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

15/4/22

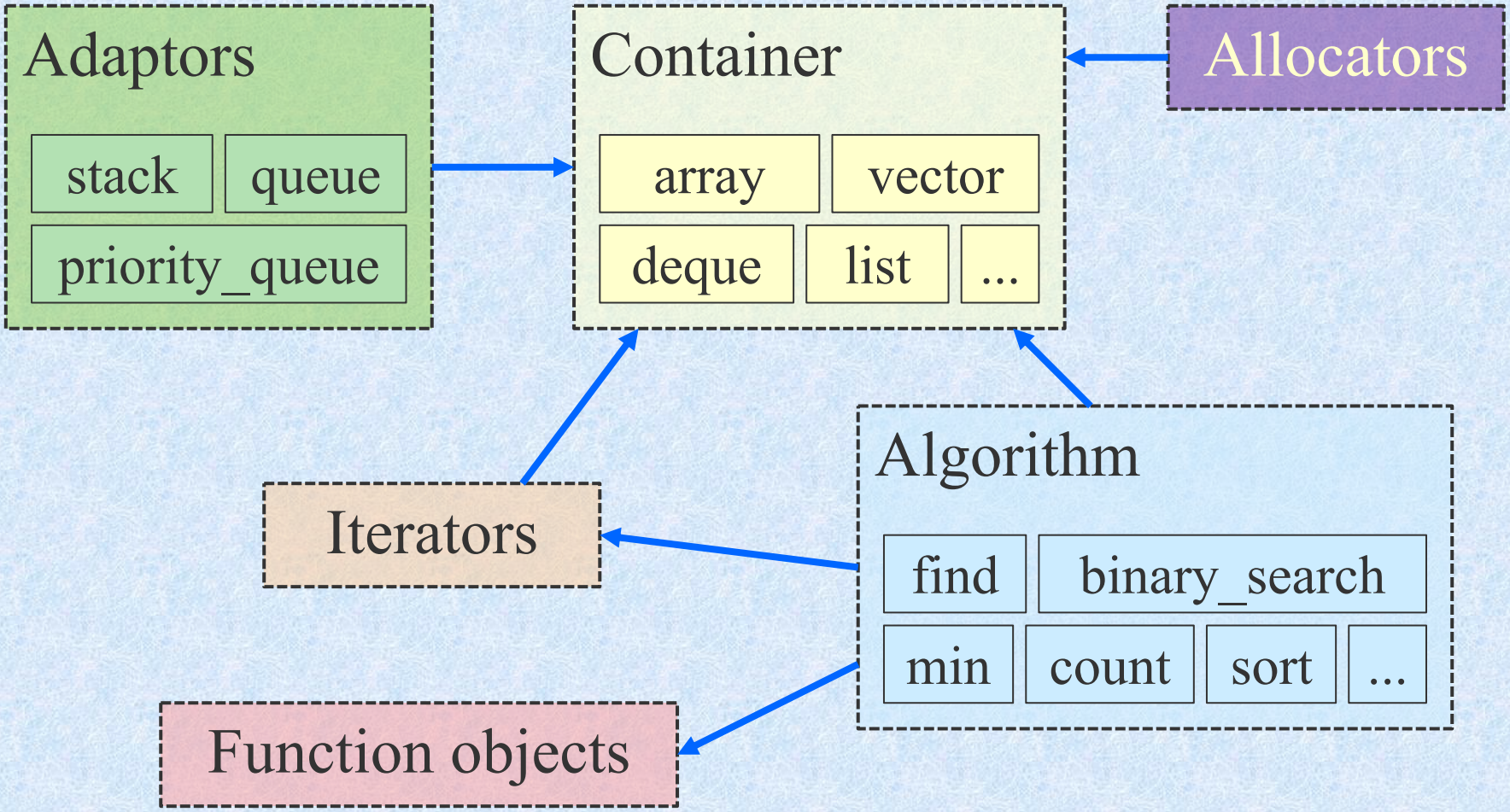
Διαφάνειες παρουσιάσεων

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

Standard Template Library

ΕΙΣΑΓΩΓΗ ΣΤΗΝ STL

Εισαγωγή στην STL



Εισαγωγή στην STL

- Συμβολοσειρά (string)
- Πίνακας (array)
- Διάνυσμα (vector)
- Λίστα (list)
- Στοίβα (stack)
- Ουρά (queue)
- Ζεύγος (pair)
- Πλειάδα (tuple)
- Σύνολο (set)
- Χάρτης (map)

