Programmazione procedurale e ad oggetti - 22/01/2020

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

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

Sol. Si vedano (lucidi e libro di testo) le funzioni `fopen, fclose, fseek, ftell, rewind, fprint, fwrite, fscanf, fread`.

2. Considerate la struttura così definita `struct nodo{int k; struct nodo* next;}` e progettate una funzione che accetta in ingresso un intero n e restituisce una lista di n nodi, dove il nodo i-esimo contiene il valore i ($1 \leq i \leq n$).

Sol.

```
struct nodo* IntList(int n){
 struct nodo *pt, *pt1; int i=1;
 if(n==0)return NULL;
 pt=(struct nodo*)malloc(sizeof(struct nodo));
 pt->k=i;pt->next=NULL; pt1=pt;
 while(i<n){
  pt->next=(struct nodo*)malloc(sizeof(struct nodo));
  pt=pt->next;
  pt->k=++i;
  pt->next=NULL;}
 return pt1;
}
```

3. Definite una macro `MEDIANA(a,b,c)` da utilizzare per calcolare il valore di mezzo tra 3 elementi `a,b,c` (supponete che per il tipo degli elementi sia definito l'operatore `<`). Ci sono dei limiti nell'uso di tale macro?

Sol.

```
#define MEDIANA(a,b,c) (a>b)?((a<c)?(a):((b<c)?(c):(b))):((a>c?(a):(b<c?(b):c))))
```

Il limite principale è costituito dall'uso con parametri che fanno uso di operatori con side effect. Ad esempio, se `x, y, z` valgono 7 allora `MEDIANA(++x,y,z)` ha come valore 10 (`x` è incrementata 3 volte).

4. A cosa servono in C++ le parole chiave `public, protected` e `public`? Elencatene tutti i possibili usi.

Sol. Si veda (lucidi del corso e libro di testo) le descrizione dei nove casi ottenibili combinando le tre modalità di ereditarietà con i tre modi di etichettare un membro della classe.

5. Che problemi pone l'ereditarietà multipla? Come si pone il C++ al riguardo e quali strumenti offre per la soluzione di tali problemi?

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. Considerate la classe `IntVector` seguente

```
class IntVector {int lenght, int* m;
 public: IntVector(int n) {m=new int[n];lenght=n;};
  ~IntVector(void){delete []m;};
  int Lenght() const {return n;};
}
```

e dite come andrebbe completata in modo che

- `A[i]` corrisponda all'elemento i-esimo di `A` di classe `IntVector` ($1 \leq i \leq$`A.Length()`);
- la valutazione dell'espressione `A+B` restituisca il vettore somma nel caso `A` e `B` abbiamo la stessa lunghezza (o arresti la computazione altrimenti).

Sol. Si ricorre all'overloading degli operatori `=`, `[]` e `+`.

```
class IntVector {int lenght; int* m;
public: IntVector(int n){m=new int[n];lenght=n;}
   ~IntVector(void){delete []m;};
    int Length() const {return lenght;} ;
    int operator[](int i) const{return m[i-1];};
    int& operator[](int i){return m[i-1];};
    friend IntVector operator+(const IntVector&, const IntVector&);
    void operator=(const IntVector &V);
};

void IntVector::operator=(const IntVector &V){
    for(int i=0;i<this->Length();i++)
        this->m[i]=V[i];};

IntVector operator+(const IntVector& A,const IntVector& B){
    IntVector R(A.Length());
    if(A.Length()!=B.Length()) exit(1);
    for(int i=1;i<=A.Length();i++)
        R[i]=A[i]+B[i];
    return R;}
```

Si noti come sia presente un doppio overloading dell'operatore di accesso in modo da poterlo usare anche con oggetti dichiarati costanti (ovvero non modificabili), consentendo così l'accesso in lettura. La modifica del copy constructor di default si rende necessaria in quanto il costruttore alloca memoria dinamicamente.