Προγραμματισμός ΙΙ

Εισαγωγή στην C++

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Η γλώσσα C++

- Σχεδιάστηκε το 1979 από τον Bjarne Stroustrup στα Bell Laboratories
- Βασίζεται στην C προσθέτοντας σύγχρονες δυνατότητες
  1. Αντικειμενοστραφή Προγραμματισμό (Object-Oriented Programming)
  2. Γενικό Προγραμματισμό (Generic Programming)
- Είναι υπερσύνολο της γλώσσας C

Αντικειμενοστραφής Προγραμματισμός

Object-Oriented Programming

- Γενική τεχνική προγραμματισμού
- Οργάνωση γύρω από τα δεδομένα.
- Ορίζουμε τα δεδομένα και τις συναρτήσεις που επιτρέπεται να πράξουν επάνω στα δεδομένα.
- Τρείς βασικές αρχές
  1. Ενθυλάκωση (encaptulation)
  2. Πολυμορφισμός (polymorphism)
  3. Κληρονομικότητα (inheritance)

Ενθυλάκωση

Encaptulation

Μηχανισμός που συνδέει λογικά τον κώδικα και τα δεδομένα που αυτός ο κώδικας διαχειρίζεται.

Διατηρεί και τον κώδικα και τα δεδομένα ασφαλή από εξωτερικούς παράγοντες.

Αυτός ο συνδυασμός είναι ένα ”αντικείμενο”

Ενθυλάκωση
Encaptulation

Μέσα σε ένα αντικείμενο, ο κώδικας, τα δεδομένα ή και τα δύο μπορούν να είναι **ιδιωτικά** (private) στο αντικείμενο ή **δημόσια** (public).

Οτιδήποτε private (δεδομένα ή κώδικας) είναι γνωστό και προσβάσιμο μόνο σε άλλα μέρη του ίδιου αντικειμένου. Δεν μπορούν δηλαδή να χρησιμοποιηθούν από ένα κομμάτι του προγράμματος που υπάρχει εκτός του αντικειμένου.

Όταν έχουν κάτι public (δεδομένα ή κώδικα) η πρόσβαση επιτρέπεται και από κώδικα που βρίσκεται εκτός αντικειμένου. Συνήθως τα public μέρη του αντικειμένου χρησιμοποιούνται για να προσφέρουν ένα μέσο διασύνδεσης του αντικειμένου με τον έξω κόσμο.

Πολυμορφισμός
Polymorphism

Η ιδέα του πολυμορφισμού συνοψίζεται σε μια φράση

”**one interface, multiple methods**”

Επιτρέπει σε μια διασύνδεση (interface) να ελέγχει την πρόσβαση σε μια γενική κλάση από υλοποιήσεις. Η ακριβής λειτουργία καθορίζεται από την φύση της περίπτωσης.

Παράδειγμα Πολυμορφισμού

Ένα πραγματικό παράδειγμα πολυμορφισμού είναι ο θερμοστάτης.

Ανεξάρτητα από την εγκατάσταση της θέρμανσης που έχετε στο σπίτι σας, π.χ πετρέλαιο, φυσικό αέριο, κ.τ.λ, ο θερμοστάτης λειτουργεί με ακριβώς τον ίδιο τρόπο.

Ο θερμοστάτης είναι η διασύνδεση (interface) και η εγκατάσταση είναι η υλοποίηση (implementation).

Εαν θέλουμε 25 βαθμούς, απλά βάζουμε τον θερμοστάτη στους 25 βαθμούς. Δεν μας ενδιαφέρει τι εγκατάσταση έχουμε.

Πολυμορφισμός

Στον προγραμματισμό

Η ίδια λογική με τον θερμοστάτη ισχύει και στον προγραμματισμό.

Έστω πως έχουμε υλοποιήσει 2 στοίβες, η μία χρησιμοποιεί λίστα για την υλοποίηση της ενώ η άλλη έναν πίνακα.

