LABORATORIO FONDAMENTI DI INFORMATICA
3 DICEMBRE 2019 – Incontro 8 di 8
Passaggio Parametri, Allocazione Dinamica Vettori, Liste Bidirezionali e Circolari

Esercizio 1 - Passaggio di parametri

Il Sistema Operativo può trasmettere dei parametri ad un programma scritto in C quando al momento dell'esecuzione viene invocata la funzione `main`

```
int main(int argc, char *argv[])
```

Il valore di `argc` è il numero di parametri trasmessi e `argv[]` è un vettore di stringhe che contiene i valori degli `argc` parametri.
Il primo parametro `argv[0]` esiste sempre (e quindi `argc` è sempre >= 1) e corrisponde al nome (eventualmente comprensivo di tutto il percorso nel filesystem) del file eseguibile che il Sistema Operativo ha lanciato in esecuzione.
Scrivere un programma che stampi inizialmente a video il numero di parametri ricevuti e la lista dei parametri ricevuti.
Si supponga che il programma si aspetti un numero minimo di parametri per la sua corretta esecuzione definito tramite una costante `#define MIN_ARGC_ES1 3`. Se il numero di parametri è insufficiente il programma deve terminare con un messaggio di errore.
Se il numero di parametri è sufficiente, il programma deve convertire i parametri `argv[1] ... argv[argc - 1]` in numeri interi tramite la funzione `atoi`, che riceve in ingresso una stringa e ritorna un numero intero, e stamparne a video la loro somma.
NB: La funzione `atoi` ritorna 0 nei casi in cui la stringa passata come parametro non rappresenti un numero intero valido.
E' possibile consultare il documento PDF "Passaggio di parametri" contenente le istruzioni su come passare parametri a un programma tramite IDE o tramite terminale.

```
D:\Lab2019\Laboratorio08_Soluzioni.exe
Il programma ha ricevuto argc=1 parametri.
argv[0] = D:\Lab2019\Laboratorio08_Soluzioni.exe
Errore: numero di parametri inferiore al necessario. Attesi almeno
3 VS Ricevuti 1.

D:\Lab2019\Laboratorio08_Soluzioni.exe 1 2 3
Il programma ha ricevuto argc=4 parametri.
argv[0] = D:\Lab2019\Laboratorio08_Soluzioni.exe
argv[1] = 1
argv[2] = 2
argv[3] = 3
Numero di parametri sufficiente. Attesi almeno 3 VS Ricevuti 4.
1 + 2 + 3 = 6.
```

Esercizio 2 - Allocazione dinamica di vettori

Fino ad ora nelle sessioni di Laboratorio è stata utilizzata l'allocazione statica dei vettori, cioè si è prevista a tempo di compilazione una dimensione massima del vettore (solitamente indicata per mezzo di una costante).

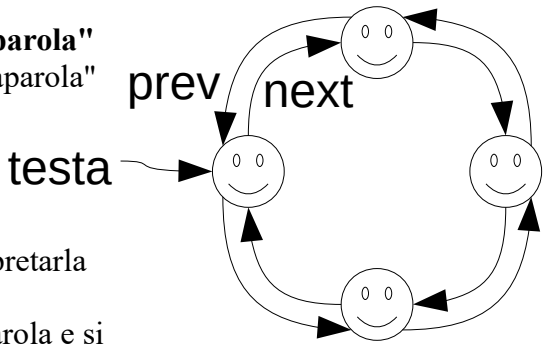
```
#define DIM_MAX 100
...
int v[DIM_MAX];
```

Tramite l'utilizzo della funzione `malloc` è però anche possibile allocare dinamicamente un vettore a tempo di esecuzione una volta noto il numero di elementi del vettore.
Scrivere un programma che riceva dal Sistema Operativo un parametro che, una volta convertito a numero intero >= 1, rappresenti la dimensione di un vettore di numeri interi da allocare

dinamicamente tramite la funzione `malloc`.
La quantità di memoria in byte necessaria si calcola come
`sizeof(int) * numero_elementi`
Stampare a video la quantità di memoria necessaria in byte.
Prevedere un messaggio di errore se la memoria disponibile è insufficiente (cioè se `malloc` dovesse ritornare `NULL`).
Riempire in modo casuale il vettore con dei numeri interi `rand() % 10`, stamparlo a video e infine liberare tramite la funzione `free(...)` la memoria occupata dal vettore.

```
D:\Lab2019\Laboratorio08_Soluzioni.exe 5
Il programma ha ricevuto argc=2 parametri.
argv[0] = D:\Lab2019\Laboratorio08_Soluzioni.exe
argv[1] = 5
Numero di parametri sufficiente. Attesi almeno 2 VS Ricevuti 2.
Ok, dimensione del vettore 5 >= 1.
Sto allocando un vettore di 5 interi, per un totale di 5 x 4 = 20
bytes.
Vettore: 3 7 2 9 3
Libero la memoria con free...
```

