Templates (πρότυπα)

#16

Τμήμα Πληροφορικής και Τηλεπικοινωνιών (Άρτα)

Πανεπιστήμιο Ιωαννίνων

Γκόγκος Χρήστος

Πρότυπα

- Τα πρότυπα επιτρέπουν την περιγραφή λειτουργιών που μπορούν να εφαρμοστούν σε πολλαπλούς τύπους δεδομένων.
  - Γενερικός προγραμματισμός (generic programming)
  - Επαναχρησιμοποίηση κώδικα
- Πρότυπα συναρτήσεων (function templates)
  - Επιτρέπουν τον ορισμό λογικής με τέτοιο τρόπο έτσι ώστε η ίδια αλγοριθμική λογική να μπορεί να εφαρμοστεί σε πολλαπλούς τύπους δεδομένων.
- Πρότυπα κλάσεων (class templates)
  - Επιτρέπουν τον ορισμό generic προτύπων κλάσεων που δίνουν τη δυνατότητα προσάρτησης συγκεκριμένων τύπων δεδομένων έτσι ώστε να προκύπτουν νέες κλάσεις.

Πρότυπα συναρτήσεων

- Οι συναρτήσεις της C++ λειτουργούν με συγκεκριμένους τύπους δεδομένων. Συχνά προκύπτει η ανάγκη του να γράψουμε διαφορετικές συναρτήσεις για να πραγματοποιηθεί η ίδια λειτουργία με διαφορετικούς τύπους δεδομένων.

```
int maximum(int a, int b, int c)
{
        int max = a;
        if (b>max) max = b;
        if (c>max) max = c;
        return max;
}
```

```
float maximum(float a, float b, float c)
{
        float max = a;
        if (b>max) max = b;
        if (c>max) max = c;
        return max;
}
```

- Η λογική είναι ακριβώς η ίδια, αλλά ο τύπος δεδομένων είναι διαφορετικός.
- Τα πρότυπα συναρτήσεων επιτρέπουν στη λογική να γραφεί μια φορά και να χρησιμοποιείται για όλους τους τύπους δεδομένων (generic functions).

Πρότυπα συναρτήσεων

- Generic συνάρτηση εύρεσης μεγαλύτερης τιμής από τρεις τιμές.

```
template <class T>
T maximum(T a, T b, T c)
{
        T max = a;
        if (b>max) max = b;
        if (c>max) max = c;
        return max;
}
```

ή

```
template <typename T>
T maximum(T a, T b, T c)
{
        T max = a;
        if (b>max) max = b;
        if (c>max) max = c;
        return max;
}
```

- Ένα πρότυπο συνάρτησης δεν αποτελεί μια πλήρως ορισμένη συνάρτηση για το μεταγλωττιστή, καθώς ο μεταγλωττιστής χρειάζεται να γνωρίζει τον πραγματικό τύπο δεδομένων για να δημιουργήσει τον αντίστοιχο κώδικα. Συχνά, πρότυπα συναρτήσεων τοποθετούνται σε αρχεία επικεφαλίδων (.h ή .hpp) για να συμπεριληφθούν σε προγράμματα που χρησιμοποιούν τη συνάρτηση. Ο μεταγλωττιστής δημιουργεί τον κώδικα της συνάρτησης με βάση την πραγματική χρήση του προτύπου συνάρτησης.

Χρήση προτύπων συνάρτησης

- Από τη στιγμή που ένα πρότυπο συνάρτησης έχει οριστεί, μπορεί να χρησιμοποιηθεί περνώντας παραμέτρους πραγματικών τύπων.

```
template <class T>
T maximum(T a, T b, T c)
{
        T max = a;
        if (b>max) max = b;
        if (c>max) max = c;
        return max;
}
int i1, i2, i3;
int m = maximum(i1, i2, i3);
```

- Η κλήση θα προκαλέσει την κλήση του προτύπου συνάρτησης με T==int. Η δε τιμή επιστροφής της συνάρτησης θα είναι int.

