LABORATORIO DI PROGRAMMAZIONE 2
Corso di laurea in matematica

Strutture dati avanzate

Marco Lapegna
Dipartimento di Matematica e Applicazioni
Universita' degli Studi di Napoli Federico II

wpage.unina.it/lapegna

Caratteristiche degli array

Principali caratteristiche degli array:

PRO:
- Efficienza (accesso diretto tramite indice)

CONTRO
- Dimensione prefissata non modificabile
- Elementi omogenei (tutti dello stesso tipo)

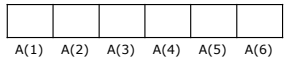
In molti problemi i dati presentano una organizzazione diversa da quella tabellare, oppure è più conveniente rappresentare le relazioni tra i dati con una struttura diversa oppure e' necessario aggiungere spazio per nuovi dati

OBIETTIVO

Definire nuove strutture dati

```
begin esempio
     var  a[100]: array of real
…
     for  i=1 to n
          print  a(i)
     endfor
…
 end esempio
```

Lettura di un array in pascal-like



Il tipo RECORD

in molte applicazioni i dati, pur essendo organizzati a tabella,
non sono tutti dello stesso tipo

ES: uno studente
- Nome (char)
- Cognome (char)
- Voto matematica (int)
- Voto informatica (int)
- Media (float)

studente

nome	Gennaro
cognome	Esposito
matematica	25
informatica	28
media	26,5

Dato strutturato di tipo RECORD

Il record

Il record e' una struttura dati di tipo statico piu' generale rispetto all'array

Con l'array condivide alcune caratteristiche:
- Dimensione fissata
- Accesso diretto ai vari campi

Un record puo' avere diversi aspetti, per cui deve essere visto come un nuovo tipo di dato
non disponibile in pascal-like o nei linguaggi di programmazione

Per utilizzare un record e' quindi necessario

- Dichiarare l'aspetto del nuovo tipo di record
- Dichiarare i record che avranno tale aspetto

Dichiarazione del nuovo tipo di record in P-like

Dichiarare un nuovo tipo di record significa:

- Dichiarare il nome del nuovo tipo di record
- Dichiarare nome e tipo dei campi

```
type <nomerecord> : record
        <campo_1>:<tipo>
        ……..
        <campo_k>:<tipo>
end
```

ESEMPIO

```
type studente: record
        nome[10]:array of char
        cognome[10]: array of char
        matematica: integer
        informatica: integer
        media: real
end
```


Dichiarazione dei record

L'istruzione **type** definisce solo la struttura del nuovo tipo di record, ma non dichiara nessun record di tale tipo !!

Con l'istruzione **var** si dichiarano le variabili del nuovo tipo di record

```
var <nome> : <nomerecord>
```

ESEMPIO

var stud1, stud2, stud3: studente

stud1, stud2 e stud3 sono 3 record di tipo studente !

stud1	stud2	stud3
Gennaro	Elisabetta	Giulio
Esposito	Russo	Federici
28	25	28
27	28	29
27,5	26,5	28,5

Campi del record

Le componenti di una variabile record sono denotabili in modo esplicito mediante i selettori di record

I campi di un record possono essere definiti e modificati come le tradizionali variabili

```
<nomerecord>.<campo_1> = valore1
<nomerecord>.<campo_2> = valore2
<nomerecord>.<campo_3> = valore3
…
```

ESEMPIO

```
stud1.nome = 'Gennaro'
stud1.cognome = 'Esposito'
stud1.matematica = 28
stud1.informatica = 27
stud1.media = 27.5
```

stud1	
Gennaro	nome
Esposito	cognome
28	matematica
27	informatica
27,5	media

campi del record

Spesso ci si ritrova a dover fare riferimento ad un record attraverso un puntatore

PT

stud1	
Gennaro	nome
Esposito	cognome
28	matematica
27	informatica
27,5	media

PT e' un puntatore al record stud1

in questo caso per fare riferimento ai campi del record si usa il simbolo ->

ESEMPIO

```
var stud1 : studente
var PT: puntatore a studente

PT-> nome = 'Gennaro'
PT-> cognome = 'Esposito'
PT-> matematica = 28
PT-> informatica = 27
PT-> media = 27.5
```

I record in C

I record trovano una naturale implementazione in C con il comando

struct

E' necessario specificare:
- Il nome del nuovo tipo di record
- Il tipo ed il nome dei campi

Cio' descrive solamente l'aspetto del nuovo tipo di record!!

