ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ 20256

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

Διδάσκοντες:

Βασίλης Βεσκούκης, Καθ.
Νίκος Λεονάρδος, Επ.Καθ.
Νίκος Παπασπύρου, Καθ.
Σωτήρης Κοκόσης, ΕΔΙΠ
Πέτρος Ποτίκας, ΕΔΙΠ
Γιώργος Σιόλας, ΕΔΙΠ

progtech@courses.softlab.ntua.gr

20/3/2026

Διαφάνειες παρουσιάσεων

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

Πίνακες ως ΑΤΔ
(αφηρημένοι τύποι δεδομένων)

Πίνακες ως ΑΤΔ

- Σύνολο «δεικτών» (indices), $\mathbf{I}$
- Σύνολο «τιμών» (values), $\mathbf{V}$
- Βασική πράξη: προσπέλαση στοιχείου

$$a[i] \in \mathbf{V} \qquad \text{όπου } i \in \mathbf{I}$$

- Μονοδιάστατοι πίνακες: $\mathbf{I} = \{1, 2, 3, \ldots n\}$
- Πολυδιάστατοι πίνακες (d διαστάσεων):

$$\mathbf{I} = \{1, 2, 3, \ldots n_1\} \times \{1, 2, 3, \ldots n_2\} \times \ldots \times \{1, 2, 3, \ldots n_d\}$$

Πίνακες ως ΑΤΔ

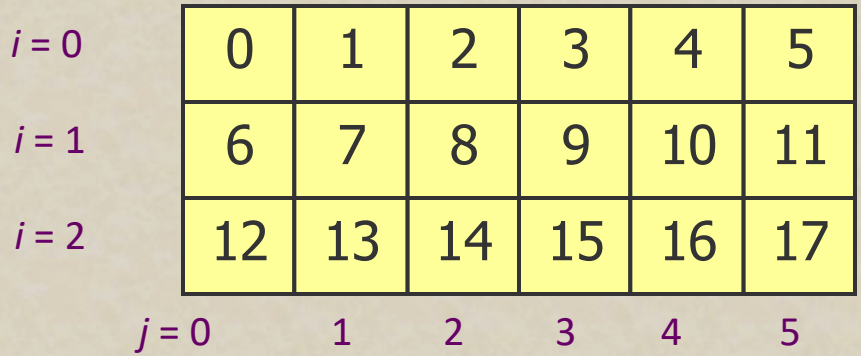
- Συνήθως υλοποιούνται με κάποιο ΣΤΔ πινάκων (arrays)
- Ο ΣΤΔ του μονοδιάστατου πίνακα επαρκεί για την υλοποίηση κάθε ΑΤΔ πίνακα

```
int data[SIZE];                 // στατικά
int *data = new int[SIZE];      // δυναμικά
```

  - Κόστος προσπέλασης: O(1)
  - Συνάρτηση *loc* : $\mathbf{I} \rightarrow \{0, 1, 2, \ldots \mathtt{SIZE} - 1\}$ υπολογίζει τη θέση ενός στοιχείου του ΑΤΔ πίνακα στο ΣΤΔ πίνακα της υλοποίησης

Πίνακες ως ΑΤΔ

- ΑΤΔ πίνακα δύο διαστάσεων $n \times m$



- Αρίθμηση κατά γραμμές

$$loc\,(i,j) = cols * i + j$$

$$i = loc \,/\, cols,\;\; j = loc \,\%\, cols$$

Πίνακες ως ΑΤΔ

(0,0)->**0**	(0,1)->**1**	(0,2)->**2**	(0,3)->**3**	(0,4)->**4**	(0,5)->**5**
(1,0)->**6**	(1,1)->**7**	(1,2)->**8**	(1,3)->**9**	(1,4)->**10**	(1,5)->**11**
(2,0)->**12**	(2,1)->**13**	(2,2)->**14**	(2,3)->**15**	(2,4)->**16**	(2,5)->**17**

```
for (i=0; i < rows; i++)
  for (j=0; j < cols; j++)
    loc = cols * i + j ;
```

```
for (loc=0; loc < rows*cols; loc++){
  i = loc / cols;
  j = loc % cols;
}
```

