

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΜΕ C++

Προετοιμασία για το εργαστήριο 2

Structs, κλάσεις και αντικείμενα, δημόσια και ιδιωτική πρόσβαση, κατασκευαστές, συναρτήσεις μέλη και φίλες συναρτήσεις

`struct vs class`

`private: / public:`

`A(int x) : x(x) {}`

`c.area()`

`friend void fun(A&)`

STRUCTS

Ένα struct ομαδοποιεί δεδομένα, οι συναρτήσεις μένουν έξω

```
struct circle {  
    double center_x, center_y;  
    double radius;  
};  
  
double area(const circle &c) {  
    return M_PI * c.radius * c.radius;  
}  
  
circle c;  
c.center_x = 0.0;  c.center_y = 0.0;  
c.radius = 10.0;  
cout << area(c);    // 314.159
```

Πεδία με τελεία

c.radius: κάθε μεταβλητή circle έχει τα δικά της center_x, center_y, radius.

Πέρασμα με const αναφορά

const circle &c: χωρίς αντίγραφο, και η συνάρτηση δεν μπορεί να αλλάξει τον κύκλο.

Αρχικοποίηση με άγκιστρα

circle c{0.0, 0.0, 10.0}; τα πεδία παίρνουν τιμές με τη σειρά που δηλώθηκαν.

Όλα είναι ανοιχτά: τίποτα δεν εμποδίζει τη main να γράψει c.radius = -5;

ΚΛΑΣΕΙΣ ΚΑΙ ΑΝΤΙΚΕΙΜΕΝΑ

Η κλάση είναι το καλούπι, τα αντικείμενα τα αντίτυπα

```
class circle {  
private:  
    double center_x, center_y;  
    double radius;  
public:  
    circle();  
    circle(double x, double y, double r);  
    double area() const;  
};
```

a

```
center_x 0.0  
center_y 0.0  
radius 1.0
```

b

```
center_x 0.0  
center_y 0.0  
radius 10.0
```

c

```
center_x 7.2  
center_y 3.9  
radius 16.2
```

Τρία αντικείμενα, ίδια κλάση, δικά τους δεδομένα.

class circle

Ο τύπος: ποια δεδομένα και ποιες λειτουργίες έχει κάθε κύκλος.

circle a;

Ένα αντικείμενο: μεταβλητή τύπου circle με δική της μνήμη.

a.area()

Κλήση συνάρτησης μέλους πάνω σε συγκεκριμένο αντικείμενο.

ΑΠΟ ΤΟ STRUCT ΣΤΗΝ ΚΛΑΣΗ

Από συνάρτηση με όρισμα σε συνάρτηση μέλος

STRUCT · ΕΛΕΥΘΕΡΗ ΣΥΝΑΡΤΗΣΗ

double area(const circle &c)

- Ο κύκλος περνά ως όρισμα: area(c)
- Μέσα στη συνάρτηση γράφουμε c.radius
- Τα πεδία είναι public: όποιος έχει το c μπορεί να τα αλλάξει
- Δεδομένα και λειτουργίες είναι χωριστά

CLASS · ΣΥΝΑΡΤΗΣΗ ΜΕΛΟΣ

double area() const

- Το αντικείμενο μπαίνει πριν την τελεία: c.area()
- Μέσα στη συνάρτηση γράφουμε σκέτο radius
- Τα δεδομένα είναι private: αλλάζουν μόνο μέσω μελών
- Δεδομένα και λειτουργίες ζουν μαζί (ενθυλάκωση)

struct και **class** διαφέρουν μόνο στην προκαθορισμένη πρόσβαση: struct → public, class → private.

Το `private` κλειδώνει τα μέλη έξω από την κλάση

`private:`

- Προσβάσιμα μόνο από συναρτήσεις μέλη και φίλες συναρτήσεις
- Το `obj.x` στη `main` δίνει σφάλμα μεταγλώττισης
- Προστατεύουν την κατάσταση, π.χ. καμία αρνητική ακτίνα
- Προεπιλογή όταν γράφουμε `class`

`public:`

- Προσβάσιμα από οποιονδήποτε έχει το αντικείμενο
- Το `obj.y` στη `main` μεταγλωττίζεται κανονικά
- Εδώ μπαίνει η διεπαφή: κατασκευαστές, συναρτήσεις μέλη
- Προεπιλογή όταν γράφουμε `struct`

Το `private` απαγορεύει στον κώδικα έξω από την κλάση να τα αγγίξει.