Successivamente e' necessario dichiarare
- I nomi dei record del nuovo tipo

```
main ( ){

// specifica del nuovo tipo di variabile strutturata (studente)
struct studente{

// specifica dei campi della struttura
    char nome[10];
    char cognome[10];
    int matematica;
    int informatica;
    float media;
};

// dichiarazione di una variabile strutturata di tipo studente
 struct studente  stud1;
 ...
 ...
```

Specifica di una nuovo record di tipo "studente" e dichiarazione di una variabile (stud1) di tipo studente

I record in C

Una volta dichiarato un record e' possibile , accedere ai singoli campi con

nomevariabile.nomecampo

Esempio:
- stud1.nome
- stud1.cognome
- stud1.matematica
- stud1.informatica
- stud1.media

```
...
// lettura
scanf("%s", stud1.nome);
scanf("%s", stud1.cognome);
scanf("%d", &stud1.matematica);
scanf("%d", &stud1.informatica);
scanf("%f",  &stud1.media);

// stampa
printf("%s", stud1.nome);
printf("%s", stud1.cognome);
printf("%d", stud1.matematica);
printf("%d", stud1.informatica);
printf("%f",  stud1.media);
```

Frazione di codice C che legge e stampa i campi di una variabile strutturata

Strutture dati statiche e dinamiche

L'organizzazione a tabella (e i tipi di dato array e record) non esaurisce le possibili relazioni strutturali dei dati

Principale caratteristica di array e record: **staticita'**

In alcuni problemi non e' possibile prevedere la quantita' di dati da memorizzare

Necessita' di definire strutture dati **dinamiche**
in cui il numero di componenti e la loro organizzazione varia nel tempo;

LISTE **PILE** **CODE** **ALBERI**

Strutture lineari

Lista (list), Pila (stack), Coda (queue) sono strutture dati lineari

Gli elementi delle strutture sono comunemenenti chiamati nodi

In una struttura lineare ogni componente è "collegata" al più a 2 componenti: predecessore e successore

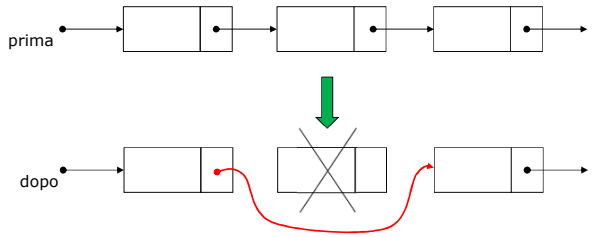
Ogni nodo ha almeno due campi:
- Campo informazione
- Campo collegamento (link) al nodo successivo

Strutture lineari

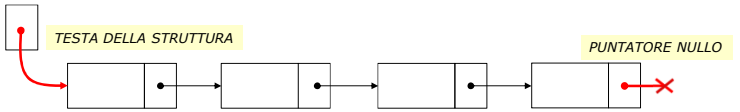
In una struttura dinamica lineare, l'ordine è determinato dal collegamento di un nodo al suo successore.

Le principali operazioni (inserimenti, cancellazione, ricerca) agiscono solo sui link

Esempio cancellazione:



Strutture lineari



Ad ogni struttura dinamica lineare si accede tramite un puntatore chiamato

TESTA della struttura (head)

Il campo informazione dell'ultimo nodo (CODA della struttura) e' un puntatore particolare chiamato NULLO (o VUOTO)

Le strutture dati lineari (lista, pila, coda) si differenziano per le modalita' di accesso

Strutture statiche vs dinamiche

Strutture statiche (array, record)

Vantaggi:
- Efficienza (locazioni di memoria contigue, per cui si accede alle componenti direttamente)

Svantaggi:
- Rigidita' (non e' possibile aggiungere nodi)

Strutture dinamiche (liste, pile, code)

Vantaggi:
- Flessibilita' (dimensione variabile e molte organizzazioni)

Svantaggi:
- Inefficienza (poiche' la dimensione puo' variare, i nodi non sono allocati in maniera contigua e l'accesso a un nodo avviene passando di nodo in nodo)

PILA (stack)

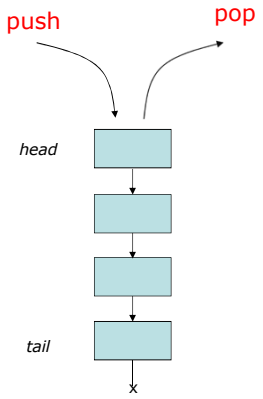
Lo ***stack*** è una struttura lineare (aperta) in cui è possibile accedere **solo** dalla testa della struttura

Organizzazione LIFO
(Last In First Out)

Uniche operazioni consentite:

- Inserimento (push)
- Estrazione (pop)

Sufficiente un puntatore alla testa



Stack (creazione)

La prima azione da fare su uno stack e'
- la dichiarazione della struttura del nodo
- La dichiarazione della testa dello stack

Un nodo ha la struttura di un record

ESEMPIO

```
type nodo: record
        info: integer
        pointer: puntatore a nodo
end
var head: puntatore a nodo
```

