UNIVERSITY OF TRENTO - Italy
Department of Information
and Communication Technology

Laboratorio 20

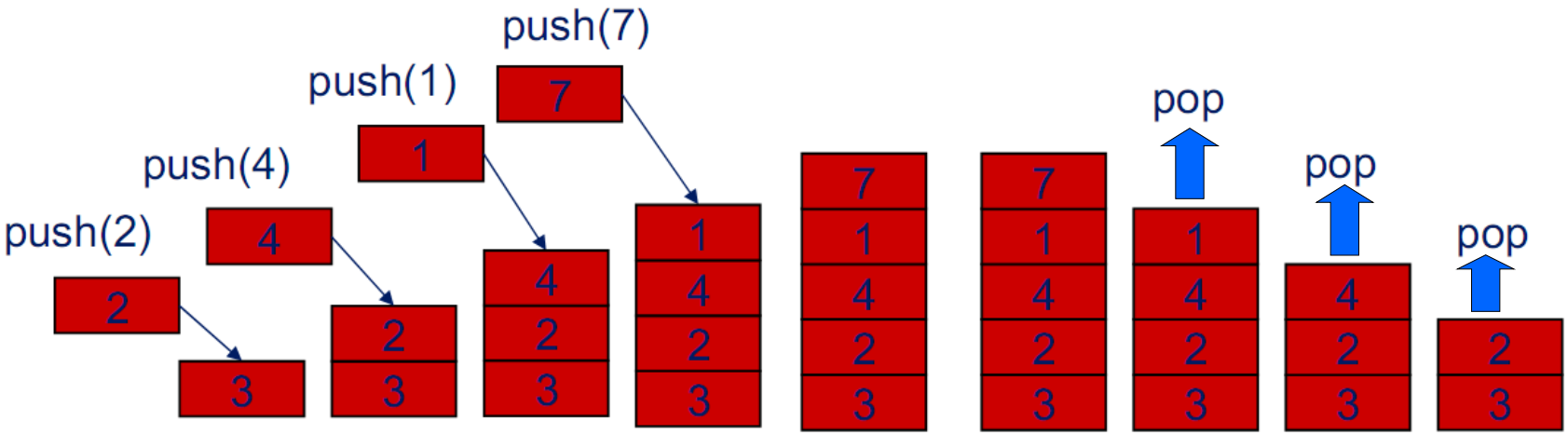
DISI – aa 2024/25

Pierluigi Roberti
Carmelo Ferrante

Università degli Studi di Trento

ADT: Stack

- Uno **stack**, o **pila**, o lista **LIFO** (last in first out, come in una pila di piatti) è una struttura dati astratta (ADT) che consente due operazioni:
  - Inserimento di un nuovo elemento in cima alla pila (**push**)
  - Estrazione di un elemento dalla cima della pila (**pop**)

