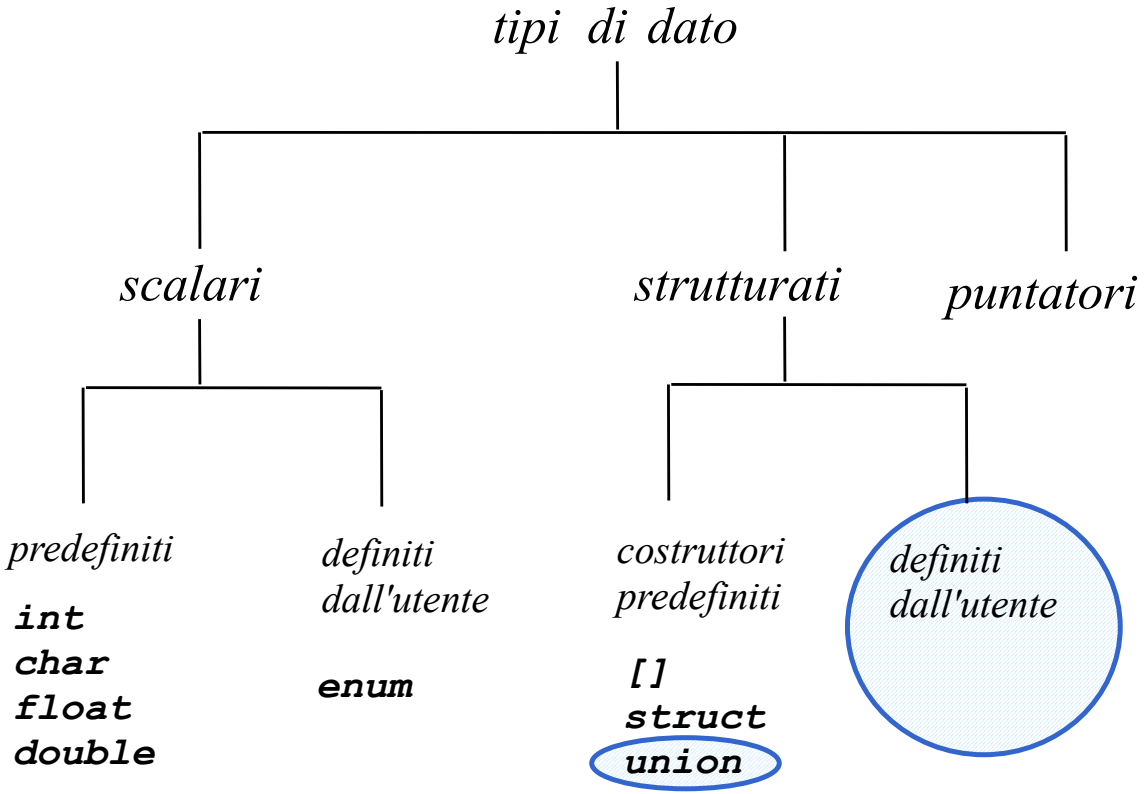
Strutture di dato - Tavole

- Obiettivi:
  - Introdurre le principali strutture dato elementari per la gestione di collezioni di oggetti
  - Strutture dato tavole (tabelle)

Tipi di dato

I tipi di dato si differenziano in *scalari* e *strutturati*



Come creare una nuova libreria di componenti?

Strutture dati

- Termine usato per indicare insiemi (collezioni) con elementi del dominio compositi, ad esempio:
  - tavole (dizionari);
  - liste;
  - insiemi;
  - pile e code;
  - alberi e grafi
- corredate delle operazioni per la loro gestione (efficiente)

Il tipo dato Dizionario (Tavola)

tipo `Dizionario`:
dati:
un insieme S di coppie $(elem, chiave)$.

operazioni:

`insert`$(elem\ e, chiave\ k)$
aggiunge a S una nuova coppia (e, k).

`delete`$(chiave\ k)$
cancella da S la coppia con chiave k.

`search`$(chiave\ k) \rightarrow elem$
se la chiave k è presente in S restituisce l'elemento e ad essa associato, e `null` altrimenti.

