Namespace

Nella programmazione modulare, problema dell'*inquinamento dello spazio dei nomi*.

```
// file Complex.h

struct Complex {
  ... // implementazione1
};

double module(Complex);
```

```
// qualche altro file
#include "Complex.h"

// dichiarazione ILLEGALE
struct Complex {
 ... // implementazione2
}

void f(...) {
// vorrebbe usare
// entrambi i Complex
}
```

```
// file Lib_UNO.h

namespace SPAZIO_UNO {
  struct Complex {...};
  void f(Complex c) {...};
};
```

```
// file Lib_DUE.h

namespace SPAZIO_DUE {
  struct Complex {...};
  void g(Complex c) {...};
};
```

Operatore di scoping

`namespace::nome_dichiarazione;`

```
#include "Lib_UNO.h"
#include "Lib_DUE.h"

void funzione() {
  SPAZIO_UNO::Complex var1; SPAZIO_UNO::f(var1);
  SPAZIO_DUE::Complex var2; SPAZIO_DUE::g(var2);
  SPAZIO_DUE::g(var1); // ERRORE IN COMPILAZIONE !!
}
```

Alias di namespace

```
#include "Lib_UNO.h"
#include "Lib_DUE.h"

namespace UNO = SPAZIO_UNO;
namespace DUE = SPAZIO_DUE;

void funzione() {
  UNO::Complex var1; UNO::f(var1);
  DUE::Complex var2; DUE::g(var2);
}
```

Il meccanismo dei **namespace** permette di incapsulare dei nomi che altrimenti inquinerebbero il namespace globale

```
// file Definizioni.h

namespace ProgOgg {
  struct Complex {
    ...
  };

  double module(Complex);

  ... // altri membri del namespace ProgOgg
}
```

```
// file Lib_UNO.h

namespace SPAZIO_UNO {
  struct Complex {...};
  void f(Complex c) {...};
};
```

```
// file Lib_DUE.h

namespace SPAZIO_DUE {
  struct Complex {...};
  void g(Complex c) {...};
};
```

Operatore di scoping

`namespace::nome_dichiarazione;`

```
#include "Lib_UNO.h"
#include "Lib_DUE.h"

void funzione() {
  SPAZIO_UNO::Complex var1; SPAZIO_UNO::f(var1);
  SPAZIO_DUE::Complex var2; SPAZIO_DUE::g(var2);
  SPAZIO_DUE::g(var1); // ERRORE IN COMPILAZIONE !!
}
```

Dichiarazione d'uso rende visibile in un namespace una singola dichiarazione.

```
#include "Lib_UNO.h"
#include "Lib_DUE.h"

using SPAZIO_UNO::Complex;
using SPAZIO_DUE::g;

void funzione() {
  Complex var1; SPAZIO_UNO::f(var1);
  SPAZIO_DUE::Complex var2; g(var2);
}
```

Direttiva d'uso

`using namespace`

```
#include "Lib_UNO.h"
//rende visibili tutti i nomi del namespace SPAZIO_UNO
using namespace SPAZIO_UNO;

void funzione() {
  Complex var1; f(var1);
  SPAZIO_DUE::Complex var2; SPAZIO_DUE::g(var2);
}
```

Le componenti di tutte le librerie del C++ standard sono dichiarate in un namespace chiamato **`std`**.

Nel C++ standard (ad esempio il compilatore **`g++`**), il seguente codice non compila.

```
#include <iostream>

int main() {
  cout << "Ciao!" << endl;
}
// COMPILATORE g++:
// `cout' undeclared
// `endl' undeclared
```

```cpp
#include <iostream>

using namespace std;

int main() {
  cout << "Ciao!" << endl;
}
// Compila correttamente con g++
```

Alternativamente usando l'operatore di scoping:

```cpp
#include <iostream>

int main() {
  std::cout << "Ciao!" << std::endl;
}
// Compila correttamente con g++
```

Una direttiva d'uso è considerata generalmente una scelta discutibile per rendere visibili i nomi dichiarati nel namespace **`std`**. Preferibile una dichiarazione d'uso.

```
#include <iostream>

using std::cout;
using std::endl;

int main() {
  cout << "Ciao!" << endl;
}
```

Compilatore

La versione del compilatore **`g++`** disponibile su **`ssh.studenti.math.unipd.it`**: 7.5.0

