Progetto di Programmazione 2018
Solve

v1.0.0

December 12, 2018

Abstract

Work in progress. In questo documento vi verrano spiegate le specifiche del progetto *Solve* per l'esame di Introduzione alla Programmazione 2018. Questo documento contiene un ripasso degli argomenti che vi serviranno per svolgere il progetto più le specifiche di implementazione e valutazione del codice. Vi consigliamo di **controllare regolarmente** la **versione** e la **data** di aggiornamento di questo documento in quanto sarà soggetto a revisioni future al fine di rendere più chiare le specifiche del progetto .

Indice

1 Introduzione

Il progetto consiste nell'implementazione di un programma in grado di **risolvere equazioni** di grado non superiore al secondo, di **semplificare**, **normalizzare** e **derivare** polinomi. Vi viene richiesto di **completare** l'implementazione di un programma che è gia in grado di leggere da terminale i comandi dell'utente ma che non è in grado di riconoscerli o di svolgere le operazioni sui polinomi. Il vostro compito è quello di completare l'implementazione andando a modificare solamente i file **Main.fs** e **Impl.fs**, qualsiasi codice scritto al di fuori di questi due file non verrà considerato al fine della valutazione del progetto.

2 Indicazioni generali

- Si accettano gruppi di massimo **3** persone.
- Leggere con attenzione le istruzioni di consegna che trovate nella home del corso, progetti **consegnati in modo errato** verrano considerati **insufficienti**.
- Progetti **non compilanti** verranno considerati **insufficienti**.
- **Non** è consentito cambiare la struttura del progetto, è obbligatorio che il progetto compili nella sua interezza, potete modificare i file **Main.fs** e **Impl.fs**.

3 Plagio

I progetti verranno **controllati** al fine di verificare che non vi sia stato plagio tra consegne di gruppi differenti. **Non vogliamo ostacolare** il normale **scambio di idee tra gruppi e l'aiuto reciproco** ma nel momento in cui vi mettete a scrivere il vostro codice dovete farlo in **totale indipendenza** dagli altri gruppi. In caso di plagio si considera **annullato l'esame di laboratorio** per **tutti i componenti dei gruppi coinvolti**, questo indipendentemente da chi abbia copiato il progetto di chi, **tutti gli studenti coinvolti sono ritenuti ugualmente responsabili del plagio**.

3.1 Sul serio non copiate

In modo meno formale e più personale vi consiglio vivamente di non copiare, cercate di essere sicuri che il vostro progetto non possa essere ricondotto a quello di altri gruppi, vi assicuro che se succede abbiamo una precisione del 99% nel riconoscere il plagio dalla semplice casualità. Se scopriamo un plagio siamo costretti a intraprendere una serie di azioni che dispiacciono tanto a

voi quanto a noi (credetemi non lo dico per semplice retorica), il professore di laboratorio può decidere di annullarvi l'esame e l'universita può bloccare la vostra carriera accademica per un anno intero. Se avete problemi nello svolgimento del progetto venite a chiedere aiuto al professore, ai tutor o agli assistenti, non ci facciamo problemi ad aiutarvi se dimostrate interesse e impegno ed è più sicuro che copiare codice scritto da altri che non sareste in grado di spiegare all'orale (inoltre è gratis e siamo tutti molto simpatici).

4 Elementi di valutazione

Sebbene il progetto sia un lavoro che può essere svolto in gruppo, la **valutazione** sarà **individuale**. Oltre alla discussione orale, saranno valutate:

- qualità del codice: oltre alla correttezza, saranno considerate l'eleganza della soluzione implementata, l'utilizzo di indentazione corretta e consistente, la suddivisione in sottofunzioni, etc;
- qualità della **documentazione**: codice **non commentato** non sarà valutato;

5 Indicazioni per la consegna

Il progetto è contenuto nella cartella SharpSolver contenuta nell'omonimo zip. Vi viene richiesto di consegnare un file zip SharpSolver contenente la cartella SharpSolver che contiene il vostro progetto, essenzialmente dovete consegnare lo stesso zip che scaricate in cui i file Main.fs e il file Impl.fs sono stati implementati da voi.