La creazione di uno stack consiste in
- Assegnazione di NULL a head

```
procedure creastack(out: head)
var  head : puntatore a nodo
head = NULL
end procedure
```

Marco Lapegna –
Laboratorio di Programmazione 2
Strutture dati avanzate

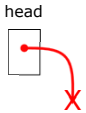
Stack (creazione)

La prima azione da fare su uno stack e'
- la dichiarazione della struttura del nodo
- La dichiarazione della testa dello stack

Un nodo ha la struttura di un record

ESEMPIO

```
type nodo: record
        info: integer
        pointer: puntatore a nodo
end
var head: puntatore a nodo
```



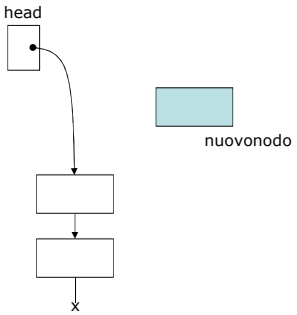
La creazione di uno stack consiste in
- Assegnazione di NULL a head

```
procedure creastack(out: head)
var  head : puntatore a nodo
head = NULL
end procedure
```

Marco Lapegna –
Laboratorio di Programmazione 2
Strutture dati avanzate

Stack (push)

Per inserire un nodo in uno stack
- Creare il nuovo nodo
- Riempire i campi informazione
- Collegare il nodo allo stack
  - Collegarlo al successivo
  - Collegarlo a head

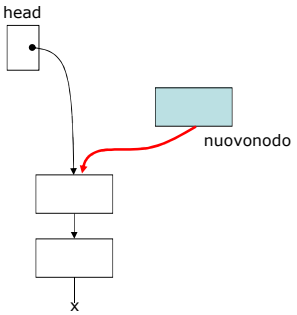



```
procedure push(in: head, dato; out: head)
var  head: puntatore a nodo
var  nuovonodo: nodo
var dato: integer
crea nuovonodo
nuovonodo.info = dato
nuovonodo.pointer = head
head = &nuovonodo
end procedure
```

Marco Lapegna –
Laboratorio di Programmazione 2
Strutture dati avanzate

Stack (push)

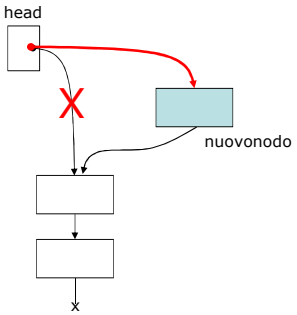
Per inserire un nodo in uno stack
- Creare il nuovo nodo
- Riempire i campi informazione
- Collegare il nodo allo stack
  - Collegarlo al successivo
  - Collegarlo a head




```
procedure push(in: head, dato; out: head)
var  head: puntatore a nodo
var  nuovonodo: nodo
var dato: integer
crea nuovonodo
nuovonodo.info = dato
nuovonodo.pointer = head
head = &nuovonodo
end procedure
```

Stack (push)

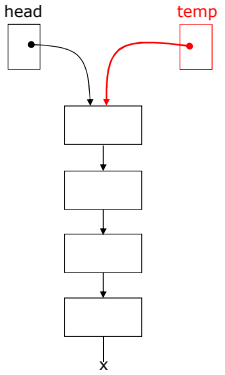
Per inserire un nodo in uno stack
- Creare il nuovo nodo
- Riempire i campi informazione
- Collegare il nodo allo stack
  - Collegarlo al successivo
  - Collegarlo a head




```
procedure push(in: head, dato; out: head)
var  head: puntatore a nodo
var  nuovonodo: nodo
var dato: integer
crea nuovonodo
nuovonodo.info = dato
nuovonodo.pointer = head
head = &nuovonodo
end procedure
```


Stack (pop)

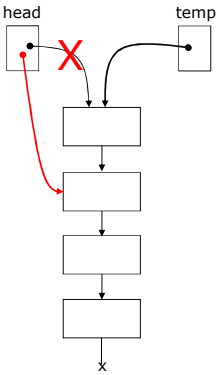
Per eliminare un nodo in uno stack
- Verificare se lo stack e' vuoto
- Salvataggio di head
- Estrazione del campo info
- Deallocazione dello spazio puntato da head
- Collegamento di head allo stack




```
procedure pop(in: head; out: head, dato, flag)
var  head, temp: puntatore a nodo
if (head == NULL) then
    flag = -1
else
    flag = 0
    temp = head
    dato = head -> info
    head = head -> pointer
    dealloca nodo puntato da temp
endif
end procedure
```


Stack (pop)

