ΠΟΛΥΜΟΡΦΙΣΜΟΣ, ΠΡΟΤΥΠΑ, ΚΑΘΙΕΡΩΜΕΝΗ ΒΙΒΛΙΟΘΗΚΗ ΠΡΟΤΥΠΩΝ

Polymorphism,

Templates,

Standard Template Library (STL)

Πολυμορφισμός

- Τι είναι πολυμορφισμός?
  - (πολυμορφισμός = πολλές μορφές. Η χρήση μιας κλάσης ή μεθόδου με διαφορετικούς τρόπους).
- Δύο είδη πολυμορφισμού:
  - Στατικός Πολυμορφισμός (compile time – early binding)
    - Method Overloading
    - Method Overriding
  - Δυναμικός Πολυμορφισμός (run time – late binding)
    - Virtual methods που αποφασίζονται δυναμικά με βάσει το αντικείμενο.
    - Templates

Method Overriding

- Η παραγόμενη κλάση επανα-ορίζει μια μέθοδο της βασικής κλάσης ώστε να χρησιμοποιεί τα τοπικά δεδομένα.
- Ανάλογα με τον τύπο του αντικειμένου (ή δείκτη) που καλεί τη μέθοδο, καλείτε και η αντίστοιχη μέθοδος.

```
class Person {
private:
  char fname[40];
  char lname[40];
public:
  Person(char fn[], char ln[]);
  char *getPersonalDetails();
  void setPersonalDetails(char fn[], char ln[]);
};

class Employee : public Person
{
private:
  int basicSalary;
public:
  Employee(char fn[], char ln[], int sal);
  int getSalary();
  char *getPersonalDetails();
};


class Customer : public Person
{
private:
  int credit;
  char creditType[10];
public:
  Customer(char fn[], char ln[], char ct[]);
  void chargeCredit(int amount);
  int getCredit();
  char *getCreditType();
  char *getPersonalDetails();
};
```

```
char *Person::getPersonalDetails(){
   char *ret = new char [strlen(fname) + strlen(lname)+ 40];
   sprintf(ret, "1st Name %s - Last Name %s", fname, lname);

   return ret;
}
char *Employee::getPersonalDetails(){
   char *name = Person::getPersonalDetails();
   char * ret = new char [strlen(name) + 10];
   sprintf(ret, "%s %d", name, basicSalary);

   return ret;
}

char *Customer::getPersonalDetails(){
   char *name = Person::getPersonalDetails();
   char * ret = new char [strlen(name) + strlen(creditType) + 10];
   sprintf(ret, "%s %s %d", name, creditType, credit);
   return ret;
}
```

Παράδειγμα

```
int main(){
  char fname[40];
  char lname[40];
  int sal;
  int credit;

  cin >> fname >> lname >> sal;
  Employee john(fname, lname, sal);
  cin >> fname >> lname;
  Customer pete(fname, lname, "VISA");

  pete.chargeCredit(250);

  cout << john.getPersonalDetails() << endl;

  cout << pete.getPersonalDetails() << endl;

  return 0;
}
```

Πιο περίπλοκο παράδειγμα

```
int main(){
  Person *all[10];

  for (int i = 0; i < 10; i++){
    char fname[40];
    char lname[40];
    int sal;
    char choice;
    cin >> choice;
    if (choice == 'p'){
       cin >> fname >> lname;
       all[i] = new Person(fname, lname);
    }
    if (choice == 'e'){
       cin >> fname >> lname >> sal;
       all[i] = new Employee(fname, lname, sal);
    }
    if (choice == 'c'){
       cin >> fname >> lname;
       all[i] = new Customer(fname, lname, "VISA");
     }
  }

 for (int i = 0; i < 10; i++){
   cout << all[i]->getPersonalDetails() << endl;
 }
}
```

Καλείται η μέθοδος της Person
Ο δείκτης καθορίζει τη συνάρτηση που θα κληθεί.