6 Polinomi

Il progetto di quest'anno è fortemente inspirato a concetti di matematica: polinomi, equazioni, derivate ed altri fondamenti dell'algebra. In questa sezione cercheremo di presentarvi velocemente alcune nozioni fondamentali legate ai polinomi. I concetti presentati dovrebbero essere già stati trattati esaustivamente nei corsi di Calcolo e Matematica del primo anno, per questo motivo in questa sezione proponiamo un veloce ripasso dei concetti strettamente necessari allo svolgimento del progetto. Nel caso in cui le spiegazioni riportate non fossero sufficienti vi consigliamo di leggere con attenzione i materiali proposti nelle **References** che trovate alla fine di questo documento.

- In matematica un **monomio** è un'espressione algebrica costituita da

un coefficiente ed una parte letterale dove tra le lettere compaiono moltiplicazioni e elevamenti a potenza aventi esponente naturale;

- In matematica un **polinomio** [1] è un'espressione composta da costanti e variabili combinate usando soltanto addizione, sottrazione e moltiplicazione. In altre parole, un polinomio *tipico*, cioè *ridotto in forma normale*, è la somma algebrica di **monomi** non simili tra loro, vale a dire con parti letterali e grado diversi;

In questo progetto tratteremo sempre e solo **polinomi con 1 sola variabile** x.

Polinomi in 1 sola variabile, d'altronde, possono essere visti come **funzioni** in 1 variabile - ad esempio, il seguente polinomio può essere rappresentato da una funzione $P(x)$:

$$P(x) \quad = \quad 7x^2 + 5x - 3$$

L'applicazione di un argomento alla funzione $P(x)$ produrrebbe la sostituzione di tutte le occorrenze di x nell'espressione del polinomio a destra del simbolo di uguale (=). Quale significato hanno dunque le **radici**? Le radici di un polinomio sono quei valori che, sostituiti alle variabili, riducono l'espressione polinomiale a **0**. Il che equivale a risolvere la seguente equazione:

$$7x^2 + 5x - 3 \quad = \quad 0$$

Oppure, in maniera equivalente, l'insieme delle soluzioni può essere definito come $S = \{\ x \mid P(x) = 0\ \}$.

Per **polinomio in forma normale** (anche detto *polinomio normalizzato*) si intende un polinomio formato da **monomi** che hanno tutti parte letterale e grado diversi e compaiono nel polinomio in ordine **decrescente** di grado. Nel nostro caso, trattando polinomi in una sola variabile, la cosa si riduce alla diversità del grado.

Mostriamo ad esempio un polinomio *non* normalizzato di grado 3:

$$2x + 4x^2 + 3 - x + x^3 + 6x^2 - 1\,.$$

La normalizzazione consiste nel raccogliere i coefficienti di tutti i termini **simili** (cioè aventi parte letterale di grado uguale nel nostro caso) e sommarli:

$$(2-1)x + (4+6)x^2 + x^3 + 3 - 1\,.$$

Riducendo le somme e riordinando i monomi in ordine descrescente per grado otteniamo:

$$x^3 + 10x^2 + x + 2\,.$$

6.1 Equazioni di grado zero

Possiamo intendere come equazione di grado 0 una equazione in cui non compare nessuna incognita, cioè una uguaglianza tra due espressioni riducibili a costanti numeriche. Risolvere una equazione di grado 0 significa di fatto *verificare una identità*, pertanto si tratta di una operazione avente un risultato booleano - una identità può essere vera o falsa, ad esempio:

$$\begin{aligned} 5+3 &= 7 \\ 5+3-7 &= 0 \\ -2 &= 0 \end{aligned}$$

Dopo la normalizzazione, spostando il polinomio di destra sul lato sinistro monomio per monomio[1], l'equazione si riduce ad una costante confrontata con zero. L'identità $c = 0$ è verificata se c è effettivamente uguale a 0; falsa altrimenti.

