Εισαγωγή στη Standard Template Library για τη C++

Νίκος Ποθητός
pothitos@di.uoa.gr

3/12/2009

Εισαγωγή στη Standard Template Library για τη C++

- Η *Standard Template Library (STL)* μάς δίνει τη δυνατότητα να χρησιμοποιήσουμε στη C++ έτοιμες δομήσεις δεδομένων, όπως συμβολοσειρές, πίνακες, λίστες κ.ά.
- Ένα από τα πλεονεκτήματα αυτής της προσέγγισης είναι ότι ο περιεχόμενος τύπος μιας δόμησης δεδομένων είναι template, δηλαδή μπορεί να είναι οτιδήποτε. Επίσης η διαχείριση μνήμης γίνεται αυτόματα και δεν χρειάζεται αποσφαλμάτωση. Γενικότερα, έχουμε στη διάθεση μας πολλούς αποδοτικούς αλγορίθμους για την κάθε δόμηση δεδομένων, χωρίς να χρειάζεται να γράψουμε δική μας υλοποίησή τους.
- Η STL εισήχθη στο πρότυπο της ANSI/ISO C++ μόλις το 1994. Επομένως πρέπει να προσέχουμε να χρησιμοποιούμε σχετικά πρόσφατες υλοποιήσεις αυτής της βιβλιοθήκης.

Διαθέσιμες δομήσεις δεδομένων (containers)

`string`	συμβολοσειρά
`rope`	(συνίσταται για μεγάλη) συμβολοσειρά
`vector<T>`	πίνακας
`deque<T>`	πίνακας (κομματιασμένος στη μνήμη)
`list<T>`	διπλά συνδεδεμένη λίστα
`slist<T>`	συνδεδεμένη λίστα
`stack<T>`	στοίβα
`queue<T>`	ουρά
`priority_queue<T>`	ουρά προτεραιότητας
`map<T1,T2>`	
`multimap<T1,T2>`	δυαδικό
`set<T>`	δένδρο[1]
`multiset<T>`	
`bitset<n>`	πίνακας
`vector<bool>`	από bit

[1] Σε κάθε τύπο αυτής της κατηγορίας αντιστοιχεί επίσης και ένας τύπος δεδομένων με πρόθεμα "`unordered_`", ο οποίος υλοποιείται με κατακερματισμό.

string

- Υπάρχουν πολλές μέθοδοι της `string` τις οποίες μπορούμε να χρησιμοποιήσουμε όπως οι `append()`, `assign()`, `insert()`, `remove()`, `replace()`, `resize()`, `copy()`, `find()`, `rfind()`, `find_first_of()`, `find_last_of()`, `find_first_not_of()`, `find_last_not_of()`, `substr()`, `compare()` και `c_str()`.
- Ο τελεστής της πρόσθεσης '`+`' έχει υπερφορτωθεί και χρησιμοποιείται για τη συνένωση `string`.
- Οι τελεστές συγκρίσεων (`==`, `!=`, `<`, `<=`, `>` και `>=`) είναι επίσης υπερφορτωμένοι. Έτσι, δέν υπάρχει τώρα ανάγκη χρήσης συναρτήσεων τύπου `strcmp()`.

Παράδειγμα χρήσης `string`

- Στην πρότυπη βιβλιοθήκη υπάρχει η συνάρτηση `strtok()` (δηλωμένη στο αρχείο επικεφαλίδας `cstring`) για να παίρνουμε τα *λεκτικά σύμβολα* (tokens) μιας δεδομένης συμβολοσειράς τύπου `char*`. Π.χ. τα tokens μιας πρότασης στα ελληνικά είναι οι λέξεις, οι οποίες διαχωρίζονται είτε με κενά, είτε με σημεία στίξης.
- Όσον αφορά τις συμβολοσειρές τύπου `string`, δεν υπάρχει κάτι ανάλογο της `strtok()`. Εμείς θα προσπαθήσουμε να καλύψουμε αυτό το κενό που υπάρχει δημιουργώντας μία κλάση.

