Εισαγωγή στον Προγραμματισμό

Μέρος 8ο: Η καθιερωμένη βιβλιοθήκη προτύπων STL

Εξάμηνο Σπουδών: 3ο
Κωδικός Μαθήματος: 343

Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Μιχάλης Α. Μπέκος
bekos@uoi.gr

Μέρος 1°:
Εισαγωγή

Η καθιερωμένη βιβλιοθήκη προτύπων (Standard Template Library)

- Είναι μέρος της τυπικής (standard) βιβλιοθήκης της C++.
- Περιέχει πρότυπα κλάσεων και συναρτήσεων τα οποία μπορούν να χρησιμοποιηθούν για την αποθήκευση, αναζήτηση και εκτέλεση αλγορίθμων σε δομές δεδομένων.
- Καθιστά τη (συνήθως πολύπλοκη και χρονοβόρα) διαδικασία υλοποίησης αλγορίθμων και δομών δεδομένων απαραίτητη, μόνο σε περιπτώσεις που αυτές δεν υποστηρίζονται από τη βιβλιοθήκη.
- Αποτελείται από:
  - Κλάσεις αποδέκτες [Container classes] — περιέχουν ομοειδή αντικείμενα
  - Διαπροσπελαστές [Iterators] — δείκτες σε αντικείμενα των αποδεκτών
  - Αλγορίθμους [Algorithms] — επεξεργαγίας των αντικειμένων των αποδεκτών

Αποδέκτες (Containers)

- **Ακολουθιακοί αποδέκτες** [sequence containers] - σειριακές συλλογές δεδομένων όπου τα αντικείμενα διατηρούν την αρχική σειρά καταχώρησης τους.
  - `array, vector, deque`

    υλοποιούνται ως δυναμικοί πίκανες ⇒ υποστηρίζουν τυχαία (και σειριακή) προσπέλαση

  - `list`

    υλοποιούνται με δείκτες ⇒ υποστηρίζουν μόνο σειριακή προσπέλαση

- **Συνειρμικοί αποδέκτες** [associative containers] - διατεταγμένες συλλογές αντικειμένων ως προς κάποιο “κλειδί”, το οποίο χρησιμοποιείται για την πρόσβαση σε αυτά (αντί της θέσης).
  - `set, multiset, map, multimap`

    δενδρικές δομές

- **Προσαρμογείς** [container adapters] - τροποποιούν τις δυνατότητες άλλων αποδεκτών.
  - `stack, queue, priority_queue`

    διαφορετικές εκδόσεις της κλάσσης `deque` των ακολουθιακών αποδεκτών

Μέρος 2°:
Ακολουθιακοί αποδέκτες

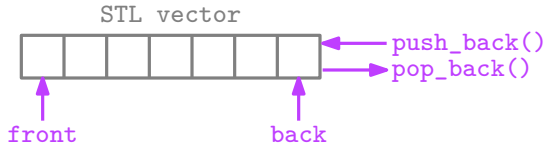
Ακολουθιακοί αποδέκτες: `std::array`

- Πίνακας σταθερού μεγέθους.
  - `#include <array>`
  - `array<string, 10> arr;`
- Γρήγορη τυχαία προσπέλαση (με δείκτη θέσης).
  - `operator [], at, front, back`
- Άλλες διεργασίες:
  - `size, empty, clear, fill`
- Η συμπεριφορά του τελεστή `[]` είναι απροσδιόριστη, εάν ο δείκτης θέσης δεν είναι εντός του εύρους του αποδέκτη.

