

Linguaggi di Programmazione  
AA 2022-2023  
2023-06 Progetto E3P  
**Polinomi Multivariati**

Marco Antoniotti, Rafael Peñaloza, Gabriella Pasi e Fabio Sartori  
Dipartimento di Informatica, Sistemistica e Comunicazione  
Università degli Studi di Milano Bicocca

27 maggio, 2023

## Scadenza

La consegna del progetto è fissata per il giorno 24 giugno 2023 entro le 23:55 GMT+1.

## 1 Introduzione

Una delle prime e più importanti applicazioni dei calcolatori fu la manipolazione *simbolica* di operazioni matematiche. In particolare, i sistemi noti come *Computer Algebra Systems* (cfr., Mathematica, Maple, Maxima, Axiom, etc.) forniscono funzionalità per la manipolazione di *polinomi multivariati*.

Lo scopo di questo progetto è la costruzione di due librerie (in Prolog ed in Common Lisp) per la manipolazione – per l'appunto – di polinomi multivariati.

Ma prima ancora, lo scopo di questo progetto è che

*...leggiate bene e con cura TUTTE le istruzioni per la consegna.*

Le trovate nella Sezione 8.

### 1.1 Polinomi multivariati

Un polinomio multivariato è un'espressione matematica che contiene diverse *variabili* a cui possono essere associati dei valori in un certo *dominio*<sup>1</sup>. Due esempi sono (con le solite regole di associatività):

$$x^2 + y^2,$$

$$42 \times x \times y^3 - z^2 \times y \times w \times x - x^3 \times w^3.$$

Il secondo polinomio viene normalmente riscritto in modo compatto come:

$$42xy^3 - z^2ywx - x^3w^3,$$

omettendo il simbolo di moltiplicazione  $\times$ .

Un polinomio è composto da *monomi*: i termini corrispondenti alle moltiplicazioni di *coefficienti* (elementi del dominio) e delle variabili (elevate a potenze). Nel secondo esempio qui sopra i monomi sono

$$42xy^3,$$

$$-z^2ywx,$$

$$-x^3w^3.$$

---

<sup>1</sup>Il dominio deve in realtà essere un *campo algebrico*  $\mathbf{F} = \langle U, +, \times \rangle$ .

Notate anche che il polinomio qui sopra può anche essere scritto come

$$-w^3x^3 + 42xy^3 - wxyz^2,$$

ovvero *riordinando* i monomi (i quali a loro volta sono ordinati secondo un preciso schema).

*Buona parte di questo progetto consiste nel costruire delle procedure di riordinamento delle variabili (con le loro potenze) in un monomio e dei monomi in un polinomio.*

## 2 Operazioni da implementare e rappresentazione

Le librerie che implementerete dovranno contenere alcune operazioni standard per la manipolazione di vari polinomi: estrazione dei coefficienti, calcolo del grado del polinomio, sua valutazione in un punto  $\mathbf{v} = \langle v_0, v_1, \dots, v_k \rangle$  (dove  $k$  è il numero di variabili che compaiono nel polinomio), somma, moltiplicazione, etc, etc.

La rappresentazione di monomi e polinomi dovrà essere la seguente.

**Prolog.** I monomi devono essere rappresentati da termini siffatti:

```
m(Coefficient, TotalDegree, VarsPowers)
```

per i quali si può scrivere il predicato<sup>2</sup>:

```
is_monomial(m(_C, TD, VPs)) :-  
    integer(TD),  
    TD >= 0,  
    is_list(VPs).
```

Tralasciamo al momento come controllare `Coefficient`. La lista `VarsPowers` contiene termini come il seguente:

```
v(Power, VarSymbol)
```

per i quali possiamo scrivere il predicato:

```
is_varpower(v(Power, VarSymbol)) :-  
    integer(Power),  
    Power >= 0,  
    atom(VarSymbol).
```

Ovvero la query seguente è verificata.

```
?- m(_, _, [v(2, x), v(2, y)] = m(_, _, VPs),  
|   foreach(member(VP, VPs), is_varpower(VP)).  
true
```

I polinomi sono rappresentati da termini più semplici.

```
poly(Monomials)
```

dove `Monomials` è una lista di monomi. Ovvero possiamo scrivere:

```
is_polynomial(poly(Monomials)) :-  
    is_list(Monomials),  
    foreach(member(M, Monomials), is_monomial(M)).
```

---

<sup>2</sup>Non è necessario che usiate *esattamente* questo ed altri predicati o funzioni presentati nel testo.

