

**Analisi e progettazione del software**  
**Compito di metà corso**  
**24 novembre 2015**

**Esercizio 1 (punti 8)** Si considerino la classe A e la funzione main qui sotto.

```
class A
{
public:
    A(unsigned n = 0);
    void Set(double x, unsigned i) { if (i < dim) vet[i] = x; }
    double Get(unsigned i) const { if (i < dim) return vet[i]; else return 0.0; }
private:
    double* vet;
    unsigned dim;
};

A::A(unsigned n)
{
    dim = n;
    if (dim > 0)
    {
        vet = new double[dim];
        for (unsigned i = 0; i < dim; i++)
            vet[i] = 0.0;
    }
    else
        vet = nullptr;
}

int main()
{
    A a1, a2(4);
    a2.Set(2.5, 3);
    a2.Set(-2.3, 4);
    a1 = a2;
    a1.Set(1.4, 3);
    cout << a2.Get(3) << " " << a2.Get(4) << endl;
    return 0;
}
```

- Scrivere se il programma va in errore in compilazione, oppure in esecuzione, oppure funziona correttamente e in caso positivo cose stampa.
- Scrivere le tre funzioni speciali della classe A in modo che evitino la condivisione di memoria.

**Soluzione.** Il programma stampa 1.4 0, in quanto c'è interferenza tra gli oggetti a1 e a2. Le funzioni speciali vanno così definite.

```
A::A(const A& a)
{
    dim = a.dim;
    if (dim > 0)
    {
        vet = new double[dim];
        for (unsigned i = 0; i < dim; i++)
            vet[i] = a.vet[i];
    }
    else
        vet = nullptr;
}

A& A::operator=(const A& a)
{
    if (a.dim > 0)
    {
        if (dim != a.dim )
        { // Nota: nel caso di puntatore nullo, la delete non ha effetti
            delete[] vet;
            dim = a.dim;
            vet = new double[dim];
        }
        for (unsigned i = 0; i < dim; i++)
            vet[i] = a.vet[i];
    }
    else // a.dim == 0
    {
        if (dim > 0)
        {
            delete[] vet;
            vet = nullptr;
            dim = 0;
        }
        // else: in questo caso non c'e' niente da eseguire
    }
    return *this;
}

A::~~A()
{
    delete[] vet;
}
```

**Esercizio 2 (punti 7)** Si consideri la classe `Pila` come definita a lezione, con i campi `dim`, `top` e `vet`, i metodi `Push()`, `Pop()`, `Top()` ed `IsEmpty()`, i costruttori e gli operatori spiegati in classe.

Si definisca, nel modo che si ritiene più opportuno, l'operatore `+` che restituisce la pila ottenuta giustapponendo i due operandi di tipo `Pila`. Ad esempio, se le pile `p1` e `p2` contengono rispettivamente gli elementi  $\langle 7, 3, 4, -12, 4 \rangle$  e  $\langle -9, 0, -8 \rangle$  (elemento affiorante 4 e -8, rispettivamente), la pila `p1+p2` conterrà gli elementi  $\langle 7, 3, 4, -12, 4, -9, 0, -8 \rangle$ .

**Soluzione.** Si definisce la seguente funzione esterna *friend* della classe `Pila`.

```
Pila operator+(const Pila& p1, const Pila& p2)
{
    Pila p = p1;
    for (int i = 0; i <= p2.top ; i++)
        p.Push(p2.vet[i]);
    return p;
}
```

Si noti che la gestione del ridimensionamento del vettore interno della pila è interamente delegata al metodo `Push()`.

**Esercizio 3 (punti 6)** Si consideri la classe `Data` come definita a lezione (con i campi `giorno`, `mese` e `anno`, i metodi, gli operatori e i costruttori spiegati in classe).

Si scriva l'operatore `>>` (ovviamente *friend*) che accetti come ingresso una data scritta sia nel formato `gg-mm-aaaa` che nel formato `mm gg, aaaa`. In caso di data non valida, l'operatore deve memorizzare nell'oggetto la data 1/1/1970 (si assuma presente il metodo privato `Valida()` che verifica se la data è valida).

**Soluzione.**

```
istream& operator>>(istream& is, Data& d)
{
    int tmp;
    char ch;

    is >> tmp;
    ch = is.get();
    if (ch == '-')
    { // formato gg-mm-aaaa
        d.giorno = tmp;
        is >> d.mese >> ch >> d.anno;
    }
    else
    { // formato mm gg, aaaa
        d.mese = tmp;
        is >> d.giorno >> ch >> d.anno;
    }
}
```

```

if (!d.Valida())
{
    d.giorno = 1;
    d.mese = 1;
    d.anno = 1970;
}
return is;
}

```

**Esercizio 4 (punti 10)** Si scriva una funzione che riceva come parametro il nome di un file che contiene i risultati di un insieme di partite di calcio nel formato che si evince dal seguente esempio (i nomi non contengono spazi bianchi).

```

Bologna - Roma 2 2
Juventus - Milan 1 0
Verona - Napoli 0 2
Udinese - Sampdoria 1 0
Verona - Bologna 0 2
Roma - Lazio 2 0

```

La funzione deve restituire un oggetto di tipo `vector<pair<string,unsigned>>` che contiene, in un qualsiasi ordine, per ogni squadra presente nel file, il numero totale di punti ottenuti (vittoria: 3 punti, pareggio: 1 punto, sconfitta: 0 punti).

Ad esempio, se il file è quello mostrato, l'oggetto restituito dovrà contenere i seguenti valori.

Bologna	4
Roma	4
Juventus	3
Milan	0
Verona	0
Napoli	3
Udinese	3
Sampdoria	0
Lazio	0

Si assuma già disponibile la funzione

```
int CercaSquadra(const vector<pair<string,unsigned>>& v, string sq)
```

che restituisce la locazione del vettore `v` in cui è presente la squadra di nome `sq`, e `-1` se la squadra non è presente nel vettore.

**Soluzione.** Si utilizza una funzione ausiliaria, chiamata `SommaPunti()`, che attribuisce i punti alla squadra, eventualmente aggiungendo la squadra al vettore. Si noti che è necessario sommare anche 0 punti, per permettere alla squadra di comparire comunque nel vettore.

```

vector<pair<string,unsigned>> CalcolaClassifica(string nome_file)
{
    ifstream is(nome_file);
    string sq1, sq2;
    char ch;
    unsigned g1, g2;

    vector<pair<string,unsigned>> v;
    while (is >> sq1 >> ch >> sq2 >> g1 >> g2)
    {
        if (g1 > g2)
        {
            SommaPunti(v,sq1,3);
            SommaPunti(v,sq2,0);
        }
        else if (g1 == g2)
        {
            SommaPunti(v,sq1,1);
            SommaPunti(v,sq2,1);
        }
        else // g2 > g1
        {
            SommaPunti(v,sq1,0);
            SommaPunti(v,sq2,3);
        }
    }
    return v;
}

void SommaPunti(vector<pair<string,unsigned>>& v, string sq, unsigned p)
{
    int i = CercaSquadra(v,sq);
    if (i != -1)
        v[i].second += p;
    else
        v.push_back(make_pair(sq,p));
}

```