Struttura dati

- Organizzazione dei dati che permette di supportare le operazioni di un tipo di dato usando meno risorse di calcolo possibile

- Un nuovo tipo di dato (Abstract Data Type - ADT) e sua realizzazione fisica (ad esempio, dizionario con realizzazione indicizzata o con realizzazione collegata)

Tavole

- Un tipo tavola è un tipo di dato astratto per rappresentare insiemi di coppie:

  <chiave, attributi>

- Ciascuna coppia rappresenta dati riferiti ad un'unica entità logica (ad es., persona, documento, etc.) identificata in modo univoco dalla ***chiave***. Ad esempio, una istanza del tipo:

Nome	Cognome	Reddito	AliquotaMax
Mario	Rossi	18324	27
Luca	Bianchi	1453253	43

Il tipo dato Tavola

tipo `Tavola`
dati:
un insieme S di coppie $(elem, chiave)$.

operazioni:

`insert`$(elem\ e, chiave\ k)$
aggiunge a S una nuova coppia (e, k).

`delete`$(chiave\ k)$
cancella da S la coppia con chiave k.

`search`$(chiave\ k) \rightarrow elem$
se la chiave k è presente in S restituisce l'elemento e ad essa associato, e `null` altrimenti.

Tavole: ricerca

- Spesso la realizzazione è progettata in modo da ottimizzare la ricerca

- **Strutture ad accesso sequenziale, non ordinate** sulla chiave, ricerca esaustiva (o ricerca sequenziale): **O(N)**

- **Strutture ad accesso diretto, ordinate** sulla chiave, ricerca binaria (o ricerca dicotomica): **$O(\log_2(N))$**

- **Tavole hash**, costo unitario: **1**

Rappresentazione sequenziale

- ***In memoria centrale***, come un vettore di strutture (indicizzata, è anche ad accesso diretto …):

```
typedef struct {char Nome[20];
                char Cognome[20];
                int Reddito;
                int Aliquota;} Persone;

Persone         Tavola[100];    /* DATO */
```

- Più spesso sono memorizzate su dispositivi di memoria di massa (in **C *file binari*** con lettura e scrittura di strutture)

Rappresentazione sequenziale

- ***In memoria centrale***, come un vettore di strutture (indicizzata, è anche ad accesso diretto …):

```
typedef struct {char Nome[20];
                char Cognome[20];
                int Reddito;
                int Aliquota;} Persone;
typedef Persone TipoTavola[100];   /*tipo*/
TipoTavola      Tavola;            /* DATO */
```

- Più spesso sono memorizzate su dispositivi di memoria di massa (in **C *file binari*** con lettura e scrittura di strutture)

Esercitazione 6.1

- Scrivere un programma che realizzi una rubrica telefonica come tavola in memoria centrale. In particolare, ogni elemento della tavola è caratterizzato dalle seguenti informazioni:
  - Nome (chiave)
  - numero_telefono
- Il programma deve essere in grado di attuare varie richieste dell'utente:
- ***inserimento***: l'utente vuole inserire un nuovo record nell'archivio. Dati nome e numero di telefono della persona da inserire, il programma aggiunge un nuovo elemento all'archivio.

Esercitazione 6.1 (*cont.*)

- ***cancellazione***: l'utente vuole eliminare un elemento dall'archivio. Dato un nome, il programma dovrà eliminare (se esiste) l'elemento con il nome specificato.
- ***ricerca***: dato in ingresso il nome di una persona presente in archivio, si richiede la visualizzazione del numero di telefono relativo alla persona data;
- ***uscita***: l'utente richiede che il programma termini.

- L'interazione tra l'utente e il programma avviene in modo ciclico: l'utente può sottoporre una richiesta ad ogni ciclo ed il programma, sfruttando un meccanismo di selezione (per esempio **switch**) reagisce nel modo richiesto. L'esecuzione del programma si conclude quando l'utente richiede l'uscita.

