Containers (vectors, maps) – Iterators - algorithms

Containers

Containers

- Η C++ παρέχει ειδικές κλάσεις που ονομάζονται ***containers*** γιατί τα αντικείμενα τους περιέχουν με την σειρά τους πολλά αντικείμενα ενός άλλου τύπου.
- Υπάρχουν δύο μεγάλες κατηγορίες containers:
  - ***Sequential containers*** όπου ο χρήστης μπορεί να καθορίσει την σειρά με την οποία είναι τοποθετημένα τα αντικείμενα που περιέχονται και η οποία δεν έχει καμία σχέση με την τιμή τους.
  - ***Associative containers*** όπου η πρόσβαση σε κάθε στοιχείο γίνεται με την χρήση ενός κλειδιού.

Containers

Sequential	Associative	
Vector	**Map**	Unordered_map
String	Set	Unordered_set
List	Multimap	Unordered_multimap
Forward_list	Multiset	Unordered_multiset
Deque		
Array		

Λειτουργίες σε containers

Υπάρχουν λειτουργίες που είναι

- Κοινές σε όλους τους containers
- Κοινές σε όλους τους sequential / associative containers
- Εξειδικευμένες για κάποιον συγκεκριμένο container

Vectors

Vectors

- Τα Vectors ανήκουν σε μία κατηγορία αντικειμένων που ονομάζονται ***containers*** επειδή περιέχουν άλλα αντικείμενα
- Στην περίπτωση των vectors αυτά τα αντικείμενα πρέπει να είναι του ***ίδιου τύπου***
- Μπορούμε να έχουμε πρόσβαση σε κάποιο από τα αντικείμενα που περιέχει το container με την χρήση ενός ***index***
- Σε αντίθεση με τους πίνακες ένα vector μπορεί να ***μεταβάλει την διάσταση του***

Δήλωση ενός vector

- Η δήλωση ενός vector προϋποθέτει την χρήση της βιβλιοθήκης vector και γίνεται με τον παρακάτω τρόπο

vector<base_type> var_name;

- Παραδείγματα:
- ***vector <int> ivec;***
- ***vector <string> svec;***
- ***vector<vector<int>> array;***
- Στην πραγματικότητα τα vectors είναι ένας μηχανισμός της C++ που ονομάζονται templates

Αρχικοποίηση ενός vector

- Ένας vector μπορεί να δημιουργηθεί αρχικά κενός
- Μπορούν να ανατεθούν αρχικές τιμές
- Μπορεί να πάρει τις τιμές ενός άλλου vector
- Μπορεί να πάρει n αρχικές (ίδιες) τιμές

```
vector<int> ivec1;
vector<int> ivec2 = ivec1;
vector<int> ivec3{ 1,2,3,4,5 };
vector<int> ivec4(4);
vector<int> ivec5(10, 5);
```

Χρήση των στοιχείων ενός vec

- Μπορούμε να έχουμε πρόσβαση, και ανάθεση, σε ένα στοιχείο ενός vector με την χρήση subscripting
- ΠΡΟΣΟΧΗ! ΔΕΝ μπορούμε να επεκτείνουμε ένα vector με την χρήση subscripting
- Μέθοδοι
  - ***size()***
  - ***empty()***

```
vector<int> ivec = { 1,2,3,4,5 };

cout << ivec[0] << endl;
ivec[0] = 99;
cout << ivec.size() << endl;
cout << ivec.empty() << endl;
cout << ivec[5] << endl; // out-of-bounds error
```

Πρόσβαση σε όλα τα στοιχεία ενός vector

- Μπορούμε να έχουμε πρόσβαση σε όλα τα στοιχεία ενός vector με:
  - Range for (πιο σύνηθες). Σαν αναφορά αν θέλουμε να αλλάξουμε την τιμή
  - Index

```
vector<int> ivec = { 1,2,3,4,5 };

for (int i = 0; i < ivec.size(); ++i)
    cout << ivec[i] << endl;

for (auto &x : ivec)
    x = 0;
```

Μέθοδοι αναφοράς σε στοιχεία vector

Εκτός από το index operator μπορούμε να έχουμε πρόσβαση με τις παρακάτω μεθόδους

- ***at()*** – επιστρέφει το στοιχείο με το αντίστοιχο index
- ***front()*** – επιστρέφει το πρώτο στοιχείο
- ***back()*** – επιστρέφει το τελευταίο στοιχείο

```
vector<int> ivec = { 1,2,3,4,5 };
int x;

x = ivec.at(3);

x = ivec.front();

x = ivec.back();
```

