PILE E CODE

Sono tipi di dato che consentono di rappresentare sequenze di elementi in cui gli inserimenti e le cancellazioni sono eseguiti con particolari modalita` (***politiche*** o ***discipline***).

Pile (stack):

Multi-insiemi gestiti con disciplina LIFO (***Last-In-First-Out***): una eliminazione ha per oggetto l'elemento che è stato inserito per ultimo.

Come tipo di dato astratto <S,Op,C>:

• S=(pila, elem, boolean)
dove pila è il dominio di interesse;

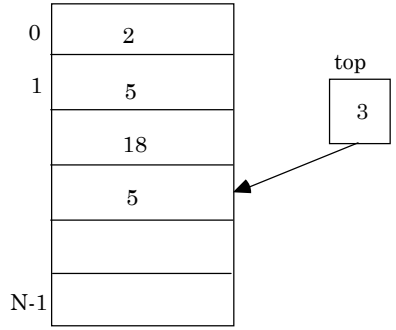
• Op = (top,push,pop,test-vuota)
top: pila $\rightarrow$ elem
push: elem $\times$ pila $\rightarrow$ pila
pop: pila $\rightarrow$ pila
test-vuota: pila $\rightarrow$ boolean

• C = {pila-vuota}

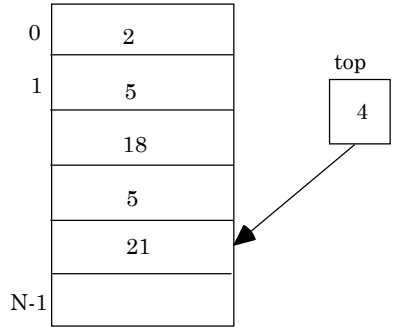
Per la realizzazione si può utilizzare un ***vettore*** o una ***lista***.

Rappresentazione sequenziale di pile (mediante vettore):

Vettore di N componenti. Variabile top, memorizza l'indice dell'elemento affiorante (top=-1, pila vuota)



Dopo il push di 21:



Rappresentazione collegata di pile (mediante lista):

È analoga al caso di rappresentazione collegata di una lista (+ disciplina LIFO).

Inserimenti e cancellazioni sempre dalla testa della lista. Il puntatore "radice" della lista punta all'elemento affiornate (***top***). Se NULL, pila vuota.

Date le corrispondenze:

push	---> cons
pop	---> tail
top	---> head
test-pila-vuota	---> empty

il tipo stack viene realizzato in modo immediato sulla base del modulo list (basta ridefinire i nomi delle operazioni).

Code:

Multi-insiemi gestiti con disciplina FIFO (***First-In-First-Out***): una eliminazione ha per oggetto l'elemento che è stato inserito per primo.

Come tipo di dato astratto <S,Op,C>:

- S=(coda, elem, boolean)

dove coda è il dominio di interesse;

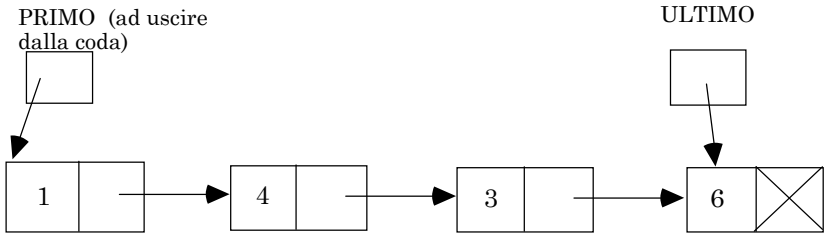
- Op = (primo, in, out, test-coda-vuota)
  primo: coda $\rightarrow$ elem
  in: elem $\times$ coda $\rightarrow$ coda
  out: coda $\rightarrow$ coda
  test-coda-vuota: coda $\rightarrow$ boolean