STL – online reference

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)

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
```

Χάρτης

- Τύπος **`map<K, T>`**
  - απεικόνιση κλειδιών τύπου **`K`** σε τιμές τύπου **`T`**
  - μοναδικές τιμές κλειδιών (αλλιώς **`multimap<K, T>`**)
  - υλοποίηση με ισοζυγισμένο δένδρο δυαδικής αναζήτησης
  - εισαγωγή, εύρεση και αφαίρεση σε O(logn)
  - **`insert, erase, find, size`**
  - **`operator[]`**

Χάρτης

```
#include <map>

map<int, string> m;
m.insert(make_pair(1, "one"));
m.insert(make_pair(2, "two"));
m.insert(make_pair(3, "three"));

int x;
cout<< "what? "; cin >> x;

map<int, string>::iterator p;

p = m.find(x);

if (p != m.end())
   cout << p->second << endl;
```

Συνήθως όμως, τα maps δε χρησιμοποιούνται έτσι!

Χάρτης

```
map<int, string> m;
m[1] ="one";
m[2] ="two";
m[3] ="three";

int x;
cin >> x;
if (!m[x].empty())
   cout << m[x] << endl;

for (x=0; x<5; x++)
   cout << x << " -> ("<<m[x] <<")"<<endl;
```

Αλλά έτσι!

```
what? 2
two
0 ->  ()
1 ->  (one)
2 ->  (two)
3 ->  (three)
4 ->  ()
```

Προσοχή!
Αν δεν υπάρχει το m[x] τότε η αναφορά σε αυτό θα το δημιουργήσει με τιμή ένα κενό string

Χάρτης

```
map<string, string> m;
string en[] = {"one", "two", "three", "four"};
string it[] = {"uno", "due", "tre", "quattro"};

for (int i=0; i < 4; i++)
  m[en[i]] = it[i];
```

```
m["one"] = "uno"
m["two"] = "due"
m["three"] = "tre"
m["four"] = "quattro"
```

```
for (string n:en)
  cout << n << " ->  (" << m[n] <<")" <<endl;
```

```
one ->  (uno)
two ->  (due)
three ->  (tre)
four ->  (quattro)
```

Χάρτης

```
int fib(int n) {
  return n < 2 ? n : fib(n-1) + fib(n-2);
}
```

$O(2^n)$

```
int memo_fib(int n) {
  static map<int, int> cache;
  // Αν είναι εύκολο, τελείωνε!
  if (n < 2) return n;
  // Αν υπάρχει ήδη, μην το υπολογίσεις.
  map<int, int>::iterator p = cache.find(n);
  if (p != cache.end()) return p->second;
  // Αλλιώς υπολόγισε και αποθήκευσέ το.
  int v = memo_fib(n-1) + memo_fib(n-2);
  cache[n] = v;  return v;
}
```

Memoization

$O(n \log n)$

Χρήσιμοι αλγόριθμοι στην STL

- Αναζήτηση
- Ταξινόμηση
- Λεξικογραφική διάταξη
- Γεννήτρια μεταθέσεων
- κ.λπ.

```
#include <algorithm>
```

Αναζήτηση (i)

- **`find(first, last, value)`**
  - σε χρόνο O(*n*)

- **`binary_search(first, last, value)`**
- **`binary_search(first, last, value, compare)`**
  - σε χρόνο O(log*n*)

Αναζήτηση (ii)

```
vector<int> v;
...
vector<int>::iterator i =
  find(v.begin(), v.end(), 42);
if (i == v.end())
  cout << "not found" << endl;

// Αν οι αριθμοί είναι ταξινομημένοι:
vector<int>::iterator i =
  binary_search(v.begin(), v.end(), 42);
if (i != v.end())
   *i = 17;              // Άλλαξε το 42 σε 17
```

Ταξινόμηση (i)

- **`sort(first, last)`**
- **`sort(first, last, compare)`**
  - σε χρόνο O(*n*log*n*) στη μέση περίπτωση
- **`stable_sort(first, last)`**
- **`stable_sort(first, last, compare)`**
  - σε χρόνο O(*n*log*n*) στη χειρότερη περίπτωση
  - διατηρεί τη σειρά ίσων στοιχείων

Ταξινόμηση (ii)