Εικονικές Συναρτήσεις

- Λύση: ορισμός της συνάρτησης ως εικονική (virtual) .
- Όταν μια συνάρτηση ορίζεται ως virtual δίνεται η οδηγία στον compiler ότι η επιλογή της υλοποίησης της δεν θα γίνει κατά τη μετάφραση του προγράμματος, με βάση τον τύπο του δείκτη, αλλά κατά την εκτέλεση του προγράμματος με βάση τον τύπο του αντικειμένου στο οποίο δείχνει ο δείκτης κάθε φορά που εκτελείται ο κώδικας (late binding).

Παράδειγμα

```
# include <iostream>
# include <cstring>

using namespace std;

class Person {
private:
   char fname[40];
   char lname[40];
public:
   Person(char fn[], char ln[]);
   virtual char *getPersonalDetails();
   void setPersonalDetails(char fn[], char ln[]);
};
```

Παράδειγμα

Καλείται η μέθοδος του αντίστοιχου αντικειμένου

```
int main(){
  Person *all[10];

  for (int i = 0; i < 10; i++){
    char fname[40];
    char lname[40];
    int sal;
    char choice;
    cin >> choice;
    if (choice == 'p'){
       cin >> fname >> lname;
       all[i] = new Person(fname, lname);
    }
    if (choice == 'e'){
       cin >> fname >> lname >> sal;
       all[i] = new Employee(fname, lname, sal);
    }
    if (choice == 'c'){
       cin >> fname >> lname;
       all[i] = new Customer(fname, lname, "VISA");
    }
  }

  for (int i = 0; i < 10; i++){
    cout << all[i]->getPersonalDetails() << endl;
  }
}
```

Παράδειγμα – Πέρασμα δείκτη

```
void HTMLFormattedPrint( Person *p ){
  cout << "<HTML>" << p->getPersonalDetails()
       << "</HTML>" << endl;
}
```

Με τη virtual μέθοδο, η συνάρτηση πλέον δουλεύει για οποιοδήποτε αντικείμενο κλάσης Person, Employee, Customer και τυπώνει σωστά τα αντίστοιχα πεδία.
Η συνάρτηση **`HTMLFormattedPrint`** είναι επαναχρησιμοποιήσιμη για οποιαδήποτε παράγωγη κλάση της Person.

Αφηρημένες βασικές κλάσεις

- Μια κλάση ονομάζεται αφηρημένη αν περιέχει τουλάχιστον μια εικονική μέθοδο που δεν περιλαμβάνει υλοποίηση
  - δηλώνεται ως virtual methodName() = 0;
- Η αφηρημένη κλάση λειτουργεί σαν καλούπι για την κατασκευή παραγόμενων που προσφέρουν εναλλακτικές υλοποιήσεις στις εικονικές μεθόδους.
- Η δημιουργία αντικειμένων αφηρημένης κλάσης ΔΕΝ επιτρέπεται από τον compiler
- Η μη υλοποίηση εικονικών μεθόδων από τις παράγωγες κλάσεις ΔΕΝ επιτρέπεται από τον compiler

Παράδειγμα

```
# include <iostream>
# include <cstring>

using namespace std;

class Person {
private:
   char fname[40];
   char lname[40];
public:
   Person(char fn[], char ln[]);
   char *getPersonalDetails();
   virtual char* getFinancialDetails() = 0;
   // γνησια εικονική μέθοδος χωρίς υλοποίηση
   void setPersonalDetails(char fn[], char ln[]);
};
```

Παράδειγμα

```
class Employee : public Person
{
private:
  int basicSalary;
public:
  Employee(char fn[], char ln[], int sal);
  int getSalary();
  char *getFinancialDetails();
};

class Customer : public Person
{
private:
  int credit;
  char creditType[10];
public:
  Customer(char fn[], char ln[], char ct[]);
  void chargeCredit(int amount);
  int getCredit();
  char *getCreditType();
  char *getFinancialDetails();
};
```

