UNIVERSITÀ PARTHENOPE
DiST
Dipartimento di Scienze e Tecnologie

Programmazione II e Lab di PII

Overload
e
membri friend

Angelo Ciaramella

Inizializzazione

```
struct X {
        X(int);
};

X x0; //errore : nessun inizializzatore
X x1 {}; // errore : inizializzatore vuoto
X x2 {2}; // OK
```

Uso di costruttori

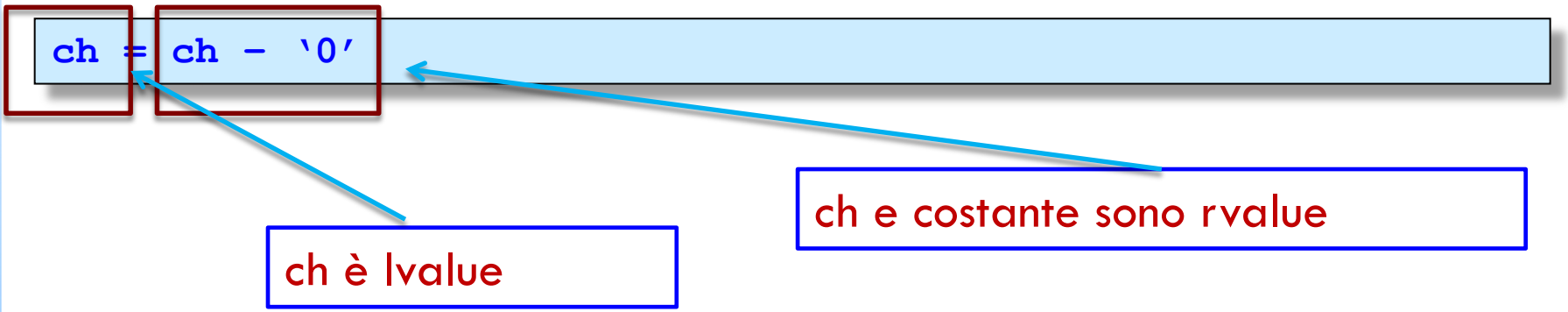
```
class Vector {
public:
        Vector();
};

Vector v1; //OK
Vector v2 {}; // OK
```

Il costruttore predefinito scompare quando definiamo un costruttore con argomenti

rvalue e lvalue

- rvalue
  - valore memorizzato in qualche indirizzo di memoria
  - “right value” o “read value”
- lvalue
  - indirizzo di memoria che contiene il valore memorizzato
  - “left value” o “location value”



Riferimento lvalue

- X&
  - "riferimento a X"
  - usato per riferimento a lvalue

```
void f() {
  int var = 1;
  int& r {var}; // r e var si riferiscono allo stesso int
  int x = r;    // x diventa 1

  r = 2;        // var diventa 2
};
```


Riferimento rvalue

- X&&
  - “riferimento rvalue”
  - si riferisce ad un oggetto temporaneo modificabile

```
string f(string&& s) {
  if (s.size())
        s[0] = toupper(s[0]);
  return s;
};
```


Copia

- Copia per una classe X
  - x = y
    - i valori di x e y sono entrambi uguali al valore di y prima dell'assegnamento
- Costruttore di copia
  - `X(const X&)`
- Assegnamento di copia
  - `X& operator=(const X&)`

Copia

```
class Point2D {
        // . . .
        Point2D(const Point2D& other);
};
```

```
Geometry::Point2D::Point2D( const Point2D& other)
{
    x = other.X();
    y = other.Y();

    return *this;
}
```

Esempio di costruttore di copia

Copia

```
class Point2D {
        // . . .
        Point2D& operator=(const Point2D& other);
};
```

```
Geometry::Point2D&   Geometry::Point2D::operator=   (   const
Point2D& other)
{
    x = other.x;
    y = other.y;

    return *this;
}
```

Esempio di assegnamento di copia

Spostamento

- Spostamento per una classe X
  - consente di riconoscere a tempo di compilazione se un oggetto è temporaneo o no
- Costruttore di spostamento
  - `X(X&&)`
- Assegnamento di spostamento
  - `X& operator=(X&&)`

Spostamento

```
namespace Geometry {
    class Point2D;
}
```

```
class Geometry::Point2D
{
public:
   Point2D(); // costruttore di default
   Point2D(double xValue, double yValue);
   // costruttore sovraccaricato
   Point2D(const Point2D& other);
  // costruttore di copia
   Point2D(Point2D&& other);
  // costruttore di spostamento (overload di copia)
   Point2D& operator=(Point2D&& other);
// operatore di assegnamento di spostamento
    double X();
    void setX(double value);

    double Y();
    void setY(double value);
```


