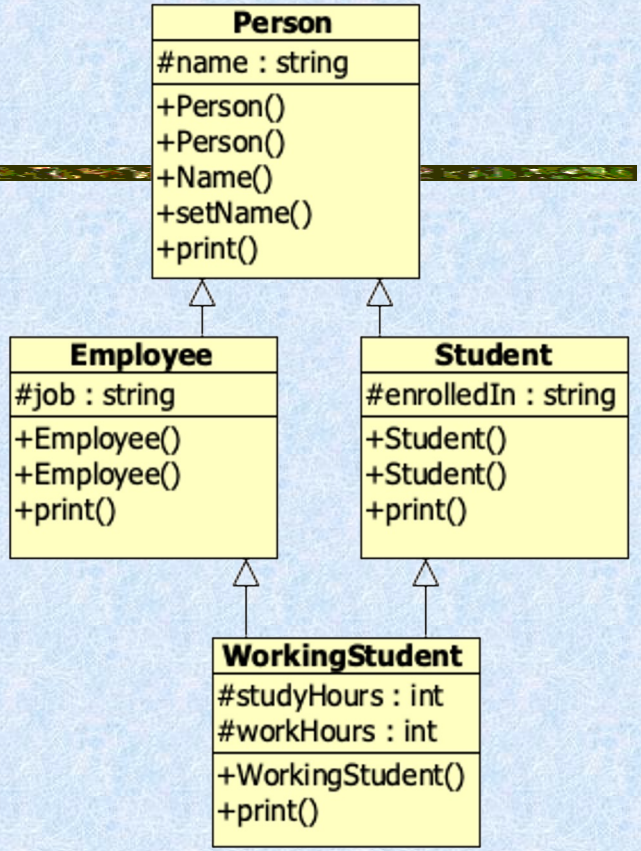
Πολυμορφισμός

```
void allInfo(int n, Person *p[]) {
    for (int i = 0; i < n; i++)
        p[i] -> print();
}

int main() {
  Person *p[] = { new Person, new Student,
                  new Child,  new Employee };
  allInfo(4, p);
}
```

```
person John Doe sleeps
student John Doe studies maths
child John Doe eats Milk
employee John Doe works as a c++ programmer
```

Εικονικές κλάσεις

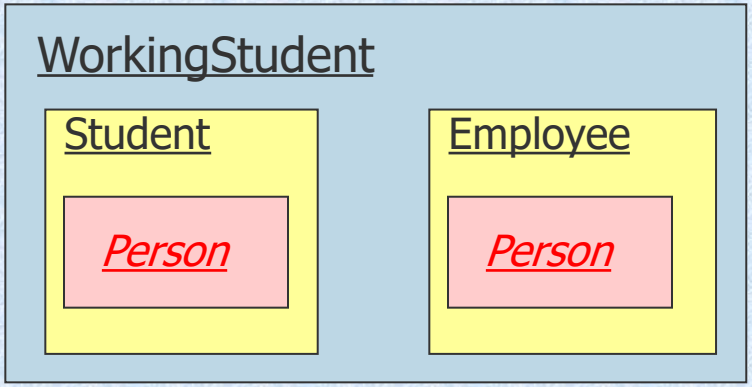
```
class Person { … }

class Student :
  public Person { … }

class Employee :
  public Person { … }

class WorkingStudent :
  public Student,
  public Employee { … }
```



Non-static member 'name' found in multiple base-class subobjects of type 'Person':
class WorkingStudent -> class Student -> class Person
class WorkingStudent -> class Employee -> class Person
member found by ambiguous name lookup

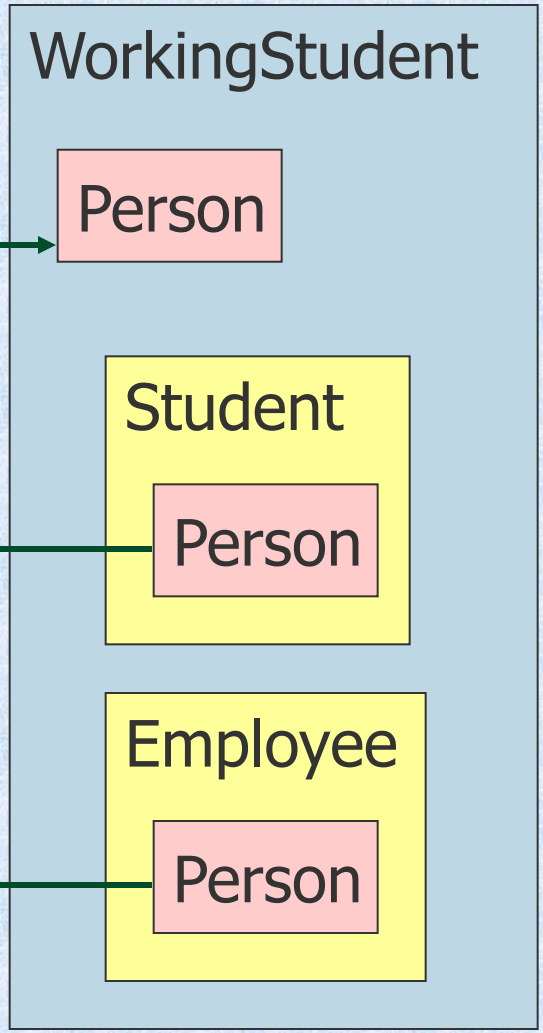
Declared in: main.cpp

```
protected:
basic_string<char, char_traits<char>, allocator<char>> Person::name
```



Εικονικές κλάσεις

```
class Person { … }

class Student :
  virtual public Person { … }

class Employee :
  virtual public Person { … }

class WorkingStudent :
  public Student,
  public Employee { … }
```



