Programmazione procedurale e ad oggetti - 5/06/2017

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

1. Presentate le principali funzioni per la gestione della memoria dinamica in C.

Sol. Si vedano (lucidi e libro di testo) le funzioni `malloc`, `calloc` e `free`.

2. Definite una macro `scambio(a,b)` da utilizzare per scambiare il contenuto di due variabili `a` e `b` di tipo `int`.

Sol.

```
#define scambio(a,b) {int tmp_scambio; tmp_scambio=a;a=b;b=tmp_scambio;}
```

3. Definite una funzione che riceve in ingresso una stringa (vettore di caratteri) e restituisce la somma delle lunghezze di tutte le stringhe su cui è stata chiamata.

Sol. Occorre utilizzare una variabile locale statica (inizializzata a zero) come contatore

```
int sommaLun(char* s)
{static int count=0;
 while(*s!=0){count++;s++;};
 return count;}
```

4. Data la seguente dichiarazione (punti a coordinate intere in uno spazio n-dimensionale)

```
class nDPoint {
int* p; int dim;
public:
 nDPoint(int n){p=new int[n]; dim=n;};
 ~nDPoint(){delete[] p;};
...};
```

effettuate l'overloading di `<<` in modo che se `punto` corrisponde a $(a_1, a_2, \ldots, a_n)$ l'istruzione `std::cout << punto;` stampi $[a_1, a_2, \ldots, a_n]$.

Sol. All'interno della classe `nDPoint` deve essere posta (è irrilevante se nella parte pubblica, privata o protetta) la dichiarazione di amicizia

```
friend ostream& operator<<(ostream& os, const nDPoint& c);
```

Al di fuori della classe definiremo l'overloading di `<<` nel seguente modo:

```
ostream& operator<<(ostream& os, const nDPoint& c)
{
 int i=1;
 os << "[" ;
 while(i< c.dim) {os<< c.p[i-1] <<',';i++;}
 os<<c.p[i-1]<<"]";
 return os;
}
```

5. Date le 4 classi seguenti

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

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

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

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

   La classe D eredita in maniera pubblica da B, che a sua volta eredita pubblicamente da A. Di conseguenza, è possibile assegnare l'indirizzo di un oggetto di classe D ad un puntatore ad un oggetto di classe A. Dato che la funzione `f()` è definita virtuale in A, la chiamata `pA->f()` esegue la versione presente nella classe D (viene stampato "f() di D"). Similmente, la chiamata `pA->g()` esegue la versione presente nella classe C (viene stampato "g() di C"), da cui D eredità in maniera protetta.

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

   D non rispetta l'interfaccia di C dato che eredita in modalità protetta. Si ha quindi un errore in compilazione sull'istruzione `pC=&d`.

3. `d.g();`

   Dato che D eredita da C in maniera protetta, `d.g()` causa un errore in compilazione poiché `g()` risulta protetta in D (la versione presente in A non è più accessibile).

6. Illustrate l'ordine in cui vengono chiamati i costruttori coinvolti nella creazione di un oggetto di una classe `C` che eredita da una classe `B` la quale eredita da una classe `A`.

Sol. Il compilatore assicura che per un oggetto di una classe derivata venga prima chiamato il costruttore della classe base. Di conseguenza, nell'esempio in questione osserveremo prima la chiamata del costruttore della classe A, poi quello di B e infine quello di C.