APPELLO SCRITTO DI PROGRAMMAZIONE 1
CORSO DI LAUREA IN MATEMATICA
UNIVERSITÀ DEGLI STUDI DI MILANO
2018–2019

25.6.2019

MASSIMO SANTINI

Indice

Avvertenza. Il tema d'esame richiede lo sviluppo di un singolo programma che implementi alcune semplici operazioni tra polinomi multivariati. Le definizioni necessarie sono date nel testo. Svilupperete delle funzioni ausiliarie nel risolvere gli esercizi. Integrerete poi tutte le funzioni ausiliarie in un unico programma finale, che sarà costituito da un solo file sorgente. (Il tema d'esame richiede esplicitamente l'implementazione di alcune funzioni ausiliarie. È comunque sempre ammissibile aggiungere altre funzioni ausiliarie qualora lo riteniate opportuno per una migliore implementazione della vostra soluzione.) Si raccomanda di leggere interamente il tema d'esame con attenzione, inclusa la Premessa, prima di cominciare a scrivere le soluzioni dei singoli esercizi, in modo da avere chiaro l'obiettivo finale.

Principali argomenti del corso coinvolti dal tema.
- Programmazione strutturata,
- Implementazione di semplici algoritmi,
- Vettori e strutture,
- Ricorsione.

Premessa sui polinomi multivariati

Un *polinomio* è una espressione ottenuta a partire da un insieme di *costanti* e di simboli, chiamati *variabili*, mediante l'uso delle operazioni di addizione, sottrazione e moltiplicazione (e quindi elevamento a potenza).

Ultima revisione: 8 luglio 2019.

In questo tema d'esame, per semplicità, ci limiteremo al caso in cui le costanti siano scelte dall'insieme dei numeri interi $\mathbb{Z}$ e le variabili siano scelte tra x, y, w, z; un esempio di polinomio di tal genere è

$$3x^2 + 2xy - y^3.$$

Un polinomio è composto da un insieme finito (eventualmente vuoto) di *termini* ciascuno dei quali consiste nel prodotto di un intero, chiamato *coefficiente* del termine, con un insieme non vuoto di indeterminate, elevate a potenze intere non negative; i termini del polinomio nell'esempio precedente sono: $3x^2$, $2xy$ e $-y^3$. Due termini sono detti *simili* se vi figurano le stesse variabili elevate rispettivamente alle medesime potenze, essi possono essere combinati (grazie alla proprietà distributiva) in un singolo termine (simile ad entrambi) che ha per coefficiente la somma algebrica dei coefficienti.

Due polinomi sono detti *uguali* se e solo se possono essere trasformati l'uno nell'altro applicando le usuali proprietà commutative, associative e distributive di addizione e moltiplicazione.

Usando la proprietà associativa dell'addizione è possibile definire la *somma* di due polinomi; ad esempio

$$(3x^2 - 2x + 5xy - 2) + (-3x^2 + 3x + 4y^2 + 8) = 4y^2 + 5xy + x + 6$$

Similmente mediante l'uso ripetuto della proprietà distributiva, è possibile definire il *prodotto* di due polinomi; ad esempio

$$(2x + 3y + 5) \times (2x + 5y + xy + 1) = 4x^2 + 21xy + 2x^2 + 12x + 15y^2 + 3xy^2 + 28y + 5$$

Esercizio 1

Input output di termini.

Punti: 6.

Definite una struttura denominata `term` utile a memorizzare il termine di un polinomio, tale struttura deve avere cinque campi di tipo `int` necessari a memorizzare il coefficiente e le potenze delle quattro variabili.

Scrivete una funzione `read_term` di segnatura

```
struct term read_term(void);
```

che legga dal flusso di ingresso standard un termine e restituisca la struttura che lo memorizza; tale funzione non deve emettere alcun testo nel flusso d'uscita standard (in particolare non deve fare alcuna "richiesta" all'utente). Tale funzione può fare affidamento sul fatto che, nel flusso di ingresso, il termine rispetti strettamente questo formato:

```
C V P V P... .
```

le cui parti sono così descritte:

(1) `C` è un numero intero corrispondente al coefficiente del termine,
(2) ad esso seguono una sequenza di una o più coppie `V P` tali che:
   (a) `V` corrisponde ad una variabile (ossia un carattere scelto tra `x`, `y`,`w`, `z`),
   (b) `P` è un intero non negativo corrispondente alla potenza della variabile che lo precede (sempre presente, anche qualora valga `1`),
(3) dopo l'ultima coppia è sempre presente il carattere `.` (punto).

