# Templates (πρότυπα)

#16

Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Πανεπιστήμιο Ιωαννίνων (Άρτα)
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.

# 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

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

# 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

# Ερωτήσεις σύνοψης

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

# Αναφορές

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