Strutture dati e realizzazione in C

Strutture dati:

Termine usato per indicare tipi con elementi del dominio compositi, ad esempio:

- vettori e matrici;
- tavole;
- liste;
- insiemi;
- pile e code;
- alberi e grafi.

Obiettivo:

Introdurre le principali strutture di dato e la loro realizzazione come componenti ***software (astrazione di dato*** e ***tipo di dato astratto***)

Tipo di dato astratto:

<S, Op, C>

- Un insieme di valori detto dominio del tipo (**S**);
- Un insieme di operazioni che si applicano a valori del dominio o che hanno come risultato valori del dominio (**Op**);
- Un insieme di costanti che denotano valori del dominio (**C**).

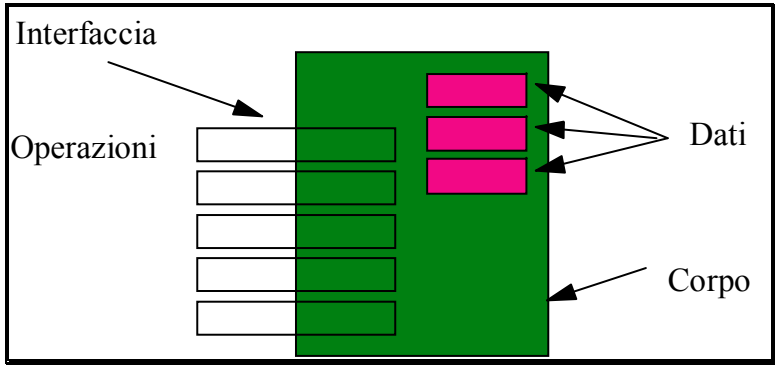
Componenti software:

- ***Librerie:***

  Il componente rende visibili procedure e funzioni che non fanno uso di variabili non locali. Il componente è una collezione di ***operazioni*** (ad esempio, funzioni matematiche).

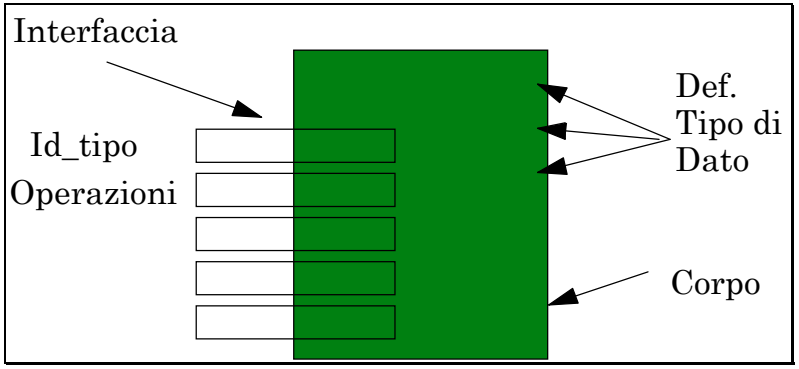
- ***Astrazioni di dato:***

  Il componente ha dati locali (nascosti) e rende visibili all'esterno i prototipi delle operazioni invocabili (procedure e funzioni) su questi dati locali, ma non gli identificatori dei dati. Attraverso una di queste operazioni si può assegnare un valore iniziale ai dati locali nascosti.



- ***Tipo di dato astratto:***

  Il componente esporta un identificatore di tipo T ed i prototipi delle operazioni eseguibili su dati dichiarati di questo tipo. I "clienti" del componente dichiarano e controllano quindi il tempo di vita delle variabili di tipo T.



Tavole

Una tavola è un tipo di dato astratto per rappresentare insiemi di coppie **<*chiave, attributi*>**.

Ciascuna coppia rappresenta dati riferiti ad un'unica entitàlogica (ad es., persona, documento, etc.) identificata in modo univoco dalla ***chiave***.

Esempio:

NOME	COGNOME	REDDITO	ALIQUOTA
...			

