Εισαγωγή στον Προγραμματισμό

Μέρος 7ο: Πρότυπα

Εξάμηνο Σπουδών: 3ο

Κωδικός Μαθήματος: 343

Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Μιχάλης Α. Μπέκος
bekos@uoi.gr

Μέρος 1°:
Εισαγωγή με ένα παράδειγμα

Προσέγγιση με ένα παράδειγμα

- Ας υποθέσουμε ότι επιθυμούμε να γράψουμε:
  - μια συνάρτηση σύγκρισης δύο ακεραίων αριθμών.
  - μια συνάρτηση σύγκρισης δύο χαρακτήρων.

- **Λύση:** Υπερφόρτωση συναρτήσεων !

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

// Function to compare two integers.
int compare(const int &x, const int &y) {
  if (y < x) return 1;
  if (x < y) return -1;
  return 0;
}

// Function to compare two characters.
int compare(const char &x, const char &y) {
  if (y < x) return 1;
  if (x < y) return -1;
  return 0;
}

int main() {
  int x = 1, y = 2;
  char a = 'a', b = 'b';
  cout << compare(x,y) << endl;
  cout << compare(a,b) << endl;
  return 0;
}
```

Part07/tmplintro.cpp

Προσέγγιση με ένα παράδειγμα

- Ας υποθέσουμε ότι επιθυμούμε να γράψουμε:
  - μια συνάρτηση σύγκρισης δύο ακεραίων αριθμών.
  - μια συνάρτηση σύγκρισης δύο χαρακτήρων.

- **Λύση:** Υπερφόρτωση συναρτήσεων !

- **Παρατήρηση:** Οι δύο υλοποιήσεις είναι σχεδόν πανομοιότυπες! [code duplication]
  - Τι θα γινόταν αν επιθυμούσαμε μια συνάρτηση σύγκρισης για κάθε συγκρίσιμο τύπο?
  - Θα προτιμούσαμε γράψουμε “γενικό κώδικα”, ανεξάρτητο του τύπου δεδομένων (πολυμορφικό).

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

// Function to compare two integers.
int compare(const int &x, const int &y) {
  if (y < x) return 1;
  if (x < y) return -1;
  return 0;
}

// Function to compare two characters.
int compare(const char &x, const char &y) {
  if (y < x) return 1;
  if (x < y) return -1;
  return 0;
}

int main() {
  int x = 1, y = 2;
  char a = 'a', b = 'b';
  cout << compare(x,y) << endl;
  cout << compare(a,b) << endl;
  return 0;
}
```

Part07/tmplintro.cpp

Λύση: Πρότυπα

- Η C++ υποστηρίζει πρότυπα:
  - Συναρτήσεις ή κλάσεις που δέχονται έναν ή περισσότερους τύπους ως παράμετρο.
- Ορίζετε τη συνάρτηση ή την κλάση μία φορά με τύπο-αναγνωριστικό τρόπο (πρότυπο).
- Κατά τη μεταγλώττιση, θα δημιουργηθεί “εξειδικευμένος” κώδικας από το πρότυπό σας χρησιμοποιώντας κατάλληλους τύπους.

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

// Function to compare two (comparable)
// types.
template <typename T>
int compare(const T &x, const T &y) {
  if (y < x) return 1;
  if (x < y) return -1;
  return 0;
}

int main() {
  int x = 1, y = 2;
  char a = 'a', b = 'b';
  string s = "Bod", t = "Alice";

  cout << compare(x,y) << endl;
  cout << compare(a,b) << endl;
  cout << compare(s,t) << endl;
  // Also correct (but not necessary):
  // cout << compare<int>(x,y) << endl;
}
```

Part07/tmplcomp.cpp

Μέρος 2°:
Πρότυπα

Πρότυπα: Συναρτήσεις

- Ο ορισμός μιας συνάρτησης πρότυπο ως προς τους τύπους `T1, T2,...` γίνεται με την εισαγωγή της παρακάτω εντολής πριν την δήλωση της συνάρτησης:

  `template <typename T1, typename T2, ...>`

- Η γραμμή αυτή ονομάζεται πρόθεμα προτύπου και λέει στον μεταγλωττιστή ότι ακολουθεί πρότυπο και ότι οι παράμετροι `T1, T2,..` είναι τύποι.

- Η λέξη-κλειδί "`typename`" στον ορισμό μπορεί να αντικατασταθεί από την λέξη-κλειδί "`class`", αλλά αυτό μπορεί να δημιουργήσει σύγχυση, καθώς δεν αντιστοιχεί στην κλάση όπως την έχουμε μάθει ως τώρα.