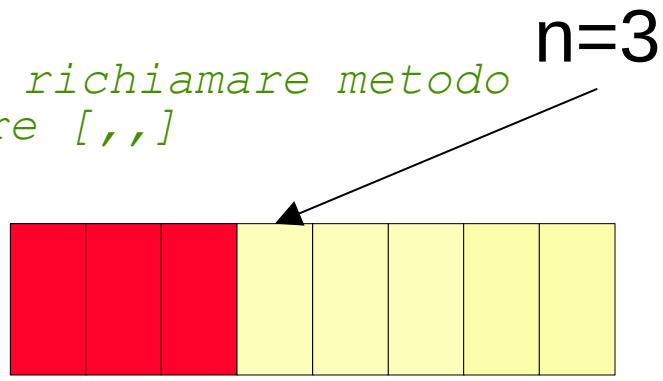


Definizione di Tdato

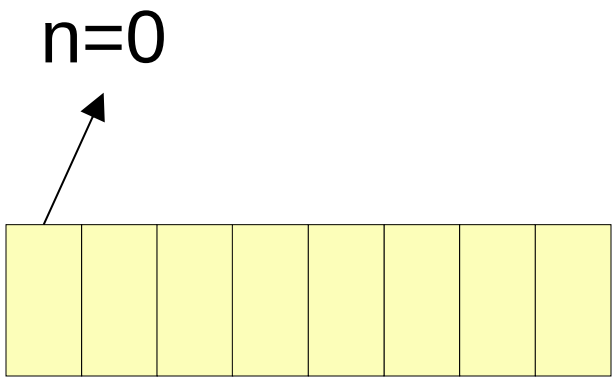
```
typedef struct Tdato {
    int x, y, z;
    // costruttore default
    Tdato() {
       // tutti i valori pari a 0
    }
    // costruttore specifico
    Tdato(int _x, int _y, int _z){
       // valori dei tre elementi passati come parametri
    }
    // metodo stampa
    void stampa()const{
       // formato richiesto: [x,y,z]
    }
} Tdato;
```

Definizione del TipoStack

```
typedef struct TipoStack {
   int n;                    // numero elementi presenti
   int dim;                  // dimensione massima array
   Tdato* s;                 // array

   TipoStack (int _dim) { // costruttore
      n = 0;
      dim = _dim;
      s = new Tdato [dim];
   }
//aggiungere:
  // costruttore default (dimensione pari a 3)
  // distruttore (dealloco array s)
  // metodo stampa: se dato presente richiamare metodo
   stampa di Tdato altrimenti stampare [,,]
} TipoStack;

typedef TipoStack Stack;
typedef TipoStack* StackPtr;
```