virtual: δεν θα δημιουργηθεί στιγμιότυπο μελών δηλωμένων ως virtual

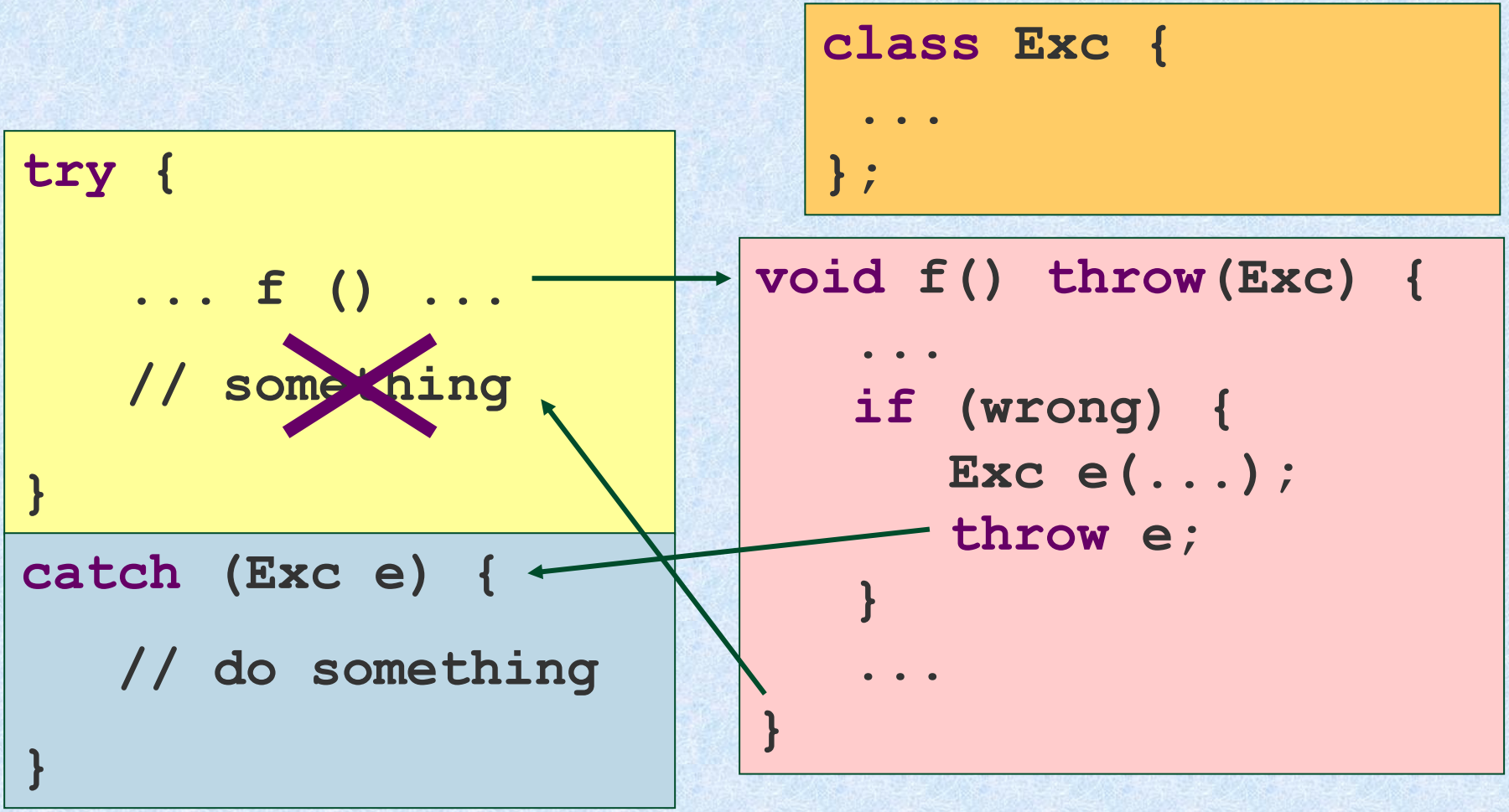
Χειρισμός εξαιρέσεων

- Τι είναι εξαιρέσεις (exceptions)
  - Κάθε είδους ανωμαλίες ή σφάλματα που προκύπτουν κατά την εκτέλεση του προγράμματος και πρέπει να ξεπερασθούν με ειδικό τρόπο
- Εξαιρέσεις προκαλούνται
  - Από το σύστημα, π.χ. εξάντληση μνήμης
  - Οπως αποφασίζουμε εμείς, σύμφωνα με τη λογική του προγράμματός μας

Χειρισμός εξαιρέσεων

- Χειρισμός εξαιρέσεων στη C++
  - Τι θέλουμε να γίνει όταν προκύψει μια εξαίρεση:
    - τερματισμός του προγράμματος;
    - μεταβολή που αναιρεί την εξαίρεση;
    - άλλο;
- Μηχανισμός: **throw / try - catch**
  - η εξαίρεση προκαλείται με **throw** είτε αυτόματα, είτε ρητά στο πρόγραμμα
  - ανιχνεύεται μέσα σε block εντολών **try**
  - αντιμετωπίζεται μέσα σε block εντολών **catch**

Χειρισμός εξαιρέσεων



Χειρισμός εξαιρέσεων - παράδειγμα

1. Γνωστή εξαίρεση (ορισμένη στη γλώσσα ή από τον προγραμματιστή)

2. Πρόκληση εξαίρεσης (από τον προγραμματιστή, ενίοτε από τη γλώσσα)

```
class runtime_error: public exception
double division(double num, double den) {
    if (den) return num/den;
    throw runtime_error("Division by zero!\n");
}
int main() {
    double a = 1, b = 0, c;
    try {
        c = division(a, b);
        cout << a << "/" << b << "= " << c << endl; }
    catch(runtime_error &e) {
        cout << "exception: " << e.what();
    }
}
```

3. Χειρισμός εξαίρεσης (από τον προγραμματιστή)

`what()`:
Ορίζεται στην `runtime_error`

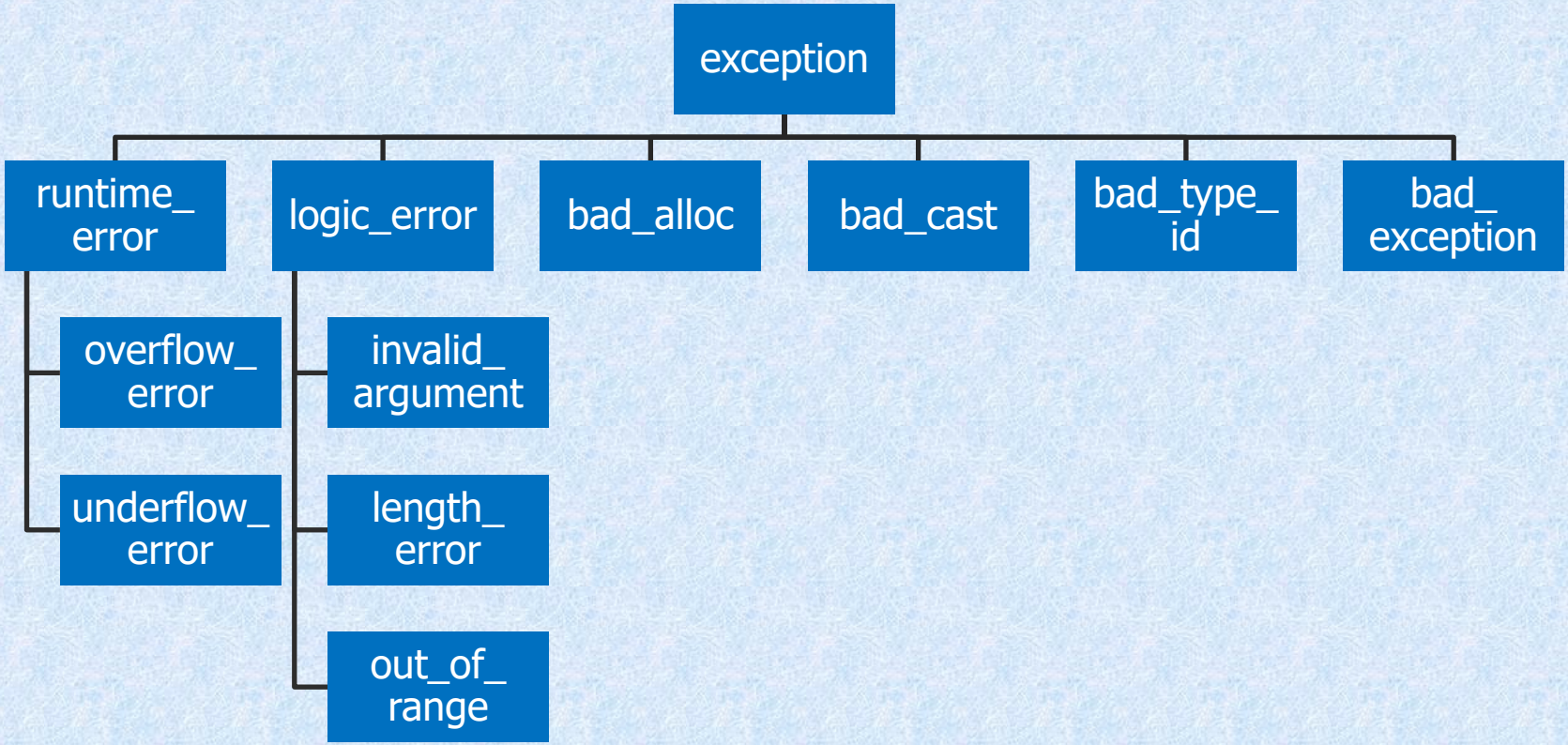
Χειρισμός εξαιρέσεων - παραδείγματα

```
class unreal_rectangle : public exception {};

class Rectangle {
...
public:
  ...
  Rectangle(double A, double B) {
    try {
      a = A; b = B;
      if ((a<0) || (b<0)) throw unreal_rectangle();
    }
    catch (unreal_rectangle &e) {
      cout << "illegal value of a or b\n";
      a = a < 0 ? 0 : a;
      b = b < 0 ? 0 : b;
    }
```

