Scheda di lavoro N. 7

Questa è la *scheda dell'aritmetica*. Si tratta di esercizi sulla conversione della rappresentazione di numeri tra basi diverse. Anche qui, ogni funzione dovrà essere corredata da un semplice programma di prova.

Per rappresentare i numeri in una base `b` qualsiasi ocorrono b simboli, una per ciascuna cifra. Se $b < 10$ basta usare le prime b cifre della nostra notazione consueta. Se $b > 10$ invece bisogna inventare dei nuovi simboli. Una scelta comoda consiste nel far uso delle lettere alfabetiche (meglio far uso di tutte le 26 lettere dell'alfabeto inglese), intendendo che `a` sta per 10, `b` per 11, &c, fino a `z` che sta per 35.

Gli esercizi che seguono ti illustrano come costruire un pacchetto di sottoprogrammi che eseguono tutte le conversioni necessarie. Allo stesso tempo lascia che ti inviti a riflettere su come l'utilizzo delle funzioni ti permetta di programmare senza eccessiva difficoltà anche operazioni a prima vista più complesse di quelle che hai svolto fin qui.

L'invito, come sempre, è a riflettere bene prima di metterti davanti alla tastiera, e cercare di scrivere personalmente il codice. Poi potrai confrontare la tua soluzione con quella che trovi nel file `conversioni_base.c`, sul dischetto di laboratorio.

Ultima nota: questi esercizi possono richiedere l'inserimento di diversi files nella fase di compilazione. In altre parole, linee di comando lunghe come treni. Un modo per semplificarti la vita è il ricorso al programma di utilità `make`. Troverai le inforamzioni essenziali alla fine di questa scheda.

1. Scrivi due funzioni

```
char bin_a_cifra(int k)
int cifra_a_bin(char c)
```

che convertano una singola cifra in una base qualsiasi tra 2 e 36. Precisamente:

- `char bin_a_cifra(int k)` deve ricevere un intero tra 0 e 35, e convertirlo in cifra usando i caratteri numerici `0,...9` e quelli alfabetici minuscoli `a,...,z` dell'alfabeto inglese. Se `k` non ripetta i limiti assegnati la funzione restituirà il carattere `NUL`.
- `int cifra_a_bin(char c)` deve ricevere un carattere numerico o alfabetico e restituire il valore binario della cifra corrispondente. I caratteri maiuscoli devono essere accettati, e trattati come se fossero minuscoli. Se il carattere ricevuto non è alfabetico o numerico la funzione dovrà restituire `-1`.

La scrittura di queste funzioni non richiede un algoritmo particolarmente complesso: basta osservare con un pò di attenzione la tabella del codice ASCII. Se non hai a portata di mano le *Note su linguaggio C e dintorni* non entrare in fibrillazione: la pagina di manuale consultabile col comando `man ascii` riporta la tabella. Noterai che le cifre da 0 a 9 hanno i codici ottali compresi tra `060` e `071` (in decimale tra 48 e 57). Quindi, per convertire un numero binario `k` compreso tra 0 e 9 nella cifra corrispondente basta scrivere `k+060`, oppure `k+48`, o `k+'\060'`, o anche `k+'0'`. Tutte queste scritture sono equivalenti. Analogamente per convertire in binario un carattere `c` contenente la rappresentazione ASCII di una cifra tra 0 e 9 basterà usare una delle espressioni `c-060`, `c-48`, `c-'\060'` o `c-'0'`. Per numeri binari compresi tra 10 e 35 si segue la stessa regola, con la sola avvertenza di sostituire al codice ASCII della cifra zero quello del carattere a, minuscolo o maiuscolo.

2. Scrivi due funzioni

```
void bin_a_base(int k,int b,char str[])
int base_a_bin(int b,char str[])
```

La prima deve convertire un dato `int` nella rappresentazione in base `b`. La seconda deve convertire un dato dalla rappresentazione in base `b` a binario. Le funzioni dovranno tener conto anche del segno.

Queste funzioni non sono semplici. Non illuderti di scriverle di getto davanti al terminale: riflettere con in mano la penna e un blocco per gli appunti è praticamente indispensabile. Ma prima leggi bene le pagine sulla rappresentazione degli interi con una base nel capitolo 4 delle *Note su linguaggio C e dintorni*.