```
vector<int> v;
...
// Σε αύξουσα σειρά:
sort(v.begin(), v.end());

// Σε φθίνουσα σειρά:
sort(v.begin(), v.end(), greater<int>());
```

- Το **greater<int>()** κατασκευάζει ένα **function object**. Χρειάζεται:

```
#include <functional>
```

Ταξινόμηση (iii)

```
bool my_less_than(int x, int y) {
  return abs(x) < abs(y);
}

struct my_comparator {
  bool operator<(int x, int y) {
    return abs(x) < abs(y);
  }
};

vector<int> v;
...
// Σε αύξουσα σειρά κατ' απόλυτο τιμή, εναλλακτικά:
sort(v.begin(), v.end(), my_less_than);
sort(v.begin(), v.end(), my_comparator);
```

Ταξινόμηση (iv)

- `partial_sort(first, middle, last)`
- `partial_sort(first, middle, last, compare)`
  - σε χρόνο O(*n*log*m*) στη χειρότερη περίπτωση (*n*=last–first, *m*=middle–first)
  - ταξινομεί μόνο τα *m* πρώτα στοιχεία

```
vector<int> v;
...
partial_sort(v.begin(), v.begin() + 10,
             v.end());
```

Ταξινόμηση (v)

- **`nth_element(first, nth, last)`**
- **`nth_element(first, nth, last, compare)`**
  - σε χρόνο O(*n*) στη μέση περίπτωση
  - το ζητούμενο στοιχείο θα είναι στη θέση του
  - τα προηγούμενά του θα είναι μικρότερα, τα επόμενά του θα είναι μεγαλύτερα

```
vector<int> v;
...
nth_element(v.begin(), v.begin() + 10,
            v.end());
```

Ταξινόμηση (vi)

- `min_element(first, last)`
- `min_element(first, last, compare)`
- `max_element(first, last)`
- `max_element(first, last, compare)`
  - σε χρόνο O(*n*)

ΠΙΝΑΚΕΣ ΩΣ ΑΤΔ

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

Παράδειγμα

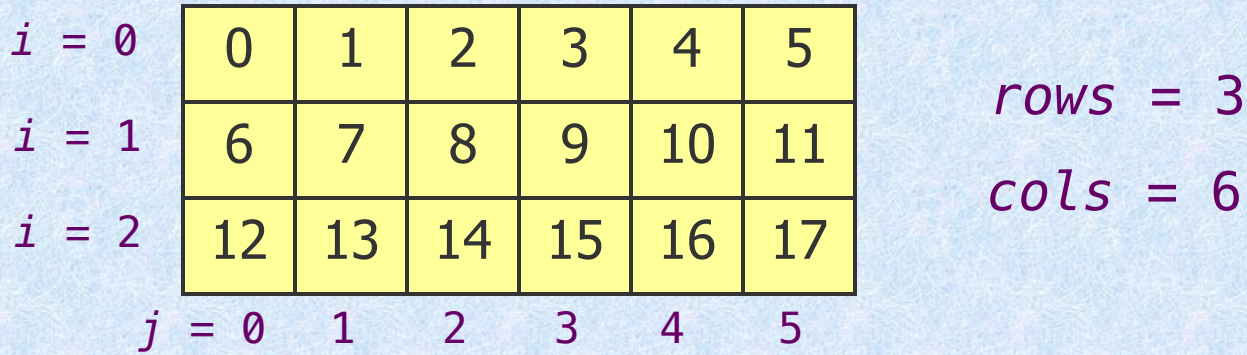
```
template <typename T, size_t N>
class arrayNN {
private:
    T (*data)[N];
    int n;
public:
    arrayNN(): n(N), data(NULL) {};
    arrayNN(T Data[N][N]);
    void print(ostream &out);
    void transpose();
    void deleteRow(int Row);
    void deleteCol(int Col);
    void swapCol(int c1, int c2);
    void swapRow(int r1, int r2);
};
```