Θα παρέχουμε μια κοινή διασύνδεση μέσω της οποίας θα μπορούμε να διαχειριστούμε και τις δύο στοίβες. Η διασύνδεση αυτή θα παρέχει π.χ τις συναρτήσεις `push` και `pop`.

Κληρονομικότητα
Inheritance

Κληρονομικότητα είναι η διαδικασία όπου ένα αντικείμενο κληρονομεί τις ιδιότητες ενός άλλου αντικειμένου.

Με αυτόν τον τρόπο φτιάχνουμε ιεραρχικές κατηγοριοποιήσεις αντικειμένων.

Για παράδειγμα ένα μήλο ανήκει στην κατηγορία φρούτο που με την σειρά της ανήκει στην κατηγορία φαγητό.

Θα δείτε την κληρονομικότητα εκτεταμένα στα επόμενα μαθήματα προγραμματισμού, οπότε δεν θα δώσουμε πολύ έμφαση. Σημειώστε όμως πως η κληρονομικότητα είναι ένα από τα πιο σημαντικά χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού.

Ένα πρώτο πρόγραμμα

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      int i;
7
8      cout << "output␣something.\n"; // single line comment
9
10     /* you can still use C style comments */
11
12     cout << "Enter␣a␣number:␣";
13
14     // input a number using >>
15     cin >> i;
16
17     // now output a number using <<
18     cout << i << "␣squared␣is␣" << i*i << endl;
19
20     return 0;
21 }
```

Είσοδος/Έξοδος
I/O

Τα header αρχεία της C++ έχουν διαφορετική ονομασία από της C.

```
#include <iostream>
```

Η παραπάνω εντολή χρησιμοποιείται για να μπορούμε να χρησιμοποιήσουμε τις μεθόδους εισόδου/εξόδου της C++.

Είναι ουσιαστικά για την C++ ότι είναι το `stdio.h` για την C.

Namespaces

Η C++ μας επιτρέπει να ορίσουμε ότι κάποια κομμάτια κώδικα ανήκουν σε περιοχές ονομάτων. π.χ

```
namespace myname {

    int temperature;

}
```

Οι περιοχές ονομάτων βοηθούν στην διαχείριση μεγάλων προγραμμάτων και στην χρήση πολλών βιβλιοθηκών ταυτόχρονα.

Για να έχω πρόσβαση στην παραπάνω μεταβλητή πρέπει να γράψω

```
myname::temperature = 5;
```

Namespaces

Με την χρήση των namespaces μπορώ να έχω πολλές μεταβλητές ή και συναρτήσεις με το ίδιο όνομα που ανήκουν όμως σε διαφορετικά namespaces.

Έτσι μπορώ να αποφεύγω τα προβλήματα εαν θέλω να χρησιμοποιήσω π.χ ταυτόχρονα δύο διαφορετικές βιβλιοθήκες φτιαγμένες από διαφορετικούς κατασκευαστές.

Ο κάθε κατασκευαστής θα έχει φροντίσει να συμπεριλάβει όλους τους τύπους και τις μεθόδους του στο δικό του namespace.

Namespaces

Η standard C++ library βρίσκεται όλη στο namespace με όνομα `std`.

Για να χρησιμοποιήσω οτιδήποτε ή θα γράψω π.χ

```
std::cout << "Print this";
```

ή μπορώ στην αρχή του προγράμματος να δώσω

```
using namespace std;
```

και άρα να εξηγήσω στον μεταγλωττιστή πως θέλω να χρησιμοποιώ συνέχεια από εδώ και πέρα το namespace std, χωρίς να το επαναλαμβάνω διαρκώς.

Είσοδος/Έξοδος

```
cout << "output␣something\n";
```

Η παραπάνω γραμμή εκτυπώνει στην οθόνη **output something** και αλλάζει γραμμή.

Στην C++ ο τελεστής « χρησιμοποιείται εκτός από το left-shift και για να στείλουμε δεδομένα στην έξοδο (output operator).

Το `cout` είναι για την C++ ότι είναι για την C το `stdout`.

Μπορείτε να χρησιμοποιήσετε το `cout` και τον τελεστή « για να στείλετε στην τυπική έξοδο όλους τους βασικούς τύπους συμπεριλαμβανομένων και συμβολοσειρών.

Είσοδος/Έξοδος