Soluzione

Rappresentazione sequenziale, come un vettore di strutture (in memoria centrale). Il tipo di ciascun elemento di una rubrica è:

```
typedef struct {   char   nome[20];
                   char   tel[16];     } elemento;
```

Ogni rubrica è quindi del tipo:

```
#define N 100
typedef elemento rubrica[N];
```

Una variabile intera mantiene il numero di elementi inseriti (rappresenta la ***dimensione logica*** dell'array):

```
rubrica R;
int inseriti;
```

Dichiarazioni globali

```
#include <stdio.h>
#include <string.h>
typedef struct {  char  nome[20];
                  char  tel[16];    } elemento;

#define N 100
typedef elemento rubrica[N];

/* prototipi */
int menu(void);
int inserimento(rubrica R, int DIM);
int cancellazione(rubrica R, int DIM);
void ricerca(rubrica R, int DIM);
int individua(rubrica R, int DIM, elemento e);
```

main

```
void main(void)
{int scelta, inseriti, fine;
 rubrica R;                    /* R è il dato tavola */


 inseriti=0;                   /*dim. logica, tavola vuota*/
 fine=0;
 do
 { scelta=menu();
   switch(scelta){
     case 1: inseriti = inserimento(R,inseriti); break;
     case 2: inseriti = cancellazione(R,inseriti); break;
     case 3: ricerca(R, inseriti);  break;
     case 4: fine=1; break;
     default: printf("Scelta sbagliata\n");
             }
      }while (!fine);
}
```

Dopo alcuni inserimenti:

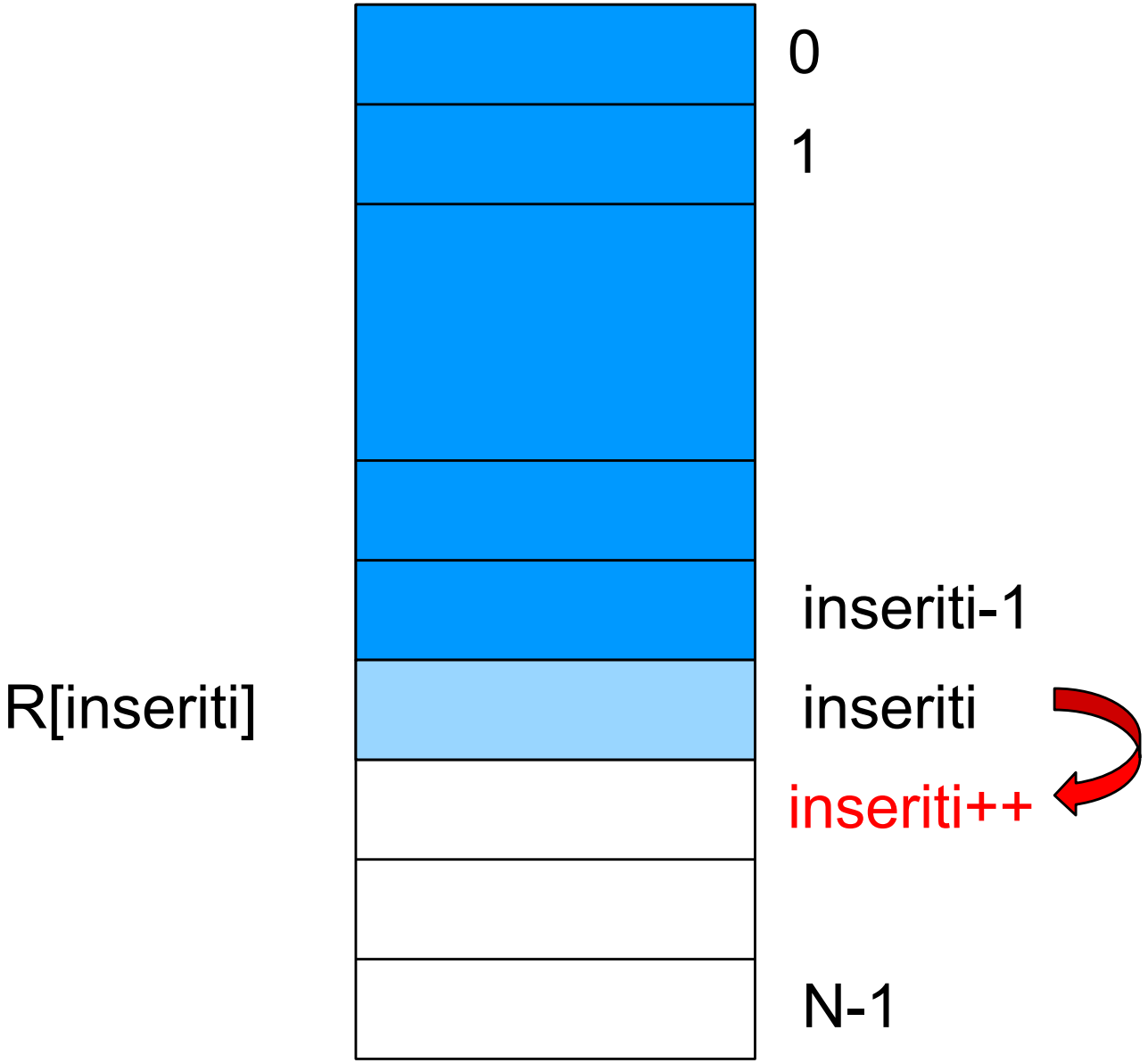
- **R**

menu

```
int menu(void)
{     int ris;
      printf("Scegli l'operazione:\n");
      printf("\t1\tInserimento\n");
      printf("\t2\tCancellazione\n");
      printf("\t3\tRicerca\n");
      printf("\t4\tUscita\n");
      printf("\n\nScelta:  ");
      scanf("%d", &ris);
      return ris;
}
```

Inserimento:

- **R**



inserimento

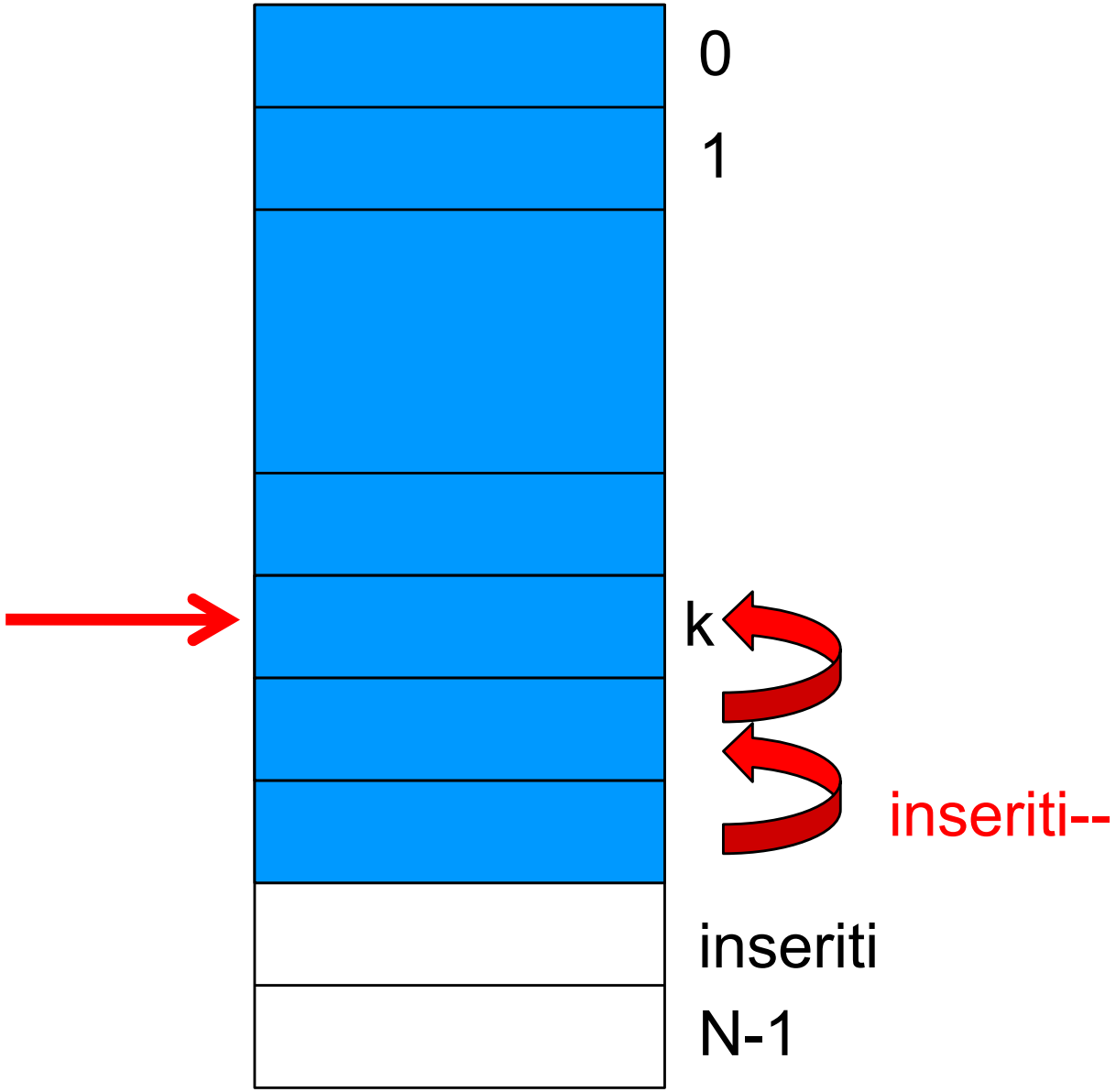
L'inserimento determina la memorizzazione dell'elemento dato da standard input nella prima posizione libera del vettore:

```
int inserimento (rubrica R, int DIM)
{       if (DIM<=N-1)
        {       printf("\nInserire nome:   ");
                scanf("%s",R[DIM].nome);
                printf("\nInserire numero:   ");
                scanf("%s",R[DIM].tel);
                DIM++;
        }
        else printf("Vettore pieno!\n");
        return DIM;
}
```