`strtok.h` (1/2)

```
#ifndef STRTOK_H
#define STRTOK_H

#include <string>

class  StrTokenizer  {
  private:
    const std::string  str;    // string to break up into tokens
    const std::string  delim;  // the characters in 'delim'
                               //   separate the tokens of 'str'
    unsigned  c;               // current character

  public:
    StrTokenizer(const std::string str_init,
                 const std::string delim_init=" ")
      : str(str_init), delim(delim_init), c(0)
    {
      //  Eating up delimiters (in the begining of 'str')
      while (c < str.size()
        &&  delim.find( str[c] )  !=  std::string::npos)
          ++c;
    }
```

strtok.h (2/2)

```
    std::string  next (void)  // returns the next token
    {
      //  Creating character by character the 'token'
      std::string  token;
      while (c < str.size()
        &&  delim.find( str[c] )  ==  std::string::npos)  {
          token += str[c];
          ++c;
      }

      //  Eating up delimiters
      while (c < str.size()
        &&  delim.find( str[c] )  !=  std::string::npos)
          ++c;
      return  token;
    }
};

#endif  // STRTOK_H
```


strtok_main.cpp

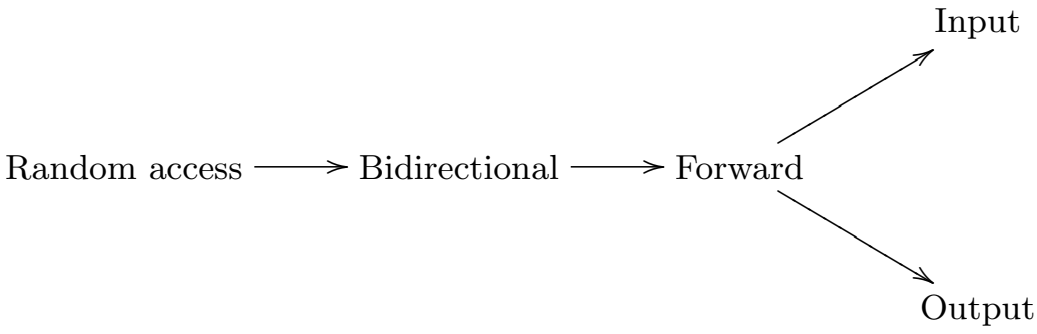
```
#include "strtok.h"
#include <iostream>
using namespace std;

int  main (void)
{
  string  input, delimiters, tok;

  cout << "Please give some input: ";
  getline(cin, input);

  cout << "Please give the delimiters: ";
  getline(cin, delimiters);

  cout << "Found the following tokens:";
  StrTokenizer  tokens(input,delimiters);
  while ((tok = tokens.next()).size() != 0)
      cout << " \"" << tok << "\"";
  cout << "\n";
}
```

strtok_main.cpp: Αποτελέσματα

```
% g++ -o strtok_main strtok_main.cpp
% ./strtok_main
Please give some input: This,is a test sentence. Hello
Please give the delimiters: . ,
Found the following tokens: "This" "is" "a" "test" "sentence"
"Hello"
%
```


Iterators

- Ένας *iterator* (επαναλήπτης) είναι ένας δείκτης σε ένα στοιχείο μιας δόμησης δεδομένων της STL. Οι iterators χρησιμοποιούνται κατά κόρον για την προσπέλαση των περιεχομένων μιας τέτοιας δόμησης δεδομένων.
- Στο παρακάτω σχήμα απεικονίζονται οι σχέσεις μεταξύ των κατηγοριών των iterators. Η κατηγορία "Random access" είναι η γενικότερη.



- Για να ελέγξουμε ότι ένας iterator `it` δεν έχει φτάσει στο τέλος μιας δόμησης δεδομένων `d`, κάνουμε τη σύγκριση `it != d.end()`. Δεν χρησιμοποιείται ο τελεστής '<' όπως ίσως θα περίμενε κανείς, αλλά το '`!=`'.

Γενικές μέθοδοι

- Μερικές από τις μεθόδους ενός τύπου της STL που χρησιμοποιούνται συχνά είναι οι `empty()`, `size()`, `begin()`, `end()`, `rbegin()`, `rend()`, `clear()` και `erase()`.
- Σημαντικές είναι επίσης οι μέθοδοι που αφορούν τα άκρα ενός τύπου (όπως π.χ. οι τύποι `vector`, `deque` και `list`, οι οποίοι αποτελούν κατά κάποιον τρόπο τον «πυρήνα» της STL): `push_back()`, `push_front()`, `back()`, `front()`, `pop_back()` και `pop_front()`.
- Κάτι άλλο που είναι κοινό στους τύπους της STL, έχει να κάνει με την αρχικοποίηση των περιεχομένων τους. Όταν αυτά είναι αριθμητικά, αρχικοποιούνται σε 0 –εφόσον βέβαια δεν τους έχει ανατεθεί άλλη τιμή.