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

template <typename T, unsigned int N>
void print(array<T,N> A);

int main() {
  array<int,10> digits;
  for (int i=0; i<digits.size(); i++)
    digits[i] = i;
  print(digits);
  cout<<"Length:"<< digits.size()<<endl;
  cout<<"First:" <<digits.front()<<endl;
  cout<<"Last:"  << digits.back()<<endl;
  return 0;
}

template <typename T, unsigned int N>
void print(array<T,N> A) {
  for (int i=0; i<N; i++)
    cout << A.at(i) << " ";
  cout << endl;
}
```

Part08/std_array.cpp

Ακολουθιακοί αποδέκτες: `std::vector`

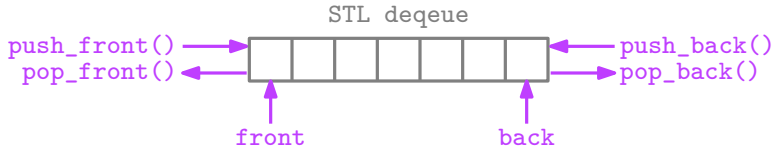


- Πίνακας μεταβλητού μεγέθους.
  - `#include <vector>`
  - `vector<string> vec;`
- Γρήγορη τυχαία προσπέλαση (με δείκτη θέσης).
  - `operator [], at, front, back`
- Αργή εισαγωγή και διαγραφή στη μέση.
  - `insert, erase`
- Γρήγορη εισαγωγή και διαγραφή στο τέλος.
  - `push_back, pop_back`
- Άλλες διεργασίες:
  - `size, empty, clear, fill`

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> prms = {1, 2, 3, 5, 7};

  cout<<"1st prime: "<<prms.front()<<endl;
  cout<<"3rd prime: "<< prms.at(2) <<endl;

  // Add more prime numbers to the vector.
  prms.push_back(11);
  prms.push_back(13);
  prms.push_back(17);
  // Remove the last added prime.
  prms.pop_back();

  // Like an array, we use [] for access.
  for (int i=0; i<prms.size(); i++)
    cout << prms[i] << " ";
  cout << endl;

  return 0;
}
```

Part08/std_vector.cpp

Ακολουθιακοί αποδέκτες: `std::deque`

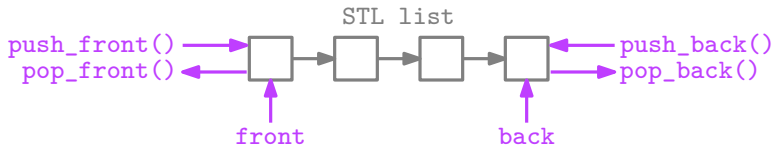


- Όπως η `std::vector`, αλλά με γρήγορη εισαγωγή και στα δύο άκρα.
  - `#include <deque>`
  - `deque<string> dq;`
- Γρήγορη τυχαία προσπέλαση (με δείκτη θέσης).
  - `operator [], at, front, back`
- Αργή εισαγωγή και διαγραφή στη μέση.
  - `insert, erase`
- Γρήγορη εισαγωγή και διαγραφή στα δύο άκρα.
  - `push_front, push_back, pop_front, pop_back`
- Άλλες διεργασίες:
  - `size, empty, clear, fill`

```cpp
#include <iostream>
#include <deque>
using namespace std;

int main() {
  int sum = 0;
  deque<int> dq;
  dq.push_front(10);
  dq.push_back(20);
  dq.push_front(30);
  dq.push_back(40);
  dq.push_front(50);

  // while dq is not empty, pop front
  while (!dq.empty()) {
    sum += dq.front();
    cout << "Pop " << dq.front() << endl;
    dq.pop_front();
  }

  cout << "Sum: " << sum << endl;

  return 0;
}
```

Part08/std_deque.cpp

Ακολουθιακοί αποδέκτες: `std::list`



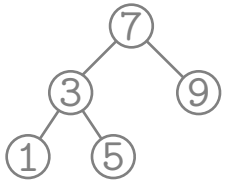
- Συνδεδεμένη λίστα
  - `#include <list>`
  - `list<string> lst;`
- Αργή τυχαία προσπέλαση.
  - δεν υποστηρίζεται ο τελεστής []
  - προσπέλαση μέσω διαπροσπελαστών (σε λίγο)
- Γρήγορη εισαγωγή και διαγραφή σε κάθε θέση.
  - `insert, erase`
  - `push_front, push_back, pop_front, pop_back`
- Άλλες διεργασίες:
  - `size, empty, clear`
  - `front, back`
  - `reverse, sort, unique, merge`

```cpp
#include <iostream>
#include <list>
using namespace std;

int main() {
  list<int> L1, L2;

  for (int i = 0; i <= 10; ++i) {
    L1.push_back(i * 2);
    L2.push_front(30 - i * 3);
  }

  L1.pop_front();
  L2.push_back(100);

  L1.reverse();
  L2.sort();
  L1.merge(L2);

  cout<<"L1.front(): "<<L1.front()<<endl;
  cout<<"L1.back(): " <<L1.back() <<endl;

  return 0;
}
```

