Programmazione procedurale e ad oggetti - 09/01/2019

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

1. Presentate le principali funzioni che il C offre per la gestione dei file.

Sol. Si vedano i lucidi del corso.

2. Cosa fa il preprocessore? Mostrate e commentate le direttive che ritenete più significative.

Sol. Si vedano i lucidi del corso.

3. Scrivete una funzione `miamal` che assegna un blocco di un KB di memoria dinamica al puntatore costituente l'unico parametro, e che restituisce un intero che rappresenta il numero di KB allocati fino a quel momento. La funzione deve permettere al massimo l'allocazione di 100 blocchi (alla richiesta del 101-esimo blocco il puntatore risulterà nullo).

```
int miamal(void **pt){
 static int count=0;
 if pt==NULL return -1;
 if(i<100){
 *pt=malloc(1024);
 if(*pt!=NULL) return ++count; else return -1}
 }
 *pt=NULL;
 return 100;
}
```

4. Cosa si intende per conflitto in una gerarchia di classi? Esiste una tecnica automatica per la risoluzione dei conflitti? Come si comporta C++ al riguardo?

Sol. Si osserva un conflitto in presenza di nomi di metodi (o attributi) presenti in classi non confrontabili rispetto alla relazione di ereditarietà. Non esiste una tecnica generale per la risoluzione dei conflitti e che rispetti i vincoli espressi tramite molteplicità e visione modulare. C++ si limita a segnalare il conflitto lasciando al programmatore la sua soluzione (tramite l'uso dell'operatore `::`).

5. Quali sono le parole chiave utilizzate per il controllo della visibilità degli elementi di una classe? Costruite una tabella che presenti tutte le combinazioni (e il loro significato) in abbinamento con le diverse modalità di derivazione.

Sol. Si veda la tabella (3×3) presente sui lucidi e che illustra il significato delle parole `public, protected, private` (intesi come attributi di visibilità) relativamente alle modalità di derivazione pubblica, protetta e privata (`public, protected, private`).

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; int f(void){return 2;};};
class B: public A{protected: int el,m; A objA; virtual int f(void){return 1;};};
class C:  public A{public: int el; A objA;};
class D:  public B, public C{public: int el,el1; 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 :24
Dimensione oggetto C :12
Dimensione oggetto D :48

Il primo messaggio è motivato dal fatto che la classe `A` ha un solo attributo, di tipo `int`, la funzione `f` (non essendo virtuale) non occupa spazio all'interno dell'oggetto. 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`, 4 dall'attributo `B::m`, 4 da `B::objA` e 8 per il puntatore associato alla funzione virtuale `f`. Si noti che le funzioni (non virtuali) non richiedono spazio all'interno dell'oggetto. Per `objC` lo spazio è quello occupato da `C::el` (4), `A::el` (4) e `C::objA` (altri 4 byte). Infine, poiché `A` non è classe base virtuale, vale che `sizeof(D)`=`sizeof(B)`+`sizeof(C)`+2`sizeof(int)` (`D::el` `D::el1`) +`sizeof(A)` = 48. Si noti che l'oggetto `objD` avrà al suo interno due copie di `A::el` (8 byte).