Το y διαβάζεται απευθείας, το x μόνο μέσω getter

```
class A {  
private:  
    int x;  
public:  
    int y;  
    A(int x, int y) : x(x), y(y) {}  
};  
int main() {  
    A obj(1, 2);  
    cout << obj.y;    // εντάξει  
    cout << obj.x;    // σφάλμα  
}
```

Μεταγλώττιση

error: 'int A::x' is private within this context

```
class A {  
private:  
    int x;  
public:  
    int y;  
    A(int x, int y) : x(x), y(y) {}  
    int get_x() { return x; }  
};  
int main() {  
    A obj(1, 2);  
    cout << obj.get_x() << endl;  
    cout << obj.y << endl;  
}
```

Εμφανίζει **1 2**

μία τιμή ανά γραμμή· η get_x() είναι μέλος, άρα βλέπει το x

ΚΑΤΑΣΚΕΥΑΣΤΕΣ

Τέσσερα πράγματα για τους κατασκευαστές

Ίδιο όνομα με την κλάση

Χωρίς τύπο επιστροφής, ούτε καν void: `circle(double c_x, double c_y, double r)`

Καλείται αυτόματα

Μία φορά, τη στιγμή που δημιουργείται το αντικείμενο. Δεν ξανακαλείται σε αντικείμενο που ήδη υπάρχει.

Μπορούν να είναι πολλοί

Υπερφόρτωση: `circle()` για προεπιλεγμένο κύκλο, `circle(x, y, r)` για συγκεκριμένο. Διαλέγουν τα ορίσματα.

Προκαθορισμένες τιμές

`A(int x, int y = 42)`: το `A(3)` δίνει `y = 42`. Οι προκαθορισμένες παράμετροι μπαίνουν στο τέλος.

Χωρίς κανέναν κατασκευαστή, ο compiler φτιάχνει έναν κενό `A()`. Μόλις γράψουμε οποιονδήποτε, ο αυτόματος δεν δημιουργείται.

ΚΑΤΑΣΚΕΥΑΣΤΕΣ

Λίστα αρχικοποίησης ή αναθέσεις στο σώμα

```
circle() : center_x(0.0),  
          center_y(0.0),  
          radius(1.0)  
{  
    cout << "created";  
}
```

```
circle()  
{  
    center_x = 0.0;  
    center_y = 0.0;  
    radius = 1.0;  
    cout << "created";  
}
```

: center_x(0.0)

Λίστα αρχικοποίησης: τα μέλη παίρνουν τιμή καθώς δημιουργούνται, πριν από το σώμα.

radius = 1.0;

Ανάθεση στο σώμα: το μέλος υπάρχει ήδη και απλώς αλλάζει τιμή. Εδώ, ίδιο αποτέλεσμα.

A(int x) : x(x)

Έξω από την παρένθεση το μέλος, μέσα η παράμετρος. Στο σώμα θα γράφαμε `this->x = x;`

Ποιος κατασκευαστής καλείται για κάθε αντικείμενο;

```
class A {  
    int x, y;  
public:  
    A() : x(0), y(0) {}  
    A(int x, int y = 42) : x(x), y(y) {}  
};  
int main() {  
    A obj1;  
    A x[] = {A(), A(1), A(2),  
             A(3), A(4, 50)};  
}
```

Έκφραση	Κατασκευαστής	x y
obj1	A()	0 0
A()	A()	0 0
A(1)	A(int, int), y = 42	1 42
A(2)	A(int, int), y = 42	2 42
A(3)	A(int, int), y = 42	3 42
A(4, 50)	A(int, int)	4 50

Το όρισμα που λείπει παίρνει την προκαθορισμένη τιμή 42· το A(4, 50) δίνει ρητά y = 50.