Possiamo, per esempio, rappresentare la tavola in memoria centrale, con un vettore di strutture:

```
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 (***file***).

→ In **C** si utilizzano ***file binari*** (lettura e scrittura di strutture).

Tavole: operazioni

Operazioni tipiche sulle tavole:

- **inserimento** di un elemento <Chiave, Attributi>

inserisci: tavola $\times$ *chiave* $\times$ *attributi* $\rightarrow$ *tavola*

- **cancellazione** di un elemento (nota la chiave)

cancella: tavola $\times$ *chiave* $\rightarrow$ *tavola*

- verifica di **appartenenza** di un elemento

esiste: tavola $\times$ *chiave* $\rightarrow$ *boolean*

- **ricerca** di un elemento nella tavola

ricerca: tavola $\times$ *chiave* $\rightarrow$ *attributi*

L'operazione di ***ricerca*** è la più importante.

Spesso la realizzazione è progettata in modo da ottimizzare questa operazione.

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): $O(\log_2(N))$

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 (campo 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.
- *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:

La rubrica telefonica viene rappresentata come un vettore di record (***rappresentazione sequenziale*** in memoria centrale).
La struttura di ciascun record è data dalla dichiarazione:

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

La rubrica (inizialmente vuota) sarà quindi rappresentata come variabile del tipo:

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

Una variabile intera mantiene il numero di elementi inseriti:

```
int inseriti;
```



```
#include <stdio.h>
#include <string.h>
/* 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()
{   int scelta, inseriti, fine;
    rubrica R;

    inseriti=0;   /*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);
}
```

```
int menu()
{   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

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;
}
```

La funzione **`inserimento`** produce come risultato il numero di elementi effettivamente inseriti nel vettore: nel caso in cui nella rubrica ci sia spazio sufficiente per l'inserimento del nuovo elemento il risultato è **`DIM+1`**; altrimenti, l'inserimento non può avere luogo e viene restituito **`DIM`** come risultato della chiamata.

Cancellazione

La cancellazione provoca l'eliminazione dalla rubrica (data come parametro) dell'elemento specificato attraverso l'immissione da input del suo nome:

```
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;
}
```

Dopo aver letto il nome dell'elemento da cancellare, la funzione ricerca, mediante la funzione **individua**, l'elemento da eliminare nella rubrica **R**.
Se l'elemento esiste (**k<N**), il valore di **k** rappresenta l'indice all'interno di **R** (shift, in questo caso; il valore di **DIM** viene decrementato).
Se l'elemento da cancellare non è presente nella rubrica, viene stampato un messaggio.
DIM (valore restituito) rappresenta il numero di elementi contenuti nella rubrica.

Ricerca

La ricerca si attua in due fasi: la prima individua la posizione dell'elemento cercato, mentre la seconda fase stampa (se l'elemento è presente nel vettore) l'elemento stesso:

```
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("\n%s\t%s\n",R[k].nome,R[k].tel);
    else
  printf("\n%s\t non trovato\n", e.nome);
}
```

Una volta acquisito il nome della persona da ricercare nella rubrica, la chiamata a **individua** produce come risultato un intero che viene assegnato alla variabile locale **k**; nel caso in cui l'elemento cercato non sia presente nella rubrica **R**, la funzione **individua** restituisce il valore predefinito **N**, altrimenti restituisca l'indice dell'elemento individuato nella rubrica.

In base al valore di **k** è quindi possibile differenziare il comportamento della funzione.

```
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;
}
```

Ricerca esaustiva, caso peggiore O(DIM).

✎ Esercitazione 6.1bis:

Ordinare la tavola tramite la funzione **`qsort`** (definendo una 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, DIM, sizeof(elemento), fcmpt);

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


Discussione

Tavole, rappresentazione sequenziale

In memoria centrale, realizzata attraverso *vettori*.

In memoria secondaria, attraverso *file sequenziali*.

I. Per vettori:

E` necessario porre 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)

II. La ricerca è sequenziale ed esaustiva in entrambi i casi (caso peggiore, N confronti, O(N))

III. Visibilità completa della struttura dati (sia come dato sia come tipo)

I. Rappresentazione sequenziale come file:

Esercizio 6.2:

Realizzare una rubrica telefonica come tavola in memoria di massa (file binario). Prevedere le operazioni di:
creazione della rubrica
stampa della rubrica
ricerca nella rubrica

```
#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);

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



```
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);
}


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);
}
```

```
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);
}

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);
}
```


II. Ottimizzazione della ricerca:

Si ottiene se gli elementi sono memorizzati in modo ordinato nella tavola.