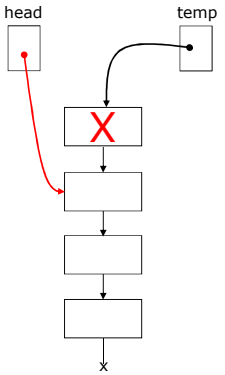
Per eliminare un nodo in uno stack
- Verificare se lo stack e' vuoto
- Salvataggio di head
- Estrazione del campo info
- Deallocazione dello spazio puntato da head
- Collegamento di head allo stack




```
procedure pop(in: head; out: head, dato, flag)
var  head, temp: puntatore a nodo
if (head == NULL) then
    flag = -1
else
    flag = 0
    temp = head
    dato = head.info
    head = head.pointer
    dealloca nodo puntato da temp
endif
end procedure
```


Stack (pop)

Per eliminare un nodo in uno stack
- Verificare se lo stack e' vuoto
- Salvataggio di head
- Estrazione del campo info
- Deallocazione dello spazio puntato da head
- Collegamento di head allo stack




```
procedure pop(in: head; out: head, dato, flag)
var  head, temp: puntatore a nodo
if (head == NULL) then
    flag = -1
else
    flag = 0
    temp = head
    dato = head.info
    head = head.pointer
    dealloca nodo puntato da temp
endif
end procedure
```

Note implementative in C

OSSERVAZIONE

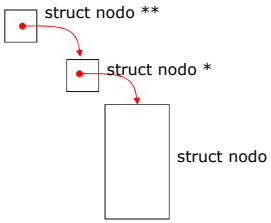
Le tre funzioni pop, push, create hanno come dato di output head (un puntatore a struttura)

Necessario passare l'indirizzo di head
cioe' un puntatore di puntatore a struttura

```
void creastack(struct nodo ** head){
  *head = NULL;
}
```

Esempio di funzione creastack

La funzione riceve un puntatore di puntatore a struttura



Note implementative in C

La procedura di inserimento prevede la creazione di un nuovo nodo

Come fare a creare un nuovo nodo?

La funzione malloc () restituisce un puntatore ad un'area di memoria di dimensione size

malloc(size)

La funzione malloc richiede come argomento la dimensione dello spazio occupato dal nuovo nodo

Quanto e' grande l'area di memoria occupata da un nodo?

La funzione sizeof () restituisce la dimensione dell'area di memoria occupata dal nodo

malloc(sizeof(struct nodo))

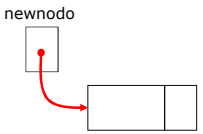
Note implementative in C

La funzione malloc restituisce un indirizzo ad un'area di memoria generica di dimensione size non strutturata in alcun modo

Come fare per organizzare l'area di memoria come un'area struturata come il nodo che vogliamo?

La conversione del puntatore avviene anteponendo a malloc il tipo dell'area di memoria (casting)

newnodo = (struct nodo *) malloc(sizeof(struct nodo))



N.B. newnodo e' una variabile puntatore a nodo

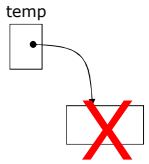
Note implementative in C

La procedura di rimozione di un nodo prevede la liberazione dello spazio in memoria

Come liberare lo spazio in memoria?

La funzione free() libera l'area di memoria indirizzata dall'argomento

free(temp)



CODA (queue)

Lo ***coda*** è una struttura lineare (aperta) in cui è possibile

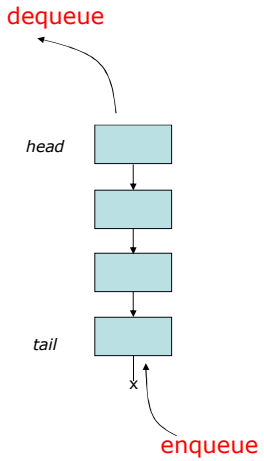
- Inserire nodi dalla coda
- Estrarre nodi dalla testa

Organizzazione FIFO
(First In First Out)

Uniche operazioni consentite:

- Inserimento (enqueue)
- Estrazione (dequeue)

Utili due puntatori alla testa e alla fine



Queue (creazione)

La prima azione da fare su una coda e'

- la dichiarazione della struttura del nodo
- La dichiarazione della testa e della fine della coda

Un nodo ha la struttura di un record

ESEMPIO

```
type nodo: record
        info: integer
        pointer: puntatore a nodo
end
var head, tail: puntatore a nodo
```

La creazione di una coda consiste in

- Assegnazione di NULL a head e tail

```
procedure creaqueue(out: head, tail)
var  head, tail : puntatore a nodo
head = NULL; tail = NULL
end procedure
```


Queue (creazione)

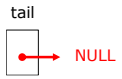
La prima azione da fare su una coda e'

- la dichiarazione della struttura del nodo
- La dichiarazione della testa e della fine della coda

Un nodo ha la struttura di un record

ESEMPIO

```
type nodo: record
        info: integer
        pointer: puntatore a nodo
end
var head, tail: puntatore a nodo
```



