Scheda di lavoro N. 8

Questa è la *scheda del playbit*. Si tratta di esercizi sulla manipolazione di bit, singoli o a gruppetti, entro un dato di tipo intero. Per eseguire questi esercizi dovrai aver assimilato il contenuto del capitolo 4 delle *Note su linguaggio C e dintorni*, ed in particolare il paragrafo 4.4.1. Tutti gli esercizi che ti propongo sembrano piuttosto difficili, ma hanno una soluzione semplice che fa uso delle operazioni bit–a–bit, ossia *and* (`&`), *or* (`|`), *xor* (`^`), *not* (`~`), *shift* (`<<` o `>>`).

Anche qui, naturalmente, ogni funzione dovrà essere corredata da un programma di prova.

1. Supponi di lavorare su una macchina che non hai mai usato in precedenza, avendo a disposizione un compilatore C, e di voler rispondere alle domande seguenti:

(1) di quanti bit è composto un dato di tipo `char`, `short int` o `long int`?
(2) la rappresentazione intera usa l'aritmetica modulo 2^n?
(3) assumendo che la risposta alla domanda (ii) sia positiva, qual è il massimo intero rappresentabile in un dato di tipo `char`, `short int` o `long int`?

Prova a scrivere delle funzioni che ti permettano di rispondere a queste tre domande.

Nota che le domande non sono del tutto peregrine: la struttura della memoria a bytes di 8 bit, con tutto quel che segue, è molto comune, ma non è detto che proprio tutte le macchine siano organizzate a questo modo. Le funzioni che scrivi dovrebbero funzionare su qualunque macchina. Quindi devi trovare un algoritmo che ti permetta di trovare la risposta senza assumere nulla sulla lunghezza dei dati in bit. Eccoti alcuni suggerimenti.

Per rispondere alla domanda (1) dovrai scrivere 3 funzioni, che potrai chiamare ad esempio `l_char`, `l_short_int` e `l_long_int`. Le tre funzioni saranno quasi identiche: dopotutto si tratta solo di cambiare una dichiarazione.

Per sapere quanti bit ci sono in una cella basta alzarli tutti, e poi azzerarli uno per volta, tenendo un conteggio. Un po' come estrarre ad una ad una delle palline da un cestello, contandole. Il problema è: come faccio ad azzerarli uno ad uno? Semplice: basta estrometterli uno per volta, con uno shift. Il conteggio finisce quando l'ultimo bit esce dalla cella e svanisce nel nulla. Ecco l'algoritmo per il caso del tipo `char`:

(i) crea una cella `char mask` che contiene `11..11` (tutti i bit col valore 1). Ricorda: la negazione cambia i bit 0 in 1;
(ii) costruisci un ciclo:
  (ii.a) inizializza un contatore `j` a zero;
  (ii.b) ad ogni passo esegui uno shift della cella `mask` verso sinistra, ed incrementa il contatore;
  (ii.c) termina il ciclo quando `mask` si è azzerato.
(iii) il contatore dice quanti sono i bit.

Il tutto si riduce in pratica a due istruzioni:

```
char mask=~0;
for(j=0; mask!=0;j++) mask<<=1;
```

Il valore di `j` all'uscita dal ciclo dà il numero di bit. La funzione `l_char` dovrà restituire il numero di bit come intero, e non richiederà nessun argomento. In altre parole, la dichiarazione dovrà essere

`int l_char()`

Per i dati `short int` o `long int`, così come per qualunque altro dato di tipo intero, potrai usare lo stesso schema. A te completare l'esercizio scrivendo tutte le funzioni richieste. Se proprio ti trovassi in difficoltà, prova a gurdare il file `lunghezze_dati.c` sul dischetto.

Veniamo alla domanda (2). Sulla rappresentazione dei numeri positivi tutti i progettisti sono d'accordo: si usa la base 2. Le differenze si vedono nella rappresentazione dei numeri negativi. Le scelte tipiche sono tre:

(a) si alza il bit di segno, tenendo il resto invariato;
(b) si rappresenta un numero negativo mediante il complemento a 1 del corrispondente positivo;
(c) rappresenta un numero negativo mediante il complemento a 2 del corrispondente positivo;

la scelta (c) è quella fondata sull'aritmetica modulo 2^n.

Un test banale consiste proprio nel controllare come viene rappresentato un numero negativo, ad esempio `-1`. Avremo i casi seguenti: (a) `100...001` (i puntini sono sostituiti da zeri); (b) `111...110` (i puntini sono sostituiti da uno); (c) `111...111` (i puntini sono sostituiti da uno).

A noi importa il caso (c). Queste informazioni ti dovrebbero bastare al fine di scrivere la funzione. A te la scelta del nome.

L'esercizio (c), una volta nota la lunghezza del dato, si riduce semplicemente a mettere in una cella la configurazione che corrisponde al massimo intero positivo e stamparla. Se hai ben studiato il capitolo 4 delle *Note su linguaggio C e dintorni* non dovresti avere difficoltà.

2. Scrivi una funzione che stampi sul terminale la sequenza di bit contenuti in una cella di tipo intero `char`, `short int` o `long int`.

Qui hai due possibilità: o scrivere tre funzioni distinte, una per ciascun tipo, oppure prevedere come argomento della funzione il numero di bit da esaminare. La seconda soluzione è più comoda e generale, quindi ti illustro quella.