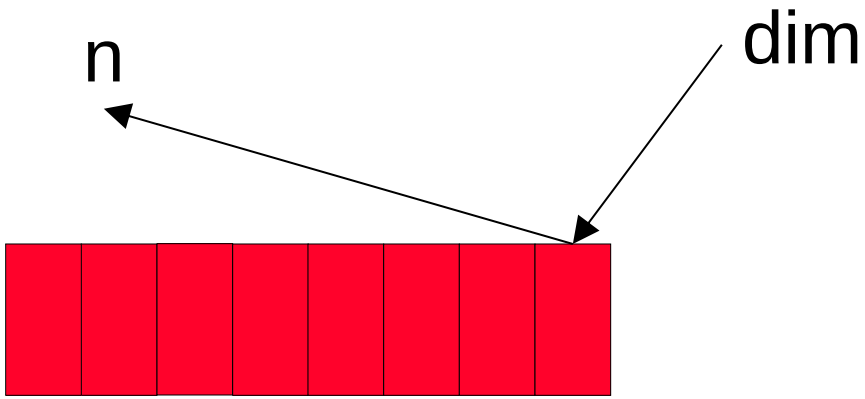


Stack (LIFO) con array



Stack vuoto

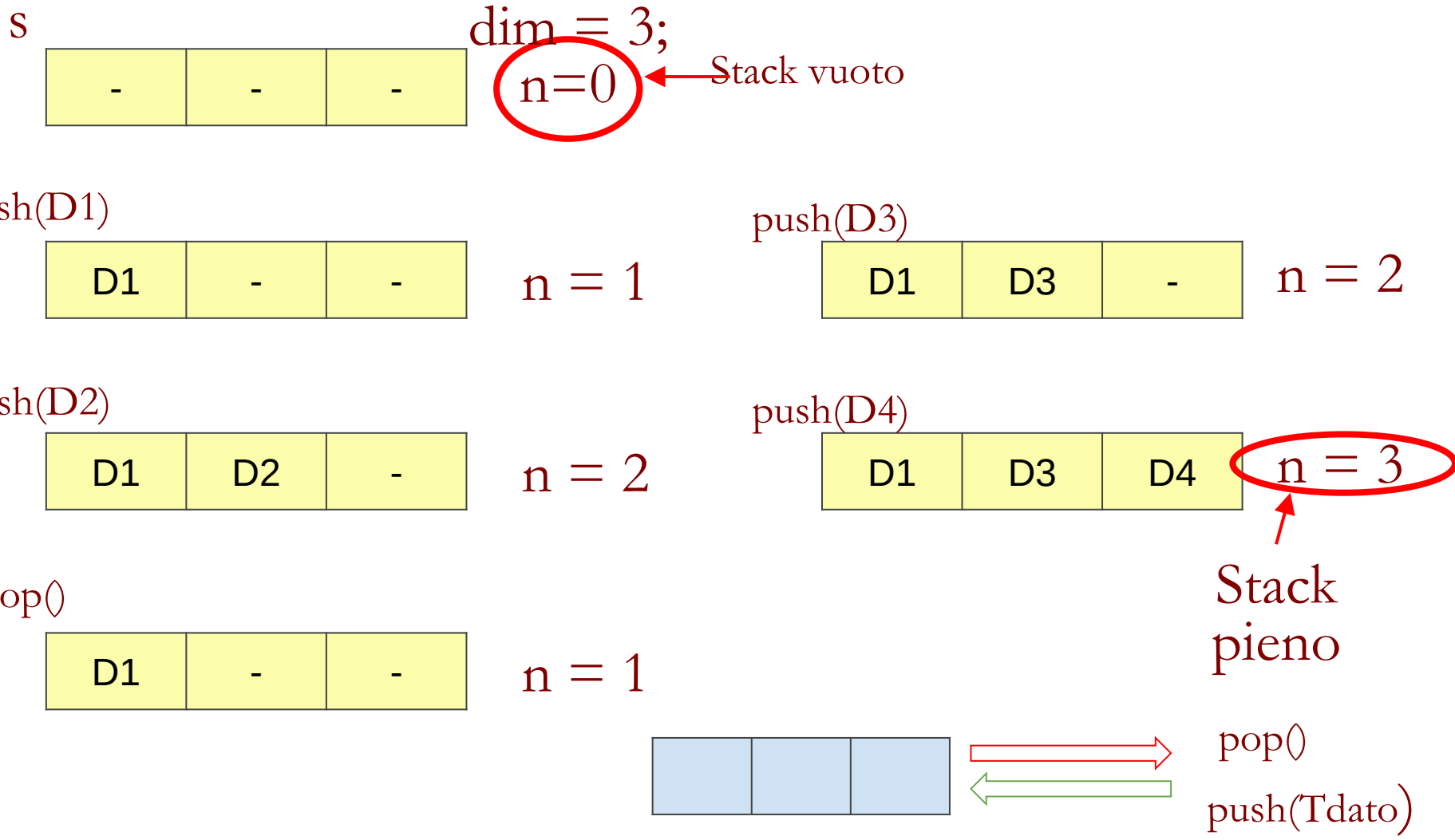
n == 0



Stack pieno

n == dim

Stack (LIFO) – Esempio



Esercizio 1 – Stack con array

- Vogliamo scrivere un programma che implementa uno **stack** di usando i vettori
- Andremo a definire le seguenti funzioni:
  - **bool stackIsFull(StackPtr p)** : controlla se lo stack è pieno
  - **bool stackIsEmpty(StackPtr p)** : verifica se lo stack è vuoto o no
  - **void push(StackPtr p, Tdato x)** : inserisce x (eventuale controllo disponibilità spazio)
  - **Tdato pop(StackPtr p)** : estrae l'elemento in cima allo stack
  - **void stampa(StackPtr p)** : stampa il contenuto dello Stack
- Creare i file **stack.h** e **stack.cpp** che implementano lo stack e le funzioni per usarlo
- Scrivere un file con il main che prova le funzionalità dello stack (push, pop, stackIsEmpty, etc)

Funzioni da implementare

```
// Verifica se lo stack è pieno o no
bool stackIsFull(StackPtr p);

// Verifica se lo stack è vuoto o no
bool stackIsEmpty(StackPtr p);

// Inserisce l'elemento d nello stack incrementando
   la dimensione dello stack
void push(StackPtr p, Tdato d);

// Rimuove un elemento dallo stack, riducendo la
   dimensione dello stack e ritorna il valore
Tdato pop(StackPtr p);

// Stampa (a video) del contenuto dello Stack
void stampa(StackPtr p);
```