Δύο σφάλματα κατά τη δημιουργία αντικειμένων

Κλήση κατασκευαστή σε υπάρχον αντικείμενο

```
circle c;  
c.circle(0.0, 0.0, 10.0); // σφάλμα
```

Ο κατασκευαστής τρέχει μόνο κατά τη δημιουργία· δεν είναι συνάρτηση μέλος που καλούμε.

Σωστά: `c = circle(0.0, 0.0, 10.0);`

Λείπει ο προκαθορισμένος κατασκευαστής

```
class B { int v;  
public: B(int v) : v(v) {} };  
B b; // σφάλμα
```

Υπάρχει μόνο ο `B(int)`, οπότε ο αυτόματος `B()` δεν δημιουργείται.

Σωστά: `B b(5);` ή προσθήκη `B() : v(0) {}`

Και το `B arr[5];` θέλει `B()`: κάθε στοιχείο του πίνακα κατασκευάζεται με τον προκαθορισμένο κατασκευαστή.

Κάθε αντικείμενο καταστρέφεται όταν τελειώνει η ζωή του

```
int main() {  
    {  
        circle a;  
    }  
    circle c(0.0, 0.0, 10.0);  
    cout << c.area() << endl;  
    cout << (circle(7.2, 3.9, 16.2)  
        .is_point_inside(8.0, 19.01)  
        ? "YES" : "NO") << endl;  
}
```

Σημείο	Τυπώνει
circle a;	a ... created
} τέλος μπλοκ	a ... destroyed
circle c(...)	c ... created
c.area()	314.159
circle(7.2, ...)	προσωρινό ... created
.is_point_inside(...)	YES
; τέλος έκφρασης	προσωρινό ... destroyed
} τέλος main	c ... destroyed

Ο ~circle() δεν παίρνει παραμέτρους και δεν τον καλούμε ποτέ εμείς για αντικείμενα της στοίβας.

ΚΑΤΑΣΤΡΟΦΕΑΣ

Με new και delete αποφασίζουμε εμείς πότε

```
class A {  
    int x[100];  
public:  
    ~A() {  
        cout << this << " size=" <<  
            << sizeof(*this) << endl;  
    }  
};  
A *p[10];  
for (int i = 0; i < 10; i++) p[i] = new A;  
for (int i = 0; i < 10; i++) delete p[i];
```

Εμφανίζει **0x... size=400**

δέκα φορές, μία για κάθε delete

Ένας μόνο καταστροφέας

~A(): χωρίς παραμέτρους και χωρίς τύπο επιστροφής, άρα δεν υπερφορτώνεται.

this και sizeof(*this)

this: η διεύθυνση του αντικειμένου. *this: το ίδιο το αντικείμενο, $100 \times 4 = 400$ byte.

Το delete καλεί τον ~A()

Χωρίς delete ο καταστροφέας δεν τρέχει ποτέ και η μνήμη χάνεται (memory leak).

ΣΥΝΑΡΤΗΣΕΙΣ ΜΕΛΗ

Οι συναρτήσεις μέλη δουλεύουν πάνω στο αντικείμενο

```
class circle {
    double center_x, center_y, radius;
public:
    double area() const {
        return M_PI * radius * radius;
    }
    bool is_point_inside(double x,
                        double y) const {
        double dx = x - center_x;
        double dy = y - center_y;
        return dx*dx + dy*dy ≤ radius*radius;
    }
};
```

Κλήση με τελεία

c.area(): το c είναι το αντικείμενο που την καλεί

Κρυφό this

radius μέσα στην area $\equiv$ this->radius $\equiv$ c.radius

const στο τέλος

Δεν αλλάζει μέλη· μια ανάθεση εκεί είναι σφάλμα

Βλέπουν τα private

Πρόσβαση σε όλα τα μέλη της κλάσης τους

Μέσα ή έξω από τον κύκλο; Συγκρίνουμε τετράγωνα

```
circle c(0.0, 0.0, 10.0);  
c.is_point_inside(0.0, 9.99);  
c.is_point_inside(0.0, 10.01);  
  
circle(7.2, 3.9, 16.2)  
    .is_point_inside(8.0, 19.01);
```

Σημείο	$dx^2 + dy^2$	r^2	Αποτέλεσμα
(0, 9.99) στο c	99.80	100	YES
(0, 10.01) στο c	100.20	100	NO
(8, 19.01) στο προσωρινό	228.95	262.44	YES

Εμφανίζει

YES NO YES

Τρίτη γραμμή: $dx = 0.8$, $dy = 15.11$, άρα $0.64 + 228.31 = 228.95 \leq 262.44$.

