Scheda di lavoro N. 5

Questa è la *scheda delle funzioni.* Si tratta di scrivere delle funzioni semplici, con il corredo dei programmi che le richiamano.

Prima di torturare la tastiera rileggi le *Avvertenze importanti* della scheda 4.

1. Trasforma il programma che calcola il fattoriale in una funzione `fatt(k)`. Poi scrivi un programma di prova che richiami la funzione e ne controlli il buon funzionamento.

Cominciamo con lo stabilire come deve essere costruita la funzione (rileggi il §3.3.1 delle *Note su linguaggio C e dintorni*).

- Il nome della funzione: `fatt`.
- Dati in ingresso: un intero non negativo `k`.
- Operazioni da eseguire: il calcolo del fattoriale di `k`.
- Valore da restituire: `k!` come intero.
- Se il numero in ingresso non è positivo la funzione dovrà restituire `-1`.

Se ci pensi un momento, ti renderai conto che il nucleo del programma che hai già scritto – la parte che esegue il calcolo della funzione – non richiede modifiche. Devi solo cambiare il contorno: la lettura di `k` e la scrittura del risultato dovranno restare nel programma che chiama la funzione.

Eccoti lo scheletro della funzione:

```
int fatt(k)
     int k;
{
  int n;
    ...  qui il calcolo di n = k!
  return(n)
}
```

A te il compito di inserire le istruzioni per il calcolo, prendendole dal programma per il fattoriale che hai già scritto.

Il programma di prova dovrà:

- Leggere un numero `k`;
- calcolare `n = fatt(k);`
- scrivere il risultato.

In altre parole, nel programma della scheda 4 devi solo sostituire il blocco di calcolo del fattoriale con la chiamata alla funzione. Il resto funziona ancora. Se proprio non ci riesci, prova a guardare il programma `prova_fatt` tra i files del dischetto.

2. Trasforma in funzione anche il programma che calcola il coefficiente binomiale. Non dovresti aver bisogno di molte indicazioni: ti basterà ripetere quello che hai fatto per la funzione fattoriale.

3. L'*algoritmo di Euclide* per il calcolo del massimo comun divisore.

Alla scuola media hai appreso come calcolare il massimo comun divisore con un metodo basato sulla scomposizione in fattori primi. Non tentare di tradurlo in un programma:

l'algoritmo di Euclide funziona molto meglio. Si fonda sull'osservazione seguente: sia $a > b > 0$; allora ogni sottomultiplo di a e b è anche sottomultiplo di $a(\bmod b)$ (il resto della divisione di a per b).

Ecco l'esposizione più dettagliata:

(i) disponi i numeri in modo che sia $a \geq b$ (eventualmente scambiandoli);
(ii) se $a \pmod b$ è zero, b è il MCD tra a e b;
(iii) altrimenti sostituisci la coppia (a, b) con la coppia $\big(b, a \pmod b\big)$ e torna al passo (ii).

Nota: in C l'operazione $a \pmod b$ si scrive `a% b` (il resto della divisione di `a` per `b`.

Ed ecco come deve essere costruita la funzione:

- Il nome: `mcd`, ovviamente.
- Dati in ingresso: una coppia di numeri interi. La funzione dovrebbe accettare anche numeri negativi (il massimo comun divisore è definito comunque
- Operazioni da eseguire:
  - se uno o ambedue gli argomenti sono negativi sostituiscili con il valore assoluto;
  - se uno solo dei due numeri è zero, il MCD è l'altro numero (zero è multiplo di qualunque numero);
  - se ambedue i numeri sono zero la funzione deve restituire zero (nota: questo lo fai già con l'operazione precedente);
  - procedi al calcolo del massimo comun divisore con làlgoritmo che ti ho già illustrato;
  - restituisci il risultato.
- Valore da restituire: il massimo comun divisore, o zero in caso di argomenti ambedue nulli.

Scrivi anche un programma di prova per verificare il buon comportamento della funzione.

Dopo mezz'ora di tentativi prova a guardare il file `mcd.c` sul dischetto. In questo file viene riportato l'algoritmo enunciato sopra. Nel file `mcd_fast.c` trovi una variante più veloce (evita il trasferimento di dati in memoria). Il programma di prova si chiama `p_mcd.c`. Il programma funziona con ambedue le versioni: puoi compilare con uno dei due comandi

`gcc -o p_mcd p_mcd.c mcd.c`

`gcc -o p_mcd p_mcd.c mcd_fast.c` per scegliere l'una o l'altra versione.

4. Supponi di conoscere i coefficienti $a_0, \ldots, a_n$ di un polinomio di grado n, e di voler calcolare il valore di questo polinomio in un punto x assegnato. Scrivi una funzione che esegua questa operazione e restituisca il valore calcolato. Scrivi anche un programma di prova che ne verifichi il buon funzionamento.