```
cin >> i;
```

Ο τελεστής » ακόμη σημαίνει right-shift αλλά όταν χρησιμοποιείται όπως παραπάνω τότε είναι ο τελεστής εισόδου (input operator).

Η παραπάνω γραμμή αναθέτει στην μεταβλητή `i` μια τιμή από το πληκτρολόγιο.

Το `cin` είναι για την C++ ότι είναι για την C το `stdin`.

Μπορείτε να χρησιμοποιήσετε το `cin` και τον τελεστή » για να διαβάσετε από την τυπική είσοδο όλους τους βασικούς τύπους συμπεριλαμβανομένων και συμβολοσειρών.

Είσοδος/Έξοδος

Μπορείτε βέβαια να χρησιμοποιήσετε ακόμη τις `printf`() και `scanf`().

Συνήθως όμως στην C++ χρησιμοποιούμε τα `cin` και `cout` μιας και παρέχουν πιο εύκολη πρόσβαση στην είσοδο και στην έξοδο.

```
cout << i << "␣squared␣is␣" << i*i << endl;
```

Μπορούμε για παράδειγμα να γράψουμε στην έξοδο πολλές φορές με την ίδια εντολή. Εδώ `endl` είναι το C++ αντίστοιχο του χαρακτήρα `'\n'`.

Δήλωση Τοπικών Μεταβλητών στην C++

Στην C++ μπορούμε να δηλώσουμε τοπικές μεταβλητές σε οποιοδήποτε σημείο ενός block.

```
#include <iostream>

using namespace std;

int main() {
    cout << "Set␣A␣=␣{␣"

    for (int i = 0; i < 10; i++)
        cout << i << "␣";

    cout << "}" << endl;
}
```

Τύπος `bool` στην C++

Η C++ παρέχει έναν τύπο `bool` ο οποίος μπορεί να πάρει τιμές `true` και `false`.

```
#include <iostream>

using namespace std;

int main() {
    bool n = true;

    if (n) {
        cout << "n is true" << endl;
    }
    else
        cout << "n is false" << endl;
}
```

Αυτόματη μετατροπή από `bool` σε `int` και ανάποδα, γίνεται έτσι ώστε η τιμή `true` να μετατρέπεται σε 1 και η `false` σε 0.

Εισαγωγή στις Κλάσεις

Το πιο σημαντικό στοιχείο της C++ είναι η ύπαρξη των κλάσεων.

Για να φτιάξουμε στην C++ ένα αντικείμενο πρέπει να ορίσουμε την γενική του μορφή χρησιμοποιώντας την δεσμευμένη λέξη `class`.

- Συντακτικά μια κλάση δηλώνεται περίπου όπως και μια εγγραφή (`struct`).
- Εκτός όμως από δεδομένα περιέχει και συναρτήσεις που διαχειρίζονται τα δεδομένα αυτά.

Εισαγωγή στις Κλάσεις

Στοίβα (πρώτη προσπάθεια)

```
1  class stack {
2  private:
3      int array[100];
4      int top;
5
6  public:
7      void init();
8      void push(int i);
9      int pop();
10 };
```

Μια κλάση μπορεί να περιέχει `private` και `public` στοιχεία.

Στην παραπάνω κλάση τα **`array`** και **`top`** είναι προσωπικά που σημαίνει πως δεν μπορούν να προσπελαστούν από κώδικα που δεν είναι μέρος της κλάσης.

Μπορούμε να έχουμε και συναρτήσεις που να είναι `private` και άρα να μπορούν να κληθούν μόνο από άλλες συναρτήσεις της κλάσης.

Εισαγωγή στις Κλάσεις

Στοίβα (πρώτη προσπάθεια)

```
1  class stack {
2  private:
3      int array[100];
4      int top;
5
6  public:
7      void init();
8      void push(int i);
9      int pop();
10 };
```

Μια κλάση μπορεί να περιέχει `private` και `public` στοιχεία.

Οι συναρτήσεις παραπάνω είναι `public` που σημαίνει πως μπορούν να κληθούν από οποιοδήποτε άλλο κώδικα.

Αφού δηλώσουμε μια κλάση μπορούμε να φτιάξουμε αντικείμενα δηλώνοντας μεταβλητές του αντίστοιχου τύπου.