6.2 Equazioni di primo grado

Le equazioni di **primo grado** [2] in 1 incognita sono equazioni in cui l'incognita x è elevata a esponente 1 e può essere solamente moltiplicata o sommata a costanti numeriche o espressioni comunque costanti - espressioni in cui, in altre parole, non compare la x. Un'equazione di primo grado è un'uguaglianza tra due polinomi che, una volta ridotti in forma normale e raccolti da un solo lato, riducono l'equazione ad una uguaglianza tra un polinomio e 0.

$$\begin{aligned} 4x+5 &= 2x \\ 4x+5-2x &= 0 \\ 6x+5 &= 0 \\ x &= -\tfrac{5}{6}\,. \end{aligned}$$

La **normalizzazione di una equazione**, dunque, è una operazione che da 2 polinomi $P_1(x)$ e $P_2(x)$ produce 1 solo polinomio $P'(x)$, il quale è la normalizzazione della differenza di polinomi $P_1(x) - P_2(x)$. Trovare le soluzioni dell'equazione significa trovare le radici di $P'(x)$. La procedura è rappresentata dai seguenti passaggi:

$$\begin{aligned} P_1(x) &= P_2(x) && \texttt{forma iniziale} \\ P_1(x) - P_2(x) &= 0 && \texttt{raccogliamo a sinistra} \\ P'(x) &= 0 && \texttt{equazione normalizzata} \end{aligned}$$

[1] Sebbene appaiano come costanti numeriche, dal punto di vista sintattico sono monomi di grado 0.

Per trovare le soluzioni dell'equazione, ovvero le radici di $P'(x)$, è necessario isolare la variabile x tramite manipolazione di termini:

$$bx + c = 0$$

$$x = -\frac{c}{b}\,.$$

6.3 Equazioni di secondo grado

Un'equazione di **secondo grado** [3][4] è un'equazione polinomiale in cui l'incognita compare almeno una volta con esponente di grado 2 e tale che l'incognita compaia con grado massimo pari a 2.

$$4x^2 + 5 = 2x\,.$$

La forma normale di una equazione di secondo grado è definita come segue:

$$ax^2 + bx + c = 0\,.$$

La risoluzione delle equazioni di secondo grado richiede il calcolo del discriminante:

$$\Delta = b^2 - 4ac\,.$$

Il segno del discriminante ci permette di capire la molteplicità e il tipo delle soluzioni dell'equazione di secondo grado:

- Se il discriminante ha valore **positivo**, l'equazione ha due soluzioni reali distinte.
- Se il discriminante ha valore **nullo**, l'equazione ha due soluzioni reali coincidenti.
- Se il discriminante ha valore **negativo**, l'equazione non ammette soluzioni reali.

Il discriminante ci permette di scrivere la formula risolutiva:

$$x_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a}\,.$$

6.4 Derivazione di un polinomio

In matematica, la **derivata** [5] è la misura di quanto la crescita di una funzione cambi al variare del suo argomento.

La derivata di una funzione è una grandezza puntuale, cioè si calcola punto per punto. Nel caso di funzioni a una variabile nel campo reale, essa è la pendenza della tangente al grafico della funzione in quel punto e ne

rappresenta la migliore approssimazione lineare. Nel caso in cui la derivata esista (cioè la funzione sia derivabile) in ogni punto del dominio, la si può vedere a sua volta come una funzione che associa a ogni punto proprio la derivata in quel punto.