Part08/std_list.cpp

Μέρος 3°:
Συνειρμικοί αποδέκτες

Συνειρμικοί αποδέκτες: `std::set`

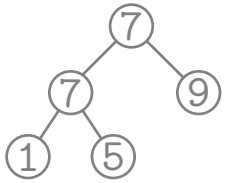


- Αποθηκεύουν σύνολα συγκρίσιμων τιμών (κλειδιά)
  - `bool operator<(T one, T other);`
- Οι τιμές είναι μοναδικές (αποθηκεύονται μια φορά)
- Η υλοποίηση βασίζεται σε δενδρικές δομές
  - `#include <set>`
  - `set<string> s;`
- Γρήγορη εισαγωγή και διαγραφή
  - `insert, erase`
- Γρήγορη αναζήτηση (ελαχίστου και μεγίστου)
  - `find`
- Άλλες διεργασίες:
  - `size, empty, clear`

```cpp
#include <iostream>
#include <set>
using namespace std;

int main() {
  set<int> s;

  for (int i=10; i>=0; i--) {
    s.insert(10*i);
  }
  s.insert(-10); // <-- this works
  s.insert(10);  // <-- this does nothing

  cout << "Min " << *(s.begin()) << endl;
  cout << "Max " << *(s.rbegin())<< endl;

  int x = 3;
  if (s.find(x) != s.end())
    cout << x << " is found";
  else
    cout << x << " is not found";

  return 0;
}
```

Part08/std_set.cpp

Συνειρμικοί αποδέκτες: `std::multiset`

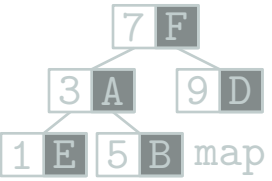


- Όπως η `std::set`, αλλά οι τιμές που αποθηκεύονται δεν είναι απαραίτητα μοναδικές.
- Η υλοποίηση βασίζεται σε δενδρικές δομές
  - `#include <set>`
  - `multiset<string> s;`
- Γρήγορη εισαγωγή και διαγραφή
  - `insert, erase`
- Γρήγορη αναζήτηση (ελαχίστου και μεγίστου)
  - `find`
- Άλλες διεργασίες:
  - `size, empty, clear`

```cpp
#include <iostream>
#include <set>
using namespace std;

int main() {
  multiset<int> ms;

  for (int i=10; i>=0; i--) {
    ms.insert(10*i);
  }
  ms.insert(-10); // <-- this works
  ms.insert(10);  // <-- this also works

  int x = 10;
  while (ms.find(x) != ms.end()) {
    ms.erase( ms.find(x) );
    cout << x << " was removed" << endl;
  }

  // Equivalently one could write:
  // ms.erase(x);

  return 0;
}
```

Part08/std_multiset.cpp

Συνειρμικοί αποδέκτες: `std::map`

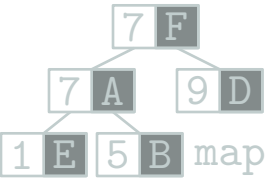


- Αποθηκεύουν ζεύγοι ⟨ κλειδί, τιμή ⟩
- Κάθε κλειδί έχει μια τιμή
- Η υλοποίηση βασίζεται σε δενδρικές δομές
  - `#include <map>`
  - `map<string, int> mp;`
- Γρήγορη εισαγωγή και διαγραφή
  - `mp["Kate"] = 7777`
  - `insert, erase`
- Γρήγορη αναζήτηση
  - `int x = mp["Kate"];`
  - `find`
- Άλλες διεργασίες:
  - `size, empty, clear`