Πρόκληση εξαίρεσης

Χειρισμός εξαίρεσης

Εξαιρέσεις στη C++



Χειρισμός εξαιρέσεων

```
#include <exception>
using namespace std;
class full_stack : public exception {};
class stack {
   ...
stack(int NN) {
   top=0; mysize=NN;
   try {
     data = new int[mysize]; }
   catch(bad_alloc) {
     mysize = 5; data = new int[mysize];
     cerr<<"not enough memory, size set to 5";
   }
};
```

το new κάνει throw

Δεν είναι υποχρεωτικό να διακόπτουμε το πρόγραμμα...

ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ

https://helios.ntua.gr/course/view.php?id=869

Διδάσκοντες:

Γιώργος Γκούμας
Άρης Παγουρτζής
Γιώργος Στάμου
Βασίλης Βεσκούκης
Θάνος Βουλόδημος
Γιώργος Αλεξανδρίδης
Γιώργος Σιόλας
Παρασκευή Τζούβελη

progtech@cslab.ece.ntua.gr

8/4/22

Διαφάνειες παρουσιάσεων

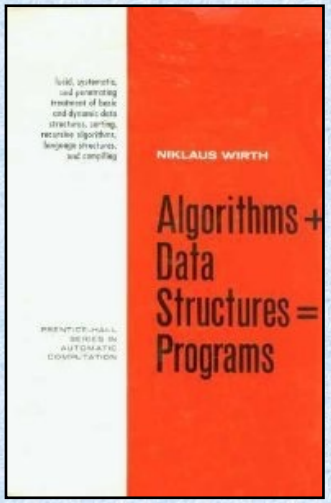
- Η γλώσσα προγραμματισμού C++
- Αρχές αντικειμενοστρεφούς προγραμματισμού
- Δομές δεδομένων

Εισαγωγή στις δομές δεδομένων

- Αλγόριθμος

  Πεπερασμένο σύνολο εντολών που,
  όταν εκτελεστούν,
  επιτυγχάνουν κάποιο επιθυμητό αποτέλεσμα

  - Δεδομένα εισόδου και εξόδου
  - Εκτέλεση σειριακή, υπό συνθήκη, επαναλαμβανόμενη
  - Κάθε εντολή πρέπει να είναι:
    - καλά ορισμένη
    - απλή
  - Η εκτέλεση πρέπει να σταματά



Εισαγωγή στις δομές δεδομένων

- Πολυπλοκότητα

  Μέτρο εκτίμησης της απόδοσης αλγορίθμων ως συνάρτηση του μεγέθους του προβλήματος που επιλύουν

  - Χρόνος εκτέλεσης
  - Χωρητικότητα μνήμης που απαιτείται
- Ακριβής μέτρηση πολυπλοκότητας

$$t = f(n)$$

- Τάξη μεγέθους, συμβολισμοί O, Ω, Θ

$$t = O(f(n))$$

Εισαγωγή στις δομές δεδομένων

- Ορισμός των κλάσεων O, Ω, Θ
  - Άνω όριο

    $g = O(f) \Leftrightarrow \exists c.\ \exists n_0.\ \forall n > n_0.\ g(n) < c\ f(n)$
  - Κάτω όριο

    $g = \Omega(f) \Leftrightarrow \exists c.\ \exists n_0.\ \forall n > n_0.\ g(n) > c\ f(n)$
  - Τάξη μεγέθους

    $g = \Theta(f) \Leftrightarrow \exists c_1, c_2.\ \exists n_0.\ \forall n > n_0.$
    $c_1\ f(n) < g(n) < c_2\ f(n)$
- Διάταξη μερικών κλάσεων πολυπλοκότητας

$$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$$

Αφαίρεση (abstraction)

- Νοητική διαδικασία αντίληψης των θεμελιωδών ιδιοτήτων μιας οντότητας, πέρα από τα φυσικά χαρακτηριστικά ή/και λεπτομέρειες που ξεχωρίζουν τα συγκεκριμένα αντικείμενα του φυσικού κόσμου
  - π.χ. η *έννοια* του μεγέθους στη φύση
  - π.χ. ένας χάρτης
- Χρησιμοποιείται:
  - για να συγκαλύψει άσχετες λεπτομέρειες
  - για να δώσει έμφαση σε σχετικές ιδιότητες με το πρόβλημα και την επίλυσή του.

Μια (γνωστή!) μορφή αφαίρεσης

- Ξεχωριστή μεταγλώττιση – Αρχείο επικεφαλίδας (header file)

```
int gcd(int x, int y);
```
gcd.hpp

Αφαιρετική αναφορά στην ύπαρξη του συμβόλου gcd(int, int)

- Αρχείο υλοποίησης

gcd.cpp

```
#include "gcd.hpp"

int gcd(int x, int y) {
  while (x > 0 && y > 0)
    if (x > y) x %= y; else y %= x;
  return x + y;
}
```

Υλοποίηση πηγαίου κώδικα του συμβόλου gcd(int, int)

