Programmazione procedurale e ad oggetti - 05/02/2019

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

1. Presentate le principali funzioni che il C offre per la gestione della memoria dinamica.

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

2. Mostrate e commentate i possibili usi della parola riservata `static`.

Sol. Un primo uso riguarda la classe di memorizzazione. Una variabile locale dichiarata `static` viene allocata nello spazio occupato dalle variabili globali (e quindi in vita per tutta la durata del programma), pur rimanendo inaccessibile quando la funzione in cui è dichiarata non è in esecuzione. Ad esempio, la seguente è una funzione che restituisce il numero di volte che è stata chiamata,

```
int count(void){
static int c=0;
return ++c;
}
```

Un secondo uso riguarda la visibilità. In questo caso la parola riservata `static` viene utilizzata per restringere la visibilità di una variabile globale al solo codice del file di appartenenza (dal punto in cui è stata dichiarata in poi).

```
void f (void){
return ++a;
/* errore, a non risulta visibile */
}
...
static int a;
/* a visibile da questa linea di codice in avanti, fino alla fine del file */
...
void g (void){
return a++;
/* ok, a risulta visibile */
}
...
```

3. Considerate la struttura `struct nodo{int i; struct nodo* next;}` e scrivete una funzione `deleteList` che accetta in ingresso una lista (nota attraverso un puntatore al primo nodo) e la cancella. Il puntatore passato come parametro dovrà al termine risultare nullo.

```
void deleteList(struct nodo** l){
struct nodo * pt1, pt2;
if(l==NULL||*l==NULL) return;
pt1=*l; pt2=pt1->next;
while(pt2!=NULL){
free(pt1);pt1=pt2;pt2=pt2->next;
}
free(pt1);
*l=NULL;
}
```

4. Qual è la differenza tra binding dinamico e binding statico?

Sol. In un linguaggio ad oggetti, per binding si intende l'associazione selettore-metodo. Questa può avvenire in due momenti distinti, ovvero in compilazione (nel caso di binding statico) o in esecuzione (nel caso di binding dinamico). Il C++ supporta il binding dinamico attraverso il meccanismo delle funzioni virtuali.

5. Quali sono i principali problemi legati all'ereditarietà multipla? Quali strumenti mette a disposizione il C++ per la loro risoluzione?

Sol. Due sono gli aspetti critici riguardanti l'ereditarietà multipla. Il primo consiste nei cosiddetti conflitti, ovvero nella presenza di attributi e/o metodi con lo stesso nome e posti in classi non confrontabili rispetto alla relazione d'ordine indotta dalla relazione di ereditarietà. Il C++ non ha alcuno strumento automatico per la risoluzione dei conflitti, limitandosi a segnalarli e lasciando al programmatore il compito di risolverli attraverso l'uso dell'operatore di scope `::`.

Il secondo problema riguarda la duplicazione di dati che si viene a creare nel momento una classe risulta essere classe base indiretta attraverso percorsi distinti nel grafo di ereditarietà. In altre parole, un oggetto di una classe `A` che eredita (indirettamente) da una classe `B` attraverso due percorsi distinti, avrà al suo interno due copie degli attributi presenti nella classa `B`. Se questo non è ciò che si desidera occorre intervenire nella definizione del grafo di ereditarietà definendo la classe base `B` come classe base virtuale (attraverso la parola chiave `virtual`) in tutte le classi che ereditano direttamente da `B`.

6. Modificate la classe

```
class Persona{char* nome; char* cognome; int matricola;};
```

in modo che se `p` è un oggetto della classe `Persona` e rappresenta lo studente Fabio Rossi con matricola 75643, l'espressione `std::cout<<p` produca il seguente output

```
Nome:Fabio
Cognome:Rossi
Matricola:75643
```

Sol. Oltre ad un costruttore per la classe `Persona` (da mettere nella parte pubblica), occorre definire un operatore `<<` amico della classe `Persona` e quindi in grado di accedere agli attributi privati.

```
class Persona{char* nome; char* cognome; int matricola;
public: Persona(char* n,char* c,int m);
friend std::ostream & operator<<(std::ostream &os, const Persona& p);}

std::ostream & operator<<(std::ostream &os, const Persona& p)
{
return os<<''Nome':'<<p.nome<<'\n'<<''Cognome:''<<p.cognome<<'\n'<<''Matricola:''
<<matricola<<'\n';
}
```