TEMPLATES, STL
ΠΡΟΓΡΑΜΜΑΤΑ ΜΕ ΠΟΛΛΑ ΑΡΧΕΙΑ

ΑΝΑΚΕΦΑΛΑΙΩΣΗ

Η κλάση Array

```
class Array
{
private:
   Element **A;
   int size;
public:
   Array(int s);
   Element *& operator [](int);
   void Print();
   void Sort();
};
```

Abstract Class Element

```
class Element
{
public:
  virtual bool operator < (Element *other) = 0;
  virtual void PrintElement() = 0;
};
```

```
Array::Array(int s)
{
  size = s;
  A = new Element*[size];
}

Element *& Array::operator [] (int i)
{
  return A[i];
}

void Array::Print()
{
  for (int i = 0; i < 10; i ++){
    A[i]->Print();
  }
}

void Array::Sort()
{
  for (int i = 0; i < 10; i ++){
    for (int j = i+1; j < 10; j ++){
      if (*A[i] < A[j]){
        Element *temp = A[i];
        A[i] = A[j];
        A[j] = temp;
      }
    }
  }
}
```

Η κατασκευή αυτή είναι απαραίτητη γιατί χρειαζόμαστε ένα αντικείμενο Element (το *A[i]) για να χρησιμοποιήσουμε τον τελεστή < πάνω σε ένα δείκτη σε αντικείμενο Element (το A[j])

Εναλλακτικά:
A[i]->operator<(A[j])

Θα θέλαμε:
***A[i] < *A[j]**

```
Array::Array(int s)
{
  size = s;
  A = new Element*[size];
}

Element *& Array::operator [] (int i)
{
  return A[i];
}

void Array::Print()
{
  for (int i = 0; i < 10; i ++){
    A[i]->Print();
  }
}

void Array::Sort()
{
  for (int i = 0; i < 10; i ++){
    for (int j = i+1; j < 10; j ++){
      if (*A[i] < *A[j]){
        Element *temp = A[i];
        A[i] = A[j];
        A[j] = temp;
      }
    }
  }
}
```

Λύση: Χρήση αναφορών

Υλοποίηση της Array με αναφορές

```
class Array
{
private:
   Element **A;
   int size;
public:
   Array(int s);
   Element *& operator [](int);
   void Print();
   void Sort();
};
```

Η κλάση Array παραμένει ως έχει.

Abstract Class Element

```
class Element
{
public:
  virtual bool operator < (Element &other) = 0;
  virtual void Print() = 0;
};
```

IntElement

```
class IntElement: public Element
{
private:
   int val;
public:
   IntElement(int);
   int GetVal();
   bool operator < (Element &);
   void Print();
};
```

```
IntElement::IntElement(int i)
{
  val = i;
}

int IntElement::GetVal()
{
  return val;
}


bool IntElement::operator <(Element &cOther)
{
  IntElement &other = dynamic_cast<IntElement &>(cOther);
  if (val < other.GetVal()){
    return true;
  }
  return false;
}

void IntElement::Print()
{
  cout << val << endl;
}
```

Το dynamic casting δουλεύει και στην περίπτωση που έχουμε αναφορά αντί για δείκτη.

PointElement

```
class PointElement:public Element
{
private:
  point val;
public:
  PointElement(int,int);
  point GetVal();
  bool operator < (Element &);
  void Print();
};
```

```
PointElement::PointElement(int x,int y)
{
  val.x = x;
  val.y = y;
}

point PointElement::GetVal()
{
  return val;
}


bool PointElement::operator < (Element &cOther)
{
  PointElement &other = dynamic_cast<PointElement &>(cOther);
  if (val.x < other.GetVal().x){
    return true;
  }else if (val.x == other.GetVal().x){
    if (val.y < other.GetVal().y ){
      return true;
    }else {
      return false;
    }
  }else {
    return false;
  }
}

void PointElement::Print()
{
  cout << val.x << " " << val.y << endl;
}
```

```
Array::Array(int s)
{
  size = s;
  A = new Element*[size];
}

Element *& Array::operator [] (int i)
{
  return A[i];
}

void Array::Print()
{
  for (int i = 0; i < 10; i ++){
    A[i]->Print();
  }
}

void Array::Sort()
{
  for (int i = 0; i < 10; i ++){
    for (int j = i+1; j < 10; j ++){
      if (*A[i] < *A[j]){
        Element *temp = A[i];
        A[i] = A[j];
        A[j] = temp;
      }
    }
  }
}
```

Η Array είναι πλέον όπως τη θέλουμε

Dynamic Casting

```
IntElement &other = dynamic_cast<IntElement &>(cOther);
```

Μετατρέπει μια αναφορά τύπου **Element** (base class) σε αναφορά τύπου **IntElement** (derived class).

