Analisi e progettazione del software
Compito di metà corso
24 novembre 2017

Esercizio 1 (punti 13) Si considerino la classe `A` e la funzione `main()` definite di seguito

```
class A
{
 public:
  A(unsigned d, bool split = true);
  double Get1(unsigned i) const
                    { return p1[i]; }
  double Get2(unsigned i) const
                    { return p2[i]; }
  void Set1(unsigned i, double val)
                    { p1[i] = val; }
  void Set2(unsigned i, double val)
                    { p2[i] = val; }
  void Merge();
  void Split();
 private:
  double* p1;
  double* p2;
  unsigned dim;
};

A::A(unsigned d, bool split)
{
  unsigned i;
  dim = d;
  p1 = new double[d];
  for (i = 0; i < dim; i++)
    p1[i] = 0.0;
  if (split)
    {
      p2 = new double[d];
      for (i = 0; i < dim; i++)
        p2[i] = 0.0;
    }
  else
    p2 = p1;
}
```

```
void A::Merge()
{
  if (p1 != p2)
    {
      delete[] p2;
      p2 = p1;
    }
}

void A::Split()
{
  unsigned i;
  if (p1 == p2)
    {
      p2 = new double[dim];
      for (i = 0; i < dim; i++)
        p2[i] = p1[i];
    }
}


int main()
{
  A a1(10), a2(10, false);
  a2.Set1(4,8.5);
  a1 = a2;
  a1.Set2(4,11.2);
  a2.Split();
  cout << a2.Get2(4) << endl;
  return 0;
}
```

1.1 (2 punti) Riportare cosa stampa il programma, mostrando anche il procedimento con cui si è giunti al risultato.

1.2 (4 punti) Scrivere il costruttore di copia della classe `A` in modo che eviti la condivisione di memoria.

1.3 (5 punti) Scrivere l'operatore di assegnazione della classe `A` in modo che eviti la condivisione di memoria.

1.4 (2 punti) Scrivere il distruttore della classe `A` in modo che rilasci la memoria dinamica non più utilizzata.

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 `EstVuota()`, i costruttori e gli operatori spiegati in classe.

Si definisca, nel modo che si ritiene più opportuno, l'operatore `-=` che riceve come operando destro un intero i e modifica l'operando sinistro (di tipo `Pila`) eliminando tutte le occorrenze del valore i. Ad esempio, se la pila `p` contiene gli elementi $\langle$`5,2,4,4,-1,4`$\rangle$, a seguito dell'istruzione `p -= 4;` questa conterrà gli elementi $\langle$`5,2,-1`$\rangle$.

Esercizio 3 (punti 7) Si consideri la classe `Polinomio` sviluppata nell'esercitazione 3, considerando già disponibile anche l'operatore `()` che riceve un parametro di tipo `double` e restituisce il valore del polinomio nel punto corrispondente al parametro.

Sia dato un file che contiene una lista di polinomi, uno per riga, nel formato letto e scritto dagli operatori di input e output sviluppati nell'esercitazione. Come esempio, si consideri il seguente file.

```
<2.3x^5 + -3.1x^2>
<1.22x^1 + 4.2x^0>
<3x^2 + -2x^1 + 4x^0>
<2x^2 + -1x^0>
<3x^2 + -2x^1 + 5.22x^0>
<6x^4 + -4x^3 + 5x^2 + 2x^1 + -4x^0>
<-2.3x^5 + -3x^2 + 2x^1 + -4.21x^0>
<2.3x^5 + 3x^2 + -2x^1 + 2.78x^0>
<3x^2 + -2x^1 + 9x^0>
```

Si scriva una funzione booleana che riceva come parametro il nome di un file siffatto e due valori reali p ed ε e restituisca vero se e solo se ci sono nel file due polinomi che hanno lo stesso valore nel punto p. L'uguaglianza va intesa con una tolleranza pari a ε, cioè sono considerati uguali due valori che differiscono di al più ε.

Nell'esempio, se $p = 1.0$ e $\varepsilon = 0.0001$ allora, la funzione restituisce `true` in quando i polinomi $\langle 3x^2 - 2x + 4\rangle$ e $\langle 6x^4 - 4x^3 + 5x^2 + 2x - 4\rangle$ hanno lo stesso valore (5.0 in questo caso). Se invece $p = 0.0$ e $\varepsilon = 0.0001$, allora la funzione restituisce `false` perché non ci sono due polinomi che hanno lo stesso valore nel punto 0.0.

Esercizio 4 (punti 4) Si scriva un *driver* per la verifica della funzione dell'esercizio 3 che riceva *sulla riga di comando* il nome del file e, opzionalmente, il valore p. Il valore di ε invece deve essere fissato a 0.0001 e scritto in una costante locale della funzione `main()`.