ΔΩΔΕΚΑΤΗ ΔΙΑΛΕΞΗ

Κατασκευή νέου τύπου (1/2)

Η κατασκευή ενός νέου τύπου περιλαμβάνει συνοπτικά

- τη δήλωση των ποσοτήτων που χρειάζονται για την αποθήκευση των πληροφοριών και της κατάστασης ενός αντικειμένου αυτού του τύπου. Στην απλή περίπτωση οι ποσότητες αυτές είναι κάποιες μεταβλητές και αποτελούν την *εσωτερική αναπαράσταση* του αντικειμένου.
- τη λεπτομερή περιγραφή του τρόπου *δημιουργίας* ενός αντικειμένου. Η δημιουργία του μπορεί να γίνει
  - από ανεξάρτητες ποσότητες που θα δώσουν τιμές στις εσωτερικές μεταβλητές ή
  - από άλλο αντικείμενο της ίδιας κλάσης, με αντιγραφή ή μετακίνηση.
- τη λεπτομερή περιγραφή του τρόπου *καταστροφής* ενός αντικειμένου.
- τη συμπεριφορά του *τελεστή εκχώρησης* ενός αντικειμένου σε άλλο. Η εκχώρηση μπορεί να γίνει με αντιγραφή ή μετακίνηση.
- τον ορισμό συναρτήσεων για την προσπέλαση ή τροποποίηση των μελών της εσωτερικής αναπαράστασης.

Κατασκευή νέου τύπου (2/2)

Πιθανόν να χρειάζεται για νέο τύπο, αν έχουν νόημα, να ορίσουμε

- τελεστές που δρουν σε αντικείμενα του συγκεκριμένου τύπου (αριθμητικοί, σύγκρισης, τελεστής '( )', τελεστής '[ ]', κλπ.) και
- πώς γίνεται η μετατροπή του συγκεκριμένου τύπου σε άλλο.

Constructor αντιγραφής (1/2)

Για τη δημιουργία ενός αντικειμένου τύπου X με αντιγραφή από άλλο, ήδη κατασκευασμένο, X, πρέπει να ορίσουμε τον *κατασκευαστή αντιγραφής* (copy constructor).
Ο copy constructor της κλάσης X καλείται αυτόματα όταν

- δηλώνουμε αντικείμενο τύπου X με αρχική τιμή άλλο τύπου X,
- όταν περνάμε αντικείμενο τύπου X ως όρισμα συνάρτησης τύπου X, που δεν είναι αναφορά.

Αν το όρισμα συνάρτησης είναι αναφορά, δεν γίνεται αντιγραφή.
Επομένως, το όρισμα του copy constructor *πρέπει* να είναι αναφορά (σταθερή ή μη). Αν δεν ήταν, θα έπρεπε να κληθεί ο copy constructor (που δεν έχει γραφεί ακόμα).

Παράδειγμα

Για την κλάση date ο copy constructor είναι

```
date(date const & other)
: d{other.d}, m{other.m}, y{other.y}
{}
```

Constructor αντιγραφής (2/2)

Αν σε μια κλάση X δεν ορίσουμε ρητά κατασκευαστή με αντιγραφή (ούτε κατασκευαστή με μετακίνηση ή τελεστή εκχώρησης με μετακίνηση) ο μεταγλωττιστής, όταν προσπαθήσουμε να δημιουργήσουμε ένα αντικείμενο με αντιγραφή άλλου της ίδιας κλάσης, ορίζει αυτόματα στο τμήμα **public** ένα copy constructor με δήλωση `X(X const & other)` (ή `X(X & other)` ανάλογα με κάποια κριτήρια). Αυτός δημιουργεί ένα αντικείμενο αντιγράφοντας κάθε μέλος από το όρισμα στο αντίστοιχο του νέου αντικειμένου, καλώντας τους copy constructors των μελών με τη σειρά που παρατίθενται στην κλάση.
Στη συνήθη περίπτωση που αρκεί ο default copy constructor, μπορούμε να μην τον ορίσουμε ρητά αλλά να ζητήσουμε από το μεταγλωττιστή να το κάνει γράφοντας στο σώμα της κλάσης X, σε τμήμα **public**:

```
X(X const & other) = default;
```