Αλγόριθμοι (1/2)

- Εκτός από δομήσεις δεδομένων και iterators για να προσπελαύνουμε τα στοιχεία τους, η STL παρέχει μία πληθώρα αλγορίθμων για να τα επεξεργαζόμαστε. Το αρχείο επικεφαλίδας με τις αντίστοιχες συναρτήσεις είναι το `algorithm`.
- `adjacent_find()`, `prev_permutation()`, `binary_search()`, `push_heap()`, `copy()`, `random_shuffle()`, `copy_backward()`, `remove()`, `count()`, `remove_copy()`, `count_if()`, `remove_copy_if()`, `equal()`, `remove_if()`, `equal_range()`, `replace()`, `fill()`, `replace_copy()`, `fill_n()`, `replace_copy_if()`, `find()`, `replace_if()`, `find_if()`, `reverse()`, `for_each()`, `reverse_copy()`, `generate()`, `rotate()`, `generate_n()`, `rotate_copy()`, `includes()`, `search()`, `inplace_merge()`, `set_difference()`, `lexicographical_compare()`, `set_intersection()`, `lower_bound()`, `set_symmetric_difference()`, `make_heap()`, `set_union()`, `max()`, `sort()`, `max_element()`, `sort_heap()`, `merge()`, `stable_partition()`, `min()`, `stable_sort()`, `min_element()`, `swap()`, `mismatch()`, `swap_ranges()`, `next_permutation()`, `transform()`, `nth_element()`, `unique()`, `partial_sort()`, `unique_copy()`, `partial_sort_copy()`, `upper_bound()`, `partition()`

Αλγόριθμοι (2/2)

- Παρατηρούμε ότι υπάρχουν, μεταξύ των άλλων, αλγόριθμοι ταξινόμησης, αναζήτησης, αριθμητικών πράξεων, πράξεων μεταξύ συνόλων και σειριακής επεξεργασίας των στοιχείων ενός τύπου της STL. Οι συναρτήσεις αυτές είναι γενικές και αφορούν τους περισσότερους τύπους της STL.
- Ωστόσο πρέπει να προσέχουμε όταν καλούμε μία συνάρτηση π.χ. `sort()` για μια λίστα μέσα στην οποία περιέχονται στιγμιότυπα μιας κλάσης την οποία έχουμε ορίσει εμείς, να ορίζεται για αυτήν την κλάση ο τελεστής σύγκρισης '<', ειδάλλως δεν είναι δυνατόν να γίνει η ταξινόμηση. Όμοια, για τον αλγόριθμο ισότητας `equal()` θα πρέπει να έχουμε ορίσει τον αντίστοιχο τελεστή '==' για τα στοιχεία που πρόκειται να συγκριθούν κ.ο.κ.

vector_list.cpp (1/6)

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


#include <vector>
#include <list>
#include <iterator>
#include <algorithm>


int  square (int i)
{
  return  (i*i);
}


int  main (void)
{
  unsigned  i;
```

vector_list.cpp (2/6)

```
vector<int>  arr(10);

//  Filling 'arr'
for (i=0;  i < arr.size();  ++i)
        arr[i]  =  i % 5;
//  Printing 'arr'
cout << "arr:     ";
for (i=0;  i < arr.size();  ++i)
        cout << arr[i] << " ";

//  Doing the above operations to 'arr', using iterators
i = 0;
for (vector<int>::iterator it=arr.begin();  it!=arr.end();  ++it)
        *it  =  (i++) % 5;
cout << "\n         ";
//  When not modifying the elements, it is
//   better to use a const_iterator.
for (vector<int>::const_iterator cit=arr.begin();
     cit!=arr.end();  ++cit)
        cout << *cit << " ";
cout << "\t[We printed 'arr' using an iterator]\n";
```


vector_list.cpp (3/6)

```
vector<int>  barr;
barr = arr;   // all the elements of 'arr' are copied to 'barr'

