# Friends – Overloading operators

# Friend function

# Class Point

- Στο προηγούμενο μάθημα δημιουργήσαμε μια κλάση Point που περιγράφει σημεία στον δισδιάστατο χώρο
- Η δήλωση της κλάσης είναι η διπλανή

```
class Point
{
private:
    int x;
    int y;
public:
    Point();
    Point(int a, int b);
    ~Point();
    void set_x(int a);
    void set_y(int b);
    int get_x();
    int get_y();
};
```

# Συνάρτηση equal

- Ας πούμε ότι θέλουμε να δημιουργήσουμε μία συνάρτηση equal
- Αυτή θα παίρνει σαν ορίσματα δύο Point objects
- Και θα επιστρέφει true αν τα σημεία αυτά έχουν την ίδια x και y συντεταγμένη
- Ο κώδικας φαίνεται δίπλα

```
bool equal(const Point &a,const Point &b) {
    bool tmp;

    tmp = (a.get_x() == b.get_x()) &&
          (a.get_y() == b.get_y());

    return tmp;
}
```

# Objects call by reference

- Γιατί τα ορίσματα στην equal περνάν by reference και μάλιστα const;
- Ένα από τα πλεονεκτήματα της παραπάνω προσέγγισης είναι ότι μπορούμε να δώσουμε ορίσματα const Point

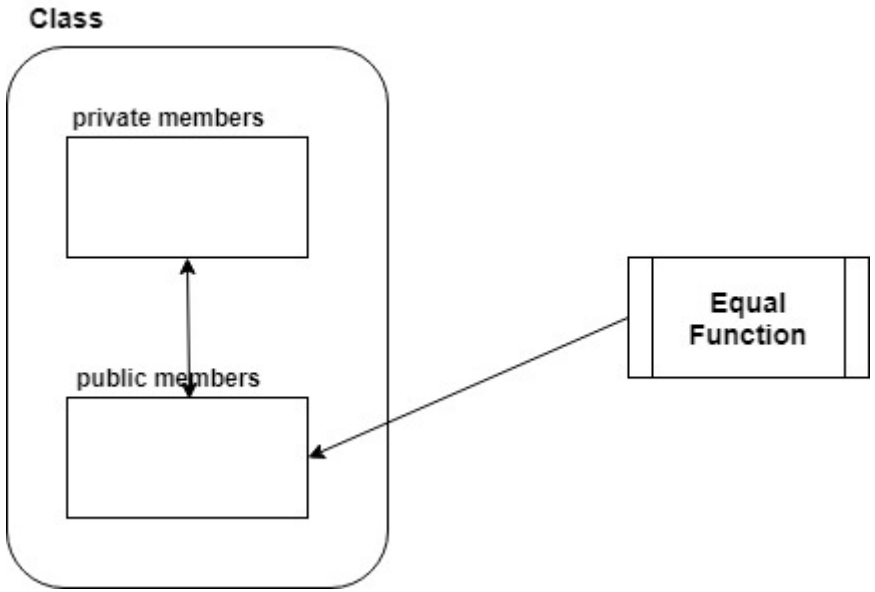
```
void set_y(int b);
int get_x() const;
int get_y() const;
```

```
int Point::get_x() const
{
    return x;
}
```

```
int Point::get_y() const
{
    return y;
}
```

# Πρόβλημα με equal

- Το πρόβλημα με την function equal είναι ότι δεν μπορεί να έχει απευθείας πρόσβαση στα private members της κλάσης
- Αποτέλεσμα είναι να χρησιμοποιεί τα public members (accessors & mutators) και ο κώδικας της να είναι αρκετά μεγάλος



# Η λύση της friend function

- Λύση στο πρόβλημα του πολύπλοκου κώδικα που προκύπτει για την συνάρτηση equal μπορεί να δώσει η ***friend function***
- Η friend function είναι μια συνάρτηση που μπορεί να έχει απευθείας πρόσβαση στα private τμήματα μιας κλάσης
- ΠΡΟΣΟΧΗ! **Η friend function ΔΕΝ ΕΊΝΑΙ method!!!!**
- Επομένως καλείται σαν κανονική function και όχι μέσω του αντικειμένου

```
Point a{ 3,4 }, b{ 4,5 };
bool tmp;

tmp = equal(a, b);

cout << "Ta dyo simeia einai idia (0 gia oxi):" << tmp<<endl;
```

# Friend function

- Η friend function είναι μια function (και όχι method) που δηλώνεται μέσα στην κλάση με το πρόθεμα friend
- Ο ορισμός (κώδικας) της friend
  - Περιέχεται στο αρχείο με τις μεθόδους της κλάσης
  - ΔΕΝ έχει το scope resolution (class::method) ΓΙΑΤΙ ΔΕΝ ΕΊΝΑΙ METHOD

```
    int get_x() const;
    int get_y() const;
    friend bool equal(const Point &a, const Point &b);
};
```

```
bool equal(const Point & a, const Point & b)
{
    bool tmp;

    tmp = (a.x == b.x) &&
        (a.y == b.y);

    return tmp;
}
```