- C = {coda-vuota}

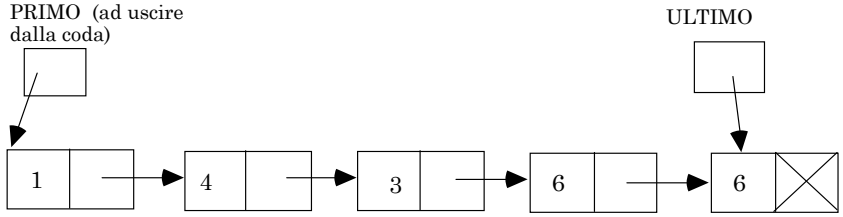
Le possibili rappresentazioni (sequenziale o collegata) sono analoghe a quelle delle pile, ma qui conviene avere accesso sia al primo elemento (per l'estrazione) che all'ultimo (per l'inserimento).

Rappresentazione collegata (liste):

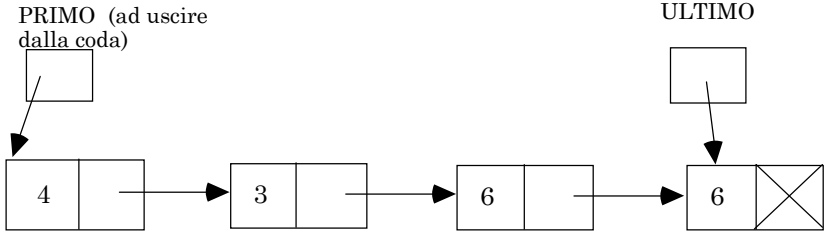
Coda C:



in(6,C):



out(C):



Si utilizzano due variabili di tipo ***puntatore*** (first, last).

Coda vuota, rappresentata da first==NULL, last==NULL.


```
/* QUEUE INTERFACE  - file queue.h*/
#include "el.h"

typedef struct   list_element
     { el_type   value;
       struct list_element  *next;
     } item;
typedef  item*  punt;
typedef  struct {punt primo;
        punt ultimo;} queue;
typedef  int boolean;

/*  PROTOTIPI DI FUNZIONE (extern) */
queue emptyqueue(void);
boolean test_coda_vuota(queue);
el_type top(queue);
queue out(queue);
queue in(el_type, queue);
void showqueue(queue);
```

```
/* QUEUE IMPLEMENTATION - file queue.c */
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

/* OPERAZIONI PRIMITIVE */
queue  emptyqueue(void)
{ queue C;
  C.primo=NULL;
  C.ultimo=NULL;
  return(C); };

boolean  test_coda_vuota(queue C)
{ return (   (C.primo==NULL)
          && (C.ultimo==NULL)); };

el_type  top(queue C)
{if (test_coda_vuota(C))
       { printf("coda vuota\n"); }
 else  { return (C.primo->value); }
}

queue  out(queue C)
{if (test_coda_vuota(C))
       { printf("coda vuota\n"); }
 else  { C.primo=C.primo->next;
         return (C); }
}
```



```
queue in(el_type e, queue C)
{item* p;
 p=(item *)malloc(sizeof(item));
 p->value=e;
 p->next=NULL;
 if (C.primo==NULL) {C.primo=p; C.ultimo=p;}
 else {C.ultimo->next=p; C.ultimo=C.ultimo-
>next;}
 return(C);
}



/* OPERAZIONI NON PRIMITIVE */
void   showqueue(queue C)
{item* l=C.primo;
printf("[");
 while (l!=NULL) { printf("%d",l->value);
          l=l->next;
          if (l!=NULL) printf(", "); }
 printf("]\n");
}
```

```
/* FILE PROGRAMMA */
#include "queue.h"
void main(void)
{queue coda;            /* dich. dato */
 coda=emptyqueue();
 showqueue(coda);
 coda=in(3,coda);
 showqueue(coda);
 coda=in(5,coda);
 showqueue(coda);
 coda=in(7,coda);
 showqueue(coda);
 coda=out(coda);
 showqueue(coda);
}
```