Poiché un polinomio è la somma di uno o più monomi, per calcolare la sua derivata è sufficiente sommare la derivata di ogni monomio. La derivata di un monomio è data dal prodotto tra il suo esponente, ad esempio:

$$D[2x^4] = 4(2x^{4-1}) = 8x^3\,.$$

In generale possiamo scrivere la formula di **derivazione di un monomio** come segue:

$$\begin{array}{lcll} D[cx^n] & = & (n \cdot c)x^{n-1} & \texttt{grado } n >= 1 \\ D[cx^0] & = & D[c] = 0 & \texttt{costante} \end{array}$$

Per calcolare la derivata di un polinomio è sufficiente sommare le derivate dei monomi che lo costituiscono. Ricordiamo che una costante può essere vista come un monomio di grado 0 dove il coefficiente è la costante:

$$D[c_0 + c_1 x \;..\; c_n x^n] = D[c_0 x^0 + c_1 x^1 \;..\; c_n x^n]$$

La derivata di un polinomio può quindi essere espressa come sommatoria di monomi derivati:

$$D\left[\sum_{i=0}^{n} c_i x^i\right] = \sum_{i=0}^{n} D[c_i x^i]$$

Ad esempio:

$$D[2x^2 + 3x + 5] = 2(2x^{2-1}) + 1(3x^{1-1}) + 0 = 4x + 3$$

Altro esempio:

$$D[2x^4 + 3x^3 - 5x^2] = 4(2x^{4-1}) + 3(3x^{3-1}) + 2(-5x^{2-1}) = 8x^3 + 9x^2 - 10x\,.$$

Da notare che la derivata di un polinomio di grado n è un polinomio di grado $n-1$, dove il lower bound $n >= 0$ garantisce che la derivata di una costante (cioè di un monomio di grado 0) sia 0 e non un monomio di grado -1. Virtualmente, applicando la regola generale, un monomio di grado 0 dovrebbe produrre una frazione: $D[3] = D[3x^0] = (-1 \cdot 3)x^{-1} = -\frac{3}{x}$.

7 Specifica del progetto

Il progetto consiste nell'implementare un programma in grado di manipolare polinomi e compiere alcune operazioni su di essi. Tale programma si

comporta come un interprete di comandi, o una console interattiva, in cui l'utente può inserire da tastiera espressioni polinomiali che rispettano una certa sintassi ed il programma scrive in output il risultato.

7.0.1 Moduli

Forniamo già il progetto F# per Visual Studio 2017 con lo scheletro del programma e i tipi di dato principali già definiti. Il codice è suddiviso in **moduli**, ciascuno dei quali ha il proprio file sorgente. Ogni sorgente dichiara in alto il nome del modulo, ad esempio la dichiarazione `module Solve.Absyn` in testa al sorgente `Absyn.fs` definisce il modulo `Absyn` all'interno del *namespace* `Solve`, in modo del tutto simile ai percorsi di file e cartelle nel filesystem.

Segue una descrizione di ciascun modulo e di ciò che contiene:

Prelude Raccoglie tipi e funzioni globali utili a tutto il programma. La definizione del tipo `rational` è qui: rappresentiamo i numeri razionali (l'insieme $\mathbb{Q}$ in matematica) e le loro operazioni. Abbiamo definito un numero razionale come una frazione tra interi, quindi come coppia (n, d) dove n è il numeratore e d è il denominatore. Il tipo `rational` supporta gli operatori di somma `(+)`, sottrazione `(-)`, moltiplicazione `(*)`, divisione `(/)` e negazione unaria `(~-)`, oltre a svarianti altri metodi[2] come la radice quadrata `(Sqrt)`, il valore assoluto `(Abs)`, la potenza `(Pow)` e gli operatori di confronto ed uguaglianza.

Config Contiene definizioni di costanti globali utili all'intero programma, come stringhe di testo e valori numerici di configurazione.

Absyn Definisce i tipi che rappresentano la **sintassi astratta** (***ab**stract **syn**tax* in inglese) le entità sintattiche che il programma manipola: i tipi `monomial` e `polynomial` sono definiti qui come *tipi unione aventi un solo costruttore*, chiamati anche tipi *heavyweight*.