Destructor

- Μια ποσότητα τύπου X *καταστρέφεται* όταν η ροή του προγράμματος συναντήσει το τέλος της εμβέλειάς της.
- Η διαδικασία καταστροφής περιγράφεται από τον destructor (καταστροφέα). Αυτός καλείται αυτόματα όποτε χρειαστεί, είναι μοναδικός, για μια κλάση X έχει όνομα `~X` και δεν δέχεται ορίσματα. Στο σώμα του γράφουμε εντολές για την "ακύρωση" των αλλαγών που έγιναν από τον constructor.
- Αν τον παραλείψουμε σε μια κλάση X, ο compiler ορίζει αυτόματα τον `~X(){}`.
- Όποτε κληθεί, εκτελεί τις εντολές στο σώμα του και κατόπιν καλεί τους destructors κάθε μέλους της υλοποίησης του αντικειμένου, με *αντίστροφη* σειρά από αυτή που ορίζονται στην κλάση.

Παράδειγμα

Για την κλάση date ο destructor είναι `~date() {}`.

Στη συνήθη περίπτωση που αρκεί ο default destructor, μπορούμε να μην τον ορίσουμε ρητά αλλά να ζητήσουμε από το μεταγλωττιστή να το κάνει γράφοντας στο σώμα της κλάσης X, σε τμήμα **`public`**:

```
~X() = default;
```

Τελεστής εκχώρησης με αντιγραφή (1/3)

Η εκχώρηση ενός αντικειμένου μιας κλάσης X σε άλλο με τον τελεστή '=' προκαλεί την κλήση της συνάρτησης–μέλους **operator**=( ):

$$\text{a = b} \longleftrightarrow \text{a.}\mathbf{operator}\text{=(b).}$$

Η συνάρτηση

- δέχεται μια αναφορά σε σταθερό X,
- δίνει κατάλληλες τιμές στα μέλη του αντικειμένου για το οποίο κλήθηκε (το αριστερό μέλος του =), *καταστρέφοντας τις προηγούμενες*,
- επιστρέφει οτιδήποτε επιθυμούμε, κατά προτίμηση `X &` ώστε να έχει νόημα η εντολή `a=b=c;` αν τα a,b,c είναι τύπου X.

Τελεστής εκχώρησης με αντιγραφή (2/3)

Παράδειγμα

Για την κλάση date ο ορισμός του τελεστή εκχώρησης είναι απλός

```
date & operator=(date const & other)
{
    d = other.d;
    m = other.m;
    y = other.y;

    return *this;
}
```

Παρατήρηση

Μέσα σε οποιαδήποτε συνάρτηση–μέλος, το `*this` είναι το αντικείμενο για το οποίο καλείται αυτή.

Τελεστή εκχώρησης με αντιγραφή (3/3)

Για μια κλάση X, οσοδήποτε πολύπλοκη, ο ακόλουθος, εναλλακτικός ορισμός είναι προτιμότερος καθώς εκμεταλλεύεται τον copy constructor:

```
#include <utility>
using std::swap;

X & operator=(X other)
{
    swap(*this, other);
    return *this;
}
```

Προσέξτε ότι

- το όρισμα περνά με *αντιγραφή*.
- Η εναλλαγή γίνεται με τη συνάρτηση `swap()` που θα έχουμε ορίσει για τον τύπο `X` ή, αν αυτή δεν υπάρχει, με την `std::swap()`.

Άλλοι τελεστές

- Σε μια κλάση συνήθως χρειάζεται να ορίσουμε πώς συμπεριφέρονται τα αντικείμενά της όταν δρουν σε αυτά διάφοροι τελεστές (‘+’, ‘<’, ‘( )’, κλπ.).
- Η συμπεριφορά των αντικειμένων κατά τη δράση του δυικού τελεστή $\otimes$ προσδιορίζεται από τη συνάρτηση–μέλος της κλάσης με όνομα **operator**$\otimes$. Η συνάρτηση αυτή δέχεται ένα όρισμα, το δεξί μέλος του τελεστή $\otimes$. Το αριστερό μέλος είναι το αντικείμενο για το οποίο καλείται:

$$\mathrm{a} \otimes \mathrm{b} \longleftrightarrow \mathrm{a}.\mathbf{operator}\otimes(\mathrm{b}).$$

