

Τεχνολογίες Υλοποίησης Αλγορίθμων

Εισαγωγή στον Γενικευμένο Προγραμματισμό και τη Standard Template Library (STL)

Χρήστος Ζαρολιάγκης
Καθηγητής

Γρηγόρης Πράσιнос
Υποψήφιος Διδάκτωρ

Τμήμα Μηχανικών Η/Υ και Πληροφορικής
Πανεπιστήμιο Πατρών

Γενικευμένος Προγραμματισμός (Generic Programming)

Ορισμοί

My working definition of generic programming is “programming with concepts”, where a concept is defined as a family of abstractions that are all related by a common set of requirements. (Dave Musser)

Generic programming is a subdiscipline of computer science that deals with finding abstract representations of efficient algorithms, data structures, and other software concepts, and with their systematic organization. (Czarnecki, Eisenecker)

Γενικευμένος προγραμματισμός

Ο σκοπός του γενικευμένου προγραμματισμού είναι η ανάπτυξη λογισμικού που να επιτρέπει την επαναχρησιμοποίηση (reuse) με απλό και αποδοτικό τρόπο.

Η πράξη έχει δείξει ότι ο αντικειμενοστρεφής προγραμματισμός έχει σαν αποτέλεσμα μη ευέλικτα σύνολα αντικειμένων με μικρή δυνατότητα επαναχρησιμοποίησης.

Παράδειγμα μη γενικευμένου αλγορίθμου

Αντιγραφή στοιχείων

```
void* memcpy(void* region1, const void* region2, size_t n)
{
    const char* first = region2;
    const char* last = region2 + n;
    char* result = region1;
    while (first != last)
        *result++ = *first++;
    return result;
}
```

Η `memcpy()` είναι γενικευμένη ως προς τον τύπο των στοιχείων αλλά όχι ως προς τη σειρά προσπέλασης.

Γενίκευση

Πώς γενικεύεται η λειτουργία της αντιγραφής ενός συνόλου στοιχείων; Μετά από εξέταση της `memcpy()` προκύπτει ότι οι ελάχιστες απαιτήσεις της είναι να απαριθμήσει τα στοιχεία του συνόλου, να γνωρίζει που πρέπει να σταματήσει και να γράψει τα στοιχεία στον προορισμό.

Η διαδικασία εύρεσης των ελάχιστων απαιτήσεων ενός αλγορίθμου ονομάζεται *lifting*.

Στον γενικευμένο προγραμματισμό αυτές οι απαιτήσεις ομαδοποιούνται σε *ιδέες (concepts)*.

Παράδειγμα - γενικευμένη αντιγραφή

```
template <typename InputIterator, typename OutputIterator>
OutputIterator copy(InputIterator first,
                   InputIterator last,
                   OutputIterator result)
{
    while (first != last)
        *result++ = *first++;
    return result;
}
```

Concepts

Γενικά μία ιδέα είναι ένα σύνολο απαιτήσεων. Οι απαιτήσεις μπορεί να είναι:

- Έγκυρες παραστάσεις (valid expressions)
- Συμπληρωματικοί τύποι
- Invariants
- Πολυπλοκότητες χώρου ή χρόνου

Concepts

Μία ιδέα που επεκτείνει τις απαιτήσεις μιας άλλης λέγεται ότι αποτελεί *εκλέπτυνσή (refinement)* της (π.χ. Το back insertion sequence αποτελεί refinement του sequence).

Ένας τύπος που πληρεί τις απαιτήσεις λέγεται ότι *μοντελοποιεί (models)* την ιδέα (π.χ. Το vector μοντελοποιεί τα random access container και back insertion sequence).

Concepts και C++

Στην τρέχουσα μορφή της C++ (C++03) τα concepts ορίζονται μόνο έμμεσα, με χρήση templates, ή καταγράφονται σε τεκμηρίωση εκτός της γλώσσας.

Στην επόμενη έκδοση του προτύπου (C++0x) τα concepts (και γενικώς οι ιδέες του γενικευμένου προγραμματισμού) θα είναι ενσωματωμένο χαρακτηριστικό της γλώσσας, με δυνατότητες για τον ορισμό και τη χρήση τους.

Παράδειγμα: InputIterator concept

```
concept InputIterator<typename Iter, typename Value>
{
    requires Regular<Iter>;
    Value operator*(const Iter&);
    Iter& operator++(Iter&);
    Iter operator++(Iter&, int);
}
```

Εισαγωγή στην STL

Η Standard Template Library χρησιμοποιεί τις αρχές του γενικευμένου προγραμματισμού και παρέχει ένα σύνολο δομών δεδομένων και αλγορίθμων. Η δομή της βιβλιοθήκης εγγυάται ότι αλγόριθμοι της STL μπορούν να χρησιμοποιηθούν σε αντικείμενα χρηστών και δομές της STL μπορούν να χρησιμοποιηθούν σε αλγορίθμους χρηστών. Αυτή η ευελιξία οφείλεται στην πλήρη περιγραφή των απαιτήσεων κάθε δομής και αλγορίθμου. Η STL είναι πολύ αποδοτική και βασίζεται σε σταθερά θεωρητικά θεμέλια.