Parser Il file `Parser.fs` con il modulo `Parser` è generato automaticamente da un programma esterno che Visual Studio chiama automaticamente durante la compilazione. Si tratta di `fsyacc`, il porting per F# del celebre generatore di parser *Yacc*. Il file `Parser.fsy` funge da input per `fsyacc`, specificando la *grammatica* che il parser dovrà rispettare e le regole per costruire le strutture dati definite in `Absyn.fsy` opportunamente. La sintassi in notazione BNF è riportata in forma semplificata nella tabella 1[3].

[2] Si chiamano *metodi* le funzioni definite come *membri* di un tipo.

[3] Per approfondire le grammatiche formali si consulti [6], per la notazione EBNF [7] e per il parsing in generale [8].

Table 1: Sintassi semplificata.

L	$\rightarrow$		**linea**
		$\underline{\texttt{\#}}\text{s}$	comando
	$\mid$	E	espressione
	$\mid$	$E \underline{=} E$	equazione
E	$\rightarrow$		**espressioni**
		P	polinomio
	$\mid$	$\underline{\texttt{D[}}E\underline{\texttt{]}}$	derivata
P	$\rightarrow$		**polinomi**
		M	monomio
	$\mid$	$P + M$	somme di monomi
M	$\rightarrow$		**monomi**
		c	costante
	$\mid$	$c\underline{\texttt{x}}^n$	grado

dove s è una stringa, $c \in \mathbb{Q}$ e $n \geqslant 1$.

Lexer Il lexer è generato anch'esso da un programma esterno invocato durante la compilazione - `fslex`, il generatore di lexer di F#. Similmente al parser, nel file `Lexer.fsl` compare la specifica delle regole lessicali che il programma deve riconoscere. Intuitivamente, si tratta delle keyword, le parentesi, i caratteri speciali come gli operatori e le costanti numeriche.

Main contiene la funzione *main* e le funzioni principali che si occupano dell'interprete di comandi.

L'ordine dei file sorgente in un progetto F# conta: `Main.fs` è l'ultimo perché deve poter vedere tutte le definizioni (tipi e funzioni) degli altri moduli; in generale ogni sorgente deve stare *sotto* i sorgenti di cui utilizza i nomi (di funzioni o di tipi), allo stesso modo in cui, all'interno di un singolo sorgente F#, ogni let-binding deve stare sotto i let-binding a cui fa riferimento.

7.1 Sintassi e tipi di dato

La nostra rappresentazione di monomi e polinomi è **sintattica**, non numerica. Questo significa che le strutture dati che modellano monomi e polinomi non sono valori numerici, ma tipi di dato appositamente definiti. Ad esempio, per rappresentare un monomio cx^n è necessario conservare 2 informazioni: il coefficiente c ed il grado n; la variabile x non fa parte della rappresentazione di un monomio perché, nel nostro caso, non permettiamo all'utente di usare variabili diverse da x - se, ipoteticamente, avessimo permesso all'utente di scrivere polinomi con nomi di variabili qualunque, la stringa col nome della variabile sarebbe stata un'informazione importante da conservare nella struttura dati che rappresenta un monomio.

Possiamo, in sintesi, rappresentare un monomio come una coppia (c, n) dove $c \in \mathbb{Q}$ e $n \in \mathbb{N}$. Anziché definire i monomi come un *type alias* in F#:

```
type monomial = rational * int  // poco sicuro
```

abbiamo definito un nuovo tipo di dato con un costruttore:

```
type monomial = Monomial of rational * int
```

La differenza è che la prima definizione non introduce un vero nuovo tipo per i monomi ma stiamo solamente dicendo al compilatore che la coppia `rational * int` ha un nome. Questo è profondamente diverso da definire un *nuovo* tipo i cui valori necessitano di una *costruzione esplicita* e che non è equivalente ad una coppia. Per costruire un valore di tipo `monomial` è quindi necessario applicare il costruttore `Monomial` ad una coppia di tipo `rational * int`, alla stessa maniera in cui, in generale, si costruiscono valori di *tipi unione* (anche detti *disjoint union*, *discriminated union* oppure *variant*), tuttavia con un solo costruttore nel nostro caso.