Ένα ακόμα παράδειγμα

```
template <class T>
void printArray(const T *a, const int count)
{
        for(int i=0;i<count;i++)
                cout << array[i] << “ ”;
        cout << endl;
}
…
char cc[100];
int ii[100];
double dd[100];
myclass xx[100]; // μπορεί να χρησιμοποιηθεί ακόμα και τύπος ορισμένος από το χρήστη
…
printArray(cc,100);
printArray(ii,100);
printArray(dd,100);
printArray(xx,100);
```

Χρήση προτύπου συνάρτησης

- Μπορεί οποιοσδήποτε τύπος ορισμένος από το χρήστη να χρησιμοποιηθεί με ένα πρότυπο συνάρτησης;
  - Όχι πάντα, μόνο οι τύποι δεδομένων που υποστηρίζουν όλες τις λειτουργίες που χρησιμοποιούνται στη συνάρτηση.
  - Στο προηγούμενο παράδειγμα αν η κλάση `myclass` δεν έχει υπερφορτώσει τον τελεστή <<, τότε η συνάρτηση `printArray` δεν θα λειτουργήσει για το συγκεκριμένο τύπο δεδομένων.

Άσκηση #1: Templated function

- Γράψτε μια γενερική συνάρτηση με όνομα index_of που δέχεται ένα std::vector οποιουδήποτε τύπου, το πλήθος των στοιχείων και μια τιμή στόχο, και επιστρέφει τη θέση της πρώτης εμφάνισης της τιμής στόχου ή -1 αν δεν βρεθεί.
- Γράψτε μια δεύτερη γενερική συνάρτηση με όνομα swap_values που ανταλλάσσει τις τιμές δύο μεταβλητών του ίδιου τύπου.
- Στη main() δημιουργήστε διανύσματα δύο διαφορετικών τύπων (int και std::string), και καλέστε την index_of για κάθε περίπτωση και εμφανίσετε τα αποτελέσματα.
- Επίσης καλέστε μέσα από main() την swap_values για δύο τύπους δεδομένων (int και float) και εμφανίστε αποτελέσματα που να επιδεικνύουν την λειτουργία της συνάρτησης.