Μια (γνωστή!) μορφή αφαίρεσης

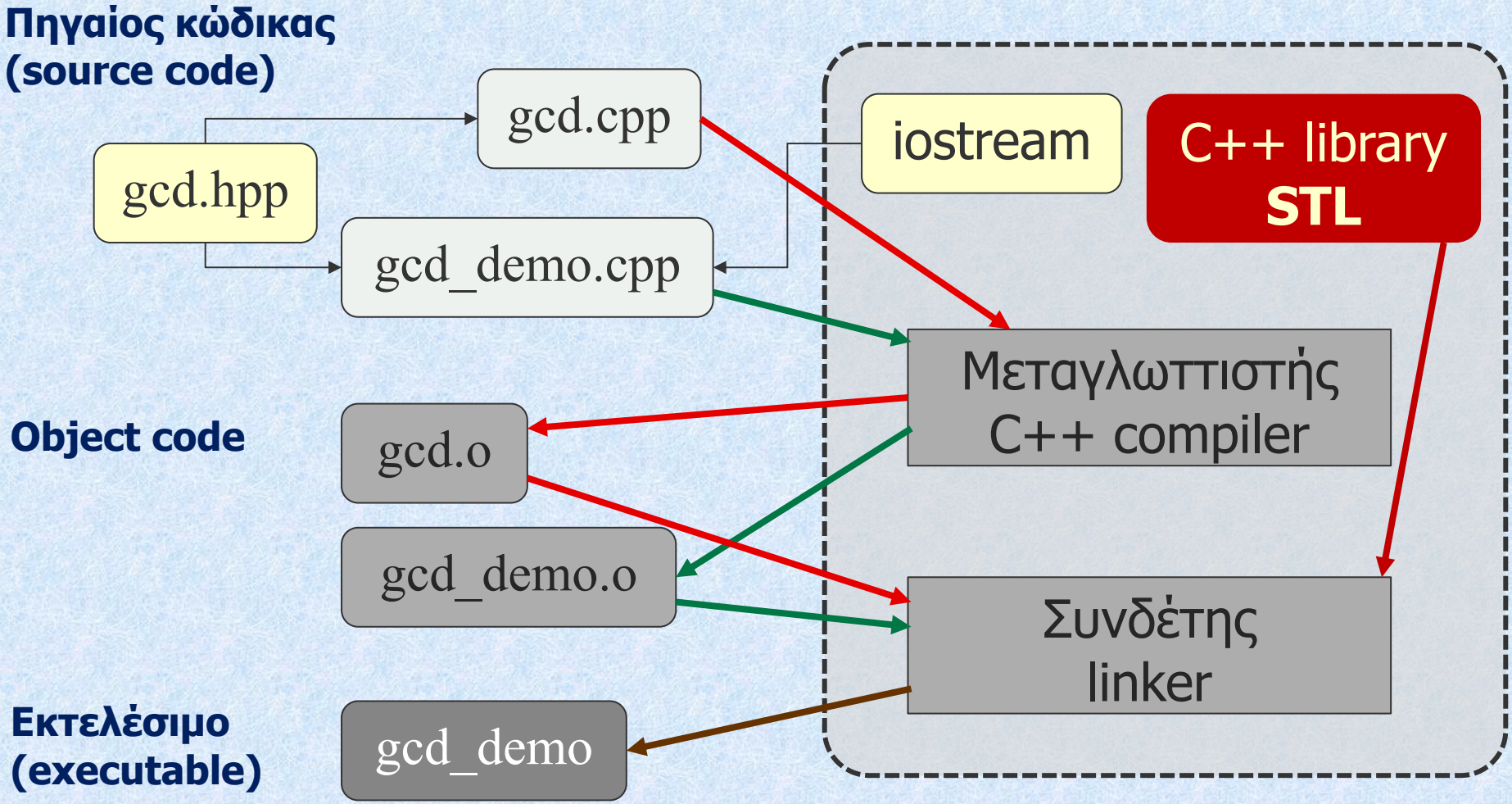
```
int gcd(int x, int y);
```
gcd.hpp

gcd_demo.cpp

```
#include <iostream>
#include "gcd.hpp"

using namespace std;

int main() {
  cout << "Give me two numbers: ";
  int a, b;
  cin >> a >> b;
  cout << "GCD(" << a << ", " << b
       << ") = " << gcd(a, b) << endl;
}
```

Χρήση του συμβόλου gcd(int, int) χωρίς να μας ενδιαφέρει η υλοποίησή του

Μια (γνωστή!) μορφή αφαίρεσης



Αφαίρεση και δομές δεδομένων (i)

- Αφαίρεση δεδομένων (data abstraction)
  - Διαχωρισμός ιδιοτήτων και υλοποίησης
  - Ιδιότητες ενός τύπου δεδομένων:
    ο τρόπος με τον οποίο δημιουργεί κανείς και χρησιμοποιεί δεδομένα αυτού του τύπου
  - Υλοποίηση ενός τύπου δεδομένων:
    ο τρόπος με τον οποίο αναπαρίστανται τα δεδομένα στη μνήμη του υπολογιστή και προγραμματίζονται οι διαθέσιμες πράξεις

Αφαίρεση και δομές δεδομένων (ii)

- Οι λεπτομέρειες υλοποίησης επηρεάζουν την πολυπλοκότητα των αλγορίθμων
- Είναι συχνή επομένως η αλλαγή τρόπου υλοποίησης κάποιου τύπου δεδομένων
- Η αφαίρεση δεδομένων ελαχιστοποιεί τις αλλαγές που απαιτούνται στο πρόγραμμα που χρησιμοποιεί έναν τύπο δεδομένων, όταν αλλάξει ο τρόπος υλοποίησης

Αφηρημένοι (αφαιρετικοί;) τύποι δεδομένων

- Αφηρημένος τύπος δεδομένων – ΑΤΔ (abstract data type)
  - καθορίζει τις ιδιότητες του τύπου δεδομένων
  - δεν καθορίζει την υλοποίησή του
  - ⇒ το public μέρος μιας κλάσης στη C++
- Αλγεβρικός ορισμός ΑΤΔ
  - σύνταξη: όνομα του τύπου και επικεφαλίδες των πράξεων
  - σημασιολογία: κανόνες που περιγράφουν τη λειτουργία των πράξεων

Παράδειγμα: stack

- stack<T>
  - Λειτουργίες: **`push(), pop()`**
- Χρήση
  - **`stack<int> s;`**
  - **`s.push(5);`**
  - **`int x=s.pop();`**

Παράδειγμα: stack

```
class someStack {
public:
    someStack() ;
    someStack(int NN) ;
    void push(int a);
    int pop();
private:
    int p, N;
    int data[1000];
};
```

```
someStack() {
    p=0; N=1000;
}
someStack(int NN) {
    p=0; N=1000;
}
void push(int a) {
    if (p<N) data[p++] = a ;
}
int pop() {
    if (p>0) return data[--p];
    return data[0];
}
```

