Listati

Capitolo 5

Listato 5.1 *Un programma in linguaggio C.*

```
#include <stdio.h>
main()
{
printf("Tre");
printf(" casettine");
printf(" dai");
printf(" tetti");
printf(" aguzzi");
}
```

Listato 5.2 *Uso di variabili.*

```
/* Calcolo area rettangolo */
#include <stdio.h>
main()
{
int base;
int altezza;
int area;
base = 3;
altezza = 7;
area = base*altezza;
printf("%d\n", area);
}
```

Listato 5.3 *Uso di costanti.*

```
/* Calcolo area rettangolo, prova utilizzo costanti */
#include <stdio.h>
#define BASE 3
#define ALTEZZA 7
main()
{
int area;
  area = BASE*ALTEZZA;
  printf("Base: %d\n", BASE);
  printf("Altezza: %d\n", ALTEZZA);
  printf("Area: %d\n", area);
}
```

Listato 5.4 *Immissione di valori.*

```
/* Calcolo area rettangolo */
#include <stdio.h>
int base, altezza, area;
main()
{
```

```
  printf("AREA RETTANGOLO\n\n");
  printf("Valore base: ");
  scanf("%d", &base);
  printf("Valore altezza: ");
  scanf("%d", &altezza);
  area = base*altezza;
  printf("Base: %d\n", base);
  printf("Altezza: %d\n", altezza);
  printf("Area: %d\n", area);
}
```

Listato 5.5 *Esempio di utilizzo di una funzione predefinita.*

```
/* Esempio di utilizzo di abs() */
#include <stdio.h>
#include <math.h>
main()
{
  int a, b, segmento, lunghezza;
  printf("\n\nLUNGHEZZA SEGMENTO\n");
  printf("Primo estremo: ");
  scanf("%d", &a);
  printf("Secondo estremo: ");
  scanf("%d", &b);
  segmento = a-b;
  lunghezza = abs(segmento);
  printf("Lunghezza segmento: %d\n", lunghezza);
}
```

Capitolo 6

Listato 6.1 *Due esempi di input/output di caratteri.*

```
#include <stdio.h>
main() main()
{
  char x, y, z;
  printf("Digita tre carat.: ");
  scanf("%c%c%c", &x, &y, &z);
  printf("Hai digitato: ");
  printf("%c%c%c\n", x, y, z);
}

#include <stdio.h>
main() main()
{
  char x, y, z;
  printf("Digita tre carat.: ");
  x = getchar();
  y = getchar();
  z = getchar();
  printf("Hai digitato: ");
  putchar(x);
  putchar(y);
  putchar(z);
  putchar('\n');
}
```

Capitolo 7

Listato 7.1 *Esempio di diramazione del flusso di esecuzione.*

```
/* Utilizzo di if-else */
#include <stdio.h>
main ()
{
int i;
printf("Dammi un intero: ");
scanf("%d", &i);
if(i<100)
printf("minore di 100\n");
else
printf("maggiore o uguale a 100\n");
}
```

Listato 7.2 *Esempio di utilizzo di istruzioni composte.*

```
/* Esempio di istruzioni composte */
#include <stdio.h>
int i;
int mag_100;
int min_100;
main ()
{
mag_100 = 0;
min_100 = 0;
printf("Dammi un intero: ");
scanf("%d", &i);
if(i<100) {
  printf("minore di 100\n");
  min_100 = 1;
}
else {
  printf("maggiore o uguale a 100\n");
  mag_100 = 1;
}
}
```

Listato 7.4 *Esempio di diramazione multipla del flusso di esecuzione.*

```
/* Esempio di utilizzo di case */
#include <stdio.h>
int x;
main()
{
printf("Digita una cifra: ");
scanf("%d", &x);
switch(x) {
  case 0:
  printf("zero\n");
  break;
  case 1:
    printf("uno\n");
    break;
  case 2:
```

```
    printf("due\n");
    break;
  case 3:
    printf("tre\n");
    break;
  case 4:
    printf("quattro\n");
    break;
  case 5:
    printf("cinque\n");
    break;
  default:
  printf("non compreso\n");
  break;
}
}
```

Listato 7.5 *Più valori costanti corrispondono allo stesso gruppo di istruzioni.*

```
/* Esempio di utilizzo di case */
#include <stdio.h>
char x;
main()
{
printf("Digita una cifra: ");
scanf("%c", &x);
switch(x) {
  case '2':
  case '4':
  case '6':
    printf("pari\n");
    break;
  case '1':
  case '3':
  case '5':
    printf("dispari\n");
    break;
  default:
    printf("altro\n");
}
}
```

Capitolo 7

Listato 9.1 *Iterazione con l'istruzione* `for`*.*

```
/* Esempio di utilizzo dell'istruzione for
   Calcola la somma di cinque numeri interi
   immessi dall'utente*/
#include <stdio.h>
int i, somma, numero;
main()
{
printf("SOMMA 5 NUMERI\n");
somma = 0;
for(i=1; i<=5; i=i+1) {
  printf("Inser. intero: ");
```

```
  scanf("%d", &numero);
  somma = somma + numero;
}
printf("Somma: %d\n",somma);
}
```

