Programmazione procedurale e ad oggetti - 2/09/2019

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

1. Elencate i diversi usi del valore 0 in C.

Sol.

- rappresenta il valore di verità falso
- viene usato come valore per il puntatore nullo
- viene usato come terminatore di stringa (carattere appeso in coda a ogni stringa)
- è l'indice del primo elemento di un vettore
- è l'intero 0 (nelle espressioni aritmetiche)

2. Considerate alberi binari i cui nodi sono così definiti
`struct nodo{int k; struct nodo* sx; struct nodo* dx;}`
e progettate una funzione che accetta in ingresso un albero, ne elimina le foglie e restituisce un puntatore alla radice dell'albero così ottenuto

Sol. Facendo uso della ricorsione si ha:

```
struct nodo* DeleteLeaves(struct nodo* t)
{if(t==NULL) return t;
if(t->sx==NULL&&t->dx==NULL){free(t);return NULL};
if(t->sx!=NULL) t->sx=DeleteLeaves(t->sx);
if(t->dx!=NULL) t->dx=DeleteLeaves(t->dx);
return t;
}
```

3. Supponete che un `char` occupi 1 byte, un `int` 4 byte, un `float` 6 byte e un puntatore 8. Date le dichiarazioni

```
char a=2;
short b=4;
float c=6.0;
char nome[]="paperoga";
char* pchar=nome;
```

dite quali sono i valori delle seguenti espressioni (con spiegazione)

- `sizeof(a+2)` : la costante 2 viene promossa a `int` e quindi il tipo dell'espressione è `int`, il valore è 4
- `sizeof(a+c)` : `a` viene promosso a `float`, quindi il valore è 6
- `sizeof(nome)` : il vettore `nome` occupa 9 byte (8 caratteri + il terminatore di stringa), il valore è 9
- `sizeof(2*a==b)` : il tipo dell'espressione è `int`, quindi il valore è 4
- `sizeof(*pchar)` : `pchar` punta a un carattere, quindi il valore è 1

4. Cosa si intende per conflitto in un grafo di ereditarietà? Come vengono risolti i conflitti in C++? Mostrate due grafi d'ereditarietà che illustrano rispettivamente l'assenza e la presenza di conflitti.

Sol. Si ha un conflitto quando esistono due classi A e B che presentano un metodo (o un attributo) con ugual nome e nè A è superclasse (diretta o indiretta) di B né viceversa (in altre parole A e B sono inconfrontabili rispetto alla relazione d'ordine indotta dalla relazione di ereditarietà). Ovviamente i conflitti possono costituire un problema solo nel caso di ereditarietà multipla e nel caso in cui un oggetto riceva un messaggio con un selettore che compare in due classi inconfrontabili da cui la classe dell'oggetto eredita (direttamente o indirettamente). Il C++ permette la risoluzione manuale dei conflitti attraverso l'operatore `::`. Il più semplice grafo che illustra la situazione è ad esempio formato da tre classi A, B e C. A eredita da B e C e sia B che C hanno un metodo `p()` (non presente in A). Se a è un oggetto di classe A, `a.p()` genera un errore in compilazione e il programmatore sarà tenuto a risolverlo scrivendo `B::a.p()` oppure `A::a.p()`. Ovviamente, se all'interno della classe A ci fosse un metodo `p()`, questo verrebbe a mascherare i metodi di ugual nome posti nelle superclassi e `a.p()` non darebbe alcun problema (verrebbe utilizzata la versione presente nella classe A).

5. A quali vincoli è soggetto l'overloading di funzioni in C++? Mostrate un esempio che illustri i diversi casi.

Sol. Si vedano i lucidi del corso

6. Considerate il seguente programma e determinatene l'output (sequenza di messaggi stampati in esecuzione). In generale, quali sono le regole che determinano l'ordine secondo il quale costruttori e distruttori sono chiamati?

```
class A {public: A(int a){std::cout<<"Costruttore A("<<a<<")\n";el=a;};
         ~A(){std::cout<<"Distruttore A("<<el<<")\n";};int el;};

class B:public A{
        public: B(int b):A(b-1){std::cout<<"Costruttore  B("<<b<<")\n";el=b;};
        ~B(){std::cout<<"Distruttore B("<<el<<")\n";};
                int el; A objA=A(5);};
A objA=A(8);
void f(void){A objA(50);};
int main(void){B objB=B(2);return 0;}
```

Sol. La sequenza di messaggi stampati è la seguente

```
Costruttore A(8)
Costruttore A(1)
Costruttore A(5)
Costruttore B(2)
Distruttore B(2)
Distruttore A(5)
Distruttore A(1)
Distruttore A(8)
```

Si noti come l'oggetto di classe `A` presente nella funzione `f()` non venga mai creato poiché la funzione non viene chiamata. In generale, al momento della creazione di un oggetto di una classe derivata (`objB` di `main` nell'esercizio) viene prima chiamato il costruttore della classe base (A nell'esercizio), poi quelli per gli eventuali oggetti presenti come attributi nella classe (`objA` nell'esercizio), e solo in ultimo il costruttore della classe derivata. Nel momento in cui un oggetto esce dallo scope (locale o globale che sia) si osserva il processo inverso, ovvero una sequenza di chiamate ai distruttori nell'ordine inverso rispetto a quello osservato per i costruttori. Ovviamente,

i costruttori degli oggetti locali vengono chiamati nel momento in cui si entra nella funzione, e il loro distruttore quando si esce (o al termine del programma se l'oggetto è statico).