La creazione di una coda consiste in

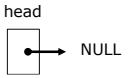
- Assegnazione di NULL a head e tail

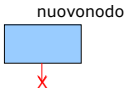
```
procedure creaqueue(out: head, tail)
var  head, tail : puntatore a nodo
head = NULL; tail = NULL
end procedure
```

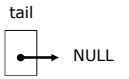

Queue (enqueue)

Per inserire un nodo in una coda

- Creare il nuovo nodo
- Riempire i campi informazione
- Collegare il nodo alla coda distinguendo i 2 casi
  - Coda vuota
  - Coda non vuota





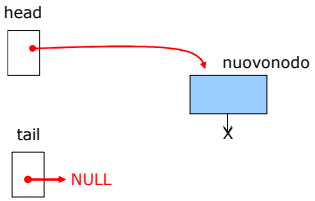



```
procedure enqueue(in: head, tail, dato; out: head, tail)
var  head, tail: puntatore a nodo
var  nuovonodo: nodo
crea nuovonodo
nuovonodo.info = dato
nuovonodo.point = NULL
if (head == tail == NULL) then
        head = &nuovonodo
else
        tail.pointer = nuovonodo
endif
tail = &nuovonodo
end procedure
```

Queue (enqueue)

Per inserire un nodo in una coda
- Creare il nuovo nodo
- Riempire i campi informazione
- Collegare il nodo alla coda distinguendo i 2 casi
  - Coda vuota
  - Coda non vuota

Caso coda vuota: nuovonodo e' collegato a head

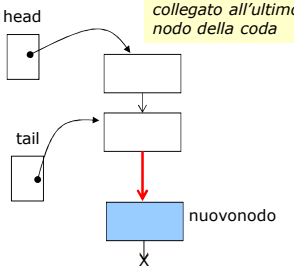



```
procedure enqueue(in: head, tail, dato; out: head, tail)
var  head, tail: puntatore a nodo
var  nuovonodo: nodo
crea nuovonodo
nuovonodo.info = dato
newnodo.pointer = NULL
if (head == tail == NULL) then
          head = &nuovonodo
else
          tail.pointer = nuovonodo
endif
tail = &nuovonodo
end procedure
```


Queue (enqueue)

Per inserire un nodo in una coda
- Creare il nuovo nodo
- Riempire i campi informazione
- Collegare il nodo allo coda distinguendo i 2 casi
  - Coda vuota
  - Coda non vuota

Caso coda non vuota: nuovonodo e' collegato all'ultimo nodo della coda

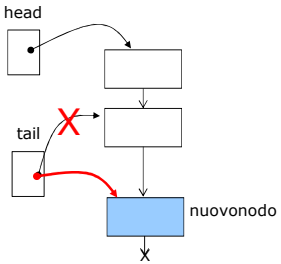



```
procedure enqueue(in: head, tail, dato; out: head, tail)
var  head, tail: puntatore a nodo
var  nuovonodo: nodo
crea nuovonodo
nuovonodo.info = dato
newnodo.pointer = NULL
if (head == tail == NULL) then
          head = &nuovonodo
else
          tail->pointer = &nuovonodo
endif
tail = &nuovonodo
end procedure
```


Queue (enqueue)

Per inserire un nodo in una coda
- Creare il nuovo nodo
- Riempire i campi informazione
- Collegare il nodo allo coda distinguendo i 2 casi
  - Coda vuota
  - Coda non vuota

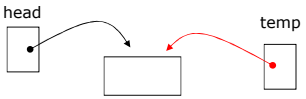
In entrambi i casi: nuovonodo e' collegato a tail




```
procedure enqueue(in: head, tail, dato; out: head, tail)
var  head, tail: puntatore a nodo
var  nuovonodo: nodo
crea nuovonodo
nuovonodo.info = dato
newnodo.pointer = NULL
if (head == tail == NULL) then
          head = &nuovonodo
else
          tail.pointer = &nuovonodo
endif
tail = &nuovonodo
end procedure
```


Queue (dequeue)

Per rimuovere un nodo in una coda
- Verificare se la coda e' vuota
- Salvataggio di head
- Estrazione del campo info
- Distinguere 2 casi
  - 1 nodo
  - 2 o piu' nodi

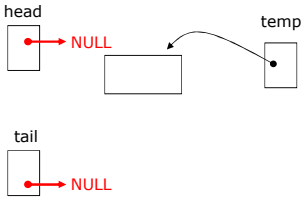



```
procedure dequeue(in: head, tail; out: head, tail, dato)
var  head, tail, temp: puntatore a nodo
if(head == NULL && tail == NULL) then
   flag = -1
else
   flag = 0
   temp = head
   dato = head->info
   if (head = tail) then
      head = NULL ; tail = NULL
   else
      head = head.pointer
   endif
    dealloca nodo puntato da temp
endif
```