- Οι συναρτήσεις–μέλη της κλάσης X, που δεν τροποποιούν το αντικείμενο για το οποίο καλούνται, συμπληρώνονται με το **const** μεταξύ της λίστας ορισμάτων και του σώματος:

```
bool operator⊗(X const & other) const
{ ... }
```

Τελεστές σύγκρισης (1/2)

- Μπορούμε να παραγάγουμε τους τελεστές σύγκρισης αν γνωρίζουμε τη δράση των τελεστών '<' και '==':
  - Το `x!=y` ισοδυναμεί με `!(x==y)`,
  - Το `x>y` ισοδυναμεί με `y<x`,
  - Το `x>=y` ισοδυναμεί με `!(x<y)`,
  - Το `x<=y` ισοδυναμεί με `!(y<x)`.
- Η έκφραση `!(a < b) && !(b < a)` προσδιορίζει την ισοδυναμία των a,b (και όχι την ισότητα όπως στα μαθηματικά).
- Σε οποιαδήποτε κλάση, επομένως, οι συναρτήσεις–μέλη **`operator`**`<()` και **`operator`**`==()` αρκούν για να ορίσουμε τις υπόλοιπες των τελεστών σύγκρισης.
- Μπορούμε να αποφύγουμε να προσδιορίσουμε ρητά τις υπόλοιπες συναρτήσεις–μέλη, αν συμπεριλάβουμε τον header `<utility>` και χρησιμοποιήσουμε την εντολή

```
using namespace std::rel_ops;
```

Τελεστές σύγκρισης (2/2)

Παράδειγμα

Για να παρέχουμε τους τελεστές σύγκρισης στην κλάση date γράφουμε

```
#include <utility>

using namespace std::rel_ops;

bool operator==(date const & other) const
{
   return d == other.d && m == other.m && y == other.y;
}
bool operator<(date const & other) const
{
    if (y != other.y) return y < other.y;
    if (m != other.m) return m < other.m;
    return d < other.d;
}
```

Αριθμητικοί τελεστές (1/3)

Οι συναρτήσεις–μέλη ενός ορίσματος **operator**+=( ), **operator**−=( ), **operator**∗=( ), κλπ. προσδιορίζουν τη δράση των αντίστοιχων αριθμητικών τελεστών, αν έχουν νόημα. Ως αριστερό μέλος χρησιμοποιείται το αντικείμενο για το οποίο καλούνται οι συναρτήσεις και ως δεξί μέλος το όρισμα.

Παράδειγμα

Η προώθηση ενός date x κατά k ημέρες μπορεί να γίνει με την εντολή `x+=k;` αρκεί να έχουμε ορίσει σε τμήμα **public**: εντός της κλάσης date τη συνάρτηση

```
void operator+=(int k)
{ ... }
```

Αν έχει οριστεί η δράση του τελεστή '+=', εύκολα ορίζουμε τη δράση του '++' (ή αντίστροφα). Για παράδειγμα στην κλάση date γράφουμε

```
void operator++()
{
   return *this += 1;
}
```

Αριθμητικοί τελεστές (2/3)

Οι συναρτήσεις–μέλη ενός ορίσματος **operator**+(), **operator**−(), κλπ., προσδιορίζουν τι εκτελείται όταν ένα αντικείμενο ακολουθείται από τους τελεστές '+', '-', κλπ. και μια ακόμα ποσότητα. Ο ορισμός τους είναι εύκολος αν μπορούμε να βασιστούμε στους αντίστοιχους τελεστές **operator**+=(), **operator**−=(), κλπ.