L'ultima versione di **`g++`** è la 10.2 (July 2020)

Il tipo `string`

Il tipo `string` è **un esempio di classe** definita nella libreria standard STL.

```
// stringhe nello stile C
char * st = "ecco una stringa/0";
char * st = "ecco una stringa"; // equivalente!
```

Esempio

```
#include<iostream>

using namespace std;

int main() {
  char * s1 = "pippo\0";
  for(int i=0; *(s1+i); i++) cout << *(s1+i); cout << endl;
  // stampa: pippo
  char * s2 = "pippo";
  for(int i=0; *(s2+i); i++) cout << *(s2+i); cout << endl;
  // stampa: pippo
}
```

`string` è una classe definita nella libreria STL. Piu' precisamente, **`string`** è una istanziazione di un *template di classe*:

```
typedef basic_string<char> string
```

```
#include <string>
using namespace std;
int main() {
 char * s = "ecco una stringa\0";   // stile C
 string st("ecco una stringa");     // stile C++
 string st = "ecco una stringa";    // stile C++
}
```

```
cout << "La lunghezza di '" << st
     << "' è di " << st.size()
     << " caratteri;"
     << " l'eventuale carattere nullo terminante"
<< " non è contato";
```

st è un *oggetto* della classe **string** e **size()** è un *metodo* della classe **string**.

Dichiarazioni di una stringa:

```
string st1;        // costruttore: dichiara una stringa e la
                   // inizializza come vuota
string st2(st);    // costruttore di copia:
                   // st2 è una copia di st
```

Tipi di dato astratti

Tipo di dato astratto

Da Wikipedia, l'enciclopedia libera.

Un **tipo di dato astratto** o **ADT** (**A**bstract **D**ata **T**ype), in informatica e specificamente nel campo della programmazione, è un tipo di dato le cui istanze possono essere manipolate con modalità che dipendono esclusivamente dalla semantica del dato e non dalla sua implementazione.

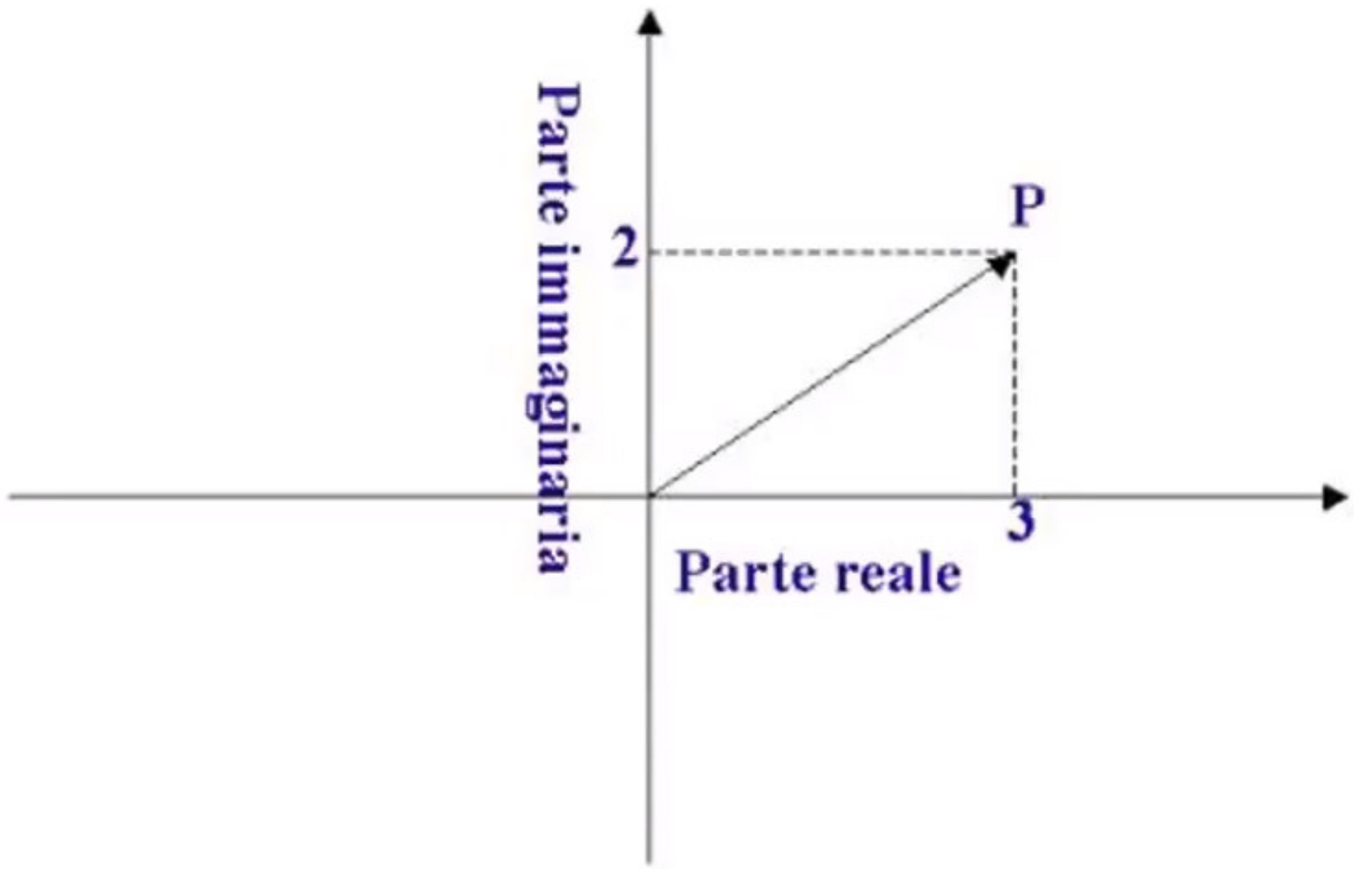
Definizione e caratteristiche [modifica | modifica wikitesto]