Η μετατροπή από βασική κλάση σε παραγόμενη επιτρέπεται μόνο αν η βασική κλάση είναι πολυμορφική

Τι γίνεται όμως αν η αναφορά **cOther** είναι αναφορά σε **PointElement** αντί για **IntElement**?

Run-time error: segmentation fault

Η εντολή dynamic_cast μας επιτρέπει να ελέγξουμε αν η μετατροπή έγινε σωστά

```
IntElement &other = dynamic_cast<IntElement &>(cOther);
if (other == 0){ cout << "unsuccessful type conversion" << endl;}
```

Συντακτικό

```
template<class DATA_TYPE>
class SomeClass<DATA_TYPE>
{
private:
        ...
        DATA_TYPE someData;
        ...
public:
        ...
        DATA_TYPE SomeMethod(DATA_TYPE *);
        ...
};

template<class DATA_TYPE>
DATA_TYPE SomeClass<DATA_TYPE>::SomeMethod(DATA_TYPE *x)
{
        ...
        if (someData.methodX() == x.methodX())
        ...
}
```

Απλά ένα όνομα που μετά θα αντικατασταθεί από το όνομα του τύπου δεδομένων

Ο τύπος δεδομένων που θα αντικαταστήσει το **DATA_TYPE** θα πρέπει να υλοποιεί την **methodX**

Class Tempate Array

```
template <class DATA_TYPE>
class Array
{
private:
   DATA_TYPE **A;
   int size;
public:
   Array(int s);
   DATA_TYPE *& operator [](int);
   void Sort();
   void Print();
};
```

```
template <class DATA_TYPE>
Array<DATA_TYPE>::Array(int s)
{
  size = s;
  A = new DATA_TYPE*[size];
}

template <class DATA_TYPE>
DATA_TYPE *& Array<DATA_TYPE>::operator [] (int i)
{
  return A[i];
}

template <class DATA_TYPE>
void Array<DATA_TYPE>::Sort()
{
  for (int i = 0; i < 10; i ++){
    for (int j = i+1; j < 10; j ++){
      if (*A[i] < *A[j]){
       DATA_TYPE *temp = A[i];
       A[i] = A[j];
       A[j] = temp;
      }
    }
  }
}
```

```
template <class DATA_TYPE>
void Array<DATA_TYPE>::Print()
{
  for (int i = 0; i < 10; i ++){
    A[i]->Print();
  }
}
```

Χρήση στη main()

```
int main()
{
  srand(time(NULL));
  Array<IntElement> iA(10);
  Array<PointElement> pA(10);

  for (int i = 0; i < 10; i ++){
    iA[i] = new IntElement(rand()%10);
    pA[i] = new PointElement(rand()%10, rand()%10);
  }

  iA.Print();
  pA.Print();

  iA.Sort();
  pA.Sort();

  iA.Print();
  pA.Print();
}
```

Χρήση στη main()

```
struct point
{
  int x;
  int y;
};

int main()
{
  srand(time(NULL));
  Array<point> pA(10);
  Array<int> iA(10);

  for (int i = 0; i < 10; i ++){
    iA[i] = new int(rand()%10);
    pA[i] = new point; pA[i]->x = rand()%10; pA[i]->y = rand()%10;
  }

  iA.Sort(); //OK! Int has < operator
  pA.Sort(); // COMPILE ERROR: point does not have < operator
  iA.Print(); // COMPILE ERROR: int does not have Print method
  pA.Print(); // COMPILE ERROR: point does not have Print method
}
```

Το Compiler Error εμφανίζεται μόνο όταν χρησιμοποιούμε την μέθοδο που έχει πρόβλημα

STANDARD TEMPLATE LIBRARY (STL)

STL

- Υπάρχουν τριών ειδών οντότητες που μας ενδιαφέρουν
  - Αποδέκτες (Containers)
    - Οι διαφορετικοί τρόποι για να αποθηκεύουμε τα δεδομένα μας.
  - Επαναλήπτες (Iterators)
    - Μας παρέχουν δείκτες στα στοιχεία του container και μας επιτρέπουν να διατρέχουμε τα στοιχεία του αποδέκτη.
  - Αλγόριθμοι (Algorithms)
    - Υλοποιημένοι αλγόριθμοι που μας δίνουν βασικές λειτουργίες πάνω στους αποδέκτες

