Programmazione procedurale e ad oggetti - 1/07/2019

Nome: .....
Cognome: .....
Matricola: .....

1. Presentate le principali funzioni per l'accesso ai file in C.

Sol. Si vedano le funzioni `fopen, fread, fwrite, fprintf, fscanf, fseek, ftell, fclose` (lucidi del corso e libro di testo).

2. Illustrate tramite esempi i diversi usi della parola riservata `static`.

Sol. Si vedano le variabili locali statiche e la restrizione della visibilità delle variabili globali (lucidi del corso e libro di testo).

3. Definite una funzione `Max` che restituisce il massimo valore in una matrice di `double`.

Sol. Per poter definire una funzione in grado di accettare una matrice di ordine arbitrario, occorre che le matrici siano rappresentate tramite vettori di puntatori alle righe. Si ha quindi:

```
double Max(double** mat,int nrighe,int ncolonne)
{int i,j; double max=mat[0][0];
for(i=0;i<nrighe;i++)
 for(j=0;j<ncolonne;j++)
  if(max<mat[i][j]) max=mat[i][j];
return max;
}
```

4. Illustrate i possibili usi delle parole riservate `public, protected, private`. *Sol.* Si veda (lucidi del corso e libro di testo) le descrizione dei nove casi ottenibili combinando le tre modalità di ereditarietà con i tre modi di etichettare un membro della classe.

5. Completate la seguente dichiarazione

```
class Complex {
double re; double im;
public:
...};
```

in modo tale che, dato un oggetto `Complex a(1.1,2.2)`, l'espressione `cout<< a` stampi `(1.1,2.2)`

Sol. Occorre effettuare l'overloading dell'operatore `<<`, che dovrà risultare amico della classe `Complex` affinché possa accedere ai membri privati di un oggetto.

```
class Complex {
double re; double im;
public:
friend ostream& operator<<(ostream& os,const Complex& c);
...};
ostream& operator<<(ostream& os,const Complex& c)
{
    os << '('<<c.re << ',' << c.im << ')' ;
    return os;
}
```

6. Date le 4 classi seguenti

```
class A {public:
void f() {std::cout <<"f di A\n";g();};
virtual void g() {std::cout <<"g di A\n";};};
class B: virtual public A {public: void f() {std::cout <<"f di B\n";g();};};
class C: virtual public A {public: virtual void g() {std::cout <<"g di C\n";};};
class D: public B, public C {public: virtual void g() {std::cout <<"g di D\n";};};
```

e le dichiarazioni `A* pA; C* pC; D d;` spiegate quale sarebbe l'effetto dell'esecuzione delle seguenti istruzioni:

1. `pA=&d; pA->f(); pA->g();`

   Viene stampato

   ```
   f di A
   g di D
   g di D
   ```

   Infatti, la funzione `f` invocata tramite il puntatore `pA` non è virtuale, stampa quindi `f di A` dopodiché chiama `g`. Questa è invece una funzione virtuale e quindi di fatto invoca la versione specifica per la classe reale dell'oggetto, da cui la stampa "`g di D`". Ovviamente la chiamata di `g` tramite `pA` comporta l'esecuzione della versione di `g` della classe `D`.

2. `pC=&d; pC->f();`

   Partendo dalla classe `C`, la ricerca del metodo `f` porta alla versione presente nella classe `A`. Essendo la funzione `g` virtuale, si ha quindi

   ```
   f di A
   g di D
   ```

3. `d.f();`

   Partendo da `D`, la classe più vicina in cui si trova un metodo `f` è `B`. Ricordando che `g` è virtuale, si ha

   ```
   f di B
   g di D
   ```