**Common Lisp.** I monomi devono essere rappresentati (analogamente al caso del Prolog) con oggetti siffatti:

```
(m coefficient total-degree vars-n-powers)
```

per i quali si può scrivere il predicato:

```
(defun is-monomial (m)
  (and (listp m)
        (eq 'm (first m))
        (let ((mtd (monomial-total-degree m))
              (vps (monomial-vars-and-powers m))
              )
          (and (integerp mtd)
                (>= mtd 0)
                (listp vps)
                (every #'is-varpower vps))))))
```

Sorvoliamo anche in questo caso su come controllare *coefficient*. La lista *vars-n-powers* contiene termini come il seguente:

```
(v power var-symbol)
```

per i quali possiamo scrivere il predicato:

```
(defun is-varpower(vp)
  (and (listp vp)
        (eq 'v (first vp))
        (let ((p (varpower-power vp))
              (v (varpower-symbol vp))
              )
          (and (integerp p)
                (>= p 0)
                (symbolp v))))))
```

Anche nel caso Common Lisp, i polinomi sono rappresentati da termini più semplici.

```
(poly monomials)
```

Ovvero possiamo scrivere:

```
(defun is-polynomial (p)
  (and (listp p)
        (eq 'poly (first p))
        (let ((ms (monomials p)))
          (and (listp ms)
                (every #'is-monomial ms))))))
```

(Naturalmente possiamo aggiungere altri test di consistenza delle strutture dati).

**Rappresentazioni dello zero.** Il monomio pari al numero 0 è rappresentato con *m*(0, 0, []) in Prolog e con (M 0 0 ()) in Common Lisp. Le vostre implementazioni devono normalizzare lo zero a questi casi. Naturalmente, anche il polinomio *poly*([]) in Prolog e il polinomio (POLY ()) in Common Lisp sono rappresentazioni dello 0.

**Note.** Ripetiamo! I predicati e le funzioni riportate nel testo sono solo *esempi*, non necessariamente completi. Le vostre versioni possono essere diverse e tener conto di più casi.

## 2.1 Operazioni da implementare

Le vostre librerie dovranno implementare le operazioni seguenti. Notate che ci si aspetta che i vostri algoritmi ritornino sempre monomi e polinomi “minimizzati”. Ad esempio,  $2x + y^3 - 2x \Rightarrow y^3$ .

**Prolog.** I predicati che dovrete implementare (oltre a quelli descritti sopra) servono a ispezionare le strutture dati e a fare calcoli simbolici con i polinomi.

***Predicate is\_zero( $X$ )***

Il predicato `is_zero` è vero quanto  $X$  è una rappresentazione dello 0 (incluso, ovviamente il caso in cui sia proprio 0).

***Predicate coefficients( $Poly$ ,  $Coefficients$ )***

Il predicato `coefficients` è vero quando  $Coefficients$  è una lista dei – ovviamente – coefficienti di  $Poly$ .

***Predicate variables( $Poly$ ,  $Variables$ )***

Il predicato `variables` è vero quando  $Variables$  è una lista dei simboli di variabile che appaiono in  $Poly$ .

***Predicate monomials( $Poly$ ,  $Monomials$ )***

Il predicato `monomials` è vero quando  $Monomials$  è la lista – *ordinata*, si veda sotto – dei monomi che appaiono in  $Poly$ .

***Predicate max\_degree( $Poly$ ,  $Degree$ )***

Il predicato `max_degree` è vero quando  $Degree$  è il massimo grado dei monomi che appaiono in  $Poly$ .