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

int main() {
  map<string,int> phones;
  phones["Jack"] = 12345;
  phones["Mike"] = 12346;
  phones["Eric"] = 12347;
  phones["Kate"] = 12348;

  if (phones.find("Mike")!=phones.end())
    phones.erase(phones.find("Mike"));

  // print content:
  cout<<"Elements in phonebook:"<<endl;
  cout<<"Jack => "<<phones["Jack"]<<endl;
  cout<<"Mike => "<<phones["Mike"]<<endl;
  cout<<"Eric => "<<phones["Eric"]<<endl;
  cout<<"Kate => "<<phones["Kate"]<<endl;

  return 0;
}
```

Part08/std_map.cpp

Συνειρμικοί αποδέκτες: `std::multimap`



- Όπως η `std::map`, αλλά κάθε κλειδί μπορεί να αντιστοιχεί σε περισσότερες από μια τιμές.
- Η υλοποίηση βασίζεται σε δενδρικές δομές
  - `#include <map>`
  - `multimap<string, int> mm;`
- Γρήγορη εισαγωγή και διαγραφή
  - `insert, erase`
- Γρήγορη αναζήτηση
  - `find`
- Άλλες διεργασίες:
  - `size, empty, clear`

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

int main() {
  multimap<string,string> mmp;

  mmp.insert({"India", "New Delhi"});
  mmp.insert({"India", "Hyderabad"});
  mmp.insert({"Greece", "Athens"});
  mmp.insert({"Greece", "Patras"});
  mmp.insert({"UK", "Birmingham"});
  mmp.insert({"UK", "Manchester"});

  cout << "Stored cities in Greece: "
       << mmp.count("Greece") << endl;

  cout << "Stored cities in UK: "
       << mmp.count("UK") << endl;

  //Remove all cities in India from mmp
  mmp.erase("India");
}
```

Part08/std_multimap.cpp

Μέρος 4°:
Προσαρμογείς

Προσαρμογείς: `std::stack`

- Παρέχει διεπαφή στοίβας σε αποδέκτες
  - `#include <stack>`
  - `stack<string> st;`
- Λειτουργίες στοίβας:
  - `push, pop, top`
  - `size, empty, clear`
- Μπορεί να εφαρμοστεί σε `vector`, `deque` ή `list`
  - `stack<string> st; //deque by default`
  - `stack<string, vector<string> > st;`
  - `stack<string, list<string> > st;`
- Απαιτείται επιπλέον κενό για την αποφυγή >>

```cpp
#include <iostream>
#include <string>
#include <stack>
using namespace std;

// utility function to print a stack
void print_stack(stack<string>);

int main() {
  // create a stack of strings
  stack<string> languages;
  languages.push("C++");
  languages.push("Java");
  languages.push("Python");
  // print its elements and size
  print_stack(languages);
  cout << "(" << languages.size() << ")";
}
void print_stack(stack<string> st) {
  while(!st.empty()) {
    cout << st.top() << " ";
    st.pop();
  }
}
```

Part08/std_stack.cpp

Προσαρμογείς: `std::queue`

- Παρέχει διεπαφή ουράς σε αποδέκτες
  - `#include <queue>`
  - `queue<string> q;`
- Λειτουργίες ουράς:
  - `push, pop, front`
  - `size, empty, clear`
- Μπορεί να εφαρμοστεί σε `deque` ή `list`
  - `queue<string> q; //deque by default`
  - ~~`queue<string, vector<string> > q;`~~
  - `queue<string, list<string> > q;`
- Απαιτείται επιπλέον κενό για την αποφυγή >>

```cpp
#include <iostream>
#include <string>
#include <queue>
using namespace std;

// template function to print a queue
template <typename T>
void print_queue(queue<T> q) {
  while(!q.empty()) {
    cout << q.front() << " ";
    q.pop();
  }
}

int main() {
  // create a queue of strings
  queue<string> languages;
  languages.push("C++");
  languages.push("Java");
  languages.push("Python");
  // print its elements and size
  print_queue(languages);
  cout << "(" << languages.size() << ")";
}
```