Nei linguaggi di programmazione che consentono la *programmazione per tipi di dati astratti*, un tipo di dati viene definito distinguendo nettamente la sua interfaccia, ovvero le operazioni che vengono fornite per la manipolazione del dato, e la sua implementazione interna, ovvero il modo in cui le informazioni di stato sono conservate e in cui le operazioni manipolano tali informazioni al fine di esibire, all'interfaccia, il comportamento desiderato. La conseguente inaccessibilità dell'implementazione viene spesso identificata con l'espressione incapsulamento (detto anche *information hiding*: *nascondere informazioni*).

Tipi di dato astratti

- Concetto di **Abstract Data Type** (ADT) = Valori + operazioni
- **Interfaccia** dell'ADT: metodi pubblici o operazioni proprie
- Rappresentazione interna dell'ADT **inaccessibile**
- Esempio di ADT: tipo primitivo **`int`**
- **`struct`** del C++ non rispetta il concetto di ADT

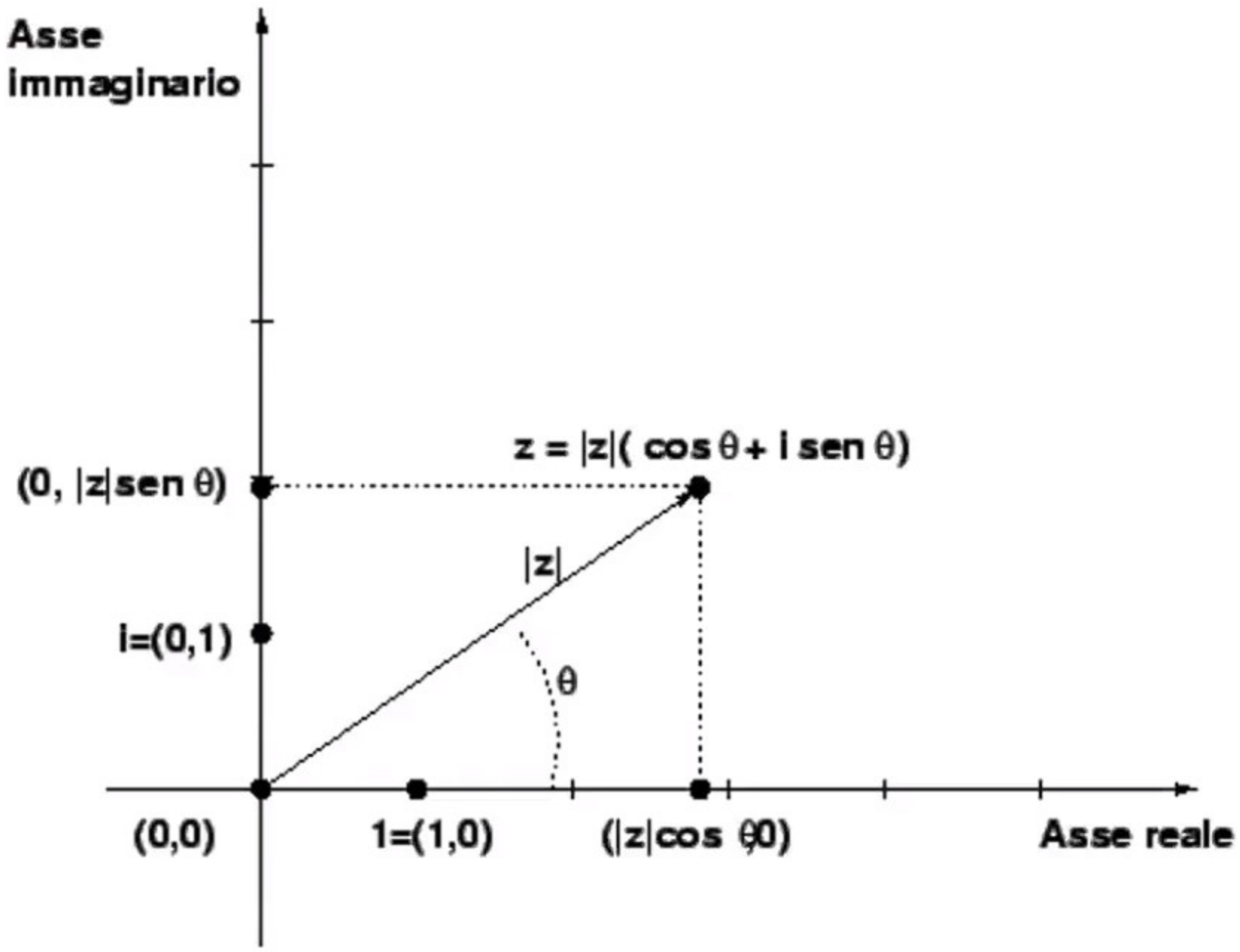
```
//complessi.h
struct comp { double re, im; };

comp inizializzaComp(double, double);
double reale(comp);
double immag(comp);
comp somma(comp, comp);
```



Rappresentazione Cartesiana

Alternativa:

Rappresentazione polare



```
//complessi.cpp
#include"complessi.h"

comp inizializzaComp(double re, double im){
   comp x;
   x.re=re; x.im=im;
   return x;
}

double reale(comp x){
   return x.re;
}

double immag(comp x){
   return x.im;
}

comp somma(comp x, comp y){
   comp z;
   z.re=x.re+y.re; z.im=x.im+y.im;
   return z;
}
```

