Moduli

Introduzione

- Un modulo è una parte di programma costituita da uno o più file che svolge una funzionalità
- Interazione tra moduli può essere vista in termini di offerta ed uso di servizi
  - I servizi offerti da un modulo possono essere le risorse di varia natura messe a disposizione, come ad esempio, tipi, variabili, strutture dati, funzioni
  - Servitore è il modulo che offre i servizi (esporta le risorse)
  - Cliente è il modulo che utilizza i servizi (importa le risorse)
  - L'insieme dei servizi è detto interfaccia tra il modulo servitore ed il modulo cliente

modulo

- File di intestazione (.h) che contiene la dichiarazione delle risorse
- File di realizzazione (.cpp) che contiene
  1. la definizione delle risorse di interfaccia
  2. La definizione di altre entità usate solo all'interno del modulo (risorse private)
- Di norma
  - per non dover ripetere le dichiarazioni, il file .cpp include il file .h
  - Il file .cpp non rende disponibili (visibili) ad altri moduli le risorse private rendendole statiche o racchiudendole in uno spazio di nomi

Information Hiding

- L'interazione clienti-servitori basata solo sulle interfacce semplifica le dipendenze fra i moduli
  - Permette di modificare la realizzazione di un modulo servitore senza influenzare i moduli clienti che altrimenti dovrebbero essere riscritti ogni volta che il modulo servitore viene modificato
- Il principio della **(i)** separazione fra interfaccia e realizzazione di un modulo servitore e **(ii)** dell'indipendenza dei moduli clienti dai moduli servitori è detta information hiding (occultamento dell'informazione)

Esempio di modulo servitore (static)

```
// serv.h
struct ss { int a; double d; };
extern int y;
extern void f2(ss)
```

```
// serv.cpp
#include "serv.h"
static int x;
static void f1() { /*…*/}
int y;
void f2(ss s) {
      ss sa;
      x = 5;
      f1();
      sa = s;
      // …
}
```


Esempio di modulo servitore (namespace)

```
// serv.h
struct ss { int a; double d; };
extern int y;
extern void f2(ss)
```

```
// serv_ns.cpp
#include "serv.h"
namespace mio {
      static int x;
      static void f1() { /*…*/}
}
using namespace mio;
int y;
void f2(ss s) {
      ss sa;
      x = 5;
      f1();
      sa = s;
      // …
}
```

Astrazioni procedurali, oggetti astratti, tipi di dato astratto

- Un modulo che realizza un'astrazione procedurale mette a disposizione di altri moduli un insieme di funzioni
  - I clienti non hanno bisogno di conoscere la realizzazione interna delle funzioni
  - Esempio: libreria di funzioni che operano su stringhe (**<string.h>**)
- Un modulo che realizza un oggetto astratto mette a disposizione di altri moduli una struttura dati
  - La rappresentazione interna della struttura dati non deve essere accessibile ai clienti che possono accedere alla struttura solo attraverso le operazioni nell'intestazione senza conoscerne la realizzazione
- Un modulo che realizza un tipo di dato astratto mette a disposizione di altri moduli un tipo di dato
  - La rappresentazione interna del tipo non deve essere accessibile ai clienti che possono accedere agli oggetti del tipo solo attraverso le operazioni dichiarate nell'intestazione senza conoscerne la realizzazione

Oggetto astratto pila (static)

```
// oa_pila.h

void inizializza();
int vuoto();
int pieno();
int push(int s);
int pop(int &s);
```

```
// oa_pila.cpp
#include "una_pila.h"

const int N = 5;
static struct {int tt; int ee[N]; } p; // collegamento interno

void inizializza() { p.tt = 0;}
void vuoto() { return p.tt == 0; }
void pieno() { return p.tt >= N; }
void push(int s) {
        if (p.tt >= N) return 0;
        p.ee[p.tt++] = s; return 1;}

void pop(int& s) {
        if (p.tt <= 0) return 0;
        s = p.ee[--p.tt]; return 1;}
```

Modularità: le classi

- Le regole di visibilità e collegamento consentono di realizzare l'oggetto astratto
- Le sole regole di visibilità e collegamento non consentono di avere moduli che realizzano compiutamente i tipi di dato astratto
- La classe individua un campo di visibilità
  - Gli identificatori dichiarati all'interno di una classe sono visibili dal punto della loro dichiarazione fino alla fine della classe stessa
  - Per le funzioni membro tutto avviene come se la loro definizione fosse posta alla fine della classe per cui nel corpo sono visibili gli identificatori presenti nella classe ma non ancora dichiarati

Classe Complesso

- Due moduli
  - Complesso che realizza il tipo di dato astratto numero complesso
    - File **complesso.cpp**, **complesso.h**
  - Complessoio che realizza le operazioni di I/O (stream) sui numeri complessi
    - File **complessoio.cpp**, **complessoio.h**

Modulo complesso

```
// complesso.h
class complesso {
   friend complesso operator+(complesso,
                              complesso);
   double re, im;
public:
   complesso(double r = 0, double i = 0);
   double reale();
   double immag();
};
```

```
//complesso.cpp
#include "complesso.h"
complesso::complesso(double r, double i) {
          re = r; im = i;}
double complesso::reale() { return re; }
double complesso::immag() { return im; }
complesso operator+(complesso x, complesso y) {
   complesso z;
   z.re = x.re + y.re;
   z.im = x.im + y.im;
   return z;
}
```


Modulo complessoio

```
//complessoio.h
using namespace std;

ostream& operator<<(ostream& os,
                    complesso z);
istream& operator>>(istream& is,
                    complesso& z);
ostream& write(ostream& is,
                    complesso* z);
istream& read(istream& os,
                    complesso* z);
```

```
//complessoio.cpp
#include <fstream>
#include "complesso.h"
#include "complessoio.h"

using namespace std;

ostream& operator<<(ostream& os,
                    complesso z) {
   os << '(' << z.reale()    << ','
             << z.immag()  << ')';
}

// continua
```

Modulo complessoio

```
//complessoio.cpp
istream& operator>>(istream& is,
                    complesso& z) {
   double re = 0, im = 0; char c = 0;

   is >> c;
   if ( c != '(') is.clear(ios::failbit);
   else {
      is >> re >> c;
      if (c != ',') is.clear(ios::failbit);
      else {
         is >> im >> c;
         if (c != ')') is.clear(ios::failbit);
      }
   }
   if (is) z = complesso(re, im);
   return is;
}                                  // continua
```

```
//complessoio.cpp

istream& read(istream& is, complesso* z)
{
   return is.read((char *)z,
                  sizeof(complesso));
}

ostream& write(ostream& os, complesso* z)
{
   return os.write((char *)z,
                  sizeof(complesso));
}

// fine
```