Προσθήκη σε vector

- Μπορούμε να προσθέσουμε ένα καινούργιο στοιχείο σε vector με την χρήση της μεθόδου ***push_back***
- Προσθέτει το στοιχείο στο τέλος του vector (και μεγαλώνει το μέγεθος κατά ένα)
- Δεν χρειάζεται αρχική δήλωση του μεγέθους του vector

```
cout << ivec1.size() << endl;
ivec1.push_back(44);
cout << ivec1.size() << endl;
```

Τελεστές που εφαρμόζονται σε vector

- Ανάθεση (=)
  - Μπορούμε να αναθέσουμε τα στοιχεία ενός άλλου vector
  - Μπορούμε να αναθέσουμε με άγκιστρα
- Τελεστές ισότητας , ανισότητας (==, !=)
- Τελεστές σύγκρισης

```
vector<int> ivec = { 1,2,3,4,5 };
vector<int> ivec2;
vector<int> ivec3;

ivec2 = ivec;
ivec2.push_back(44);
ivec3 = { 3,4,5 };

cout << (ivec == ivec2) << endl;
cout << (ivec < ivec2) << endl;
```

Μέθοδοι μεταβολής vector

- ***clear*** – Απομακρύνει όλα τα περιεχόμενα ενός vector
- ***pop_back()*** – διαγράφει το τελευταίο στοιχείο ενός vector
- ***insert()*** – εισάγει στοιχείο σε συγκεκριμένη θέση
- ***erase()*** – διαγράφει στοιχεία σε συγκεκριμένη θέση

```
vector<int> ivec = { 1,2,3,4,5 };
int x;

// diagrafei to 5o
ivec.pop_back();

//eisagei meta to 2o
ivec.insert(ivec.begin()+2, 11);

// diagrafei to 2o
ivec.erase(ivec.begin() + 1);

// diagrafei olo to vector
ivec.clear();
```

iterators

iterators

- Η C++ παρέχει έναν πολύ πιο εξειδικευμένο μηχανισμό για την πρόσβαση στα elements ενός container, εκτός από το indexing, που ονομάζεται iterator
- Ένας iterator στην ουσία «δείχνει» σε ένα element του container
- Πράξεις που μπορούμε να κάνουμε με ένα iterator
  - Να «φέρουμε» το element που «δείχνει»
  - Να κάνουμε να «δείξει» σε επόμενα ή σε προηγούμενα element
- Ένας iterator μπορεί να είναι
  - Valid – Δηλαδή να «δείχνει» σε ένα element του container ή μία θέση μετά από το τελευταίο στοιχείο ενός element
  - Invalid – Όλες οι άλλες θέσεις που μπορεί να δείχνει ένα element

Δημιουργώντας iterators

- Οι τύποι που παρέχουν iterators παρέχουν και μεθόδους που τους δημιουργούν. Μερικοί κλασικοί είναι
- **begin()** – παρέχει iterator που «δείχνει» στο πρώτο element του container
- **end()** – παρέχει iterator που «δείχνει» μία θέση μετά το τελευταίο element του container
- **cbegin()** – παρέχει iterator που «δείχνει» στο πρώτο element του container. Δεν επιτρέπει την αλλαγή της τιμής του element (constant)
- **cend()** – παρέχει iterator που «δείχνει» μία θέση μετά το τελευταίο element του container. Δεν επιτρέπει την αλλαγή της τιμής του element (constant)

Πράξεις σε iterators

Πράξη	Λειτουργία
***iter**	Επιστρέφει το element στο οποίο «δείχνει» ο iterator
iter->mem	Αν ο iterator «δείχνει» σε object, φέρνει το mem member του object αυτού. Ισοδύναμο με το (*iter).mem
++iter / --iter	Κάνει τον iter να δείχνει στο επόμενο / προηγούμενο element
iter+=n / iter-=n	Κάνει τον iter να δείχνει n θέσεις μπροστά / πίσω
iter1 == iter2 / iter1!=iter2	Ελέγχει αν τα iterators «δείχνουν» ή όχι στο ίδιο element
iter1 – iter2	Επιστρέφει τον αριθμό των στοιχείων «ανάμεσα» στα iter1 και iter2

Παράδειγμα με iterators

- Παρατηρήσεις
- Πως βρίσκω αν έχω φτάσει στο τέλος
- Διαφορά μεταξύ cbegin και begin
- Τι επιστρέφει η end;
- Τι επιστρέφει η διαφορά δύο iterators;

```
vector<int> ivec{ 6,7,8,9,10,11 };

for (auto c = ivec.begin(); c != ivec.end(); ++c)
     *c+=10;

cout <<"last element:"<< *(ivec.end() - 1) << endl;

for (auto c = ivec.cbegin(); c != ivec.end(); ++c)
    cout << "elem:" << *c << endl;


cout <<"size:"<< ivec.end() - ivec.begin() << endl;
```

