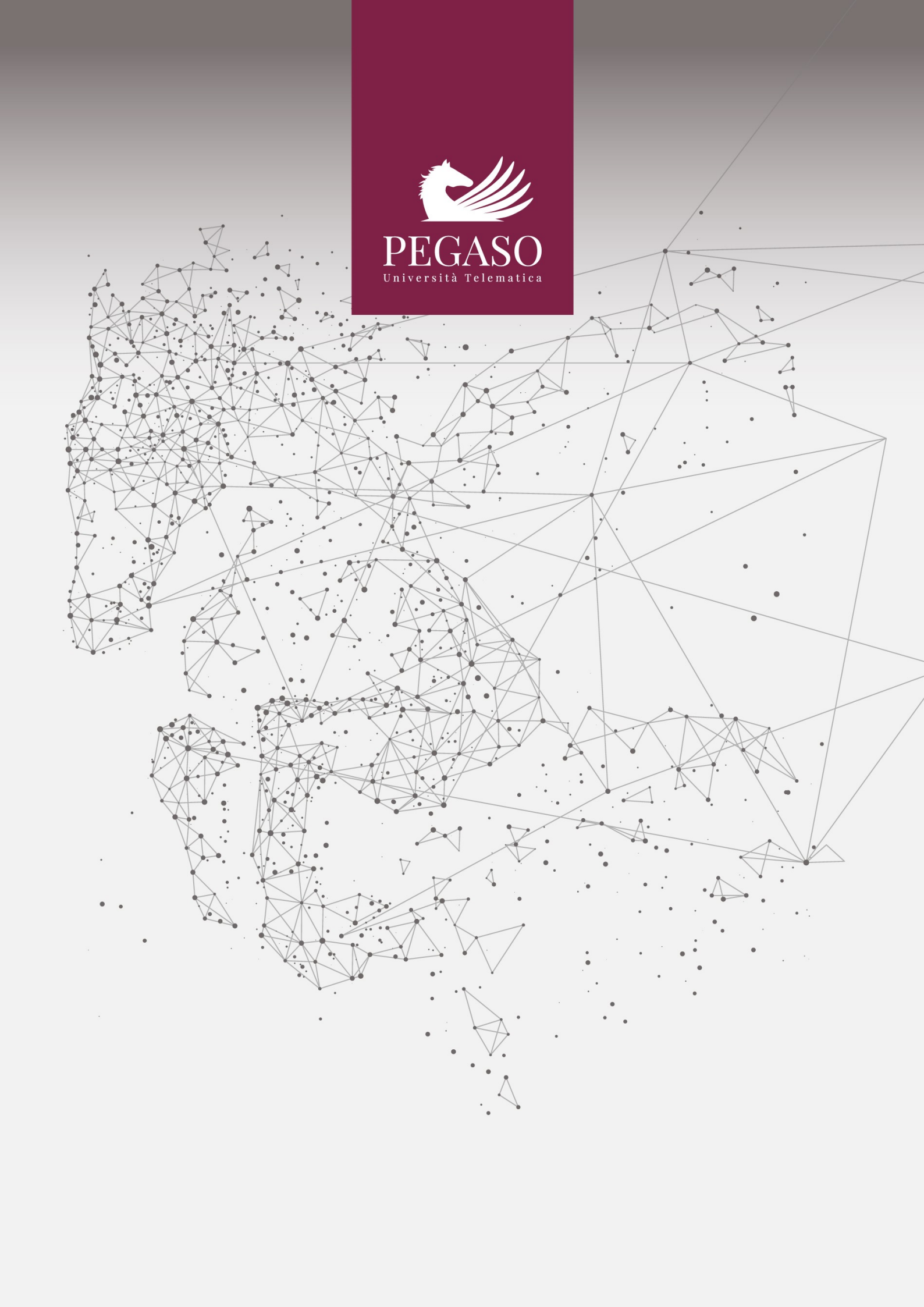
PEGASO
Università Telematica

Indice

1. Pila/Stack

Una pila (stack in inglese) è una struttura dati lineare che segue il principio LIFO (last-in, first-out). Ciò significa che l'elemento inserito per ultimo nella pila sarà il primo a essere rimosso.