Παράδειγμα: stack

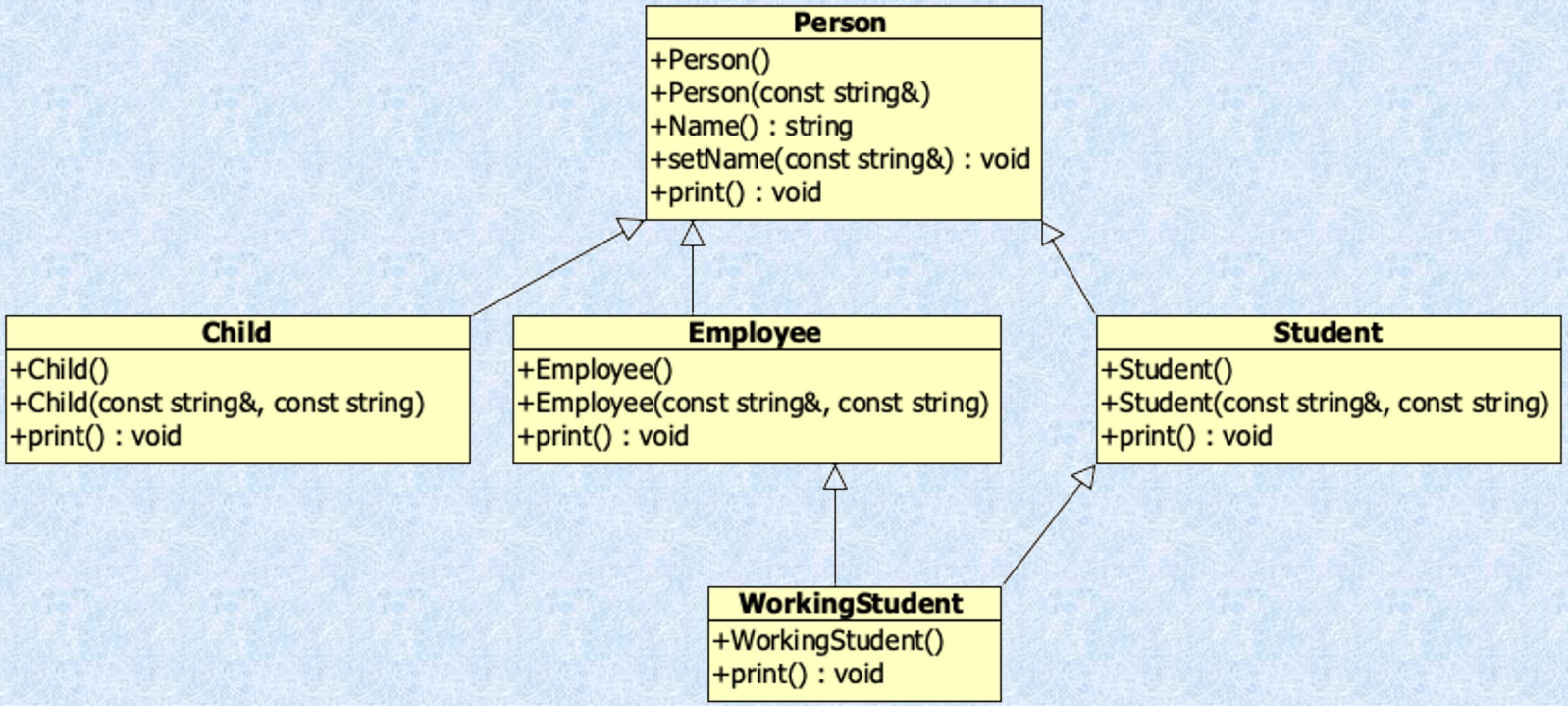
```
class someStack {
public:
    someStack() ;
    someStack(int NN) ;
    void push(int a);
    int pop();
private:
    int p, N;
    int *data;
};
```

Εναλλακτική υλοποίηση

```
someStack() {
    p=0; N=5;
    data = new int[N];
}
someStack(int NN) {
    p=0; N=NN;
    try { data = new int[N]; }
    catch(bad_alloc) { ... }
}
void push(int a) {
    if (p<N) data[p++] = a ;
    else throw full_stack();
}
int pop() {
    if (p>0) return data[--p];
    else throw empty_stack();
}
```

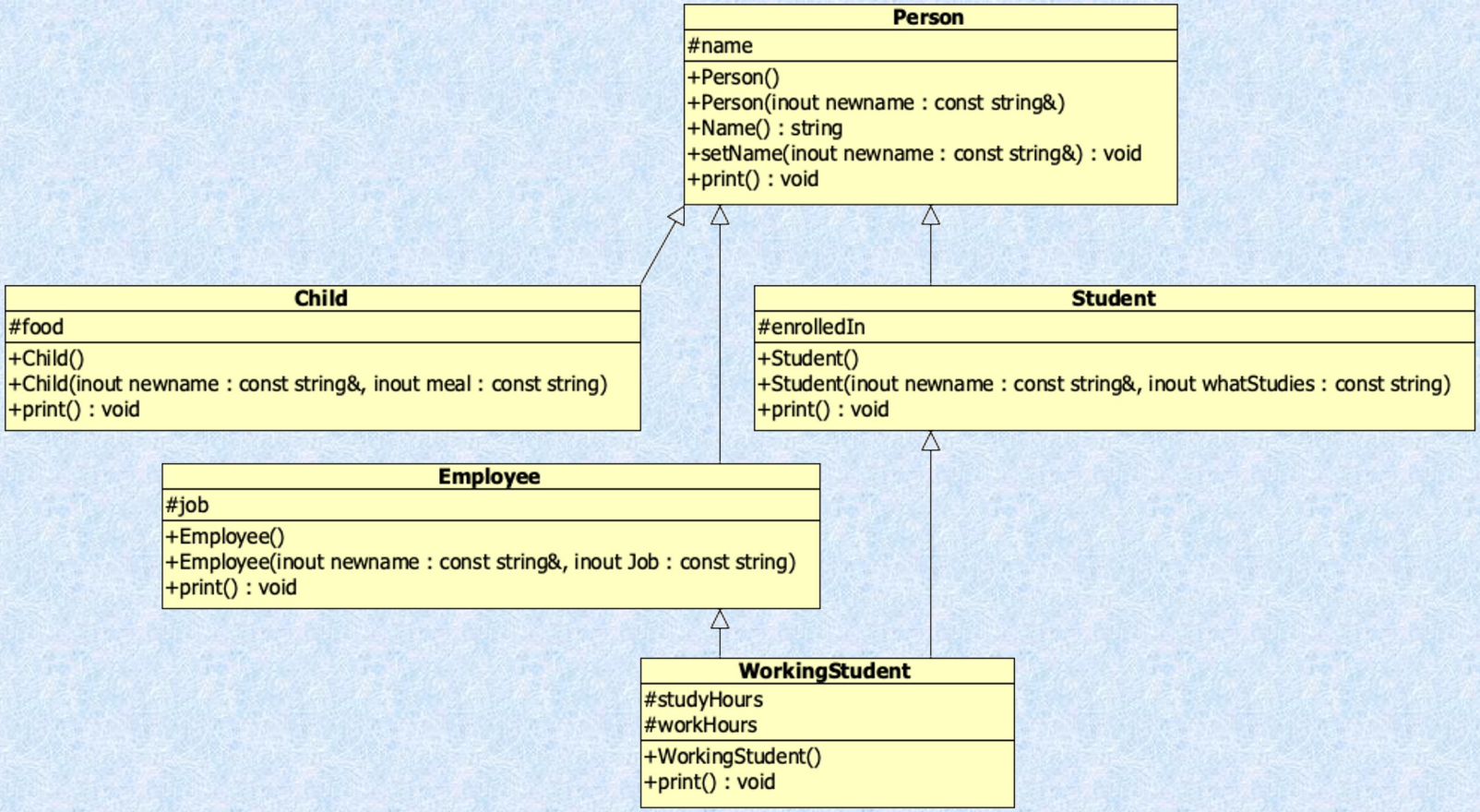
Αφηρημένοι (αφαιρετικοί;) τύποι

- Η ορατότητα προς τα έξω
- Ενδιαφέρει τους χρήστες της κλάσης



Σχεδίαση με εργαλεία

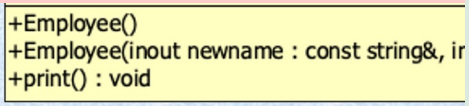
- Πλήρης περιγραφή των μελών κάθε κλάσης