Μια private συνάρτηση μέλος καλείται μόνο από άλλα μέλη

```
class A {  
private:  
    void fun(int a) {  
        cout << a << endl;  
    }  
};  
int main() {  
    A obj;  
    obj.fun(5);    // σφάλμα  
}
```

Η fun υπάρχει, αλλά η main δεν έχει δικαίωμα να την καλέσει.

```
class A {  
private:  
    void fun(int a) { cout << a << endl; }  
public:  
    void delegate_to_fun(int a) { fun(a); }  
};  
int main() {  
    A obj;  
    obj.delegate_to_fun(5);  
}
```

Εμφανίζει

5

η delegate_to_fun είναι μέλος, άρα καλεί τη fun

ΦΙΛΕΣ ΣΥΝΑΡΤΗΣΕΙΣ

Η φίλη συνάρτηση δεν είναι μέλος, αλλά βλέπει τα `private`

ΣΥΝΑΡΤΗΣΗ ΜΕΛΟΣ

`int get_x()`

- Καλείται πάνω σε αντικείμενο: `obj.get_x()`
- Έχει `this`: βλέπει τα μέλη χωρίς όρισμα
- Ορισμός εκτός κλάσης με `A::get_x()`
- Πρόσβαση στα `private`: ναι

ΦΙΛΗ ΣΥΝΑΡΤΗΣΗ

`friend void fun(A &a)`

- Καλείται σαν απλή συνάρτηση: `fun(obj)`
- Δεν έχει `this`: το αντικείμενο έρχεται ως παράμετρος
- Ορισμός χωρίς `friend` και χωρίς `A::`
- Πρόσβαση στα `private`: ναι, γράφοντας `a.x`

Τη φιλία τη δίνει η κλάση: μια συνάρτηση δεν μπορεί να δηλώσει μόνη της ότι είναι φίλη.

Δύο φίλες συναρτήσεις από τις ασκήσεις

```
class A {  
    int x;  
public:  
    A(int x) : x(x) {}  
    friend void fun(A &a);  
};  
void fun(A &a) {  
    cout << a.x << endl;  
}  
int main() {  
    A obj(42);  
    fun(obj);  
}
```

Εμφανίζει

42

η fun διαβάζει το private x μέσω της παραμέτρου a

```
class A {  
    int x, y;  
public:  
    A(int x, int y = 42) : x(x), y(y) {}  
    friend ostream &operator<<(  
        ostream &os, const A &obj) {  
        os << obj.x << " " << obj.y;  
        return os;  
    }  
};  
A obj(7);  
cout << obj << endl;
```

Εμφανίζει

7 42cout << obj $\equiv$ operator<<(cout, obj)

ΣΥΝΟΨΗ

Ποιος έχει πρόσβαση σε τι

Κώδικας	public μέλη	private μέλη	Παράδειγμα
Συνάρτηση μέλος	Ναι	Ναι	c.area() διαβάζει το radius
Φίλη συνάρτηση	Ναι	Ναι	fun(obj) διαβάζει το a.x
main ή άλλη συνάρτηση	Ναι	Όχι: σφάλμα	obj.y εντάξει, obj.x σφάλμα

struct

Προεπιλογή public. Κατάλληλο για απλές ομάδες δεδομένων.

class

Προεπιλογή private. Δεδομένα κρυφά, διεπαφή public.

A() / ~A()

Κατασκευή και καταστροφή: καλούνται αυτόματα, μία φορά.

Στο εργαστήριο: ξεκινήστε από το struct_circle_example.cpp και μετατρέψτε το σε class circle με private δεδομένα, δύο κατασκευαστές, καταστροφή και συναρτήσεις μέλη.