Esercizio 4 - Riordinare un array multidimensionale

Quesito

E' fornito uno array multidimensionale di numeri non negativi in ingresso. L'array è multidimensionale e vengono forniti sia il numero di dimensioni (o assi) sia quanti elementi vi sono su ciascuna dimensione. L'array che viene fornito è linearizzato in un unico stream di byte che termina con `\n`.

Ad esempio, se consideriamo l'array bidimensionale:

```
# array
  1   2   3
  4   5   6
  7   8   9
 10  11  12
```

abbiamo che gli assi sono `2` e che le dimensioni sono `4x3`, infatti abbiamo 4 righe e 3 colonne.

Queste due informazioni sono passate nel file come:

```
#assi
2
#dimensioni
4
3
```

Questo array viene linearizzato come:

`1 2 3 4 5 6 7 8 9 10 11 12`

Le cifre inserite nell'array **non** sono codificate in un file binario **bensì** in un file di testo con **codifica ascii**. Inoltre le cifre sono decimali e riportate su **5** posizioni. Si noti che nel caso precedente le cifre sono riportate senza il padding a `5` digits per semplificare ma altrimenti nell'esempio concreto abbiamo:

```
# array
00001  00002  00003
00004  00005  00006
00007  00008  00009
00010  00011  00012
```

Ossia uno stream:

`000010000200003000040000500006000070000800009000100001100012`

Inoltre nel file di input viene anche fornito gli indici che mostrano come dobbiamo riordinare gli assi dell'array.

Sempre riguardo all' esempio precedente, le dimensioni dell'array sono `4x3`. Se gli indici forniti per il riordinamento sono:

```
#order
1
0
```

allora significa che l'array deve invertire gli assi in modo tale che la prima dimensione di partenza corrisponda con ultima. Il primo `1` ci dice che `4` elementi di riga devono diventare le colonne; mentre il secondo valore `0` ci dice che colonne nell'array sorgente diventano le righe in quello riordinato. In altre parole gli assi del nuovo array sono sempre `2` ma della forma `3x4`.

Ossia in questo caso lo stream diventa:

`000010000400007000100000200005000080001100003000060000900012`

Il programma deve riordinare l'array multidimensionale in base all'ordine dei nuovi assi forniti e deve farlo stampando ogni cifra come una stringa esadecimale. Ossia per l'input sottostante, già riordinato:

`000010000400007000100000200005000080001100003000060000900012`

deve generare:

`147a258b369c\n`

stampandolo a video tramite apposite systemcall. Ho inserito forzatamente \n intendendo che alla fine della stringa va stampato il carattere ascii new line.

Array 3D e multidimensionali

Questo vale anche per array multidimensionali, come ad esempio array 3D oppure array 4-D etc. Quindi possiamo avere degli assi come `3x2x5` e gli indici di ordinamento come:

```
#order
1
2
0
```

il che ci porta all'array di dimensioni `5x3x2`. Questo si estende ad array N-dimensionali che sono considerati in questo esercizio (con `N<=9` il numero massimo di dimensioni, si veda sotto le Assunzioni.)

Maggiori informazioni su come si linearizzano gli array multidimensionali sono reperibilie alle slides del corso `7` e `8` che abbiamo visto in una delle lezioni su array e matrici:

- Link a slides lezione - Indirizzamento, System Calls, Lettura immagine
- Link al materiale lezione
- Altro materiale nelle lezione precedente

Esempio simile a quello sulle slide:

```
# assi
3
# dimensioni
3
2
6
# array
# da notare come i 3 Least Significant Digits indichino
# 00210 --> 2-th strato, 1-th riga, 0-th colonna.
#
# strato = 0
# ----------- 6 ---------------------
|  00000  00001  00002  00003  00004  00005
2
|  00010  00011  00012  00013  00014  00015
# strato = 1
# ----------- 6 ---------------------
|  00100  00101  00102  00103  00104  00105
2
|  00110  00111  00112  00113  00114  00115
# strato = 2
# ----------- 6 ---------------------
|  00200  00201  00202  00203  00204  00205
2
|  00210  00211  00212  00213  00214  00215
```

viene linearizzato come:

```
00000  00001  00002  00003  00004  00005 00010  00011  00012  00013  00014  00015 00100  00101
```

che senza spazi diventa:

```
000000000100002000030000400005000100001100012000130001400015001000010100102001030010400105001100
```

quindi se vogliamo invertire assi come:

```
order
2
1
0
```

deve diventare:

```
00000 00100 00200 00010 00110 00210 00001 00101 00201 00011 00111 00211
00002 00102 00202 00012 00112 00212 00003 00103 00203 00013 00113 00213
00004 00104 00204 00014 00114 00214 00005 00105 00205 00015 00115 00215
```

ossia un array da `3x2x6` a `6x2x3`, che senza spazi diventa:

```
000000010000200000100011000210000010010100201000110011100211000020010200202000120011200212000
```

ergo, in versione finale hex:

```
064c8a6ed2165c9b6fd3266cac70d4367cbd71d5468cce72d6569cdf73d7
```