STL Sequential Containers

Container	Χαρακτηριστικά	+/-
Πίνακας C++	Σταθερό μέγεθος	+ Γρήγορη τυχαία πρόσβαση. - Αργή προσθήκη σε ενδιάμεση θέση - Σταθερό μέγεθος
vector	Δυναμικός πίνακας	+ Γρήγορη τυχαία πρόσβαση. - Αργή προσθήκη σε ενδιάμεση θέση - Προσθήκη μόνο στο τέλος
list	Διπλά συνδεδεμένη λίστα	+ Γρήγορη προσθήκη σε ενδιάμεση θέση - Αργή τυχαία πρόσβαση.
deque	Δυναμικός πίνακας με πρόσβαση και από τις δύο μεριές	+ Γρήγορη τυχαία πρόσβαση. + Προσθήκη σε αρχή και τέλος - Αργή προσθήκη σε ενδιάμεση θέση

vector

Μέθοδος
`int size()`
`void push_back(const TYPE &)`
`void pop_back()`
`TYPE & back()`
`TYPE & operator [](int)`
`bool empty()`
`iterator insert(iterator, const TYPE &)`
`iterator erase(iterator)`

list

Μέθοδος
`int size()`
`void push_back(const TYPE &)`
`void pop_back()`
`TYPE & back()`
`bool empty()`
`iterator insert(iterator, const TYPE &)`
`iterator erase(iterator)`
`void push_front(const TYPE &)`
`void pop_front()`
`TYPE & front()`

Επίσης, μεθόδους: reverse, merge, unique

deque

- Έχει τις μεθόδους και των δύο παραπάνω.
- Και προσθήκη στην αρχή και το τέλος, και τυχαία πρόσβαση.

iterators

```
#include <iostream>
#include <list>

using namespace std;

int main(){
  list<int> L;
  int x;

  do{
    cin >> x;
    L.push_back(x);
  }while (x != -1);

  cout << "list elements: ";
  for (list<int>::iterator iter = L.begin(); iter != L.end(); iter ++){
    *iter += 2;
    cout << *iter << " ";
  }
  cout << endl;
}
```

list<int>::iterator iter: Δήλωση του iterator

L.begin(): Συνάρτηση που επιστρέφει iterator στην αρχή του container

L.end(): Συνάρτηση που επιστρέφει iterator στο τέλος του container

iter++: Κάνει τον iterator να δείχνει στο επόμενο στοιχείο της λίστας

***iter:** Το περιεχόμενο της θέσης στην οποία δείχνει ο iterator, μπορεί να χρησιμοποιηθεί και για να πάρει και για να δώσει τιμή.

iterators

```
#include <iostream>
#include <list>

using namespace std;

int main(){
  list<int> L;
  int x;

  do{
    cin >> x;
    L.push_back(x);
  }while (x != -1);

  cout << "list elements in reverse: ";
  list<int>::reverse_iterator riter;
  for (riter = L.rbegin(); riter != L.rend(); iter ++){
    cout << *iter << " ";
  }
  cout << endl;
}
```

Εκτύπωση των στοιχείων σε αντίστροφη σειρά χρησιμοποιώντας έναν **reverse_iterator**

iterators

```
#include <iostream>
#include <list>

using namespace std;

int main(){
  list<int> L;
  int x;

  do{
    cin >> x;
    L.push_back(x);
  }while (x != -1);

  list<int>::iterator  iter;
  for (iter = L.begin(); iter != L.end(); iter ++){
    if (*iter == 8){ break;}
  }
  if (iter != L.end()) { // υπάρχει στοιχείο με τιμή 8
    L.erase(iter);
  }
}
```

Βρες και σβήσε το πρώτο στοιχείο της λίστας με την τιμή 8, εφόσον υπάρχει.

Associative Containers

Container	Χαρακτηριστικά
set	Αποθηκεύει μόνο αντικείμενα-κλειδιά Δεν επιτρέπει πολλαπλές τιμές
map	Κρατάει ζεύγη κλειδιών και τιμών (key-value pairs). Συσχετίζει αντικείμενα-κλειδιά με αντικείμενα-τιμές. Το κάθε κλειδί μπορεί να συσχετίζεται με μόνο μία τιμή
multiset	Επιτρέπει πολλαπλές εμφανίσεις ενός κλειδιού
multimap	Επιτρέπει πολλαπλές τιμές για ένα κλειδί.