0 ← (0,0)	1 ← (0,1)	2 ← (0,2)	3 ← (0,3)	4 ← (0,4)	5 ← (0,5)	6 ← (1,0)	7 ← (1,1)	8 ← (1,2)	9 ← (1,3)	10 ← (1,4)	11 ← (1,5)	12 ← (2,0)	13 ← (2,1)	14 ← (2,2)	15 ← (2,3)	16 ← (2,4)	17← (2,5)

Πίνακες ως ΑΤΔ

- ΑΤΔ πίνακα δύο διαστάσεων $n \times m$

- Εναλλακτικά, αρίθμηση κατά στήλες

$$loc\,(i,j) = \text{rows} * j + i$$

$$i = loc\ \%\ rows,\ \ j = loc\ /\ rows$$

Πίνακες ως ΑΤΔ

(0,0)->**0**	(0,1)->**3**	(0,2)->**6**	(0,3)->**9**	(0,4)->**12**	(0,5)->**15**
(1,0)->**1**	(1,1)->**4**	(1,2)->**7**	(1,3)->**10**	(1,4)->**13**	(1,5)->**16**
(2,0)->**2**	(2,1)->**5**	(2,2)->**8**	(2,3)->**11**	(2,4)->**14**	(2,5)->**17**

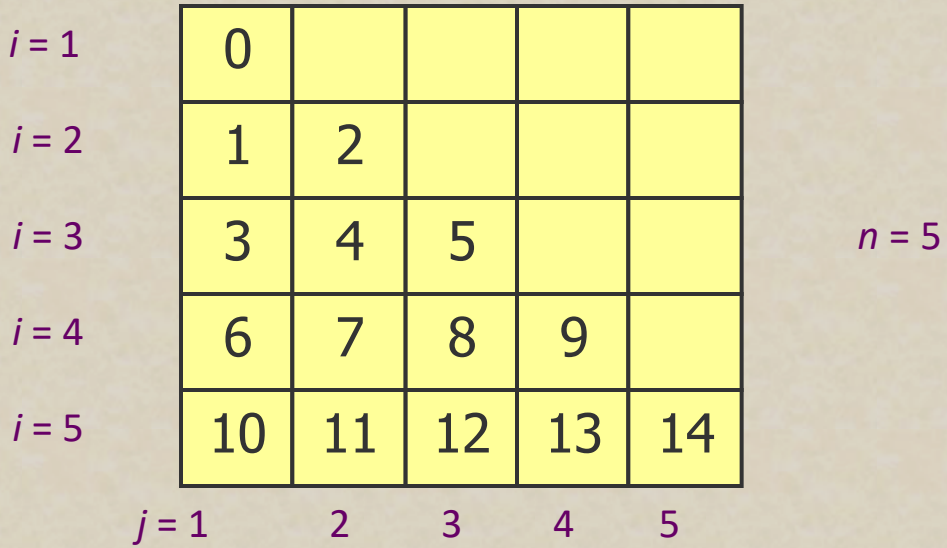
```
for (j=0; j < cols; j++) {
  for (i = 0; i < rows; i++)
    loc = rows * j + i;
```

```
for (loc=0; loc < rows*cols; loc++){
  j = loc / rows;
  i = loc % rows;
}
```

0 ← (0,0)	1 ← (1,0)	2 ← (2,0)	3 ← (0,1)	4 ← (1,1)	5 ← (2,1)	6 ← (0,2)	7 ← (1,2)	8 ← (2,2)	9 ← (0,3)	10 ← (1,3)	11 ← (2,3)	12 ← (0,4)	13 ← (1,4)	14 ← (2,4)	15 ← (0,5)	16 ← (1,5)	17← (2,5)