```
int main() {

    stack mystack;

}
```

Το αντικείμενο `mystack` είναι ένα στιγμιότυπο της `stack`.

Η κλάση λοιπόν είναι μια λογική αφαίρεση, ενώ ένα αντικείμενο είναι αληθινό (με την έννοια πως υπάρχει μέσα στην μνήμη του υπολογιστή).

Εκτός βέβαια από την δήλωση της κλάσης πρέπει να δώσουμε και την υλοποίηση των μεθόδων της κλάσης.

```
void stack::init()
{
    top = 0;
}
```

Πρέπει να χρησιμοποιήσουμε τον τελεστή :: (scope resolution operator) που λέει στον μεταγλωττιστή πως αυτή η έκδοση της `init()` ανήκει στην κλάση `stack`.

Για να προσπελάσουμε μια μέθοδο ή δεδομένα μιας κλάσης πρέπει να χρησιμοποιήσουμε (όπως και στις δομές) τον τελεστή τελεία.

```
int main() {
    stack stack1, stack2;

    stack1.init();
}
```

Ο παραπάνω κώδικας φτιάχνει δυο ξεχωριστά αντικείμενα. Στην συνέχεια καλή την συνάρτηση `init()` του αντικειμένου `stack1`. Το αντικείμενο `stack2` δεν αρχικοποιείται.

Εισαγωγή στις Κλάσεις

Στοίβα (πρώτη προσπάθεια)

Μέσα σε μια συνάρτηση μια κλάσης μπορούμε να έχουμε πρόσβαση στις μεταβλητές της κλάσεις χωρίς την χρήση του τελεστή τελεία.

```
void stack::push(int i)
{
    if (top == 100) {
        cout << "Stack is full." << endl;
        return;
    }
    array[top++] = i;
}

int stack::pop()
{
    if (top == 0) {
        cout << "Stack underflow." << endl;
        return 0;
    }
    return array[--top];
}
```

Εισαγωγή στις Κλάσεις

Στοίβα (πρώτη προσπάθεια)

Το παρακάτω κομμάτι κώδικα δεν περνάει από τον μεταγλωττιστή αφού προσπαθούμε να έχουμε πρόσβαση σε `private` μέρη της κλάσης εκτός της κλάσης.

```
int main() {

    stack mystack;

    mystack.init();

    mystack.top = 1;              // COMPILE ERROR!
    mystack.array[0] = 6;         // COMPILE ERROR!

}
```

Τι τυπώνει το παρακάτω κομμάτι κώδικα;

```
int main() {

    stack mystack;

    mystack.init();

    for (int i = 0; i < 10; i++)
        mystack.push(i);

    for (int i = 0; i < 10; i++)
        cout << mystack.pop() << endl;
}
```

Υπερφόρτωση Συναρτήσεων
Function Overloading

Η C++ μας δίνει την δυνατότητα να ορίσουμε πολλές συναρτήσεις με το ίδιο όνομα αρκεί να έχουν διαφορετικά σύνολα παραμέτρων (διαφορετικούς τύπους).

Για να δούμε γιατί είναι χρήσιμη η υπερφόρτωση συναρτήσεων θυμηθείτε τις συναρτήσεις της C

- `int abs(int j)`
- `float fabs(double d)`
- `long labs(long l)`

Παρόλο που οι συναρτήσεις κάνουν ουσιαστικά την ίδια δουλειά, είμαστε υποχρεωμένοι να διατηρούμε 3 ξεχωριστά ονόματα.

Υπερφόρτωση Συναρτήσεων

Function Overloading

Στην C++ μπορούμε να διατηρήσουμε ένα όνομα.