Παραδειγμα set

```
#include <iostream>
#include <set>

using namespace std;

int main(){
  set<string> S;

  string name;
  while (!cin.eof()){
    cin >> name;
    S.insert(name);
  }

  set<string>::iterator iter = S.find("bob");

  if (iter == S.end()){ cout << "bob is not in\n";}
  else{cout << "bob is in\n";}

  for (set<string>::iterator iter =  S.begin(); iter != S.end(); iter ++){
    cout << *iter << " ";
  }
  cout << endl;
}
```

Set με δικές μας κλάσεις

- Τι γίνεται αν θέλουμε να βάλουμε αντικείμενα από μία δική μας κλάση σε ένα Set?
  - Τι σημαίνει ότι το set περιέχει ήδη ένα αντικείμενο?

- Για να χρησιμοποιήσουμε το Set, η κλάση θα πρέπει να έχει ορισμένους τον τελεστή == και τον τελεστή < .

Παραδειγμα Set

```
class Person {
private:
  int id;
  string fname;
  string lname;
public:
  Person();
  Person(int);
  void SetDetails(int id, string fn, string ln);
  bool operator == (const Person &) const;
  bool operator < (const Person &) const;
  int getId() const;
  void PrintDetails() const;
};
```

```
Person::Person() {}

Person::Person(int i)
{ id = i; }

void Person::SetDetails(int i, string f, string l){
  id = i;  fname = f;  lname = l;
}

bool Person::operator == (const Person &p) const
{ return (id == p.getId()); }

bool Person::operator < (const Person &p) const
{ return (id < p.getId()); }

int Person::getId() const
{ return id; }

void Person::PrintDetails() const
{
  cout << "id: " << id
       <<" first name:" << fname << " last name:" << lname << endl;
}
```

```
int main(){
  set<Person> S;
  Person P[3];

  int id;
  string fname, lname;
  for(int i =0 ; i< 3; i ++){
    cin >>id >> fname >> lname;
    P[i].SetDetails(id, fname, lname);
    S.insert(P[i]);
  }

  cin >> id;
  Person sp(id);
  set<Person>::iterator iter = S.find(sp);
  if (iter == S.end()){
    cout << "id not found\n";
  }else{
    iter->PrintDetails();
  }
}
```

Η αναζήτηση γίνεται με ένα αντικείμενο της κλάσης

Παράδειγμα map

```
#include <iostream>
#include <map>
using namespace std;

int main(){
  map<int,Person*> M;
  int id;

  while (!cin.eof()){
    string fname,lname;
    cin >> id >> fname >> lname;
    Person *p =new Person; p->setDetails(id,fname,lname);
    M[id] = p;
  }

  cin >> id;
  map<int,Person*>::iterator iter = M.find(id);
  if (iter == M.end()){
    cout << "id is not in\n";
  }else{
    M[id]->PrintDetails();
  }
}
```

Το κλειδί στην περίπτωση αυτή είναι int, αλλά μπορεί να είναι οποιοσδήποτε τύπος δεδομένων που πληροί τις προϋποθέσεις για να είναι κλειδί σε Set.

Παράδειγμα map iterator

```
#include <iostream>
#include <map>
using namespace std;

int main(){
  map<int,Person*> M;
  int id;

  while (!cin.eof()){
    string fname,lname;
    cin >> id >> fname >> lname;
    Person *p =new Person; p->setDetails(id,fname,lname);
    M[id] = p;
  }

  map<int,Person*>::iterator iter = M.find(id);
  for (iter = M.begin(); iter != M.end(); i++){
    cout << (*iter).first << ":";
    (*iter).second->PrintPersonalDetails();
  }
}
```

Αλγόριθμοι

- Στην STL υπάρχουν ήδη υλοποιημένοι αλγόριθμοι που μπορούν να εφαρμοστούν σε containers.
- Ο κάθε αλγόριθμος χρειάζεται κάποιον τύπο iterator, το οποίο καθορίζει και το αν μπορούμε να τον εφαρμόσουμε σε ένα συγκεκριμένο τύπο container.

find

- Βρίσκει την πρώτη εμφάνιση μίας τιμής μέσα σε ένα container.
- Δουλεύει για όλους τους containers
- Συντακτικό:
  - `iterator find(start, finish, value)`
- Παραδειγμα:

```
include<algorithm>
vector<int> v
vector<int>::iterator i =
    find(v.begin(), v.end(), 100)
```

- Επιστρέφει iterator στην θέση του στοιχείου αν είναι μέσα στον container, ή `v.end()` αν δεν είναι.

sort

- Ταξινομεί τα στοιχειά του αποδέκτη.
- Χρειάζεται τυχαία πρόσβαση, και άρα δουλεύει μόνο για vectors και deque
- Συντακτικό:
  - `sort(start, finish)`
  - Ταξινομεί σε αύξουσα σειρά
- Παραδειγμα:

```
include <algorithm>
vector<int> v;
sort(v.begin(), v.end());
```