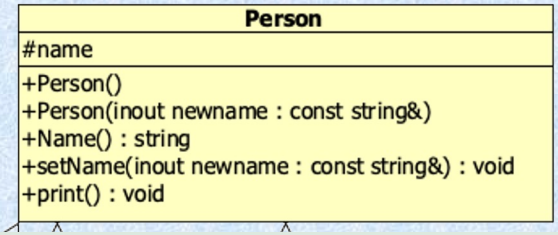


Σχεδίαση με εργαλεία

- Αυτόματη παραγωγή κώδικα (όχι της λύσης ☹)

```
//person.h
class Person {
protected:
    string name;
public:
    Person();
    Person(const string& newname);
    string Name();
    void setName(const string&
newname);
    void print();
};
```






```
//person.cpp
#include "Person.h"

Person::Person() {
    // TODO - implement Person::Person
    throw "Not yet implemented";
}

Person::Person(const string& newname) {
    // TODO - implement Person::Person
    throw "Not yet implemented";
}

string Person::Name() {
    // TODO - implement Person::Name
    throw "Not yet implemented";
}
```

Συγκεκριμένοι τύποι δεδομένων

- Συγκεκριμένος τύπος δεδομένων – ΣΤΔ (concrete data type)
  - καθορίζει τις ιδιότητες του τύπου δεδομένων
  - καθορίζει επακριβώς την υλοποίησή του
- Κάθε γλώσσα προγραμματισμού υποστηρίζει ορισμένους ΣΤΔ, π.χ.
  - απλοί τύποι: ακέραιοι αριθμοί, πραγματικοί αριθμοί, χαρακτήρες, λογικές τιμές
  - σύνθετοι τύποι: πίνακες (arrays), εγγραφές (records), δείκτες (pointers), σύνολα (sets)

Σχεδιαστικό μοτίβο (design pattern)

- Επαναχρησιμοποιήσιμη μορφή λύσης σε κάποιο συχνά εμφανιζόμενο σχεδιαστικό πρόβλημα
- Τα σχεδιαστικά μοτίβα είναι γενικές, καταγεγραμμένες καλές πρακτικές που οι προγραμματιστές μπορούν να χρησιμοποιούν
- Συνήθως αφορούν γενικές μορφές τύπων δεδομένων και αλγορίθμων, αλλά συνηθέστερα τρόπους αρχιτεκτονικής δόμησης συστημάτων λογισμικού
- Στη C++, τα σχεδιαστικά μοτίβα σχεδιάζονται και παρουσιάζονται σαν μία ιεραρχία κλάσεων

Design patterns

Creational	Structural	Behavioral
Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight Observer State Strategy Visitor

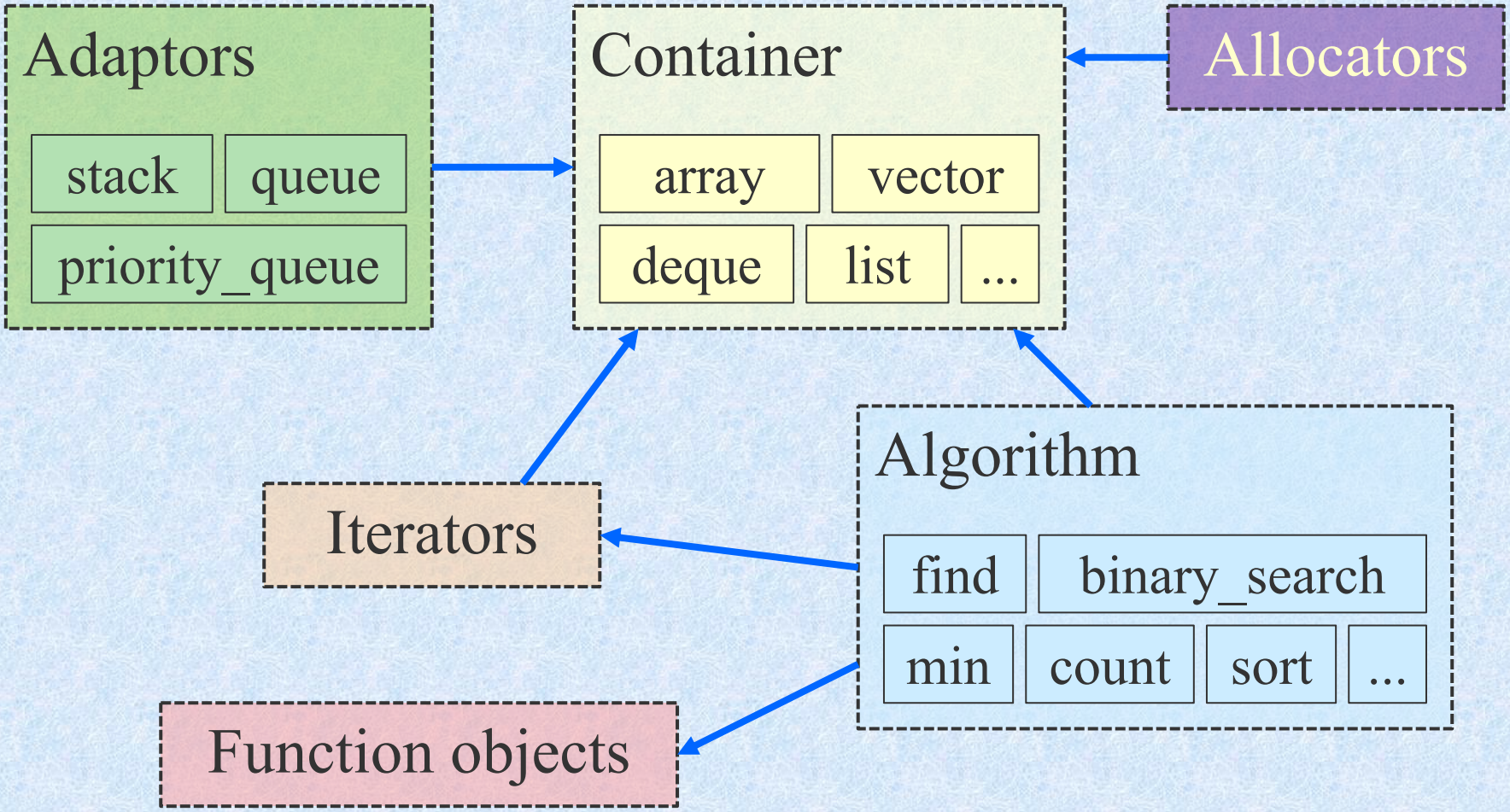
Standard Template Library

ΕΙΣΑΓΩΓΗ ΣΤΗΝ STL

Εισαγωγή στην STL

- Standard Template Library
- Μέρος της βιβλιοθήκης της C++
- Αρχική σχεδίαση: Alexander Stepanov
- Περιέχει χρήσιμες υλοποιήσεις:
  - δομών δεδομένων
  - αλγορίθμων
- Για να καταλάβουμε σε βάθος την STL:
  - templates
  - ιεραρχίες κλάσεων

