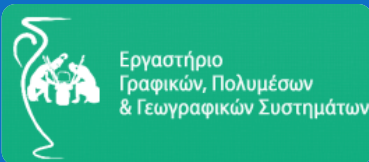


Boost - Boost Graph Library

C++ library

Δομές Δεδομένων

Τεστέμπασης Αθανάσιος
Ε.Τ.Υ.



17 Μαρτίου 2015

testebasis.thanos@gmail.com

Boost

(1/2)

- Η Boost δεν είναι μια βιβλιοθήκη, αλλά μια συλλογή με περισσότερες από 50 βιβλιοθήκες που λειτουργούν καλά μαζί.
- Πολλοί χρήστες ξεκινούν με μόνο μία ή δύο Boost βιβλιοθήκες, και στη συνέχεια επεκτείνουν την χρήση τους με την πάροδο του χρόνου.
- Όλες οι πληροφορίες για την Boost, συμπεριλαμβανομένου του κώδικα, των εγγράφων τεκμηρίωσης, και οτιδήποτε άλλο περιλαμβάνεται στην ιστοσελίδα της Boost είναι δωρεάν.
- Για να γίνουν αποδεκτές οι βιβλιοθήκες από την Boost, είναι απαραίτητο να περάσουν από δημόσια αξιολόγηση. Αυτή η διαδικασία αξιολόγησης έχει οδηγήσει σε πολύ υψηλής ποιότητας βιβλιοθήκες.
- Ο κώδικας υποστηρίζει φορητότητα, έτσι ώστε να επιτραπεί η χρήση σε πολλαπλούς compilers και λειτουργικά συστήματα.
- Ο πηγαίος κώδικας διανέμεται για όλες τις βιβλιοθήκες, ώστε οι χρήστες να μπορούν να ελέγχουν, και να τροποποιούν τον κώδικα για να ανταποκρίνονται στις εκάστοτε ανάγκες.

Boost

(2/2)

- Οι βιβλιοθήκες της Boost λειτουργούν άψογα με τις C++ Standard Βιβλιοθήκες, και περίπου δώδεκα από αυτές έχουν γίνει δεκτές για την επερχόμενη Τεχνική Έκθεση της Standard Library. Αυτό σημαίνει ότι οι βιβλιοθήκες αυτές τελικά θα είναι συμβατές με τους περισσότερους μεταγλωττιστές της C++.
- Τυπικές Βιβλιοθήκες της Boost
 - Smart Pointers
 - Expressions
 - Boost Graph Library

Smart Pointers

(1/3)

- Η διαχείριση της μνήμης στην C++ επαφίεται στον προγραμματιστή.
- Απαιτείται λήψη κρίσιμων αποφάσεων για τη διαχείριση μνήμης.
- Η απρόσεκτη χρήση της μνήμης σωρού μπορεί να οδηγήσει σε απώλειες μνήμης και αποτυχία του προγράμματος.
- Οι Smart Pointers αποτελούν τον ιδανικότερο τρόπο στην C++ για να διασφαλιστεί η ορθή διαχείριση της δυναμικής μνήμης που δεσμεύεται από εκφράσεις τύπου **new**.
- Οι Smart Pointers είναι παρόμοιοι με τους regular pointers, αλλά φροντίζουν για τις λεπτομέρειες της διαγραφής του αντικειμένου στο οποίο δείχνει ο pointer, όταν δεν είναι πλέον απαραίτητο.
- Η C++ πρότυπη βιβλιοθήκη περιέχει ήδη τον `std::auto_ptr`, έναν Smart Pointer που μεταφέρει σημασιολογία ιδιοκτησίας.
- Η Boost προσθέτει επιπρόσθετους smart pointers για χειρισμό άλλων κοινών αναγκών, όπως η σημασιολογία της από κοινού-ιδιοκτησίας.

Smart Pointers

(2/3)

- Η πρότυπη κλάση *shared_ptr* της Boost παρέχει έναν smart pointer κοινής ιδιοκτησίας που μπορεί να διαχειριστεί δυναμικά την κατανομημένη μνήμη σε ένα ευρύ φάσμα εφαρμογών, και μπορεί να χρησιμοποιηθεί για να επικοινωνήσει με τρίτες βιβλιοθήκες.
- Σε αντίθεση με την *std::auto_ptr*, η *shared_ptr* λειτουργεί καλά με τους STL containers.
- Ο ακόλουθος κώδικας αποσκοπεί στη δημιουργία ενός *std::vector* της *shared_ptr* σε ένα τύπο που ονομάζεται *Foo*

```
typedef boost::shared_ptr<Foo>FooPtr;  
std::vector<FooPtr> foo_vector;
```