Άσκηση #1: Λύση

```cpp
template <typename T>
int index_of(const std::vector<T>& vec,
int size, const T& target) {
  for (int i = 0; i < size; ++i) {
    if (vec[i] == target) {
      return i;
    }
  }
  return -1;
}

template <typename T>
void swap_values(T& a, T& b) {
  T temp = a;
  a = b;
  b = temp;
}
```

```cpp
int main() {
  std::vector<int> v_int = {10, 20, 30, 40, 50};
  std::vector<std::string> v_str = {"arta", "ioannina", "preveza",
                                    "igoumenitsa"};

  int idx_int = index_of(v_int, v_int.size(), 30);
  int idx_str = index_of(v_str, v_str.size(), std::string("preveza"));

  std::cout << "position of 30 in v_int: " << idx_int << std::endl;
  std::cout << "position of \"preveza\" in v_str: " << idx_str << std::endl;

  int a = 5, b = 9;
  std::cout << "\nbefore swap (int): a = " << a << ", b = " << b << std::endl;
  swap_values(a, b);
  std::cout << "after swap (int): a = " << a << ", b = " << b << std::endl;

  float x = 1.5f, y = 3.7f;
  std::cout << "\nbefore (float): x = " << x << ", y = " << y << std::endl;
  swap_values(x, y);
  std::cout << "after (float): x = " << x << ", y = " << y << std::endl;

  return 0;
}
```

```
position of 30 in v_int: 2
position of "preveza" in v_str: 2
before swap (int): a = 5, b = 9
after swap (int): a = 9, b = 5

before (float): x = 1.5, y = 3.7
after (float): x = 3.7, y = 1.5
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/oop16_ask1.cpp

Πρότυπα κλάσεων (class templates)

- Μερικές φορές είναι χρήσιμο να επιτρέπεται η αποθήκευση τιμών διαφορετικών τύπων σε μια κλάση.
- Δείτε τα παραδείγματα `simplelist1` και `simplelist2`

https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/simplelist1.h

https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/simplelist1.cpp

https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/main_simplelist1.cpp

Λίστα 10 θέσεων με ακεραίους

https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/simplelist2.cpp

https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/simplelist2.cpp

https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/main_simplelist2.cpp

Λίστα 10 θέσεων με ακεραίους ή πραγματικούς

Πρότυπα κλάσεων

- Η ίδια ιδέα με τα πρότυπα συναρτήσεων εφαρμόζεται και στα πρότυπα κλάσεων.
- Δείτε το παράδειγμα `simplelist3`

https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/simplelist3.h

https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/simplelist3.cpp

https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/main_simplelist3.cpp

Πρότυπα κλάσεων

- Για να γίνει μια κλάση, πρότυπο κλάσης χρειάζεται να τοποθετηθεί στην αρχή της δήλωσης της κλάσης ο ακόλουθος κώδικας:
  - `template<class T>` ή `template<typename T>`
  - Εδώ το T είναι απλά μια παράμετρος που υποδηλώνει έναν τύπο.
  - Όταν θα δημιουργηθούν αντικείμενα της κλάσης, το T αντικαθίσταται με έναν πραγματικό τύπο.
- Για να οριστεί μια συνάρτηση μέλος της κλάσης θα πρέπει να χρησιμοποιηθεί η ακόλουθη σύνταξη:
  - `className<T>::memberName(…){…}`
- Αντίστοιχα, για να δηλωθεί μια μεταβλητή του προτύπου κλάσης θα πρέπει να χρησιμοποιηθεί η ακόλουθη σύνταξη:
  - `className<υπαρκτός τύπος δεδομένων> variable;`

Ένα ακόμα παράδειγμα με πρότυπο κλάσης

- Το πρότυπο MemoryCell μπορεί να χρησιμοποιηθεί με οποιοδήποτε τύπο Object με τις ακόλουθες προϋποθέσεις:
  - Το Object να έχει κατασκευαστή χωρίς παραμέτρους.
  - Το Object να έχει κατασκευαστή αντιγραφής.
  - Το Object να έχει τελεστή αντιγραφής.
- Συμβάσεις
  - Η δήλωση μιας κλάσης προτύπου και ο ορισμός της συνήθως συνδυάζονται στο ίδιο αρχείο.
  - Δεν είναι εύκολο να διαχωριστούν διότι η συγγραφή του κώδικα γίνεται πολύπλοκη.
  - Αυτό αποτελεί διαφορετική σύμβαση σε σχέση με αυτή που ακολουθείται για άλλες κλάσεις και η οποία διαχωρίζει τη δήλωση και την υλοποίηση της κλάσης σε ξεχωριστά αρχεία.

```
template <class Object>
class MemoryCell
{
private:
Object storedValue;

public:
explicit MemoryCell(const Object &initialValue
= Object()) : storedValue(initialValue) {}
const Object &read() const
{
return storedValue;
}
void write(const Object &x)
{
storedValue = x;
}
};
```

Χρήση του προτύπου κλάσης

- Η κλάση MemoryCell μπορεί να χρησιμοποιηθεί για να αποθηκεύει τόσο πρωτογενείς τύπους όσο και τύπους κλάσεων.
- Η MemoryCell δεν είναι κλάση, αλλά είναι ένα πρότυπο κλάσης.
- Κλάσεις είναι οι:
  - MemoryCell<int>
  - MemoryCell<string>

```
#include <string>
#include <iostream>
#include "memorycell.h"

using namespace std;

int main()
{
    MemoryCell<int> m1;
    MemoryCell<string> m2("hello");
    m1.write(37);
    m2.write(m2.read() + " world");
    cout << m1.read() << endl << m2.read() <<
    endl;
}
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/memorycell.h
https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/main_memorycell.cpp

Άσκηση #2: Templated κλάση

- Να υλοποιήσετε μια γενερική κλάση στοίβας (stack) σε C++ χρησιμοποιώντας (templates) και εσωτερική αναπαράσταση με std::vector.
- Η κλάση να υποστηρίζει τις βασικές λειτουργίες μιας στοίβας:
  - push() για εισαγωγή στοιχείου,
  - pop() για αφαίρεση του τελευταίου στοιχείου,
  - top() για ανάγνωση του τελευταίου στοιχείου χωρίς αφαίρεση,
  - και empty() για έλεγχο κατάστασης.
- Η δομή πρέπει να μπορεί να διαχειρίζεται οποιονδήποτε τύπο δεδομένων (π.χ. int, double, std::string).
- Στο κύριο πρόγραμμα (main) να δημιουργήσετε δύο στοίβες διαφορετικών τύπων και να επιδείξετε τη λειτουργία όλων των μεθόδων της κλάσης.