Πίνακες ως ΑΤΔ

- ΑΤΔ κάτω τριγωνικού πίνακα $n \times n$

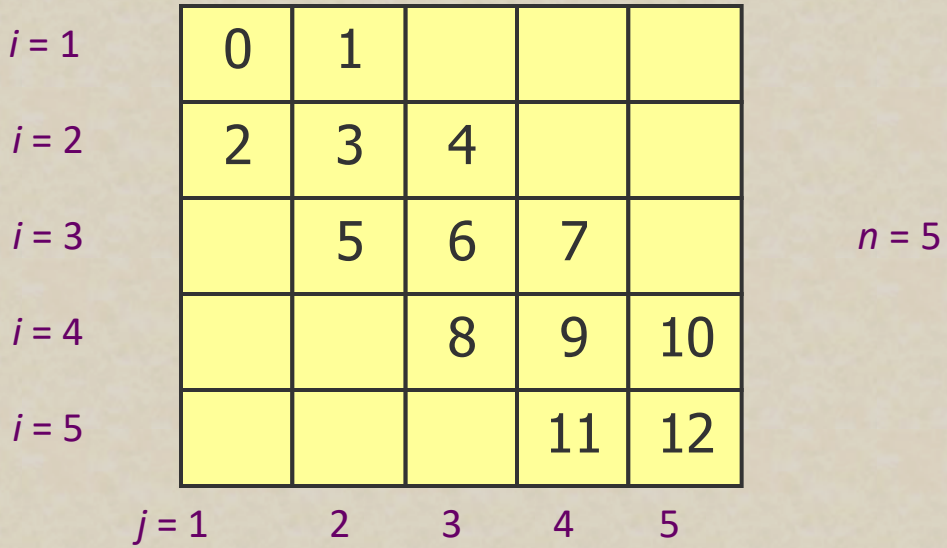


$$loc\,(i,j) = i\,(i-1)/2 + j - 1$$

- Ομοίως για συμμετρικούς πίνακες

Πίνακες ως ΑΤΔ

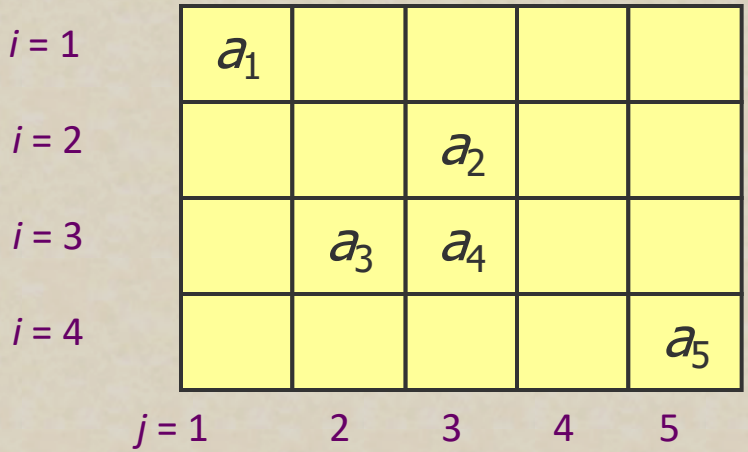
- ΑΤΔ τριδιαγώνιου πίνακα $n \times n$



$$loc\,(i,j) = 3\,(i-1) - 1 + j - i + 1$$
$$= 2\,i + j - 3$$

Πίνακες ως ΑΤΔ

- ΑΤΔ αραιού πίνακα $n \times m$



$rows = 4$

$cols = 5$

- Υλοποίηση με δυαδικό πίνακα
- Υλοποίηση με τρεις πίνακες

Κόστος προσπέλασης;

$$row = [\ 1, 2, 3, 3, 4\ ] \qquad col = [\ 1, 3, 2, 3, 5\ ]$$

$$val = [\ a_1, a_2, a_3, a_4, a_5\ ]$$

Do it yourself!