- Το διάνυσμα μπορεί να γεμίσει ως εξής:

```
foo_vector.push_back( FooPtr(new Foo(3)) );  
foo_vector.push_back( FooPtr(new Foo(2)) );  
foo_vector.push_back( FooPtr(new Foo(1)) );
```

Smart Pointers

(3/3)

```
#include <boost/shared_ptr.hpp>
struct Foo
{
    Foo( int _x ) : x(_x) {}
    ~Foo() { std::cout << "Destructing a Foo with x=" << x
    << "\n"; }
    int x;
    /* ... */
};
typedef boost::shared_ptr<Foo> FooPtr;
struct FooPtrOps
{
    bool operator()( const FooPtr & a, const FooPtr & b )
    { return a->x < b->x; }
    void operator()( const FooPtr & a )
    { std::cout << " " << a->x; }}
```

```
Original foo_vector: 3 2 1
Sorted foo_vector: 1 2 3
Destructing a Foo with x=1
Destructing a Foo with x=2
Destructing a Foo with x=3
```

```
int main()
{
    std::vector<FooPtr> foo_vector;
    foo_vector.push_back( FooPtr(new Foo(3)) );
    foo_vector.push_back( FooPtr(new Foo(2)) );
    foo_vector.push_back( FooPtr(new Foo(1)) );
    std::cout << "Original foo_vector:";
    std::for_each( foo_vector.begin(), foo_vector.end(),
    FooPtrOps() );
    std::cout << "\n";
    std::sort( foo_vector.begin(), foo_vector.end(), FooPtrOps() );
    std::cout << "Sorted foo_vector:";
    std::for_each( foo_vector.begin(), foo_vector.end(),
    FooPtrOps() );
    std::cout << "\n";
    return 0;
}
```

Boost και Regular Expressions

- Η Βιβλιοθήκη Regular Expression της Boost, η οποία συχνά αποκαλείται `regex ++`, έρχεται να καλύψει ένα σημαντικό κενό στην C++ Standard Library.
- Οι Regular Expressions επιτυγχάνουν ταίριασμα των τύπων strings (string pattern matching), και έτσι είναι απαραίτητες για διαχείριση των strings.

- Ακολουθεί ένα απλό παράδειγμα:

```
bool validate_card_format(const std::string s)
{
    static const boost::regex e("(\\d{4}[- ]){3}\\d{4}");
    return regex_match(s, e);
}
```

- Αποτύπωση κώδικα: Να γίνεται δεκτό ένα string το οποίο αποτελείται από τρεις ομάδες των τεσσάρων ψηφίων που ακολουθείται από μια παύλα ή ένα κενό, και τελειώνει με άλλα τέσσερα ψηφία.

Γνωρίσματα Τύπων - Type Traits

- Βοηθάει τους προγραμματιστές να διαχειριστούν το είδος πληροφοριών κατά τη μεταγλώττιση.
- Κάποιες χρήσεις περιλαμβάνουν την υποστήριξη μιας ευρύτερης ποικιλίας τύπων που έχουν γραφτεί από χρήστες, προσθέτοντας ισχυρισμούς χρόνου μεταγλώττισης για τη βελτίωση της ανίχνευσης σφαλμάτων, και την ενεργοποίηση βελτιστοποιήσεων.
- Τα γνωρίσματα τύπων παρέχουν πρότυπα με `value` που θα είναι αληθής ή ψευδής.
 - Π.χ. `boost::is_integral<T>::value`

Εάν ο τύπος `T` αποτελεί αναπόσπαστο τύπο σύμφωνα με τους κανόνες της γλώσσας, η έκφραση αυτή θα αποτιμηθεί σε `true`, διαφορετικά θα αποτιμηθεί σε `false`.

Boost – MetaProgramming Library (MPL)

- Ο όρος Metaprogramming μπορεί να χαρακτηριστεί ως «προγράμματα που παράγουν προγράμματα».
- Τα μεταπρογράμματα τρέχουν σε χρόνο μεταγλώττισης, προσαρμόζοντας τον κώδικα που μεταγλωττίζεται στο τελικό πρόγραμμα.
- Η MPL παρέχει μία πλούσια γκάμα λειτουργιών που εφαρμόζονται στους τύπους, συμπεριλαμβανομένου του ελέγχου της ροής, των containers, των iterators, και των αλγορίθμων, και μεταγλωττίζεται στο τελικό πρόγραμμα.

Boost – MetaProgramming Library (MPL)

- Παράδειγμα:

```
template< typename T >
class auto_size_example : private mpl::if_< sizeof(T) <= sizeof(double)
    , stack_implementation<T>
    , heap_implementation<T>
>::type
{
    // ...
};
```

- Επεξήγηση: Όταν ένα `template auto_size_example` αρχικοποιείται κατά τη μεταγλώττιση, το `if_` της MPL καθορίζει αν θα κληρονομήσει από το `stack_implementation` ή από το `heap_implementation` ανάλογα με το μέγεθος της παραμέτρου `T`. Το καθαρό αποτέλεσμα είναι η βελτιστοποίηση που θα ήταν δύσκολο να εκτελεστεί διαφορετικά.

