

Elaborazione Stringhe ed ADT Pila (Pila 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 Pila (LIFO).

Il programma deve convalidare i parametri da riga di comando (stringa di simboli base, numero di elementi n , file di output), acquisire n interi da standard input per calcolarne il fattoriale e memorizzarlo in un array allocato dinamicamente W , generare un array allocato dinamicamente di stringhe casuali Q campionando caratteri da quelli forniti, inserire ed elaborare selettivamente le stringhe in una pila dinamica in base alla loro parità di lunghezza, e salvare il contenuto finale della pila in un file di testo.

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: una stringa `simboli`, un intero `n` e il nome del file di `output` (es. `out.txt`).
- Controllare che la stringa `simboli` abbia lunghezza compresa tra 10 e 20 caratteri inclusi, e che l'intero `n` sia compreso tra 8 e 18 inclusi. In caso di errore, stampare un messaggio su `stderr` e uscire con exit code 1.
- Stampare a schermo i parametri confermati.

PUNTO B: [8 punti] Allocazione e Calcolo del Vettore W:

- **Punto 2a [4 punti]:** Allocare dinamicamente un array (vettore) W di n interi. Chiedere all'utente di inserire da standard input n interi x .
- **Punto 2b [4 punti]:** Per ciascun valore x letto, calcolare il fattoriale $y = x!$ ed inserirlo nell'array W nello stesso ordine di acquisizione. Stampare a schermo l'array W risultante.

PUNTO C: [8 punti] Allocazione e Campionamento del Vettore Q:

- **Punto 3a [4 punti]:** Allocare dinamicamente un array (vettore) di stringhe Q di dimensione n (array di puntatori a char allocati su heap).
- **Punto 3b [4 punti]:** Ciascun elemento $Q[i]$ deve puntare a una stringa allocata dinamicamente di lunghezza $h = W[i] \pmod L$ (dove L è la lunghezza della stringa `simboli`) popolata da caratteri estratti casualmente da `simboli` tramite la funzione `get_random()`. Stampare a schermo l'array di stringhe Q .

PUNTO D: [8 punti] ADT Pila ed Elaborazione Stringhe:

- **Punto 4a [4 punti]:** Definire la struttura dati per un nodo di una pila (`Node`) e le relative funzioni base di inserimento (`push`) ed estrazione (`pop`) per stringhe dinamiche.
- **Punto 4b [4 punti]:** Inizializzare la pila dinamica vuota e popolarla scorrendo sequenzialmente l'array Q . Per ciascuna stringa $Q[i]$, se la sua lunghezza è pari o se si tratta del primo inserimento, effettuarne il `push` sulla pila. Se è dispari, estrarre la stringa in cima (sia essa a) tramite `pop`, individuare i caratteri di $Q[i]$ che compaiono anche in a mantenendo l'ordine originale, ed inserire questa nuova stringa risultante in pila. Stampare a schermo il contenuto della pila.

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

- Scrivere nel file di output indicato dai parametri le stringhe contenute nella pila (separate da spazio su una singola riga).
- Chiudere il file, liberare correttamente tutta la memoria dinamica allocata per vettori e stringhe, 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.
- **readInput**: legge da tastiera gli input inseriti e restituisce l'array W allocato dinamicamente.
- **sampleString**: alloca dinamicamente e restituisce una stringa di caratteri estratti casualmente da **simboli**.
- **getStringArray**: alloca dinamicamente l'array Q e restituisce il puntatore corrispondente.
- **common**: date due stringhe a e b , restituisce una stringa allocata dinamicamente contenente i caratteri di a presenti in b .
- **getStack**: elabora l'array Q e restituisce la testa della pila dinamica risultante.
- **writeStackToFile**: scrive gli elementi della pila 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 reindirizzamento dell'input da file

Considerando un file `input.txt` contenente:

```
2 5 3 5 2 3 4 2 3 3
```

Eseguendo il programma con il comando:

```
./main jshdywjdufhfghdywb 10 out.txt < input.txt
```

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

Punto A - Parametri: simboli = jshdywjdufhfghdywb, output = out.txt, n = 10

Punto B - Contenuto di W:

2 120 6 120 2 6 24 2 6 6

Punto C - Contenuto di Q:

gh hufwssfdssjg dhdhfw gujwydyggyff ww dhsywf dfdgsh hd wdhbdu fjhhfu

Punto D - Contenuto della pila:

fjhhfu wdhbdu hd dfdgsh dhsywf ww gujwydyggyff dhdhfw hufwssfdssjg gh

Output salvato con successo in out.txt

In questo scenario, il file out.txt generato conterrà un testo simile al seguente:

fjhhfu wdhbdu hd dfdgsh dhsywf ww gujwydyggyff dhdhfw hufwssfdssjg gh

(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)

Se riscontrate problemi con l'acquisizione dei dati da standard input (Punto 4a) o con il popolamento casuale dell'array di stringhe (Punto 4b), potete caricare dei dati di test statici richiamando la seguente funzione. L'utilizzo di questa funzione annulla il punteggio del modulo Array.

```
void carica_dati_mock(char*** Q, int n) {
    char* test_data[] = {
        "gh", "hufwssfdjsjg", "dhdhfw", "gujwydydgyff", "ww",
        "dhsywf", "dfdgsh", "hd", "wdhbdu", "fjjhfu"
    };
    *Q = (char**)calloc(n, sizeof(char*));
    for (int i = 0; i < n && i < 10; i++) {
        (*Q)[i] = (char*)malloc(30);
        strcpy((*Q)[i], test_data[i]);
    }
    printf("\n[!] Dati di emergenza caricati (Modulo Array saltato).\n");
}
```

APPENDICE B: Strutture Dati Alternative (Downgrade ad Array)

Se l'implementazione delle strutture dinamiche (Punto 2a) o la loro elaborazione (Punto 2b) causano crash bloccanti, potete decidere di aggirare la Pila dichiarando un array statico di supporto:

```
char stack_emergenza[100][30];
int top_emergenza = -1;
```

Sviluppate i punti successivi lavorando su questa struttura statica. In questo caso, i punti relativi alla struttura dinamica (Punti 2a, 2b) saranno valutati 0 punti, e il voto massimo dell'intera prova sarà bloccato a 18/30.