Υλοποίηση ΑΤΔ πινάκων

ΑΤΔ για μονοδιάστατους πίνακες (i)

```
template <typename T>
class Array {
public:
  Array(unsigned n = 0, int b = 0);
  Array(const Array &a);
  ~Array();
  Array & operator=(const Array &a);
  T & operator[](int i);
  const T & operator[](int i) const;
};
```

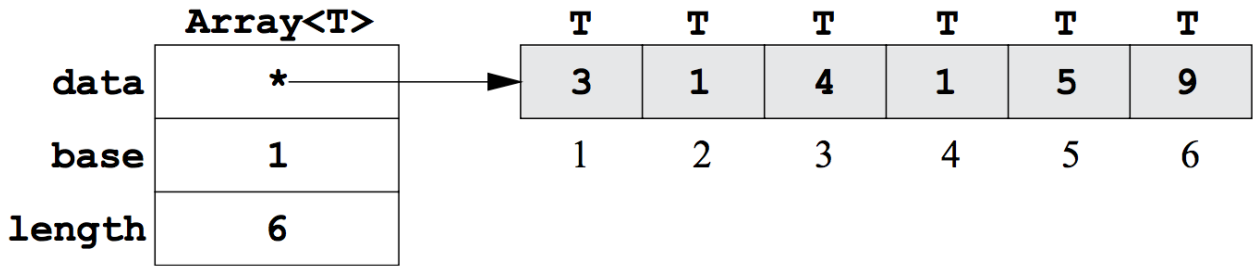
Βλ. συνολικό κώδικα στην ιστοσελίδα του μαθήματος.

ΑΤΔ για μονοδιάστατους πίνακες (ii)

```
protected:
   T *data;
   int base;
   unsigned length;
   unsigned loc(int i) const;
                        // may throw out_of_range
```

ΣΧΗΜΑ 4.1

Σχηματική αναπαράσταση της δομής αντικειμένου τύπου **Array<T>**



ΑΤΔ για μονοδιάστατους πίνακες (iii)

```
Array(unsigned n = 0, int b = 0)
  : data(new T[n]), base(b), length(n) {}

Array(const Array &a)
  : data(new T[a.length]), base(a.base),
    length(a.length) {
  for (unsigned i = 0; i < length; ++i)
    data[i] = a.data[i];
}

~Array() {
  delete [] data;
}
```

ΑΤΔ για μονοδιάστατους πίνακες (iv)

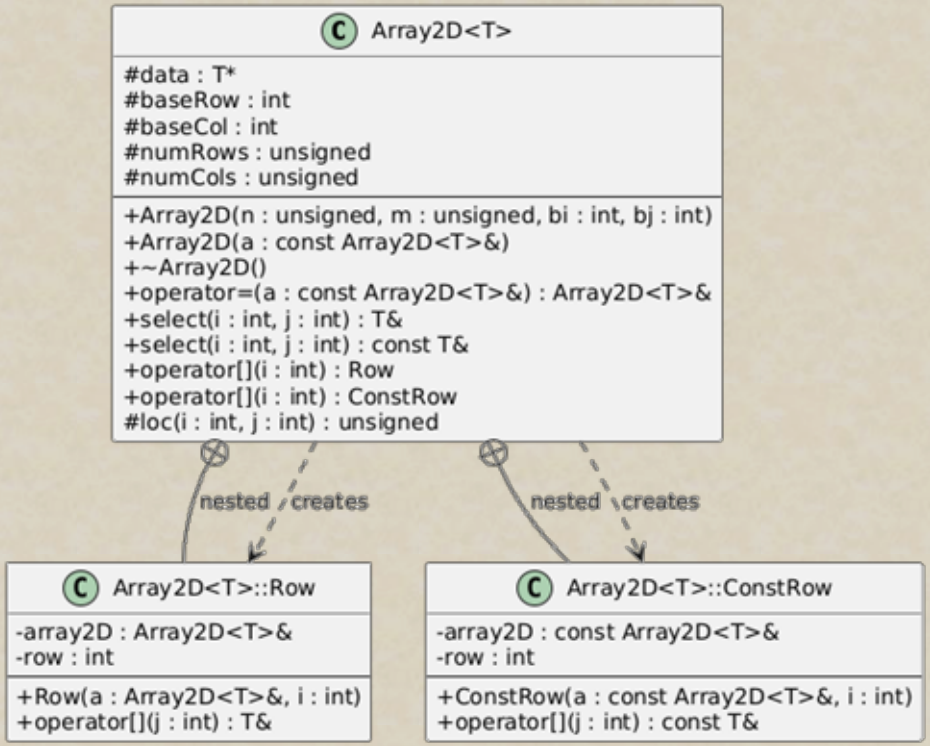
```
Array & operator=(const Array &a) {
  if (&a != this) {
    delete [] data;
    base = a.base;
    length = a.length;
    data = new T[length];
    for (unsigned i = 0; i < length; ++i)
      data[i] = a.data[i];
  }
  return *this;
}
```

