Esercizi di Programmazione ad Oggetti, a.a. 10/11

Esercizio 1

```
class Z {
public:
  ...
};

template <class T1, class T2=Z>
class C {
public:
  T1 x;
  T2* p;
};

template<class T1,class T2>
void fun(C<T1,T2>* q) {
  ++(q->p);
  if(true == false) cout << ++(q->x);
  else cout << q->p;
  (q->x)++;
  if(*(q->p) == q->x) *(q->p) = q->x;
  T1* ptr = &(q->x);
  T2 t2 = q->x;
}

main(){
  C<Z> c1; fun(&c1); C<int> c2; fun(&c2);
}
```

Si considerino le precedenti definizioni. Fornire una dichiarazione (non è richiesta la definizione) dei membri pubblici della classe `Z` nel **minor numero possibile** in modo tale che la compilazione del precedente `main()` non produca errori. **Attenzione:** ogni dichiarazione in `Z` non necessaria per la corretta compilazione del `main()` è penalizzata.

Esercizio 2

Definire una superclasse `ContoBancario` e due sue sottoclassi `ContoCorrente` e `ContoDiRisparmio` che soddisfano le seguenti specifiche:

- Ogni `ContoBancario` è caratterizzato da un saldo e rende disponibili due funzionalità di deposito e prelievo: `int deposita(int)` e `int preleva(int)` che ritornano il saldo aggiornato dopo l'operazione di deposito/prelievo.
- Ogni `ContoCorrente` è caratterizzato anche da una spesa fissa uguale per ogni `ContoCorrente` che deve essere detratta dal saldo ad ogni operazione di deposito e prelievo.
- Ogni `ContoDiRisparmio` deve avere un saldo non negativo e pertanto non tutti i prelievi sono permessi; d'altra parte, le operazioni di deposito e prelievo non comportano costi aggiuntivi e restituiscono il saldo aggiornato.
- Si definisca inoltre una classe `ContoArancio` derivata da `ContoDiRisparmio`. La classe `ContoArancio` deve avere un `ContoCorrente` di appoggio: quando si deposita una somma S su un `ContoArancio`, S viene prelevata dal `ContoCorrente` di appoggio; d'altra parte, i prelievi di una somma S da un `ContoArancio` vengono depositati nel `ContoCorrente` di appoggio.

Esercizio 3

```
class B {
public:
  int x;
  B(int z=1): x(z) {}
};

class D: public B {
public:
  int y;
  D(int z=5): B(z-2), y(z) {}
};

void fun(B* a, int size) {
  for(int i=0; i<size; ++i) cout << (*(a+i)).x << " ";
}

int main() {
  fun(new D[4],4); cout << "**1\n";
  B* b = new D[4]; fun(b,4); cout << "**2\n";
  b[0] = D(6); b[1] = D(9);  fun(b,4); cout << "**3\n";
  b = new B[4]; b[0] = D(6); b[1] = D(9);
  fun(b,4); cout << "**4\n";
}
```

La compilazione delle precedenti definizioni non provoca errori (con gli opportuni `include` e `using`) e la loro esecuzione non provoca errori a run-time. Si scrivano nell'apposito spazio le stampe provocate in output dall'esecuzione del `main()`.

Esercizio 4

Definire un template di classe `albero<T>` i cui oggetti rappresentano un **albero 3-ario** ove i nodi memorizzano dei valori di tipo `T` ed hanno 3 figli (invece dei 2 figli di un usuale albero binario). Il template `albero<T>` deve soddisfare i seguenti vincoli:

1. Deve essere disponibile un costruttore di default che costruisce l'albero vuoto.
2. Gestione della memoria senza condivisione.
3. Overloading dell'operatore di uguaglianza.
4. Overloading dell'operatore di output.