Constructors and Destructors in C++

Σύνθεση

Πολύ συχνά στη C++ μία κλάση μπορεί να περιέχει σαν μέλη-δεδομένα αντικείμενα άλλων κλάσεων. Ποια είναι η σειρά με την οποία δημιουργούνται και καταστρέφονται αυτά τα αντικείμενα; Με ποια σειρά καλούνται οι constructors και οι destructors των κλάσεων;

Πρώτα δημιουργούνται τα αντικείμενα και καλείται ο constructor της εσωτερικής κλάσης και μετά της εξωτερικής.

Αντίθετα, πρώτα καλείται ο destructor της εξωτερικής κλάσης και μετά της εσωτερικής.

Παράδειγμα

```
class Student {
        public:
            Student()
            {
                cout << "Registering a student!" << endl;
            }
            ~Student()
            {
                cout << "Expelling a student :( " << endl;
            }
        };

class Classroom {
        Student   students[5];
        public:
            Classroom()
            {
                cout << "Creating a classroom!" << endl;
            }
            ~Classroom()
            {
                cout << "Destroying a classroom :( " << endl;
            }
        };
```

```
class School {
        Classroom classrooms[2];
        public:
            School()
            {
                cout << "Creating a school!" << endl;
            }
            ~School()
            {
                cout << "Destroying a school :( " << endl;
            }
        };


 int main()
{
        School *a_school =  new School();
        delete a_school;
        return 0;
}
```

Αποτέλεσμα

```
Registering a student!
Registering a student!
Registering a student!
Registering a student!
Registering a student!
Creating a classroom!
Registering a student!
Registering a student!
Registering a student!
Registering a student!
Registering a student!
Creating a classroom!
Creating a school!
Destroying a school :(
Destroying a classroom :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Destroying a classroom :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
```

Δυναμική δέσμευση

Σε περίπτωση που μια κλάση έχει σαν μέλη δεδομένα όχι αντικείμενα κλάσεων αλλά δείκτες σε αντικείμενα κλάσεων και κάνουμε δυναμική δέσμευση η σειρά των εκτυπώσεων θα αλλάξει διότι οι constructors θα κληθούν τότε που γίνεται η δέσμευση.

Έτσι κατά την δημιουργία/καταστροφή της εξωτερικής κλάσης καλείται ο δικός της constructor/destructor, μέσα στον οποίο είναι δική μας ευθύνη να καλέσουμε τον constructor/destructor της εσωτερικής κλάσης με χρήση new/delete.

Παράδειγμα

```
class Student {

        public:
            Student()
            {
                cout << "Registering a student!" << endl;
            }
            ~Student()
            {
                cout << "Expelling a student :( " << endl;
            }
        };

class Classroom {
        Student *students;
        public:
            Classroom()
            {
                students = new Student[5];
                cout << "Creating a classroom!" << endl;
            }
            ~Classroom()
            {
                delete[] students;
                cout << "Destroying a classroom :( " << endl;
            }
        };
```

```
class School {
        Classroom *classrooms;
        public:
            School()
            {
                classrooms = new Classroom[2];
                cout << "Creating a school!" << endl;
            }
            ~School()
            {
                delete[] classrooms;
                cout << "Destroying a school :( " << endl;
            }
        };


int main()
{
        School *a_school =  new School();
        delete a_school;
        return 0;
}
```

Αποτέλεσμα

Registering a student!
Registering a student!
Registering a student!
Registering a student!
Registering a student!
Creating a classroom!
Registering a student!
Registering a student!
Registering a student!
Registering a student!
Registering a student!
Creating a classroom!
Creating a school!
Expelling a student :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Destroying a classroom :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Destroying a classroom :(
Destroying a school :(

Το αποτέλεσμα μπορεί επίσης να εξαρτάται από το σημείο που βάζουμε τις εκτυπώσεις.

Αυτό δεν πρέπει να μας μπερδεύει.

Παράδειγμα

```
class Student {

        public:
            Student()
            {
                cout << "Registering a student!" << endl;
            }
            ~Student()
            {
                cout << "Expelling a student :( " << endl;
            }
        };

class Classroom {
        Student *students;
        public:
            Classroom()
            {
               cout << "Creating a classroom!" << endl;
               students = new Student[5];
            }
            ~Classroom()
            {
               cout << "Destroying a classroom :( " << endl;
               delete[] students;
            }
        };
```

```
class School {
        Classroom *classrooms;
        public:
         School()
         {
                cout << "Creating a school!" << endl;
                classrooms = new Classroom[2];
          }
          ~School()
          {
             cout << "Destroying a school :( " << endl;
             delete[] classrooms;
          }
         };


 int main()
{
                School *a_school =  new School();
                delete a_school;
                return 0;
 }
```

Αποτέλεσμα

Creating a school!
Creating a classroom!
Registering a student!
Registering a student!
Registering a student!
Registering a student!
Registering a student!
Creating a classroom!
Registering a student!
Registering a student!
Registering a student!
Registering a student!
Registering a student!
Destroying a school :(
Destroying a classroom :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Destroying a classroom :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Expelling a student :(
Expelling a student :(

Κληρονομικότητα

Ένας άλλος τρόπος σύνδεσης κλάσεων μεταξύ τους είναι η κληρονομικότητα.

Και σε αυτή την περίπτωση πρώτα δημιουργούνται τα αντικείμενα και καλείται ο constructor της parent/base κλάσης και μετά της child/derived κλάσης.