Immaginiamo di voler costruire il monomio $3x^2$ in F#:

```
let m1 = (3Q, 2)           // m1 e' solo una coppia
let m2 = Monomial (3Q, 2) // m2 e' un monomial
```

7.2 Input del programma

Il programma accetta in input numeri, monomi, polinomi ed equazioni; è inoltre possibile derivare numeri, monomi e polinomi. L'operazione di derivazione è ricorsiva, è quindi possibile derivare una derivata se questa dà luogo ad un polinomio valido. Non è possibile sommare numeri, monomi, polinomi o derivate ad una derivata ma è possibile avere derivare liberamente entrambi i lati di un'equazione. Per maggiori dettagli ed esempi di input vi consigliamo di visualizzare il manuale interno al programma scrivendo il comando `#help` nella console.

7.3 Tipi di dato

Nel progetto sono state definite alcune strutture dati che dovrete utilizzare, diamo la descrizione di quelle più importanti:

- **rational** rappresenta un numero razionale, è una coppia costituita da un intero n e un bigint d dove n è il numeratore e d è il denominatore del numero razionale (lo zero razionale è rappresentato dal valore **0Q**)
- **monomial** rappresenta un monomio, è formato da una coppia (q, n) costituita da un *rational* q, che rappresenta il coefficiente del monomio, e da un intero n, che rappresenta il grado del monomio
- **polynomial** rappresenta un polinomio, è una lista di monomi
- **normalized_polynomial** rappresenta un polinomio normalizzato, è un vettore di *rational* dove ogni elemento rappresenta il coefficiente di un monomio e la posizione relativa dell'elemento rappresenta il grado dell'incognita del monomio corrispondente.

7.4 Cosa implementare

Oltre al main del programma vi viene richiesto di implementare le seguenti funzioni che trovate nel file Impl.fs:

```
val rationalize : float -> rational

val monomial_degree : monomial -> int
val monomial_negate : monomial -> monomial
val polynomial_degree : polynomial -> int
val polynomial_negate : polynomial -> polynomial
```

```
val normalized_polynomial_degree :
    normalized_polynomial -> int

val normalize : polynomial -> normalized_polynomial
val reduce : expr -> polynomial
val derive : polynomial -> polynomial

val solve0 : normalized_polynomial -> bool
val solve1 : normalized_polynomial -> rational
val solve2 : normalized_polynomial -> (float * float
    option) option
```

7.4.1 rationalize

```
val rationalize : float -> rational
```

Questa funzione prende in input un numero float e restituisce la sua rappresentazione in numero razionale.

7.4.2 monomial_degree

```
val monomial_degree : monomial -> int
```

Questa funzione restituisce il grado del monomio dato in input.

7.4.3 monomial_negate

```
val monomial_negate : monomial -> monomial
```

Questa funzione prende in input un monomio e restituisce il monomio cambiato di segno.

7.4.4 polynomial_degree

```
val polynomial_degree : polynomial -> int
```

Questa funzione prende in input un polinomio e restituisce il suo grado (ricordiamo che il grado di un polinomio corrisponde al grado del monomio di grado più alto).

7.4.5 polynomial_negate

```
val polynomial_negate : polynomial -> polynomial
```

Questa funzione prende in input un polinomio e restituisce il polinomio negato (il polinomio con tutti i monomi cambiati di segno).

7.4.6 normalized_polynomial_degree

```
val normalized_polynomial_degree :
    normalized_polynomial -> int
```

Questa funzione prende in input un polinomio normalizzato e restituisce il suo grado.

7.4.7 normalize

```
val normalize : polynomial -> normalized_polynomial
```

Questa funzione prende in input un polinomio e restituisce il suo corrispondente polinomio normalizzato.