Παράδειγμα

Για την κλάση date x έχει νόημα η πρόσθεση ενός ακέραιου k σε μια ημερομηνία. Θεωρούμε ότι θα επιστραφεί η ημερομηνία k ημέρες μετά (ή πριν, αν το k είναι αρνητικό). Η εντολή `date y{x + k};` καλεί τη συνάρτηση

```
date operator+(int k) const
{
   date a{*this};
   a += k;
   return a;
}
```

Αριθμητικοί τελεστές (3/3)

Παράδειγμα

Για την κλάση date έχει νόημα η διαφορά δύο ημερομηνιών. Μπορούμε να θεωρήσουμε ότι είναι το πλήθος των ημερών που μεσολαβούν. Επομένως,

```
int operator-(date const & other) const  // *this-other
{
   int k{0};
   if (other < *this) {
      for (auto x = other; x != *this; ++x) {
         ++k;
      }
   } else {
      for (auto x = other; x != *this; --x) {
         --k;
      }
   }
   return k;
}
```

Αντικείμενο–Συνάρτηση

Σε μια κλάση X μπορούμε να ορίσουμε μία ή περισσότερες συναρτήσεις–μέλη με όνομα **operator**( ) και διαφορετικό πλήθος ή τύπο ορισμάτων. Όποτε ένα αντικείμενο της κλάσης ακολουθείται από τον τελεστή '( )', με τιμές μεταξύ των παρενθέσεων που αντιστοιχούν κατά πλήθος, σειρά και τύπο με τα ορίσματα κάποιας από τις **operator**( ), αυτή καλείται αυτόματα.

Παράδειγμα

Στο εσωτερικό της date μπορούμε να δηλώσουμε:

```
int operator()() const
{
   return y * 10000 + m * 100 + d;
}
```

Οπότε

```
date x{3,12,2005};
std::cout << x() << '\n';  // 20051203
```

Το x είναι αντικείμενο που συμπεριφέρεται ως συνάρτηση.

Άλλες συναρτήσεις–μέλη

Μπορούμε να ορίσουμε όποια συνάρτηση επιθυμούμε ως μέλος μιας κλάσης. Για να την καλέσουμε για κάποιο αντικείμενο, παραθέτουμε το όνομα του αντικειμένου και της συνάρτησης (με τα ορίσματά της) με τελεία '.' ανάμεσά τους.

Παράδειγμα

Για την κλάση date μπορούμε να έχουμε μια συνάρτηση που να επιστρέφει την ημέρα (Δευτέρα, Τρίτη, κλπ.) της ημερομηνίας που αποθηκεύεται σε ένα αντικείμενο:

```
std::string dayname() const { ... }
```

Η κλήση της γίνεται όπως παρακάτω:

```
date x{2,3,2019};
std::cout << x.dayname() << '\n';
```

Συναρτήσεις πρόσβασης

Ιδιαίτερα χρήσιμες είναι οι συναρτήσεις–μέλη που δίνουν ή διαβάζουν τιμές της εσωτερικής αναπαράστασης. Μετά από ετικέτα **public**: γράφουμε, π.χ. για την κλάση date:

```
int day() const { return d; }
int month() const { return m; }
int year() const { return y; }
void set_day(int x) { d = x; }
void set_month(int x) { m = x; }
void set_year(int x) { y = x; }
```

Αυτές μπορούν να χρησιμοποιούν όλες οι συναρτήσεις, **private** ή **public**, ώστε ο κώδικας να είναι ανεξάρτητος από την εσωτερική αναπαράσταση.

Λοιποί τελεστές (1/2)

Υπάρχουν δυικοί τελεστές που το δεξί τους μέλος είναι αντικείμενο της κλάσης X ενώ το αριστερό μέλος είναι αντικείμενο της κλάσης Y, δηλαδή

`Y ⊗ X;`

Δεν μπορούμε να γράψουμε ως μέλη της κλάσης X τις συναρτήσεις που προσδιορίζουν τη δράση τους (δεν δρούν σε αντικείμενα τύπου X) και υποθέτουμε ότι δεν επιτρέπεται να συμπληρώσουμε την κλάση Y. Μπορούμε όμως να τις γράψουμε εκτός και των δύο κλάσεων.