Εισαγωγή στην STL



Εισαγωγή στην STL

www.cplusplus.com/reference/stl/

Container class templates

Sequence containers:

array C++11	Array class (class template)
vector	Vector (class template)
deque	Double ended queue (class template)
forward_list C++11	Forward list (class template)
list	List (class template)

Container adaptors:

stack	LIFO stack (class template)
queue	FIFO queue (class template)
priority_queue	Priority queue (class template)

Associative containers:

set	Set (class template)
multiset	Multiple-key set (class template)
map	Map (class template)
multimap	Multiple-key map (class template)

Unordered associative containers:

unordered_set C++11	Unordered Set (class template)
unordered_multiset C++11	Unordered Multiset (class template)
unordered_map C++11	Unordered Map (class template)
unordered_multimap C++11	Unordered Multimap (class template)

Συμβολοσειρά

- Τύπος **`string`**
  - περιτύλιγμα για array από **`char`**
- Χρήσιμες μέθοδοι
  - **`size`**, **`length`**
  - **`c_str`**
  - **`empty`**
  - **`operator[]`**, **`substr`**
  - **`append`**, **`operator+=`**
  - **`compare`**, **`operator==`**, **`operator<`**, κ.λπ.
  - **`insert`**, **`replace`**
  - **`find`**, **`rfind`**

Συμβολοσειρά

```
string s = "to be, or not to be? that is the question!";
string q = s;
cout << q << endl;
```

```
to be, or not to be? that is the question!
```

```
q += " _W.Shakespeare!";
cout << q << endl;
```

```
to be, or not to be? that is the question! _W.Shakespeare!
```

```
cout<<"s.size="<<s.size()<<" length="<<s.length()<<endl;
```

```
s.size=42 length=42
```

```
cout << s[1] << endl;
```

```
o
```

```
cout << s.substr(3, 2) << endl;
```

```
be
```

```
s.replace(21,4,"THAT");
cout << s << endl;
```

```
to be, or not to be? THAT is the question!
```

Συμβολοσειρά

```
s.insert(28, " indeed");
cout << s << endl;
```

```
to be, or not to be? THAT is indeed the question!
```

```
int w = s.find("indeed",0);
cout << w << endl;
```

```
29
```

```
w = s.find("really",0);
cout << w << endl;
```

```
-1
```

```
cout << s.data() << endl;
```

```
to be, or not to be? THAT is indeed the question!
```

```
w = s.find("o");
int x = s.rfind("o");
cout << w << " " << x << endl;
```

```
1 46
```

Πίνακας

- Τύπος **`array<T, n>`**
  - παρόμοια συμπεριφορά με τους απλούς πίνακες
  - σταθερού μεγέθους, γνωστού at compile time

```
#include <array>
...
array<int, 10> a;    // παρόμοιο με int
  a[10];
array<string, 20> s;
a[1] = 42;
s[2] = "Hello";
```

Πίνακας

```
array<int, 10> a10;
for(int i=0; i<a10.size(); i++) a10.at(i) = i * 10;
for(int i=0; i<a10.size(); i++) cout<<a10.at(i)<<" ";
```

```
0 10 20 30 40 50 60 70 80 90
```

```
for (int d:a10) cout << d << " ";
```

```
0 10 20 30 40 50 60 70 80 90
```

```
array<array<int, 10>, 10> a100;
for (int i=0; i<10; i++)
    for (int j=0; j<10; j++)
        a100[i][j] = i + j;

for (array<int,10> line:a100) {
    for (int d: line)
        cout << d << "\t";
    cout << endl;
}
```

```
0   1   2   3   4   5   6   7   8   9
1   2   3   4   5   6   7   8   9   10
2   3   4   5   6   7   8   9   10  11
3   4   5   6   7   8   9   10  11  12
4   5   6   7   8   9   10  11  12  13
5   6   7   8   9   10  11  12  13  14
6   7   8   9   10  11  12  13  14  15
7   8   9   10  11  12  13  14  15  16
8   9   10  11  12  13  14  15  16  17
9   10  11  12  13  14  15  16  17  18
```

Διάνυσμα

- Τύπος **`vector<T>`**
  - παρόμοια συμπεριφορά με array
  - μεταβλητού μεγέθους
    (εισαγωγή και αφαίρεση στο τέλος)
  - **`push_back`**, **`pop_back`**
  - ειδική αποδοτική υλοποίηση: **`vector<bool>`**

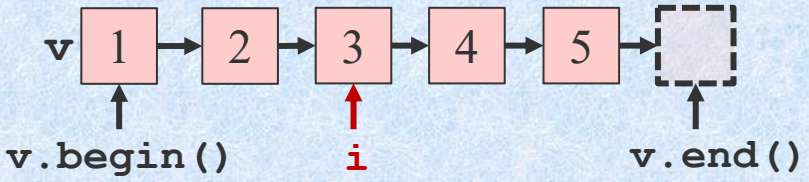
Διάνυσμα

```
#include <vector>
...
vector<int> x(100), y(200, 42);
cout << y[199] << endl;              // 42
cout << x.size() << endl;            // 100

x.push_back(17);
cout << x[100] << endl;              // 17
cout << x.size() << endl;            // 101

x.pop_back();
cout << x.size() << endl;            // 100
```

Iterators

- Iterators (πχ) **`vector<T>::iterator`**
  - επιτρέπουν τη διάσχιση ενός container
  - συμπεριφέρονται σαν να ήταν δείκτες

```
void showvector(vector<int> v) {
  cout<<"size= "<<v.size()<< "  contents= ";
  vector<int>::iterator it;
  for (it = v.begin(); it != v.end(); ++it)
      cout << *it << " ";
  cout << endl;
}
```



Iterators

- Εναλλακτική, απλούστερη διάσχιση (C++11)

```
list<int> l;

for (int x : l)
  cout << x << endl;

for (int &x : l)
  x = 42;

for (const int &x : l)
  cout << x << endl;


void showvector(vector<int> v) {
  cout <<"size= "<<v.size()<<"  contents= ";
  for (int d: v) cout << d << " ";
  cout << endl;
}
```

Διάνυσμα

```
vector<int> v;
for (int i=0; i<5; i++) v.push_back(i);
showvector(v);
```

```
size= 5  contents= 0 1 2 3 4
```

```
v.push_back(5); showvector(v);
```

```
size= 6  contents= 0 1 2 3 4 5
```

```
v.pop_back(); showvector(v);
```

```
size= 5  contents= 0 1 2 3 4
```

```
v.insert(v.begin()+2,40);
showvector(v);
```

```
size= 6  contents= 0 1 40 2 3 4
```

```
vector<int> r(5,10);
showvector(r);
```

```
size= 5  contents= 10 10 10 10 10
```

```
r.resize(8); r[7] = 20;
showvector(r);
```

```
size= 8  contents= 10 10 10 10 10 0 0 20
```

```
r.resize(3);
showvector(r);
```

```
size= 3  contents= 10 10 10
```

```
cout << r.max_size();
```

```
4611686018427387903
```