# Equal (εναλλακτική με member function)

- Μπορούμε αντί για friend function να δημιουργήσουμε μια method επομένως αυτή θα έχει πρόσβαση στα private members της function
- ΠΡΟΣΟΧΗ! Θα έχει ΜΟΝΟ ΈΝΑ όρισμα
- Ο ορισμός θα έχει το scope resolution
- Η κλήση θα γίνεται με τον τελεστή (.)

```
bool mequal(const Point &a);
friend bool equal(const Point &a, const Point &b);
```

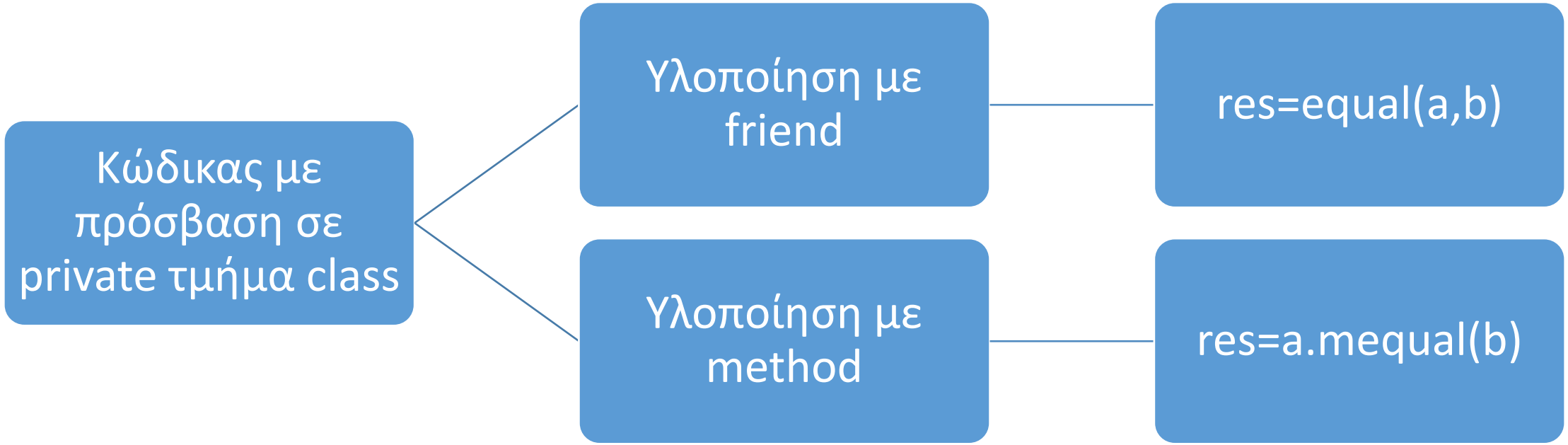
```
bool Point::mequal(const Point & a)
{
    bool tmp;

    tmp = (x == a.x) && (y == a.y);
    return tmp;
}
```

```
Point a{ 3,4 }, b{ 4,5 };
bool tmp;

tmp = a.mequal(b);
```

# Κώδικας με πρόσβαση στο private



# Friend ή method

- Όταν θέλουμε να έχουμε εύκολη πρόσβαση στο private μέρος μιας κλάσης μπορούμε να χρησιμοποιήσουμε ***friend*** ή ***method***
- Αν η συνάρτηση περιλαμβάνει :
  - ΈΝΑ object τότε είναι προτιμότερη η λύση της method
  - ΔΥΟ ή περισσότερα τότε είναι προτιμότερη η λύση της fiend function
- Η λύση της friend function είναι γενικά προτιμώμενη γιατί παράγει πιο απλό κώδικα. Ότι γίνεται με friend μπορεί να γίνει και με non-friend αρκεί να υπάρχουν οι συναρτήσεις που δίνουν πρόσβαση στα private τμήματα που χρειαζόμαστε

# Operator overloading

# Operators και objects

- Οι τελεστές έχουν ξεκάθαρη ερμηνεία για τους βασικούς τύπους δεδομένων
- Στην C++ μπορούμε να δηλώσουμε και την ερμηνεία των τελεστών και για τους καινούργιους τύπους που δημιουργούμε (κλάσεις)
- **Αυτό που κάνουμε είναι «υπερφορτώνουμε» (overload) τους υπάρχοντες τελεστές για τα καινούργια classes**

```
Point a{ 3,4 }, b{ 4,5 };
int x = 2, y = 4;

x = y + x;
a = a + b;
```

