Esercitazione: Implementazione in linguaggio C dell'ADT Stack con l'utilizzo di linked list

Laboratorio di Programmazione

Introduzione

un ADT (Abstract Data Type) è un modello (formale)

che generalizza ed estende i tipi di dato primitivi e le

operazioni definite su di essi;

Un ADT (es. contatore) è connotato da:

- lo stato dell'ADT (es. conteggio);
- i metodi che definiscono le operazioni ammesse sul dato astratto (modifica e/o accesso allo stato) (es. crea, elimina, azzera, incrementa, decrementa).

consente pertanto di estendere l'insieme dei tipi di dato

predefiniti del linguaggio

Esempio di ADT : La Pila

una pila (detta anche stack) è un ADT in cui l'accesso ai dati è realizzato con strategia LIFO (Last In First Out): l'operazione di estrazione restituisce l'elemento più recentemente inserito; i metodi sono:

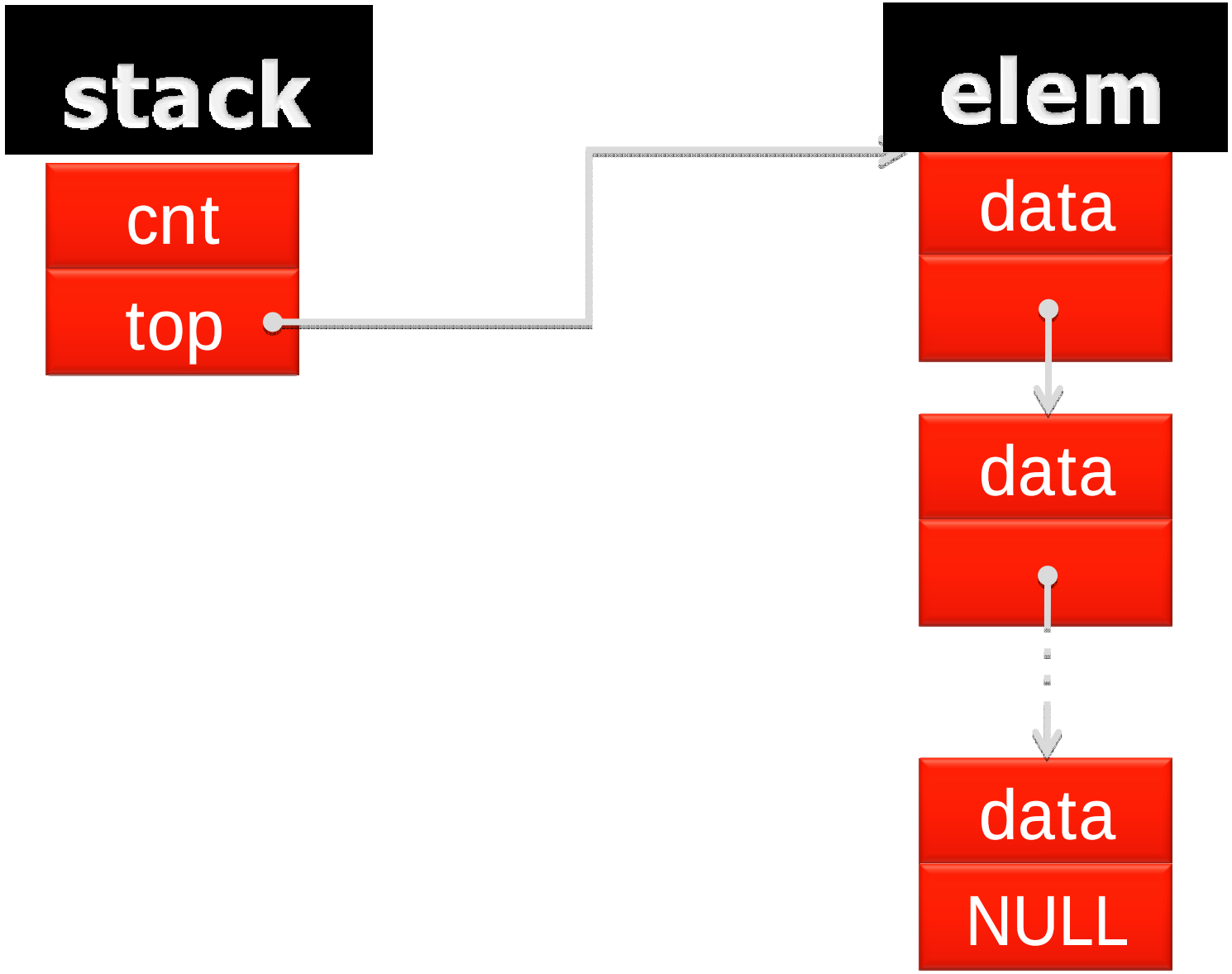
- init, crea e inizializza un'istanza vuota dell'ADT;
- push, inserisce un elemento;
- pop, estrae l'elemento più recentemente inserito;
- empty, restituisce vero se la pila è vuota, viceversa falso;
- destroy, elimina l'istanza dell'ADT.