Eccoti il prototipo della funzione:

```
void scrivi_bit(long int k,int n);
```

dove `k` è il dato, e `n` il numero di bit che vuoi scrivere. La funzione non restituisce nessun valore. Per scrivere l'intero dato potrai determinare `n` con le funzioni che hai preparato svolgendo l'esercizio 1.

Veniamo all'algoritmo. Si tratta in sostanza di far scorrere i bit da quello più significativo a quello meno significativo, e scrivere `0` o `1`, a seconda del valore del bit. L'operazione è più semplice di quanto non sembri a prima vista: pensa all'operazione `and` (`&`).

Sia `n` il numero dei bit da esaminare. Occorrerà iniziare con un bit alzato nella posizione `n-1`, e poi spostarlo verso sinistra fino al bit in posizione `0`. Ci vuole un ciclo con una maschera di bit (un modo complicato per dire: un dato intero della lunghezza massima che intendi considerare) che si comporti così:

(a) all'inizio contiene `1` nella posizione `n-1`, e tutti gli altri bit sono `0`;

(b) ad ogni passo del ciclo il bit che vale `1` si sposta verso destra;
(c) il ciclo termina quando la maschera si è azzerata (il bit che indica la posizione da esaminare è caduto nel vuoto).

Realizzare un tal ciclo è più semplice di quanto tu non creda. Diamo il nome `mask` alla maschera di bit (dichiarata `long int`). Eccoti l'istruzione:

```
for(mask= 1<<(n-1); mask!=0; mask >>= 1) ...
```

L'istruzione `mask= 1<<(n-1)` significa: metti `1` in mask, e poi esegui uno shift verso sinistra di `n-1` posti. L'istruzione `mask >>= 1` significa: sposta tutti i bit di `mask` a destra di una posizione.

Non resta che individuare lo stato del bit di una cella, chiamiamola `j`. Ricorda cosa fa l'operazione *and*: il risultato di `mask & j` è diverso da zero se e solo se `mask` e `j` hanno dei bit alzati nella stessa posizione (ricorda anche che mask ha un solo bit alzato). Quindi stamperai `0` o `1`, come appropriato. Eccoti le istruzioni

```
if((mask & j) == 0) printf("0"); else printf("1");
```

Completare l'esercizio ora dovrebbe essere facile. In ogni caso, c'è sempre la soluzione nel file `playbit.c`.

3. Numera con 0,1,2,3,... i bit a partire dal meno significativo.

- Scrivi una funzione `int cerca_bit_1(long int k)` che determini la posizione del primo bit che ha il valore 1. La funzione dovrà restituire la posizione del bit. Se tutti i bit sono zero dovrà restituire la lunghezza di un dato `long int` (32 sui processori della serie Intel o AMD).
- Scrivi una seconda funzione `int cerca_bit_0(long int k)` che restituisca la posizione del primo bit che ha il valore 0.

Anche qesto esercizio non è difficile, ed in fondo negli esercizi che hai fatto fin qui hai già visto tutti gli strumenti che ti servono. Proviamo ad esaminare il primo. Ti basterà realizzare un ciclo con una maschera di bit che inizia con `1` (il bit in posizione 0 vale `1`, e tutti gli altri sono abbassati), e spostare il bit verso sinistra fin che non trovi il bit che vale `1` in k. Il tutto si fa con un'istruzione semplice:

```
for(j=0; mask!=0 && (mask&k)==0; mask <<= 1) j++;
```

alla fine del ciclo `j` da la posizione del bit `1`.

Il secondo esercizio è praticamente una copia del primo, con una modifica banale.

4. Scrivi una funzione che accetti come argomento un dato di tipo intero, e restituisca un intero della stessa lunghezza dell'argomento e con gli stessi bit, ma disposti in ordine inverso. Ad esempio, se un dato di tipo `char` contiene `10010110` la funzione dovrà restituire `01101001`.

Questo è un po' più difficile degli altri, ma non impossibile. Svolgiamolo per un dato `long int`, trasferendo i bit da `k` a `j`. Un metodo semplice (anche se non il più efficiente) è il seguente: esamina uno per volta i bit di `j` partendo da destra (dal meno significativo), e inseriscili sempre da destra in `k`, spostandoli successivamente verso sinistra.

Chiameremo la funzione `long int inverti_bit(long int k)`. Ecco l'algoritmo in termini più precisi.

(i) costruisci un ciclo:
   (i.a) inizializza una maschera di bit `mask` con 1, e azzera `j`;
   (i.b) ad ogni passo sposta i bit di `j` a sinistra di un posto, e se `mask & k` non è zero alza il primo bit di `j`;
   (i.c) sposta a sinistra di un posto il bit della maschera; il ciclo termina quando la maschera si è annullata.

(ii) Restituisci il valore di `j`.

A te tradurre tutto in istruzioni. Troverai la soluzione nel solito `playbit.c`. Per controllare la correttezza della funzione potrai ben sfruttare `scrivi_bit` – che hai già scritto e controllato, vero? – per scrivere il risultato sul terminale.

Il programma di prova per tutte le funzioni di cui ti ho fornito il testo lo trovi nel file `prova_bit.c`.