Αντικείμενα συναρτήσεων

- Τι γίνεται όμως αν θέλουμε να ταξινομήσουμε τα στοιχεία με κάποια άλλη σειρά?
- Τότε πρέπει να περάσουμε ως όρισμα στην sort ένα αντικείμενο-συνάρτησης.
  - Βασικά ένα αντικείμενο μιας template class στην οποία υπερφορτώνουμε τον τελεστή ().
- Η C++ έχει ήδη υλοποιήσει κάποια από αυτά τα αντικείμενα.

```
vector<int> v;
sort(v.begin(), v.end(), greater<int>());
```

```
class ComparePersons
{
public:
  bool operator ()(const Person &p1, const Person &p2) const
  {
    return p1 < p2;
  }
};


int main(){
  vector<Person> V;
  Person P[3];

  for(int i =0 ; i< 3; i ++){
    int id;
    string fname, lname;
    cin >>id >> fname >> lname;
    P[i].SetDetails(id, fname, lname);
    V.push_back(P[i]);
  }

  sort(V.begin(),V.end(),ComparePersons());
  sort(P,P+3,ComparePersons());

  for (int i =0; i < 3; i ++){
    P[i].PrintDetails();
  }
}
```

```
bool compare(const Person &p1, const Person &p2)
{
  return p1 < p2;
}



int main(){
  vector<Person> V;
  Person P[3];

  for(int i =0 ; i< 3; i ++){
    int id;
    string fname, lname;
    cin >>id >> fname >> lname;
    P[i].SetDetails(id, fname, lname);
    V.push_back(P[i]);
  }

  sort(V.begin(),V.end(),compare);
  sort(P,P+3,compare);

  for (int i =0; i < 3; i ++){
    P[i].PrintDetails();
  }
}
```

for_each

- Καλεί μια συνάρτηση, στην οποία περνάμε ως όρισμα, για κάθε στοιχείο του container.
- Μπορεί να χρησιμοποιηθεί για όλους τους containers.
- Συντακτικό:
  - **for_each(start, finish, function)**
  - Η συνάρτηση **function** παίρνει όρισμα τον τύπο δεδομένων που περιέχει ο container, και εφαρμόζεται σε όλα τα στοιχεια μεταξύ **start** και **finish**.
- Παράδειγμα:

```
void inc(int *i){ cout << ++(*i) << endl;}
vector<int *> v;
for_each(v.begin(), v.end(), inc);
```

Αυξάνει όλες τις τιμές κατά ένα και τις τυπώνει

Φίλες συναρτήσεις

- Χρησιμοποιώντας το keyword friend, μπορούμε να κάνουμε μια συνάρτηση να βλέπει τα private μέλη μιας κλάσης.
  - Σε ορισμένες περιπτώσεις αυτό είναι βολικό άλλα δεν θα πρέπει να χρησιμοποιείται πολύ συχνά.

Παραδειγμα Set

```
class Person {
private:
   int id;
   string fname;
   string lname;
public:
   Person();
   Person(int);
   void SetDetails(int id, string fn, string ln);
   friend bool operator == (const Person&,const Person &);
   friend bool operator < (const Person &,const Person &);
   int getId() const;
   void PrintDetails() const;
};
```

```
Person::Person() {}

Person::Person(int i)
{ id = i; }

void Person::SetDetails(int i, string f, string l){
  id = i;   fname = f;   lname = l;
}

bool operator == (const Person &p1, const Person &p2)
{ return (p1.id == p2.id); }

bool operator < (const Person &p1, const Person &p2)
{ return (p1.id < p2.id); }

int Person::getId() const
{ return id;}

void Person::PrintDetails() const
{
  cout << "id: " << id
       <<" first name:" << fname << " last name:" << lname << endl;
}
```

```
int main(){
  set<Person> S;
  Person P[3];

  int id;
  string fname, lname;
  for(int i =0 ; i< 3; i ++){
    cin >>id >> fname >> lname;
    P[i].SetDetails(id, fname, lname);
    S.insert(P[i]);
  }

  cin >> id;
  Person sp(id);
  set<Person>::iterator iter = S.find(sp);
  if (iter == S.end()){
    cout << "id not found\n";
  }else{
    iter->PrintDetails();
  }
}
```

Ο κώδικας παραμένει ο ίδιος

ΚΩΔΙΚΑΣ ΣΕ ΠΟΛΛΑ ΑΡΧΕΙΑ

Διαχείριση μεγάλων προγραμμάτων

- Σε μεγάλα projects όπου έχουμε μεγάλη ποσότητα κώδικα και πολλαπλές κλάσεις δεν είναι δυνατόν να έχουμε όλο τον κώδικα σε ένα αρχείο.
- Στον αντικειμενοστρεφή προγραμματισμό ο διαμερισμός του κώδικα σε πολλά αρχεία γίνεται τελείως φυσικά
  - Η κάθε κλάση τοποθετείται σε ξεχωριστό αρχείο.