```
1  #include <iostream>
2  using namespace std;
3
4  int abs(int i) {
5      cout << "using integer abs()" << endl;
6      return i < 0 ? -i : i ;
7  }
8
9  double abs(double d) {
10     cout << "using double abs()" << endl;
11     return d < 0.0 ? -d : d ;
12 }
13
14 long abs(long l) {
15     cout << "using long abs()" << endl;
16     return l < 0 ? -l : l ;
17 }
18
19 int main() {
20     cout << abs(-10) << endl;
21     cout << abs(-11.0) << endl;
22     cout << abs(-9L) << endl;
23 }
```

Υπερφόρτωση Συναρτήσεων
Function Overloading

Το πρόγραμμα της προηγούμενης διαφάνειας τυπώνει:

```
using integer abs()
10
using double abs()
11
using long abs()
9
```

Γενικά για να κάνουμε υπερφόρτωση παραμέτρων πρέπει ο τύπος και/ή ο αριθμός των παραμέτρων κάθε υπερφορτωμένης συνάρτησης να διαφέρουν.

Δεν αρκεί να διαφέρουν μόνο στον τύπο επιστροφής.

Υπερφόρτωση Τελεστών
Operator Overloading

Η C++ μας επιτρέπει να υπερφορτώσουμε τελεστές π.χ +, -, κ.τ.λ ώστε να συμπεριφέρονται διαφορετικά ανάλογα με το αντικείμενο.

Για παράδειγμα η χρήση του τελεστή « στην περίπτωση του `cout << "output"` είναι ακριβώς αυτή η περίπτωση.

Επειδή είναι κάπως πολύπλοκο αυτό το θέμα, θα το εξετάσουμε σε λεπτομέρεια αργότερα.

Κληρονομικότητα

Inheritance

Στην C++ η κληρονομικότητα υποστηρίζεται επιτρέποντας σε μια κλάση να εισάγει μια άλλη κλάση μέσα στον ορισμό της. Μπορούμε έτσι να φτιάξουμε μια ιεραρχία κλάσεων, από την πιο γενική προς την πιο συγκεκριμένη.

Πρώτα ορίζουμε την βασική κλάση (base class), που ορίζει τις ιδιότητες εκείνες που είναι κοινές σε όλα τα αντικείμενα που θα κληρονομήσουν την κλάση αυτή.

Μια κλάση που κληρονομεί την βασική κλάση, μπορεί να προσθέσει και ιδιότητες πιο συγκεκριμένες.

Κληρονομικότητα
Inheritance

```
class building {
    int floors;
public:
    int getFloors() { /* ... */ }
    void setFloors(int floors) { /* ... */ }
};

class house : public building {
    int bedrooms;
    int bathrooms;
public:
    /* ... */
};
```

Η κλάση house (derived) έχει πρόσβαση μόνο στα `public` μέλη της κλάσης building. Για να έχει πρόσβαση και σε μέλη που δεν είναι `public` πρέπει στην κλάση building να χρησιμοποιήσουμε την πρόσβαση `protected`.

Η κληρονομικότητα μας επιτρέπει να κάνουμε και δυναμικό πολυμορφισμό, με την χρήση συναρτήσεων που αποκαλούνται `virtual`.

Επειδή στα επόμενα εξάμηνα θα έχετε ένα μάθημα σε Java και ένα μάθημα για αντικειμενοστραφή προγραμματισμό, δεν θα μπούμε ακόμη σε λεπτομέρειες.

Constructors

Είναι συνήθες κάποιο μέρος ενός αντικειμένου να χρειάζεται αρχικοποίηση πριν χρησιμοποιηθεί.

Στην περίπτωση της στοίβας που είδαμε προηγουμένως, είχαμε μια συνάρτηση `init`() την οποία έπρεπε να καλέσουμε αμέσως αφού φτιάχναμε ένα αντικείμενο.

Η C++ μας παρέχει την έννοια του constructor για ακριβώς αυτή την διαδικασία.

Constructors

Στοίβα (δεύτερη προσπάθεια)

Μια συνάρτηση constructor είναι μια ειδική συνάρτηση που είναι μέλος μιας κλάσης και έχει το ίδιο όνομα με την κλάση.