ΑΤΔ για μονοδιάστατους πίνακες (v)

```
unsigned loc(int i) const {
  int di = i - base;
  if (di < 0 || di >= length)
    throw out_of_range("invalid index");
  return di;
}
T & operator[](int i) {
  return data[loc(i)];
}
const T & operator[](int i) const {
  return data[loc(i)];
}
```

ΑΤΔ για διδιάστατους πίνακες (i)

- Class template Array2D<T>
  - Constructor με ορίσματα
  - Copy constructor
  - Destructor
  - Μέθοδοι για const αντικείμενα
  - Το overload του [][] δεν είναι προφανές (γιατί;)



ΑΤΔ για διδιάστατους πίνακες (ii)

```
template <typename T>
class Array2D {
public:
  Array2D(unsigned n = 0, unsigned m = 0,
          int bi = 0, int bj = 0);
  Array2D(const Array2D &a);
  ~Array2D();
  Array2D & operator=(const Array2D &a);
  T & select(int i, int j);
  const T & select(int i, int j) const;
  Row operator[](int i);
  ConstRow operator[](int i) const;
};
```

Βλ. συνολικό κώδικα στην ιστοσελίδα του μαθήματος.

ΑΤΔ για διδιάστατους πίνακες (iii)

```
protected:
  T *data;
  int baseRow, baseCol;
  unsigned numRows, numCols;

  unsigned loc(int i, int j) const;
                    // may throw out_of_range
```

ΑΤΔ για διδιάστατους πίνακες (iv)

```
Array2D(unsigned n = 0, unsigned m = 0,
        int bi = 0, int bj = 0)
  : data(new T[n*m]), baseRow(bi),
    baseCol(bj), numRows(n), numCols(m) {}
Array2D(const Array2D &a)
  : data(new T[a.numRows * a.numCols]),
    baseRow(a.baseRow), baseCol(a.baseCol),
    numRows(a.numRows), numCols(a.numCols) {
  for (unsigned i = 0;
       i < numRows * numCols; ++i)
    data[i] = a.data[i];
}
~Array2D() { delete [] data; }
```

ΑΤΔ για διδιάστατους πίνακες (v)