Iniziamo con la prima funzione. Eccoti lo schema generale delle operazioni da eseguire:

(i) Controlla che la base `b` sia consentita: `b` deve essere compreso tra 2 e 36. Altrimenti scrivi una segnalazione di errore e fa terminare il programma.

(ii) Se `k` vale 0 è inutile perdere tempo: restituisci la stringa `"0"`, e basta. Attenzione però: non dimenticare il `NUL` che chiude la stringa! Userai la coppia di istruzioni

```
str[0]='0'; str[1]='\000';
```

(iii) Definisci un nuovo intero `m` come il valore assoluto di `k`; questo lo puoi fare con l'istruzione

```
if(k>0) m=k; else m= -k;
```

(iv) Ora riempi la stringa, partendo da `str[0]`, con i resti delle divisioni successive di `m` per `b`, tradotti in cifra. Qui potrai sfruttare la funzione `bin_a_cifra` che hai scritto nello svolgere l'esercizio 1. Dovrai predisporre un ciclo `for` con un contatore `j`:

(iv.a) inizializzazione: poni `j=0`;

(iv.b) il ciclo continua fin che `m!=0`;

(iv.c) ad ogni iterazione memorizza una cifra, e dividi `m` per `b`; potrai usare le istruzioni

```
tr[j]=bin_a_cifra(m%b);
m/=b;
```

(iv.d) incrementa `j` e torna ad inizio ciclo.

Nota che le cifre vengono memorizzate in ordine inverso rispetto a quello di scrittura: `str[0]` contiene la cifra meno significativa.

(v) Se `k` è negativo aggiungi il segno `'-'` in fondo alla stringa che hai generato. Nota che all'uscita dal ciclo il contatore `j` punta già al carattere che si trova dopo l'ultima cifra. Ti basterà scrivere

```
str[j++]='-';
```

Attenzione: è essenziale incrementare `j`.

(vi) Chiudi la stringa col `NUL` finale:

```
str[j]='\000';
```

(vii) Infine, inverti la stringa facendo uso della funzione `inverti_stringa` che hai scritto mentre ti esercitavi con la scheda 6 (li hai fatti quegli esercizi, vero?).

(viii) Hai fatto tutto: non ti resta che inserire un buon `return;`

Prima di proseguire scrivi un programma di prova che verifichi il buon comportamento della funzione che hai scritto.

Ed ora veniamo alla funzione `base_a_bin(int b,char str[])`. Eccoti lo schema.

(i) Anche qui, controlla che la base `b` sia consentita, e fa terminare il programma con una segnalazione di errore se non lo è.

(ii) Controlla se il primo carattere di `str` è un segno `'-'` o `'+'`; in questo caso la conversione dovrà iniziare dal secondo carattere, ossia da `str[1]`.

(iii) Esegui un ciclo `for` di conversione dei caratteri in binario. Ad ogni passo potrai far uso della funzione `cifra_a_bin` che converte una singola cifra, e che dovresti aver già scritto. Eccoti i passi del ciclo:

- (iii.a) inizia azzerando un intero `m`, e con un contatore `j` che partirà da `1` se `str[0]` è un segno, o da `0` se `str[0]` è già una cifra. L'inizializzazione del contatore `j` la potrai fare esternament al ciclo, nel corso del passo (ii).
- (iii.b) Il ciclo continua fin che `str[j]` è il carattere `NUL`.
- (iii.c) Ad ogni passo del ciclo dovrai moltiplicare `m` per `b` ed aggiungergli la cifra `str[j]` convertita in binario. Ad esempio, potrai usare le istruzioni

  `k=cifra_a_bin(str[j]);  m=m*b+k;`

  Attenzione però: non è detto che la stringa contenga cifre accettabili. Sarà bene controllare che `k` sia compreso tra `0` e `b-1`, ed interrompere il programma in caso di errore.
- (iii.d) Incrementa `j` e continua il ciclo.