7.4.8 reduce

```
val reduce : expr -> polynomial
```

Questa funzione prende in input un'espressione e la semplifica fino a restituire il polinomio da essa rappresentato.

7.4.9 derive

```
val derive : polynomial -> polynomial
```

Questa funzione prende in input un polinomio e restituisce la sua derivata.

7.4.10 solve0

```
val solve0 : normalized_polynomial -> bool
```

Questa funzione prende in input un polinomio e risolve l'equazione di grado zero che si ottiene eguagliando il polinomio dato a zero.

7.4.11 solve1

```
val solve1 : normalized_polynomial -> rational
```

Questa funzione prende in input un polinomio e risolve l'equazione di primo grado che si ottiene eguagliando il polinomio dato a zero.

7.4.12 solve2

```
val solve2 : normalized_polynomial -> (float * float
   option) option
```

Questa funzione prende in input un polinomio e risolve l'equazione di secondo grado che si ottiene eguagliando il polinomio dato a zero. La funzione può restituire zero, una o due soluzioni in base al numero di soluzioni dell'equazione fornita.

8 Esempi di input e output

```
>> 5
[absyn]     Expr (Poly (Polynomial [Monomial (5,0)]))
[pretty]    5
[derived]   5
```

```
[norm]      5
[degree]    0


>> 5+3
[absyn]     Expr (Poly (Polynomial [Monomial (5,0);
   Monomial (3,0)]))
[pretty]    5 + 3
[derived]   5 + 3
[norm]      8
[degree]    0


>> 5 + 2 = 7
[absyn]     Equ
  (Poly (Polynomial [Monomial (5,0); Monomial (2,0)]),
   Poly (Polynomial [Monomial (7,0)]))
[pretty]    5 + 2 = 7
[derived]   5 + 2 = 7
[norm]      0 = 0
[degree]    0
[identity]  true


>> 5 + 2 = 3
[absyn]     Equ
  (Poly (Polynomial [Monomial (5,0); Monomial (2,0)]),
   Poly (Polynomial [Monomial (3,0)]))
[pretty]    5 + 2 = 3
[derived]   5 + 2 = 3
[norm]      4 = 0
[degree]    0
[identity]  false

>> 5x
[absyn]     Expr (Poly (Polynomial [Monomial (5,1)]))
[pretty]    5x
[derived]   5x
[norm]      0 + 5x
[degree]    1


>> 5x + 3
[absyn]     Expr (Poly (Polynomial [Monomial (5,1);
   Monomial (3,0)]))
[pretty]    5x + 3
[derived]   5x + 3
[norm]      3 + 5x
[degree]    1
```

```
>> 5x + 3 = 0
[absyn]     Equ
  (Poly (Polynomial [Monomial (5,1); Monomial (3,0)]),
   Poly (Polynomial [Monomial (0,0)]))
[pretty]    5x + 3 = 0
[derived]   5x + 3 = 0
[norm]      3 + 5x = 0
[degree]    1
[sol]       x = -3/5


>> 5x^2 + 4x - 3
[absyn]     Expr (Poly (Polynomial [Monomial (5,2);
   Monomial (4,1); Monomial (-3,0)]))
[pretty]    5x^2 + 4x - 3
[derived]   5x^2 + 4x - 3
[norm]      -3 + 4x + 5x^2
[degree]    2


>> 5x^2 + 4x - 3 = 0
[absyn]     Equ
  (Poly (Polynomial [Monomial (5,2); Monomial (4,1);
     Monomial (-3,0)]),
   Poly (Polynomial [Monomial (0,0)]))
[pretty]    5x^2 + 4x - 3 = 0
[derived]   5x^2 + 4x - 3 = 0
[norm]      -3 + 4x + 5x^2 = 0
[degree]    2
[sol]       x1 = 0.471779788708135 vel x2 =
   -1.27177978870813


>> D[5x^2 + 4x - 3]
[absyn]     Expr
  (Derive (Poly (Polynomial [Monomial (5,2); Monomial
     (4,1); Monomial (-3,0)])))
[pretty]    D[5x^2 + 4x - 3]
[derived]   10x + 4
[norm]      4 + 10x
[degree]    1


>> D[5x^2 + 4x - 3] = 3x
[absyn]     Equ
  (Derive (Poly (Polynomial [Monomial (5,2); Monomial
     (4,1); Monomial (-3,0)])),
```

```
   Poly (Polynomial [Monomial (3,1)]))
[pretty]    D[5x^2 + 4x - 3] = 3x
[derived]   10x + 4 = 3x
[norm]      4 + 7x = 0
[degree]    1
[sol]       x = -4/7


>> D[5x^2 + 4x - 3] = D[3x]
[absyn]     Equ
  (Derive (Poly (Polynomial [Monomial (5,2); Monomial
     (4,1); Monomial (-3,0)])),
   Derive (Poly (Polynomial [Monomial (3,1)])))
[pretty]    D[5x^2 + 4x - 3] = D[3x]
[derived]   10x + 4 = 3
[norm]      1 + 10x = 0
[degree]    1
[sol]       x = -1/10


>> D[5x^2 + 4x - 3] + x
[error]     at characters 0-0: input "+": syntax error at
    token <PLUS>


>> D[D[5x^2 + 4x - 3]] = D[D[3x]]
[absyn]     Equ
  (Derive
     (Derive
        (Poly (Polynomial [Monomial (5,2); Monomial (4,1)
           ; Monomial (-3,0)]))),
   Derive (Derive (Poly (Polynomial [Monomial (3,1)]))))
[pretty]    D[D[5x^2 + 4x - 3]] = D[D[3x]]
[derived]   10 = 0
[norm]      10 = 0
[degree]    0
[identity]  false


>> D[D[D[5x^2 + 4x - 3]]]
[absyn]     Expr
  (Derive
     (Derive
        (Derive
           (Poly (Polynomial [Monomial (5,2); Monomial
              (4,1); Monomial (-3,0)])))))
[pretty]    D[D[D[5x^2 + 4x - 3]]]
[derived]   0
[norm]      0
```

```
[degree]    0
```