Listato 9.2 *Calcolo del fattoriale di* n.

```
/* Calcolo di n! (n fattoriale) */
#include <stdio.h>
main()
{
int n, fat, m;

printf("CALCOLO DI N!\n\n");
printf("Inser. n: ");
scanf("%d", &n);

fat = n;
for(m=n; m>2; m--)
  fat = fat*(m-1);

printf("Il fattoriale di: %d ha valore: %d\n", n, fat);
}
```

Listato 9.3 *Un'altra possibilità per il calcolo del fattoriale.*

```
/* Calcolo n! (n fattoriale) */
#include <stdio.h>
main()
{
int n, fat, aux;

printf("CALCOLO DI N!\n\n");
printf("Inser. n: ");
scanf("%d", &n);

fat = 1;
for(aux=2; aux<=n; aux++) fat = fat*aux;

printf("Il fattoriale di: %d ha valore: %d\n", n, fat);
}
```

Listato 9.4 *Esempio di utilizzo dell'istruzione* `while`.

```
/* Calcola la somma dei valori interi passati dall'utente
   Termina quando viene immesso il valore 0 (zero) */
#include <stdio.h>

main()
{
int somma, numero;
printf("SOMMA NUMERI\n");
printf("zero per terminare\n");
numero = 1;
somma = 0;
while(numero!=0) {
  printf("Inser. intero: ");
  scanf("%d", &numero);
```

```
  somma = somma+numero;
}
printf("Somma: %d\n",somma);
}
```

Listato 9.5 *Diramazione* `if` *all'interno di una iterazione* `while`.

```
/* Determina somma e maggiore dei valori immessi */
#include <stdio.h>

main()
{
int somma,numero,max,i;
printf("SOMMA E MAGGIORE\n");
printf("zero per finire\n");
numero = 1;
somma = 0;
max = 0;
i = 1;
while(numero!=0 && i<=10)
{
  printf("Valore int.: ");
  scanf("%d", &numero);
  if(numero>max)
    max = numero;
  somma = somma+numero;
  i++;
}
printf("Somma: %d\n", somma);
printf("Maggiore: %d\n", max);
}
```

Listato 9.6 *Esempio di utilizzo del costrutto* `do-while`.

```
/* Determina somma e maggiore dei valori immessi
   (esempio di uso do-while) */
#include <stdio.h>

main()
{
int somma,numero,max,i;
printf("SOMMA E MAGGIORE\n");
printf("zero per finire\n");
numero = 1;
somma = 0;
max = 0;
i = 1;
do {
    printf("Valore int.: ");
    scanf("%d", &numero);
    if(numero>max)
      max = numero;
    somma = somma+numero;
    i++;
}
while(numero!=0 && i<=10);
printf("Somma: %d\n", somma);
printf("Maggiore: %d\n", max);
}
```

Listato 9.7 *Esempio di cicli annidati.*

```
#include <stdio.h>

main() /* esempio cicli annidati */
{
int n, m, i, j;
printf("Inserire il numero di linee: \t");
scanf("%d", &n);
printf("Inserire il numero di colonne: \t");
scanf("%d", &m);
for(i=1; i<=n; i++) { /* inizio blocco ciclo esterno */
  printf("\n");
  for(j=1; j<=m; j++)
    printf("+");
} /* fine blocco ciclo esterno */
}
```

Listato 9.8 *Programma per il calcolo dello zero di una funzione.*

```
/* Determina lo zero della funzione f(x) = 2x3-4x+1 */
#include <stdio.h>
#include <math.h>

#define ERR 0.001

main()
{
float a, b, m;
float fa, fb, fm;
char x;

/* controllo validità a, b */
do {
    printf("Inserire a: ");
    scanf("%f", &a);
    printf("Inserire b: ");
    scanf("%f", &b);
    fa = 2*a*a*a-4*a+1; /* Calcolo della funzione per x=a */
    fb = 2*b*b*b-4*b+1; /* Calcolo della funzione per x=b */
}
while(fa*fb>0);

/* calcolo zero f */
do {
    m = (a+b)/2;
    fm = 2*m*m*m-4*m+1;   /* Calcolo della funzione per x=m */
    if(fm!=0) {
      fa = 2*a*a*a-4*a+1; /* Calcolo della funzione per x=a */
      fb = 2*b*b*b-4*b+1; /* Calcolo della funzione per x=b */
      if(fa*fm<0) b=m; else a=m;
      fm = 2*m*m*m-4*m+1; /* Calcolo della funzione per x=m */
}
}
while(fabs(fm) > ERR);

printf("Zero di f in %7.2f\n", m);
}
```