# Σχέση τελεστή και συνάρτησης

- Στην πραγματικότητα ένας τελεστής είναι μία συνάρτηση με δύο ορίσματα
- Για παράδειγμα το

***3+4***

- Μπορεί να γραφεί σαν

***add(3,4)***

- Επομένως όπως μπορούμε να «υπερφορτώσουμε» μια συνάρτηση έτσι μπορούμε να υπερφορτώσουμε και έναν τελεστή
- Το όνομα της συνάρτησης είναι

***operator+***

# Overloading operator ==

Class declaration

```
    void set_x(int a);
    void set_y(int b);
    int get_x() const;
    int get_y() const;
    friend bool operator ==(const Point &a, const Point &b);
};
```

Overloaded operator definition

```
bool operator==(const Point & a, const Point & b)
{
    bool tmp;

    tmp = (a.x == b.x) &&
        (a.y == b.y);

    return tmp;
}
```

main

```
int main()
{
    Point a{ 3,4 }, b{ 4,5 };

    bool tmp;

    tmp = (a==b);
```

# Κανόνες για την υπερφόρτωση των τελεστών

- Τουλάχιστον ένα όρισμα του τελεστή που υπερφορτώνουμε πρέπει να είναι τύπου κλάσης
- Μπορεί να είναι friend function μιας κλάσης
- ΔΕΝ μπορούμε να δημιουργήσουμε νέους τελεστές
- Ο αριθμός των ορισμάτων για έναν τελεστή ΔΕΝ μπορεί να αλλάξει
- Η προτεραιότητα των τελεστών ΔΕΝ μπορεί να αλλάξει
- ΔΕΝ μπορούμε να υπερφορτώσουμε τους τελεστές (.) , (::), (*) και (?)

# Αυτόματη μετατροπή τύπων

- Αν στην κλάση υπάρχει constructor με ένα όρισμα μπορεί να χρησιμοποιηθεί από τον compiler για την αυτόματη μετατροπή τύπου
- Για παράδειγμα

```
tmp = (a == 3);
```

Θα δουλέψει αν:

- Υπερφορτώσουμε το == για Point και int
- Δημιουργήσουμε constructor από int σε point

```
public:
    Point();
    Point(int a);
    Point(int a, int b);
```

```
Point::Point(int a): x{a}, y{0}
{
}
```

# Περισσότερες υπερφορτώσεις τελεστών

# Υπερφόρτωση τελεστών

- ΠΡΟΣΟΧΗ στην υπερφόρτωση τελεστών. Πρέπει να είναι ξεκάθαρο το νόημα των τελεστών για την κλάση σας
  - Αν η κλάση σας χρειάζεται IO τότε ορίστε τους τελεστές << και >>
  - Αν η κλάση σας χρειάζεται σύγκριση μεταξύ των αντικειμένων τότε ορίστε και τον τελεστή ==
  - Αν στα αντικείμενα της κλάσης σας ορίζεται μια ξεκάθαρη σχέση σύγκρισης ορίστε τον τελεστή < (μπορούν να οριστούν και όλοι οι υπόλοιποι σχεσιακοί)
  - Ο τύπος επιστροφής του υπερφορτωμένου τελεστή θα πρέπει να είναι ίδιος με αυτό του αρχικού τελεστή (σχεσιακοί bool, αριθμητικοί κλάση, ανάθεση αναφορά στο αριστερό μέλος)

# Επιλογή friend vs method

- Για την υπερφόρτωση ενός τελεστή έχουμε να διαλέξουμε ανάμεσα σε friend και method. Μερικές οδηγίες
  - Οι τελεστές (=, [], (), ->) πρέπει να οριστούν σαν methods
  - Οι τελεστές compound assignment μπορούν να οριστούν και με τους δύο τρόπους καλό είναι όμως να οριστούν σαν members
  - Οι τελεστές που αλλάζουν την τιμή του αντικειμένου πρέπει να οριστούν σαν members (++, --, *).
  - Συμμετρικοί τελεστές που έχουν δύο ορίσματα συνήθως ορίζονται σαν friends (αριθμητικοί, σχεσιακοί, ισότητας)
  - Οι τελεστές εισόδου εξόδου ορίζονται σαν friends

# Τελεστές I/O

# Υπερφόρτωση τελεστών εισόδου εξόδου

- Η υπερφόρτωση του τελεστή εξόδου << γίνεται όπως και στους άλλους τελεστές μόνο που πρέπει να δούμε ποια ακριβώς είναι τα ορίσματα και τι επιστρέφει
- Για παράδειγμα

***cout<<"Hello world!";***

# Υπερφόρτωση τελεστών εισόδου εξόδου

- Ο τύπος που επιστρέφει ο τελεστής εισόδου εξόδου μπορεί να γίνει κατανοητός αν δούμε μια περίπτωση όπου έχουμε παραπάνω από ένα ορίσματα
- Για παράδειγμα