Boost Spirit Library

- Αποτελεί ένα σύνολο από C++ βιβλιοθήκες για το παρσάρισμα και την παραγωγή αποτελεσμάτων που υλοποιούνται ως ενσωματωμένες γλώσσες που εξαρτώνται από το πεδίο - Domain Specific Embedded Languages (DSEL) που χρησιμοποιούν πρότυπα έκφρασης και πρότυπο μετα-προγραμματισμό.
- Επιτρέπει σε BNF γραμματικές να εκφράζονται απευθείας στην C++, και ως εκ τούτου αποτελεί ένα σημαντικό επίτευγμα.
- Η γραμματική μπορεί να αναμιχθεί ελεύθερα με C++ κώδικα και, χάρη στην παραγωγική δύναμη των C++ προτύπων, είναι αμέσως εκτελέσιμη.

Boost Graph Library

(1/7)

- Ένα μέρος της βιβλιοθήκης των γράφων της Boost είναι μια γενική διεπαφή που επιτρέπει πρόσβαση στη δομή ενός γράφου, αλλά κρύβει τις λεπτομέρειες της υλοποίησης.
- Είναι μια «ανοικτή» διεπαφή υπό την έννοια ότι οποιαδήποτε βιβλιοθήκη γράφων εφαρμόζει αυτήν την διεπαφή θα αλληλεπιδρά με τους BGL γενικούς αλγορίθμους και με άλλους αλγορίθμους που χρησιμοποιούν επίσης αυτήν την διεπαφή.
- Η βιβλιοθήκη BGL παρέχει μερικές κλάσεις γράφων γενικού σκοπού που προσαρμόζονται σε αυτήν την διεπαφή, αλλά δεν προορίζονται να είναι οι «μόνες» κλάσεις γράφων.

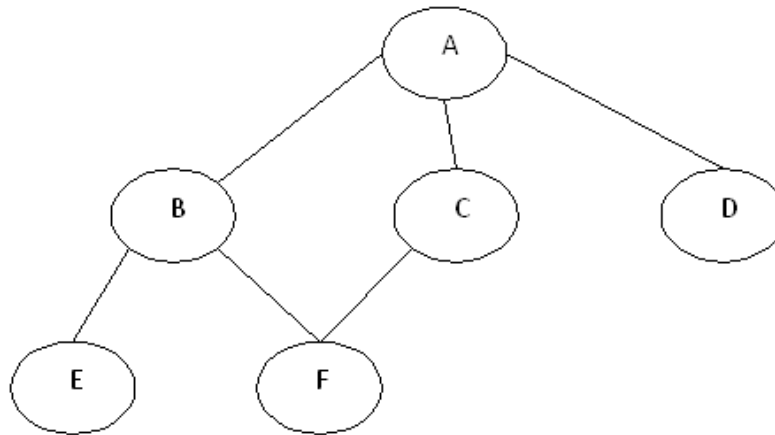
Boost Graph Library

(2/7)

- Γράφημα $G = (V, E)$

Όπου V ένα σύνολο κορυφών $V = \{v_1, v_2, v_3, \dots, v_n\}$

Και E ένα σύνολο ακμών της μορφής $\{v_i, v_j\}$ με $1 \leq i, j \leq n$



Boost Graph Library

(3/7)

■ Τρόποι αναπαράστασης γραφημάτων:

■ Edge List: { (A,B), (A,C), (A,D), (B,E), (B,F), (C, F) }

■ Adjacency List:

A: { B, C, D } B: { A, E, F }

C: { A, F } D: { A }

E: { B } F: { B, C }

■ Adjacency Matrix :

	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	0	0	1	1
C	1	0	0	0	0	1
D	1	0	0	0	0	0
E	0	1	0	0	0	0
F	0	1	1	0	0	0

Boost Graph Library

(4/7)