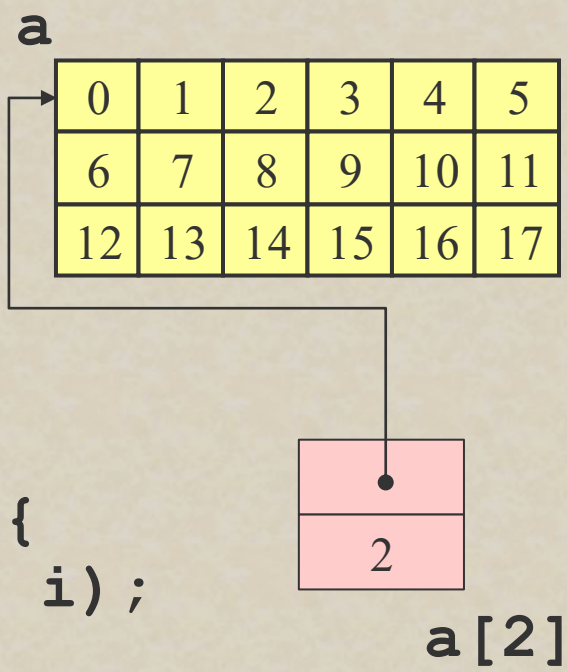
```
unsigned loc(int i, int j) const {
  int di = i - baseRow, dj = j - baseCol;
  if (di < 0 || di >= numRows ||
      dj < 0 || dj >= numCols)
    throw out_of_range("invalid index");
  return di * numCols + dj;
}
T & select(int i, int j) {
  return data[loc(i, j)];
}
const T & select(int i, int j) const {
  return data[loc(i, j)];
}
```

ΑΤΔ για διδιάστατους πίνακες (vi)

```
Array2D & operator=(const Array2D &a) {
  if (&a != this) {
    delete [] data;
    baseRow = a.baseRow;
    baseCol = a.baseCol;
    numOfRows = a.numRows;
    numOfCols = a.numCols;
    data = new T[numRows * numCols];
    for (unsigned i = 0;
         i < numRows * numCols; ++i)
      data[i] = a.data[i];
  }
  return *this;
}
```

ΑΤΔ για διδιάστατους πίνακες (vii)

```
Row operator [] (int i) {
  return Row(*this, i);
}

class Row {
public:
  Row(Array2D &a, int i)
    : array2D(a), row(i) {}
  T & operator[](int i) const {
    return array2D.select(row, i);
  }

private:
  Array2D &array2D;
  int row;
};
```



ΑΤΔ για διδιάστατους πίνακες (viii)

```
ConstRow operator[](int i) const {
  return ConstRow(*this, i);
}

class ConstRow {
public:
  ConstRow(const Array2D &a, int i)
    : array2D(a), row(i) {}
  const T & operator[](int i) const {
    return array2D.select(row, i);
  }

private:
  const Array2D &array2D;
  int row;
};
```

Ίδιο με το **Row**
αλλά για σταθερούς πίνακες!

ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ 20256

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

Διδάσκοντες:

Βασίλης Βεσκούκης, Καθ.
Νίκος Λεονάρδος, Επ.Καθ.
Νίκος Παπασπύρου, Καθ.
Σωτήρης Κοκόσης, ΕΔΙΠ
Πέτρος Ποτίκας, ΕΔΙΠ
Γιώργος Σιόλας, ΕΔΙΠ