Επικοινωνία μεταξύ κλάσεων.

- Τι γίνεται όμως όταν μία κλάση Α θέλει να δημιουργήσει αντικείμενα μιας άλλης κλάσης Β?
  - Για να μπορέσει ο compiler να δεσμεύσει μνήμη για το αντικείμενο, θα πρέπει να ξέρει τι είναι αυτό για το οποίο δεσμεύει μνήμη.
  - Άρα η κλάση Α θα πρέπει να ξέρει τον ορισμό της κλάσης Β.
  - Πρέπει να κάνουμε include τον ορισμό της κλάσης Β στην κλάση Α.

Αρχεία επικεφαλίδες (header files)

- Η θεσμοθετημένη πλέον πρακτική είναι για κάθε κλάση να δημιουργείται ένα αρχείο επικεφαλίδα (.h file) που περιέχει τον ορισμό της κλάσης.
  - Π.χ. για την κλάση Person, θα δημιουργήσουμε ένα αρχείο Person.h που θα περιλαμβάνει τον ορισμό της κλάσης Person.
- Ο κώδικας για την υλοποίηση της κλάσης, τοποθετείται σε ένα αρχείο κώδικα (.cpp file)
  - Οι μέθοδοι της Person υλοποιούνται στο αρχείο Person.cpp

Inclusion

- Όποιος χρειάζεται τον ορισμό της Person κάνει include τον ορισμό το αρχείο Person.h
  - `#include "Person.h"`
- Βολεύει οι include εντολές να είναι μέσα στα header files. Μετά το κάθε .cpp αρχείο κάνει include μόνο το δικο του header file
  - Δηλαδή το Person.cpp κάνει include μόνο το Person.h header file.

Πολλαπλοί ορισμοί

- Αν έχουμε πολλά header αρχεία που το ένα κάνει include το άλλο, τότε υπάρχει κίνδυνος σε ένα αρχείο να δηλώσουμε πολλαπλές φορές την ίδια κλάση.
  - Στην περίπτωση αυτή θα πάρουμε λάθος από τον compiler.
- Για να το αποφύγουμε αυτό δίνουμε εντολή στον preprocessor να μην ορίσει δύο φορές την ίδια κλάση

Εντολή προεπεξεργαστή

```
#if !defined(__CLASSNAME__)
#define __CLASSNAME__

Class ClassName
{
        …
        …
        …
}

#endif
```

Με αυτό τον τρόπο εξασφαλίζουμε ότι η κλάση θα οριστεί μόνο μία φορά. Τότε θα οριστεί και η μεταβλητή `__CLASSNAME__` οπότε την επόμενη φορά δεν θα μπούμε μέσα στο if

Το αρχείο Person.h

```
using namespace std;

#if !defined(__PERSON__)
#define __PERSON__

class Person {
private:
  int id;
  string fname;
  string lname;
public:
  Person();
  Person(int);
  void SetDetails(int id, string fn, string ln);
  bool operator == (const Person &) const;
  bool operator < (const Person &) const;
  int getId() const;
  void PrintDetails() const;
};

#endif
```

Το αρχείο Person.cpp

```
#include <iostream>
#include "Person.h"

Person::Person() {}

Person::Person(int i) { id = i; }

void Person::SetDetails(int i, string f, string l)
{ id = i;  fname = f;  lname = l; }

bool Person::operator == (const Person &p2)
{ return (id == p2.id); }

bool Person::operator < (const Person &p2)
{ return (id < p2.id); }

int Person::getId() const
{ return id;}

void Person::PrintDetails() const
{
  cout << "id: " << id
       <<" first name:" << fname << " last name:" << lname << endl;
}
```

