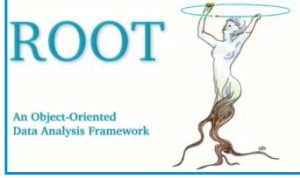
Πρότυπα και διανύσματα

- Πρότυπα συναρτήσεων
- Πρότυπα κλάσεων
  - Παραδείγματα
- Διανύσματα της καθιερωμένης C++
- Επαναλήπτες σε διανύσματα

Πρότυπα συναρτήσεων



- Το πρότυπο (template) είναι μια αφηρημένη συνταγή για την παραγωγή συμπαγούς κώδικα. Τα πρότυπα μπορούν να χρησιμοποιηθούν για τη δημιουργία συναρτήσεων και κλάσεων.
- Για παράδειγμα οι παρακάτω δύο συναρτήσεις αντιμεταθέτουν δύο ακεραίους ή δύο strings:

```
void exchange(int& m, int& n){
    int temp = m;
    m=n;
    n=temp; }
```

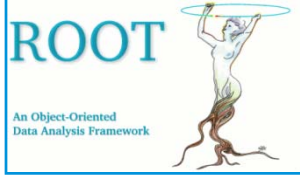
```
void exchange(string& s1, string& s2) {
    string temp = s1;
    s1=s2;
    s2=temp; }
```

- Οι παραπάνω δύο συναρτήσεις κάνουν ουσιαστικά το ίδιο πράγμα. Η μόνη διαφορά έγκειται στον τύπο του αντικειμένου που αντιμεταθέτουν.
- Μπορούμε να αποφύγουμε αυτή την επανάληψη αντικαθιστώντας τις δύο συναρτήσεις με το ακόλουθο πρότυπο συνάρτησης (function template).

```
template <class T>
 void exchange(T& x, T& y) {
    T temp = x;
    x=y;
    y=temp;}
```

- Το σύμβολο T ονομάζεται παράμετρος τύπου (type parameter). Είναι απλώς ένα δεσμευτικό θέσης το οποίο αντικαθίσταται από τον πραγματικό τύπο ή την κλάση κατά την κλήση της συνάρτησης.

Πρότυπα συναρτήσεων




```
// template example
#include <iostream>
using namespace std;

template <class T>
void exchange(T& x, T& y)
{ T temp = x;
  x=y;
  y=temp;
}

int main(){
  int a=7, b=12;
  exchange(a,b);
  cout << "a=" << a << "  b=" << b << endl;

  string s1="Physics", s2="Department";
  exchange(s1,s2);
  cout << "s1=" << s1 << "  s2=" << s2 << endl;
}
```

```
[panos@pc-247 Cpp]$ c++ template.cpp
[panos@pc-247 Cpp]$ a.out
a=12  b=7
s1=Department  s2=Physics
[panos@pc-247 Cpp]$
```

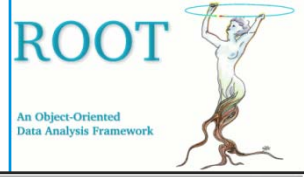
Πρότυπα Κλάσεων

- Τα πρότυπα κλάσεων (class template) δίνουν την δυνατότητα να σχεδιάζουμε κλάσεις οι οποίες έχουν μέλη ο τύπος των οποίων προσδιορίζεται από πρότυπες παραμέτρους. Ακολουθεί απλό παράδειγμα:

```
// template class example
#include <iostream>
using namespace std;

template <class T>
class Zevgos
{ public :
    void set (T x, T y) {a=x; b=y;}
    T add() {return a+b;}
  private :
    T a;
    T b;
};

int main(){
  Zevgos<int> a;
  a.set(4,5);
  cout << a.add() << endl;

  Zevgos<float> b;
  b.set(3.1415,5.27);
  cout << b.add() << endl;
}
```

```
[panos@pc-247 Cpp]$ c++ template_class.cpp
[panos@pc-247 Cpp]$ a.out
9
8.4115
[panos@pc-247 Cpp]$
```

Πρότυπα Κλάσεων: Παράδειγμα Stack

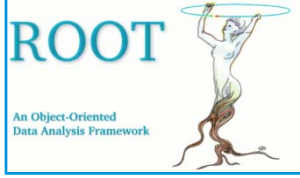


- Στο διπλανό παράδειγμα αναπτύσσουμε το παράδειγμα μιας πρότυπης στοίβας.