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

int main() {
 comp z1;
 comp x1 = inizializzaComp(0.3,3.1);
 comp y1 = inizializzaComp(3,6.3);
 z1=somma(x1,y1);

 // possiamo però usare la rappresentazione interna dell'ADT !
 comp x2 = {0.3,3.1}, y2 = {3,6.3};
 comp z2;
 z2.re=x2.re+y2.re; z2.im=x2.im+y2.im;

 std::cout << "z1 => (" << reale(z1) <<"," << immag(z1) << ")\n";
 std::cout << "z2 => (" << z2.re <<"," << z2.im << ")\n";
}
```

- Concetto OO di **classe** implementa gli ADT
- **Dichiarazione** dei membri della classe
    1) Campi dati
    2) Metodi
- **Definizione** o **implementazione** della classe
    - Definizione dei metodi

Running example: classe orario

Dichiarazione della classe orario

```
class orario {
private:              // unico campo dati della classe
 int sec;             // scegliamo di rappresentare l'ora
                      // del giorno con il numero di
                      // secondi trascorsi dalla mezzanotte




};
```

Secondi in un giorno: $24 \times 60 \times 60 - 1 = 86399$

Dichiarazione della classe orario

```
class orario {
private:              // unico campo dati della classe
 int sec;             // scegliamo di rappresentare l'ora
                      // del giorno con il numero di
                      // secondi trascorsi dalla mezzanotte

public:               // metodi pubblici della classe
 int Ore();           // selettore delle ore
 int Minuti();        // selettore dei minuti
 int Secondi();       // selettore dei secondi
};
```