Capitolo 10

Listato 10.1 *Esempio di utilizzo di una variabile array.*

```
/* Memorizza in un array di interi i voti ottenuti da
   sei studenti e ne determina il maggiore, il minore e
   e la media                                              */
#include <stdio.h>
main()
{
int voti[6];
int i, max, min;
float media;

printf("VOTI STUDENTI\n\n");
/* Immissione voti */
for(i=0; i<=5; i++) {
  printf("Voto %d° studente: ", i+1);
  scanf("%d", &voti[i]);
}

/* Ricerca del maggiore */
max = voti[0];
for(i=1; i<=5; i++)
  if(voti[i]>max)
max = voti[i];

/* Ricerca del minore */
min = voti[0];
for(i=1; i<=5; i++)
  if(voti[i]<min)
min = voti[i];

/* Calcolo della media */
media = voti[0];
for(i=1; i<=5; i++)
  media = media + voti[i];
media = media/6;

printf("Maggiore: %d\n", max);
printf("Minore: %d\n", min);
printf("Media: %f\n", media);
}
```

Listato 10.2 *Esempio di utilizzo di un array.*

```
/* Carica i punteggi di n concorrenti su due prove
   Determina la classifica                          */
#include <stdio.h>
#define MAX_CONC 1000 /* massimo numero di concorrenti */
#define MIN_PUN 1     /* punteggio minimo per ogni prova */
#define MAX_PUN 10    /* punteggio massimo per ogni prova */

main()
{
float prova1[MAX_CONC], prova2[MAX_CONC], totale[MAX_CONC];
int i, n;

do {
    printf("\nNumero concorrenti: ");
    scanf("%d", &n);
```

```
}
while(n<1 || n>MAX_CONC);

/* Per ogni concorrente, richiesta punteggio
   nelle due prove                              */
for(i=0; i<n; i++) {
  printf("\nConcorrente n.%d \n", i+1);
  do {
      printf("Prima prova: ");
      scanf("%f", &prova1[i]);
  }
  while(prova1[i]<MIN_PUN || prova1[i]>MAX_PUN);

  do {
      printf("Seconda prova: ");
      scanf("%f", &prova2[i]);
  }
  while(prova2[i]<MIN_PUN || prova2[i]>MAX_PUN);
}

/* Calcolo media per concorrente */
for(i=0; i<n; i++)
  totale[i] = (prova1[i]+prova2[i])/2;

printf("\n CLASSIFICA\n");
for(i=0; i<n; i++)
  printf("%f %f %f \n", prova1[i], prova2[i], totale[i]);
}
```

Listato 10.3 *Esempio di utilizzo di un array bidimensionale.*

```
/* Caricamento di una matrice */
#include <stdio.h>

int mat[4][3];

main()
{
int i, j;
printf("\n \n CARICAMENTO DELLA MATRICE \n \n");
for(i=0; i<4; i++)
  for(j=0; j<3; j++) {
     printf("Inserisci linea %d colonna %d val: ", i, j);
     scanf("%d", &mat[i][j]);
};
/* Visualizzazione */
for(i=0; i<4; i++) {
  printf("\n");
  for(j=0; j<3; j++)
    printf("%5d", mat[i][j]);
}
}
```

Listato 10.4 *Inizializzazione di un array bidimensionale, seconda versione.*

```
/* Caricamento di una matrice
   le cui dimensioni vengono decise dall'utente */
#include <stdio.h>

#define MAXLINEE 100
#define MAXCOLONNE 100
```

```
int mat[MAXLINEE][MAXCOLONNE];

main()
{
int n, m;
int i, j;

/* Richiesta delle dimensioni */
do {
    printf("\nNumero di linee: ");
    scanf("%d", &n);
}
while((n>=MAXLINEE) || (n<1));

do {
    printf("Numero di colonne: ");
    scanf("%d", &m);
}
while((m>=MAXCOLONNE) || (m<1));
printf("\n \n CARICAMENTO DELLA MATRICE \n \n");
for(i=0; i<n; i++)
  for(j=0; j<m; j++) {
    printf("Inserisci linea %d colonna %d val:", i, j);
    scanf("%d", &mat[i][j]);
};

/* Visualizzazione */
for(i=0; i<n; i++) {
  printf("\n");
  for(j=0; j<m; j++)
    printf("%5d", mat[i][j]);
}
}
```

Listato 10.5 *Calcolo del prodotto tra matrici.*

```
/* Calcolo del prodotto di due matrici */
#include <stdio.h>

#define N 4
#define P 3
#define M 5

int mat1[N][P]; /* prima matrice */
int mat2[P][M]; /* seconda matrice */
int pmat[N][M]; /* matrice prodotto */

main()
{
int i, j, k;

printf("\n \n CARICAMENTO DELLA PRIMA MATRICE \n\n");
for(i=0; i<N; i++)
  for(j=0; j<P; j++) {
    printf("Inserisci linea %d colonna %d val:", i, j);
    scanf("%d", &mat1[i][j]);
};

printf("\n \n CARICAMENTO DELLA SECONDA MATRICE \n \n");
for(i=0; i<P; i++)
  for(j=0; j<M; j++) {
    printf("Inserisci linea %d colonna %d val:", i, j);
```

```
    scanf("%d", &mat2[i][j]);
};

/* Calcolo del prodotto */
for(i=0; i<N; i++)
  for(j=0; j<M; j++) {
    pmat[i][j] = 0;
  for(k=0; k<P; k++)
    pmat[i][j] = pmat[i][j] + mat1[i][k] * mat2[k][j];
};

printf("\n \n PRIMA MATRICE \n ");
for(i=0; i<N; i++) {
  printf("\n");
  for(j=0; j<P; j++)
    printf("%5d", mat1[i][j]);
}
printf("\n \n SECONDA MATRICE \n ");
for(i=0; i<P; i++) {
  printf("\n");
  for(j=0; j<M; j++)
    printf("%5d", mat2[i][j]);
}
printf("\n \n MATRICE PRODOTTO \n ");
for(i=0; i<N; i++) {
  printf("\n");
  for(j=0; j<M; j++)
    printf("%5d", pmat[i][j]);
}
}
```