Esempio di costruttore di spostamento e assegnazione di spostamento

Spostamento

```
    std::string Label();
    void setLabel(std::string value);

    double distanceFrom(Point2D other);

private:
    double x;
    double y;

    std::string label;
};
```

```
Geometry::Point2D::Point2D(Point2D&& other) :
    x(other.x),
    y(other.y),
    label(std::move(other.label))
{
    // ripristino di other
    other.x = 0;
    other.y = 0;
    other.label = "";
}
```

converte il suo unico argomento in un rvalue reference

Default

- Per dafault una classe fornisce
  - costruttore predefinito
    - `X()`
  - costruttore di copia
    - `X(const X&)`
  - assegnamento di copia
    - `X& operator=(const X&)`
  - costruttore di spostamento
    - `X(X&&)`
  - assegnamento di spostamento
    - `X& operator=(X&&)`
  - distruttore
    - `~X()`

- Soppresse se il programmatore implementa una delle operazioni

Overload di operatori

- Adattare le operazioni ad oggetti

+	-	*	/	%	^	&
|	~	!	=	<	>	+=
-=	*=	/=	%=	^=	&=	|=
<<	>>	>>=	<<=	==	!=	<=
>=	&&	||	++	--	->*	,
->	[]	()	new	new[]	delete	delete[]

Operatori che possono essere definiti dal programmatore

::	sizeof	
.	alignof	?:
.*	typeid	

Operatori che non possono essere definiti dal programmatore

Overload di operatori

```
class complex {
double re, im;
public:
complex(double r, double i) :re{r}, im{i} { }
complex operator+(complex);
complex operator*(complex);
};
```

Semplice classe complex

```
void f()
{
  complex a = complex{1,3.1};
  complex b {1.2, 2};
  complex c {b};
  a = b+c;
  b = b+c*a;
  c = a*b+complex(1,2);
}
```

Esempio di costruttore di spostamento e assegnazione di spostamento

Overload di operatori

```
void f(complex a, complex b)
{
        complex c = a + b; // abbreviato
        complex d = a.operator+(b); // chiamata esplicita
}
```

Esempio di uso degli operatori (inizializzatori sinonimi)

```
class X {
public:
// ...
void operator=(const X&) = delete;
void operator&() = delete;
void operator,(const X&) = delete;
// ...
};
```

Eliminazione del significato predefinito degli operatori

Overload di operatori

```
Matrix operator+(const Matrix& a, const Matrix& b)
// ritorno per valore
{
    Matrix res {a};
return res+=b;
}
```

Passaggio per riferimento e ritorno per valore

```
Matrix& Matrix::operator+=(const Matrix& a)
// ritorno per riferimento
{
if (dim[0]!=a.dim[0] || dim[1]!=a.dim[1])
throw std::exception("bad Matrix += argument");
  double* p = elem;
  double* q = a.elem;
  double* end = p+dim[0]*dim[1];
while(p!=end)
  *p++ += *q++
return *this;
}
```

Passaggio per riferimento e ritorno per riferimento

Aritmetica mista

```
class complex {
  double re, im;
public:
complex& operator+=(complex a)
{
 re += a.re;
 im += a.im;
return *this;
}
complex& operator+=(double a)
{
re += a;
return *this;
}
// ...
};
```

Esempio di aritmetica mista. Le tre varianti di operator+ possono essere definite nel Namespace

Conversioni

```
class complex {
 double re, im;
public:
  complex(double r) :re{r}, im{0} { }
// costruisce un complex da double
// ...
};
```

Esempio di costruttore di conversione

Conversioni di tipi

- Conversioni di tipi
  - Costruttore che accetta un singolo argomento
  - Operatore di conversione
    - `X::operatorT()` (conversione da X a T)

Overload di operatori speciali

- Operatori che non riguardano operazioni aritmetiche o logiche
  - `[]  ()  ->  ++  --  new  delate`

Membri friend

- Membri friend
  - i membri privati della classe visibili anche a funzioni esterne alla classe
  - queste funzioni sono dette amiche (friend) della classe

```
class suono {

  friend void filtro (const suono&);

...

};  // filtro può accedere agli attributi di suono
```

Esempio di membro dichiarato friend