Παράδειγμα

```
Person::Person(char fn[], char ln[]){
  strcpy(fname, fn);
  strcpy(lname, ln);
}

void Person::setPersonalDetails(char fn[], char ln[]){
  strcpy(fname, fn);
  strcpy(lname, ln);
}

char *Person::getPersonalDetails(){
  char *ret = new char [strlen(fname) + strlen(lname) +40];
  sprintf(ret, "1st Name %s - Last Name %s", fname, lname);

  return ret;
}

// Δεν υπάρχει υλοποιηση της getFinancialDetails
// ούτε καν τετριμένη!!!
```

Παράδειγμα

```
Employee::Employee(char fn[], char ln[], int sal) :
                    Person(fn, ln) {
  basicSalary = sal;
}

int Employee::getSalary(){
  return basicSalary;
}

char *Employee::getFinancialDetails(){
  char * ret = new char [30];
  sprintf(ret, " Basic Salary %d ", basicSalary);

  return ret;
}

// Αν δεν ορίσουμε την getFinancialDetails θα έχουμε error
```

Παράδειγμα

```
Customer::Customer(char fn[], char ln[], char ct[]) : Person(fn, ln) {
  credit = 0;
  strcpy(creditType, ct);
}
void Customer::chargeCredit(int amount){
  credit -= amount;
}
int Customer::getCredit(){
  return credit;
}
char *Customer::getCreditType(){
  return creditType;
}
```

```
char *Customer::getFinancialDetails(){
   char * ret = new char [strlen(creditType) + 40];
   sprintf(ret, "Credit Type %s Credit %d", creditType, credit);
   return ret;
}

// Αν δεν ορίσουμε την getFinancialDetails θα έχουμε error
```

Παράδειγμα

```
void HTMLFormattedPrint (Person *p){
  cout << "<HTML>" << p->getPersonalDetails()
       << "</HTML>" << endl;
  cout << "<HTML>" << p->getFinancialDetails()
       << "</HTML>" << endl;
}
```

Η χρηση των εικονικών κλάσεων γίνεται μόνο με δείκτη!
Δεν μπορουμε να περάσουμε αντικείμενο
Ουτε καν δείκτη σε δείκτη

Παράδειγμα

```
int main(){
  char fname[40];
  char lname[40];
  int sal;

  cin >> fname >> lname >> sal;
  Employee john(fname, lname, sal);
  cin >> fname >> lname;
  Customer pete(fname, lname, "VISA");
  pete.chargeCredit(250);

  // Person mary(fname, lname) θα δώσει compile error!

  HTMLFormattedPrint (&pete, 1);
  HTMLFormattedPrint (&john, 1);
  HTMLFormattedPrint (&pete, 2);
  HTMLFormattedPrint (&john, 2);

  return 0;
}
```

ΠΑΡΑΔΕΙΓΜΑ

Πρόβλημα

- Υλοποιήστε ένα πίνακα ο οποίος μας δίνει τις εξής δυνατότητες:
  - Μπορούμε να αποθηκεύσουμε δεδομένα οποιαδήποτε μορφής.
  - Μπορούμε να γράψουμε και να διαβάσουμε κάποιο στοιχείο.
  - Μπορούμε να τυπώσουμε τα στοιχεία του πίνακα
  - Μπορούμε να ταξινομήσουμε (sort) τα στοιχεία του πίνακα

Η κλάση Array

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

Τι τύπο δεδομένων θα πρέπει να κρατάει ο πίνακας Α?