Εισαγωγή στην STL

Δυσκολίες κατά την ανάπτυξη μιας βιβλιοθήκης τύπων δεδομένων:

- Μεγάλο πλήθος τύπων (int, float, user-defined types κτλ.)
- Μεγάλο πλήθος δομών δεδομένων (πίνακες, λίστες, σωροί κτλ.)
- Μεγάλο πλήθος αλγορίθμων (αναζήτηση, ταξινόμηση κτλ.)

Απαιτείται κώδικας για πολύ μεγάλο αριθμό συνδυασμών.

Εισαγωγή στην STL

Οι λύσεις της STL

- Γενικευμένες δομές δεδομένων
- Γενικευμένος τρόπος πρόσβασης στις δομές δεδομένων
- Γενικευμένοι αλγόριθμοι που ενεργούν σε ομάδες διαφορετικών δομών δεδομένων (π.χ. ένας αλγόριθμος ταξινόμησης για λίστες και πίνακες)

Αποτέλεσμα: λιγότερος και πολύ πιο ευέλικτος κώδικας

Συστατικά στοιχεία της STL

Η STL αποτελείται κυρίως από πέντε κατηγορίες αντικειμένων:

- Containers
- Algorithms
- Iterators
- Function Objects
- Adaptors

STL Containers

Τα Containers είναι αντικείμενα που περιέχουν μία οργανωμένη συλλογή άλλων αντικειμένων.

Η STL παρέχει δύο κατηγορίες containers:

- Sequence (ακολουθίες): vector, list, deque
- Associative (συσχετιστικά): set, map, κτλ.

Παράδειγμα sequence container: STL vector

Το vector είναι ένας δυναμικός πίνακας. Παράδειγμα χρήσης:

```
#include <vector>

using namespace std; // all STL objects are in std
...
vector<int> v;
v.push_back(7);
v.push_back(2);
for (int i = 0; i < v.size(); i++)
    cout << v[i];
```

STL Iterators

Οι iterators είναι μία γενίκευση των δεικτών και περιγράφουν τις απαιτήσεις της προσπέλασης των δομών δεδομένων (π.χ. containers). Έτσι επιτρέπουν τον χειρισμό των δομών με ομοιόμορφο τρόπο χωρίς να απαιτείται επιπλέον πληροφορία για την εσωτερική οργάνωση της δομής.

Οι iterators χρησιμοποιούνται στην STL για τον προσδιορισμό θέσης ή διαστήματος σε μία δομή δεδομένων.

STL Iterators

Οι iterators της STL κατατάσσονται σε πέντε κατηγορίες ανάλογα με τις λειτουργίες που επιδέχονται.

Iterator Category	Βασικές λειτουργίες
Input	*it, ++, ==, !=
Output	*it = x, ++
Forward	Input & Output
Bidirectional	Forward, --
Random Access	Bidirectional, +n, -n, +=n, -=n

Κατηγορίες Iterators

Κατηγορίες iterators που παρέχονται από τα containers της STL.

Container	Iterator Category
vector	random access
list	bidirectional
deque	random access
set, multiset	bidirectional
map, multimap	bidirectional

STL Iterators

Παράδειγμα:

```
vector<int> v;  
vector<int>::iterator vi;  
// Could also be vector<int>::const_iterator vi;  
for (vi = v.begin(); vi != v.end(); vi++)  
    cout << *vi;
```

STL Algorithms

Οι αλγόριθμοι στην STL παραμετροποιούνται με διαφορετικούς τύπους iterators (ανάλογα με τις απαιτήσεις του κάθε αλγορίθμου) και επομένως δεν εξαρτώνται από την εσωτερική υλοποίηση των δομών δεδομένων.

Οι κατηγορίες των iterators που παρέχονται από κάθε δομή καθορίζουν και το είδος των αλγορίθμων που μπορούν να εκτελεστούν (αποδοτικά) στη δομή.

Η STL παρέχει μεγάλο αριθμό γενικευμένων αλγορίθμων από απλούς, όπως ο `for_each()`, μέχρι πιο σύνθετους, όπως αλγορίθμους αναζήτησης και ταξινόμησης (π.χ. `sort()`, `binary_search()`).