```
void transpose() {
  for (int i=0; i<n; ++i)
    for (int j=0; j<i; ++j) {
      T temp = data[i][j];
      data[i][j] = data[j][i];
      data[j][i] = temp;
    }
}
```

Συνήθεις πράξεις σε πίνακες

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

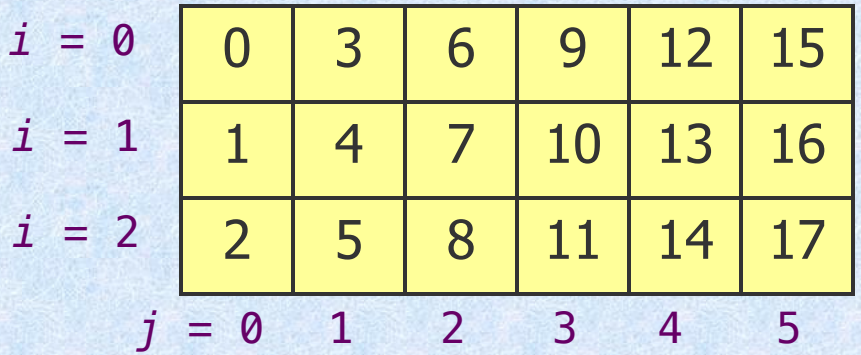
```
for (i=0; i<rows; i++)
  for (j=0; j < cols; j++)
    loc = cols * i + j ;
```