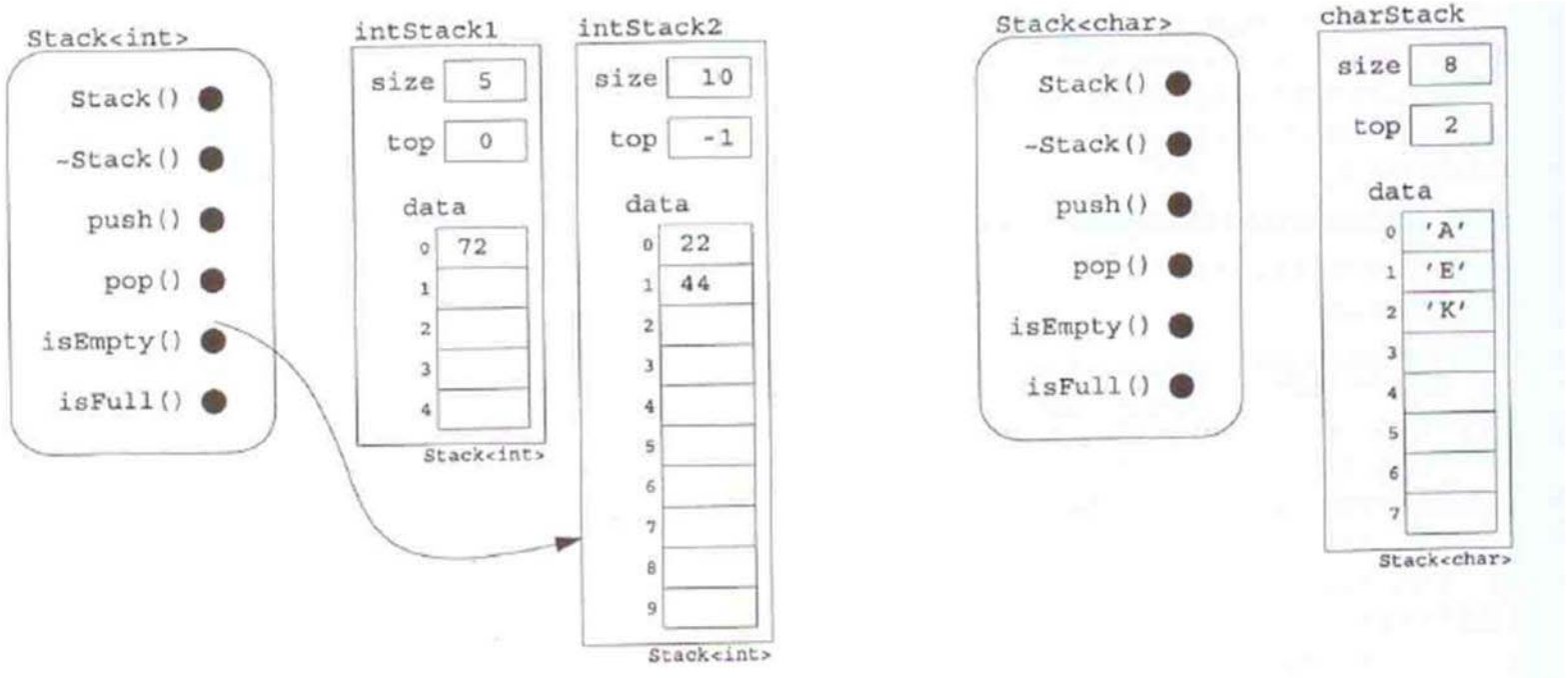
```
// example Stack

#include <iostream>
using namespace std;

template <class T>
class Stack
{
public:
  Stack(int s = 100) : size(s), top(-1) { data = new T[size]; }
  ~Stack() { delete [] data; }
  void push(const T& x) { data[++top] = x; }
  T pop() { return data[top--]; }
  int isEmpty() const { return top == -1; }
  int isFull() const { return top == size - 1; }
private:
  int size;
  int top;
  T* data;
};

int main()
{
  Stack<int> intStack1(5);
  Stack<int> intStack2(10);
  Stack<char> charStack(8);
  intStack1.push(77);
  charStack.push('A');
  intStack2.push(22);
  charStack.push('E');
  charStack.push('K');
  intStack2.push(44);
  cout << intStack2.pop() << endl;
  cout << intStack2.pop() << endl;
  if (intStack2.isEmpty()) cout << "inStack2 is empty.\n";
}
```

```
[panos@pc-247 Cpp]$ c++ stack.cpp
[panos@pc-247 Cpp]$ a.out
44
22
inStack2 is empty.
[panos@pc-247 Cpp]$
```