progtech@courses.softlab.ntua.gr

20/3/2026

Διαφάνειες παρουσιάσεων

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

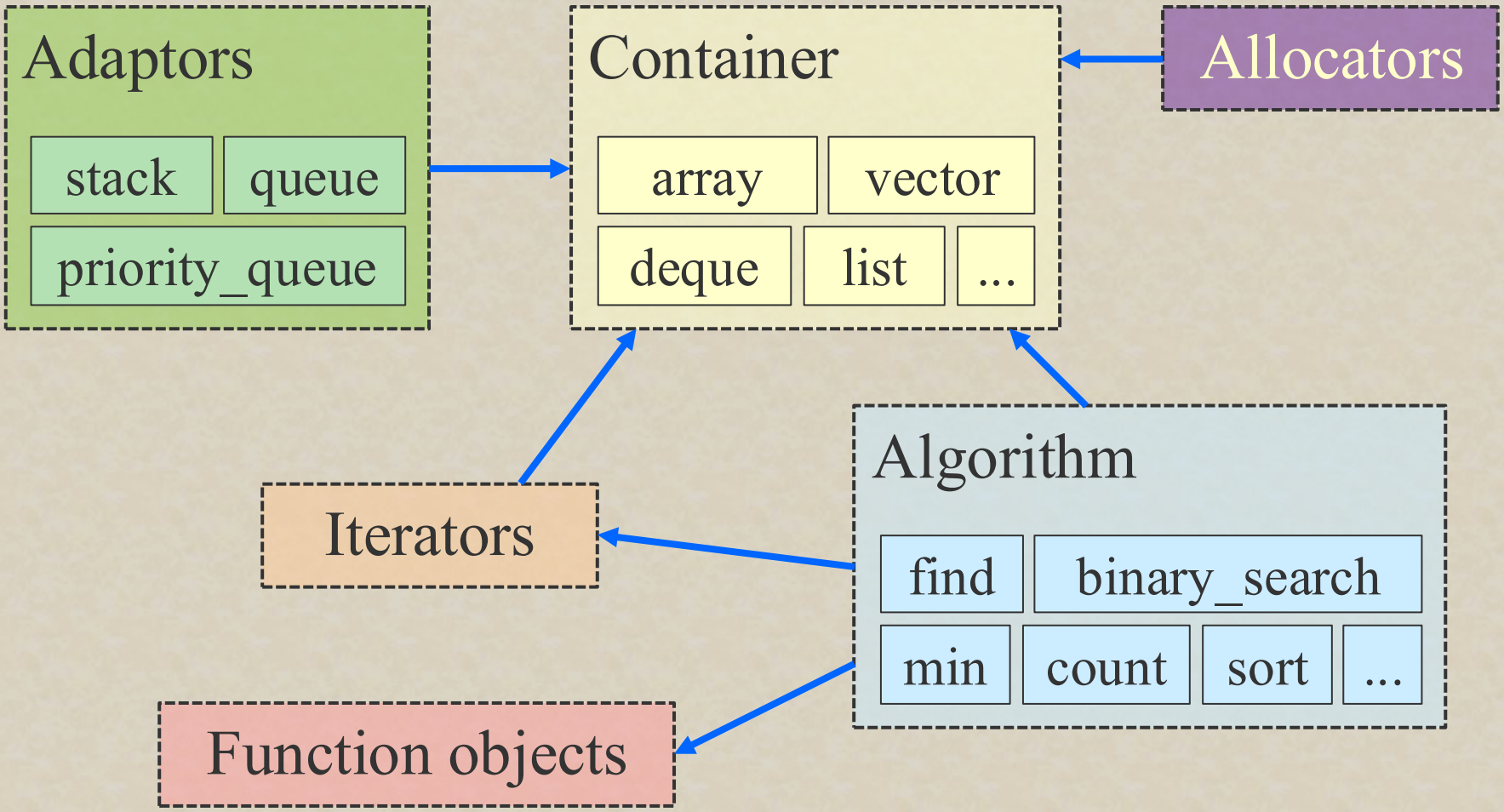
Standard Template Library

Εισαγωγή 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
  - operator overloading
  - 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 (wrapper) 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 << "size=" << s.size()
     << " length=" << s.length() << endl;
```

```
size=42 length=42
```

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

```
o
```

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

```
be
```

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

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

Συμβολοσειρά

```
s.insert(34, " indeed");
cout << s << endl;
```

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

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

```
35
```

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

```
-1
```

```
const char *raw_pointer = s.data();  // or s.c_str()
cout << raw_pointer << endl;
```

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

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

```
1 52
```

Containers: 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";
```

Containers: Πίνακας

- Χρήσιμες μέθοδοι της κλάσης array
  - size(): επιστρέφει το μέγεθος του πίνακα
  - at(): λειτουργεί ως getter και setter
  - operator[]: όπως το at() αλλά χωρίς έλεγχο ορίων

