

Esercizio 1.

```
// costruttore della classe Ricambio
Ricambio::Ricambio( MyString c, MyString p, MyString d )
    : codice( c ), produttore( p ), descrizione( d ) {}

// metodo setCodice() della classe Ricambio
void Ricambio::setCodice( MyString c ) {
    codice = c;
}

// operatore << della classe Ricambio
ostream& operator<<( ostream& out, const Ricambio& r ) {
    out << "Ricambio\n"
        << "-----\n"
        << "Codice:\t" << codice << endl
        << "Produttore:\t" << produttore << endl
        << "Descrizione:\t" << descrizione << endl
        << endl;
    return out;
}

// costruttore di copia della classe VettoreRicambio
VettoreRicambio::VettoreRicambio( const VettoreRicambio& init ) : lunghezza(
init.lunghezza ) {
    dati = new Ricambio[lunghezza];
    for( int i = 0; i < lunghezza; i++ )
    {
        dati[ i ] = init.dati[ i ];
    }
}

// operatore [] della classe VettoreRicambio
Ricambio& operator[]( int index ) {
    assert( 0 <= index && index < lunghezza );
    return dati[ index ];
}

// distruttore classe VettoreRicambio
VettoreRicambio::~VettoreRicambio() {
    delete [] dati;
}
```

Esercizio 2.

```
bool prodottoOriginale( const Ricambio& ric, const ListaProduttore& lprod )
{
    Produttore* prod = trovaProduttore( ric.getProduttore(), lprod );

    assert( prod != NULL );    // il produttore DEVE esistere

    if( prod->getNome() == MyString( "Audi" )
        || prod->getNome() == MyString( "BMW" )
        || prod->getNome() == MyString( "Citroen" )
        || prod->getNome() == MyString( "Fiat" )
        || prod->getNome() == MyString( "Mercedes" )
        || prod->getNome() == MyString( "Peugeot" )
        || prod->getNome() == MyString( "Renault" ) )
        return true;
    else
        return false;
}
```

```
// funzione di utilità : cerca un produttore in una lista
// restituisce il produttore se lo trova, altrimenti NULL
const Produttore* trovaProduttore( const MyString& codice, const
ListaProduttore& lprod )
{
    IteratoreProduttore it( lprod );
    it.VaiInTesta();
    while( ! it.isNull() )
    {
        if( it.getCurrentValue().getCodice == codice )
            return &( it.getCurrentValue() );
        it.moveToNext();
    }
    return NULL;
}
```

```
ListaRicambio& trovaRicambio(
    const MyString& descrizione,
    const ListaRicambio& lric,
    const ListaProduttore& lprod,
    bool soloOriginali = false )
{
    ListaRicambio* result = new ListaRicambio();

    IteratoreRicambio it( lric );
    it.VaiInTesta();
    while( ! it.isNull() )
    {
        Ricambio r = it.getCurrentValue();
        if( r.getDescrizione() == descrizione )
        {
            if( ! soloOriginali || prodottoOriginale( r, lprod ) )
                result.addInCoda( r );
        }
        it.moveToNext();
    }

    return *result;
}
```

Esercizio 3.

```
// interfaccia ProduttoreCertificato
class ProduttoreCertificato : public Produttore
{
public:
    ProduttoreCertificato( MyString d, MyString cod, MyString cert );
    ProduttoreCertificato( const ProduttoreCertificato& );

    const MyString& getCertificato() const;
    void setCertificato( const MyString& cert );

private:
    MyString certificato;
};

// bisogna implementare uno dei due costruttori (a scelta)

// costruttore con parametri
ProduttoreCertificato::ProduttoreCertificato( MyString d, MyString cod, MyString
cert )
    : Produttore( d, cod ), certificato( cert )
{
}

// costruttore di copia
ProduttoreCertificato::ProduttoreCertificato( const ProduttoreCertificato& init
)
    : Produttore( init.getCodice(), init.getNome() ), certificato(
init.certificato )
{
}
```