***Predicate min\_degree( $Poly$ ,  $Degree$ )***

Il predicato `min_degree` è vero quando  $Degree$  è il minimo grado dei monomi che appaiono in  $Poly$ .

***Predicate poly\_plus( $Poly1$ ,  $Poly2$ ,  $Result$ )***

Il predicato `poly_plus` è vero quando  $Result$  è il polinomio somma di  $Poly1$  e  $Poly2$ .

***Predicate poly\_minus( $Poly1$ ,  $Poly2$ ,  $Result$ )***

Il predicato `poly_minus` è vero quando  $Result$  è il polinomio differenza di  $Poly1$  e  $Poly2$ .

***Predicate poly\_times( $Poly1$ ,  $Poly2$ ,  $Result$ )***

Il predicato `poly_times` è vero quando  $Result$  è il polinomio risultante dalla moltiplicazione di  $Poly1$  e  $Poly2$ .

***Predicate as\_monomial( $Expression$ ,  $Monomial$ )***

Il predicato `as_monomial` è vero quando  $Monomial$  è il termine che rappresenta il monomio risultante dal “parsing” dell’espressione  $Expression$ ; il monomio risultante deve essere appropriatamente ordinato (si veda sotto).

***Predicate as\_polynomial( $Expression$ ,  $Polynomial$ )***

Il predicato `as_polynomial` è vero quando  $Polynomial$  è il termine che rappresenta il polinomio risultante dal “parsing” dell’espressione  $Expression$ ; il polinomio risultante deve essere appropriatamente ordinato (si veda sotto).

***Predicate poly\_val( $Polynomial$ ,  $VariableValues$ ,  $Value$ )***

Il predicato `poly_val` è vero quanto  $Value$  contiene il valore del polinomio  $Polynomial$  (che può anche essere un monomio), nel punto  $n$ -dimensionale rappresentato dalla lista  $VariableValues$ , che contiene un valore per ogni variabile ottenuta con il predicato `variables/2`.

***Predicate* pprint\_polynomial(*Polynomial*)**

Il predicato `pprint_polynomial` risulta vero dopo aver stampato (sullo “standard output”) una rappresentazione **tradizionale** del termine polinomio associato a *Polynomial*. Si può omettere il simbolo di moltiplicazione.

**Common Lisp.** Le funzioni che dovreste implementare (oltre a quelle descritti sopra) servono a ispezionare le strutture dati e a fare calcoli simbolici con i polinomi.

Si noti che in **Common Lisp** sarà necessario costruire anche delle funzioni che servono ad estrarre parti delle varie strutture dati che rappresentano monomi e polinomi. In **Prolog** possiamo usare l’unificazione per ottenere questo risultato, in **Common Lisp** no<sup>3</sup>.

***Function* is-zero *X* → *Result***

La funzione ritorna `T` come *Result*, quando *X* è una rappresentazione dello 0 (incluso, ovviamente il caso in cui sia proprio 0).

***Function* var-powers *Monomial* → *VP-list***

Data una struttura *Monomial*, ritorna la lista di *varpowers VP-list*.

***Function* vars-of *Monomial* → *Variables***

Data una struttura *Monomial*, ritorna la lista di variabili *Variables*.

***Function* monomial-degree *Monomial* → *TotalDegree***

Data una struttura *Monomial*, ritorna il suo grado totale *TotalDegree*.

***Function* monomial-coefficient *Monomial* → *Coefficient***

Data una struttura *Monomial*, ritorna il suo coefficiente *Coefficient*.

***Function* coefficients *Poly* → *Coefficients***

La funzione `coefficients` ritorna una lista *Coefficients* dei – ovviamente – coefficienti di *Poly*.

***Function* variables *Poly* → *Variables***

La funzione `variables` ritorna una lista *Variables* dei simboli di variabile che appaiono in *Poly*.

***Function* monomials *Poly* → *Monomials***

La funzione `monomials` ritorna la lista – *ordinata*, si veda sotto – dei monomi che appaiono in *Poly*.