(iv) Se la stringa `str` inizia con un segno `'-'` cambia il segno di `m`

(v) È fatta: puoi restituire il risultato con un `return(m);`

3. Scrivi un programma che stampi la tavola pitagorica per una base `b` arbitraria.

Se dovressi risolvere questo esercizio mettendo tutto in un solo programma dovresti sudare un po' prima di mettere tutto al posto giusto. Ma se ci provi usando le funzioni che hai appena scritto ti renderai conto che diventa una faccenda semplice. Dovrai stampare una tabella di `b-1` righe e `b-1` colonne, quindi ti serviranno due cicli annidati uno dentro l'altro: il ciclo esterno corre sulle righe, quello interno sulle colonne. Per ciascun elemento esegui la conversione con la funzione `bin_a_base` e stampa il risultato senza inserire il carattere `\n` che manda a capo. Alla fine della riga stampa un singolo `\n`.

A te completare i dettagli, senza dimenticare che anche l'occhio vuole la sua parte: le colonne dovranno essere ben allineate. Se proprio ti trovassi in difficoltà prova a guardare il file `tavola_pitagorica.c` sul dischetto.

4. Scrivi un programma che legga un numero in una base `b_1` e lo riscriva in una base `b_2`. Per controllo potrai anche stampare il numero in decimale.

Questo dovresti essere in grado di farlo senza aiuto. In ogni caso, troverai la soluzione nel file `prova_conversione.c`, sul dischetto.

Come promesso, ti aggiungo qualche nota sull'uso di `make`. Supponi di aver preparato tre files che contengono delle funzioni di uso generale, e che i files si chiamino `qui.c`, `quo.c` e `qua.c`. Supponi inoltre di aver scritto due programmi distinti `minnie.c` e `paperoga.c` che richiamano queste funzioni. Potrai preparare un file dal nome `Makefile` che contiene le linee seguenti:

```
minnie:  minnie.c qui.c quo.c qua.c
        gcc -o minnie minnie.c qui.c quo.c qua.c -lm
paperoga:  paperoga.c minnie.c qui.c quo.c qua.c
        gcc -o paperoga paperoga.c minnie.c qui.c quo.c qua.c -lm
```

Eccoti il significato. La prima linea significa: vorrei costruire un file che si chiama `minnie`; per costruirlo ho bisogno dei files`minnie.c`, `qui.c`, `quo.c` e `qua.c`; le istruzioni su come costruirlo le trovi sulle linee successive, precedute da un carattere `TAB`. In questo caso le istruzioni si compongono di una sola riga:

```
gcc -o minnie ...
```

La linea che inizia con `paperoga` ha lo stesso significato, mutatis mutandis.
Attenzione: `make` è molto schizzinoso. La linea di comando `gcc ...` deve iniziare esattamente con un carattere `TAB`. Se ci metti degli spazi si arrabbia molto.

Come si usa? Semplice: batti il comando

```
make paperoga
```

e il sistema eseguirà le operazioni che seguono:

- Confronta l'ora di creazione del file `paperoga` con quella dei files `paperoga.c`, `qui.c`, `quo.c` e `qua.c`;
- se il file `paperoga` è più recente di tutti gli altri assume che sia già aggiornato, e non fa nulla;
- se uno almeno dei files `paperoga.c`, `qui.c`, `quo.c` e `qua.c` è più recente di `paperoga`, o più semplicmente il file `paperoga` non esiste, allora ne crrea uno nuovo eseguendo le istruzioni che trova sulle righe successive; in questo caso esegue la compilazione.

Lo stesso avviene se batti il comando

```
make minnie
```

Questo è già sufficiente per risolvere il problema di come compilare programmi che richiedono diverse funzioni.

Se vuoi, potrai anche automatizzare altre operazioni, come ad esempio la pulizia del directory con la cancellazione di tutti i files eseguibili che non ti servono più, e che puoi comunque ricostruire in qualunque momento. Basterà aggiungere in `Makefile` le linee

```
clean:
        rm minnie
        rm paperoga
```

(anche qui, attenzione ai `TAB`). Il comando

```
make clean
```

provvederà alla pulizia.

Se poi volessi compilare in blocco tutti i programmi ti basterà aggiungere la linea

```
all:  minnie paperoga
```

e battere il comando

```
make all
```

Troverai un esempio di `Makefile` sul dischetto.