Queue (dequeue)

Per rimuovere un nodo in una coda
- Verificare se la coda e' vuota
- Salvataggio di head
- Estrazione del campo info
- Distinguere 2 casi
  - 1 nodo
  - 2 o piu' nodi

1 solo elemento:

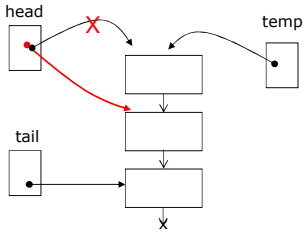



```
procedure dequeue(in: head, tail; out: head, tail, dato)
var  head, tail, temp: puntatore a nodo
if(head == NULL && tail == NULL) then
   flag = -1
else
   flag = 0
   temp = head
   dato = head.info
   if (head = tail) then
      head = NULL ; tail = NULL
   else
      head = head.pointer
   endif
   dealloca nodo puntato da temp
endif
```


Queue (dequeue)

Per rimuovere un nodo in una coda
- Verificare se la coda e' vuota
- Salvataggio di head
- Estrazione del campo info
- Distinguere 2 casi
  - 1 nodo
  - 2 o piu' nodi

Piu' elementi :

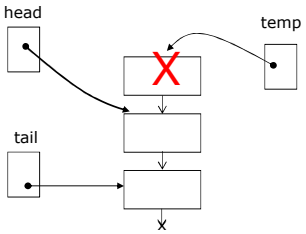



```
procedure dequeue(in: head, tail; out: head, tail, dato)
var  head, tail, temp: puntatore a nodo
if(head == NULL && tail == NULL) then
   flag = -1
else
   flag = 0
   temp = head
   dato = head.info
   if (head = tail) then
      head = NULL ; tail = NULL
   else
      head = head->pointer
   endif
   dealloca nodo puntato da temp
endif
```


Queue (dequeue)

Per rimuovere un nodo in una coda
- Verificare se la coda e' vuota
- Salvataggio di head
- Estrazione del campo info
- Distinguere 2 casi
  - 1 nodo
  - 2 o piu' nodi

In entrambi i casi




```
procedure dequeue(in: head, tail; out: head, tail, dato)
var  head, tail, temp: puntatore a nodo
if(head == NULL && tail == NULL) then
   flag = -1
else
   flag = 0
   temp = head
   dato = head.info
   if (head = tail) then
      head = NULL ; tail = NULL
   else
      head = head.pointer
   endif
   dealloca nodo puntato da temp
endif
```


Lista (list)

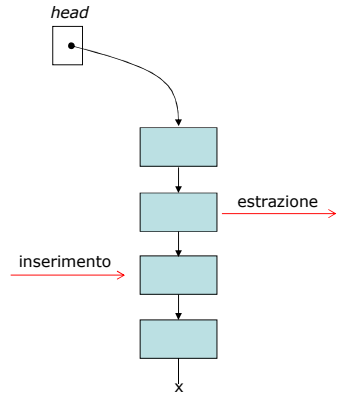
Lo ***lista*** è una struttura lineare (aperta) in cui e' definito un ORDINAMENTO

e' possibile Inserire ed Estrarre nodi da qualunque punto della struttura

operazioni consentite:

- Inserimento (secondo l'ordinamento)
- Estrazione
- Visita con stampa e/o conteggio nodi
- Distruzione lista
- Ricerca
- …

Sufficiente un solo puntatore alla testa della lista



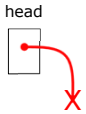
List (creazione)

La prima azione da fare su una lista e'
- la dichiarazione della struttura del nodo
- La dichiarazione della testa della lista

Un nodo ha la struttura di un record

ESEMPIO

```
type nodo: record
           info: integer
           pointer: puntatore a nodo
end
var head: puntatore a nodo
```



La creazione di una lista consiste in
- Assegnazione di NULL a head

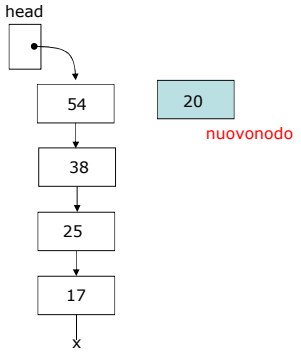
```
procedure crealista(out: head)
var  head : puntatore a nodo
head = NULL
end procedure
```

Identico alla creazione di uno stack

List (insert)

Per inserire un nodo in una lista ordinata
- Creare il nuovo nodo e riempire i campi informazione
- cercare il posto
- Collegare il nodo alla lista
  - Inserimeto in testa o inserimento in mezzo