Λοιποί τελεστές (2/2)

Παράδειγμα

Θα θέλαμε να γράψουμε στον κώδικά μας το

```
date a{3,7,2009};
std::cout << a << '\n';
```

ώστε να τυπώνεται το 3/7/2009. Ο τελεστής '<<' έχει στο αριστερό μέλος το `std::cout`, ένα αντικείμενο της κλάσης `std::ostream` του `<ostream>`, ενώ στο δεξί μέλος έχει ένα date. Δεν μπορούμε να συμπληρώσουμε την κλάση `std::ostream` αλλά μπορούμε να γράψουμε εκτός της κλάσης date τη συνάρτηση

```
std::ostream & operator<<(std::ostream & out, date const & x)
{
    out << x.day() << '/' <<  x.month() << '/' << x.year();
    return out;
}
```

Παρατηρήστε τη χρήση των συναρτήσεων πρόσβασης. Είναι απαραίτητες καθώς η **operator**<<() δεν είναι μέλος της κλάσης date (ώστε να έχει απευθείας πρόσβαση στα μέλη private της κλάσης).

Υπόδειγμα (template) κλάσης

Μια κλάση μπορεί να παραμετροποιείται με ένα ή περισσότερους *τύπους* ποσοτήτων ή ακέραιες ποσότητες. Έτσι, αν επιθυμούμε να έχουμε στην κλάση X κάποιο μέλος με όχι συγκεκριμένο τύπο T, δηλαδή, θέλουμε να έχουμε ως παράμετρο της X ένα απροσδιόριστο τύπο T, συμπληρώνουμε τον ορισμό της με το **template<typename T>**:

```
template<typename T>
class X {
  T a;
...
};
```

Στη δήλωση αντικειμένου της κλάσης X πρέπει να προσδιορίσουμε τον τύπο T ως εξής

```
X<double> x1;
X<int>    x2;
```

Οργάνωση κώδικα της κλάσης (1/5)

Για λόγους ευκρίνειας μπορούμε να αποφύγουμε να ορίσουμε μια συνάρτηση–μέλος μιας κλάσης X εντός αυτής. Μέσα στο σώμα της κρατούμε τη δήλωση του μέλους (δηλαδή διαγράφουμε το σώμα του και τη λίστα αρχικοποίησης αν είναι constructor). Εκτός της κλάσης γράφουμε τον πλήρη ορισμό γράφοντας ακριβώς πριν το όνομα της συνάρτησης το `X::`. Την ίδια συμπλήρωση με το `X::` πρέπει να κάνουμε και σε τυχόν τύπους που χρησιμοποιούμε εκτός της κλάσης, οι οποίοι έχουν οριστεί εντός της κλάσης.

Οργάνωση κώδικα της κλάσης (2/5)

Παράδειγμα

Το

```
class date {
   int day() const { return d; }
};
```

μπορεί να γραφτεί

```
class date {
   int day() const;
};
int date::day() const { return d; }
```

Οργάνωση κώδικα της κλάσης (3/5)

Αν η κλάση είναι template, π.χ.

```
template<typename T> class X { ... };
```

τότε ο ορισμός εκτός κλάσης μιας συνάρτησης–μέλους της που δεν είναι template συμπληρώνεται στην αρχή με το

```
template<typename T>
```

και στο όνομα με το `X<T>::`

Οργάνωση κώδικα της κλάσης (4/5)

Παράδειγμα template

Το

```
template<typename T>
class X {
   void f() { ... }
};
```

μπορεί να γραφεί

```
template<typename T>
class X {
   void f();
};

template<typename T>
void X<T>::f()
{ ... }
```

Οργάνωση κώδικα της κλάσης (5/5)

Σε μια κλάση X, που δεν είναι template, οι ορισμοί των συναρτήσεων–μελών της που είναι εκτός αυτής, μπορούν να συγκεντρωθούν σε αρχείο με όνομα *x.cpp*. Η δήλωση της κλάσης μπορεί να γραφτεί στο αρχείο *x.hpp*. Το *x.hpp* πρέπει να γίνει **`#include`** σε κάθε αρχείο που χρειαζόμαστε την κλάση (και στο *x.cpp*). Το αρχείο *x.cpp* πρέπει να μεταγλωττιστεί και να συνδεθεί με τον υπόλοιπο κώδικα κατά τη φάση του linking.

Παράδειγμα

Αν το αρχείο *main.cpp* έχει τον κώδικα που χρειάζεται την κλάση X, πρέπει να έχει την οδηγία

```
#include "x.hpp"
```

Την ίδια οδηγία πρέπει να έχει και το *x.cpp*. Κατά τη μεταγλώττιση γράφουμε

```
g++ main.cpp x.cpp
```