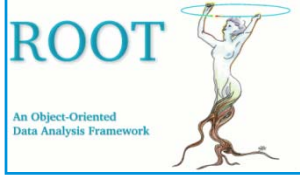
Πρότυπα Κλάσεων: Παράδειγμα Stack



- Η στοίβα (stack) είναι μια απλή δομή δεδομένων η οποία προσομοιώνει μια συνηθισμένη στοίβα αντικειμένων ιδίου τύπου, με τους περιορισμούς ότι ένα αντικείμενο μπορεί να προστεθεί στη στοίβα και να αφαιρεθεί από αυτή μόνον από την κορυφή της.
- Στο παρακάτω διάγραμμα εικονίζονται τα αντικείμενα που δημιουργεί το πρόγραμμα.



Πρότυπα Κλάσεων: Παράδειγμα Vector

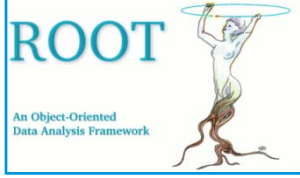


- Στο διπλανό παράδειγμα αναπτύσσουμε ένα απλό παράδειγμα Vector.

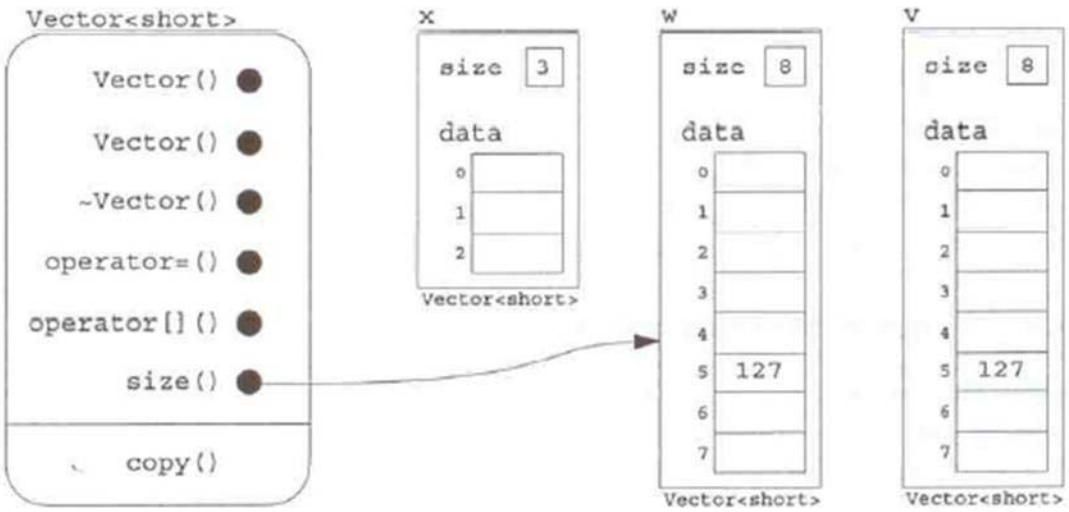
```
// example a Vector class
#include <iostream>
using namespace std;

template<class T>
class Vector
{ public:
    Vector(unsigned n=8) : _sz(n), _data(new T[n]) { }
    Vector(const Vector<T>& v) : _sz(v._sz), _data(new T[v._sz]) { copy(v); }
    ~Vector() { delete [] _data; }
    Vector<T>& operator=(const Vector<T>&);
    T& operator[](unsigned i) const { return _data[i]; }
    unsigned size() { return _sz; }
  protected:
    T* _data;
    unsigned _sz;
    void copy(const Vector<T>&);
};

template<class T>
Vector<T>& Vector<T>::operator=(const Vector<T>& v)
{ _sz = v._sz;
  _data = new T[_sz];
  copy(v);
  return *this;
}

template<class T>
void Vector<T>::copy(const Vector<T>& v)
{ unsigned min_sz = (_sz<v._sz ? _sz : v._sz);
  for (int i=0; i<min_sz; i++)
    _data[i] = v._data[i];
}

int main()
{Vector<short> v;
  v[5] = 127;
  Vector<short> w = v, x(3);
  cout << "x.size()=" << x.size() << endl;
}
```

```
[panos@pc-247 Cpp]$ c++ vector.cpp
[panos@pc-247 Cpp]$ a.out
x.size()=3
[panos@pc-247 Cpp]$
```