```cpp
#include <iostream>

// Bubble sort using a template type.
template <typename T>
void bubbleSort(T a[], int n) {
  for (int i=0; i<n-1; i++)
    for (int j=0; j<n-i-1; j++)
      if (a[j] > a[j+1])
        swap(a[j], a[j+1]);
}

// Swaps two elements.
template <typename T>
void swap(T &x, T &y) {
  T temp = x; x = y; y = temp;
}

int main() {
  int digits[] = {1,7,3,9,5,4,8,0,2,6};
  bubbleSort(digits, 10);
  for (int i=0; i<10; i++)
    std::cout << digits[i] << " ";
  return 0;
}
```

Part07/tmplsort.cpp

Πώς λειτουργεί;

- Θεωρήστε το παράδειγμα του προτύπου `abs`
- Στο παράδειγμα, το `T` μπορεί να αντικατασταθεί από οποιονδήποτε τύπο:
  - Προκαθορισμένο ή ορισμένο από το χρήστη
- Στο σώμα της συνάρτησης:
  - Το `T` χρησιμοποιείται όπως οποιοσδήποτε άλλος τύπος
- **Σημείωση:** Μπορεί να χρησιμοποιηθεί και άλλος συμβολισμός πέραν του `T`, αλλά ο `T` είναι ο “παραδοσιακός”.

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

template <typename T>
T abs(const T &x) {
  if (x >= 0) return x;
  return -x;
}

int main() {
  int i = -2;
  long l = 10;
  double d = -7.2;

  cout << "abs(i) = " << abs(i) << endl;
  cout << "abs(l) = " << abs(l) << endl;
  cout << "abs(d) = " << abs(d) << endl;

  return 0;
}
```

```
$ abs(i) = 2
  abs(l) = 10
  abs(d) = 7.2
```

Part07/tmplabs.cpp

Πώς λειτουργεί;

- Θεωρήστε το παράδειγμα του προτύπου `abs`
- Κατά την εκτέλεση, ο μεταφραστής αποφασίζει τον πραγματικό τύπο του `T` με βάση τη μεταβλητή εισόδου.
- Π.χ., ο τύπος είναι `int` για την κλήση `abs(i)`.
- Οπότε, ο μεταφραστής δημιουργεί την παρακάτω συνάρτηση με βάση το πρότυπο:

```
int abs(const int &x) {
  if (x >= 0) return x;
  return -x;
}
```

- Ο κώδικας αυτός δημιουργείται μόνο για τους τύπους που χρησιμοποιούν το πρότυπο.

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

template <typename T>
T abs(const T &x) {
  if (x >= 0) return x;
  return -x;
}

int main() {
  int i = -2;
  long l = 10;
  double d = -7.2;

  cout << "abs(i) = " << abs(i) << endl;
  cout << "abs(l) = " << abs(l) << endl;
  cout << "abs(d) = " << abs(d) << endl;

  return 0;
}
```

```
$ abs(i) = 2
  abs(l) = 10
  abs(d) = 7.2
```

Part07/tmplabs.cpp

Συνηθισμένα προβλήματα

- Ποιο είναι το πρόβλημα σε αυτές τις συναρτήσεις;

```
#include <iostream>

template <typename T1, typename T2>
T1 max(const T1 &x, const T1 &y) {
  if (x >= y) return x;
  return y;
}

T2 min(const T2 &x, const T2 &y) {
  if (x <= y) return x;
  return y;
}

int main() {
  int x = 1, y = 2;
  double a = 1.0, b = 2.0;
  std::cout << min(a,x) << std::endl;
  std::cout << min(b,y) << std::endl;
}
```

Part07/tmplminmax.cpp

Συνηθισμένα προβλήματα

- Ποιο είναι το πρόβλημα σε αυτές τις συναρτήσεις;

Το πρότυπο `T2` δεν χρησιμοποιείται

```
#include <iostream>

template <typename T1, typename T2>
T1 max(const T1 &x, const T1 &y) {
  if (x >= y) return x;
  return y;
}

T2 min(const T2 &x, const T2 &y) {
  if (x <= y) return x;
  return y;
}

int main() {
  int x = 1, y = 2;
  double a = 1.0, b = 2.0;
  std::cout << min(a,x) << std::endl;
  std::cout << min(b,y) << std::endl;
}
```

```
$note:couldn't deduce template parameter 'T2'
```

Part07/tmplminmax.cpp

Συνηθισμένα προβλήματα

- Ποιο είναι το πρόβλημα σε αυτές τις συναρτήσεις;

Το πρότυπο `T2` δεν χρησιμοποιείται

Λείπει το αναγνωριστικό του προτύπου

```
#include <iostream>

template <typename T1, typename T2>
T1 max(const T1 &x, const T1 &y) {
  if (x >= y) return x;
  return y;
}

T2 min(const T2 &x, const T2 &y) {
  if (x <= y) return x;
  return y;
}

int main() {
  int x = 1, y = 2;
  double a = 1.0, b = 2.0;
  std::cout << min(a,x) << std::endl;
  std::cout << min(b,y) << std::endl;
}
```