Οι αλγόριθμοι περιέχονται στο header file `<algorithm>`.

STL Algorithms

Παράδειγμα χρήσης:

```
vector<int> v;  
list<int> l; // Assume these have values
```

```
sort(v.begin(), v.end());
```

```
// iter points to element if found, to v.end() otherwise  
vector<int>::iterator iter = find(v.begin(), v.end(), 5);
```

```
// Same algorithm, different structure  
list<int>::iterator iter2 = find(l.begin(), l.end(), 5);
```

STL Function Objects

Τα function objects (functors) είναι κλάσεις στις οποίες έχει υπερφορτωθεί ο τελεστής () οπότε μπορούν να συμπεριφέρονται σαν συναρτήσεις. Τα functors υπερέχουν έναντι των συναρτήσεων καθώς είναι πιο κομψά κι ευέλικτα στη χρήση τους (π.χ. αντί για δείκτες σε συναρτήσεις) και μπορούν να έχουν υψηλού επιπέδου λειτουργικότητα (π.χ. αποθηκεύοντας εσωτερική κατάσταση).

Η STL παρέχει μεγάλο πλήθος από function objects για διάφορες κατηγορίες λειτουργιών όπως αριθμητικές πράξεις (plus, minus, κτλ.) και συγκρίσεις (less, greater, κτλ.).

STL Function Objects

```
vector<int> v;  
sort(v.begin(), v.end(), less<int>());  
sort(v.begin(), v.end(), greater<int>());
```

```
transform(v.begin(), v.end(), v.begin(),  
          bind1st(plus<int>(), 1));
```

// We can use our own functor with STL algorithms

```
template<typename T> struct Adder {  
    Adder() : total(0) {}  
    void operator()(const T& t) { total += t; }  
  
    T total;  
};
```

// We can generalise for loops

```
Adder<int> A = for_each(v.begin(), v.end(), Adder<int>());  
cout << A.total << endl;
```

STL Adaptors

Οι Adaptors είναι κλάσεις που βασίζονται σε άλλες κλάσεις για την παροχή νέας λειτουργικότητας.

Είναι δυνατόν να «αποκρύπτουν» μεθόδους ή να παρέχουν νέες.

Υπάρχουν adaptors για διάφορα τμήματα της STL (π.χ. containers, iterators).

STL Container Adaptors

Οι πιο συχνά χρησιμοποιούμενοι container adaptors είναι οι `stack`, `queue` και `priority_queue`. Είναι δυνατόν να παραμετροποιηθούν με τη χρήση `vector`, `list` ή `deque`. Παρέχουν τις συνήθεις λειτουργίες στοίβας, ουράς και ουράς προτεραιότητας.

Παράδειγμα χρήσης:

```
stack< vector<int> > s1;  
stack< list<int> > s2;  
s1.push(2); s1.pop();  
s2.push(3); s2.pop();
```

STL Iterator Adaptors

Η STL παρέχει adaptor για τη μετατροπή ενός *bidirectional* ή *random access* iterator σε *reverse iterator*, δηλαδή ενός iterator που διατρέχει τα στοιχεία μίας δομής κατά την αντίθετη φορά.

Ένας *reverse iterator* παρέχεται από τις μεθόδους `rbegin()` και `rend()` των κλάσεων που μοντελοποιούν το concept *reversible container*.

```
vector<int> v;  
v.push_back(1);  
v.push_back(2);  
vector<int>::reverse_iterator vi;  
for (vi = v.rbegin(); vi != v.rend(); vi++)  
    cout << *vi << " "; // prints "2 1"  
  
// Even more generic  
copy(v.rbegin(), v.rend(),  
      ostream_iterator<int>(cout, " "));
```

Άλλα στοιχεία της STL

- Πλήθος από έτοιμους αλγόριθμους και function objects
- Διαχείριση μνήμης: allocators, auto_ptr, κτλ.
- Άλλοι adaptors: negators, iterator adaptors, function pointer adaptors κτλ.

Περισσότερες πληροφορίες

Technical Report 1

Το TR1 περιγράφει επεκτάσεις στην βιβλιοθήκη της C++ που πρόκειται πιθανότατα να περιληφθούν στο επόμενο πρότυπο. Πολλοί μεταγλωτιστές υποστηρίζουν ήδη αρκετά στοιχεία του.

Το TR1 περιέχει μεταξύ άλλων:

- Κλάσεις κατακερματισμού: `unordered_set`, `unordered_map`
- Λειτουργικότητα για `regular expressions`
- Γεννήτριες τυχαίων αριθμών
- Νέα `function objects`

Σχετικοί ιστότοποι

- <http://www.generic-programming.org>
- <http://www.boost.org>