```
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 $n \times m$



rows = 3

cols = 6

- 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

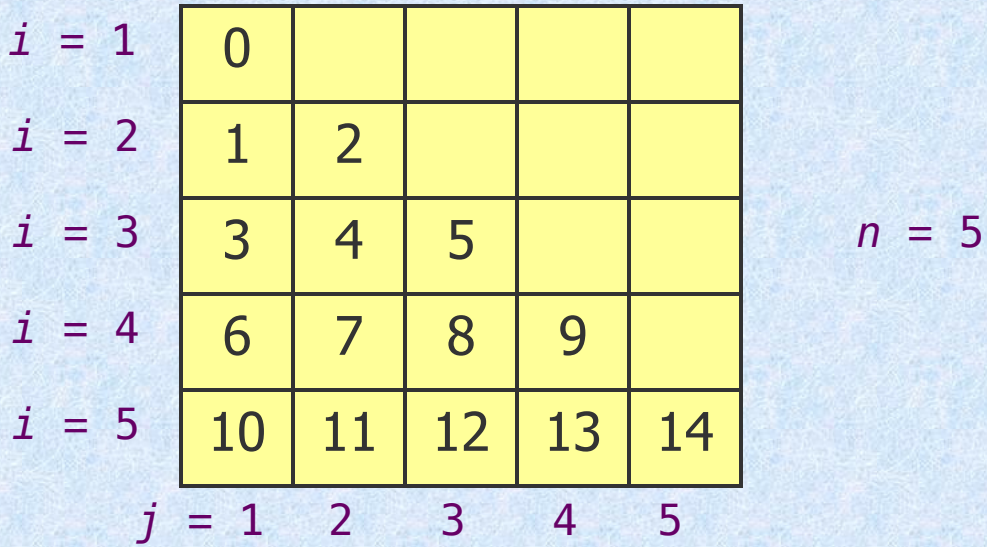
```
for (j=0; j < cols; j++) {
  for (i = 0; i < rows; i++)
    loc = rows * j + i;
```

```
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 $n \times n$

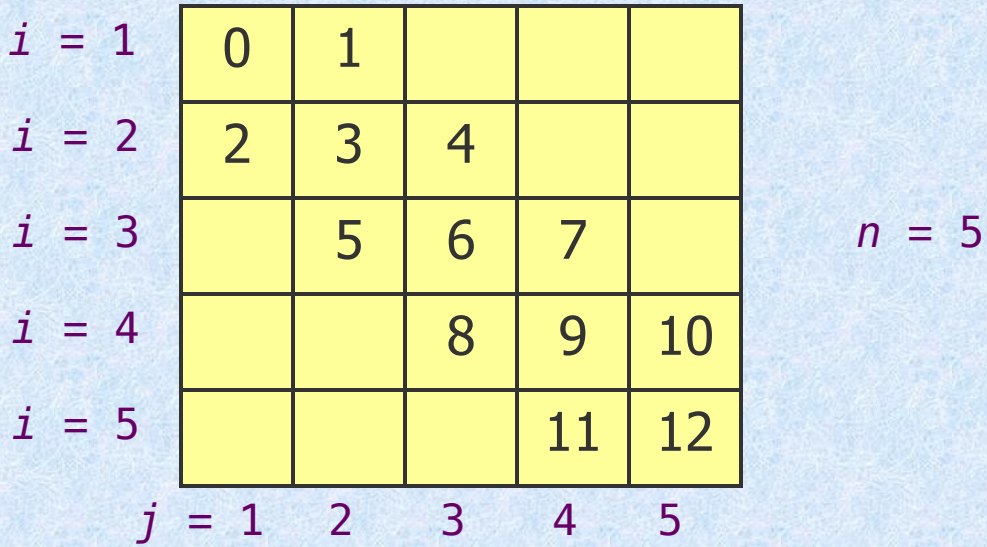


$$loc\,(i,j) = i\,(i-1)/2 + j - 1$$

- Ομοίως για συμμετρικούς πίνακες

Πίνακες ως ΑΤΔ

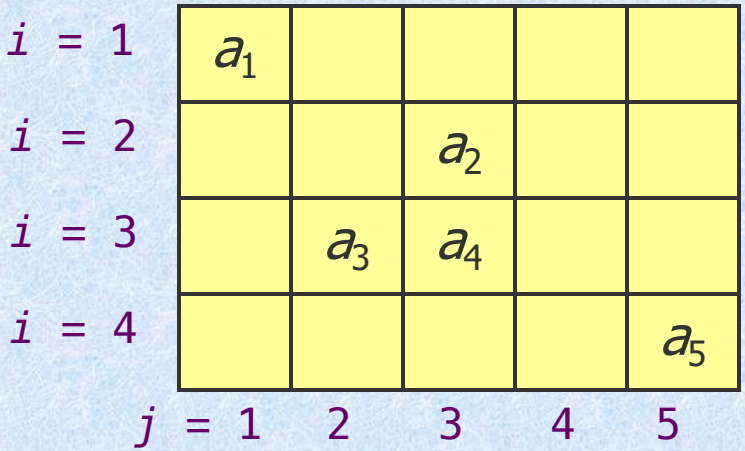
- ΑΤΔ τριδιαγώνιου πίνακα $n \times n$



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

Πίνακες ως ΑΤΔ

- ΑΤΔ αραιού πίνακα $n \times m$



$rows = 4$

$cols = 5$

- Υλοποίηση με δυαδικό πίνακα
- Υλοποίηση με τρεις πίνακες

Κόστος προσπέλασης;

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

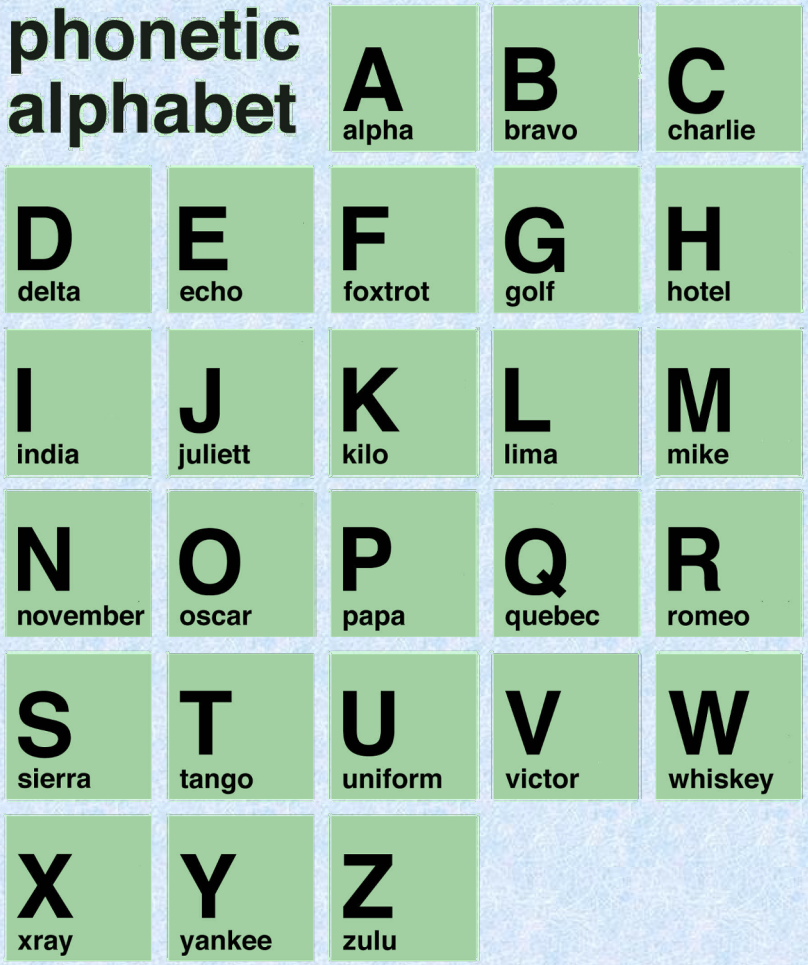
$val = [\ a_1, a_2, a_3, a_4, a_5\ ]$

sierra tango lima echo x-ray alpha mike papa lima echo

ΠΑΡΑΔΕΙΓΜΑ STL

Παράδειγμα STL

- Μετατροπή από/προς το φωνητικό αλφάβητο



Παράδειγμα STL

- Υλοποίηση με map

  Εναλλακτικά: pair

  - string char2phonetic
  - string string2phonetic
  - char phonetic2char
  - string phonetic2word (memoization, too)