Λίστα

- Τύπος `list<T>`
  - διπλά συνδεδεμένη λίστα, γραμμική διάσχιση
  - εισαγωγή και αφαίρεση σε αρχή, τέλος
  - `front`, `back`, `empty`
  - `push_front`, `pop_front`
  - `push_back`, `pop_back`
  - εισαγωγή και διαγραφή ενδιάμεσα
  - `insert`, `erase`

Λίστα

```
void showlist(list<int> l) {
  list<int>::iterator it;
  cout << "size= " << l.size() << "  contents= ";
  for (it = l.begin(); it!= l.end(); ++it) cout << *it << " ";
  cout << endl; }

list<int> l;
for (int i=1; i<5; i++) l.push_back(i);
for (int i=0; i<5; i++) l.push_front(-i);
showlist(l);
```

```
size= 9  contents= -4 -3 -2 -1 0 1 2 3 4
```

```
for (int i=1; i<5; i++) {
  cout << l.front() << " " << l.back() << " ";
  l.pop_front(); l.pop_back();
}
showlist(l);
```

```
-4 4 -3 3 -2 2 -1 1
```

```
size= 1  contents= 0
```

Λίστα

```
size= 11  contents= 0 1 2 3 4 5 9 8 7 6 5
```

```
l.reverse();
showlist(l);
```

```
size= 11  contents= 5 6 7 8 9 5 4 3 2 1 0
```

```
l.sort();
showlist(l);
```

```
size= 11  contents= 0 1 2 3 4 5 5 6 7 8 9
```

```
l.push_back(5);
showlist(l);
```

```
size= 12  contents= 0 1 2 3 4 5 5 6 7 8 9 5
```

```
l.unique();
showlist(l);
```

```
size= 12  contents= 0 1 2 3 4 5 6 7 8 9 5
```

Στοίβα και ουρά

- Τύπος **`stack<T>`**
  - container adapter, όχι container
  - εισαγωγή και αφαίρεση στην κορυφή
  - push, pop, top, empty

```
stack<int> s;
for (int i=0; i<5; i++) s.push(i*10);
for (int i=0; i<10; i++) if (!s.empty()) {
    cout << s.top() << " ";
    s.pop();
}
```

Στοίβα και ουρά

- Τύπος **`queue<T>`**
  - container adapter, όχι container
  - εισαγωγή στο τέλος, αφαίρεση από την αρχή
  - **`push, pop, front, empty`**

- Στοίβες και ουρές μπορούν να υλοποιηθούν με χρήση διαφορετικών containers, πχ:

```
stack<int, vector<int>> s;
```

Τύπος που περιέχει το stack

Container που υλοποιεί το stack

Στοίβα και ουρά

```
void showq(queue<int> Q) {
    queue<int> sq = Q;
    while (!sq.empty()) {
        cout << sq.front() << " "; sq.pop();
    }
}

queue<int> q;
for (int i=1; i<=5; i++) q.push(i*10);
showq(q);
for (int i=1; i<=5; i++) {
    q.pop();
    showq(q);
}
```

```
10 20 30 40 50
20 30 40 50
30 40 50
40 50
50
```

Ζεύγη

- Τύπος `pair<T1, T2>`
  - `make_pair`, `first`, `second`

```
#include <utility>
double measure(const pair<double, double> d) {
  return sqrt(d.first*d.first + d.second*d.second);
}

int main() {
  pair<double, double> c1 (3, 4);
  pair<double, double> c2;
  c2 = make_pair(1, 1);
  cout << measure(c1) << "  " << measure(c2);
}
```

Πλειάδες (tuples) στη C++11

- Τύπος **tuple<T1, ... Tn>**
  - **make_tuple, get**

```
#include <tuple>

...

void f(const tuple<string, int, bool> &t) {
  if (get<2>(t)) cout << get<0>(t) << endl;
  else           cout << get<1>(t) << endl;
}

tuple<string, int, bool> t("Hi", 1, true);

f(t); f(make_tuple("No", 42, false));
```

Πλειάδες (tuples) στη C++11

```
typedef tuple<int, string, int> myDate;

void printdate(const myDate d, const string dateFormat) {
  if (dateFormat == "us")
    cout << get<1>(d) << " " << get<0>(d)
         <<", " << get<2>(d) << endl;
  else
    cout << get<0>(d) << " " << get<1>(d)
         <<" " << get<2>(d) << endl;
}

...
myDate easter22, easter23(16, "April", 2023);
easter22 = make_tuple(22,"April", 2022);
printdate(easter22, "us");
printdate(easter23, "eu");
```

```
April 22, 2022
16 April 2023
```

Σύνολο

- Τύπος **`set<T>`**
  - μοναδικά στοιχεία (αλλιώς **`multiset<T>`**)
  - υλοποίηση με ισοζυγισμένο δένδρο δυαδικής αναζήτησης
  - εισαγωγή, εύρεση και αφαίρεση σε O(log*n*)
  - **`insert`**, **`erase`**, **`find`**, **`size`**
  - πράξεις συνόλων (υλοποιούνται και για άλλους containers, λιγότερο αποδοτικά)
  - **`includes`**, **`set_union`**, **`set_intersection`**, **`set_difference`**

Σύνολο

```
#include <set>
...
set<int> s;
while (true) {
  int x;
  cin >> x;
  if (s.find(x) == s.end())
    s.insert(x);
  else
    cout << "You lose, I've seen" << x
         << "before!" << endl;
}
```

Σύνολο

```
set<int> someInts= {10, 15, 3, 19, 2, 67, 69, 68};
for (int i:someInts) cout << i << " "; cout << endl;
```

```
2 3 10 15 19 67 68 69
```

```
someInts.insert(11);
for (int i:someInts) cout << i << " "; cout << endl;
```

```
2 3 10 11 15 19 67 68 69
```

```
set<int> moreInts= {70, 8, 81, 5, 10, 69, 67, 31, 3};
set<int> unionDemo;
set_union(someInts.begin(), someInts.end(),
          moreInts.begin(), moreInts.end(),
              inserter(unionDemo, unionDemo.begin()));

for (int i:unionDemo) cout << i << " "; cout << endl;
```

```
2 3 5 8 10 11 15 19 31 67 68 69 70 81
```

Σύνολο

```
set<int> someInts= {10, 15, 3, 19, 2, 67, 69, 11, 68};

set<int> moreInts= {70, 8, 81, 5, 10, 69, 67, 31, 3};


set<int> intersectionDemo;
set_intersection(someInts.begin(), someInts.end(),
                 moreInts.begin(), moreInts.end(),
                 inserter(intersectionDemo,
                          intersectionDemo.begin()));

for (int i:intersectionDemo) cout << i << " "; cout << endl;
```

```
3 10 67 69
```