Le operazioni principali che si possono effettuare su una pila sono: inserimento di un elemento (**push**), rimozione di un elemento (**pop**), e recupero dell'elemento in cima alla pila (**peek**).

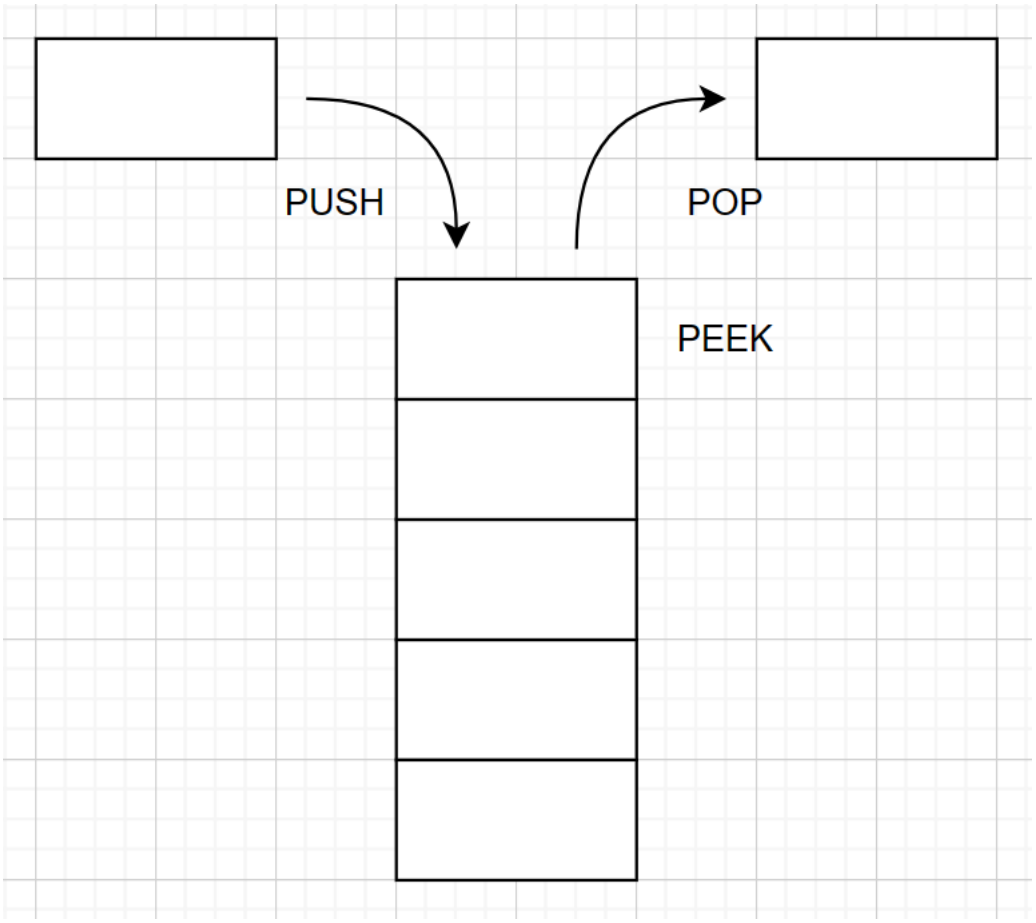

Figura 1 *- Rappresentazione di una pila*

Queste operazioni sono generalmente molto veloci, poiché l'inserimento e la rimozione avvengono sempre nella stessa posizione. Le pile sono utilizzate in molte applicazioni, come la gestione delle chiamate di sistema, la risoluzione di espressioni postfisse e la navigazione web "Indietro".

La scelta tra l'utilizzo di un array o di puntatori per implementare una pila dipende dalle esigenze specifiche del problema che si vuole risolvere. Se si conosce a priori la dimensione massima della pila, l'utilizzo di un array può essere la soluzione più semplice e veloce. In questo caso, l'inserimento e la rimozione degli elementi avvengono semplicemente incrementando o decrementando un indice che punta

alla posizione corrente della pila. Se invece non si conosce la dimensione massima della pila o se si vuole gestire dinamicamente la memoria, l'utilizzo di puntatori diventa la soluzione più flessibile. In questo caso, è possibile allocare la memoria necessaria per ogni nuovo elemento della pila, il che significa che la pila può crescere e ridursi dinamicamente a seconda delle esigenze.