Questo formato rende molto semplice la lettura dell'input tramite ripetute invocazioni della funzione `scanf`, con opportune stringhe di formato. Osservate che non è garantito che le variabili siano tutte presenti, o presenti in un ordine particolare, o compaiano una sola volta. Ad esempio, a fronte dell'input

```
4 z 1 y 2 x 2 y 1 .
```

la funzione `read_term` deve restituire la struttura corrispondente al termine $4x^2y^3z$.

Scrivete una funzione `write_term` di segnatura

```
void write_term(struct term term);
```

che, avuto un termine come parametro, lo emette nel flusso d'uscita standard secondo il seguente formato

```
[C][V[P]][V[P]]...
```

dove le parti racchiuse da parentesi quadre si intendono facoltative; in dettaglio:

(1) se il coefficiente è pari a 0 non va emesso nulla, altrimenti:
(2) `C` è il numero intero corrispondente al coefficiente del termine, va omesso se pari a 1 e almeno una potenza è diversa da 0, prefissato dal segno `-` se e solo se è negativo (ma non va prefissato dal segno `+` se è positivo),
(3) le coppie che seguono corrispondono (strettamente nell'ordine `x`, `y`, `w`, `z`) alle variabili con le relative potenze, una coppia va omessa se e solo se la potenza della variabile è 0, e a sua volta la potenza va omessa qualora sia pari a 1.

Nel caso del termine $4x^2y^3z$ dell'esempio precedente, la funzione deve emettere

```
4x2y3z
```

Si osservi che, per come è definito il formato d'ingresso, la funzione `read_term` non è in generale in grado di leggere i termini emessi da `write_term`.

Esercizio 2

Input output di polinomi.

Punti: 8 **(se comprensivo della parte facoltativa).**

Notate prima di tutto che un polinomio può essere convenientemente rappresentato come un vettore di strutture del tipo definito all'esercizio precedente. Scrivete una funzione `read_poly` di segnatura

```
int read_poly(struct term *poly);
```

che, dato un vettore di termini di dimensioni opportune come argomento, legga un polinomio dal flusso di ingresso standard e lo memorizzi in esso, restituendo il numero di termini letti. Tale funzione può fare affidamento sul fatto che, nel flusso di ingresso, il polinomio rispetti strettamente questo formato:

(1) la prima linea dell'input contiene un numero intero positivo `N` pari al numero di termini del polinomio,
(2) l'input è quindi costituito da `N` linee ciascuna delle quali contiene un termine nel formato descritto al punto precedente.

Scrivete la funzione `write_poly` di segnatura

```
void write_poly(int n, struct term *poly);
```

che, dato un polinomio dal numero assegnato di termini, lo emetta nel flusso d'uscita emettendo i termini (nel formato specificato per `write_term`), prefissando tutti e soli i termini con coefficiente positivo con un `+`, a meno che non si tratti del primo termine emesso. Se il numero di termini `n` è pari a 0, la funzione emette `0`.

Parte facoltativa: scrivete una funzione `term_cmp` che, dati due termini come parametri, restituisca $-1, 0, 1$ a seconda che, rispettivamente, il primo termine sia *lessicograficamente* minore, uguale, o maggiore del secondo. L'ordine lessicografico tra termini è definito come l'ordine lessicografico per le parole, considerando ordinatamente le potenze delle quattro variabili seguite dal coefficiente. Altrimenti detto, $ax^by^cw^dz^e$ è minore di $a'x^{b'}y^{c'}w^{d'}z^{e'}$ se e solo se: $b < b'$, o $b = b'$ e $c < c'$, o $b = b'$ e $c = c'$ e $d < d'$, o $b = b'$ e $c = c'$ e $d = d'$ e $e < e'$, o $b = b'$ e $c = c'$ e $d = d'$ e $e = e'$ e $a' < a$.