Capitolo 11

Listato 11.1 *Dichiarazione, definizione e invocazione di una funzione.*

```
#include <stdio.h>
double cubo(float);
main()
{
  float a;
  double b;
  printf("Inserisci un numero: ");
  scanf("%f", &a);
  b = cubo(a);
  printf("%f elevato al cubo è uguale a %f", a, b);
}

double cubo(float c)
{
  return (c*c*c);
}
```

Listato 11.2 *Dichiarazioni e definizioni di funzioni.*

```
#include <stdio.h>

double quad(float);
```

```
double cubo(float);
double quar(float);
double quin(float);
double pote(float, int);

main()
{
  int base, esponente;
  double ptnz;
  printf(" Inserire base: " );
  scanf("%d", &base);
  printf(" Inserire esponente (0-5): ");
  scanf("%d", &esponente);

  ptnz = pote(base, esponente);

  if (ptnz == -1)
    printf("Potenza non prevista\n");
  else
    printf("La potenza %d di %d e' %f \n", esponente, base, ptnz);
}

double quad(float c)
{
  return(c*c);
}

double cubo(float c)
{
  return(c*c*c);
}

double quar(float c)
{
  return(c*c*c*c);
}

double quin(float c)
{
  return(c*c*c*c*c);
}

double pote(float b, int e)
{
  switch (e) {
    case 0: return (1);
    case 1: return (b);
    case 2: return (quad( b ));
    case 3: return (cubo( b ));
    case 4: return (quar( b ));
    case 5: return (quin( b ));
    default : return (-1);
  }
}
```

Listato 11.3 *Esemplificazione di mascheramento dei nomi.*

```
int x; /* nome globale */

f()
{
  int x;   /* x locale che nasconde x globale */
```

```
  x = 1;   /* assegna 1 a x locale */
  {
    int x; /* nasconde il primo x locale */
    x = 2; /* assegna 2 al secondo x locale */
  }
  x = 3;   /* assegna 3 al primo x locale */
  }
  scanf ("%d", &x); /* inserisce un dato in x globale */
…
```

Listato 11.4 *Chiamata di funzione.*

```
#include <stdio.h>

double area(float, float, char);

main()
{
  float b, h;
  double a;
  char p;

  printf("Inserire poligono (Triangolo/Rettangolo): ");
  scanf("%c", &p);

  printf("\nInserire base: ");
  scanf("%f", &b);
  printf("\nInserire altezza : ");
  scanf("%f", &h);

  a = area(b, h, p);

  printf("Il poligono (b = %f, h = %f) ha area %f\n", b, h, a);
}

double area(float base, float altezza, char poligono)
{
  switch (poligono) {
    case 'T': return (base * altezza/2.0);
    case 'R': return (base * altezza);
    default : return -1;
}
}
```

Listato 11.5 *Le funzioni come strumento di riutilizzo del codice.*

```
#include <stdio.h>

double area(float, float, char);

main()
{
  float b, h;
  double tri, ret;
  printf("Inserire base: ");
  scanf("%f", &b);
  printf("Inserire altezza: ");
  scanf("%f", &h);

  tri = area(b, h, 'T');
```

```
  ret = area(b, h, 'R');

  printf("Il triangolo (b = %f, h = %f) ha area %f\n", b, h, tri);
  printf("Il rettangolo (b = %f, h = %f) ha area %f\n", b, h, ret);
}

double area(float base, float altezza, char poligono)
{
  switch (poligono) {
    case 'T': return (base * altezza/2.0);
    case 'R': return (base * altezza);
    default : return -1;
  }
}
```

Listato 11.6 *Passaggio di parametri con variabile globale.*

```
#include <stdio.h>
char str[] = "Lupus in fabula";
int lungString(void);

main()
{
  int l;
  l = lungString();
  printf("La stringa %s ha %d caratteri\n", str, l);
}

int lungString(void)
{
  int i;
  for (i = 0; str[i] != '\0'; i++);
  return i;
}
```

Listato 11.7 *Esempio di funzione "lavandino".*

```
#include <stdio.h>
#define DIM_INT 16

void stampaBin ( int );
main()
{
  char resp[2];
  int num;

  resp[0] = 's';

  while( resp[0] == 's' ) {
    printf("\nInserisci un intero positivo: ");
    scanf("%d", &num);
    printf("La sua rappresentazione binaria e': ");

    stampaBin(num);

    printf("\nVuoi continuare? (s/n): ");
    scanf("%s",resp);
  }
}

void stampaBin(int v)
```

```
{
  int i, j;
  char a[DIM_INT];

  if (v == 0)
    printf("%d", v);
  else {
    for( i=0; v != 0; i++) {
      a[i] = v % 2;
      v /= 2;
  }
  for(j = i-1 ; j >= 0; j--)
    printf("%d", a[j]);
  }
}
```