Assunzioni

- L'array è multidimensionale e può avere un massimo di `9` dimensioni, ossia un array in 9D (ossia il numero di assi che viene fornito come primo valore nel file vale al più 9 compreso)
- Il numero massimo di elementi per un asse e' `20`, compreso.
- In totale, contando anche le `5` posizioni su cui sono codificati i numeri, l'array in memoria sarà grande al più `1e5` bytes.
- Se il numero di assi e' `0` allora il resto dei valori in ingresso non compare e quindi va stampato solo il carattere di accapo.

Svolgimento

Scrivere un programma assembly che prende in ingresso un file testuale (senza i commenti) in questo modo:

```
2 # assi
4 # num. elementi primo asse
3 # num. elementi secondo asse
```

```
1 # primo indice del nuovo ordine degli assi
0 # secondo indice del nuovo ordine degli assi
000010000200003000040000500006000070000800009000100001100012 # stream
```

e scriva a video tramite opportune system calls la stringa in modo tale che il vettore multidimensionale abbiamo gli assi permutati nell'ordine giusto in cui viene fornito. Per il file sopra si deve scrivere:

`147a258b369c\n`.

Attenzione, ci sono dei requisiti addizionali:

- **NON** è possibile usare **cicli annidati** nell'implementazione, mi raccomando questo è molto importante.
- Gli elementi di ogni dimensione e l'indice dei nuovo ordine degli assi deve essere memorizzato in un array di `half word`. Si veda sotto.
- Quindi, **DEVE essere usata la ricorsione**:
  - sia per generare l'array
  - sia per stampare il numero in esadecimale
- Non si può usare memoria addizionale tipo heap. Si può (deve) usare lo stack. Abbiamo dello spazio nel data segment come segue, **ma non e' possibile aggiungere niente altro al data segment, quindi NON manipolate il data segment.**
- Il mio programma in totale impiega circa `5180546` (~5.2M) di istruzioni, quindi cercate di stare sotto questa soglia.

Dettagli tecnici

Si assume che abbiamo spazio nel `data segment` in questo modo:

```
.data

string: .space 100000   # memorizzare la string di byte
dims:   .half 0:9       # memorizza gli elementi per ogni asse
order:  .half 0:9       # memorizza l'indice del nuovo ordine per ogni asse sorgente
```

Altre cose importanti:

1. Si legge il primo valore degli assi con systemcall `read_int` codice `5`.
2. A questo punto si procede a leggere i restanti numeri "singoli" sempre con system call `5`.
3. Infine si legge lo stream di byte con codice systemcall `8`. Quindi va passato in `$a0 ⟵ 8` e come lunghezza massima della stringa passate pure `1e5`. La systemcall caricherà la stringa al indirizzo `string` fintatoche' non trova il `\n` che termina la stringa. A quel punto la systemcall ritorna la programma utente e la stringa e' in memoria con in fondo ***anche il carattere*** `\n`.

4. Il consiglio che vi do è di verificare che l'input in ingresso arrivi correttamente prima di iniziare a svolgere la parte centrale dell'esercizio.

Sono allegati i file di input del programma e anche il file di testo atteso (expected output). Dato che avete expected output potete anche debuggare il risultato usando diff sui sistemi Unix.

Per i sistemi Windows per la comparazione è possibile usare programmi grafici appositi oppure seguire questo link che usa Powershell https://serverfault.com/questions/5598/how-do-i-diff-two-text-files-in-windows-powershell

Per eseguire i test in locale:

- aprite una finestra di comando/console e posizionatevi nella directory in cui avete messo i file:
- il programma `program01.asm` da voi sviluppato
- il simulatore `Mars4_5.jar`
- un file di esempio di input `00_demo.in`
- il corrispondente file di output atteso `demo_expected.txt`

Eseguite in console il comando:

- `java -jar Mars4_5.jar me nc sm ic program01.asm < 00_demo.in > demo_myoutput.txt`
- in questo modo produrrete nel file `demo_myoutput.txt` le stampe del vostro programma
- confrontate il file `demo_myoutput.txt` con il file `00_demo_expected.out`, se sono uguali il test è superato

Trovate i casi di test su cui allenarvi in allegato.

Scadenza

`27 Maggio ore 8:00`

Per maggiori informazioni su come eseguire mars con input e output si veda:

https://q2a.di.uniroma1.it/21689/homework-test-homework?course=annunci/architetture-20-21

Come sempre scrivere nei commenti sotto se trovate bug o incongruenze che sistemiamo.