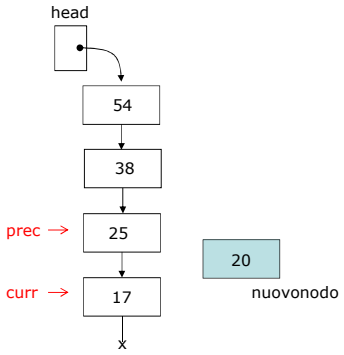
```
procedure insert(in: head, dato; out: head)
var  prec, curr, head: puntatore a nodo
var  nuovonodo: nodo
crea nuovonodo  ;   nuovonodo.info = dato
prec = NULL;   curr = head; flag = 0;
while ( curr != NULL  &&  flag== 0)
      if(curr->info  >  dato )
                prec = curr;  curr = curr->pointer;
      else
                flag = 1;
      endif
endwhile
if ( prec == NULL) then
   nuovonodo.pointer = head
   head = &nuovonodo
else
   prec->pointer = &nuovonodo
   nuovonodo.pointer = current
 endif
end procedure
```



List (insert)

Per inserire un nodo in una lista ordinata
- Creare il nuovo nodo e riempire i campi informazione
- cercare il posto
- Collegare il nodo alla lista
  - Inserimeto in testa o inserimento in mezzo

```
procedure insert(in: head, dato; out: head)
var  prec, curr, head: puntatore a nodo
var  nuovonodo: nodo
crea nuovonodo  ;   nuovonodo.info = dato
prec = NULL;   curr = head; flag = 0;
while ( curr != NULL  &&  flag== 0)
      if(curr->info  >  dato )
                prec = curr;  curr = curr->pointer;
      else
                flag = 1;
      endif
endwhile
if ( prec == NULL) then
   nuovonodo.pointer = head
   head = &nuovonodo
else
   prec->pointer = &nuovonodo
   nuovonodo.pointer = current
 endif
end procedure
```

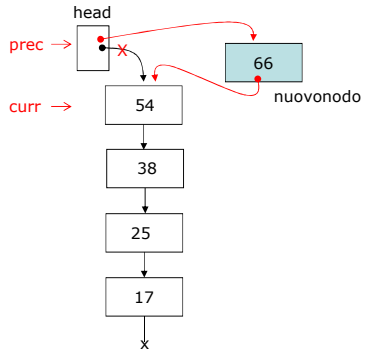


List (insert)

Per inserire un nodo in una lista ordinata
- Creare il nuovo nodo e riempire i campi informazione
- cercare il posto
- Collegare il nodo alla lista
  - Inserimeto in testa o inserimento in mezzo

Inserimento in testa.
N.B. prec = NULL

```
procedure insert(in: head, dato; out: head)
var  prec, curr, head: puntatore a nodo
var  nuovonodo: nodo
crea nuovonodo  ;   nuovonodo.info = dato
prec = NULL;   curr = head; flag = 0;
while ( curr != NULL  &&  flag== 0)
      if(curr->info  >  dato )
                prec = curr;  curr = curr->pointer;
      else
                flag = 1;
      endif
endwhile
if ( prec == NULL) then
   nuovonodo.pointer = head
   head = &nuovonodo
else
   prec->pointer = &nuovonodo
   nuovonodo.pointer = current
 endif
end procedure
```

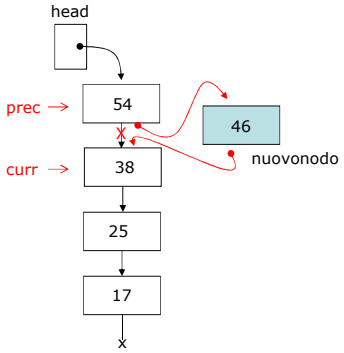


List (insert)

Per inserire un nodo in una lista ordinata
- Creare il nuovo nodo e riempire i campi informazione
- cercare il posto
- Collegare il nodo alla lista
  - Inserimeto in testa o inserimento in mezzo

Inserimento non in testa

```
procedure insert(in: head, dato; out: head)
var  prec, curr, head: puntatore a nodo
var  nuovonodo: nodo
crea nuovonodo  ;   nuovonodo.info = dato
prec = NULL;   curr = head; flag = 0;
while ( curr != NULL  &&  flag== 0)
      if(curr->info  >  dato )
                  prec = curr;  curr = curr->pointer;
      else
                  flag = 1;
      endif
endwhile
if ( prec == NULL) then
    nuovonodo.pointer = head
    head = &nuovonodo
else
    prec->pointer = &nuovonodo
    nuovonodo.pointer = current
 endif
end procedure
```

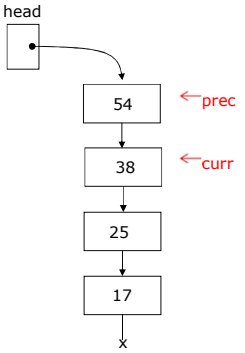


List (remove)