Άσκηση #2: Λύση

```
template <typename T>
class Stack {
  std::vector<T> data;

 public:
  void push(const T& value) { data.push_back(value); }
  void pop() {
    if (!data.empty()) data.pop_back();
  }
  T top() const { return data.back(); }
  bool empty() const { return data.empty(); }
};
```

```
int main() {
  Stack<int> s1;
  s1.push(10);
  s1.push(20);
  std::cout << "Top of int stack: " << s1.top() << "\n";
  s1.pop();
  std::cout << "After pop, top: " << s1.top() << "\n";

  Stack<std::string> s2;
  s2.push("Arta");
  s2.push("Ioannina");
  std::cout << "Top of string stack: " << s2.top() << "\n";
  s2.pop();
  std::cout << "After pop, top: " << s2.top() << "\n";

  return 0;
}
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/oop16_ask2.cpp

```
Top of int stack: 20
After pop, top: 10
Top of string stack: Ioannina
After pop, top: Arta
```

Επίλυση της άσκησης 2 με κλάσεις της βιβλιοθήκης STL

```cpp
#include <iostream>
#include <stack>
#include <string>
#include <vector>

int main() {
  std::stack<int, std::vector<int>> s1;
  s1.push(10);
  s1.push(20);
  std::cout << "Top of int stack: " << s1.top() << "\n";
  s1.pop();
  std::cout << "After pop, top: " << s1.top() << "\n";

  std::stack<std::string, std::vector<std::string>> s2;
  s2.push("Arta");
  s2.push("Ioannina");
  std::cout << "Top of string stack: " << s2.top() << "\n";
  s2.pop();
  std::cout << "After pop, top: " << s2.top() << "\n";

  return 0;
}
```

```
Top of int stack: 20
After pop, top: 10
Top of string stack: Ioannina
After pop, top: Arta
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect16/oop16_ask2b.cpp

Ερωτήσεις σύνοψης

- Τι είναι τα πρότυπα συναρτήσεων (function templates);
- Τι είναι τα πρότυπα κλάσεων (class templates);
- Τι είναι ο γενερικός προγραμματισμός (generic programming);

Απαντήσεις στις ερωτήσεις σύνοψης

- Τι είναι τα πρότυπα συναρτήσεων (function templates);
  - Τα πρότυπα συναρτήσεων είναι ένα μηχανισμός της C++ που επιτρέπει τη συγγραφή μιας συνάρτησης σε γενική μορφή ώστε να λειτουργεί με οποιονδήποτε τύπο δεδομένων. Ο μεταγλωττιστής αναλαμβάνει να παράξει την κατάλληλη υλοποίηση ανάλογα με την κλήση της συνάρτησης.
- Τι είναι τα πρότυπα κλάσεων (class templates);
  - Τα πρότυπα κλάσεων είναι ένα μηχανισμός της C++ μέσω του οποίου μπορούμε να γράφουμε έναν ενιαίο ορισμό κλάσης που είναι παραμετρικός ως προς τους τύπους δεδομένων. Ο μεταγλωττιστής δημιουργεί διαφορετικές εκδόσεις της κλάσης για τους συγκεκριμένους τύπους δεδομένωνκλάσης που χρησιμοποιούνται.
- Τι είναι ο γενερικός προγραμματισμός (generic programming);
  - Γενερικός προγραμματισμός είναι ο προγραμματισμός που στοχεύει στη συγγραφή κώδικα που είναι ανεξάρτητος από συγκεκριμένους τύπους δεδομένων. Η C++ υποστηρίζει τον γενερικό προγραμματισμό μέσω των προτύπων συναρτήσεων (ή συναρτήσεων με πρότυπα) και των προτύπων κλάσεων (ή κλάσεων με πρότυπα). Ο γενερικός προγραμματισμός ενισχύει την επαναχρησιμοποίηση.

Αναφορές

- http://www.cs.fsu.edu/~xyuan/cop3330/