Πανεπιστήμιο Δυτικής Αττικής
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Αντικειμενοστρεφής Προγραμματισμός (C++)

Ενότητα 6: Templates – Standard Template Library

Καθηγήτρια Κλειώ Σγουροπούλου

Templates συναρτήσεων 1/4

- Επιτρέπουν τη δημιουργία γενικών (generic) συναρτήσεων, οι οποίες δέχονται ως παράμετρο οποιοδήποτε τύπο δεδομένων και επιστρέφουν μια τιμή, χωρίς την ανάγκη υπερφόρτωσης της συνάρτησης με όλους τους πιθανούς τύπους
  - `template <class identifier > func_declaration;`
  - `template <typename identifier > func_declaration;`

```
template <class GenericType>

GenericType GetMax (GenericType a, GenericType b)
{
 return (a>b?a:b);
}
```

Templates συναρτήσεων 2/4

- Η κλήση μιας template **συνάρτησης** γίνεται με τον εξής τρόπο:
  - *function* **<***pattern***>** **(***parameters***);**

  όπου κάθε εμφάνιση του γενικού τύπου δεδομένων στον αρχικό ορισμό της συνάρτησης αντικαθίσταται από τον εκάστοτε τύπο (pattern) των παραμέτρων κλήσης

```
int x,y;
GetMax <int> (x,y);
```

Templates συναρτήσεων 3/4

```
#include <iostream.h>

template <class T>
T GetMax (T a, T b) {
  T result;
  result = (a>b)? a : b;
  return (result);
}

int main () {
  int i=5, j=6, k;
  long l=10, m=5, n;
  k=GetMax<int>(i,j); ή k=GetMax(i,j);
  n=GetMax<long>(l,m); ή n=GetMax(l,m);
  cout << k << endl;
  cout << n << endl;
  return 0;
}
```

```
6
10
```

Κλήση της template συνάρτησης με δυο διαφορετικά patterns

Templates συναρτήσεων 4/4

- Μπορούμε επίσης να δημιουργούμε template συναρτήσεις που δέχονται περαμέτρους διαφορετικού pattern

```
template <class T, class U>

T GetMin (T a, U b)
{
  return (a<b?a:b);
}
```

Κλήση:

```
int i,j;
long l;
i = GetMin<int,long> (j,l); ή i = GetMin(j,l);
```

Templates κλάσεων 1/3

- Τα templates **κλάσεων** επιτρέπουν σε μια κλάση να διαθέτει μέλη γενικού τύπου, ο οποίος δεν επιθυμούμε να προσδιοριστεί τη στιγμή προδιαγραφής της κλάσης

```
template <class T>

class pair {
    T values[2];
  public:
    pair (T first, T second) //inline δήλωση συνάρτησης
    {
      values[0]=first; values[1]=second;
    }
};
```

```
pair<float> myfloats (3.0, 2.18);
```

Templates κλάσεων 2/3

```
#include <iostream.h>

template <class T>
class pair {
    T value1, value2;
  public:
    pair (T first, T second)
      {value1=first; value2=second;}
    T getmax ();
};

template <class T>   // εξωτερική δήλωση συνάρτησης template
T pair<T>::getmax () // κλάσης
{
  T retval;
  retval = value1>value2? value1 : value2;
  return retval;
}
```

Templates κλάσεων 3/3

```
int main () {
  pair <int> myobject (100, 75);
  cout << myobject.getmax();
  return 0;
}
```

Εξειδίκευση templates 1/2

- Επιτρέπει διαφοροποίηση υλοποιήσεων για συγκεκριμένους τύπους
  - **template <> class** *class_name* **<***type***>;**

```
#include <iostream.h>

template <class T>
class pair {
    T value1, value2;
  public:
    pair (T first, T second)
      {value1=first; value2=second;}
    T module () {return 0;}
};
```

Εξειδίκευση templates 2/2


```
template <>
class pair <int> {
    int value1, value2;
  public:
    pair (int first, int second)
      {value1=first; value2=second;}
    int module ();
};

template <>
int pair<int>::module() {return value1%value2;}

int main () {
  pair <int> myints (100,75);
  pair <float> myfloats (100.0,75.0);
  cout << myints.module() << '\n';
  cout << myfloats.module() << '\n';
  return 0;
}
```

Τιμές παραμέτρων για templates 1/2

- Οι παράμετροι templates συναρτήσεων και κλάσεων δεν είναι απαραίτητα γενικού τύπου

```
#include <iostream.h>

template <class T, int N>
class array {
    T memblock [N];
  public:
    void setmember (int x, T value);
    T getmember (int x);
};

template <class T, int N>
array<T,N>::setmember (int x, T value) {
  memblock[x]=value;
}
```

Τιμές παραμέτρων για templates 2/2

```
template <class T, int N>
T array<T,N>::getmember (int x) {
  return memblock[x];
}

int main () {
  array <int,5> myints;
  array <float,5> myfloats;
  myints.setmember (0,100);
  myfloats.setmember (3,3.1416);
  cout << myints.getmember(0) << '\n';
  cout << myfloats.getmember(3) << '\n';
  return 0;
}
```

```
100
3.1416
```