In generale, l'utilizzo di puntatori per implementare una pila è più complesso rispetto all'utilizzo di un array, ma offre una maggiore flessibilità e controllo sulla gestione della memoria.

Le strutture di questo tipo prendono anche il nome di **Sistemi LIFO – Last In First Out**: l'ultimo che entra è il primo che esce; l'analogia può essere con delle persone che entrano in un ascensore, oppure addentrandoci in ambito informatico, vi sono diversi esempi in contesti differenti:

- nell'architettura dei moderni processori, fornisce il supporto per l'implementazione del concetto di funzione, le cui informazioni risiedono nella **call stack;**
- le macchine virtuali di quasi tutti i linguaggi di programmazione ad alto livello usano una pila dei record di attivazione per implementare il concetto di subroutine (generalmente, ma non necessariamente, basandosi sulla pila del processore);
- la memoria degli automi a pila dell'informatica teorica è una pila;
- la gestione di più versioni dello stesso software nel medesimo sistema operativo: per installare, occorre partire in ordine cronologico, dalla più vecchia alla più recente; per disinstallare, occorrerà seguire l'ordine opposto, dalla più recente alla più vecchia.

Di seguito riportiamo la lista delle operazioni che si svolgono su una pila:

- **IsFull**: verifica se la pila è piena.
- **IsEmpty**: verifica se la pila è vuota.
- **Push**: inserimento di un elemento nella pila (nuovo elemento affiorante nella pila).
- **Pop**: eliminazione di un elemento dalla pila (elemento affiorante).
- **Peek**: restituisce (senza estrarlo) l'elemento affiorante della pila.
- **Clear**: cancella tutti i dati della pila (esegue il Pop ripetutamente fino a che IsEmpty è pari a true).

2. Esempio di applicazione

Un esempio di utilizzo di una pila può essere per la valutazione di una espressione. Supponiamo ad es. di avere l'espressione (3*(2+5))

Utilizziamo poi 3 pile:

- Una per i simboli della formula da valutare.
- Una per gli operatori.
- Una per i valori parziali generati dalla valutazione.

Inizialmente si ha tale scenario:

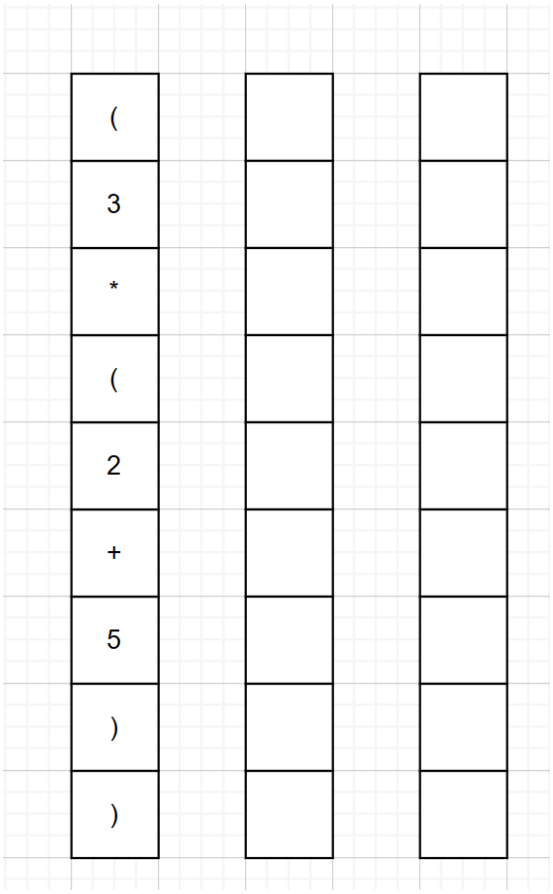

Figura 2 *- Situazione iniziale*

Nella prima pila sono inseriti tutti i simboli in modo tale che il primo elemento sia quello "affiorante".