Restituisce il numero di elementi effettivamente inseriti nel vettore: se rubrica non piena, il risultato è **`DIM+1`**; altrimenti, l'inserimento non può avere luogo e viene restituito **`DIM`** come risultato della chiamata.

Cancellazione:

- **R**



cancellazione

Eliminazione dalla rubrica (data come parametro) dell'elemento identificato attraverso il suo nome (letto da input):

```
int cancellazione(rubrica R, int DIM)
{ int k, j;
  elemento e;
  printf("\nInserire nome:  ");
  scanf("%s",e.nome);
  k=individua(R, DIM, e);
  if (k<N)
   {  printf("\nCancellazione di %s ...\n", R[k].nome);
      for (j=k; j<DIM-1; j++) R[j]=R[j+1];
      DIM--;
      }
  else printf("\n%s\t non trovato\n", e.nome);
      return DIM;
}
```

individua

Realizza la ricerca sequenziale, caso peggiore O(**DIM**)

```
int individua(rubrica R, int DIM, elemento e)
{      int i,trovato=0;
       for (i=0; i<DIM && !trovato; i++)
              if (!strcmp(e.nome, R[i].nome))
                            trovato=1;
       if (trovato) return i-1;
       else return N;
}
```

In caso di chiave composta …

Se la chiave fosse costituita da più attributi (ad esempio **nome** e **cognome**):

```
int individua(rubrica R, int DIM, elemento e)
{ int i,trovato=0;
  for (i=0; i<DIM && !trovato; i++)
     if ( !strcmp(e.nome, R[i].nome) &&
          !strcmp(e.cognome, R[i].cognome)  )
               trovato=1;
  if (trovato) return i-1;
  else return N;
}
```

ricerca

Ricerca nella rubrica (data come parametro) del telefono di un elemento attraverso il suo nome (letto da input):

```
void ricerca(rubrica R, int DIM)
{ int k;
  elemento e;
  printf("\nInserire nome:  ");
  scanf("%s",e.nome);
  k=individua(R, DIM, e);
  if (k<N)
   {  printf("\nTelefono di %s ...\n", R[k].nome);
      printf("\è  %s ...\n", R[k].tel);
            }
  else printf("\n%s\t non trovato\n", e.nome);
}
```

Esercitazione 3.1 (in Laboratorio)

- Dopo gli inserimenti, ordinare la tavola tramite la funzione **qsort** (definendo un'opportuna funzione di confronto **fcmp**) e effettuare la ricerca tramite la ricerca binaria

```
int fcmpt(elemento *e1, elemento *e2)
   { return strcmp((*e1).nome, (*e2).nome));
    }

/* chiamata qsort : */
 qsort(R, inseriti, sizeof(elemento), fcmpt);

/* definire individua come ricerca binaria
 Esercitazione 3.2                    */
```

Tutto chiaro fin qui? ...