Τύπος iterator / Iterator range

Τύπος iterator

- Κάθε container ορίζει μερικούς καινούργιους τύπους (iterator, const_iterator, size_type, const_reference κ.α.)
- Για παράδειγμα ένας const iterator σε vector<int> έχει τύπο

```
typedef vector<int>::const_iterator vec_iter;
```

Iterators range

- Πολλές φορές χρειάζεται μια ενέργεια να επαναληφθεί σε πολλά στοιχεία, όχι όλα, ενός container
- Τότε είναι χρήσιμη η έννοια του iterator range
- Αποτελείται από δύο iterators όπου το αρχικό συμμετέχει αλλά όχι το τελικό (left inclusive)

Παράδειγμα με iterators

- Παρατηρήστε καινούργιο τρόπο δημιουργίας ενός vector από μερικά στοιχεία ενός άλλου

```
vector<int> v1{ 1,2,3,4,5,6 };
vector<int> v2(v1.cbegin() + 2, v1.cend());

typedef vector<int>::const_iterator vec_iter;

for (vec_iter c = v2.cbegin() ; c != v2.cend(); ++c)
    cout << "elem:" << *c << endl;
```

Generic algorithms

Generic algorithms

- Η C++ ορίζει μια σειρά από γενικούς αλγορίθμους που περιέχονται στις βιβλιοθήκες algorithm και numeric
- Αυτό που κάνουν είναι ότι παίρνουν σαν όρισμα ένα iterator range και επαναλαμβάνουν κάτι σε κάθε στοιχείο του
- Το πλεονέκτημα τους είναι ότι μπορούν να εφαρμοστούν ανεξάρτητα με το τύπο που περιέχουν οι containers
- Υπάρχουν κατηγορίες
  - Που δεν γράφουν στα element
  - Που γράφουν στα elements

Υπάρχουν πολλοί αλγόριθμοι
θα δούμε ένα μικρό υποσύνολο!

Συνάρτηση find / count / accumulate

find(it1,it2,value)

- Παίρνει iterator range και ένα value το οποίο αναζητά σε αυτά. Επιστρέφει το iterator στην πρώτη θέση που θα το βρει. Αν δεν το βρει επιστρέφει τον iterator end()

count(it1,it2,value)

- Παίρνει iterator range και μία τιμή. Επιστρέφει το πόσες φορές βρίσκεται αυτή η τιμή μέσα στο iterator range

accumulate(it1,it2,value)

- Παίρνει iterator range και μία τιμή. Επιστρέφει το άθροισμα των στοιχείων στο range. Κάθε φορά ανάμεσα τους προσθέτει το value (βρίσκεται στο numeric header)

Παράδειγμα

```
vector<int> v1{ 0,1,2,3,4,5,6 };
int val;
cout << "Enter to search for a value:";
cin >> val;

auto result = find(v1.begin(), v1.end(), val);

if (result == v1.end())
    cout << "value not found\n";
else
    cout << "value found at position " << result - v1.begin() << endl;

auto numel = count(v1.begin(), v1.end(),val);
cout << "Value " << val << " is found " << numel << " times\n";

auto sum = accumulate(v1.begin(), v1.end(),0);
cout << "The sum in vector is " << sum << endl;
```

Insert iterators

- Όταν αναθέτουμε τιμή στο element που δείχνει ένας iterator στην ουσία αντικαθιστούμε την τιμή του element που υπήρχε ήδη
- Πολλές φορές χρειαζόμαστε όχι να αντικαταστήσουμε αλλά να προσθέσουμε μια τιμή μέσω ενός iterator
- Για αυτό τον λόγο υπάρχουν οι insert iterators
- Στον vector υπάρχει ο back_inserter. Στην ουσία όταν αναθέτουμε μέσω αυτού καλείται η push_back μέθοδος

Συνάρτηση fill_n

fill_n(dest,n,val)

- Εισάγει n φορές την val τιμή ξεκινώντας από το σημείο που δείχνει ο iterator dest
- Αν δεν υπάρχει χώρος τότε πρέπει να χρησιμοποιήσουμε έναν inserter iterator

```
vector<int> ivec{ 1,2,3 };

auto it = ivec.begin();

// number >3 crashes code
fill_n(it, 3, 10);
cout << "first----\n";
for (auto x : ivec)
    cout << "elem:" << x << endl;

auto bit = back_inserter(ivec);

fill_n(bit, 10, 0);
cout << "second----\n";
for (auto x : ivec)
    cout << "elem:" << x << endl;
```

Συνάρτηση copy

copy(iter1,iter2,dest)

