Strutture Dati

Lezione 6
Stack, coda

Oggi parleremo di …

- Tipo di dati astratto *Lista ordinata*
- Tipo di dati astratto *Stack*
  - specifica
  - rappresentazione
- Tipo di dati astratto *Coda*
  - specifica
  - rappresentazione
  - coda circolare

Il tipo di dati astratto *Lista ordinata*

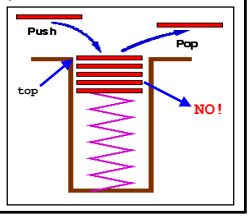
- Una **lista ordinata** di $n \geq 0$ elementi è definita come una sequenza $L=l_0,\ldots,l_{n-1}$.
- Il termine l_i rappresenta il generico *elemento* o *atomo* appartenente ad un certo *insieme*.
- Le operazioni possibili sono
  - trovare la lunghezza di una lista
  - leggere gli elementi di una lista da sin a des (o da des a sin)
  - estrarre l'*i*-esimo elemento da una lista, $0 \leq i \leq n-1$
  - sostituire l'elemento nella posizione *i*-esima, $0 \leq i \leq n-1$
  - inserire un nuovo elemento nella posizione *i*-esima, $0 \leq i \leq n-1$. Gli elementi precedentemente numerati *i*, *i*+1,…, *n*-1 diventano *i*+1, *i*+2, .., *n*
  - cancellare un elemento dalla *i*-esima posizione di una lista, $0 \leq i \leq n-1$. Gli elementi precedentemente numerati *i*+1,…, *n*-1 diventano *i*, *i*+1, …, *n*-2.

Il tipo di dati astratto *Lista ordinata*

- Una lista ordinata viene comunemente rappresentata con un array, dove si associa il generico elemento della lista l_i con l'indice *i* dell'array.
- Gli elementi successivi l_i e l_{i+1} vengono inseriti nelle posizioni successive *i* e *i*+1 dell'array.
- Ogni operazione di accesso avviene in tempo costante $O(1)$.
- L'inserimento e la cancellazione creano problemi computazionali
  - si ricorre alla rappresentazione non sequenziale, ovvero mediante liste concatenate.

Il tipo di dati astratto *Stack*

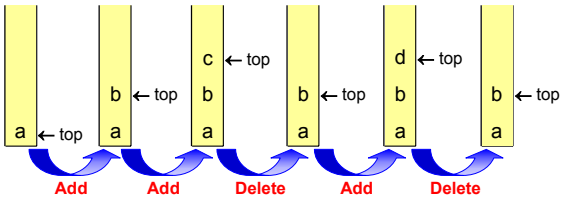
- Uno **stack** o **pila** è una lista ordinata in cui l'inserimento e la cancellazione avvengono in una posizione predeterminata: una estremità della lista detta ***top*** (o **testa**).
- Uno *stack* implementa una lista di tipo "last in, first out" (**LIFO**)
  - nuovi elementi vengono inseriti in testa e prelevati dalla testa.



Il tipo di dati astratto *Stack*

```
Struttura Stack
  oggetti: una lista ordinata di zero o più elementi

  funzioni: per ogni stack ∈ Stack, item ∈ Elemento,
      max_stack_size ∈ interi positivi

  Stack CreaS(max_stack_size)    ::=   crea uno stack vuoto la
                  cui dimensione massima è max_stack_size
  Booleano Pieno(stack, max_stack_size) ::=   if (numero di
                  elementi in stack è uguale a max_stack_size)
                  return TRUE, else return FALSE
  Booleano Vuoto(stack)   ::= if(stack==CreaS(max_stack_size))
                  return TRUE else return FALSE
  Stack Add(stack, item)   ::= if(Pieno(stack,max_stack_size))
                  stack_full, else inserisci item in cima allo
                  stack e return
  Elemento Delete(stack)   ::=  if(Vuoto(stack)) stack_empty,
                  else elimina l'elemento in cima allo stack e
                  restituiscilo
end Stack
```

Inserimento e cancellazione



Implementazione di uno *Stack*

- Mediante Arrays
  - permettono di implementare stack in modo semplice
  - flessibilità limitata, ma incontra parecchi casi di utilizzo
  - la capacità dello Stack è limitata.
- Mediante Liste concatenate.

```
#define MAX_SIZE 100
Stack CreaS(MAX_SIZE)  ::=
typedef struct {
        int key;
        // altri campi
        } elemento;

elemento stack[MAX_SIZE];
int top = -1;
```

```
Booleano Vuoto(Stack) ::=  top < 0;
Booleano Pieno(Stack) ::=  top >= MAX_SIZE-1;
```

Implementazione di *Add*

```
void add(int *top, elemento item)
{
 //aggiunge un elemento allo stack globale

 if (*top >= MAX_SIZE-1)  {
        stack_full();
        return;
        }
 stack[++*top] = item;
}
```

```
void stack_full()
{
 printf("Lo stack e' pieno\n");
}
```

Implementazione di *Delete*

```
elemento delete(int *top)
{
 if (*top == -1) return stack_empty();
 return stack[(*top)--];
}
```

```
elemento stack_empty(void)
{
  printf("Lo stack e` vuoto\n");
  return stack[top];
}
```

Le chiamate per l'inserimento e la cancellazione

```
add(&top, item);
item = delete(&top);
```