Ecco le caratteristiche della funzione:

- Nome: `polysum`. Questo è un più originale del solito.
- Argomenti: un vettore `double` di $n+1$ elementi contenente i coefficienti $a_0, \ldots, a_n$ del polinomio; un intero non negativo n che specifica il grado; un `double` che specifica il punto x in cui deve essere calcolato il valore.
- Valore restituito: $a_0 + a_1 x + \ldots + a_n x^n$, come valore `double`.

Veniamo all'algoritmo. La prima cosa che ti verrà in mente sarà probabilmente: applico la formula. Dopo aver cercato disperatamente sulle *Note su linguaggio C e dintorni* o su un manuale ti renderai conto che – ahimé, il linguaggio C non prevede una notazione per il calcolo della potenza di un numero. Se però avrai abbastanza pazienza nel fare la ricerca scoprirai che esiste una funzione di libreria `pow(x,y)` che restituisce x^y. Felice della tua scoperta, tradurrai la formula in un ciclo più o meno di questo tipo:

```
for(p=j=0; j<=n; j++) p = p + a[j]*pow(x,j);
```

Qui, naturalmente, `n` è il grado del polinomio, `a` il vettore `double` che contiene `n+1` coefficienti indicizzati da 0 a `n`, e `x` è il punto in cui va calcolato il valore del polinomio. Nello scrivere la funzione non dimenticare la linea

```
#include <math.h>
```

o mal te ne incoglierà (prova, se non ci credi!) e non dimenticare neppure di aggiungere `-lm` al comando di compilazione. Non credo che ti ci vogliano molte altre informazioni per scrivere sia la funzione che il programma di prova.

Il difetto dell'algoritmo qui sopra è la sua scarsa efficienza. È più conveniente usare la formula

$$a_0 + x\Bigl(a_1 + x\bigl(a_2 + x(\ldots)\bigr)\Bigr) \ .$$

Non ti sarà difficile verificare che si tratta propio del polinomio che ho scritto qui sopra. Anche se apparentemente più complessa, questa formula si programma in modo sorprendentemente semplice osservando che si può porre in forma ricorsiva:

poni $p = a_n$;
ridefinisci p come $a_{n-1} + xp$;
ridefinisci p come $a_{n-2} + xp$;
... ...
ridefinisci p come $a_0 + xp$.

Eccoti l'algoritmo in forma più precisa:

(i) poni `p=a[n]`;
(ii) esegui un ciclo su `j=n-1,...,0`
  (ii.a) ad ogni passo del ciclo ridefinisci `p = a[j]+x*p`
(iii) alla conclusione del ciclo `p` è il valore cercato.

La codifica in C è altrettanto semplice:

```
for(p=a[n],j=n-1; j>=0; j--) p=a[j]+x*p;
```

Dovresti essere in grado di scrivere ambedue le versioni della funzione, ed anche un breve programma di prova Se dopo 20 minuti non ne sei arrivato a capo, prova a guardare il programma `polysum.c` sul dischetto.

5. La funzione esponenziale si può calcolare, in linea di principio, mediante il suo sviluppo in serie

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots$$

Naturalmente, calcolare la somma infinita non è possibile; ma ha senso troncare lo sviluppo ad un numero finito di termini, a condizione che x non sia troppo grande. Il problema è: per un dato x, quanti termini devo calcolare? La risposta la trovi su un buon libro di analisi, ma non ti farà male provare. Per questo ti serve un programma che calcoli l'espressione qua sopra, troncata ad un certo n che potrai tenere variabile in modo da vedere che succede se si aumenta il grado di approssimazione. Per valutare la bontà del risultato potrai sempre confrontare il valore calcolato con quello che ti viene restituito dalla funzione di libreria `exp(x)`.

La scrittura del programma non è difficile: ti basterà definire il vettore dei coefficienti, ed usare la funzione di calcolo di un polinomio che hai appena scritto.

Per il calcolo dei coefficienti potresti essere tentato di usare la funzione per il calcolo del fattoriale, ma tutto sommato non è un'idea grandiosa: la formula ricorsiva

$$a_0 = 1 \ , \quad a_n = \frac{a_{n-1}}{n} \quad \text{per } n > 0$$

è più comoda. Eccoti la codifica in C:

```
for(a[0]=j=1; j<=n; j++) a[j] = a[j-1]/j;
```

Qui, `a` è il vettore dei coefficienti, di tipo `double`, e `n` è il grado dell'approssimazione polinomiale.

Ideare un algoritmo per il resto del programma è compito tuo.