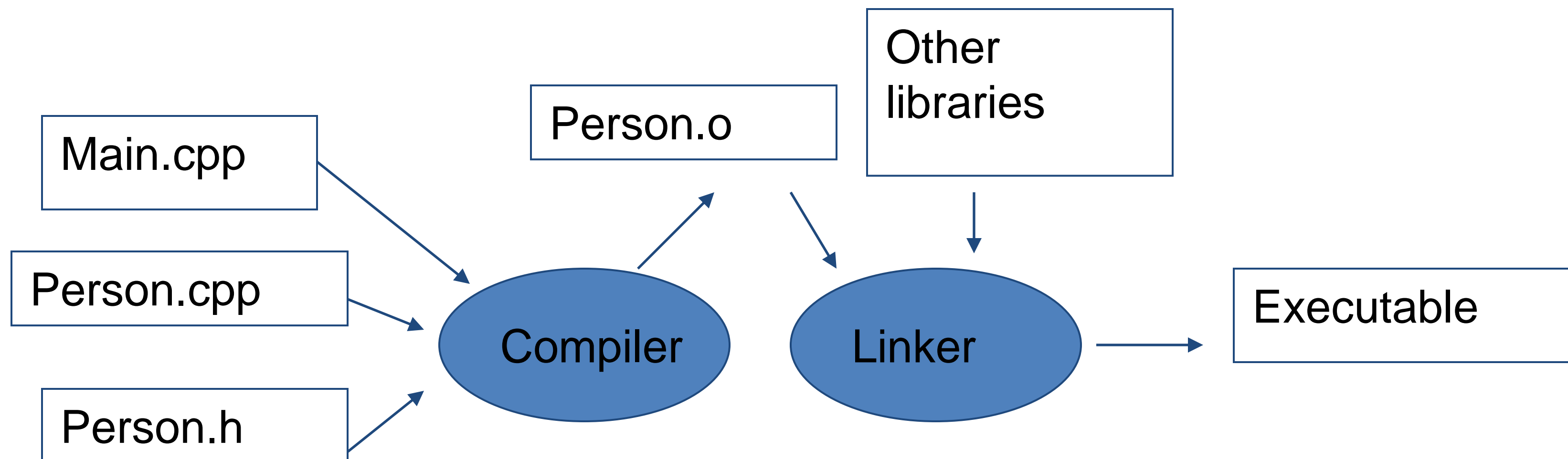
Το αρχείο main.cpp

```
#include <iostream>
#include <set>
#include “Person.h”

int main(){
  set<Person> S;
  Person P[3];

  int id;
  string fname, lname;
  for(int i =0 ; i< 3; i ++){
    cin >>id >> fname >> lname;
    P[i].SetDetails(id, fname, lname);
    S.insert(P[i]);
  }

  cin >> id;
  Person sp(id);
  set<Person>::iterator iter = S.find(sp);
  if (iter == S.end()){
    cout << "id not found\n";
  }else{ iter->PrintDetails(); }
}
```

makefiles

- Τα makefiles ρυθμίζουν τον τρόπο με τον οποίο συνδέουμε πολλά αρχεία μεταξύ τους
- Όταν μεταφράζουμε τον *πηγαίο* (source) κώδικα, ο μεταγλωττιστής παράγει ένα αρχείο με *τελικό* (object) κώδικα.
- Ο linker συνδέει τα αρχεία τελικού κώδικα με τον τελικό κώδικα από τις βιβλιοθήκες και παράγει το εκτελέσιμο.

makefiles



makefiles

- Το αρχείο makefile (ή σχεδόν ισοδύναμα και Makefile) περιέχει εντολές για το πώς θα συνδέσουμε και θα μεταγλωττίσουμε τα αρχεία μας.
- Μερικές *οδηγίες* (options ή directives):

```
# This is a comment
# Options for the compilation of C, C++ programs:
#  -O          Optimize object code
#  -c          Compile and assemble, but do not link
#  -o <file>   Place the output into <file>
```

Χρήση

- Τα περιεχόμενα του τρέχοντος directory είναι:

```
Person.h, Person.cpp, main.cpp, makefile
```

- Η μεταγλώττιση γίνεται ως εξής:

```
make main
```

File: makefile

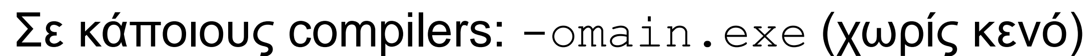
```
# Person.o DEPENDS on Person.h and Person.cpp
# We create only the .o[bject] file and do not
# link it to an executable
# main DEPENDS on the Person.o file and main.cpp
# We compile it and we produce the executable
# main.exe

main: Person.o main.cpp
	g++ Person.o -o main.exe main.cpp
Person.o: Person.h Person.cpp
	g++ -O -c Person.cpp
```

Οι γραμμές αυτές θα πρέπει να ξεκινάνε οπωσδήποτε με tab !