***cout<<“var is”<<a;***

- Γράφεται στην ουσία

***((cout<<“var is”)<<a)***

- Επομένως ο τελεστής << επιστρέφει **output stream**

# Υπερφόρτωση τελεστών εισόδου εξόδου

```
friend ostream& operator<<(ostream& out, const Point& rh);
friend istream& operator>>(istream& in, Point& rhs);
```

- Αυτό που πρέπει να προσέξουμε είναι ότι τα stream πρέπει να περνάνε και να επιστρέφονται ΠΑΝΤΑ με αναφορά

```
ostream& operator<<(ostream& out, const Point& rh)
{
    out << "x:" << rh.x << " y:" << rh.y;
    return out;
}

istream& operator>>(istream& in, Point& rhs)
{
    double a, b;
    in >> a >> b;
    if (in) {
        rhs.x = a;
        rhs.y = b;
    }
    else {
        cout << "Error in reading Point\n";
    }
    return in;
}
```

# Παρατηρήσεις στην υπερφόρτωση IO

- Υλοποιούνται με friend functions
- Επιστρέφουν stream
- Όταν χρησιμοποιούμε stream για arguments ή τιμή επιστροφής τα χρησιμοποιούμε by reference (όχι const)
- Ο τελεστής << πρέπει να κάνει ελάχιστο format
- Πρέπει να γίνεται έλεγχος για την ορθή ανάγνωση στον τελεστή >>

# Αριθμητικοί και σχεσιακοί τελεστές

# Αριθμητικοί

- Υλοποιούνται σαν friend functions
- Επιστρέφουν τιμή τύπου class (όχι reference)
- Χρησιμοποιούν το compound assignment

```
Point operator+(const Point& lhs, const Point& rhs)
{
    Point sum = lhs;
    sum.x += rhs.x;
    sum.y += rhs.y;
    return sum;
}
```

# Σύγκρισης

- Συνήθως η ισότητα ελέγχει αν τα data members είναι ίσα
- Πρέπει να ισχύει η αντιμεταθετική και η προσετεριστική ιδιότητα
- Για την ανισότητα μπορεί να χρησιμοποιηθεί η ισότητα

```
bool operator==(const Point& lhs, const Point& rhs)
{
    if (lhs.x == rhs.x && lhs.y == rhs.y)
        return true;
    else
        return false;
}

bool operator!=(const Point& lhs, const Point& rhs)
{
    if (lhs == rhs)
        return false;
    else
        return true;
}
```

# Σχεσιακοί

- Ο τελεστής < χρησιμοποιείται από τα assosiative containers και από μερικούς γενικούς αλγόριθμους (π.χ. Sort)
- Να υπάρχει συνέπεια (π.χ. αν είναι != πρέπει ένα να είναι < από το άλλο)
- Γενικά αν υπάρχει ο == τότε ο < να ορίζεται μόνο αν υπάρχει συνέπεια μεταξύ τους

```
bool operator<(const Point& lhs, const Point& rhs)
{
    double dlhs, drhs;
    dlhs = sqrt(lhs.x * lhs.x + lhs.y * lhs.y);
    drhs = sqrt(rhs.x * rhs.x + rhs.y * rhs.y);

    if (dlhs < drhs)
        return true;
    else
        return false;
}
```

# Διάφοροι τελεστές

# Assignment operator

- Δημιουργείται αυτόματα από τον compiler μπορεί όμως να οριστεί και διαφορετική
- Έχει νόημα στις κλάσεις που περιέχουν δυναμικές μεταβλητές μέλη
- Υλοποιείται σαν μέθοδος
- Επιστρέφει reference στο αριστερό τελεστή

```
Point& Point::operator=(const Point& rhs)
{
    x = rhs.x;
    y = rhs.y;
    return *this;
}
```

# Subscript

- Επιστρέφει reference στο στοιχείο που ζητείτε
- Μπορεί να χρησιμοποιηθεί και στο αριστερό σκέλος μιας ανάθεσης

```
double& Point::operator[](size_t i)
{
    if (i == 1)
        return x;
    else if (i == 2)
        return y;
    else {
        cout << "out of bounds error\n";
        exit(1);
    }
}
```

# Υπερφόρτωση increment operator

- Οι τελεστές με ένα όρισμα μπορούν να υπερφορτωθούν και αυτοί
- Ο increment (και ο decrement) υλοποιούνται σαν μέθοδοι
- Υπάρχει η prefix και η postfix έκδοση

```
Point& Point::operator++()
{
    ++x;
    ++y;
    return *this;
}

Point Point::operator++(int)
{
    Point ret = *this;
    ++* this;
    return ret;
}
```