

Elaborazione Stringhe ed ADT Coda (Coda e Array)

Prova d'Esempio — main.c

Università di Catania — Dipartimento di Matematica e Informatica

Corso di Laurea in Informatica

Programmazione 1 e Laboratorio (Canale O-Z)

Descrizione del Programma

Si scriva un programma in linguaggio C, chiamato `main.c`, per la generazione casuale ed elaborazione di stringhe tramite l'ausilio di una struttura dinamica di tipo Coda (FIFO).

Il programma deve ricevere i parametri da riga di comando (file contenente l'alfabeto, dimensione n , nome del file di output), convalidare che la stringa alfabetica abbia lunghezza compresa tra 8 e 12 caratteri, popolare un vettore allocato dinamicamente con la sequenza di Fibonacci, generare un vettore allocato dinamicamente di stringhe casuali di lunghezza definita dai numeri di Fibonacci, inserire ed elaborare sequenzialmente le stringhe in una coda dinamica con logiche di concatenazione per lunghezze dispari, e salvare il risultato finale sul file di output.

Requisiti di Implementazione

Il programma deve essere strutturato nei seguenti punti:

PUNTO A: [3 punti] Gestione Parametri (CLI): Validare i parametri da riga di comando.

- Il programma deve ricevere esattamente tre argomenti: il nome di un file di `input` (es. `input`), un intero `n` e il nome del file di `output` (es. `output`).
- Aprire il file di `input` e leggere la stringa alfabeto in esso contenuta (se ci sono più stringhe, considerare solo la prima, scartando gli spazi o a capo). Controllare che l'alfabeto abbia lunghezza compresa tra 8 e 12 caratteri inclusi, e che l'intero `n` sia compreso tra 3 e 12 inclusi. In caso di errore, stampare un errore su `stderr` e uscire con exit code 1.
- Stampare a schermo i parametri confermati e l'alfabeto letto.

PUNTO B: [8 punti] Allocazione e Calcolo del Vettore A (Fibonacci con Doppia Allocazione):

- **Punto 2a [4 punti]:** Allocare dinamicamente un array A di n puntatori a interi (`int**`) su heap. Successivamente, allocare dinamicamente ciascun intero singolarmente.
- **Punto 2b [4 punti]:** Calcolare e popolare ciascuno degli interi allocati con i primi n numeri della sequenza di Fibonacci ($F_0 = 0, F_1 = 1, F_2 = 1, \dots$). Stampare a schermo l'array A risultante dereferenziando i puntatori.

PUNTO C: [8 punti] Allocazione e Campionamento del Vettore B (Stringhe):

- **Punto 3a [4 punti]:** Allocare dinamicamente un array (vettore) di stringhe B di dimensione n (array di puntatori a char allocati su heap).
- **Punto 3b [4 punti]:** Ciascun elemento $B[i]$ deve puntare a una stringa allocata dinamicamente di lunghezza pari al valore puntato da $A[i]$ (quindi $*A[i]$) popolata da caratteri estratti casualmente dall'alfabeto tramite la funzione `get_random()`. Stampare a schermo l'array di stringhe B .

PUNTO D: [8 punti] ADT Coda ed Elaborazione Condizionale:

- **Punto 4a [4 punti]:** Definire la struttura dati per un nodo di una coda (`QueueNode`) e le relative funzioni base di inserimento in coda (`enqueue`) ed estrazione dalla testa (`dequeue`) per stringhe dinamiche.
- **Punto 4b [4 punti]:** Inizializzare la coda dinamica vuota e popolarla scorrendo sequenzialmente l'array B . Per ciascuna stringa $B[i]$, se si tratta del primo inserimento o se la sua lunghezza è pari, effettuarne l'`enqueue` regolare. Se la sua lunghezza è dispari, estrarre la stringa in testa alla coda (tramite `dequeue`), concatenare $B[i]$ davanti alla stringa estratta (es. $B[i] + \text{estratta}$), ed inserire questa stringa risultante in coda. Stampare a schermo il contenuto della coda.

PUNTO E: [3 punti] Output su File: Salvare i risultati della coda su file.