Listato 11.8 *Funzioni senza parametri.*

```
#include <stdio.h>

void messErr( void );

main()
{
  int a, b, c;

  printf("Inserire dividendo:");
  scanf("%d", &a);
  printf("Inserire divisore:");
  scanf("%d", &b);
  if (b != 0) {
    c = a/b;
    printf("%d diviso %d = %d\n", a, b, c);
  }
  else
    messErr();
}

void messErr( void )
{
  int i;
  char c;
  for (i = 0; i <= 20; i++) printf("\n");
    printf(" ERRORE! DENOMINATORE NULLO");
    printf("\n Premere un tasto per continuare\n");
    scanf("%c%c", &c, &c);
}
```

Capitolo 12

Listato 12.1 *Ricerca completa.*

```
/* Ricerca sequenziale di un valore nel vettore */
#include <stdio.h>

#define MAX_ELE 1000 /* massimo numero di elementi */

main()
```

```
{
char vet[MAX_ELE];
int i, n;
char c;

/* Immissione lunghezza della sequenza */
do {
    printf("\nNumero elementi: ");
    scanf("%d", &n);
}
while(n<1 || n>MAX_ELE);

/* Immissione elementi della sequenza */
for(i=0; i<n; i++) {
  printf("\nImmettere carattere n.%d: ",i);
  scanf("%1s", &vet[i]);
}

printf("Elemento da ricercare: ");
scanf("%1s", &c);

/* Ricerca sequenziale */
i = 0;
while(c!=vet[i] && i<n-1) ++i;
if(c==vet[i])
  printf("\nElemento %c presente in posizione %d\n",c,i);
else
  printf("\nElemento non presente!\n");
}
```

Listato 12.2 *Programma completo di immissione, ordinamento e ricerca.*

```
/* Ricerca binaria */
#include <stdio.h>

main()
{
char vet[6]; /* array contenente i caratteri immessi */
int i,n,k,p;
char aux;    /* variabile di appoggio per lo scambio */
char ele;    /* elemento da ricercare */
int basso, alto, pos; /* usati per la ricerca binaria */

/* Immissione caratteri */
n = 6;
for(i=0;i<=n-1; i++) {
  printf("vet %d° elemento: ", i+1);
  scanf("%1s", &vet[i]);
}

/* Ordinamento ottimizzato */
p = n;
do {
  k = 0;
  for(i=0; i<n-1; i++) {
    if(vet[i]>vet[i+1]) {
      aux = vet[i]; vet[i] = vet[i+1]; vet[i+1] = aux;
      k = 1; p = i+1;
    }
  }
n = p;
}
```

```
while(k==1 && n>1);

printf("\nElemento da ricercare: ");
scanf("%1s", &ele);

/* Ricerca binaria */
n = 6;
alto = 0; basso = n-1; pos = -1;
do {
  i = (alto+basso)/2;
  if(vet[i]==ele) pos = i;
  else
    if(vet[i]<ele)
      alto = i+1;
    else
      basso = i-1;
}
while(alto<=basso && pos==-1);

if(pos != -1)
  printf("\nElemento %c presente in posizione %d\n",ele,pos);
else
  printf("\nElemento non presente! %d\n", pos);
}
```

Listato 12.3 *Fusione di due array.*

```
/* Fusione di due sequenze ordinate */
#include <stdio.h>
#define MAX_ELE 1000
main()
{
char vet1[MAX_ELE];    /* prima sequenza */
char vet2[MAX_ELE];    /* seconda sequenza */
char vet3[MAX_ELE*2];  /* merge */

int n;                 /* lunghezza prima sequenza */
int m;                 /* lunghezza seconda sequenza */

char aux;              /* variabile di appoggio per lo scambio */

int i, j, k, p, n1, m1;

do {
  printf("Lunghezza prima sequenza: ");
  scanf("%d", &n);
}
while(n<1 || n>MAX_ELE);

/* Caricamento prima sequenza */
for(i = 0;i <= n-1; i++) {
  printf("vet1 %d° elemento: ",i+1);
  scanf("%1s", &vet1[i]);
}

do {
  printf("Lunghezza seconda sequenza: ");
  scanf("%d", &m);
}
while(m<1 || m>MAX_ELE);

/* Caricamento seconda sequenza */
```

```
for(i=0; i<=m-1; i++) {
  printf("vet2 %d° elemento: ",i+1);
  scanf("%1s", &vet2[i]);
}

/* Ordinamento prima sequenza */
p = n; n1 = n;
do {
  k = 0;
  for(i = 0; i < n1-1; i++) {
    if(vet1[i]> vet1[i+1]) {
      aux = vet1[i]; vet1[i] = vet1[i+1]; vet1[i+1] = aux;
      k = 1; p = i+1;
    }
  }
n1 = p;
}
while(k==1 && n1>1);

/* Ordinamento seconda sequenza */
p = m; m1 = m;
do {
  k = 0;
  for(i=0; i<m1 - 1; i++) {
    if(vet2[i]>vet2[i+1]) {
      aux = vet2[i]; vet2[i] = vet2[i+1]; vet2[i+1] = aux;
      k = 1; p = i+1;
    }
  }
m1 = p;
}
while(k==1 && n2>1);

/* Fusione delle due sequenze (merge) */
i = 0; j = 0; k = 0;
do {
  if(vet1[i]<=vet2[j])
    vet3[k++] = vet1[i++];
  else
    vet3[k++] = vet2[j++];
}
while(i<n && j<m);

if(i<n)
  for(; i<n; vet3[k++] = vet1[i++])
    ;
else
  for(; j<m; vet3[k++] = vet2[j++])
    ;

/* Visualizzazione della fusione */
for(i=0; i<k; i++)
  printf("\n%c", vet3[i]);
}
```