Si inizia a fare **pop** finché non si incontra la prima parentesi chiusa e si opera nel seguente modo:

- Si trascurano le parentesi aperte.
- Si inseriscono gli operatori nella seconda pila.
- Si inseriscono i numeri nella terza pila.

Quando si incontra la prima parentesi chiusa, la situazione nelle pile è la seguente:

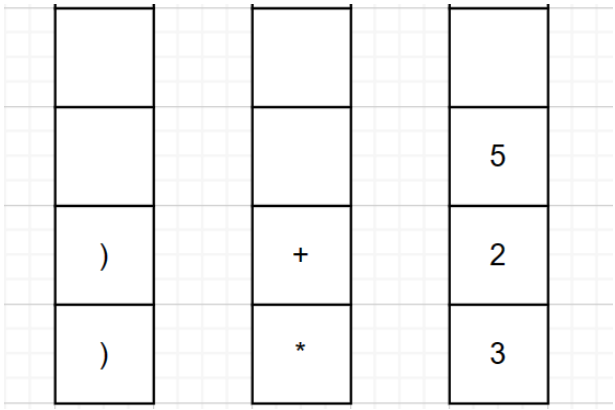

Figura 3 *- Si arriva alla prima parentesi chiusa*

Si esegue il **pop** di un operatore dalla seconda colonna (in questo caso è il **+**) e si fa il **pop** dalla terza colonna, tante volte quante previsto dall'operando (in questo caso si prelevano prima il **5** e poi il **2**); a questo punto si esegue l'operazione individuata dagli elementi estratti e dunque: **2+5**. Il risultato viene poi inserito nella terza colonna (si esegue il **push**):

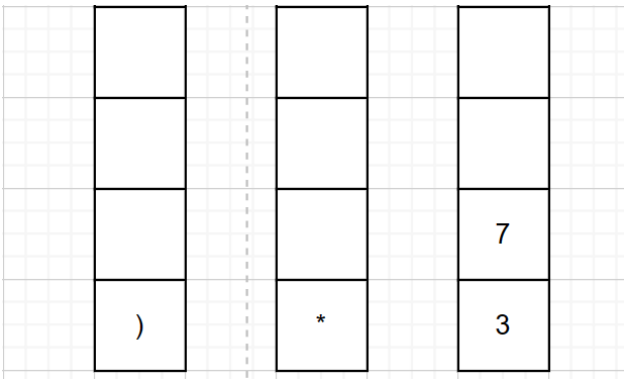

Figura 4 *- Push del risultato della prima operazione*

Si continua come al passo iniziale, perciò si inizia a fare **pop** finché non si incontra la prima parentesi chiusa e si opera come prima estraendo dunque l'operatore ***** e gli operandi **7** e **3**. Si esegue dunque l'operazione: **3*7=21** e dunque si è arrivati alla conclusione perché nella pila iniziale non c'è più nulla.

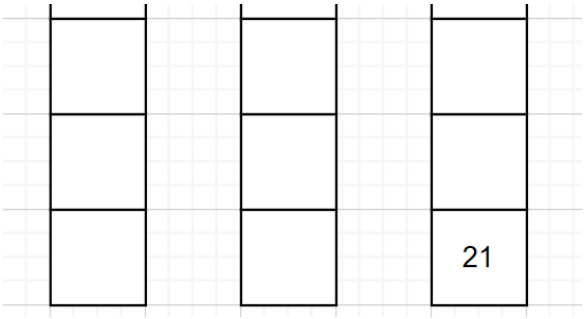

Figura 5 *- Calcolo finale*

3. Implementazione

Riportiamo di seguito le varie implementazioni iniziando dallo scenario basato su una struttura dati array:

```
#include <stdio.h>
#include <stdbool.h>

#define MAX_SIZE 100

int stack[MAX_SIZE];
int top = -1;

bool isEmpty() {
  return top == -1;
}

bool isFull() {
  return top == MAX_SIZE-1;
}

void push(int value) {
    if (isFull()) {
        printf("Pila Full\n");
        return;
    }

    top++;
    stack[top] = value;
}

int pop() {
    if (isEmpty()) {
        printf("Pila vuota!\n");
        return -1;
    }
```

```
        int value = stack[top];
        top--;
        return value;
    }
```