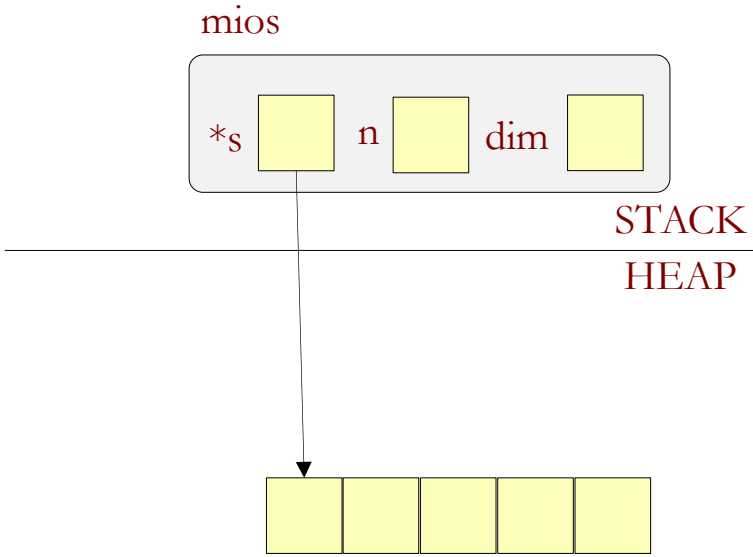
Metodi da implementare

```
// Verifica se lo stack è pieno o no
bool isFull(){ }

// Verifica se lo stack è vuoto o no
bool isEmpty(){ }

// Inserisce l'elemento intero d nello stack
   incrementando la dimensione dello stack
void push(Tdato d){ }

// Rimuove un elemento dallo stack, riducendo la
   dimensione dello stack e ritorna il valore
Tdato pop(){ }

// Stampa (a video) del contenuto dello Stack
void print(){ }
```

Esercizio 1 main di test#1

Struttura nella memoria STACK

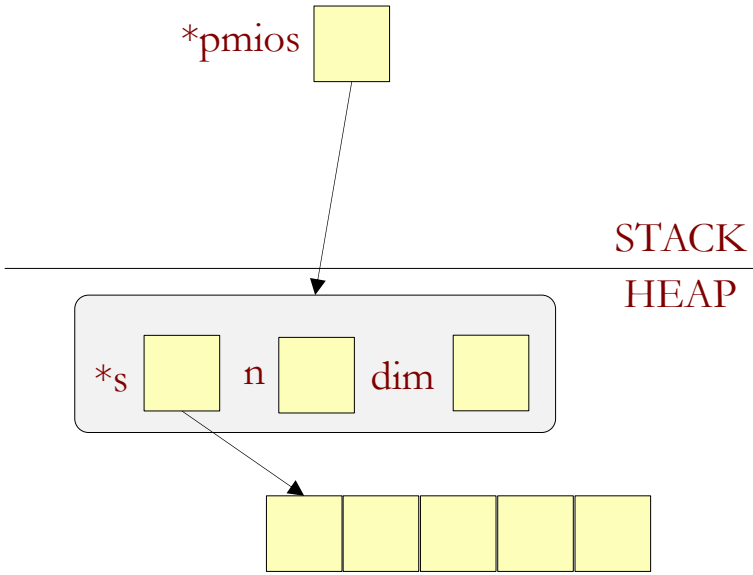
```
#include <cstdlib>
#include <iostream>
#include "stack.h"
using namespace std;
int main(int argc, char *argv[]){
  TipoStack mios(5);  //struttura nella memoria STACK
  Tdato d(9,8,7);
  mios.stampa();
  mios.push(Tdato());
  mios.push(Tdato(4,3,2));
  mios.push(d);
  mios.stampa();
  if(!mios.stackIsEmpty()){
    d = mios.pop();
    mios.stampa();
  }
}
```