```
class stack {
    private:
        int array[100];
        int top;

    public:
        stack();        // constructor
        void push(int i);
        int pop();
};
```

Προσέξτε πως ο constructor δεν έχει τύπο επιστροφής.

Constructors

Στοίβα (δεύτερη προσπάθεια)

```
class stack {
    private:
        int array[100];
        int top;

    public:
        stack();        // constructor
        void push(int i);
        int pop();
};

stack::stack() {
    top = 0;
    cout << "Stack initialized." << endl;
}
```

Τυπώνουμε το μήνυμα για λόγους εκμάθησης. Συνήθως οι constructors δεν τυπώνουν τίποτα, απλά αρχικοποιούν κατάλληλα.

Constructors

Ο constructor καλείται αυτόματα όταν ένα αντικείμενο δημιουργείται.

Για καθολικά ή στατικά αντικείμενα (μεταβλητές) καλείται μια φορά ενώ για τοπικά αντικείμενα, κάθε φορά που ”εκτελείται” η δήλωση του αντικειμένου.

Constructors

Στοίβα (δεύτερη προσπάθεια)

```
void function() {
    stack T;

    T.push(5);
    cout << T.pop() << endl;
}

int main() {
    stack S;

    S.push(2);
    cout << S.pop() << endl;

    function();
    function();
}
```

Το πρόγραμμα θα εκτυπώσει

```
Stack initialized.
2
Stack initialized.
5
Stack initialized.
5
```

Destructors

Το ακριβώς αντίθετο του constructor είναι ο destructor.

Πολλές φορές ένα αντικείμενο χρειάζεται να πραγματοποιήσει μερικές λειτουργίες πριν καταστραφεί.

Για παράδειγμα το αντικείμενο μπορεί να έχει δεσμεύσει δυναμικά μνήμη από το σύστημα και πρέπει να την επιστρέψει.

Άλλες φορές μπορεί να έχει ανοίξει ένα αρχείο και να πρέπει να το κλείσει.

Destructors

Στοίβα (δεύτερη προσπάθεια)

Ο destructor έχει το ίδιο όνομα με τον constructor με την διαφορά πως ξεκινάει με ένα ~.

```
class stack {
    private:
        int array[100];
        int top;

    public:
        stack();
        ~stack();       // destructor
        void push(int i);
        int pop();
};

stack::~stack() {
    cout << "Stack destroyed." << endl;
}
```

Constructors και Destructors

Στοίβα (δεύτερη προσπάθεια)

```
void function() {
    stack T;

    T.push(5);
    cout << T.pop() << endl;
}

int main() {
    stack S;

    S.push(2);
    cout << S.pop() << endl;

    function();
    function();
};
```

Το πρόγραμμα θα εκτυπώσει

```
Stack initialized.
2
Stack initialized.
5
Stack destroyed.
Stack initialized.
5
Stack destroyed.
Stack destroyed.
```

Συναρτήσεις Inline

Το προσδιοριστικό `inline` μας επιτρέπει να πούμε στον μεταγλωττιστή να δημιουργήσει ένα αντίγραφο του κώδικα μιας συνάρτησης, για να αποφευχθεί μια κλήση.

```
1  #include <iostream>
2
3  inline double cube(const double s) {
4      return s * s * s;
5  }
6
7  int main() {
8      double side;
9
10     std::cout << "Enter the side length of your cube: ";
11     std::cin >> side;
12
13     cout << "Volume of cube = " << cube(side) << std::endl;
14 }
```

Δίνει απλά μια συμβουλή στον μεταγλωττιστή, ο οποίος δεν είναι υποχρεωμένος να την ακολουθήσει.

Αναφορές

Η C++ παρέχει ένα μηχανισμό ώστε να κάνουμε κλήση μέσω αναφοράς χωρίς δείκτες. Επιτρέπει την δημιουργία ψευδώνυμων μεταβλητών.