```
$note:couldn't deduce template parameter 'T2'
$error:'T2' does not name a type
```

Part07/tmplminmax.cpp

Συνηθισμένα προβλήματα

- Ποιο είναι το πρόβλημα σε αυτές τις συναρτήσεις;

Το πρότυπο `T2` δεν χρησιμοποιείται

Λείπει το αναγνωριστικό του προτύπου

Ασαφής κλήση

```
#include <iostream>

template <typename T1, typename T2>
T1 max(const T1 &x, const T1 &y) {
  if (x >= y) return x;
  return y;
}

T2 min(const T2 &x, const T2 &y) {
  if (x <= y) return x;
  return y;
}

int main() {
  int x = 1, y = 2;
  double a = 1.0, b = 2.0;
  std::cout << min(a,x) << std::endl;
  std::cout << min(b,y) << std::endl;
}
```

```
$note:couldn't deduce template parameter 'T2'
$error:'T2' does not name a type
$error:no matching function for call to
'max(int&, double&)'
```

Part07/tmplminmax.cpp

Πρότυπα με παραμέτρους που δεν είναι τύποι

- Σε ένα πρότυπο (κλάσης ή συνάρτησης) μπορούμε να δηλώσουμε και παραμέτρους που δεν είναι τύποι, π.χ., ακεραίους.

  Π.χ., `template <typename T, unsigned int I>`

- Έτσι ορίζονται οι πίνακες-αντικείμενα της C++

  `std::array<int, 5> arr = {1,2,3,4,5};`

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

// Outputs the given array
template <typename T, unsigned int N>
void print(const array<array<T,N>, N>& A);

int main() {
  array<array<int,2>, 2> A =
                {{ { 1, 2 }, { 3, 4 } }};
  print(A);
  return 0;
}

template <typename T, unsigned int N>
void print(const array<array<T,N>, N>& A)
{
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++)
      cout << A[i][j] << " ";
    cout << endl;
  }
}
```

Part07/tmplmatrices.cpp

Πολλαπλασιασμός πινάκων

- Επίδειξη: Πολλαπλασιασμός πινάκων με τη χρήση προτύπων
  - `Part07/tmplmatrices.cpp`
- Συγκρίνετε την υλοποίηση με αυτή του `Part03/matrices.cpp`

Πρότυπα: Κλάσεις

- Ο ορισμός μιας κλάσης πρότυπο γίνεται όμοια με αυτόν μιας συνάρτησης πρότυπο εισάγωντας το αντίστοιχο αναγνωριστικό πριν την κλάση.
- Αν η υλοποίηση μιας μεθόδου δεν γίνει εντός της κλάσης, τότε το αναγνωριστικό του προτύπου θα πρέπει να προηγηθεί της υλοποίησης.
- Κατά την αρχικοποίηση ενός αντικειμένου, θα πρέπει να δηλώσουμε τους τύπους δεδομένων του προτύπου.

```
#include <iostream>

template <typename T>
class Pair {
private:
  T first;
  T second;
public:
  Pair(T f, T s): first(f), second(s) {}

  T getFirst() const;
  T getSecond() const;
  void setFirst(T f);
  void setSecond(T s);
};
template <typename T>
T Pair<T>::getFirst() const {
  return first;
}
...
int main() {
  Pair<int> p(1,1);
  std::cout << p.getFirst();
}
```

Part07/tmplpair.cpp

Πρότυπα: Κλάσεις

- **Υπενθύμιση:** Σε ένα πρότυπο (κλάσης ή συνάρτησης) μπορούμε να δηλώσουμε και παραμέτρους που δεν είναι τύποι.

  Π.χ., `template <typename T, int I>`

- Συγκρίνετε την υλοποίηση με αυτή του `Part03/Stack.cpp`

```cpp
#include <iostream>

template <typename T, int CAPACITY>
class Stack {
private:
  T elements[CAPACITY];
  int top;
public:
  Stack(): top(-1) {}
  bool empty() {
    return top < 0;
  }
  void push(T element) {
    elements[++top] = element;
  }
  T pop() {
    return elements[top--];
  }
};
int main() {
  Stack<int,10> s;  //Specific type+data
  for (int i=0; i<10; i++)
    s.push(i);
}
```

Part07/tmplstack.cpp

Συνδεδεμένες λίστες

- Επίδειξη: Υλοποίηση λίστας με τη χρήση προτύπων
  - `Part07/list/LinkedList.cpp`
- Συγκρίνετε την υλοποίηση με αυτή του `Part06/list/Linkedlist.cpp`

Επιπλέον υλικό

- Κεφάλαιο 14 (σ.σ., 707–723):
  R. Lafore, Αντικειμενοστρεφής προγραμματισμός με τη C++,
  ISBN: 960-209-904-6, εκδόσεις Κλειδάριθμος, 2006.