Πρότυπα Κλάσεων: Παράδειγμα Vector



- Ένα διάνυσμα (vector) είναι μια δεικτοδοτημένη ακολουθία αντικειμένων του ιδίου τύπου.
- Προσέξτε ότι σε κάθε υλοποίηση συνάρτησης μέλους πρέπει να προηγείται το ίδιο προσδιοριστικό προτύπου που προηγείται της δήλωσης της κλάσης: template<class T>.
- Στο παρακάτω διάγραμμα εικονίζονται τα αντικείμενα που δημιουργεί το πρόγραμμα.



Διανύσματα της καθιερωμένης C++

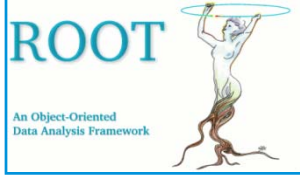
- Τα αντικείμενα vector αποτελούν μια καλή εναλλακτική λύση απέναντι στους πίνακες. Το πρότυπο vector ορίζεται στο αρχείο-κεφαλίδα <vector>.
- Το διπλανό απλό πρόγραμμα δημιουργεί ένα διάνυσμα v με οκτώ αλφαριθμητικά.
- Οι συναρτήσεις load() και print() φορτώνουν και τυπώνουν το διάνυσμα.
- Αντί του αντικειμένου vector το πρόγραμμα μπορεί να γραφεί με τη χρήση ενός πίνακα αλφαριθμητικών:

  string v[size]

```
// simple example vector
#include <iostream>
#include <string>
#include <vector>
using namespace std;
void load(vector<string>&);
void print(vector<string>);
const int SIZE=8;

void load(vector<string>& v)
{ v[0] = "Japan";
  v[1] = "Italy";
  v[2] = "Spain";
  v[3] = "Egypt";
  v[4] = "Chile";
  v[5] = "Zaire";
  v[6] = "Nepal";
  v[7] = "Kenya";
}

void print(vector<string> v)
{ for (int i=0; i<SIZE; i++)
    cout << v[i] << endl;
  cout << endl;
}

int main()
{ vector<string> v(SIZE);
  load(v);
  print(v);
}
```

```
[panos@pc-247 Cpp]$ c++ vector1.cpp
[panos@pc-247 Cpp]$ a.out
Japan
Italy
Spain
Egypt
Chile
Zaire
Nepal
Kenya

[panos@pc-247 Cpp]$
```

