La programmazione *Object Oriented* e le sue applicazioni in Fisica.

- Gabriella Cataldi, INFN Lecce
- Daniele Martello, dip. Fisica e INFN Lecce

Operatori

- E' possibile ridefinire **+**, **-**, *****, **[]**, **++**, **==**, . . .

Vector3d.h

```
class Vector3d
{
public:
  Vector3d(double x, double y, double z);
  double x() const;
  double y() const;
  double z() const;
  double r() const;
  double phi() const;
  double theta() const;
  Vector3d operator+(const Vector3d& v);
  Vector3d operator-(const Vector3d& v);
private:
  double _x, _y, _z;
};
```

Vector3d.cc

```
Vector3d Vector3d::operator+(const Vector3d& v)
{
  return Vector3d(_x + v._x, _y + v._y, _z + v._z);
}

Vector3d Vector3d::operator-(const Vector3d& v)
{
  return Vector3d(_x - v._x, _y - v._y, _z - v._z);
}
```

Operatori

- Esempio:

main.cc

```
#include <iostream.h>
#include "Vector3d.h"

int main()
{
  Vector3d v1(1, 0, 0), v2(0, 1, 0);
  Vector3d v;

  v = v1 + v2;

  cout << " v = " << v << endl;
  cout << " r = " << v.r();
  cout << " theta = " << v.theta() << endl;
}
```

ridefinizione di <<

Output:

```
v = (1, 1, 0)
r = 1.4141 theta = 1.5708
```

Operatori

- Esempio:

main.cc

```
#include <iostream.h>
#include <cmath>
#include "Vector3d.h"
#include "Matrix.h" // matrice 3x3
int main()
{
  Vector3d v1(1, 1, 0);

  double phi = M_PI/3;
  double c = cos(phi), s = sin(phi);

  Matrix m(1,  0, 0,
           0,  c, s,
           0, -s, c);

  Vector3d u = m * v1;
}
```

π

Operatore di Matrix

Overloading di operatori

- Possono esistere funzioni con lo stesso nome, ma con argomenti diversi

Vector3d.h

```
class Vector3d
{
public:
  ...
  Vector3d operator*(double s) const;
  double operator*(const Vector3d& v) const;

private:
  double _x, _y, _z;
};
```

prodotto scalare

prodotto per uno scalare

Vector3d.cc

```
Vector3d Vector3d::operator*(double s) const
{
  return Vector3d(_x *s, _y *s, _z *s);
}

double Vector3d::operator*(const Vector3d& v)
const
{
  return (_x * v._x + _y * v._y + _z * v._z);
}
```

Che succede se nel main.cc inserisco il prodotto di un ``double per un Vector3d''

Overloading di operatori

Vector3d.h

```
class Vector3d
{
public:
  ...
  friend Vector3d operator*(double s, const Vector3d& v);
  friend Vector3d operator*(const Vector3d& v, double s);

private:
  double _x, _y, _z;
};
```

Vector3d.cc

```
Vector3d operator*(double s , const Vector3d& v)
{
  return Vector3d(s*v._x,s*v._y,s*v._z);
}

Vector3d operator*(const Vector3d& v,double s)
{
  return operator*(s , v) ;
}
```

Overloading di operatori

Esercizio1

```
// binary operator
   ErrorPoint operator+(const ErrorPoint & E);
   ErrorPoint operator-(const ErrorPoint & E);
   ErrorPoint operator*(const ErrorPoint & E);
   ErrorPoint operator/(const ErrorPoint & E);
```

Esercizio2

```
// unary operator
   void operator+=(const ErrorPoint & E);
   void operator-=(const ErrorPoint & E);
   void operator*=(const ErrorPoint & E);
   void operator/=(const ErrorPoint & E);
```

Overloading di operatori

Vector3d.h

```
// output operator
friend ostream & operator<<(ostream & os, const Vector3d & v) ;
```

Vector3d.cxx

```
// output operator
ostream & operator<<(ostream & os, const Vector3d & v) {
   os<<"( "<<v._x<<" ,"<<v._y<<" ,"<<v._z<<" )";
   return os;
};
```

Overloading di operatori

Vector3d.h

```
// input operator
friend istream & operator>>(istream & is, Vector3d & v) ;
```

Vector3d.cxx

```
// input operator
istream & operator>>(istream & is, Vector3d & v) {
   is>>v._x>>v._y>>v._z;
   return is;
};
```

Overloading di operatori

ErrorPoint.h

```
// increment operator ....
// just for exercise. This does not make sense for  ErrorPoint
   friend const ErrorPoint & operator++(ErrorPoint & E);  //prefix
   friend ErrorPoint operator++(ErrorPoint & E , int);       //postfix
```

ErrorPoint.cxx

```
const ErrorPoint & operator++(ErrorPoint & E) {
   ++(E._value);
   return E;    };
ErrorPoint operator++(ErrorPoint & E , int) {
   ErrorPoint tmp(E);
   ++(E._value);
   return tmp;      };
```