# Programmazione 2
## e Laboratorio di Programmazione

Corso di Laurea in

Informatica

Università degli Studi di Napoli "Parthenope"

Anno Accademico 2023-2024

Prof. Luigi Catuogno

1

## Informazioni sul corso

| | |
|---|---|
| **Docente** | Luigi Catuogno<br>**luigi.catuogno@uniparthenope.it** |
| **Orario** | Lun: 9:00-11:00<br>Mer: 11:00-13:00 |
| **Sede** | Centro Direzionale Napoli<br>**Aula Magna** |
| **Ricevimento** | Mer: 14:00-16:00 (previo appuntamento)<br>Ufficio docente oppure Team: **cxxa3bo** |

2

# Libri di testo

**Introduzione al linguaggio – costrutti e tecniche di base**

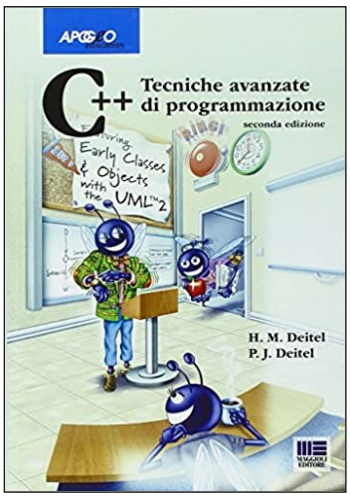
[FdP] H. M. Deitel, P. J. Deitel
C++ Fondamenti di programmazione
II ed. (2014) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8571-9


3

# Libri di testo

**Tecniche avanzate e strutture dati elementari**

[TAP] H. M. Deitel, P. J. Deitel
C++ Tecniche avanzate di programmazione
II ed. (2011) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8572-6




4

# Risorse on-line

***Team del corso***

**Programmazione 2 AA 2023-24 - Prof. Catuogno**
*Comunicazioni, incontri e avvisi per il corso*
Codice: **ftomzjx**


***Piattaforma e-learning***

**Programmazione II e Laboratorio di Programmazione II - A.A. 2023-24**
*Materiale didattico, manualistica, esercitazioni.*
URL: https://elearning.uniparthenope.it/course/view.php?id=2386

5

# Strutture dati elementari

6

# Lo stack (o *pila*)

- Uno stack (o *pila,* in italiano) è una struttura dati lineare che consente l'inserimento e l'estrazione dei dati:
  - Singolarmente
  - In maniera *sequenziale*
  - Applicando la politica *«Last-in, First-out»* (LIFO)

7

# Lo stack (o *pila*)

- Pur nella loro semplicità, gli stack sono una struttura dati importantissima!

  Chiamate a funzioni, ordinamento di eventi…
- Il funzionamento di uno stack è definito in termini generali, indipendentemente dalle sue possibile implementazioni
  - mediante strutture dati statiche (*i.e*. array)
  - oppure dinamiche (*e.g. liste a puntatori*)

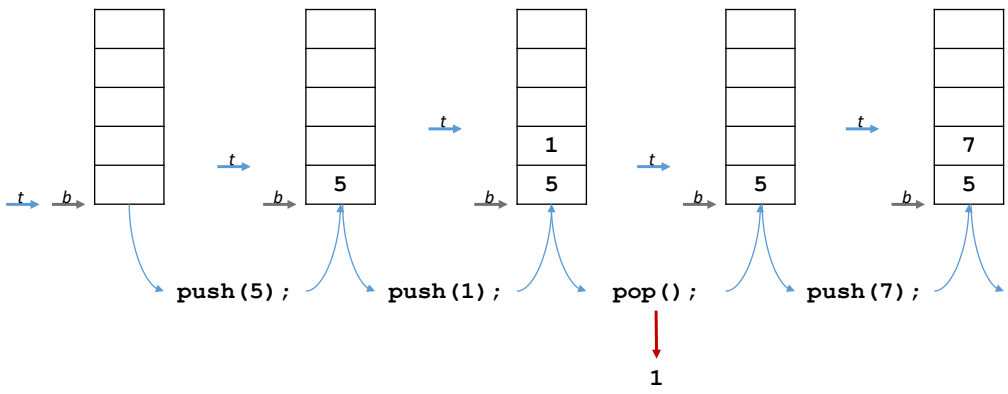
8

# Lo stack (o *pila*)

- E' costituito da:
  - Un'area di memoria in cui *«espandersi»* man mano che i dati vengono inseriti;
  - un puntatore/riferimento al *fondo* dello stack (*bottom*);
  - un puntatore/riferimento alla *cima* dello stack (*top*);
- I dati sono:
  - inseriti sempre in cima allo stack mediante l'operatore **`push()`**
  - estratti sempre dalla cima dello stack mediante l'operatore **`pop()`**