Deve esistere un ***ordinamento sul campo chiave***. Questo induce un ordinamento sugli elementi della Tavola come segue:

$el_1=<k_1,Attr_1>$
$el_2=<k_2,Attr_2>$

el_1 "<" el_2 ***se e solo se*** k_1 "<" k_2

La ricerca può arrestarsi appena si incontra un elemento con chiave maggiore di quella cercata (caso medio N/2 confronti, caso peggiore N confronti).

Si complica l'operazione di inserimento di un nuovo elemento nella tavola + occorre mantenerla ordinata.

✎ Esercizio 6.3 (proposto):

Modificare il programma 6.1 realizzando la rubrica telefonica come tavola in memoria di centrale ordinata sulla chiave (vettore ordinato in base al nome).

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

Caso peggiore: O(DIM)

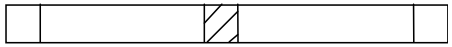
Ottimizzazione della ricerca (cont.):

Miglioramento ulteriore: ***ricerca binaria***.

Si accede all'elemento mediano <Chiave,Attr> della tavola.
Se Chiave"="k, fine della ricerca.
Altrimenti,
- se k"<"Chiave, ripeti la ricerca nella prima metà della tavola;
- se k"<"Chiave, ripeti la ricerca nella seconda metà della tavola.

✎ Esercizio 6.3 (cont.):

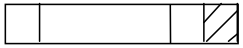
```
int individua_binaria(rubrica R, int DIM,
                                  elemento e)
{int i=0, medio, trovato=0,;
 while (i<DIM && !trovato)
    {medio=(i+DIM)/2;
     if (!strcmp(e.nome, R[medio].nome))
              trovato=1;
     else
       if (strcmp(e.nome,R[medio].nome)>0)
           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



medio = (i + DIM) /2

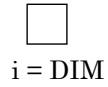
if R[medio].nome=e.nome---> trovato

if R[medio].nome > e.nome



if R[medio].nome < e.nome

Il procedimento si arresta quando:



Ad ogni passo si eliminano dalla ricerca metàdegli elementi della tavola "corrente".

Nel caso peggiore si eseguono $\log_2 N$ confronti (nel nostro caso $\log_2 DIM$)

La ricerca binaria può essere effettuata anche nel caso di file ordinati, purché sia possibile l'accesso diretto (***fseek***).

Esercizio 6.4:

Modificare l'esercizio 6.2, dove si è realizzata una rubrica telefonica come file binario, in modo tale da mantenere il file ordinato sul campo chiave Nome. Realizzare un menù che preveda la chiamata delle operazioni di:
1. creazione della rubrica (file vuoto)
2. inserimento di un elemento letto a terminale in rubrica
3. ricerca di un elemento, letta la chiave a terminale (*da svolgere*)
4. stampa dell'intera rubrica
5. fine programma.

Per posizionarsi all'interno di un file o ottenere la posizione corrente, si hanno a disposizione le funzioni:

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

dove:

`offset` dà la posizione, rispetto a **`origin`**, a cui portarsi sul file

`origin` dà la posizione rispetto a cui misurare l'offset, e può essere:
- l'inizio del file → **`SEEK_SET`**
- la posizione corrente nel file → **`SEEK_CUR`**
- la 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`**) - **`ftell()`** restituisce la lunghezza in byte rispetto all'origine

```c
/* fread example: read a complete file of
char */
#include <stdio.h>
#include <stdlib.h>

int main () {
  FILE * pFile;
  long lSize;
  char * buffer;
  size_t result;

  pFile = fopen ( "myfile.bin" , "rb" );
  if (pFile==NULL)
       {fputs ("File error",stderr);
        exit (1);}

  /* obtain file size: */
  fseek (pFile , 0 , SEEK_END);
  lSize = ftell (pFile);
  rewind (pFile);

  /* allocate memory to contain the whole
file:*/
  buffer = (char*) malloc
(sizeof(char)*lSize);
  if (buffer == NULL)
       {fputs ("Memory error",stderr);
        exit (2);}

  /*copy the file into the buffer:*/
  result = fread (buffer,1,lSize,pFile);
```



```c
  if (result != lSize)
       {fputs ("Reading error",stderr);
        exit (3);}

  /* the whole file is now loaded in the
memory buffer. */

  /* terminate */
  fclose (pFile);
  free (buffer);
  return 0;
}
```

Esercizio 6.4:

```
void ricerca_bin(char *v)
{ FILE *f; elemento e;
  char Nome[20];
  int trovato=0;
  long lSize,medium,init=0;

  printf("Nome da cercare: ");
  scanf("%s",Nome);
  f=fopen(v, "rb");
  fseek(f , 0 , SEEK_END);
  lSize = (ftell (f) / sizeof(elemento));

  while ((init<lSize) && !trovato)
   { medium=(init+lSize)/2);
     fseek(f, medium , SEEK_SET);
     if(fread(&e,sizeof(elemento),1, f)>0)
       if (!strcmp(e.nome, Nome) )
             { trovato=1;
                stampael(e);  }
        else
           if (strcmp(e.nome, Nome)>0)
                 lSize=medium;
            else  init=medium;
    }
  fclose(f);
}
```



```
/* ALTRA VERSIONE */


#include<stdio.h>
#include<string.h>
#include<stdlib.h>

/*dichiarazione dati*/
typedef struct {char nome[20];
                char tel[16];} elemento;


/*dichiarazione procedure e funzioni*/
int menu(void);
void creazione(char *name);
elemento richiesta(void);
elemento richiesta_nome(void);
int posizionamento(char *name,elemento El);
void stampafile(char *name);
void scrittura(char *name, elemento El,
               int offset);
void stampa_tel(char *name, elemento El,
               int offset);
```

```
/*SVILUPPO MAIN*/
main(void)
{ elemento dato;
  int scelta;
  int pos;
  do
  {
   scelta=menu();
   switch(scelta)
     {
      case 1: creazione("rubrica");
              break;

      case 2: dato=richiesta();
         pos=posizionamento("rubrica",dato);
              scrittura("rubrica",dato,pos);
              break;

      case 3: dato=richiesta_nome();
         pos=ricerca("rubrica",dato);
             stampa_tel("rubrica",dato,pos);
              break;

      case 4: stampafile("rubrica");
              break;

      case 5: break;
     }
   }while (scelta!=5);
}
```



```
/*SVILUPPO MENU*/
int menu(void)
{ int s;
 do
   {
    puts("\t\t\t MENU' DI CONTROLLO\n\n\n");
    puts(" 1 ---> Creazione archivio\n");
    puts(" 2 ---> Inserimento dato\n");
    puts(" 3 ---> Ricerca dato\n");
    puts(" 4 ---> Stampa dell'intero
                  archivio\n");
    puts(" 5 ---> Uscita programma\n\n");
    printf("Scelta:");
    scanf("%d",&s);
   }while ((s<1)&&(s>5));
    return s;
}


/*PROCEDURA PER LA CREAZIONE DI UN FILE*/
void creazione(char *name)
{FILE *fp;
 fp=fopen(name,"wb");
 if (fp==NULL)
    puts("\n\nERRORE DI APERTURA FILE\n\n");
 else
 puts("\n\n\n\n\n\nCREZIONE FILE
                          ESEGUITA\n\n");
 fclose(fp);
}
```

```
/*PROCEDURA PER L'INSERIMENTO DI UN DATO*/
elemento richiesta(void)
{
  elemento El;
  puts("\n\n\n\n\n\n\n\n\n");
  printf("Nome (max 20 caratteri):");
  scanf("%s",&El.nome);
  printf("Numero di telefono:");
  scanf("%s",&El.tel);
  return El;
}



/*PROCEDURA PER ACQUISIZIONE CHIAVE DI UN DATO
DA CERCARE*/
elemento richiesta_nome(void)
{
  elemento El;
  puts("\n\n\n\n\n\n\n\n\n");
  printf("Nome (max 20 caratteri):");
  scanf("%s",&El.nome);
  return El;
}
```