//  Inserting '7' in the middle of 'barr'
vector<int>::iterator it  =  barr.begin() + barr.size() / 2;
//  "Pointer" arithmetic is possible, when using a random access
//   iterator.  If the iterator was pointing to a 'list', we could
//   only increase or decrease it by 1 (that is why we call such an
//   iterator "bidirectional").
barr.insert(it, 7);

//  Printing 'barr' using 'copy()'
cout << "barr:    ";
ostream_iterator<int>  out(cout, " ");
copy(barr.begin(), barr.end(), out);
cout << "\t[We inserted '7']\n";
```

vector_list.cpp (4/6)

```
//  Erasing the element in the middle of 'barr' (that is '7')
it = barr.begin() + barr.size() / 2;
//  We forced 'it' to point again to the middle element,
//   because we cannot use its previous value.
barr.erase(it);

cout << "         ";
copy(barr.begin(), barr.end(), out);
cout << "\t[We removed  '7']\n";

sort(barr.begin(), barr.end());
cout << "         ";
copy(barr.begin(), barr.end(), out);
cout << "\t[We sorted 'barr']\n";

cout << "         " << "Max of barr[0-4]:\t"
     << *max_element(barr.begin(), barr.begin()+5) << "\n";
cout << "         " << "Searching for '7':\t" <<
    ((find(barr.begin(), barr.end(), 7) == barr.end())? "not " : "")
     << "found\n";
```


vector_list.cpp (5/6)

```
list<int>  mylist;
//  Filling 'mylist' with the elements of vector 'arr'
copy(arr.begin(), arr.end(), back_inserter(mylist));

cout << "mylist:  ";
copy(mylist.begin(),  mylist.end(),  out);
cout << "\n";

cout << "         ";
copy(mylist.rbegin(), mylist.rend(), out);
cout << "\t[We used a reverse iterator here]\n";

cout << "         "
     << "size: "    << mylist.size()
     << ", front: " << mylist.front()
     << ", back: "  << mylist.back();
cout << ", max: " << *max_element(mylist.begin(), mylist.end())
     << "\n";
```

vector_list.cpp (6/6)

```
    cout << "         "
         << "Pushing ‘0’ at front\n";
    mylist.push_front(0);

    cout << "         "
         << "We have now " << count(mylist.begin(),mylist.end(),0)
         << " zeros\n";

    reverse(mylist.begin(), mylist.end());
    cout << "         ";
    copy(mylist.begin(),  mylist.end(),  out);
    cout << "\t[We reversed ‘mylist’]\n";

    transform(mylist.begin(), mylist.end(), mylist.begin(), square);
    cout << "       ";
    copy(mylist.begin(),  mylist.end(),  out);
    cout << "\t[We squared the elements of the list]\n";

    cout << "‘mylist’ and ‘arr’ " <<
      (equal(mylist.begin(), mylist.end(), arr.begin())? "" : "do not ")
         << "contain the same sequence of elements.\n";
}
```


vector_list.cpp: Αποτελέσματα

```
% g++ -o vector_list vector_list.cpp
% ./vector_list
arr:    0 1 2 3 4 0 1 2 3 4
        0 1 2 3 4 0 1 2 3 4    [We printed ‘arr’ using an iterator]
barr:   0 1 2 3 4 7 0 1 2 3 4  [We inserted ‘7’]
        0 1 2 3 4 0 1 2 3 4    [We removed  ‘7’]
        0 0 1 1 2 2 3 3 4 4    [We sorted ‘barr’]
        Max of barr[0-4]:      2
        Searching for ‘7’:     not found
mylist: 0 1 2 3 4 0 1 2 3 4
        4 3 2 1 0 4 3 2 1 0    [We used a reverse iterator here]
        size: 10, front: 0, back: 4, max: 4
        Pushing ‘0’ at front
        We have now 3 zeros
        4 3 2 1 0 4 3 2 1 0 0  [We reversed ‘mylist’]
      16 9 4 1 0 16 9 4 1 0 0  [We squared the elements of the list]
‘mylist’ and ‘arr’ do not contain the same sequence of elements.
%
```

set.cpp (1/3)

```
#include "strtok.h"
#include <iostream>
using namespace std;


#include <set>
#include <iterator>
#include <algorithm>


int  main (void)
{
  set<string>  Set1, Set2;
  ostream_iterator<string>  out(cout, " ");
  string  input, tok;
```


set.cpp (2/3)