***Function* max-degree *Poly* → *Degree***

La funzione `max-degree` ritorna il massimo grado dei monomi che appaiono in *Poly*.

***Function* min-degree *Poly* → *Degree***

La funzione `min-degree` ritorna il minimo grado dei monomi che appaiono in *Poly*.

***Function* poly-plus *Poly1 Poly2* → *Result***

La funzione `poly-plus` produce il polinomio somma di *Poly1* e *Poly2*.

---

<sup>3</sup>A meno di implementare un “unificatore” per CL, ovviamente.

**Function** *poly-minus Poly1 Poly2*  $\rightarrow$  *Result*

La funzione *poly-minus* produce il polinomio differenza di *Poly1* e *Poly2*.

**Function** *poly-times Poly1 Poly2*  $\rightarrow$  *Result*

La funzione *poly-times* ritorna il polinomio risultante dalla moltiplicazione di *Poly1* e *Poly2*.

**Function** *as-monomial Expression*  $\rightarrow$  *Monomial*

La funzione *as-monomial* ritorna la struttura dati (lista) che rappresenta il monomio risultante dal “parsing” dell’espressione *Expression*; il monomio risultante deve essere appropriatamente ordinato (si veda sotto)

**Function** *as-polynomial Expression*  $\rightarrow$  *Polynomial*

La funzione *as-polynomial* ritorna la struttura dati (lista) che rappresenta il monomio risultante dal “parsing” dell’espressione *Expression*; il polinomio risultante deve essere appropriatamente ordinato (si veda sotto).

**Function** *poly-val Polynomial VariableValues*  $\rightarrow$  *Value*

La funzione *poly-val* restituisce il valore *Value* del polinomio *Polynomial* (che può anche essere un monomio), nel punto *n*-dimensionale rappresentato dalla lista *VariableValues*, che contiene un valore per ogni variabile ottenuta con la funzione *variables*.

**Function** *pprint-polynomial Polynomial*  $\rightarrow$  *NIL*

La funzione *pprint-polynomial* ritorna NIL dopo aver stampato (sullo “standard output”) una rappresentazione **tradizionale** del termine polinomio associato a *Polynomial*. Si può omettere il simbolo di moltiplicazione.

### 3 Ordinamento di monomi e polinomi multivariati

Un polinomio univariato è normalmente scritto in ordine decrescente (o crescente) delle potenze della variabile. Ad esempio:

$$y^4 - 3y^2 - 42y + 123.$$

I monomi ed i polinomi multivariati possono essere invece scritti e “ordinati” in molti modi diversi; ognuno di questi ordinamenti ha una sua funzione in *Computer Algebra*. Per questo progetto dovrete implementare il seguente ordinamento.

**Ordinamento di un monomio.** Un monomio deve essere ordinato in *ordine lessicografico crescente* delle variabili. Ovvero:

$$y^{42}x^4sz^2t^2 \Rightarrow st^2x^4y^{42}z^2.$$

Si noti come l’esponente non modifichi l’ordinamento.

**Ordinamento di un polinomio.** Dato un insieme di monomi (ordinati), il polinomio risultante sarà ordinato prima in *ordine crescente del grado dei monomi* con spareggi determinati dalle variabili (questa è la ragione per tener traccia del grado complessivo di un monomio). Ad esempio:

$$y^4zx^5 - yzr + y^4rz^5 \Rightarrow -ryz + ry^4z^5 + x^5y^4z.$$

L’ordinamento di due monomi con le stesse variabili va fatto in modo *crescente* rispetto alle combinazioni variabile/esponente. Ad esempio:

$$ac + a^2 + ab + a \Rightarrow a + ab + ac + a^2.$$

Dove  $ab \prec a^2$ , dato che  $a \prec a^2$ .

## Indicazioni per l'implementazione

I vostri predicati `as_monomial`, `as_polynomial` e le vostre funzioni `as-monomial` e `as-polynomial` dovranno tenere presenti questi ordinamenti. I predicati di libreria SWI `sort/4` e `msort` serviranno a questa bisogna; lo stesso dicasi per la funzione Common Lisp `sort`<sup>4</sup>. La produzione di strutture dati che non rispettano questi ordinamenti risulterà in voti insufficienti.