Tavole: ricerca (*recap*)

- Spesso la realizzazione è progettata in modo da ottimizzare la ricerca

- **Strutture ad accesso sequenziale, non ordinate** sulla chiave, ricerca esaustiva (o ricerca sequenziale): **O(N)**

- **Strutture ad accesso diretto, ordinate** sulla chiave, ricerca binaria (o ricerca dicotomica): **$O(\log_2(N))$**

- **Tavole hash**, costo unitario: **1**

Variante

- Modificare il codice dell'esercitazione 6.1 realizzando la rubrica telefonica come **tavola in memoria centrale ordinata sulla chiave (vettore ordinato in base al nome), effettuando inserimenti ordinati**.

- **Suggerimento:** ad ogni inserimento occorre determinare la posizione in cui inserire l'elemento di data chiave e creare spazio per esso nel vettore (*shift* verso il basso)

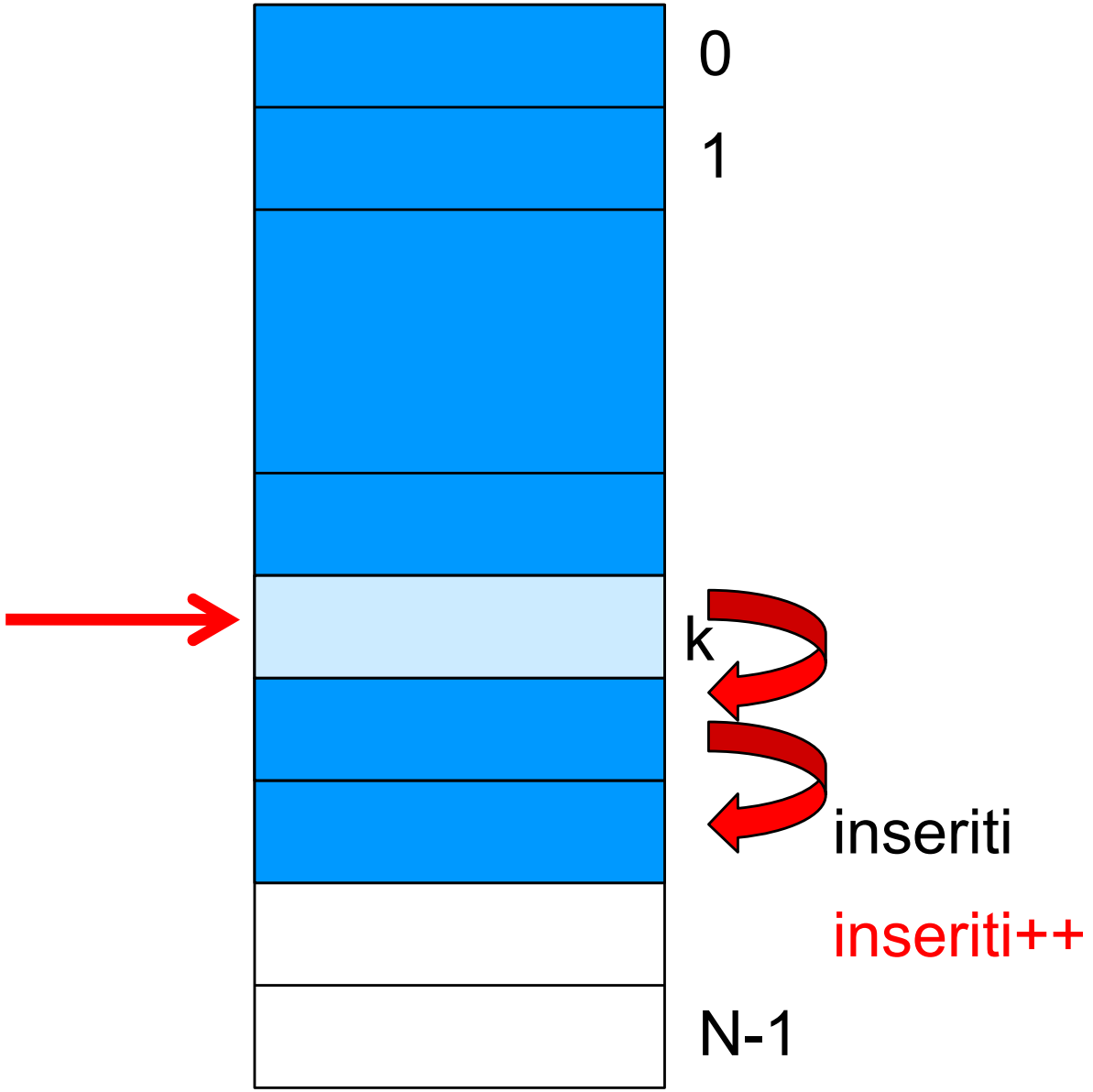
Ricerca binaria

```
int individua_binaria(rubrica R, int DIM, elemento e)
{ int i=0, medio, trovato=0,;
  while (i<DIM && !trovato)
      { medio=(i+DIM)/2;
        if (!strcmp(e.nome, R[medio].nome))
                  trovato=1;
        else
          if (strcmp(e.nome,R[medio].nome)>0)
              i=medio+1;
          else   DIM=medio-1;
    }
      if (trovato) return medio;
      else return N;
}
```