Caso di Studio I
Gestione di una sequenza

Listato Caso di studio I *Gestione di una sequenza con l'uso di funzioni per immissione, ordinamento, ricerca completa, ricerca binaria e visualizzazione.*

```
#include <stdio.h>


#define MAX_ELE 1000 /* massimo numero di elementi */
int vet[MAX_ELE];    /* array che ospita la sequenza */

void gestioneSequenza(void);
int immissione(void);
void ordinamento(int);
int ricerca(int, int);
int ricBin(int, int);
void visualizzazione(int);


int main()
{
  gestioneSequenza();
}


void gestioneSequenza()
{
int n;
int scelta = -1;
char invio;
int ele, posizione;

while(scelta != 0) {
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  printf("\t\t\t GESTIONE SEQUENZA");
  printf("\n\n\n\t\t\t 1. Immissione");
  printf("\n\n\t\t\t 2. Ordinamento");
  printf("\n\n\t\t\t 3. Ricerca completa");
  printf("\n\n\t\t\t 4. Ricerca binaria");
  printf("\n\n\t\t\t 5. Visualizzazione");
  printf("\n\n\t\t\t 0. Fine");
  printf("\n\n\n\t\t\t\t Scegliere una opzione: ");
  scanf("%d", &scelta);
  scanf("%c", &invio);
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

  switch(scelta) {
    case 1: n = immissione();
            break;
    case 2: ordinamento(n);
            break;
    case 3: printf("Elemento da ricercare: ");
            scanf("%d", &ele);
            scanf("%c", &invio);
            posizione = ricerca(n, ele);
            if(ele == vet[posizione])
              printf("\nElem. %d presente in posizione %d\n", ele, posizione);
            else
            printf("\nElemento non presente!\n");
            printf("\n\n Premere Invio per continuare...");
            scanf("%c", &invio);
            break;
    case 4: printf("Elemento da ricercare: ");
            scanf("%d", &ele);
```

```
            scanf("%c", &invio);
            posizione = ricBin(n, ele);
            if(posizione != -1)
              printf("\nElem. %d presente in posizione %d\n", ele, posizione);
            else
              printf("\nElemento non presente!\n");
            printf("\n\n Premere Invio per continuare...");
            scanf("%c", &invio);
            break;
      case 5: visualizzazione(n);
            break;
  }
}
}


int immissione()
{
int i, n;

do {
    printf("\nNumero elementi: ");
    scanf("%d", &n);
}
while (n < 1 || n > MAX_ELE);

for(i = 0; i < n; i++) {
  printf("\nImmettere un intero n.%d: ",i);
  scanf("%d", &vet[i]);
}
return(n);
}

void ordinamento(int n)
{
int i, p, k, n1;
int aux;

p = n; n1 = p;
do {
    k = 0;
    for(i = 0; i < n1-1; i++)
    if(vet[i] > vet[i+1]) {
      aux = vet[i]; vet[i] = vet[i+1];
      vet[i+1] = aux;
    k = 1; p = i + 1;
   }
  n1 = p;
}
while (k == 1 && n1>1);
}

/* Ricerca sequenziale */
int ricerca (int n, int ele)
{
int i;
i = 0;

while (ele != vet[i] && i < n-1) ++i;
return(i);
}
```

```
/* ricerca binaria */
int ricBin(int n, int ele)
{
int i, alto, basso, pos;

alto = 0; basso = n - 1; pos = -1;
do {
    i = (alto+basso)/2;
    if(vet[i] == ele) pos = i;
    else if(vet[i] < ele) alto = i + 1;
         else basso = i - 1;
}
while(alto <= basso && pos == -1);
return(pos);
}


void visualizzazione( int n )
{
int i;
char invio;

for(i = 0; i < n; i++)
  printf("\n%d", vet[i]);
printf("\n\n Premere Invio per continuare...");
scanf("%c", &invio);
}
```