Η κλάση Array

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

Τι είναι η κλάση Element?

```
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;
      }
    }
  }
}
```

Abstract Class Element

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

Απλοποίηση

- Υλοποιήστε ένα πίνακα ο οποίος μας δίνει τις εξής δυνατότητες:
  - Μπορούμε να αποθηκεύσουμε δεδομένα οποιαδήποτε μορφής.
  - Μπορούμε να γράψουμε και να διαβάσουμε κάποιο στοιχείο.
  - Μπορούμε να τυπώσουμε τα στοιχεία του πίνακα
  - Μπορούμε να ταξινομήσουμε (sort) τα στοιχεία του πίνακα
- Απλοποίηση: θα εξετάσουμε την περίπτωση που θέλουμε να κρατάμε μονοδιάστατα, ή δισδιάστατα σημεία

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;
}
```

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;
}
```

Dynamic Casting

```
pointerD = dynamic_cast<D *>(pointerB)
```

- **pointerB** είναι δεικτης της κλάσης B, **pointerD** είναι δείκτης της κλάσης D.
- Η κλάση D παράγεται από την κλάση B
- Η κλάση B είναι “πολυμορφική” δηλαδή περιέχει τουλάχιστον μια εικονική συνάρτηση.

- Μετατροπή του δεικτη της κλάσης B σε δείκτη της κλάσης D

Παράδειγμα χρήσης

```
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();
}
```

Υλοποίηση της Array με αναφορές

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

```
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;
      }
    }
  }
}
```

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;
}
```

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;
}
```

Πρόβλημα

- Τι γίνεται αν ο πίνακας μας έχει και μονοδιάστατα και δισδιάστατα σημεία?

Παράδειγμα

```
int main()
{
  srand(time(NULL));
  Array A(10);

  for (int i = 0; i < 5; i ++){
    A[i] = new IntElement(rand()%10);
  }
  for (int i = 5; i < 10; i ++){
    A[i] = new PointElement(rand()%10, rand()%10);
  }

  A.Print(); //οκ! Τυπώνει και τα δύο
  A.Sort(); // Segmentation fault! (δεν μπορεί να
            // συγκρίνει μονοδιάστατα με δισδιάστατα σημεία)
            // το dynamic casting προκαλεί λάθος
  A.Print();
}
```

Πρόβλημα

- Τι γίνεται αν ο πίνακας μας έχει και μονοδιάστατα και δισδιάστατα σημεία?
  - Ο compiler το επιτρέπει.
  - Βολεύει αν ενδιαφερόμαστε μόνο για εκτύπωση.
  - Το πρόγραμμα χτυπάει αν προσπαθήσουμε να κάνουμε sort.
- Σε κάποιες περιπτώσεις θέλουμε να ξέρουμε ακριβώς τι δεδομένα έχουμε στον πίνακα.

TEMPLATES

Class Templates

- Με τα Class Templates μπορούμε να ορίσουμε μία κλάση ή οποία να δουλεύει για πολλούς διαφορετικούς τύπους δεδομένων στα πεδία της.
  - Χρησιμοποιούνται κυρίως για κλάσεις «αποδέκτες» οι οποίες αποθηκεύουν δεδομένα
    - Όπως η κλάση Array που δημιουργήσαμε.