```
string char2phonetic(char)
```

'a' → "alpha",
'b' → "beta"

```
string string2phonetic(string)
```

"ece" →
"echo charlie echo"

```
string phonetic2char(string)
```

"alpha" → 'a',
"beta" → 'b'

```
string phonetic2word(string)
```

"echo charlie echo" →
"ece"

Υλοποίηση χωρίς κλάσεις

```
string originalText = "";
string oneWord, originalWord;

while(inFile >> inWord) {
     string p = string2phonetic (inWord);
     cout << p << " " << endl;

     originalWord = phonetic2word(p);
     originalText += originalWord + " / ";
}
```

Παράδειγμα STL

- Σχεδίαση κλάσης **readable**

```
class readable {
private:
   string normalText, string phoneticText;
   map<char, string> phoneticMap;
   map<string, char> invPhoneticMap;
   string phoneticWords[27];
   string char2phonetic(const char &c) ;
   string text2phonetic(const string &txt) ;
   char phonetic2char(const string &searchString);
   string phonetic2text(const string &t) ;
public:
   readable();
   readable(const string & sometext);
   string phonetic();
   string text();
   void setPhonetic(const string & pText);
   void setText(const string & nText);
};
```

static const

// direct

// reverse

// getter

// getter

// setter

// setter

Αφαίρεση και δομές δεδομένων

- Οι λεπτομέρειες υλοποίησης επηρεάζουν την πολυπλοκότητα των αλγορίθμων
- ...
- Η αφαίρεση δεδομένων ελαχιστοποιεί τις αλλαγές που απαιτούνται στο πρόγραμμα που χρησιμοποιεί έναν τύπο δεδομένων, όταν αλλάξει ο τρόπος υλοποίησης

Αφαίρεση

- Χρήση της κλάσης

```
vector<readable> rvector;
while(inFile >> inWord) {
     readable new_r(inWord);
     rvector.push_back(new_r);
}

for (readable item:rvector)
     cout << item.phonetic() << " / ";
cout << endl;

