Corso “Programmazione 1”
Capitolo 12: Strutture Dati Astratte

Docente: Marco Roveri - marco.roveri@unitn.it
Esercitatori: Giovanni De Toni - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: Informatica (INF)
A.A.: 2021-2022
Luogo: DISI, Università di Trento
URL: https://bit.ly/2VgfYwJ

Ultimo aggiornamento: 22 novembre 2021

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2021-2022.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Liste Concatenate

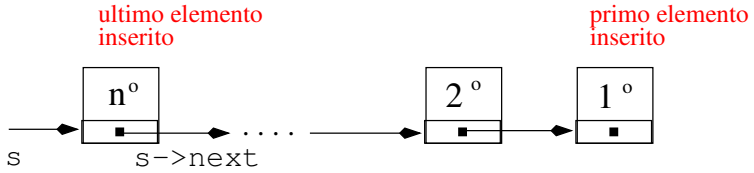
- (Nella sua versione più semplice) una lista concatenata `l` di oggetti di tipo `T` è definita come segue:
  - `l` è un puntatore **`NULL`** (lista vuota) oppure
  - `l` è un puntatore ad un nodo (**`struct`**) contenente:
    - un campo `value` di tipo `T`
    - un campo `next` di tipo lista concatenata

```
struct node;
typedef node * lista;
struct node { T value;  lista next; };
```

- Opzionalmente, possono esserci puntatori ad altri elementi

Una lista concatenata è una struttura dati dinamica, la cui struttura si evolve con l'immissione e estrazione di elementi.

Implementazione di una pila come lista concatenata

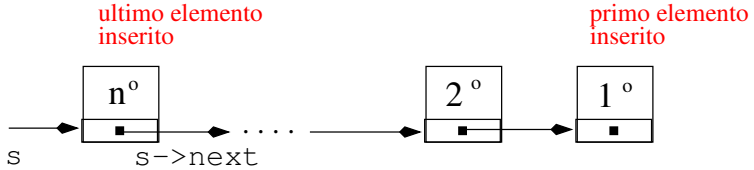


- Dati: una lista concatenata `s` di `n` elementi
  - `s` punta all'ultimo elemento inserito nella pila (inizialmente NULL)
  - [Opzionalmente un intero `n` con il numero di elementi nella lista]
  - l'ultimo elemento della lista contiene il primo elemento inserito.
  - pila vuota: `s==NULL`
  - pila piena: `out of memory`

  $\Longrightarrow$ numero di elementi contenuti nella pila limitato dalla memoria

N.B.: allocati solo gli `n` nodi necessari a contenere gli elementi

Implementazione di una pila come lista concatenata II



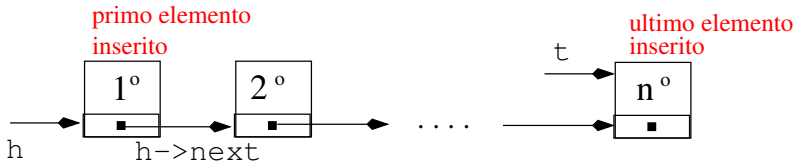
- Funzionalità:
  - `init()`: pone `s=`**`NULL`**
  - `push(T)`:
    1. alloca un nuovo nodo ad un puntatore `tmp`
    2. copia l'elemento in `tmp->value`
    3. assegna `tmp->next=s`, e `s=tmp`
  - `pop()`:
    1. fa puntare un nuovo puntatore `first` al primo nodo: `first=s`
    2. `s` aggira il primo nodo: `s=s->next`
    3. dealloca (l'ex) primo nodo: **`delete`** `first`
  - `top(T &)`: restituisce `s->value`
  - `deinit()`: ripete `pop()` finché la pila non è vuota

Esempi su pile di interi

- semplice stack di interi come struct:
  - `STACK_QUEUE_PUNT/struct_stack.h`
  - `STACK_QUEUE_PUNT/struct_stack.cc`
  - `STACK_QUEUE_PUNT/struct_stack_main.cc`
- uso di stack per invertire l'ordine:
  - `STACK_QUEUE_PUNT/struct_stack.h`
  - `STACK_QUEUE_PUNT/struct_stack.cc`
  - `STACK_QUEUE_PUNT/struct_reverse_main.cc`