Capitolo 13

Listato 13.1 *Visualizzazione di differenti rappresentazioni di caratteri.*

```
/* Visualizzazione dei caratteri di una stringa */
#include <stdio.h>
char frase[] = "Analisi, requisiti ";

main()
{
int i=0;
while(frase[i]!='\0') {
  printf("%c = %d = %o \n", frase[i], frase[i], frase[i]);
  i++;
}
}
```

Listato 13.2 *Copia di un array di caratteri.*

```
/* Copia di una stringa su un'altra */
#include <stdio.h>

char frase[] = "Analisi, requisiti ";

main()
{
int i;
char discorso[80];
for(i=0; (discorso[i]=frase[i])!='\0'; i++)
```

```
  ;
printf(" originale: %s \n copia: %s \n", frase, discorso);
}
```

Listato 13.3 *Concatenazione di array di caratteri.*

```
/* Concatenazione di due stringhe */
#include <stdio.h>

char frase[160] = "Analisi, requisiti ";

main()
{
char dimmi[80];
int i, j;

printf("Inserisci una parola: ");
scanf("%s", dimmi);

for(i=0; (frase[i])!='\0'; i++)
  ;
for(j=0; (frase[i]=dimmi[j])!='\0'; i++,j++)
  ;
printf("frase: %s \n", frase);
}
```

Listato 13.4 *Immissione di caratteri con* `getchar`.

```
/* Concatenazione di due stringhe
   Introduzione della seconda stringa con getchar */
#include <stdio.h>

char frase[160] = "Analisi, requisiti ";

main()
{
char dimmi[80];
int i, j;

printf("Inserisci una parola: ");
for(i=0; (dimmi[i]=getchar())!='\n'; i++)
  ;
dimmi[i]='\0';
for(i=0; frase[i]!='\0'; i++)
  ;
for(j=0; (frase[i]=dimmi[j])!='\0'; i++,j++)
  ;
printf(" frase: %s \n", frase);
}
```

Listato 13.5 *Confronto fra array di caratteri.*

```
#include <stdio.h>
/* Confronto fra due stringhe */

char prima[160] = "mareggiata";

main()
{
```

```
char seconda[80];
int i;

printf("Inserisci una parola: ");
for(i=0; ((seconda[i]=getchar()) != '\n') && (i<80) ;i++)
  ;
seconda[i]='\0';
for(i=0; (prima[i] == seconda[i]) && (prima[i] != '\0') && (seconda[i] != '\0');
i++)
  ;
if(prima[i]==seconda[i])
  printf("Sono uguali\n");
else
  if(prima[i]>seconda[i])
printf("La prima e' maggiore della seconda\n");
else
printf("La seconda e' maggiore della prima\n");
}
```

Listato 13.6 *Esempio di utilizzo di* `strcmp`.

```
/* Confronto tra due stringhe con strcmp */
#include <stdio.h>
#include <string.h>

char prima[160] = "mareggiata";

main()
{
char seconda[80];
int i, x;

printf("Inserisci una parola: ");
for(i=0; ((seconda[i]=getchar())!='\n') && (i<80); i++)
  ;
seconda[i] = '\0';

if( (x = (strcmp(prima, seconda))) == 0)
  printf("Sono uguali\n");
else
  if(x>0)
    printf("La prima e' maggiore della seconda\n");
  else
    printf("La seconda e' maggiore della prima\n");
}
```

Listato 13.7 *Conversione da stringa a intero con la funzione* `atoi`.

```
/* Esempio di conversione da stringa a intero */

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

main()
{
char annoNascita[10];
char annoCorrente[10];
int anni;

printf("Anno di Nascita: ");
```

```
scanf("%s", &annoNascita);
printf("Anno Corrente: ");
scanf("%s", &annoCorrente);

anni = atoi(annoCorrente)-atoi(annoNascita);

printf("Eta': %d\n", anni);
}
```

Capitolo 14

Listato 14.1 *Il passaggio dei parametri per indirizzo.*

```
#include <stdio.h>

void scambia(int, int);

main()
{
  int x, y;

  x = 8;
  y = 16;
  printf("Prima dello scambio\n");
  printf("x = %d, y = %d\n", x, y);

  scambia(x, y);

  printf("Dopo lo scambio\n");
  printf("x = %d, y = %d\n", x, y);
}


/* Versione KO di scambia */
void scambia(int a, int b)
{
  int temp;

  temp = a;

  a = b;
  b = temp;
}
```

Listato 14.2 *Ancora sullo scambio di valori.*

```
#include <stdio.h>

void scambia(int *, int *);

main()
{
  int x, y;
  x = 8;
  y = 16;
  printf("Prima dello scambio\n");
  printf("x = %d, y = %d\n", x, y);
```

```
  scambia(&x, &y);

  printf("Dopo lo scambio\n");
  printf("x = %d, y = %d\n", x, y);
}


/* Versione OK di scambia */
void scambia(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
```

Listato 14.3 *Passaggio di un array.*

```
#include <stdio.h>

char str[] = "BATUFFO";

int strlen(char *);

main()
{
printf("la stringa %s ha lunghezza %d\n", str, strlen(str));
}


int strlen(char *p)
{
  int i = 0;
  while (*p++) i++;
  return i;
}
```