Εναλλακτικά

- Ένας πιο απλός τρόπος για να κάνετε compile κατ' ευθείαν το πρόγραμμά σας:

```
g++ -o main.exe main.cpp Person.cpp
```



Ένα μεγαλύτερο παράδειγμα.

- Θα δούμε πως μπορούμε να σπάσουμε το παράδειγμα με το Array σε πολλαπλά αρχεία.

Utils.h

Ένα αρχείο με διάφορες βιβλιοθήκες που χρειάζονται οι περισσότερες κλάσεις

```
#include <iostream>
#include <cstring>
#include <stdio.h>
#include <cmath>
#include <ctime>
#include <cstdlib>

using namespace std;
```

Element.h

```
#if !defined(__ELEMENT__)
#define __ELEMENT__

class Element
{
public:
   virtual bool operator < (Element &other) = 0;
   virtual void Print() = 0;
};

#endif
```

Για την κλάση Element δεν χρειάζεται να ορίσουμε .cpp αρχείο.

Array.h

```
#if !defined(__ARRAY__)
#define __ARRAY__

#include "Utils.h"
#include "Element.h"

class Array
{
private:
  Element **A;
  int size;
public:
  Array(int s);
  Element *& operator [](int);
  void Sort();
  void Print();
};

#endif
```

Array.cpp

```
#include "Array.h"

Array::Array(int s)
{
  size = s;
  A = new Element*[size];
}

Element *& Array::operator [] (int i)
{ return A[i]; }

void Array::Sort()
{
  for (int i = 0; i < 10; i ++)
    for (int j = i+1; j < 10; j ++)
      if (*A[i] < *A[j]){
        Element *temp = A[i]; A[i] = A[j]; A[j] = temp;
      }
}

void Array::Print()
{
  for (int i = 0; i < 10; i ++) { A[i]->Print();}
}
```

IntElement.h

```
#if !defined(__INT_ELEMENT__)
#define __INT_ELEMENT__

#include "Utils.h"
#include "Element.h"

class IntElement: public Element
{
private:
  int val;
public:
  IntElement(int);
  int GetVal();
  bool operator < (Element &);
  void Print();
};

#endif
```

IntElement.cpp

```
#include "IntElement.h"

IntElement::IntElement(int i)
{ val = i; }

int IntElement::GetVal()
{ return val; }


bool IntElement::operator <(Element &cOther)
{
  IntElement &other = dynamic_cast<IntElement &>(cOther);
  if (val < other.GetVal()){
    return true;
  }
  return false;
}

void IntElement::Print()
{ cout << val << endl; }
```

PointElement.h

```
#if !defined(__POINT_ELEMENT__)
#define __POINT_ELEMENT__

#include "Utils.h"
#include "Element.h"

struct point
{
  int x;
  int y;
};

class PointElement: public Element
{
private:
  point val;
public:
  PointElement(int,int);
  point GetVal();
  bool operator < (Element &);
  void Print();
};

#endif
```

PointElement.cpp

```
#include "PointElement.h"

PointElement::PointElement(int x,int y)
{ val.x = x; val.y = y;}

point PointElement::GetVal()
{ return val; }


bool PointElement::operator <(Element &cOther)
{
  PointElement &other = dynamic_cast<PointElement &>(cOther);
  if (val.x < other.GetVal().x){ return true; }
  if (val.x == other.GetVal().x){
    if (val.y < other.GetVal().y ){
      return true;
    }else { return false; }
  }else { return false; }
}

void PointElement::Print()
{
  cout << val.x << " " << val.y << endl;
}
```

main.cpp

```
#include "Utils.h"
#include "Array.h"
#include "IntElement.h"
#include "PointElement.h"

int main()
{
  srand(time(NULL));
  Array iA(10);
  Array pA(10);

  for (int i = 0; i < 10; i ++){
    iA[i] = new IntElement(rand()%10);
    pA[i] = new PointElement(rand()%10, rand()%10);
  }

  iA.Print(); pA.Print();
  iA.Sort(); pA.Sort();
  iA.Print(); pA.Print();
}
```

makefile

```
main: main.cpp Array.o IntElement.o PointElement.o Utils.h
	g++ -o main.exe Array.o IntElement.o PointElement.o main.cpp
Array.o: Array.h Array.cpp Element.h Utils.h
	g++ -c Array.cpp
IntElement.o: IntElement.h IntElement.cpp Element.h Utils.h
	g++ -c IntElement.cpp
PointElement.o: PointElement.h PointElement.cpp Element.h Utils.h
	g++ -c PointElement.cpp
```