Programmazione procedurale e ad oggetti - 10/09/2018

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

1. In C a cosa servono i tre punti "`...`" nella dichiarazione di funzioni? Illustratene l'uso tramite un esempio.

Sol. I tre punti vengono utilizzati per dichiarare funzioni con un numero variabile di argomenti di tipo arbitrario. La regola da rispettare è che un primo argomento sia sempre presente, tipicamente utilizzato in esecuzione per capire quanti argomenti (e di che tipo) vengono caricati nel record di attivazione. Un esempio classico d'uso è costituito dalla funzione `printf`, che vede come primo argomento una stringa di controllo la cui lettura in esecuzione permette di capire il numero e il tipo dei dati da stampare. Formalmente, il suo prototipo è `int printf(const char *format,...)`. Ad esempio, `printf(''dato1:%d dato2:%d'',a)` illustra un uso erroneo in quanto nella stringa di controllo sono presenti due specifiche di conversione, alle quali dovrebbero corrispondere due dati da stampare, mentre ne esiste solo uno. Ciò che accade, è che in esecuzione ci sarà un accesso in una zona di memoria (nello stack) in cui ci aspetta una dato intero, la cui mancanza potrà dar luogo ad un errore in esecuzione.

2. Nel linguaggio C quali sono le classi di memorizzazione per le variabili? Descrivetene l'uso.

Sol. Si vedano le descrizioni (lucidi o testo) delle parole chiave `register`, `auto`, `static` e `extern`.

3. Considerate la struttura `struct nodo{int info; struct nodo* next;}` per rappresentare nodi di una lista. Scrivete una funzione C `delete` che accetta in ingresso una lista `l` e dopo averne cancellati i nodi restituisce un puntatore nullo (uso: `l=delete(l)`).

Sol.

```
struct nodo* delete(struct nodo* l){
 struct nodo* pt;
 if (l==NULL) return NULL;
 pt=l->next;
 while(pt!=NULL)
  {free(l);l=pt;pt=pt->next;}
 free(l);
 return NULL;
}
```

4. Quali sono i principali fattori di qualità del SW che traggono vantaggio da un approccio orientato agli oggetti?

Sol. Tra i principali fattori di qualità che traggono vantaggio dall'approccio OOP citiamo:

- Correttezza (comportarsi secondo le specifiche e i requisiti)
- Robustezza (capacità di funzionare anche in condizioni anomale)
- Estendibilità (facilità di adattamento del SW al cambiamento delle specifiche)
- Riutilizzo (possibilità di recuperare (porzioni di) un programma per nuove applicazioni)
- Compatibilità (facilità con cui programmi diversi interagiscono)

5. A cosa serve la parola riservata `friend` del C++. Mostrate degli esempi d'uso.

Sol. La parola riservata `friend` viene utilizzata per consentire a singole funzioni o a intere classi di acquisire gli stessi privilegi d'accesso che una classe stessa possiede nei riguardi dei propri campi e metodi. Informazioni private (o protette) sono quindi accessibili non solo dalle funzioni della classe d'appartenenza ma anche da singole funzioni esterne se queste sono dichiarate amiche, o da intere classi se la dichiarazione di amicizia riguarda un'intera classe. Ad esempio, possiamo definire una funzione amica per calcolare la distanza (euclidea) di due punti a coordinate intere.

```
class Point{int x;int y;friend float Distance(const Point& a,const Point& B);}
float Distance(const Point& a,const Point& B){
int dx=a.x-b.x;
int dy=a.y-b.y;
return sqrt(dx*dx+dy*dy);
;}
```

Nel caso in cui una classe `A` dichiarasse amica un'intera classe `B`, l'accesso sarebbe consentito a tutte le funzioni membro di `B` (se si vuole, si può dichiarare amica anche una sola funzione `f` di `B` scrivendo `friend B::f()` in `A`.

6. Assumete che un `int` occupi 4 byte e un puntatore 8. Considerate il seguente programma e mostrate la sequenza di messaggi stampati in esecuzione. Motivate i valori stampati.

```
#include <iostream>

class A {public: int el;};
class B: public A{public: int el; A objA; int f(void){return 1;};};
class C:  public A{public: int el; A objA;};
class D:  public B, public C{public: int el; A objA;};
int main(int argc, const char * argv[]) {
    A objA; B objB; C objC; D objD;
    std::cout<<"Dimensione oggetto A :"<<sizeof(objA)<<'\n';
    std::cout<<"Dimensione oggetto B :"<<sizeof(objB)<<'\n';
    std::cout<<"Dimensione oggetto C :"<<sizeof(objC)<<'\n';
    std::cout<<"Dimensione oggetto D :"<<sizeof(objD)<<'\n';
    return 0;}
```

Sol. Il programma stampa i seguenti messaggi:
Dimensione oggetto A :4
Dimensione oggetto B :12
Dimensione oggetto C :12
Dimensione oggetto D :32

Il primo messaggio è motivato dal fatto che la classe `A` ha un solo attributo, di tipo `int`, il secondo deriva dal fatto che in un oggetto di classe `B` 4 byte sono occupati dall'attributo `A::el`, 4 dall'attributo `B::el` e 4 da `B::objA`. Si noti che le funzioni (non virtuali) non richiedono spazio all'interno dell'oggetto. Per `objC` valgono le stesse considerazioni fatte per `objB`. Infine, poiché `A` non è classe base virtuale, l'oggetto `objD` avrà al suo interno due copie di `A::el` (8 byte) oltre a `B::objA, B::el` (8 byte) e a `C::A, C::el` (8 byte), e infine `C::objA, C::el` (ancora 8 byte), per un totale di 32 byte.