Caso peggiore O($\log_2$ **DIM**)

Inserimento ordinato:

- **R**



inserimento ordinato (*aggiunto*)

L'inserimento determina con la individua_bin2 (da definire) la locazione del vettore che ha il primo elemento *maggiore* di quello da inserire:

```
int inserimento_ord (rubrica R, int DIM)
{ int j, k;
  elemento e;
      if (DIM<=N-1)
      {     printf("\nInserire nome:  ");
            scanf("%s",e.nome);
            printf("\nInserire numero:  ");
            scanf("%s",e.tel);
            k=individua_bin2(R, DIM, e);
            if (k<DIM)
                for (j=DIM-1; j<=k; j--) R[j+1]=R[j];
            R[k]=e; DIM++;
      }
      else printf("Vettore pieno!\n");
      return DIM;
}
```

Rappresentazione sequenziale

- ***In memoria di massa***, come file binario
- Lettura e scrittura di blocchi (strutture)
- E' comunque consentito l'accesso diretto con **`fseek`**) … potremmo quindi (se ordinato) usare la ricerca binaria

Esercitazione 6.4

- Realizzare una rubrica telefonica come tavola in memoria di massa (file binario)

- Prevedere le operazioni di:
  - creazione della rubrica
  - stampa della rubrica
  - ricerca (sequenziale) nella rubrica

Esercitazione 6.4 (*cont.*)

```
#include <stdio.h>
#include <string.h>
typedef struct  { char nome[20];
                  char tel[16];      } elemento;
void creafile(char *v);
void vedifile(char *v);
void ricerca(char *v);

void main(void)
{     creafile("rubrica");
      vedifile("rubrica");
      ricerca("rubrica");
}
```

Esercitazione 6.4 (*cont.*)

```
#include <stdio.h>
#include <string.h>
typedef struct  { char nome[20];
                  char tel[16];      } elemento;
void creafile(char *v);
void vedifile(char *v);
void ricerca(char *v);

void main(int argc, char *argv[])
{      creafile(argv[1]);
       vedifile(argv[1]);
       ricerca(argv[1]);
}
```

Creazione file

```
void creafile(char *v)
{  FILE *f; elemento e; int fine=0;
   elemento leggiel();
   f=fopen(v, "wb");
   printf("Creazione di %s...\n", v);
   while (!fine)
   {   e=leggiel();
       fwrite(&e, sizeof(elemento), 1, f);
       printf("\nFine (SI=1, NO=0) ? ");
       scanf("%d", &fine);
       fflush(stdin);
   }
   fclose(f);
}
```

Stampa file

```
void vedifile(char *v)
{ FILE *f; elemento e;
  void stampael(elemento e);
  f=fopen(v, "rb");
  printf("Lettura di %s:\n", v);
  while( fread(&e, sizeof(elemento), 1, f)>0 )
     {    stampael(e);   }
  fclose(f);
}
```

Lettura e stampa elemento

```
elemento leggiel()
{  elemento e;
   printf("Nome ? ");
   scanf("%s",e.nome);
   printf("\nTelefono ? ");
   scanf("%s", e.tel);
   return e;
}

void stampael(elemento e)
{  printf("%s\t", e.nome);
   printf("%s\n", e.tel);
}
```

Ricerca (sequenziale)

```
void ricerca(char *v)
{  FILE *f; elemento e;
   char Nome[20];
   int k=0;

   printf("Nome da cercare: ");
   scanf("%s",Nome);
   f=fopen(v, "rb");
   while( fread(&e,sizeof(elemento),1, f)>0 )
       if (!strcmp(e.nome, Nome) )
           { stampael(e); k++; }
   printf("\n\nTrovati %d omonimi %s \n", k, Nome);
   fclose(f);
}
```

