# Laboratorio di programmazione

10 gennaio 2008

Scorrete il testo dell'esercizio dall'inizio alla fine prima di cominciare a risolverlo, per capire ad alto livello qual è lo scopo del lavoro che dovrete fare. Ogni punto è commentato con un po' di testo tra parentesi quadre che ne descrive il grado di difficoltà e chiarisce cosa ci si aspetta da voi.

**Precisione!** Alle elementari avete imparato a fare addizioni e moltiplicazioni di numeri con un arbitrario numero di cifre. Sapete, ad esempio, che per svolgere l'addizione di due numeri basta allinearli in modo che la cifra più a destra del primo sia sopra la cifra più a destra del secondo e poi procedere sommando le cifre ad una ad una, da destra a sinistra, scrivendo mano a mano le cifre del risultato e tenendo traccia del riporto. Facciamo un esempio, sommando 3653 e 271:

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 0 | · | riporto |
| 3 | 6 | 5 | 3 | |
| | 2 | 7 | 1 | |
| 3 | 9 | 2 | 4 | risultato |

La prima cifra del risultato è $3+1=4$ con riporto di 0, poi 2 con riporto di 1, corrispondente al fatto che $5+7=12$, quindi 9 che è $6+2$ più il riporto di 1 e via discorrendo.

La moltiplicazione è solo un po' più complicata. Si cominciano a calcolare alcuni risultati parziali: per prima cosa si fa il prodotto del primo numero per la prima cifra del secondo, quindi si fa il prodotto per la seconda cifra, e lo si scrive sotto al risultato parziale ottenuto in precedenza, ma spostandosi a sinistra di una cifra. Si procede in questo modo finché ci sono cifre per il secondo numero, quindi si sommano i risultati parziali così ottenuti. Facciamo ancora un esempio, moltiplichiamo 3653 e 271:

| | | | | | | |
|---|---|---|---|---|---|---|
| | | 3 | 6 | 5 | 3 | |
| | | | 2 | 7 | 1 | |
| | | 3 | 6 | 5 | 3 | risultati parziali |
| 2 | 5 | 5 | 7 | 1 | · | |
| 7 | 3 | 0 | 6 | · | · | |
| 9 | 8 | 9 | 9 | 6 | 3 | risultato |

I risultati parziali corrispondono, rispettivamente, a $3653 \times 1 = 3653$, $3653 \times 7 = 25571$ e $3653 \times 2 = 7306$. Osservate che questi prodotti possono essere, a loro volta, calcolati cifra per cifra, a patto di tener opportunamente conto del riporto. Ancora a titolo di esempio, calcoliamo il secondo risultato parziale:

| | | | | | |
|---|---|---|---|---|---|
| 2 | 4 | 3 | 2 | · | riporto |
| | 3 | 6 | 5 | 3 | |
| | | | | 7 | |
| 2 | 5 | 5 | 7 | 1 | risultato |

La prima cifra è 1 con riporto di 2, dato che $3 \times 7 = 21$, quindi la seconda cifra è 7 con riporto di 3, dato che $5 \times 7 + 2 = 37$ (dove 2 era il riporto) e via discorrendo.

È del tutto evidente che quelli che la vostra maestra vi ha insegnato quando eravate bambini sono un algoritmo per l'addizione e uno per la moltiplicazione!

L'obiettivo di questo esercizio è che voi implementiate tali algoritmi in C.

**Rappresentazione.** Per prima cosa, occupiamoci della rappresentazione. Fissiamo prima di tutto il numero di cifre su cui vogliamo operare, ovvero introduciamo un `#define PRECISIONE 1000` per indicare che i nostri numeri avranno al più 1000 cifre significative; in questo modo, se avessimo un ripensamento, potremo cambiare questo parametro in seguito modificando il nostro programma in un solo punto.

Assumiamo quindi di rappresentare ogni numero di $n$ cifre $c_{n-1}c_{n-2}\dots c_0$ con un vettore di $n$ interi tale che la posizione $i$-esima del vettore corrisponda all'$i$-esima cifra $c_i$. Ad esempio, il vettore

```
int x[PRECISIONE] = { 4, 2, 8 };
```

rappresenterà il numero 824. Fate attenzione al fatto che in questa rappresentazione gli eventuali 0 memorizzati nel vettore a destra dell'ultima cifra diversa da 0 non sono significativi, infatti corrispondono alle cifre del numero che compaiono a sinistra dell'ultima cifra diversa da 0. Come dire:

```
int x[PRECISIONE] = { 4, 2, 8, 0, 0, 0 };
```

corrisponde sempre al numero 824, infatti $824 = 000824$, dato che gli 0 a sinistra non sono significativi!

**Input/Output.** Preoccupiamoci ora di come convertire i dati tra la rappresentazione che abbiamo scelto e le strighe (che utilizzeremo per leggere e scrivere i nostri numeri a precisione arbitraria). [Questa parte è banale: dovete essere *tutti* in grado di svolgerla; se non ci riuscite, vuol dire che non siete al pari col programma e dovete darvi subito da fare per recuperare!]