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();
  }
}
```

STANDARD TEMPLATE LIBRARY (STL)

Standard Template Library

- Η STL είναι μια βιβλιοθήκη που υλοποιεί διάφορους αφηρημένους τύπους δεδομένων και δομές δεδομένων.
- Μπορούμε να τη χρησιμοποιούμε για να αποθηκεύουμε κάθε μορφής δεδομένα. Υλοποιείται χρησιμοποιώντας Templates.

Βασικές Έννοιες

- 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

Containers

- Έχουμε δύο τύπους από containers
  - Sequential (ακολουθιακούς): Αποθηκεύουν τα δεδομένα σε μια ακολουθία, υπάρχει η έννοια του πρώτου, δεύτερου, κλπ.
    - Πχ ένας απλός πίνακας είναι μια μορφή ακολουθιακού αποδέκτη.
  - Associative (συνειρμικοί): Η πρόσβαση στα δεδομένα γίνεται μέσω κλειδιών
    - Π.χ. όπως με τις ταχυδρομικές θυρίδες, ή τις διευθύνσεις
  - +Container Adapters: Containers που χτίζονται πάνω στους υπάρχοντες containers (π.χ., Stack)

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	Λειτουργία
`size()`	επιστρέφει τον αριθμό των στοιχείων μέσα στον πίνακα
`push_back()`	προσθέτει ένα στοιχείο στο τέλος του πίνακα
`pop_back()`	αφαιρεί το τελευταίο στοιχείο του πίνακα
`back()`	επιστρέφει το τελευταίο στοιχείο του πίνακα
`operator []`	τυχαία πρόσβαση στα στοιχεία του πίνακα
`empty()`	επιστρέφει true αν το vector είναι άδειο
`insert()`	προσθέτει ένα στοιχείο σε ενδιάμεση θέση (χρησιμοποιώντας iterator)
`erase()`	αφαιρεί ένα στοιχείο από ενδιάμεση θέση (χρησιμοποιώντας iterator)

Παράδειγμα

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

using namespace std;

int main(){
  vector<int> v;
  int x;

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

  v[v.size() - 1] = 0;

  cout << "vector elements: ";
  for (int i = 0; i < v.size(); i ++){
    cout << v[i] << " ";
  }
  cout << endl;
}
```

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

using namespace std;

int main(){
  vector<int> v;
  int x;

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

  v.pop_back();

  cout << "vector elements: ";
  for (int i = 0; i < v.size(); i ++){
    cout << v[i] << " ";
  }
  cout << endl;
}
```

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

using namespace std;

int main(){
  vector<int> v;
  int x;

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

  v.pop_back();
  if (v.empty()) {
    cout << “vector is empty\n”;
  } else {
    cout << "vector elements: ";
    for (int i = 0; i < v.size(); i ++){
      cout << v[i] << " ";
    }
    cout << endl;
  }
}
```

list

Όλες τις μεθόδους του vector εκτός από τον operator [], και επιπλέον:

Μέθοδος	Λειτουργία
`push_front()`	προσθέτει ένα στοιχείο στην αρχή της λίστας
`pop_front()`	αφαιρεί το πρώτο στοιχείο της λίστας.
`front()`	επιστρέφει το πρώτο στοιχείο της λίστας.

Iterators

- Οι iterators μας επιτρέπουν πρόσβαση (δείκτες) στα δεδομένα των αποθηκευτών.
- Διαφορετικοί containers έχουν διαφορετικής δύναμης iterators.
  - Αυτοί που επιτρέπουν τυχαία πρόσβαση, έχουν iterators τυχαίας πρόσβασης
  - Ο πιο δυνατός iterator δημιουργείτε για κάθε container αυτόματα.

Παράδειγμα

`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

```
#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 ++){
    cout << *iter << " ";
  }
  cout << endl;
}
```

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;
}
```

Παραδειγμα map

```
class Person {
private:
  string fname;
  string lname;
public:
  Person(string fn, string ln);
  void PrintPersonalDetails();
};

Person::Person(string f, string l){
  fname = f;
  lname = l;
}

void Person::PrintPersonalDetails(){
  cout << "first name -- " << fname
       << " last name -- " << lname << endl;
}
```

Παραδειγμα map

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

int main(){
  map<string,Person*> M;
  string fname,lname;

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

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

Iterators map

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

int main(){
  map<string,Person*> M;
  string fname,lname;

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

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

(*iter).first: Το κλειδί του map στη θέση στην οποία δείχνει ο iterator

(*iter).second: Η τιμή του map στη θέση στην οποία δείχνει ο iterator

STL με δικές μας κλάσεις

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

using namespace std;

int main(){
  vector<Person*> v;

  string fname,lname;
  while (!cin.eof()){
    cin >> fname >> lname;
    Person *p =new Person(fname,lname);
    v.push_back(p);
  }

  for (vector<Person*>::iterator i = v.begin(); i != v.end(); i ++){
    (*i)->PrintPersonalDetails();
  }
}
```