- Παίρνει τα element που περιέχονται στο iterator range [iter1,iter2) και τα αντιγράφει στο dest
- Και εδώ αν δεν υπάρχει αρκετός χώρος στο dest πρέπει να χρησιμοποιήσουμε inserter iterator

```
vector<int> ivec{ 0,1,2,3,4,5,6,7,8,9 };
vector<int> ivec2;

auto bit = back_inserter(ivec2);

copy(ivec.begin()+3, ivec.end(), bit);

for (auto x : ivec2)
    cout << "Elem:" << x << endl;
```

Συνάρτηση sort, unique

sort(iter1,iter2)

- Παίρνει σαν όρισμα ένα iterator range και επιστρέφει ταξινομημένες τις τιμές των elements με αύξουσα σειρά
- Προσοχή ΠΡΕΠΕΙ να υποστηρίζεται ο operator < από τα elements του vector

unique(iter1,iter2)

- Παίρνει iterator range ταξινομημένο βάζει τα μοναδικά μπροστά, επιστρέφει στο πρώτο μη μοναδικό

```
vector<int> ivec{ 8, 3, 7, 3 , 9, 2, 1, 2 , 1};

sort(ivec.begin(), ivec.end());

cout << "All----\n";
for (auto x : ivec)
    cout << "Elem:" << x << endl;

auto enduinq = unique(ivec.begin(), ivec.end());

cout << "Unique----\n";
for (auto x=ivec.begin();x!=enduinq;++x)
    cout << "Elem:" << *x << endl;
```

maps

maps

- Associative containers είναι αυτά στα οποία ο χρήστης δεν καθορίζει την θέση που θα τοποθετηθούν τα elements
- Στην κατηγορία αυτά ανήκουν τα ***map*** (header map) και ***set*** (header set)
- Το map περιέχει μια σειρά από ***key, value*** ζευγάρια (***pairs***).
- Ονομάζεται σαν *associative array* και στην ουσία χρησιμοποιεί το key σαν index για να μπορέσει να έχει πρόσβαση στην τιμή του value
- Στο map κάθε key πρέπει να έχει μία μοναδική τιμή (υπάρχει και το multimap)

Δημιουργία map

Μπορεί να δημιουργηθεί:

- Κενό
- Σαν αντίγραφο ενός υπάρχοντος
- Από λίστα τιμών άλλου container (φτάνει να είναι του ίδιου τύπου)
- List initialize

```
map<int, double> map1;

map<int, double> map2 =
    { {8,2.33},{4,4.33},
    {3,6.77},{1,5.67} };

map<int, double> map3{ map2 };

map<int, double> map4(map3.begin()++, map3.end());
```

- Προσοχή οι iterators στους maps έχουν μόνο ++ και --

Στοιχεία ενός map

- Τα στοιχεία που περιέχει ένα map είναι αντικείμενα της κλάσης ***pair*** (utility header).
- Περιέχει δύο πεδία τα
  - ***first*** που περιέχει το key
  - ***second*** που περιέχει το value

```
map<int, double> map2 =
{ {8,2.33},{4,4.33},
{3,6.77},{1,5.67} };

for (auto x : map2)
    cout << "Key:" << x.first << "->Value:"
    << x.second << endl;
```

- Μπορούμε να δούμε αν υπάρχει κάποιο κλειδί με την χρήση της μεθόδου find

Εισαγωγή στοιχείων σε map

- Αν χρησιμοποιήσουμε για το indexing key το οποίο δεν υπάρχει τότε δημιουργεί ένα καινούργιο element στο map
- Λύση; μπορούμε να χρησιμοποιήσουμε την ***insert***. Αυτή επιστρέφει pair όπου το πεδίο second είναι ένα boolean που λέει αν έγινε η εισαγωγή

```
map<string, int> data;

data["ena"] = 1;
data["neo"] = 3;
data["dyo"] = 3;
data["ena"] = 5;

for (auto x : data)
    cout << x.first << "->"
    << x.second << endl;

auto ret = data.insert({ "ena",10 });
cout << "Inserted? " << ret.second << endl;

ret = data.insert({ "tria",10 });
cout << "Inserted? " << ret.second << endl;

cout << "Last-----\n";
for (auto x : data)
    cout << x.first << "->"
    << x.second << endl;
```

Διαγραφή στοιχείων

- Για την διαγραφή elements από ένα map μπορεί να χρησιμοποιηθεί η συνάρτηση ***erase***
- Παίρνει σαν όρισμα το key και διαγράφει το element που έχει αυτό το key
- Αν δεν υπάρχει το key δεν δημιουργεί πρόβλημα

```
map<int, double> dynam{ {2,33.44},
    {4,5.67}, {8,3.33}, {9,10.34} };

cout << "start--\n";
for (auto x : dynam)
    cout << x.first << "->" << x.second << endl;

dynam.erase(8);
dynam.erase(1);

cout << "after erase--\n";
for (auto x : dynam)
    cout << x.first << "->" << x.second << endl;
```