for (readable item:rvector)
     cout << item.text() << " / ";
cout << endl;
```

Παράδειγμα: Διαγωνισμοί προμηθειών

- Αρχές λειτουργίας
  - Ζήτηση προϊόντων με συντήρηση για Ν έτη
  - Προσφορές από προμηθευτές (αγορά, ετήσια συντήρηση)
  - Επιλογή προμηθευτή
    - Για κάθε προϊόν
    - Μικρότερη τιμή: αγορά + συντήρηση * Ν έτη

Παράδειγμα: Διαγωνισμοί προμηθειών

Ζήτηση	Γραφεία	Καρέκλες	Laptop
Προσφορά	1000, 90	500, 17	5000, 0
	850, 95	510, 17	6000, 500
	1000, 78	490, 18	4000, 450
	900, 90	487, 19	1000, 1000
	1200, 57	500, 18	2000, 400

Χρώμα = προμηθευτής

Παράδειγμα: Διαγωνισμοί προμηθειών

Ζήτηση	Γραφεία		Καρέκλες		Laptop	
		N=10		N=10		N=5
Προσφορά	1000, 90	1900	500, 17	670	5000, 0	5000
	850, 95	1800	510, 17	680	6000, 500	8500
	1000, 78	1780	490, 18	670	4000, 450	6250
	900, 90	1800	487, 19	677	1000, 1000	6000
	1200, 57	1770	500, 18	680	2000, 400	4000

Χρώμα = προμηθευτής

Συζήτηση: υλοποίηση με STL

- Κλάσεις
  - Είδη προς προμήθεια (items)
    - description, print, newitem, offers_list
  - Προσφορές (offers)
    - supplier, item, unit price, maintenance price
- Χρήση STL
  - Εισαγωγή προσφορών σε ταξινομημένη λίστα (πχ set) με κλειδί τη συνολική τιμή
  - Αμεση επιλογή από το πρώτο μέλος της λίστας

Do It Yourself

ΑΤΔ - ΠΙΝΑΚΕΣ

ΑΤΔ για μονοδιάστατους πίνακες (i)

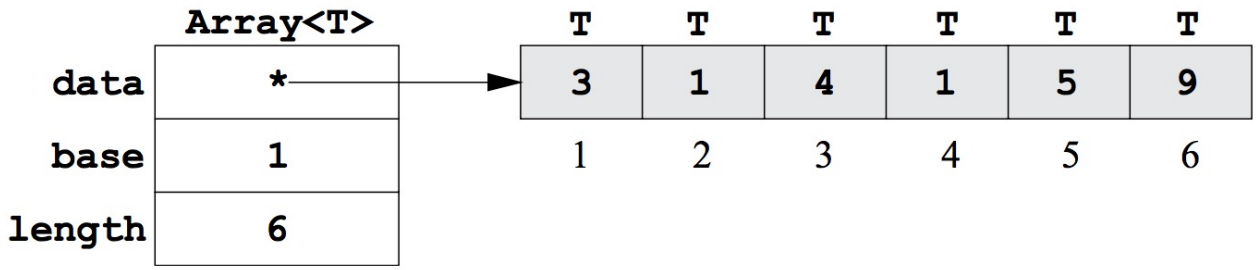
```
template <typename T>
class Array {
public:
  Array(unsigned n = 0, int b = 0);
  Array(const Array &a);
  ~Array();
  Array & operator=(const Array &a);
  T & operator[](int i);
  const T & operator[](int i) const;
};
```

ΑΤΔ για μονοδιάστατους πίνακες (ii)

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

```
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;
};
```

ΑΤΔ για διδιάστατους πίνακες (ii)

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

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

ΑΤΔ για διδιάστατους πίνακες (iii)

```
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 (iv)

```
Array2D & operator=(const Array2D &a) {
  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 (v)

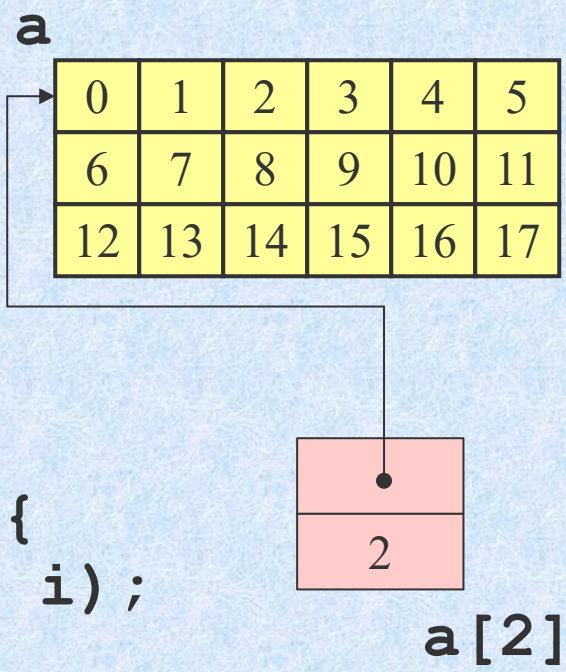
```
unsigned loc(int i, int j) const
          throw(out_of_range) {
  int di = i - baseRow, dj = j - baseCol;
  if (di < 0 || di >= numRows ||
      dj < 0 || dj >= numCols)
    throw out_of_range("invalid index");
  return di * numCols + dj;
}
T & select(int i, int j) {
  return data[loc(i, j)];
}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

ΑΤΔ για διδιάστατους πίνακες (vi)

```
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 (vii)

```
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!