Part08/std_queue.cpp

Προσαρμογείς: `std::priority_queue`

- Διεπαφή ουράς προτεραιότητας σε αποδέκτες
  - `#include <queue>`
  - `priority_queue<int> pq;`
- Λειτουργίες ουράς:
  - `push, pop, top`
  - `size, empty, clear`
- Μπορεί να εφαρμοστεί σε `deque` ή `vector`
  - `priority_queue<int> pq; //deque by default`
  - `priority_queue<int, vector<int> > pq;`
  - ~~`priority_queue<int, list<int> > pq;`~~
- Απαιτείται επιπλέον κενό για την αποφυγή >>

```cpp
#include <iostream>
#include <queue>
using namespace std;

// template function to print a PQ
template <typename T>
void print_pq(priority_queue<T> pq) {
  while(!pq.empty()) {
    cout << pq.top() << " ";
    pq.pop();
  }
}

int main() {
  // create a PQ of grades
  priority_queue<int> grades;
  grades.push(7);
  grades.push(10);
  grades.push(5);

  // print grades in descending order
  print_pq(grades);
  cout << "(" << grades.size() << ")";
}
```

Part08/std_queue.cpp

Μέρος 5°:
Διαπροσπελαστές

Διαπροσπελαστές: Εισαγωγή

- Πώς μπορούμε να προσπελάσουμε τα στοιχεία που είναι αποθηκευμένα σε ένα αποδέκτη;

- Χρησιμοποιώντας τον τελεστή `[]`, αν ο αποδέκτης είναι:
  - `std::array`
  - `std::vector`
  - `std::deque`

- Αυτό το στυλ επανάληψης δεν λειτουργεί για όλους τους τύπους αποδεκτών

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <queue>
using namespace std;

int main() {
  // create a container with names
  vector<string> names;
  //deque<string> names;

  // add certain names to it
  names.push_back("fred");
  names.push_back("wilma");
  names.push_back("barney");
  names.push_back("betty");
  names.push_back("dino");

  // print all names using []
  for (int x=0; x < names.size(); ++x)
  {
    cout << names[x] << endl;
  }
}
```

Part08/std_iterators.cpp

Διαπροσπελαστές

- Αντικείμενα που μοιάζουν με δείκτες και χρησιμοποιούνται για την προσπέλαση των στοιχείων ενός αποδέκτη.
- Όλοι οι αποδέκτες υποστηρίζουν τις μεθόδους:
  - `begin()` επιστρέφει ένα διαπροσπελαστή ο οποίος “δείχνει” στο πρώτο στοιχείο τους.
  - `end()` επιστρέφει μια ειδική τιμή που αντιστοιχεί στο τέλος του αποδέκτη.
- Οι διαπροσπελαστές υπερφορτώνουν τους ακόλουθους τελεστές:
  - `++`, `--` μετακίνηση στο επόμενο/προηγούμενο στοιχείο
  - `*`, `->` πρόσβαση στην τιμή της τρέχουσας θέσης
  - `==`, `!=` σύγκριση διαπροσπελαστών για ισότητα

```cpp
#include <iostream>
#include <string>
#include <set>
using namespace std;

int main() {
  // create a container with names
  set<string> names;

  // add certain names to it
  names.insert("fred");
  names.insert("wilma");
  names.insert("barney");
  names.insert("betty");
  names.insert("dino");

  // print all names using iterator
  set<string>::iterator it;
  it = names.begin();
  while (it != names.end()) {
    cout << *it << endl;
    it++;
  }
}
```

Part08/std_iterators.cpp

Διαπροσπελαστές

- Με ποια σειρά γίνεται η προσπέλαση των στοιχείων του αποδέκτη;
  - Ακολουθιακοί αποδέκτες: Σύμφωνα με την αρχική σειρά καταχώρησης τους.
  - Συνειρμικοί αποδέκτες: Σύμφωνα με την σειρά ταξινόμησης τους.
  - Προσαρμογείς: Δεν υποστηρίζουν διαπροσπελαστές (για προφανείς λόγους).

- Η προσπέλαση των στοιχείων ενός αποδέκτη μπορεί να γίνει μέσω μια εντολής ανακύκλωσης `while` ή `for`