- Scrivere nel file di output indicato dai parametri le stringhe contenute nella coda (una per riga).
- Chiudere il file, liberare correttamente tutta la memoria dinamica allocata per vettori, stringhe e nodi, stampare una

conferma e terminare con exit code 0.

Specifiche delle Funzioni

Il programma dovrà essere articolato in un unico file sorgente e contenere almeno le seguenti funzioni con opportuni parametri formali:

- **decodeParameters**: valida e decodifica i parametri ricevuti da riga di comando ed estrae l'alfabeto dal file di input.
- **getFibonacciArray**: alloca dinamicamente e restituisce l'array di puntatori a intero A (`int**`) contenente la sequenza di Fibonacci.
- **sampleString**: alloca dinamicamente e restituisce una stringa di caratteri estratti casualmente.
- **getStringArray**: alloca dinamicamente l'array B e vi assegna le stringhe campionate in base a $*A[i]$.
- **enqueue** e **dequeue**: operazioni di inserimento ed estrazione sulla coda.
- **processQueue**: elabora l'array B e restituisce la testa della coda risultante.
- **writeQueueToFile**: scrive gli elementi della coda sul file di output.

Note

- **Durata della prova**: 120 minuti.
 - È **VIETATO** usare variabili globali.
 - Si inseriscano i file sorgenti direttamente nella propria home directory.
 - Accesso alla documentazione disponibile tramite il browser all'indirizzo: <https://devdocs.io/c/>.
 - Per la generazione casuale, includere e utilizzare la funzione `get_random()` riportata in traccia.
-

Output di Controllo (Esempio di Esecuzione)

Caso 1 — Esecuzione corretta con parametri d'esempio

Considerando un file `input` contenente:

```
abcdefghil
```

Eseguendo il programma con il comando:

```
./main input 9 output
```

Un esempio di output stampato a schermo dal programma è il seguente:

```
=====A=====
Parameters: inputFile=input, n=9, outputFile=output
Input string: abcdefghil
=====B=====
Fibonacci array: 0 1 1 2 3 5 8 13 21
=====C=====
String array B:

i
d
bc
lld
dfaif
ialidgdf
egiicbicehfbh
hfifel fhbbebgeaclilfg
=====D=====
Queue:
ialidgdf
egiicbicehfbhllddi
hfifel fhbbebgeaclilfgdfaifbc
```

Output salvato con successo in output

In questo scenario, il file `output` generato conterrà un testo simile al seguente:

```
ialidgdf
egiicbicehfbhllddi
hfifel fhbbebgeaclilfgdfaifbc
```

(Exit code atteso: 0)

Appendici

Questo testo contiene le istruzioni e il codice di emergenza forniti per superare eventuali blocchi logici.

APPENDICE A: Dati di Test (Bypass Lettura da File e Campionamento)

Se riscontrate problemi con l'acquisizione dei dati o con il popolamento casuale dell'array di stringhe (Modulo 3), potete caricare dei dati di test statici richiamando la seguente funzione. L'utilizzo di questa funzione annulla il punteggio del modulo Array (Punti 2 e 3).

```
void carica_dati_mock(char*** B, int n) {
    char* test_data[] = {
        "i", "d", "bc", "lld", "dfaif",
        "ialidgdf", "egiicbicehfbh", "hfifelhbbebegeaclilfg"
    };
    *B = (char**)calloc(n, sizeof(char*));
    for (int i = 0; i < n; i++) {
        int idx = i % 8;
        (*B)[i] = (char*)malloc(strlen(test_data[idx]) + 1);
        strcpy((*B)[i], test_data[idx]);
    }
    printf("\n[!] Dati di emergenza caricati (Modulo Array saltato).\n");
}
```

APPENDICE B: Strutture Dati Alternative (Downgrade ad Array)

Se l'implementazione della coda dinamica (Punto 4a) o la sua elaborazione (Punto 4b) causano crash bloccanti, potete decidere di aggirare la coda dichiarando un array statico di supporto:

```
char coda_emergenza[100][100];
int head_emergenza = 0;
int tail_emergenza = 0;
```

Sviluppate i punti successivi lavorando su questa struttura statica. In questo caso, i punti relativi alla struttura dinamica (Punti 4a, 4b) saranno valutati 0 punti, e il voto massimo dell'intera prova sarà bloccato a 18/30.