Esempio di ADT: La coda

una coda è un ADT in cui l'accesso ai dati è realizzato con strategia FIFO (First In First Out): l'operazione di estrazione restituisce l'elemento presente da più tempo; i metodi sono:

- init, crea e inizializza un'istanza vuota dell'ADT;
- enqueue, inserisce un elemento;
- dequeue, estrae l'elemento presente da più tempo;
- empty, restituisce vero se la coda è vuota, viceversa falso;
- destroy, elimina l'istanza dell'ADT.

Struttura di uno Stack



Esempio: *stack.h*

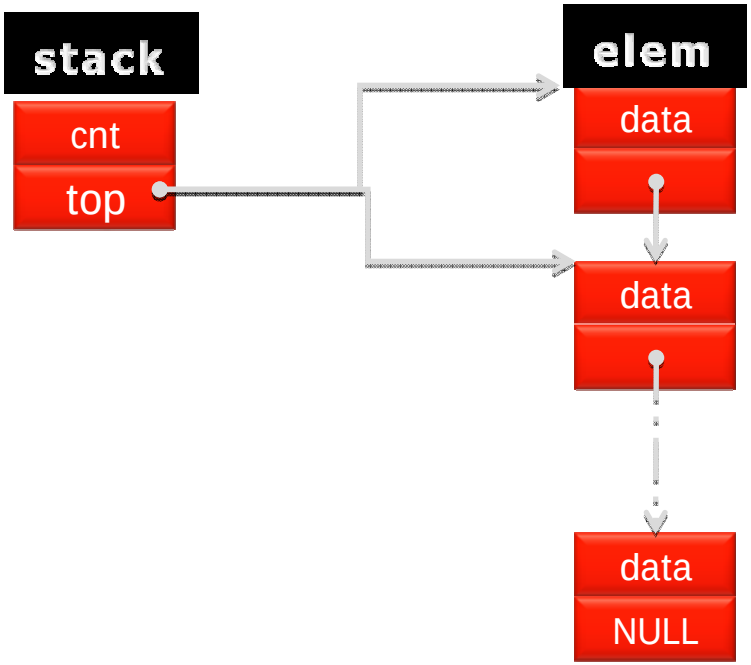
```
/* Implementazione di uno stack
   mediante una lista concatenata
   */
#include <stdio.h>
#include <stdlib.h>

#define EMPTY  0

typedef char                data;
typedef enum{false,true} boolean;

struct elem{
   data        d;
   struct elem *next;
};
typedef   struct elem   elem;
```

```
struct stack {
   int  cnt;
   elem *top;
};
typedef   struct stack   stack;

void    initialize(stack *stk);
void    push(data d, stack *stk);
void    pop(data *d, stack *stk);
boolean empty(const stack *stk);
```

Esempio: *stack.h*

```
/* Implementazione di uno stack
   mediante una lista concatenata
   */
#include <stdio.h>
#include <stdlib.h>

#define EMPTY  0

typedef char                data;
typedef enum{false,true} boolean;

struct elem{
   data         d;
   struct elem *next;
};
typedef   struct elem   elem;
```

```
struct stack {
   int  cnt;
   elem *top;
};
typedef   struct stack   stack;

void    initialize(stack *stk);
void    push(data d, stack *stk);
void    pop(data *d, stack *stk);
boolean empty(const stack *stk);
```