```
  cout << "Give 1st set's elements, separated by spaces: ";
  getline(cin, input);

  StrTokenizer  tokens1(input);
  while ( !(tok = tokens1.next()).empty() )
      Set1.insert(tok);

  cout << "1st set's elements (duplicates removed):      ";
  copy(Set1.begin(), Set1.end(), out);
  cout << "\n";

  cout << "Give 2nd set's elements, separated by spaces: ";
  getline(cin, input);

  StrTokenizer  tokens2(input);
  while ( !(tok = tokens2.next()).empty() )
      Set2.insert(tok);

  cout << "2nd set's elements (duplicates removed):      ";
  copy(Set2.begin(), Set2.end(), out);
  cout << "\n";
```

set.cpp (3/3)

```
  //  Operations between the two sets
  cout << "\n2nd is subset of 1st:  "
       << ((includes(Set1.begin(), Set1.end(),
                     Set2.begin(), Set2.end())) ? "true" : "false");
  cout << "\nUnion:                  ";
  set_union       (Set1.begin(), Set1.end(),
                   Set2.begin(), Set2.end(), out);
  cout << "\nIntersection:           ";
  set_intersection(Set1.begin(), Set1.end(),
                   Set2.begin(), Set2.end(), out);
  cout << "\nDifference:             ";
  set_difference  (Set1.begin(), Set1.end(),
                   Set2.begin(), Set2.end(), out);
  cout << "\n";
}
```


set.cpp: Αποτελέσματα

```
% g++ -o set set.cpp
% ./set
Give 1st set's elements, separated by spaces:
    apple pear tomato potato tomato
1st set's elements (duplicates removed):
    apple pear potato tomato
Give 2nd set's elements, separated by spaces:
    tomato potato
2nd set's elements (duplicates removed):
    potato tomato

2nd is subset of 1st:  true
Union:                 apple pear potato tomato
Intersection:          potato tomato
Difference:            apple pear
%
```

- Εδώ, για την αποθήκευση των δύο συνόλων Set1 και Set2 χρησιμοποιήθηκε ο τύπος set. Θα μπορούσαμε όμως να χρησιμοποιήσουμε και κάποιον άλλο τύπο (π.χ. vector), αφού οι αλγόριθμοι για τις πράξεις μεταξύ συνόλων μπορούν να εφαρμοστούν σε όλους τους τύπους και όχι μόνο στον set.

map.cpp (1/2)

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;


#include <map>


int  main (void)
{
  map<string,int>  whitepages;

  //  Inserting some telephone numbers
  whitepages.insert(make_pair("Mike",   21020));
  whitepages.insert(make_pair("George", 21050));
  whitepages.insert(make_pair("Mary",   21011));
  whitepages.insert(make_pair("Jim",    21090));
  whitepages["Tom"] = 21021;
```


map.cpp (2/2)

```
  //  Printing the whitepages (the lines are sorted by the name).
  //   Besides, 'map' is a binary tree.
  for (map<string,int>::const_iterator cit=whitepages.begin();
       cit!=whitepages.end();  ++cit)  {
          cout << setw(10) << cit->first  << " "
                           << cit->second << "\n";
  }

  //  It is possible to refer to a telephone number
  //   using the operator []
  cout << "\nwhitepages[\"Mary\"]: " << whitepages["Mary"] << "\n";
}
```

map.cpp: Αποτελέσματα

```
% g++ -o map map.cpp
% ./map
    George 21050
       Jim 21090
      Mary 21011
      Mike 21020
       Tom 21021

whitepages["Mary"]: 21011
%
```

Σημείωση: Για να μεταγλωττιστεί ο κώδικας στα μηχανήματα Sun του Τμήματος, θα πρέπει, αντί της προκαθορισμένης έκδοσης του g++, να χρησιμοποιήσουμε την πιο καινούργια έκδοσή του. Το μόνο που χρειάζεται για να γίνει αυτό, είναι να τρέξουμε, πριν τις εντολές του παραπάνω πλαισίου, την εντολή

```
alias g++ '/usr/sfw/bin/g++'
```


Αναφορές

Silicon Graphics,
The Standard Template Library,
http://sgi.com/tech/stl

Bruce Eckel and Chuck Allison,
Thinking In C++ (Second Edition),
Volume 2: Practical Programming,
Prentice Hall, 2003