Ricerca (sequenziale)

```
int ricerca(char *v)
{  FILE *f; elemento e;
   char Nome[20];
   int trovato=0;

   printf("Nome da cercare: ");
   scanf("%s",Nome);
   f=fopen(v, "rb");
   while( fread(&e,sizeof(elemento),1, f)>0 && !trovato )
          if (!strcmp(e.nome, Nome) ) {  trovato=1;
                                         stampael(e); }
   fclose(f);
   return trovato;
}
```

Esercitazione 3.4 (laboratorio)

- Modificare l'esercizio 6.4, dove si è realizzata una rubrica telefonica come file binario, realizzando un menù che preveda la chiamata delle operazioni di:
  - creazione della rubrica (file vuoto)
  - inserimento di un elemento letto a terminale in rubrica
  - ricerca di un elemento, letta la chiave a terminale
  - stampa dell'intera rubrica
  - fine programma

- Si modifichi poi l'inserimento, in modo tale da mantenere il file ordinato sul campo chiave **`nome`** ed effettuare la ricerca binaria (realizzarla)

fseek, ftell

- Per posizionarsi all'interno di un file:

```
int  fseek(FILE* f, long offset, int origin);
```

- Per ottenere la posizione corrente:

```
long ftell(FILE* f);
```

dove:

`offset`	posizione, rispetto a `origin`, a cui portarsi
`origin`	posizione rispetto a cui misurare l'`offset`, costante simbolica, che può essere:
	inizio del file — `SEEK_SET`
	posizione corrente — `SEEK_CUR`
	fine del file — `SEEK_END`

- *Per un file di testo*, `offset` deve valere 0 o il valore restituito da `ftell()` (nel qual caso, `origin` deve obbligatoriamente valere `SEEK_SET`)

☞Tavole: rappresentazione sequenziale

- In memoria centrale, realizzabile con *vettori*
  - Si impone un limite massimo alla dimensione della tavola (al massimo N elementi)
  - Lo spazio di memoria occupato è fisso (indipendente dal numero di elementi della tavola)

- In memoria secondaria, attraverso *file sequenziali*
- Ricerca sequenziale (***caso peggiore, O(N)***)

- Visibilità completa della struttura dati (sia come dato sia come tipo)

main

```
#include <stdio.h>
#include <string.h>
typedef struct {  char  nome[20];
                  char  tel[16];    } elemento;

#define N 100
typedef elemento rubrica[N];
. . .

void main(void)
{int scelta, inseriti, fine;
 rubrica R; strcpy(R[0].nome,"Evelina");

 inseriti=0;

 . . .
}
```

☞Tavole: rappresentazione sequenziale

- In memoria centrale, realizzabile con *vettori*
  - Si impone un limite massimo alla dimensione della tavola (al massimo N elementi)
  - Lo spazio di memoria occupato è fisso (indipendente dal numero di elementi della tavola)

Liste

- In memoria secondaria, attraverso *file sequenziali*
- Ricerca sequenziale (***caso peggiore, O(N)***)

Componenti software (astrazioni di dato e ADT)

- Visibilità completa della struttura dati (sia come dato sia come tipo)

☞Tavole: rappresentazione indicizzata

- In memoria centrale, realizzabile solo attraverso *vettori*
  - Contenuto ordinato, accesso diretto alla i-esima componente
- In memoria secondaria, attraverso *file binari*
  - Inserimento ordinato, accesso diretto alla i-esima componente
- Se dati ordinati sulla chiave, ricerca binaria (***caso peggiore, $O(\log_2 N)$***)

☞Tavole: rappresentazione indicizzata

- E' possibile migliorare ulteriormente la complessità della ricerca?
- Tavola ordinata, applicabile la ricerca binaria (***caso peggiore, $O(\log_2 N)$***)
- Diversa organizzazione, con una sovrastruttura:
  - tavola hash, data la chiave una funzione "calcola" qual è il *bucket* in cui scrivere/leggere l'elemento con quella chiave
  - oppure struttura secondaria di accesso tramite indice (alberi di ricerca)
- Si abbassa il costo della ricerca, ma gestione più complessa

Tutto chiaro fin qui? ...