9 Suggerimenti

Al fine di non complicare inutilmente l'implementazione del progetto vi è consentito utilizzare eventuali funzioni di libreria offerte dal linguaggio, di seguito ne elenchiamo alcune che potrebbero tornarvi utili:

- List.map
- List.filter
- List.sortByDescending
- List.maxBy
- List.groupBy
- List.tryPick
- List.sumBy
- Array.sub
- Array.findIndexBack

Per maggiori informazioni vi consigliamo di leggere la documentazione ufficiale del modulo List [9] and Array [10] di F#.

References

[1] Polinomio. [Online]. Available: https://it.wikipedia.org/wiki/Polinomio

[2] Equazioni di primo grado. [Online]. Available: https://www.youmath.it/lezioni/algebra-elementare/equazioni/44-equazioni-di-i-grado-ad-una-incognita-prima-parte.html

[3] Equazioni di secondo grado. [Online]. Available: https://www.youmath.it/lezioni/algebra-elementare/equazioni/68-equazioni-di-secondo-grado-ad-unincognita-come-si-risolvono.html

[4] Equazioni di secondo grado. [Online]. Available: https://it.wikipedia.org/wiki/Equazione_di_secondo_grado

[5] Derivata. [Online]. Available: https://it.wikipedia.org/wiki/Derivata

[6] "Formal grammar," [Apri link].

[7] "Extended backus–naur form," [Apri link].

[8] "Parsing," [Apri link].

[9] Collections.list module. [Online]. Available: https://msdn.microsoft.com/en-us/visualfsharpdocs/conceptual/collections.list-module-%5bfsharp%5d

[10] Collections.array module. [Online]. Available: https://msdn.microsoft.com/en-us/visualfsharpdocs/conceptual/collections.array-module-%5Bfsharp%5D