Παραδείγματα templates

```
template <class T>    // The most usual: one class parameter.

template <class T, class U>     // Two class parameters.

template <class T, int N>       // A class and an integer.

template <class T = char>       // With a default value.

template <int Tfunc (int)>      // A function as parameter.
```

Standard Template Library 1/2

- Η STL μας δίνει τη δυνατότητα να χρησιμοποιήσουμε στη C++ έτοιμες δομές δεδομένων, όπως συμβολοσειρές, πίνακες, λίστες, κ.λπ.
- Ένα από τα πλεονεκτήματα αυτής της προσέγγισης είναι ότι ο περιεχόμενος τύπος μιας δομής δεδομένων είναι template, δηλ. μπορεί να είναι οτιδήποτε. Επίσης, η διαχείριση μνήμης γίνεται αυτόματα και δεν χρειάζεται αποσφαλμάτωση.
- Γενικότερα έχουμε στη διάθεσή μας πολλούς αποδοτικούς αλγόριθμους για κάθε δομή δεδομένων, χωρίς να χρειάζεται να κάνουμε δική μας υλοποίηση

Standard Template Library 2/2

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

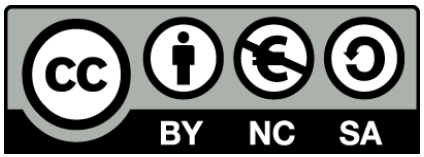
STL: String 1/2

- Υπάρχουν πολλές μέθοδοι της string τις οποίες μπορούμε να χρησιμοποιούμε:
  - **`insert()`, `remove()`, `replace()`, `resize()`, `copy()`, `find()`, `rfind()`, `find_first_of()`, `find_last_of()`, `substr()`, `compare()`, `c_str()`,** κ.α.
- Ο τελεστής "**+**" έχει υπερφορτωθεί και χρησιμοποιείται για τη συνένωση `strings`
- Οι τελεστές συγκρίσεων είναι επίσης υπερφορτωμένοι. Έτσι, δεν υπάρχει τώρα ανάγκη της χρήσης συναρτήσεων τύπου **`strcmp()`**

STL: String 2/2

- Ο τελεστής '**+**' έχει υπερφορτωθεί και χρησιμοποιείται για τη συνένωση strings
- Οι τελεστές συγκρίσεων είναι επίσης υπερφορτωμένοι. Έτσι, δεν υπάρχει τώρα ανάγκη της χρήσης συναρτήσεων τύπου **`strcmp()`**

ΤΕΛΟΣ ΕΝΟΤΗΤΑΣ



Σημειώματα

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης

Επεξήγηση όρων χρήσης έργων τρίτων

©	Δεν επιτρέπεται η επαναχρησιμοποίηση του έργου, παρά μόνο εάν ζητηθεί εκ νέου άδεια από το δημιουργό.
διαθέσιμο με άδεια CC-BY	Επιτρέπεται η επαναχρησιμοποίηση του έργου και η δημιουργία παραγώγων αυτού με απλή αναφορά του δημιουργού.
διαθέσιμο με άδεια CC-BY-SA	Επιτρέπεται η επαναχρησιμοποίηση του έργου με αναφορά του δημιουργού, και διάθεση του έργου ή του παράγωγου αυτού με την ίδια άδεια.
διαθέσιμο με άδεια CC-BY-ND	Επιτρέπεται η επαναχρησιμοποίηση του έργου με αναφορά του δημιουργού. Δεν επιτρέπεται η δημιουργία παραγώγων του έργου.
διαθέσιμο με άδεια CC-BY-NC	Επιτρέπεται η επαναχρησιμοποίηση του έργου με αναφορά του δημιουργού. Δεν επιτρέπεται η εμπορική χρήση του έργου.
διαθέσιμο με άδεια CC-BY-NC-SA	Επιτρέπεται η επαναχρησιμοποίηση του έργου με αναφορά του δημιουργού και διάθεση του έργου ή του παράγωγου αυτού με την ίδια άδεια. Δεν επιτρέπεται η εμπορική χρήση του έργου.
διαθέσιμο με άδεια CC-BY-NC-ND	Επιτρέπεται η επαναχρησιμοποίηση του έργου με αναφορά του δημιουργού. Δεν επιτρέπεται η εμπορική χρήση του έργου και η δημιουργία παραγώγων του.
διαθέσιμο με άδεια CC0 Public Domain	Επιτρέπεται η επαναχρησιμοποίηση του έργου, η δημιουργία παραγώγων αυτού και η εμπορική του χρήση, χωρίς αναφορά του δημιουργού.
διαθέσιμο ως κοινό κτήμα	Επιτρέπεται η επαναχρησιμοποίηση του έργου, η δημιουργία παραγώγων αυτού και η εμπορική του χρήση, χωρίς αναφορά του δημιουργού.
χωρίς σήμανση	Συνήθως δεν επιτρέπεται η επαναχρησιμοποίηση του έργου.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.