

Analisi e progettazione del software

Compito di metà corso

22 novembre 2018

Esercizio 1 (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 k e modifica l'operando sinistro (di tipo `Pila`) inserendo nella pila in posizione adiacente alla prima occorrenza di k (per "prima" si intende la più vicina all'elemento affiorante) un altro elemento pari a k . Nel caso k non sia già presente nella pila, allora k deve essere inserito come elemento affiorante.

Inoltre l'operatore deve essere scritto in modo da poter essere inserito in espressione che modifichino la pila stessa.

Ad esempio, se la pila `p` contiene gli elementi (4, 3, 4, 5, 3, 2) (con 2 affiorante), a seguito dell'istruzione

```
(p += 4) += 6;
```

questa conterrà gli elementi (4, 3, 4, 4, 5, 3, 2, 6).

Soluzione:

```
Pila& Pila::operator+=(int k)
{
    int i, j;
    Push(k); // inserisce l'elemento in testa, eventualmente
    // viene spostato successivamente (Push gestisce il ridimensionamento)
    for (i = top-1; i >= 0; i--)
    {
        if (vet[i] == k)
        {
            for (j = top; j > i+1; j--)
                vet[j] = vet[j-1];
            vet[j] = k;
            break;
        }
    }
    return *this;
}
```

Esercizio 2 (punti 7) Si consideri la classe `Data` come definita a lezione, con i campi `giorno`, `mese` e `anno`, e i costruttori spiegati in classe. Si considerino disponibili anche gli operatori `++`, `--`, `+`, `+=`, `==`, `!=`, `<`, `<=`, `>` e `>=`, ma non l'operatore binario `-`.

Si definisca, nel modo che si ritiene più opportuno, l'operatore binario `&` che restituisce la data ottenuta come *media* (eventualmente approssimata per difetto) tra i due operandi di tipo `Data`. Ad esempio, il seguente frammento di codice

```

Data d1(12,9,2018), d2(21,10,2018), d3(2,11,2018);
cout << (d1 & d2) << endl;
cout << (d3 & d2) << endl;

```

dovrà stampare:

```

1/10/2018
27/10/2018

```

Soluzione: Versione 1:

```

Data operator&(Data d1, Data d2)
{
    if (d1 < d2)
    {
        while (d1 < d2)
        {
            ++d1;
            --d2;
        }
        return d2; // se la distanza è dispari,
        // quando si esce dal ciclo, d2 e' il piu' piccolo
    }
    else
    {
        while (d2 < d1)
        {
            ++d2;
            --d1;
        }
        return d1;
    }
}

```

Versione 2:

```

Data operator&(const Data& d1, const Data& d2)
{
    Data d_min, d_max;
    unsigned conta = 0;
    if (d1 < d2)
    {
        d_min = d1;
        d_max = d2;
    }
    else
    {
        d_min = d2;
        d_max = d1;
    }
    while (d_min < d_max)

```

```

    {
        --d_max;
        conta++;
    }
    d_min += conta/2;
    return d_min;
}

```

Esercizio 3 (punti 8) Si consideri la classe `Polinomio` vista ad esercitazione, utilizzando però come rappresentazione un vettore di coppie composte da grado e coefficiente. Si utilizzino le classi `vector` e `pair`, dichiarando quindi come unico membro privato il seguente dato

```
vector<pair<unsigned,double>> v;
```

Il vettore `v` deve contenere solo i termini per i quali il coefficiente è diverso da zero ed essere ordinato per grado (crescente o decrescente, a scelta).

Per questa classe si definiscano:

- l'operatore `()` che riceve un parametro di tipo `double` e restituisce il valore del polinomio nel punto corrispondente al parametro;
- l'operatore di input che legge un polinomio nel formato visto nell'esercitazione (ad es: `<6.3x^5 + -4.1x^3 + 5.1x^2 + 2.3x^1 + -1.0x^0>`).

Non sono richieste le definizioni della classe e degli altri metodi.

Soluzione:

```

double Polinomio::operator()(double x) const
{
    double val = 0.0;
    for (unsigned i = 0; i < v.size(); i++)
        val += v[i].second * pow(x,v[i].first);
    return val;
}

istream& operator>>(istream& is, Polinomio& p)
{
    char ch;
    double coef;
    int grado;

    p.v.clear();
    is >> ch; // legge il carattere <
    ch = is.peek();
    while (ch != '>')
    {
        is >> coef >> ch >> ch >> grado >> ch;
    }
}

```

```

        p.v.push_back(make_pair(grado,coef));
    }
    if (p.v.size() == 0)
        is >> ch;
    return is;
}

```

Esercizio 4 (punti 8) Si considerino la classe A e la funzione main() definite di seguito

```

class A
{public:
    A(unsigned d = 5);
    double Get(unsigned i) const
        { if (b) return v[i];
          else return (v[i]+w[i])/2; }
    void Set(unsigned i, int e)
        { if (b) v[i] = e;
          else w[i] = e; }
    void X();
private:
    int* v;
    int* w;
    bool b;
    unsigned dim;
};

A::A(unsigned d)
{ dim = d;
  b = true;
  v = new int[d];
  for (unsigned i = 0; i < dim; i++)
      v[i] = 0;
  w = nullptr;
}

void A::X()
{ b = false;
  w = new int[dim];
  for (unsigned i = 0; i < dim; i++)
      w[i] = 0;
}

int main()
{ A a1, a3(2);
  a1.Set(1,8);
  A a2 = a1;
  a2.Set(1,12);
  a1.X();
  a1.Set(2,7);
  a3 = a1;
  a3.Set(2,3);
  cout << a1.Get(1) << " "
  << a1.Get(2) << endl;
  cout << a3.Get(1) << " "
  << a3.Get(2) << endl;
}

```

- Riportare cosa stampa il programma, mostrando anche il procedimento con cui si è giunti al risultato.
- Scrivere il costruttore di copia della classe A in modo che eviti la condivisione di memoria.
- Scrivere il distruttore della classe A in modo che rilasci la memoria dinamica non più utilizzata.

Soluzione: Stampa: 6 3

```

A::A(const A& a)
{
    unsigned i;
    dim = a.dim;
    b = a.b;
    v = new int[dim];
}

```

```
    for (i = 0; i < dim; i++)
        v[i] = a.v[i];
    if (b)
        w = nullptr;
    else
    {
        w = new int[dim];
        for (i = 0; i < dim; i++)
            w[i] = a.w[i];
    }
}
```

```
A::~~A()
{
    delete[] v;
    if (!b)
        delete[] w;
}
```