Expression	Return Type or Description
Graph <i>graph_traits<G>::vertex_descriptor</i> <i>graph_traits<G>::directed_category</i> <i>graph_traits<G>::traversal_category</i> <i>graph_traits<G>::edge_parallel_category</i>	The type of object used to identify vertices. Directed or undirected edges? What kind of iterator traversal is supported? Allow insertion of parallel edges?
IncidenceGraph refines Graph <i>graph_traits<G>::edge_descriptor</i> <i>graph_traits<G>::out_edge_iterator</i> <i>graph_traits<G>::degree_size_type</i> <i>out_edges(v, g)</i> <i>source(e, g)</i> <i>target(e, g)</i> <i>out_degree(v, g)</i>	The type of object used to identify edges. Iterate through the out-edges. The integer type for vertex degree. <i>std::pair<out_edge_iterator, out_edge_iterator></i> <i>vertex_descriptor</i> <i>vertex_descriptor</i> <i>degree_size_type</i>
BidirectionalGraph refines IncidenceGraph <i>graph_traits<G>::in_edge_iterator</i> <i>in_edges(v, g)</i> <i>in_degree(v, g)</i> <i>degree(e, g)</i>	Iterate through the in-edges. <i>std::pair<in_edge_iterator, in_edge_iterator></i> <i>degree_size_type</i> <i>degree_size_type</i>

Boost Graph Library

(5/7)

<p>AdjacencyGraph refines Graph</p> <p><i>graph_traits<G>::adjacency_iterator</i> <i>adjacent_vertices(v, g)</i></p>	<p>Iterate through adjacent vertices.</p> <p><i>std::pair<adjacency_iterator, adjacency_iterator></i></p>
<p>VertexListGraph refines Graph</p> <p><i>graph_traits<G>::vertex_iterator</i> <i>graph_traits<G>::vertices_size_type</i></p> <p><i>num_vertices(g)</i> <i>vertices(g)</i></p>	<p>Iterate through the graph's vertex set.</p> <p>The unsigned integer type for representing the number of vertices.</p> <p><i>vertices_size_type</i></p> <p><i>std::pair<vertex_iterator, vertex_iterator></i></p>
<p>EdgeListGraph refines Graph</p> <p><i>graph_traits<G>::edge_descriptor</i> <i>graph_traits<G>::edge_iterator</i> <i>graph_traits<G>::edges_size_type</i></p> <p><i>num_edges(g)</i> <i>edges(g)</i> <i>source(e, g)</i> <i>target(e, g)</i></p>	<p>The type of object used to identify edges.</p> <p>Iterate through the graph's edge set.</p> <p>The unsigned integer type for representing the number of edges.</p> <p><i>edges_size_type</i></p> <p><i>std::pair<edge_iterator, edge_iterator></i> <i>vertex_descriptor</i> <i>vertex_descriptor</i></p>
<p>AdjacencyMatrix refines Graph</p> <p><i>edge(u, v, g)</i></p>	<p><i>std::pair<edge_descriptor, bool></i></p>

Boost Graph Library

(6/7)

- Βασικοί αλγόριθμοι που προσφέρονται στην BGL:
 - `breadth_first_search`
 - `depth_first_search`
 - `dijkstra_shortest_paths`
 - `bellman_ford_shortest_paths`
 - `kruskal_minimum_spanning_tree`
 - `prim_minimum_spanning_tree`
 - `connected_components`
 - `strong_components`
 - `edmunds_karp_max_flow`
 - `push_relabel_max_flow`

Boost Graph Library

(7/7)

```
template <typename UndirectedGraph> void undirected_graph
demo1() {
    const int V = 3;
    UndirectedGraph undigraph(V);
    typename graph_traits<UndirectedGraph>::vertex_descriptor zero,
    one, two;
    typename graph_traits<UndirectedGraph>::out edge iterator out,
    out end;
    typename graph_traits<UndirectedGraph>::in edge iterator in, in
    end;
    zero = vertex(0, undigraph);
    one = vertex(1, undigraph);
    two = vertex(2, undigraph);
    add edge(zero, one, undigraph);
    add edge(zero, two, undigraph);
    add edge(one, two, undigraph);
```

```
    std::cout << "out edges(0): ";
    for (tie(out, out end) = out edges(zero, undigraph); out != out end;
    ++out)
        std::cout << *out;
    std::cout << std::endl << "in edges(0): ";
    for (tie(in, in end) = in edges(zero, undigraph); in != in end; ++in)
        std::cout << *in;
    std::cout << std::endl;
}
```

```
out edges(0): (0,1) (0,2)
in edges(0): (1,0) (2,0)
```

References

- Boost Site: <http://www.boost.org>
- An Introduction to Boost: <http://www.codeproject.com/KB/stl/boostintro.aspx>
- List Of Libraries In Boost:
http://en.wikipedia.org/wiki/Boost_%28programming%29
- Ενδιαφέρουσα Παρουσίαση για Generic Programming και BGL
<http://ecee.colorado.edu/~siek/boostcon2010bgl.pdf>
- Επιπλέον δείτε το **The Boost Graph Library User Guide and Reference Manual**, **Jeremy Siek** για αναλυτική παρουσίαση, παραδείγματα, γενικό προγραμματισμό, class reference κ.ά.