```
set<string> names;
...
set<string>::iterator it;
it = names.begin();
while (it != names.end()) {
  cout << *it << endl;
  it++;
}
```

```
vector<int> grades;
...
vector<int>::iterator it;
// iterate with a for loop to change element's values
// observe the usage of operator ++
for (it = grades.begin(); it != grades.end(); ++it) {
  *it = *it + 1;
}
```

Διαπροσπελαστές

- Με ποια σειρά γίνεται η προσπέλαση των στοιχείων του αποδέκτη;
  - Ακολουθιακοί αποδέκτες: Σύμφωνα με την αρχική σειρά καταχώρησης τους.
  - Συνειρμικοί αποδέκτες: Σύμφωνα με την σειρά ταξινόμησης τους.
  - Προσαρμογείς: Δεν υποστηρίζουν διαπροσπελαστές (για προφανείς λόγους).

- Η προσπέλαση των στοιχείων ενός αποδέκτη μπορεί να γίνει και με την αντίστροφη σειρά χρησιμοποιώντας αντίστροφους διαπροσπελαστές μέσω των μεθόδων `rbegin` και `rend`

```
set<string> names;
...
set<string>::reverse_iterator it;
it = names.rbegin();
while (it != names.rend()) {
  cout << *it << endl;
  it++;
}
```

```
vector<int> grades;
...
vector<int>::reverse_iterator it;
// iterate with a for loop to change element's values
// observe the usage of operator ++
for (it = grades.rbegin(); it != grades.rend(); ++it) {
  *it = *it + 1;
}
```

Διαπροσπελαστές

- Με ποια σειρά γίνεται η προσπέλαση των στοιχείων του αποδέκτη;
  - Ακολουθιακοί αποδέκτες: Σύμφωνα με την αρχική σειρά καταχώρησης τους.
  - Συνειρμικοί αποδέκτες: Σύμφωνα με την σειρά ταξινόμησης τους.
  - Προσαρμογείς: Δεν υποστηρίζουν διαπροσπελαστές (για προφανείς λόγους).
- Η προσπέλαση **των ζευγών ενός λεξικού** γίνεται επίσης μέσω μια εντολής ανακύκλωσης `while` ή `for` στην οποία ο δείκτης-διαπροσπελαστής υποστηρίζει δύο πεδία `first` και `second` για την πρόσβαση στο κλείδί και στην τιμή, αντίστοιχα.

```
map<string,string> mp;
...
map<string,string>::iterator it;
it =mp.begin();
while (it != mp.end()) {
  cout << "key: " << it->first << ", value: " << it->second << endl;
  it++;
}
```

Μέρος 6°:
Αλγόριθμοι

Αλγόριθμοι

- Η καθιερωμένη βιβλιοθήκη προτύπων STL παρέχει πολλές συναρτήσεις που μπορούν να λειτουργήσουν σε αποδέκτες.

- Οι συναρτήσεις αυτές ονομάζονται αλγόριθμοι.
  - `#include <algorithm>`

- Ορισμένοι αλγόριθμοι ωστόσο λειτουργούν μόνο σε συγκεκριμένους αποδέκτες.

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
  // create a container with names
  vector<string> names;
  names.insert("fred");
  names.insert("wilma");
  names.insert("barney");
  names.insert("fred");

  // apply two algorithms
  unique(names.begin(), names.end());
  sort(names.begin(), names.end());

  vector<string>::iterator it;
  it = names.begin();
  while (it != names.end()) {
    cout << *it << endl; it++;
  }
}
```

Part08/std_algorithms.cpp

Επιπλέον υλικό

- Κεφάλαιο 15:
  R. Lafore, Αντικειμενοστρεφής προγραμματισμός με τη C++,
  ISBN: 960-209-904-6, εκδόσεις Κλειδάριθμος, 2006.