Esercizio 3 - Lista Bidirezionale Circolare - Gioco del "Passaparola"



Scrivere un programma che simuli il gioco tra bambini del "Passaparola" tramite l'utilizzo di una Lista Bidirezionale e Circolare.
Il gioco può svolgersi sia in senso orario che antiorario.
Il giocatore iniziale "pensa" una parola e la "sussurra" al suo "vicino".
Il vicino può comprendere esattamente la parola oppure misinterpretarla e a sua volta la trasmette al suo vicino.
Il gioco si conclude quando tutti i giocatori si sono "passati" la parola e si può verificare se nel "fare il giro" la parola si è modificata o è arrivata esattamente a destinazione.

Definire tramite una costante la dimensione massima di tutte le stringhe usate nel programma (cioè per il nome del giocatore, per le parole, ...)
```
#define DIM_TXT 100
```

Definire una costante il cui valore inverso indicherà la probabilità che una parola venga "sentita" in modo scorretto nel passare da un giocatore all'altro.
```
#define DADO 5
```

Definire tramite una costante la parola "non definita", da impostare quando non è ancora arrivato il turno di un giocatore e quindi non ha ancora "ascoltato" nessuna parola.
```
#define PAROLA_ND "n.d."
```

Definire un tipo enumerato che indichi se il gioco avviene in senso orario o antiorario
```
enum e_verso {orario, antiorario};
```

Definire un tipo strutturato `s_giocatore` che rappresenti un elemento della lista bidirezionale e circolare dei giocatori e che contenga i campi:

- `nome` (il nome del giocatore)
- `parola` (la parola che il giocatore avrà ascoltato)
- `next` (un puntatore al prossimo giocatore)
- `prev` (un puntatore al giocatore precedente)

Definire la funzione `aggiungi_giocatore`
Riceve come parametri la testa della lista dei giocatori e il nome del nuovo giocatore.
Inserisce in coda (cioè prima della testa perché la lista è circolare) il nuovo giocatore impostandogli tramite la funzione `strcpy` il nome e la parola "non definita". Restituisce la testa della lista. Fare attenzione a impostare sia l'elemento successivo `next` che l'elemento precedente `prev` e a fare in modo che la lista sia "circolare" (l'ultimo elemento deve avere come `next` la testa della lista e la testa della lista deve avere come `prev` l'ultimo elemento).

Definire la funzione `stampa_giocatori`
Riceve come parametri la testa della lista e un `enum e_verso` (orario o antiorario).
Stampa tutti i giocatori (il nome e la parola "ascoltata") nel verso indicato (se orario parte dalla testa e segue tutti i "`next`", se antiorario parte dalla testa e segue tutti i "`prev`").

Definire la funzione `ascolta`
Riceve come parametro una stringa che rappresenta una parola "ascoltata" e che verrà eventualmente modificata dalla funzione.
Nella funzione si "lancia casualmente un dado" tramite l'istruzione
```
rand() % DADO;
```
Se il risultato è 0, vanno scelte due posizioni casuali di due caratteri all'interno della parola e questi due caratteri andranno scambiati tra loro.
```
posizione1 = rand() % strlen(parola);
posizione2 = rand() % strlen(parola);
```
Ad esempio con
```
parola = "laboratorio", strlen(parola) = 11, posizione1 = 2,
posizione2 = 6
```
si otterrà la parola "latoraborio".

0	1	2	3	4	5	6	7	8	9	10	11
l	a	b	o	r	a	t	o	r	i	o	\0
l	a	t	o	r	a	b	o	r	i	o	\0

Definire la funzione `partita`
Riceve come parametri la lista dei giocatori, la parola iniziale, il verso di gioco.
Controlla che nella lista dei giocatori ci siano almeno due giocatori altrimenti ritorna subito.
Imposta nel primo giocatore la parola iniziale e poi procede a "copiarla" a seconda del verso del gioco in senso orario (il giocatore successivo è il `next`) o in senso antiorario (il giocatore successivo è il `prev`). Una volta copiata, la parola viene passata alla funzione `ascolta` che potrà in modo casuale modificare la parola ascoltata. La partita termina quando tutti i giocatori si sono passati la parola e si determina tramite la funzione `strcmp` se la parola dell'ultimo giocatore coincide con quella del primo giocatore.

