

# Realizzazione ADT Pila

## Laboratorio di Algoritmi e Strutture Dati

**Domenico Redavid**  
**redavid@di.uniba.it**

**Materiale di base gentilmente concesso  
dal dott. Nicola Di Mauro  
Ricercatore presso l'Univ. di Bari**

# ADT Pila

- Gestione dati con criterio LIFO
  - Last In First Out
- Implementazione con vettore
- Introduzione di Template
- Esercizi

# Pile: rappresentazione con vettore

```
#ifndef _PILAV_H
#define _PILAV_H
#include <nodo.h>

const int MAXLUNGH=100;

class Pila{
friend void stampapila(Pila & L);
public:
    Pila();
    ~Pila();
    void creaPila();
    bool pilaVuota() const;
    tipoelem leggiPila() const;
    void fuoriPila();
    void inPila(tipoelem);
private:
    Nodo elementi[MAXLUNGH];
    int testa;
};
#endif // _PILAV_H
```

**pilav.h**

## LA SPECIFICA SINTATTICA

TIPI: PILA, BOOLEAN, TIPOELEM

OPERATORI:

CREAPILA	:	( )	→	PILA
PILAVUOTA	:	(PILA)	→	BOOLEAN
LEGGIPILA	:	(PILA)	→	TIPOELEM
FUORIPILA	:	(PILA)	→	PILA
INPILA	:	(TIPOELEM, PILA)	→	PILA

## LA SPECIFICA SINTATTICA

TIPI: PILA, BOOLEAN, TIPOELEM

OPERATORI:

CREAPILA	:	( )	—————▶	PILA
PILAVUOTA	:	(PILA)	—————▶	BOOLEAN
LEGGIPILA	:	(PILA)	—————▶	TIPOELEM
FUORIPILA	:	(PILA)	—————▶	PILA
INPILA	:	(TIPOELEM, PILA)	—————▶	PILA

# La classe nodo

```
#ifndef _NODOPV_H
#define _NODOPV_H
#include <iostream>

typedef int tipoelem;

class Nodo{
public:
    Nodo();
    Nodo(tipoelem);
    ~Nodo();
    void setElemento(tipoelem);
    tipoelem getElemento() const;
    bool operator ==(Nodo &);
private:
    tipoelem elemento;
};

std::ostream &
operator<<(std::ostream &,
const Nodo & nodo);

# endif // _NODOPV_H
```

**nodopv.h**

```
#include <nodopv.h>

using namespace std;

Nodo::Nodo(){}

Nodo::~~Nodo(){}

Nodo::Nodo(tipoelem label){
    elemento=label;
}

void Nodo::setElemento(tipoelem label){
    elemento=label;
}

tipoelem Nodo::getElemento() const{
    return elemento;
}

bool Nodo::operator==(Nodo & n){
    return (getElemento() == n.getElemento());
}

std::ostream & operator<<(std::ostream & out,
const Nodo & nodo){
    return out << nodo.getElemento();
}
```

**nodopv.cpp**

# Implementazione classe Pila

```
#include <iostream>
#include <pilav.h>

using namespace std;

Pila::Pila(){ creaPila(); }

Pila::~Pila() {};

void Pila::creaPila(){
    testa=0;
}

bool Pila::pilaVuota() const {
    return testa==0;
}

tipoelem Pila::leggiPila() const {
    return elementi[testa-1].getElemento();
}

void Pila::fuoriPila() {
    if (!pilaVuota())
        testa-=1;
    else
        cout<<"pila vuota"<<endl;
}
```

```
void Pila::inPila(tipoelem el){
    if (testa==MAXLUNGH)
        cout<<"capacità massima raggiunta"<<endl;
    else {
        elementi[testa].setelemento(el);
        testa++;
    }
}
```

**pilav.cpp**

# La funzione stampaPila

Come implementare la funzione di stampa degli elementi presenti nella pila?

**// funzione friend della classe Pila**

```
void stampapila(Pila & L){  
    for (int i = 0; i < testa; i++)  
        cout << L.elementi[i] << " ";  
}
```

**// utilizzando gli operatori: distrugge la pila**

```
void stampapila(Pila & L){  
    while (!L.pilavuota()) {  
        cout << L.leggiPila() << " ";  
        L.fuoriPila();  
    }  
}
```

**// ricorsione tail: visualizzazione in ordine inverso**

**// con ricostruzione**

```
void stampapila(Pila & L){  
    tipoelem Elemento;  
    while (!L.pilavuota()) {  
        Elemento = L.leggiPila(L);  
        L.fuoriPila();  
        stampapila(L);  
        cout << Elemento << " ";  
        L.inPila(Elemento);  
    }  
}
```

- friend della classe
- accesso a private

- utilizzo di operatori ?????
- effetto distruttivo della pila

- visualizzazione inversa
- ricorsione tail

# Template di classe Pila

```
#ifndef PILAVT_H
#define PILAVT_H

#include <iostream>
using namespace std;

template <class Elemento>
class Pila
{
public:
    typedef Elemento tipoelem;
    Pila();                // costruttori
    Pila(int);
    ~Pila();              // distruttore
    void creaPila();      //operatori
    bool pilaVuota() const;
    tipoelem leggiPila() const;
    void fuoriPila();
    void inPila(tipoelem);
private:
    tipoelem *elementi;
    int testa;
};
```

**pilavt.h**

## Interfaccia

- Pila(int)
- typedef Elemento tipoelem
- tipoelem \*elementi



# Template di classe Pila

## Implementazione

```
#include <iostream>
#include <pilavt.h>
using namespace std;

template <class Elemento> Pila<Elemento>::Pila(){
    elementi = new tipoelem[100]; // dimensione standard della pila
    MAXLUNGH = 100; creaPila();
}

template <class Elemento> Pila<Elemento>::Pila(int N){
    elementi = new tipoelem[N]; // dimensione N della pila
    MAXLUNGH = N; creaPila();
}

template <class Elemento> Pila<Elemento>::~~Pila() { delete[] elementi; };

template <class Elemento> void Pila<Elemento>::creaPila(){ testa=0; }

template <class Elemento> bool Pila<Elemento>::pilaVuota() const{ return (testa==0); }

template <class Elemento> Elemento Pila<Elemento>::leggiPila() const{ return elementi[testa-1]; }

template <class Elemento> void Pila<Elemento>::fuoriPila(){
    if (!pilaVuota()) testa-=1;
    else{ cout<<"nessun elemento nella pila"<<endl; }
}

template <class Elemento> void Pila<Elemento>::inPila(tipoelem el){
    if (testa==MAXLUNGH) cout<<"raggiunta capacità massima della pila"<<endl;
    else{ elementi[testa]=el; testa++; }
}

#endif // _PILAVT_H
```

# Esercizi

- Realizzazione con puntatori della struttura dati Pila
- Scrivere una funzione che stabilisca se una data stringa è palindromica (può essere letta indifferentemente da sx a dx o viceversa)
  - ANNA
  - AAAABBBB
  - AAACAAA
  - ACCAVALLAVACCA