Modificate quindi la funzione `write_poly` in modo che emetta i termini del polinomio in ordine lessicograficamente crescente; la nuova versione di tale funzione può essere scritta in modo molto semplice usando la **ricorsione** (e facendo uso di `term_cmp` per determinare il termine minimo in ordine lessicografico tra quelli ancora da emettere).

Esercizio 3

Semplificazione di polinomi.

Punti: 4.

Scrivete la funzione `term_like` che, dati due termini, restituisca un valore non nullo se e solo se sono simili. Usando tale funzione scrivere la funzione `simplify_poly` che, dato in ingresso un polinomio lo semplifichi nel senso che raccolga tutti i termini simili in esso contenuti, *eliminando* i termini con coefficiente pari a 0. La funzione deve operare sul polinomio stesso (ossia alterando il vettore che ha avuto come parametro), per questa ragione deve restituire il numero di termini del polinomio semplificato (tale valore può essere 0 nel caso in cui la semplificazione renda nulli i coefficienti di tutti i termini).

Esercizio 4

Somma di polinomi.

Punti: 2.

Scrivete la funzione `sum_poly` che, dati tre polinomi (ossia vettori di strutture come definiti sopra) di opportune dimensioni come argomenti, memorizzi nel terzo polinomio il risultato della somma dei primi due (dopo averlo semplificato con `simplify_poly`) e restituisca il numero dei suoi termini.

Esercizio 5

Prodotto di polinomi.

Punti: 6.

Scrivete la funzione `mult_term` che dati due termini restituisce il loro prodotto. Usando tale funzione e la funzione sviluppata al punto precedente, scrivete la funzione `mult_poly` che, dati tre polinomi (ossia vettori di strutture come definiti sopra) di opportune dimensioni come argomenti, memorizzi nel terzo polinomio il risultato del prodotto dei primi due (dopo averlo semplificato con `simplify_poly`) e restituisca il numero dei suoi termini.

Esercizio 6

Versione definitiva del programma.

Punti: 4.

Il programma deve eseguire un ciclo di lettura dal flusso di ingresso, ogni iterazione inizia con la lettura di un *comando* (specificato da una singola lettera), seguito dalla lettura dei *parametri* relativi ad esso e quindi dalla stampa del *risultato* del comando. La tabella seguente riporta i comandi, con i parametri ed i risultati attesi.

Comando	Parametri	Risultato
`T`	un termine	il termine letto
`S`	un polinomio	il polinomio
`+`	due polinomi	la somma dei parametri
`*`	due polinomi	il prodotto dei parametri
`.`	nessuno	il programma termina

I parametri sono, a seconda del dato da leggere, nel formato illustrato negli esercizi relativi alle funzioni `read_term` e `read_poly`, inoltre è garantito che nessun polinomio in ingresso ha più di 1024 termini; similmente l'output deve contenere esclusivamente il risultato prodotto da `write_term` e `write_poly`. Qualora il risultato sia un polinomio esso deve essere *sempre semplificato* (tramite la funzione `simplify_poly`) prima di essere emesso.

Ad esempio, a fronte dell'input

```
+
4
3 x 2 . -2 x 1 . 5 x 1 y 1 . -2 .
4
-3 x 2 . 3 x 1 . 4 y 2 . 8 .
*
3
2 x 1 . 3 y 1 . 5 .
4
2 x 1 . 5 y 1 . 1 x 1 y 1 . 1 .
.
```

il programma emette

```
6+4y2+x+5xy
5+28y+15y2+12x+21xy+3xy2+4x2+2x2y
```

Istruzioni per la consegna

- Funzioni da consegnare (in un unico file sorgente):

  `read_term`, `write_term`,
  `read_poly`, `write_poly`, (`term_cmp`,)
  `term_like`, `simplify_poly`
  `sum_poly`, `mult_term`, `mult_poly`

  e tutte le eventuali altre funzioni ausiliarie che abbiate scritto.
- Sito per la consegna `https://upload.mat.unimi.it`.
- Autenticarsi con le proprie credenziali di posta elettronica d'ateneo.
- Consegnare solo il codice sorgente (file `*.c`), non l'eseguibile.