Scrivete due funzioni, che chiamerete `san` e `nas` (acronimi, rispettivamente, di "stringa a numero" e "numero a stringa") che convertano i numeri così rappresentati dalle/alle stringhe. Più precisamente, la prima funzione ha per prototipo

```
int *san( char s[PRECISIONE+1], int a[PRECISIONE] );
```

e deve scrivere in `a` la rappresentazione del numero contenuto nella stringa. Dopo la chiamata `san( "345", y );`, ad esempio, il vettore `y` (dichiarato opportunamente prima della chiamata) deve contenere `{5,4,3,0,...,0}`. Per rendere più facile la composizione di questa funzione con le altre, vi sarà comodo che restituisca (il puntatore) `a`. D'altro lato, la seconda funzione ha per prototipo

```
char *nas( int a[PRECISIONE], char s[PRECISIONE + 1] );
```

e deve scrivere in `s` la stringa corrispondente al numero rappresentato nel vettore `a`. Se `x` è dichiarato come in precedenza, ad esempio, dopo la chiamata `nas( x, s );`, la stringa `s` (dichiarata opportunamente prima della chiamata) deve essere uguale a "`824`". Anche in questo caso, vi sarà comodo far si che la funzione restituisca (il puntatore) `s`.

Per verificare il funzionamento delle funzioni appena scritte potete utilizzare un segmento di codice come il seguente (sta a voi scrivere tutto quel che manca!):

```
char s1[ PRECISIONE + 1 ], s2[ PRECISIONE + 1 ];
int x[ PRECISIONE ];

scanf( "%s", s1 );
printf( "%s %s\n", s1, nas( san( s1, x ), s2 ) );
```

Se tutto funziona, la `printf` stamperà due volte la stessa stringa: la prima è il vostro input, la seconda è il risultato della conversione da stringa a numero e quindi da numero a stringa.

**Addizione.** Ora occupiamoci delle operazioni. Iniziamo dall'addizione, che è la più facile. [Questa parte è ragionevolmente semplice: con un po' di sforzo dovreste essere *tutti* in grado di svolgerla.]

Implementate la funzione con prototipo

```
int *piu( int a[PRECISIONE], int b[PRECISIONE], int c[PRECISIONE] );
```

in modo che la sua esecuzione abbia l'effetto di scrivere in `c` la rappresenazione dell'addizione dei numeri le cui rappresentazioni sono contenute in `a` e `b`. State attenti: potrebbe capitare che si generi riporto all'addizione dell'ultima cifra più a sinistra, visto che non avete abbastanza cifre per ricordarvelo, buttatelo pure via (il risultato sarà errato, ma che ci volete fare, se non avete abbastanza cifre, non c'è verso). La funzione deve restituire (il puntatore) `c`, che può far comodo per comporla con altre funzioni.

Di nuovo, per verificare il funzionamento della funzione appena scritta potete utilizzare un segmento di codice come il seguente (sta sempre a voi scrivere tutto quel che manca!):

```
char s[ PRECISIONE + 1 ];
int a[ PRECISIONE ], b[ PRECISIONE ], c[ PRECISIONE ];

scanf( "%s", s ); san( s, a );
scanf( "%s", s ); san( s, b );

printf( "%s", nas( piu( a, b, c ), s ) );
```

**Moltiplicazione.** I più coraggiosi possono ora provare a passare alla moltiplicazione. [Questa parte richiede buona dimestichezza con i concetti visti sin ora a lezione: mi aspetto che i più bravi siano grado di svolgerla (anche se con un certo sforzo).]

Implementate la funzione con prototipo

```
int *per( int a[PRECISIONE], int b[PRECISIONE], int c[PRECISIONE] );
```

in modo che la sua esecuzione abbia l'effetto di scrivere in `c` la rappresentazione della moltiplicazione dei numeri le cui rappresentazioni sono contenute in `a` e `b`. Occhio anche in questo caso: il risultato potrebbe richiedere più cifre di quelle a disposizione, nel caso, buttate tutto quel che non ci sta (di nuovo, il risultato sarà errato, ma senza aver abbastanza cifre non potete far altro). Anche questa funzione deve restituire (il puntatore) `c`, che può far comodo per comporla con altre funzioni.

Per verificare il funzionamento dell'ultima funzione potete usare il precedente segmento di codice, avendo cura di sostituire la chiamata a `piu` con una chiamata a `per`.

**Numeri di Fibonacci!** E ora, per i temerari... [Questa parte è difficile: se ve la cavate da soli, complimenti!]

Sapreste modificare la versione iterativa di `fibonacci` (quella basata sull'uso di tre sole variabili) per calcolare $F_{1000}$? Io dico che vale `434665576869374564356885276750406258025646605173717804024817290895365554179490518904038798400792551692959225930803226347752096896232398733224711616429964409065331879382989696499285160037044761377951668492288 75...` voi che dite?