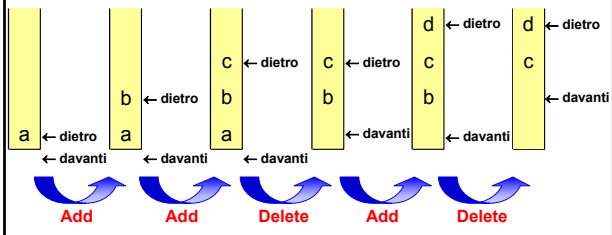
Il tipo di dati astratto *Coda*

- Una **coda** è una lista ordinata in cui l'inserimento e la cancellazione avvengono in una posizione predeterminata: gli inserimenti avvengono ad un estremo e le cancellazioni avvengono all'estremo opposto.
- Una *coda* implementa una lista di tipo "first in, first out" (**FIFO**)
  - può essere cancellato l'elemento che per più tempo è rimasto nell'insieme
  - una coda possiede una testa (davanti) e una coda (dietro)
  - quando si aggiunge un elemento, viene inserito al posto della coda
  - quando si rimuove un elemento, viene estratto quello in testa

Il tipo di dati astratto *Coda*

```
Struttura Coda
   oggetti: una lista ordinata di zero o più elementi

   funzioni: per ogni coda ∈ Coda, item ∈ Elemento,
       max_coda_size ∈ interi positivi

   Coda CreaC(max_coda_size)       ::=    crea una coda vuota la
                     cui dimensione massima è max_coda_size
   Booleano CPiena(coda, max_coda_size) ::=     if (numero di
                     elementi in coda è uguale a max_coda_size)
                     return TRUE, else return FALSE
   Booleano CVuota(coda)    ::= if(coda==CreaC(max_coda_size))
                     return TRUE else return FALSE
   Coda AddC(coda, item)    ::= if(CPiena(coda, max_coda_size))
                     coda_full, else inserisci  item in fondo
                     alla coda e return coda
   Elemento DeleteC(coda)    ::=   if(CVuota(coda)) coda_empty,
                     else elimina l'elemento in testa alla coda e
                     restituiscilo
end Coda
```

Inserimento e cancellazione



Implementazione di una *Coda*

- La più semplice utilizza un array monodimensionale e due indici, davanti e dietro
  - davanti punta alla testa della coda
  - dietro punta in fondo alla coda

```
#define MAX_SIZE 100

Coda  CreaC(MAX_SIZE)  ::=

typedef struct {
          int key;
   } elemento;

elemento coda[MAX_SIZE];

int dietro = -1;
int davanti = -1;
```

```
Booleano CVuota(Coda)   ::=  davanti == dietro;
Booleano CPiena(Coda)   ::=  dietro == MAX_SIZE-1;
```

Implementazione di *AddC*

```
void addc(int *dietro, elemento item)
{
 // aggiunge un elemento alla coda

 if (*dietro == MAX_SIZE-1)  {
        coda_full();
        return;
        }
 coda[++*dietro] = item;
}
```

```
void coda_full()
{
 printf("La coda e' piena\n");
}
```

Implementazione di *DeleteC*

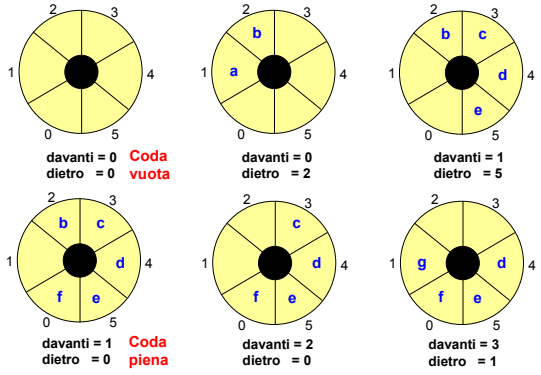
```
elemento delc(int *davanti, int dietro)
{
 // elimina il primo elemento della coda

 if (*davanti == dietro)  return coda_empty();
 return coda[++*davanti];
}
```

```
elemento coda_empty(void)
{
  printf("La coda e' vuota\n");
  return coda[davanti];
}
```

Le chiamate per l'inserimento e la cancellazione

```
addc(&dietro, item);
item = delc(&davanti, dietro);
```

Implementazione di una *Coda circolare*



Implementazione di una *Coda circolare*

```
#define MAX_SIZE 100

Coda  CreaC(MAX_SIZE)  ::=

typedef struct {
          int key;
   } elemento;

elemento coda[MAX_SIZE];

int dietro = 0;
int davanti = 0;
```

Per poter distinguere una coda piena da una coda vuota si adotta la convenzione che una coda circolare di dimensione **MAX_SIZE** può contenere al massimo **MAX_SIZE – 1** elementi.

La rotazione circolare avviene mediante una operazione di modulo

```
*dietro = (*dietro + 1) % MAX_SIZE;
*davanti = (*davanti + 1) % MAX_SIZE;
```

Implementazione di *AddC*

```
void addc(int davanti, int *dietro, elemento item)
{
 //aggiunge un elemento alla coda circolare

 *dietro = (*dietro + 1) % MAX_SIZE;
 if (*dietro == davanti)  {
        coda_full(dietro);   return; }
 coda[*dietro] = item;
}
```

```
void coda_full(int *dietro)
{
  /* ripristina dietro e
     segnala un errore */

*dietro = ?????
printf("La coda e' piena\n");
}
```

Implementazione di *DeleteC*

```
elemento delc(int *davanti, int dietro)
{
 // elimina il primo elemento della coda

 if (*davanti == dietro)  return coda_empty();
 *davanti = (*davanti + 1) % MAX_SIZE;
 return coda[*davanti];
}
```

```
elemento coda_empty(void)
{
  printf("La coda e' vuota\n");
  return coda[davanti];
}
```

Le chiamate per l'inserimento e la cancellazione

```
addc(davanti, &dietro, item);
item = delc(&davanti, dietro);
```