## 4 “Parsing” di polinomi

I predicati `as_monomial`, `as_polynomial` e le funzioni `as-monomial` e `as-polynomial` si preoccupano di trasformare un monomio e un polinomio nella rappresentazione canonica interna. Il loro ruolo è quello di fare il *parsing* di una rappresentazione superficiale di monomi e polinomi. Queste rappresentazioni sono diverse per Prolog e Common Lisp.

**Prolog** In questo caso la rappresentazione superficiale di monomi e polinomi è quella normale, con moltiplicazioni e potenze esplicite. Per semplicità potete sempre aspettarvi di avere il coefficiente come primo elemento; il coefficiente 1 può sempre essere omissso. Ad esempio:

```
?- as_monomial(3 * y * w * t^3, M).
M = m(3, 5, [v(3, t), v(1, w), v(1, y)]).

?- as_monomial(y * s^3 * t^3, M).
M = m(1, 7, [v(3, s), v(3, t), v(1, y)]).

?- as_monomial(42, M).
M = m(42, 0, []).

?- as_polynomial(y * s^3 * t^3 - 4 + x * y, P).
P = poly([m(-4, 0, []),
          m(1, 2, [v(1, x), v(1, y)]),
          m(1, 7, [v(3, s), v(3, t), v(1, y)])])
```

(N.B. L'ultimo esempio è stato indentato manualmente per facilitare la lettura).

**Common Lisp** In questo secondo caso, la rappresentazione superficiale di monomi e polinomi utilizza la semplice sintassi prefissa Common Lisp. Anche in questo caso, potete sempre aspettarvi che il primo coefficiente di un monomio sia il primo elemento e che il coefficiente 1 può sempre essere omissso. Ad esempio:

```
cl-prompt> (as-monomial '(* 3 y w (expt t 3)))
(M 3 5 ((V 3 T) (V 1 W) (V 1 Y)))

cl-prompt> (as-monomial 42)
(M 42 0 NIL)

cl-prompt> (as-monomial '(* y (expt s 3) (expt t 3)))
(M 1 7 ((V 3 S) (V 3 T) (V 1 Y)))

cl-prompt> (as-polynomial '(+ (* y (expt s 3) (expt t 3)) -4 (* x y)))
(POLY ((M -4 0 NIL)
        (M 1 2 ((V 1 X) (V 1 Y)))
        (M 1 7 ((V 3 S) (V 3 T) (V 1 Y)))))
```

(N.B. L'ultimo esempio è stato indentato manualmente per facilitare la lettura).

---

<sup>4</sup>Attenzione che la `sort` di Common Lisp è una funzione cosiddetta “distruttiva”; è sempre bene *copiare* il suo input prima di invocarla.

Come potete notare i polinomi in sintassi Common Lisp sono molto semplici: hanno un + come operatore principale, `expt` per indicare le potenze e l'eventuale segno di sottrazione - è inglobato nel coefficiente. La sintassi superficiale dei polinomi è la seguente:

```
polynomial ::= monomial
            | '(' '+' monomial+ ')')

monomial   ::= number
            | variable
            | '(' '*' [coefficient] var-expt* ')

var-expt   ::= variable
            | '(' 'expt' variable exponent ')

variable   ::= symbol

exponent   ::= non-negative integer
```

## 5 Esempi

Questi sono alcuni esempi di come si può usare questa libreria. NB. Dovete naturalmente essere preparati a calcolare anche altri esempi.