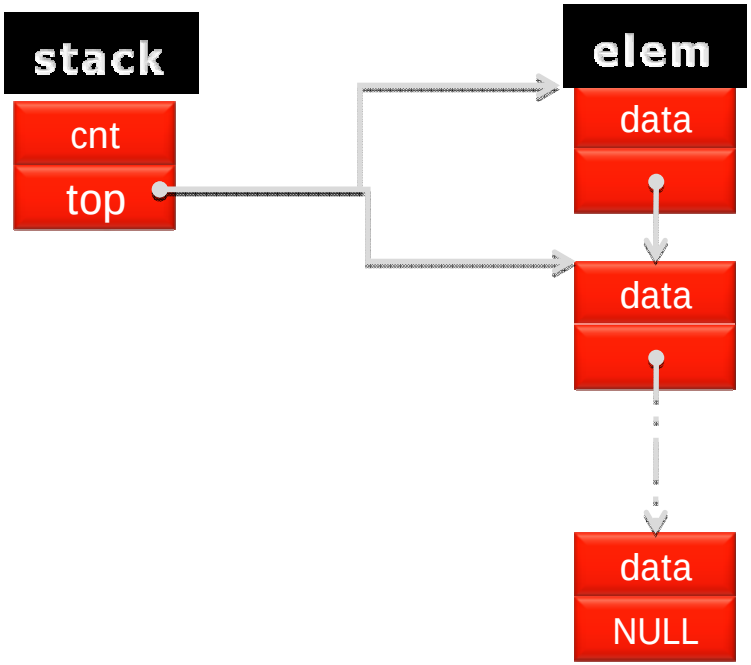
push

- Dichiara un puntatore **p** di tipo **elem**
- Alloca memoria per un elemento puntato da **p**
- **p -> d =** dato di input
- **p ->next = stk -> top**
- **stk -> top = p**
- **stk -> cnt++**

Esempio: *stack.h*

```
/* Implementazione di uno stack
   mediante una lista concatenata
   */
#include <stdio.h>
#include <stdlib.h>

#define EMPTY  0

typedef char                data;
typedef enum{false,true} boolean;

struct elem{
   data         d;
   struct elem *next;
};
typedef   struct elem   elem;
```

```
struct stack {
   int  cnt;
   elem *top;
};
typedef   struct stack   stack;

void    initialize(stack *stk);
void    push(data d, stack *stk);
void    pop(data *d, stack *stk);
boolean empty(const stack *stk);
```



pop

- Dichiara un dato **d** (già parametro di ouput)
- Dichiara un puntatore **p** di tipo elem
- **`d = stk -> top -> d`**
- **`p = stk -> top`**
- **`stk->top=stk->top->next`**
- **`stk -> cnt--`**
- Libera la memoria a cui punta **p**

1

- Implementare un set di function per la gestione di uno stack, come definite nell’esempio precedente.

Esempio: *stack.h*

```
/* Implementazione di uno stack
   mediante una lista concatenata
   */
#include <stdio.h>
#include <stdlib.h>

#define EMPTY  0

typedef char                data;
typedef enum{false,true} boolean;

struct elem{
   data        d;
   struct elem *next;
};
typedef   struct elem   elem;
```

```
struct stack {
   int  cnt;
   elem *top;
};
typedef   struct stack   stack;

void    initialize(stack *stk);
void    push(data d, stack *stk);
void    pop(data *d, stack *stk);
boolean empty(const stack *stk);
```

2

- Realizzare una function che riceva in input una stringa e restituisca in output una stringa con gli stessi caratteri in ordine inverso.
  - L’implementazione della function dovrà utilizzare uno stack.

Esempio: *simplestack.h*

```
/* Implementazione di uno stack
   mediante una lista concatenata
   */
#include <stdio.h>
#include <stdlib.h>


typedef char                data;
typedef enum{false,true} boolean;


struct head{
   data         d;
   struct head *next;
};
typedef   struct head   head;
```

```
void    initialize(head **stk);
void    push(data d, head **stk);
void    pop(data *d, head **stk);
boolean empty(const head *stk);
```

3

- Implementare un set di function per la gestione di uno stack, come definite nell’esempio precedente.

Esempio: *simplestack.h*

```
/* Implementazione di uno stack
   mediante una lista concatenata
   */
#include <stdio.h>
#include <stdlib.h>


typedef char                data;
typedef enum{false,true} boolean;


struct elem{
   data         d;
   struct elem *next;
};
typedef   struct elem   elem;
```

```
void    initialize(elem **stk);
void    push(data d, elem **stk);
void    pop(data *d, elem **stk);
boolean empty(const elem *stk);
```

FINE