

Laboratorio 8 — Pila di Interi con Lista Concatenata

Live Demo — stack_test.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 **stack_test.c**, che implementi una **pila di interi** (struttura dati LIFO) utilizzando una **lista semplicemente concatenata** allocata dinamicamente nello heap.

Il programma non è interattivo: legge da un file di testo una sequenza di operazioni da eseguire sulla pila e produce su standard output un log delle operazioni effettuate.

Il programma deve essere invocato da riga di comando nel seguente modo:

```
./stack_test test.txt
```

Il file **test.txt** contiene una operazione per riga, tra le seguenti:

- **PUSH X** — inserisce l'intero **X** in cima alla pila;
- **POP** — rimuove l'elemento in cima alla pila, se presente;
- **TOP** — legge il valore in cima alla pila senza rimuoverlo.

La struttura dati pila deve essere rappresentata tramite una lista di nodi, dove ogni nodo contiene un intero e un puntatore al nodo successivo.

Requisiti di Implementazione

Il programma deve essere strutturato nei seguenti punti:

PUNTO A: Definire la pila come lista concatenata di nodi con i tipi **Nodo** e **Pila**. Devono essere implementate le funzioni **crea_pila**, **pila_vuota**, **push**, **pop**, **top** e **svuota_pila**. Se **malloc** restituisce **NULL**, il programma deve stampare:

```
Errore: memoria insufficiente.
```

e terminare con exit status 1.

PUNTO B: Verificare che sia stato passato esattamente un argomento oltre al nome dell'eseguibile: il nome del file di test. Se il numero di argomenti è diverso da 2, stampare:

```
Errore: uso corretto ./stack_test file_test.txt
```

e terminare con exit status 1. Se il file non può essere aperto, stampare:

```
Errore: impossibile aprire il file di test.
```

e terminare con exit status 1.

PUNTO C: Leggere il file riga per riga usando `fgets` su un buffer di ampiezza sufficiente, rimuovere l'eventuale `\n` finale e analizzare il comando. È possibile usare `sscanf` oppure `strtok` per interpretare le righe `PUSH X`, `POP` e `TOP`. Le righe non conformi devono produrre il messaggio:

```
Errore: comando non valido.
```

senza interrompere l'elaborazione.

PUNTO D: Eseguire le operazioni sulla pila. Le operazioni `POP` e `TOP` su pila vuota devono stampare:

```
Errore: pila vuota.
```

Se l'operazione è valida, il programma deve stampare:

```
POP: <valore>
```

```
TOP: <valore>
```

a seconda del comando eseguito.

PUNTO E: Al termine della lettura, svuotare la pila con `svuota_pila`, chiudere il file e terminare con exit status 0.

Output di Controllo (Esempi di Esecuzione)

Caso 1 — Operazioni valide con sovrapposizione di PUSH/POP/TOP

Contenuto di `test.txt`:

```
PUSH 10
PUSH 20
TOP
POP
POP
POP
```

Comando eseguito:

```
./stack_test test.txt
```

Output atteso su standard output:

```
TOP: 20
POP: 20
POP: 10
Errore: pila vuota.
```

(Exit code atteso: 0)

Caso 2 — File inesistente

Comando eseguito:

```
./stack_test inesistente.txt
```

Output atteso:

```
Errore: impossibile aprire il file di test.
```

(Exit code atteso: 1)

Caso 3 — Comando non valido nel file di test

Contenuto di `test.txt`:

```
PUSH 5
PUSHA 10
POP
```

Output atteso:

```
Errore: comando non valido.
POP: 5
```

(Exit code atteso: 0)