Αντίθετα, πρώτα καλείται ο destructor της child/derived κλάσης και μετά της parent/base.

Σε περίπτωση που είτε η base είτε η child κλάση έχουν αντικείμενα κλάσεων ως data members (σύνθεση), ισχύει η σειρά που ειπώθηκε προηγουμένως για την σύνθεση.

Με άλλα λόγια:

Construction:

1. Data members of base class
2. Base class
3. Data members of derived class
4. Derived class

Destruction:

1. Derived class
2. Data members of derived class
3. Base class
4. Data members of base class

Παράδειγμα

```
class Driver{
        public:
        Driver()
        {
              cout << "I am a driver!" << endl;
        }
        ~Driver()
        {
               cout << "Driver leaving..." << endl;
        }
        };

class Passenger{
        public:
        Passenger()
        {
              cout << "I am a passenger!" << endl;
        }
        ~Passenger()
        {
               cout << "Passenger leaving..." << endl;
        }
        };

int main()
{
      Car *car_p =  new Car();
      delete car_p;
      return 0;
}
```

```
class Vehicle{
        Passenger  passenger;
        public:
        Vehicle()
        {
              cout << "I am a vehicle!" << endl;
        }
        ~Vehicle()
        {
              cout << "Vehicle destroyed!" << endl;
        }
        };

class Car: public Vehicle{
        Driver  driver;
        public:
        Car()
        {
            cout << "I am a car!" << endl;
        }
        ~Car()
        {
            cout << "Car destroyed!" << endl;
        }
        };
```

Αποτέλεσμα

I am a passenger!
I am a vehicle!
I am a driver!
I am a car!
Car destroyed!
Driver leaving…
Vehicle destroyed!
Passenger leaving…

Ορίσματα by value

Όταν μία συνάρτηση παίρνει ως όρισμα ένα αντικείμενο και όχι αναφορά ή δείκτη σε αντικείμενο, τότε με την κλήση της συνάρτησης αυτόματα καλείται ο copy-constructor της κλάσης του αντικειμένου για να αρχικοποιήσει το όρισμα. Δηλαδή, δημιουργείται ένα αντίγραφο του αντικειμένου που δόθηκε ως όρισμα.

Μέσα στη συνάρτηση, αντί για το ίδιο το αντικείμενο που δόθηκε ως όρισμα, χρησιμοποιείται το αντίγραφο που δημιουργήθηκε. Το αντίγραφο είναι δηλαδή μία τοπική μεταβλητή και καταστρέφεται μόλις βγει από την εμβέλεια της συνάρτησης.

Return

Όταν μία συνάρτηση επιστρέφει (return) ένα αντικείμενο, δημιουργείται (πάλι με χρήση copy constructor) και επιστρέφεται ένα αντίγραφο του αντικειμένου.

Αυτό το αντίγραφο είναι ένα προσωρινό αντικείμενο, το οποίο καταστρέφεται, αφού περαστούν τα δεδομένα του στην μεταβλητή που χρησιμοποιούμε για να αποθηκεύσουμε ό,τι μας επέστρεψε η συνάρτηση. Δηλαδή:

```
A  function()
{
   A a;
   ……
   return a;
}

A a1;
a1 = function();

A a2 = function();
```

Στο διπλανό παράδειγμα, κατά την επιστροφή από τη συνάρτηση καλείται ο copy constructor που δημιουργεί ένα αντίγραφο του a, το οποίο επιστρέφει η συνάρτηση function.

Στη συνέχεια καλείται assignment operator που αναθέτει τη τιμή του αντίγραφου στη μεταβλητή a1. Τέλος καλείται ο destructor και καταστρέφει το προσωρινό αντίγραφο του a που δημιουργήθηκε με τη return.

Στη περίπτωση που το επιστρεφόμενο αντικείμενο χρησιμοποιείται για δημιουργία **νέου** αντικειμένου (a2) με copy constructor, συνήθως καλείται μόνο ένας copy constructor (return value optimization). Διαφορετικά θα έπρεπε να κληθεί ένας copy constructor με τη return, ένας με την δημιουργία του a2, και τέλος ο destructor για να καταστρέψει το προσωρινό αντίγραφο του a που δημιουργήθηκε με τη return.

Παράδειγμα

```
class A {
        public:
            A()
            {
               cout << "Constructor!" << endl;
            }
            A(const A& a)
            {
               cout << "Copy constructor!" << endl;
            }
            A& operator=(const A& a)
            {
               cout << "Assignment operator!" << endl;
               return *this;
            }
            ~A()
            {
               cout << "Destructor!" << endl;
            }
       };
```

```
A   function(A &  a)
{
     return a;
}

void   function2(A  a)
{
      cout << "Created a local variable!" << endl;
}

int main()
{
   A a;
   A a1;
   cout << "Copy constructor, assignment operator,
destructor!" << endl;
   a1 = function(a);
   cout << "Only one copy constructor this time!" << endl;
   A a2 = function(a);
   cout << "Passing argument by value!" << endl;
   function2(a);
   cout << "End of example!" << endl;
   return 0;
}
```

Αποτέλεσμα

```
Constructor!
Constructor!
Copy constructor, assignment operator, destructor!
Copy constructor!
Assignment operator!
Destructor!
Only one copy constructor this time!
Copy constructor!
Passing argument by value!
Copy constructor!
Created a local variable!
Destructor!
End of example!
Destructor!
Destructor!
Destructor!
```