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

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

Containers: Πίνακας

- **for()** για container(*)

```
array<int, 10> a;

for (int i=0; i < a.size(); i++)
      a.at(i) = i * 10;

// for (int i=0; i < a.size(); i++)
//    cout << a.at(i) << " ";

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

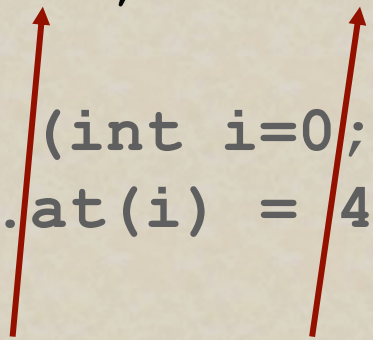
```
for (type var : container)
   ... do something
```

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

Containers: Πίνακας

- **for()** για container(*)

```
array<int, 10> a;

// for (int i=0; i < a.size(); i++)
//    a.at(i) = 42;

for (int &d : a)
     d = 42;

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



```
for (type var : container)
   ... do something
```

```
42 42 42 42 42 42 42 42 42 42
```

Containers: Πίνακας

- 2+ διαστάσεις

```
array<int, 10> a;

array<array<int, 10>, 10> b;

for (int i=0; i<10; i++)
    for (int j=0; j<10; j++)
        b[i][j] = i + j;
     // b.at(i).at(j) = i + j;

for (array<int,10> row : b) {
    for (int d : row)
        cout << d << "\t";
    cout << endl;
}

for (const array<int,10> &row : b) ...
```

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

Containers: Διάνυσμα

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

```
void showVector(const vector<int> &v);
```

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

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

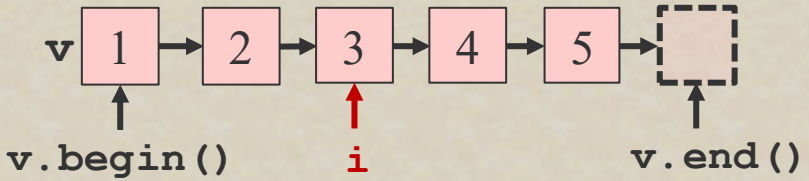
Containers: Διάνυσμα

```
#include <vector>
...
                    ΠΛΗΘΟΣ                ΑΡΧΙΚΗ ΤΙΜΗ
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 iterable object
  - συμπεριφέρονται σαν να ήταν δείκτες

```
void showVector(const 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)

```
vector<int> v;

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

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

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


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

Containers: 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
```

```
cout << v.capacity() << endl;
```

```
8
```

γενικά ≥ size, implementation dependent

```
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
```

```
r.reserve(100);
cout << r.size() << " " << r.capacity() << endl;
```

```
3 100
```

Containers: 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`**

Containers: Λίστα

```
void showList(const 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
```

Containers: 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
```

- Τύπος **`forward_list<T>`**
  - απλά συνδεδεμένη λίστα
- Τύπος **`deque<T>`**
  - όπως το vector αλλά με αποδοτική εισαγωγή και αφαίρεση στην αρχή και στο τέλος

STL και templates

```
template <typename T>
void showAnything(const T &iterable) {
  cout << "size= " << iterable.size() << "  contents= ";
  typename T::iterator it;
  for (it = iterable.begin(); it != iterable.end(); ++it)
    cout << *it << " ";
  cout << endl;
}
```

Ε: Τι απαιτείται για να λειτουργεί η **showAnything;**

Α: - το **T** να είναι iterable object
- να έχει οριστεί (αν απαιτείται) ο τελεστής **<<**

```
template <typename T>
void showAnything(const T &iterable) {
  cout << "size= " << iterable.size() << "  contents= ";
  for (auto element : iterable) cout << element << " ";
  cout << endl;
}
```

Στοίβα και ουρά

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

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

```
40 30 20 10 0
```

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 showQueue(queue<int> q) {
  while (!q.empty()) {
    cout << q.front() << " ";
    q.pop();
  }
}

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

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