Per rimuovere un nodo in una lista ordinata
- Trovare il nodo
- Se il nodo esiste distinguere 2 casi
  - Primo nodo oppure altro nodo
- Estrarre il campo informazione

Esempio: ricerca 38

```
procedure remove(in: head, dato; out: head, flag)
var  prec, curr, head: puntatore a nodo
prec = NULL;   curr = head;  flag = 0;
while ( curr != NULL  &&  flag== 0)
            if(curr->info != dato )
                        prec = curr;  curr = curr->pointer;
            else
                        flag = 1;
            endif
endwhile
if (curr != NULL) then
   flag =1
   if (prec == NULL) then
       head = curr->pointer
   else
        prec->pointer = curr->pointer
   endif
   dealloca curr
endif
end procedure
```

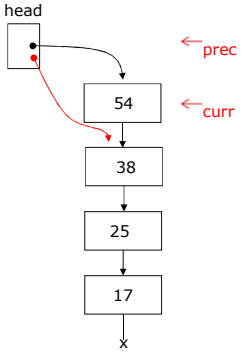


List (remove)

Per rimuovere un nodo in una lista ordinata
- Trovare il nodo
- Se il nodo esiste distinguere 2 casi
  - Primo nodo oppure altro nodo
- Estrarre il campo informazione

Esempio: Rimuovi 54 (primo nodo)

```
procedure remove(in: head, dato; out: head, flag)
var  prec, curr, head: puntatore a nodo
prec = NULL;   curr = head;  flag = 0;
while ( curr != NULL  &&  flag== 0)
            if(curr->info != dato )
                        prec = curr;  curr = curr->pointer;
            else
                        flag = 1;
            endif
endwhile
if (curr != NULL) then
   flag =1
   if (prec == NULL) then
       head = curr->pointer
   else
        prec->pointer = curr->pointer
   endif
   dealloca curr
endif
end procedure
```

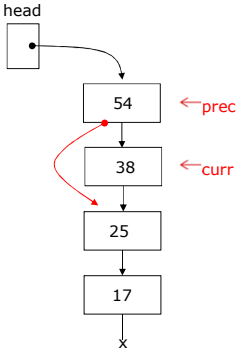


List (remove)

Per rimuovere un nodo in una lista ordinata
- Trovare il nodo
- Se il nodo esiste distinguere 2 casi
  - Primo nodo oppure altro nodo
- Estrarre il campo informazione

Esempio: Rimuovi 38 (nodo interno)

```
procedure remove(in: head, dato; out: head, flag)
var  prec, curr, head: puntatore a nodo
prec = NULL;   curr = head;  flag = 0;
while ( curr != NULL  &&  flag== 0)
            if(curr->info != dato )
                        prec = curr;  curr = curr->pointer;
            else
                        flag = 1;
            endif
endwhile
if (curr != NULL) then
   flag =1
   if (prec == NULL) then
       head = curr->pointer
   else
        prec->pointer = curr->pointer
   endif
   dealloca curr
endif
end procedure
```

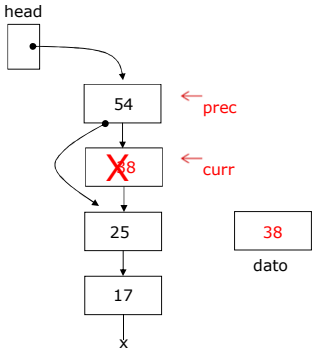


List (remove)

Per rimuovere un nodo in una lista ordinata

- Trovare il nodo
- Se il nodo esiste distinguere 2 casi
  - Primo nodo oppure altro nodo
- Estrarre il campo informazione

```
procedure remove(in: head, dato; out: head, flag)
var  prec, curr, head: puntatore a nodo
prec = NULL;   curr = head;  flag = 0;
while ( curr != NULL  &&  flag== 0)
            if(curr->info != dato )
                        prec = curr;  curr = curr->pointer;
            else
                        flag = 1;
            endif
endwhile
if (curr != NULL) then
   flag =1
   if (prec == NULL) then
        head = curr->pointer
   else
        prec->pointer = curr->pointer
   endif
   dealloca curr
endif
end procedure
```



List (altre funzioni)

In maniera analoga

```
procedure visitalista(in: head)
var curr, head: puntatore a nodo
curr = head
while (curr != NULL)
    printf  curr->info
    curr = curr->pointer
endwhile
end procedure
```

```
procedure contalista(in: head; out num)
var curr, head: puntatore a nodo
curr = head
num = 0
while (curr != NULL)
    num = num+1
    curr = curr->pointer
endwhile
end procedure
```

```
procedure distruggilista(in: head)
var curr, prec, head: puntatore a nodo
prec = head
while (prec != NULL)
     curr = prec->pointer
     dealloca  prec
     prec = curr
endwhile
end procedure
```