### Common Lisp

```
cl-prompt> (setf qd (as-monomial 42))
(M 42 0 NIL)

cl-prompt> (setf m1 (as-monomial '(* y (expt s 3) (expt t 3))))
(M 1 7 ((V 3 S) (V 3 T) (V 1 Y)))

cl-prompt> (setf p1 (as-polynomial '(+ (* -1 x) (* x w))))
(POLY ((M -1 1 ((V 1 X))) (M 1 2 ((V 1 W) (V 1 X)))))

cl-prompt> (setf p2 (as-polynomial '(+ (* y (expt s 3) (expt t 3)) -4 (* x y))))
(POLY ((M -4 0 NIL) (M 1 2 ((V 1 X) (V 1 Y))) (M 1 7 ((V 3 S) (V 3 T) (V 1 Y)))))

cl-prompt> (poly-times m1 p1) ; Output formattato per leggibilità.
(POLY ((M -1 8 ((V 3 S) (V 3 T) (V 1 X) (V 1 Y)))
      (M 1 9 ((V 3 S) (V 3 T) (V 1 W) (V 1 X) (V 1 Y)))))

cl-prompt> (pprint-polynomial *)
-1 S^3 T^3 X Y + S^3 T^3 W X Y
NIL
```

### Prolog

```
?- as_monomial(42, QD).
QD = m(42, 0, []).

?- as_polynomial(-1 * x + x * y, P1).
P1 = poly([m(-1, 1, [v(1, x)]), m(1, 2, [v(1, x), v(1, y)])])

?- as_polynomial(-1 * x + x * y, P1), variables(P1, Vs).
P1 = poly([m(-1, 1, [v(1, x)]), m(1, 2, [v(1, x), v(1, y)])])
```



```

Vs = [x, y]

?- as_polynomial(y * s^3 * t^3 - 4 + x * y, P2).
%% Formattato per leggibilità.
P2 = poly([m(-4, 0, []),
           m(1, 2, [v(1, x), v(1, y)]),
           m(1, 7, [v(3, s), v(3, t), v(1, y)])]),

?- as_monomial(y * s^3 * t^3, M1),
| as_polynomial(-1 * x + x * y, P1),
| poly_times(M1, P1, R),
| pprint_polynomial(R).

-1 * S^3 * T^3 * X * Y + S^3 * T^3 * X * Y^2

M1 = m(1, 7, [v(3, s), v(3, t), v(1, y)]),
P1 = poly([m(-1, 1, [v(1, x)]), m(1, 2, [v(1, x), v(1, y)])]),
R = poly([m(-1, 8, [v(3, s) v(3, t) v(1, x) v(1, y)]),
          m(1, 9, [v(3, s), v(3, t), v(1, x), v(2, y)])]).

```

## 6 Suggerimenti

Si suggerisce di procedere inizialmente con la costruzione dei predicati `as_...` e delle funzioni `as-...` e con il predicato `pprint_polynomial` e la funzione `pprint-polynomial`, al fine di avere una base su cui poi costruire le operazioni successive. Le funzionalità di ordinamento di Prolog, (`sort`, `msort`, `predsort`, ...) e di Common Lisp (`sort`) sono senz'altro utili per il progetto. Per Prolog sarà utile anche `list_to_set`; per Common Lisp potrete anche considerare `remove-duplicates`.

## 7 Conclusioni

La libreria di funzioni che avrete costruito è un primo passo verso la costruzione di un sistema di *Computer Algebra* quali Mathematica<sup>TM</sup>, Maxima, Axiom etc.

La rappresentazione di polinomi e monomi non è necessariamente la migliore e sono molte le variazioni sul tema; lo scopo di questa rappresentazione è di coniugare semplicità e flessibilità, oltre ad essere facile da manipolare<sup>5</sup>. Qualora si vogliano fare operazioni più sofisticate sui polinomi, ad esempio, calcolare il gcd di due polinomi o calcolare una *base di Gröbner*, allora sarà necessario adottare delle rappresentazioni e degli ordinamenti diversi.

---

<sup>5</sup>Specie per il correttore.

## 8 Istruzioni...

LEGGERE ATTENTAMENTE LE ISTRUZIONI QUI SOTTO  
(IN ITALIANO!).

PRIMA DI CONSEGNARE, CONTROLLATE **ACCURATAMENTE**  
CHE TUTTO SIA NEL FORMATO E CON LA STRUTTURA DI CARTELLE  
RICHISTI.