N.B. I "main" e gli header delle funzioni identici a quelli in `STACK_QUEUE_ARRAY` $\Longrightarrow$ Tipo di Dato Astratto

Implementazione di una coda come lista concatenata

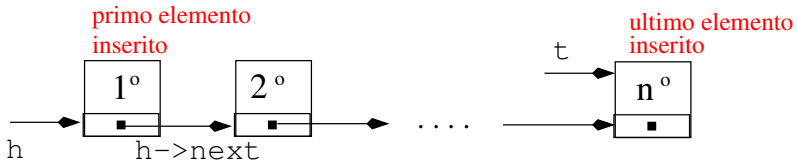


- Dati: una lista concatenata `h` di `n` elementi di tipo `T`, un puntatore `t` all'ultimo elemento
  - `h` punta al primo elemento inserito nella coda (inizialmente NULL)
  - `t` punta all'ultimo elemento inserito nella coda
  - [Opzionalmente un intero `n` con il numero di elementi nella lista]
  - coda vuota: `h==`**`NULL`**
  - coda piena: `out of memory`

  $\Longrightarrow$ numero di elementi contenuti nella coda limitato dalla memoria

N.B.: allocati solo gli `n` nodi necessari a contenere gli elementi

Implementazione di una coda come lista conc. II



- Funzionalità:
  - `init()`: pone `h`=**`NULL`**
  - `enqueue(T)`:
    1. alloca un nuovo nodo ad un puntatore `tmp`
    2. copia l'elemento in `tmp->value` e pone `tmp->next`=**`NULL`**
    3. (se coda non vuota) assegna `t->next=tmp`, e `t=tmp`
       (se coda vuota) assegna `h=tmp`, e `t=tmp`
  - `dequeue()`: come `pop()` della pila con il puntatore `h`
  - `first(T &)`: come `top()` della pila con il puntatore `h`
  - `deinit()`: ripete `dequeue()` finché la coda non è vuota

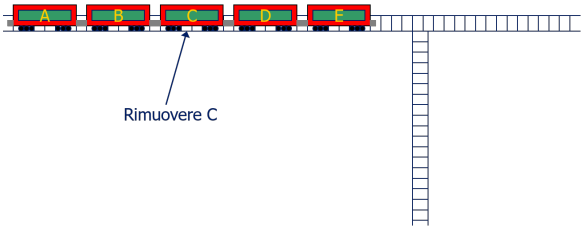
Esempi su code di interi

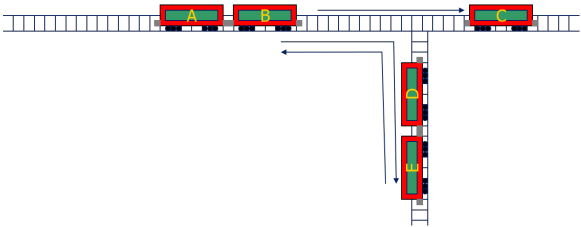
- semplice queue di interi come struct:
  - `STACK_QUEUE_PUNT/struct_queue.h`
  - `STACK_QUEUE_PUNT/struct_queue.cc`
  - `STACK_QUEUE_PUNT/struct_queue_main.cc`

N.B. l "`main`" e gli header delle funzioni identici a quelli in `STACK_QUEUE_ARRAY` $\Longrightarrow$ Tipo di Dato Astratto

Esempi uso stack

- Ho un treno composto da vagoni A B C D E, ho un binario di supporto e voglio rimuovere vagone C mantenendo ordine degli altri vagoni.





- Conversione di una espressione da infissa a postfissa
  - infissa : ((1 + 10) * (20 + 30)) + 40
  - postfissa: 1 10 + 20 30 + * 40 +

Esempio stack dir **`char`**

- TDA stack di **`char`** implementato con array:
  - `STACK_QUEUE_PUNT/cstack.h`
  - `STACK_QUEUE_PUNT/cstack_arr.cc`
  - `STACK_QUEUE_PUNT/cstack_main.cc`
- TDA stack di **`char`** implementato con liste concatenate:
  - `STACK_QUEUE_PUNT/cstack.h`
  - `STACK_QUEUE_PUNT/cstack_list.cc`
  - `STACK_QUEUE_PUNT/cstack_main.cc`

Esercizi proposti

Vedere file `ESERCIZI_PROPOSTI.txt`