```
/* RICERCA DEL PUNTO IN CUI SI DEVE INSERIRE
   IL NUOVO DATO ALL'INTERNO DEL FILE  */
int posizionamento(char *name, elemento El)
{elemento Rd;
 FILE *fp;

 int offset=0;   /* Contatore posizione */

 fp=fopen(name,"rb");
 if (fp==NULL)
    puts("\n\nERRORE DI APERTURA FILE\n\n");
 if(fread(&Rd,sizeof(elemento),1, fp)<=0)
  { fclose(fp);
    /*file ancora vuoto*/
    return -1;
  }
    else
     {
      fseek(fp,0,SEEK_SET);
      fread(&Rd,sizeof(elemento),1, fp);
      do
      {
        if (strcmp(El.nome,Rd.nome)<=0)
           {  fclose(fp);
              return offset;
           }

        fread(&Rd,sizeof(elemento),1, fp);
        offset++;
      }while (!feof(fp));

      fclose(fp);
      return -1;
/*elemento da inserire + grande di tutti quelli
nel file*/
     }
}
```

```
void scrittura(char *name, elemento El,int offset)
{
 FILE *fp;
 elemento Rd;


 if (offset<0)
      { fp=fopen(name,"ab");
      fwrite(&El, sizeof(elemento), 1, fp);
      fclose(fp);
      }
 else
 {   fp=fopen(name,"r+b");
     fseek(fp,(offset*sizeof(elemento)),SEEK_SET);
     fread(&Rd,sizeof(elemento),1, fp);
     do{
         offset=(ftell(fp)-sizeof(elemento));
         fseek(fp,offset,SEEK_SET);
         fwrite(&El, sizeof(elemento), 1, fp);
         El=Rd;

       fseek(fp,offset+sizeof(elemento),SEEK_SET);
      }while
          ((fread(&Rd,sizeof(elemento),1,fp)>0));
      fwrite(&El, sizeof(elemento), 1, fp);
      fclose(fp);
 }


}
```



```
/* RICERCA IL DATO ALL'INTERNO DEL FILE  */
int ricerca(char *name, elemento El)
{elemento Rd;
 FILE *fp;

 int offset=0;   /* Contatore posizione */

 fp=fopen(name,"rb");
 if (fp==NULL)
    puts("\n\nERRORE DI APERTURA FILE\n\n");
 if(fread(&Rd,sizeof(elemento),1, fp)<=0)
  { fclose(fp);
    /*non trovato: file vuoto*/
    return -1;
  }
 else
  { fseek(fp,0,SEEK_SET);
    fread(&Rd,sizeof(elemento),1, fp);
    do
    { if (strcmp(El.nome,Rd.nome)==0)
          { fclose(fp);
            return offset;  /* trovato */
          }
      else
        if (strcmp(El.nome,Rd.nome)>0)
            { fclose(fp);
              return -1; /* non trovato */
            }
        else
         {fread(&Rd,sizeof(elemento),1, fp);
          offset++;
         }
     }while (!feof(fp));

    fclose(fp);
    return -1;       /* non trovato */
  }
}
```

```
/*PROCEDURA PER LA STAMPA A VIDEO DEL NUMERO DI
TELEFONO DEL DATO CERCATO*/
void stampa_tel(char *name, elemento El,
                int offset)
{  elemento Rd;
   FILE *fp;

   if (offset==-1)
      /*elemento non esiste in rubrica */
      puts("\n\nELEMENTO NON ESISTE\n\n");

      else
      {   /*riposizionamento nel file */

      fp=fopen(name,"rb");
      if (fp==NULL)
    puts("\n\nERRORE DI APERTURA FILE\n\n");

      fseek(fp,offset,SEEK_SET);

      if
        (fread(&Rd,sizeof(elemento),1,fp)>0)
         printf("%s",Rd.tel);

      fclose(fp);

      }

}
```



```
/*PROCEDURA PER LA STAMPA DELL'INTERO FILE*/
void stampafile(char *name)
{
 FILE*fp;
 elemento Rd;
 int p=0;
 fp=fopen(name,"rb");
 if (fp==NULL)
   puts("\n\nERRORE DI APERTURA FILE\n\n");
 while(fread(&Rd,sizeof(elemento),1, fp)>0)
   {printf("Nome:   %s          Telefono:   %s
\n",Rd.nome,Rd.tel);
    p=1;
   }
 if             ((fread(&Rd,sizeof(elemento),1,
fp)<=0)&&(p==0))
    puts ("\n\n\n\nArchivio vuoto\n\n\n\n");
  fclose(fp);
}
```

III. Limitazione della visibilità della struttura dati:

Si può ottenere realizzando un “componente software” dotato di una parte interfaccia (file .h) e una parte implementazione (file .c).