Esercizio 1 main di test#2

Struttura nella memoria HEAP

```
#include <cstdlib>
#include <iostream>
#include "stack.h"
using namespace std;
int main(int argc, char *argv[]){
  TipoStack* pmios;   //puntatore nella memoria STACK
  pmios = new TipoStack(8); //allocazione nella momeoria HEAP
  Tdato d(9,8,7);
  pmios->stampa();
  pmios->push(Tdato());
  pmios->push(Tdato(4,3,2));
  pmios->push(d);
  pmios->stampa();
  if(!pmios->stackIsEmpty()){
    d = pmios->pop();
    pmios->stampa();
  }
  delete pmios;
}
```



Esercizio 2

Nel main dichiarare un array **pv** di NP=3 puntatori a stack, ognuno dei quali di 4 elementi

- Opzionale: definire funzione init

```
void init(StackPtr v[], int n_elem, int dim);
```

Ripetere per K volte (con K dichiarata costante le seguenti operazioni):

- Inizializzare una variabile d di tipo Tdato con x,y,z casuali compresi tra 0 e 9 (fare una funzione che ritorna una variabile di tipo Tdato)

```
Tdato dato_casuale();
```

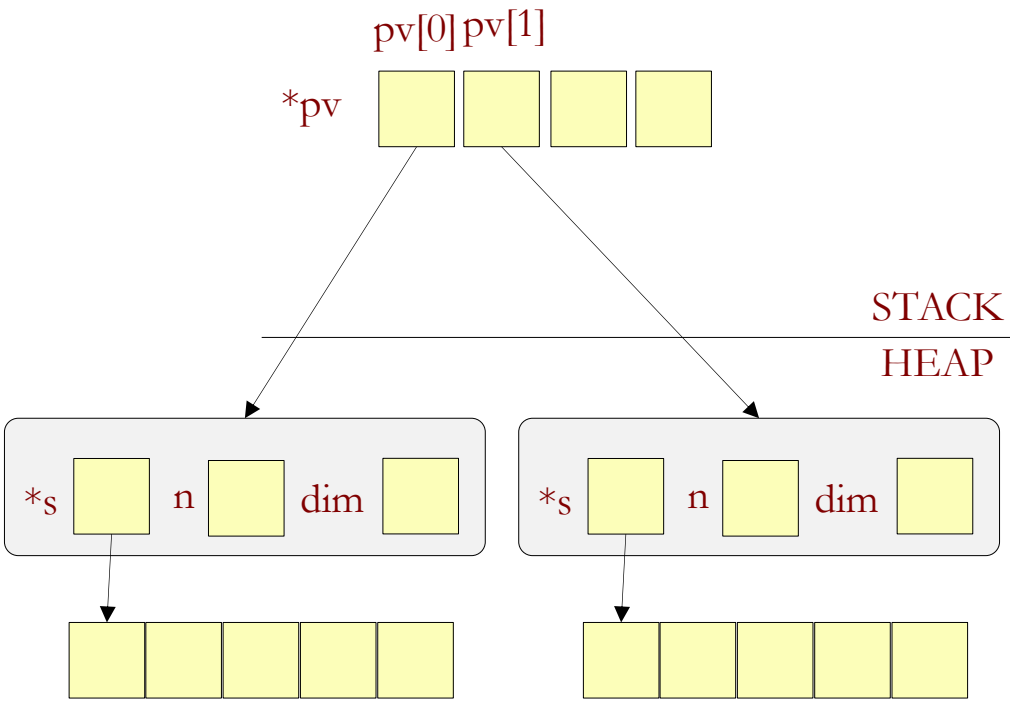
- Selezionare uno stack in modo casuale ed aggiungere la variabile d a tale stack
- Stampare il contenuto dei 3 stack (definire una funzione a cui passare l'array di puntatori a stack)

```
void printArray(StackPtr v[], int n_elem);
```

Esercizio 2 main di test

Strutture nella memoria HEAP

```
#include <cstdlib>
#include <iostream>
#include "stack.h"
using namespace std;
int main(int argc, char *argv[]){
  TipoStack* pv[4]; // array di puntatori a stack
  for(int i=0; i<4 ; ++){
    pv[i] = new TipoStack(5);
  }
  Tdato d;
  for(int k=0; k<10; k++){
     //pv[0]->push(d);
     //d = pv[1]->pop();
  }

  for(i=0; i<4 ; ++){
     delete pv[i];
  }
}
```