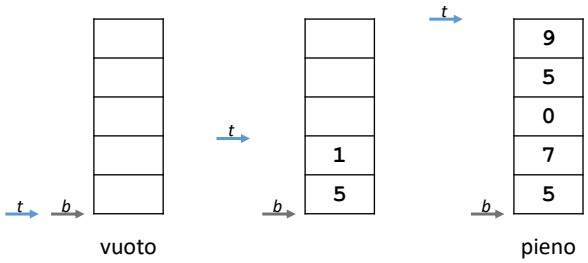
9

# Lo stack (o *pila*)




10

# *Stato* dello stack




11

# Esempio: stack di interi

- Scriviamo la classe **`StackArray`** che implementa uno stack di numeri interi, utilizzando un array:
  - Il costruttore prende un parametro int che indica la lunghezza dell'array da utilizzare (la *taglia* dello stack)
  - Gli operatori **`isFull()`** e **`isEmpty()`** che restituiscono il valore booleano **`true`** se lo stack è rispettivamente pieno o vuoto;
  - Gli opertori **`push()`** e **`pop()`** per l'inserimento e l'estrazione di dati nello/dallo stack.

12

# Esempio: stack di interi

File: **StackArray.hpp**

```
…  …
4  class StackArray {
5  protected:
6          int *data;
7          int len;
8          int top;
9  public:
10         StackArray(int);
11         ~StackArray();
12         bool isEmpty() const;
13         bool isFull() const;
14         StackArray *push(int);
15         int pop();
16 };
…  …
```

13

# Esempio: stack di interi

File: **StackArray.cpp**

```
…  …
4  StackArray::StackArray(int l) {
5          data=new int[l];
6          len=l;
7          top=-1;
8  }
9
10 StackArray::~StackArray(){
11         delete [] data;
12 }
…  …
```

14

# Esempio: stack di interi

File: **StackArray.cpp**

```
…  …
4  StackArray::StackArray(int l) {
5          data=new int[l];
6          len=l;
7          top=-1;
8  }
9
10 StackArray::~StackArray(){
11         delete [] data;
12 }
13 bool StackArray::isEmpty() const {
14         return top==-1;
15 }
16
17 bool StackArray::isFull() const {
18         return top==len-1;
19 }
…  …
```

15

# Esempio: stack di interi

File: **StackArray.cpp**

```
…  …
19 StackArray *StackArray::push(int item) {
20         if(isFull())
21                 cout << "Stack pieno!"<<endl;
22         else
23                 data[++top]=item;
24         return this;
25 }
26
27 int StackArray::pop() {
28         int item=0;
29         if(isEmpty())
30                 cout << "Stack vuoto!"<<endl;
31         else
32                 item=data[top--];
33         return item;
34 }
```

16

# Esempio: stack di interi

File: **mainStackArr.hpp**

```
…
4  int main()
5  {
6          StackArray x(10);
7          char scelta;
8          int item;
9
10         do {
11                 cout << "*** StackArray v1 ***"<<endl;
12                 cout << "(p)ush, p(o)p, is(e)mpty, is(f)ull, e(x)it" <<endl;
13                 cin >> scelta;
14                 switch(scelta) {
15                         case 'p':
16                                 cout<<" > item? ";
17                                 cin>>item;
18                                 x.push(item);
19                                 break;
…  …
```

17

# Esempio: stack di interi

File: **mainStackArr.hpp**

```
…
20                         case 'o':
21                                 item=x.pop();
22                                 cout<<" > item="<<item<<endl;
23                                 break;
24                         case 'f':
25                                 if(x.isFull())
26                                         cout << "Stack pieno"<<endl;
27                                 else
28                                         cout << "Stack non pieno"<<endl;
29                                 break;
30                         case 'e':
31                                 if(x.isEmpty())
32                                         cout << "Stack vuoto"<<endl;
33                                 else
34                                         cout << "Stack non vuoto"<<endl;
35                                 break;
…  …
```

18

# Esempio: stack di interi

File: **mainStackArr.hpp**

```
…
…
                        case 'x':
                                break;
                }
        } while (scelta!='x');
}
…
…
```

19

# Esercizio: stack di dati arbitrari

- Scriviamo la classe template **StackArrayT** perché implementi uno stack di dati di tipo arbitrario:

```
template<class T>
class StackArrayT {
        …
};
```

20

# Esercizio: classe StackArray2

- Scriviamo la classe template **`StackArray2`** che sia derivata dalla classe precedente e che aggiunga i seguenti metodi (si aggiungano eventuali attributi ritenuti necessari):

  **`int items()`** che restituisce il numero di item presenti nello stack;

  **`int flush()`** che svuota lo stack (senza distruggerlo) e restituisce il numero di item rimossi;

  **`int multipush(int[] items, num)`** che tenta di inserire nello stack i num elementi presenti in items e restituisca il numero di item effettivamente inseriti (*può essere minore di num se nel frattempo lo stack si riempe)*

21

# Esercizio: Tic-Tac-Toe

- Si consideri la versione 2 del Tic-Tac-Toe visto in precedenza. Si utilizzi uno stack per implementare la funzionalità **`undo()`** che permette di tornare indietro di un *round* (*e.g.* l'annullamento dell'ultima mossa effettuata dal giocatore richiedente e quella dell'altro giocatore).
- Ripetute invocazioni della **`undo()`** annullano a ritroso tutte le mosse effettuate, fino all'inizio della partita.

22