Definire la funzione `rimuovi_giocatore`
Riceve come parametri la lista dei giocatori e un nome.
La funzione cerca nella lista il giocatore con il nome indicato e lo rimuove dalla lista.
Ritorna la testa della lista.

La funzione `main` deve ricevere dal Sistema Operativo almeno `argc = 7` parametri
Ad esempio "`laboratorio orario paolo luca maria piero`"
`argv[0]` nome del file eseguibile
`argv[1]` parola da indovinare
`argv[2]` senso del gioco "orario" o "antiorario"

`argv[3], argv[4], argv[5], argv[6], ...` nomi dei giocatori
La lista dei giocatori viene popolata richiamando la funzione `aggiungi_giocatore` e poi stampata tramite la funzione `stampa_giocatori`.
Inizia la partita tramite la chiamata alla funzione `partita`.
Viene poi rimosso un giocatore a caso chiamando la funzione `rimuovi_giocatore` e si rigioca la partita nel senso inverso.

```
D:\Lab2019\Laboratorio08_Soluzioni.exe laboratorio orario paolo
luca maria piero
Il programma ha ricevuto argc=7 parametri.
argv[0] = D:\Lab2019\Laboratorio08_Soluzioni.exe
argv[1] = laboratorio
argv[2] = orario
argv[3] = paolo
argv[4] = luca
argv[5] = maria
argv[6] = piero
Numero di parametri sufficiente. Attesi almeno 7 VS Ricevuti 7.
Aggiungo il giocatore paolo...
Aggiungo il giocatore luca...
Aggiungo il giocatore maria...
Aggiungo il giocatore piero...
Giocatori in senso orario:
[paolo, "n.d."] <--> [luca, "n.d."] <--> [maria, "n.d."] <-->
[piero, "n.d."] <-->  ...
Giocatori in senso antiorario:
[paolo, "n.d."] <--> [piero, "n.d."] <--> [maria, "n.d."] <-->
[luca, "n.d."] <-->  ...
Il giocatore in testa paolo riceve la parola iniziale
laboratorio...
Gioco in senso orario...
Il giocatore luca ha sentito la parola laboratorio...
Il giocatore maria ha sentito la parola laboratorio...
Il giocatore piero ha sentito la parola oablratorio...
La parola laboratorio e' arrivata modificata come oablratorio!
Giocatori in senso orario:
[paolo, "laboratorio"] <--> [luca, "laboratorio"] <--> [maria,
"laboratorio"] <--> [piero, "oablratorio"] <-->  ...
Cerco il giocatore piero per rimuoverlo dalla lista...
Giocatore piero trovato...
Rimuovo il giocatore piero...
Il giocatore in testa paolo riceve la parola iniziale
laboratorio...
Gioco in senso antiorario...
…
```

Esercizio 4 - Scrittura e Lettura di File
(DA SVOLGERE QUANDO SI AVRA' AFFRONTATO L'ARGOMENTO "FILES")
Scrivere un programma che chieda all'utente due nomi di file (uno per un file testuale con estensione .txt e uno per un file binario), un testo e i dati di una variabile `struct {int anno; int mese; int giorno;}`. Scrivere nel file testuale il testo con `fprintf` e nel file binario la variabile struct con `fwrite`. Chiudere i due file tramite la funzione `fclose( )` e poi riaprirli per rileggere i dati.

```
Nome del file testuale (.txt)? file.txt
Apertura in modalita' "w" del file testuale "file.txt"...
Testo da scrivere nel file testuale? C'era una volta un file
Scrittura nel file testuale...
Chiusura del file testuale...
Nome del file binario? file.bin
Apertura in modalita' "wb" del file binario "file.bin"...
anno? 2019
mese? 12
giorno? 3
Scrittura nel file binario...
Chiusura del file binario...
Riapertura del file testuale...
Lettura dal file testuale...
Frase letta: "C'era una volta un file"
Chiusura del file testuale...
Riapertura in modalita' "rb" del file binario...
Rilettura dal file binario...
Data letta: anno=2019 mese=12 giorno=3.
Chiusura del file binario...
```