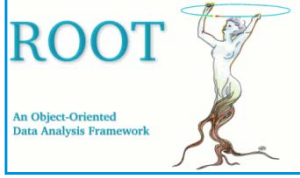
Διανύσματα της καθιερωμένης C++



- Πρόκειται για το ίδιο πρόγραμμα της προηγούμενης διαφάνειας. Εδώ χρησιμοποιείται το αναγνωριστικό τύπου strings στη θέση του vector<string>.
- Παρατηρήστε ότι, όταν δημιουργείται το διάνυσμα v , έχει 0 στοιχεία. Κάθε φορά που καλείται η συνάρτηση push_back(), προσαρτά ένα νέο στοιχείο στο τέλος του διανύσματος και αυξάνει το μέγεθός του.

```
// simple vector example
#include <iostream>
#include <string>
#include <vector>
using namespace std;
typedef vector<string> strings;
void load(strings&);
void print(strings);

void load(strings& v)
{ v.push_back("Japan");
  v.push_back("Italy");
  v.push_back("Spain");
  v.push_back("Egypt");
  v.push_back("Chile");
  v.push_back("Zaire");
  v.push_back("Nepal");
  v.push_back("Kenya");
}

void print(strings v)
{ for (int i=0; i<v.size(); i++)
    cout << v[i] << endl;
  cout << endl;
}

int main()
{ strings v;
  load(v);
  print(v);
}
```

Επαναλήπτες σε διανύσματα



- Ο επαναλήπτης (iterator) είναι ένα αντικείμενο που έχει την δυνατότητα να διατρέχει ένα σύνθετο αντικείμενο-αποδέκτη. Συμπεριφέρεται σαν δείκτης (pointer), εντοπίζοντας ένα στοιχείο κάθε φορά στον αποδέκτη.
- Οι πέντε θεμελιώδεις λειτουργίες ενός επαναλήπτη είναι οι ακόλουθες:
  - Απόδοση αρχικής τιμής στον επαναλήπτη, κάποιας αρχικής θέσης στον αποδέκτη.
  - Επιστροφή των τιμών των δεδομένων που είναι αποθηκευμένες στην τρέχουσα θέση.
  - Αλλαγή της τιμής δεδομένων που είναι αποθηκευμένη στην τρέχουσα θέση.
  - Προσδιορισμός εάν πραγματικά υπάρχει κάποιο στοιχείο στην τρέχουσα θέση του επαναλήπτη.
  - Προώθηση στην επόμενη θέση του αποδέκτη.
- Στο πρόγραμμα που ακολουθεί ορίζεται το αναγνωριστικό τύπου Sit το οποίο είναι επαναλήπτης για διανύσματα αλφαριθμητικών. Κατόπιν χρησιμοποιείται ένας τέτοιος επαναλήπτης στη συνάρτηση print(), για να διατρέξει το διάνυσμα.

Επαναλήπτες σε διανύσματα

```
//  Example vector iterator
#include <iostream>
#include <string>
#include <vector>
using namespace std;
typedef vector<string> strings;
typedef strings::iterator sit;
void load(strings&);
void print(strings);

void load(vector<string>& v)
{ v.push_back("Japan");
  v.push_back("Italy");
  v.push_back("Spain");
  v.push_back("Egypt");
  v.push_back("Chile");
  v.push_back("Zaire");
  v.push_back("Nepal");
  v.push_back("Kenya");
}

void print(strings v)
{ for (sit it=v.begin(); it!=v.end(); it++)
    cout << *it << endl;
  cout << endl;
}

int main()
{ strings v;
  load(v);
  print(v);
}
```

```
[panos@pc-247 Cpp]$ c++ iterator.cpp
[panos@pc-247 Cpp]$ a.out
Japan
Italy
Spain
Egypt
Chile
Zaire
Nepal
Kenya

[panos@pc-247 Cpp]$
```