Passiamo ora all'implementazione mediante puntatori:

```
    #include <stdio.h>
    #include <stdlib.h>

    // struttura per rappresentare un nodo della pila
    struct StackNode {
        int data;
        struct StackNode* next;
    };

    // struttura per rappresentare la pila
    struct Stack {
        struct StackNode* top;
    };

    // funzione per creare un nuovo nodo della pila
    struct StackNode* newNode(int data) {
        struct StackNode* node = (struct StackNode*) malloc( sizeof( struct
StackNode));
        node->data = data;
        node->next = NULL;
        return node;
    }

    // funzione per creare una nuova pila vuota
    struct Stack* createStack() {
        struct Stack* stack = (struct Stack*) malloc( sizeof(struct Stack));
        stack->top = NULL;
        return stack;
    }

    // funzione per controllare se la pila è vuota
```

```
int isEmpty(struct Stack* stack) {
    return !stack->top;
}

// funzione per inserire un elemento nella pila
void push(struct Stack* stack, int data) {
    struct StackNode* node = newNode(data);
    node->next = stack->top;
    stack->top = node;
    printf("%d inserito nella pila\n", data);
}

// funzione per rimuovere l'elemento in cima alla pila
int pop(struct Stack* stack) {
    if (isEmpty(stack))
        return -1;
    struct StackNode* temp = stack->top;
    int popped = temp->data;
    stack->top = stack->top->next;
    free(temp);
    return popped;
}

// funzione per recuperare l'elemento in cima alla pila
int peek(struct Stack* stack) {
    if (isEmpty(stack))
        return -1;
    return stack->top->data;
}
```

Implementazione con raw pointers in C++:

```
#include <iostream>

using namespace std;

class Stack {
private:
```

```
    struct Node {
        int data;
        Node* next;
    };
    Node* top;

public:
    Stack() { top = nullptr; }

    void push(int value) {
        Node* new_node = new Node;
        new_node->data = value;
        new_node->next = top;
        top = new_node;
    }

    int pop() {
        if (top == nullptr) {
            cout << "Pila vuota!\n";
            return -1;
        }

        int value = top->data;
        Node* temp = top;
        top = top->next;
        delete temp;
        return value;
    }
};
```

ATTENZIONE: Non è possibile accedere direttamente agli elementi all'interno della pila senza rimuoverli, poiché la pila è una struttura di dati LIFO (Last-In-First-Out), in cui l'ultimo elemento inserito è il primo ad essere rimosso; una eventuale funzione di "stampa" dei valori presenti nella pila dovrebbe essere gestita in maniera tale per cui si dovrebbe prima eseguire una copia della pila e poi eseguire i vari pop per estrarre (e leggere) gli elementi.

4. Complessità

In generale, le operazioni di base eseguite su una pila sono di complessità costante $O(1)$. Ciò significa che il tempo richiesto per eseguire un'operazione non dipende dalla dimensione della pila, ma è sempre lo stesso.

Ad esempio, inserire un elemento in cima alla pila (push) e rimuovere l'elemento in cima alla pila (pop) sono entrambi operazioni di complessità costante.

La verifica se una pila è vuota (empty) è anch'essa un'operazione di complessità costante. Questa proprietà di complessità costante rende la pila uno strumento utile per molte applicazioni in cui è necessario mantenere l'ordine degli elementi, ad esempio in molti algoritmi di ricerca e ordinamento.

Bibliografia

- Alan Bertossi, Alberto Motresor: Algoritmi e strutture di dati, Città Studi Edizioni, terza edizione.
- C. Demetrescu, I. Finocchi, G. F. Italiano: Algoritmi e strutture dati, McGraw-Hill, seconda edizione.
- Crescenzi, Gambosi, Grossi: Strutture di Dati e Algoritmi, Pearson/Addison-Wesley.
- Sedgewick: Algoritmi in C, Pearson, 2015.
- Cormen Leiserson Rivest Stein-Introduzione Agli Algoritmi E Strutture Dati-Prima Edizione.