Esercizio 3

- Estensione esercizio precedente
  - Definire una funzione cerca_stack_grande che restituisce l'indice dello stack con il massimo numero di elementi

```
int cerca_stack_grande(StackPtr v[], int n_elem);
```

  - Definire una funzione cerca_massimo che restituisce l'indice dello stack il cui elemento da estrarre ha il valore x massimo

```
int cerca_massimo(StackPtr v[], int n_elem);
```

  - Definire una funzione cerca_coppia che restituisce VERO se esiste un elemento da estrarre che ha due valori uguali tra x, y e z. Restituisce FALSO altrimenti

```
bool cerca_coppia(StackPtr v[], int n_elem);
```

Esempio con dato intero

```
// stack_int.h
typedef struct TStack{
  int n;
  int dim;
  int* s;     // array di elementi dello stack
   TStack(){ n = 0; dim = 4; s = new int[4];}
   TStack(int _dim){ n = 0; dim = _dim; s = new int[_dim];}
  ~TStack(){ delete [] s; }
  void stampa()const{
    if(n==0){ cout << "stack vuoto" << endl; return; }
    int i=0;
    for(i=0 ; i<n ; i++)
      cout << s[i] << " ";
  }
} TStack;

typedef TStack Stack;
typedef TStack* StackPtr;
```

!! Preferibile la soluzione generale dove il contenuto di ogni elemento dello stack è una struttura di tipo TipoDato !!

Esempio con dato intero

```
#include "stack_int.h"
using namespace std;

bool stackIsFull(StackPtr p)
{ return (p->n == p->dim); }

bool stackIsEmpty(StackPtr p)
{ return (p->n ==0); }

void push(StackPtr p, int d) {
  if ( stackIsFull(p)) {
    cout << "stack pieno " << endl;
    return;
  }
  p->s[p->n]=d;
  (p->n)++;
}
```

Esempio con dato intero

```
int pop(StackPtr p) {
  if (!(stackIsEmpty(p))) {
    (p->N)--;
    return p->s[p->N];
  }
  else{
    cout << "stack vuoto" << endl;
    return -1;
  }
}

void stampa(StackPtr p) {
  for (int i=0; i<p->dim; i++) {
    if (i < p->N) { cout<< "[" << p->s[i] << "] "; }
    else { cout<< " [ ] "; }
  }
  cout <<endl;
}
```

Esempio con dato intero main.cpp

```
#include <cstdlib>
#include <iostream>
#include "Stack.h"
using namespace std;

#define N_ELEM 5
int main(int argc, char *argv[])
{
    StackPtr s1 = new Stack(N_ELEM);
    cout << "Lo stack e' vuoto? " << stackIsEmpty(s1) << endl;
    cout << "Lo stack e' pieno? "<< stackIsFull(s1) << endl;
    cout << "Contenuto dello stack:" << endl;
    stampa (s1);
    cout << "Push 1, ora lo stack contiene:" << endl;
    push(s1,1);
    stampa(s1);
    cout << "Push 2, 3, 4, 5, ora lo stack contiene:" << endl;
    delete s1;
}
```

Esempio con dato intero main.cpp

```
    push(s1,2); push(s1,3); push(s1,4); push(s1,5);
    stampa(s1);
    cout << "Push 6, ora lo stack contiene:" << endl;
    push(s1,6);
    stampa(s1);  //non viene inserito: stack pieno
    cout << "Pop(s1): " << pop(s1) << endl;
    cout << "Estratto elemento, ora lo stack contiene:" << endl;
    stmapa(s1);
    cout << "Pop(s1): " << pop(s1) << endl;
    cout << "Pop(s1): " << pop(s1) << endl;
    cout << "Pop(s1): " << pop(s1) << endl;
    cout << "Pop(s1): " << pop(s1) << endl;
    cout << "Contenuto dello stack:" << endl;
    stampa(s1);
    cout << "Pop(s1): ";
    pop(s1); //non viene prelevato alcun valore: stack vuoto
    stampa(s1);
    return EXIT_SUCCESS;
}
```