### 8.1 Versioni Prolog e Common Lisp

Le versioni Prolog e Common Lisp che usiamo per la valutazione sono le più recenti pubblicate sui siti SWI Prolog (<https://www.swi-prolog.org/download/stable>) e Lispworks (<http://www.lispworks.com/downloads/index.html>).

### 8.2 Consegna

Dovete consegnare:

Uno .zip file dal nome `<Cognome>_<Nome>_<matricola>_mvpoly_LP_202306.zip` *che conterrà una cartella dal nome* `<Cognome>_<Nome>_<matricola>_mvpoly_LP_202306`.

Se il vostro nome e cognome sono: Gian Giacomo Pier Carl Luca Serbelloni Lupmann Vien Dal Mare, allora il nome del file sarà:

`Serbelloni_Lupmann_Vien_Dal_Mare_Gian_Giacomo_Pier_Carl_Luca_123456_mvpoly_LP_202306.zip`.

Inoltre...

- Nella cartella dovete avere due sottocartelle: una di nome `Lisp` e l'altra di nome `Prolog`.
- Nella directory `Lisp` dovete avere:
  - un file dal nome `mvpoly.lisp` che contiene il codice di della libreria.
    - \* Le prime linee del file **devono essere dei commenti con il seguente formato**, ovvero devono fornire le necessarie informazioni secondo le regole sulla collaborazione pubblicate su Moodle.

```
;;; -*- Mode: Lisp -*-  
;;; <Matricola> <Cognome> <Nome>  
;;; <eventuali collaborazioni>
```

Il contenuto del file deve essere ben commentato.
    - Un file `README` in cui si spiega come si possono usare le funzioni definite nel programma.
  - Nella directory `Prolog` dovete avere:
    - un file dal nome `mvpoly.pl` che contiene il codice di della libreria.
      - \* Le prime linee del file **devono essere dei commenti con il seguente formato**, ovvero devono fornire le necessarie informazioni secondo le regole sulla collaborazione pubblicate su Moodle.

```
%%% -*- Mode: Prolog -*-  
%%% <Matricola> <Cognome> <Nome>  
%%% <eventuali collaborazioni>
```

Il contenuto del file deve essere ben commentato.
      - Un file `README` (si! Anche qui anche se è una ripetizione) in cui si spiega come si possono usare i predicati definiti nel programma.

**ATTENZIONE!** Consegnate solo dei files e directories con nomi costruiti come spiegato. Niente spazi extra e soprattutto niente **.rar** or **.7z** o **.tgz** – solo **.zip**!

**Repetita juvant! NON CONSEGNARE FILES .rar!!!!**

Nel caso non si sia capito: **NON CONSEGNARE FILES .rar!!!!**

**Esempio:**

File **.zip**:

Antoniotti\_Marco\_424242\_mvpoly\_LP\_202306.zip

Che contiene:

```
prompt$ unzip -l Antoniotti_Marco_424242_mvpoly_LP_202306.zip
```

```
Archive:  Antoniotti_Marco_424242_mvpoly_LP_202306.zip
```

Length	Date	Time	Name
0	12-02-23	09:59	Antoniotti_Marco_424242_mvpoly_LP_202306/
0	12-04-23	09:55	Antoniotti_Marco_424242_mvpoly_LP_202306/Lisp/
4783	12-04-23	09:51	Antoniotti_Marco_424242_mvpoly_LP_202306/Lisp/mvpoly.lisp
10598	12-04-23	09:53	Antoniotti_Marco_424242_mvpoly_LP_202306/Lisp/README.txt
0	12-04-23	09:55	Antoniotti_Marco_424242_mvpoly_LP_202306/Prolog/
4623	12-04-23	09:51	Antoniotti_Marco_424242_mvpoly_LP_202306/Prolog/mvpoly.pl
10622	12-04-23	09:53	Antoniotti_Marco_424242_mvpoly_LP_202306/Prolog/README.txt
30626			7 files

### 8.3 Valutazione

Il programma sarà valutato sulla base di una serie di test standard. In particolare si valuterà la copertura e correttezza delle operazione di base sui polinomi.