```
1  int main() {
2      int x = 5;
3
4      std::cout << x << std::endl;
5
6      int& alias = x;    // different name for x
7      alias++;
8
9      std::cout << x << std::endl;
10 }
```

- Μια μεταβλητή αναφοράς πρέπει να αρχικοποιείται πάντα, αλλιώς προκύπτει συντακτικό λάθος.
- Κάθε λειτουργία στην αναφορά γίνεται απευθείας στην μεταβλητή της οποίας η αναφορά είναι ψευδώνυμο.

Κλήση μέσω Αναφοράς

```
1  #include <iostream>
2
3  void square(int &vref) {
4      vref *= vref;
5  }
6
7  int main() {
8      int x = 5;
9
10     square(x);
11     std::cout << x << std::endl;
12 }
```

Αντί για να χρησιμοποιούμε δείκτες μπορούμε να κάνουμε κλήση μέσω αναφοράς με την χρήση αναφορών.

Προεπιλεγμένα Ορίσματα

Η C++ μας επιτρέπει να θέσουμε προεπιλεγμένες τιμές για τα ορίσματα μιας συνάρτησης.

```
1  #include <iostream>
2
3  double log(double x, double base = 2) {
4      // implementation
5  }
6
7  int main() {
8      std::cout << log(1024) << std::endl;
9      std::cout << log(9, 3) << std::endl;
10 }
```

Για να θέσουμε προεπιλεγμένη τιμή σε ένα όρισμα είμαστε υποχρεωμένοι να θέσουμε προεπιλεγμένες τιμές σε όλα τα ορίσματα που ακολουθούν από τα δεξιά.

Πρότυπα Συναρτήσεων

Η C++ μας επιτρέπει μέσω της λέξης κλειδιού `template` να φτιάξουμε συναρτήσεις οι οποίες παραμετροποιούνται με τύπους.

```
1  template<class T>
2  T max(T value1, T value2, T value3)
3  {
4      T max = value1;
5      if (value2 > max)
6          max = value2;
7      if (value3 > max)
8          max = value3;
9      return max;
10 }
11
12 int main() {
13     int c1, c2, c3;
14     std::cin >> c1 >> c2 >> c3;
15     std::cout << max(c1, c2, c3) << std::endl;
16
17     double f1, f2, f3;
18     std::cin >> f1 >> f2 >> f3;
19     std::cout << max(f1, f2, f3) << std::endl;
20 }
```

Πρότυπα Συναρτήσεων

Ο μεταγλωττιστής μόλις συναντήσει την κλήση της συνάρτησης καταλαβαίνει μέσω των παραμέτρων τον τύπο και δημιουργεί την κατάλληλη συνάρτηση

- μπορούμε να έχουμε παραπάνω από μια template παράμετρο
- τα templates είναι μια από τις πιο δυνατές και πιο σημαντικές δυνατότητες της C++ και είναι ο βασικός μηχανισμός του Γενικού Προγραμματισμού (Generic Programming).

Λέξεις κλειδιά της C++

Η C++ εκτός από τις λέξεις κλειδιά της C έχει και πολλές νέες λέξεις κλειδιά.

asm, bool, catch, class, const_cast, delete, dynamic_cast, explicit, false, friend, inline, mutable, namespace, new, operator, private, protected, public, reinterpret_cast, static_cast, template, this, throw, true, try, typeid, typename, using, virtual, wchar_t

Μεταγλωττιστής C++

Τα αρχεία της C++ έχουν συνήθως κατάληξη

- .cpp
- .C

και τα header

- .hpp
- .h

Ο μεταγλωττιστής της GNU για την γλώσσα C++ καλείται με την χρήση της εντολής `g++`.