Listato 14.4 *Esempio di uso della funzione* `memcpy`.

```
/* Esempio di uso di memcpy */

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

int main ()
{
  char sorg[]="BATUFFO";
  char dest[40];
  char dest2[40];
  char pausa;
  memcpy (dest, sorg, strlen(sorg)+1);
  memcpy (dest2, "Copia riuscita", 15);
  printf ("sorgente: %s\ndestinazione: %s\n", sorg, dest);
  printf ("destinazione2: %s\n", dest2);
  scanf("&c", pausa);
}
```

Listato 14.5 *Esempio di uso della funzione* `memmove`.

```
/* Esempio di uso di memmove */

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

int main ()
{
  char sorgDest[100]= "gattonava cespuglio per nascondersi fuggire";
  char pausa;
  printf ("Prima: %s\n", sorgDest);
  memmove(sorgDest+20, sorgDest+10, 34);
  printf ("Dopo: %s\n", sorgDest);
  scanf("&c", pausa);
}
```

Listato 14.6 *Esempio di uso della funzione* `memcmp`.

```
/* Esempio di uso di memcmp */
#include <stdio.h>
#include <string.h>
int main ()
{
  char uno[]= "A";
  char due[]= "Z";
  int r;
  char pausa;
  r = memcmp(uno, due, 1);
  printf ("%i ", r);
  r = memcmp(due, uno, 1);
  printf ("%i\n", r);
  scanf("&c", pausa);
}
```

Caso di Studio II
Gestione di una sequenza con uso dei puntatori

Listato Caso di studio II *Gestione di una sequenza con il passaggio di un puntatore ad array alle funzioni di immissione, ordinamento, ricerca completa, ricerca binaria e visualizzazione.*

```
#include <stdio.h>

#define MAX_ELE 1000 /* massimo numero di elementi */
void gestioneSequenza(void);
int immissione(int *);
void ordinamento(int, int *);
int ricerca(int, int , int *);
int ricBin(int, int , int *);
void visualizzazione(int, int *);

main()
{
  gestioneSequenza();
}


void gestioneSequenza()
{
```

```
int sequenza[MAX_ELE]; /* array che ospita la sequenza */
int n;
int scelta = -1;
char invio;
int ele, posizione;

while(scelta != 0) {
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  printf("\t\t\t GESTIONE SEQUENZA");
  printf("\n\n\n\t\t\t 1. Immissione");
  printf("\n\n\t\t\t 2. Ordinamento");
  printf("\n\n\t\t\t 3. Ricerca completa");
  printf("\n\n\t\t\t 4. Ricerca binaria");
  printf("\n\n\t\t\t 5. Visualizzazione");
  printf("\n\n\t\t\t 0. fine");
  printf("\n\n\n\t\t\t\t Scegliere una opzione: ");
  scanf("%d", &scelta);
  scanf("%c", &invio);
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

  switch(scelta) {
    case 1: n = immissione(sequenza);
              break;
    case 2: ordinamento(n, sequenza);
              break;
    case 3: printf("Elemento da ricercare: ");
            scanf("%d", &ele);
            scanf("%c", &invio);
            posizione = ricerca(n, ele, sequenza);
          if(ele == sequenza[posizione])
            printf("\nElem. %d presente in posizione %d\n", ele, posizione);
          else
            printf("\nElemento non presente!\n");
          printf("\n\n Premere Invio per continuare...");
          scanf("%c", &invio);
          break;
    case 4: printf("Elemento da ricercare: ");
            scanf("%d", &ele);
            scanf("%c", &invio);
            posizione = ricBin(n, ele, sequenza);
            if(posizione != -1)
              printf("\nElem. %d presente in posizione %d\n", ele, posizione);
            else
              printf("\nElemento non presente!\n");
            printf("\n\n Premere Invio per continuare...");
            scanf("%c", &invio);
            break;
    case 5: visualizzazione(n, sequenza);
            break;
  }
}
}


int immissione(int *vet)
{
int i, n;

do {
    printf("\nNumero elementi: ");
    scanf("%d", &n);
}
while (n < 1 || n > MAX_ELE);
```

```
for(i = 0; i < n; i++) {
  printf("\nImmettere un intero n.%d: ",i);
  scanf("%d", &vet[i]);
}
return(n);
}


void ordinamento(int n, int *vet)
{
int i, p, k, n1;
int aux;
p = n; n1 = p;
do {
    k = 0;
    for(i = 0; i < n1-1; i++)
    if(vet[i] > vet[i+1]) {
      aux = vet[i]; vet[i] = vet[i+1];
      vet[i+1] = aux;
      k = 1; p = i + 1;
}
n1 = p;
}
while (k == 1 && n1>1);
}


/* Ricerca sequenziale */
int ricerca(int n, int ele, int *vet)
{
int i;
i = 0;

while (ele != vet[i] && i < n-1) ++i;
return(i);
}


/* ricerca binaria */
int ricBin(int n, int ele, int *vet)
{
int i, alto, basso, pos;
alto = 0; basso = n - 1; pos = -1;

do {
    i = (alto+basso)/2;
    if(vet[i] == ele) pos = i;
    else if(vet[i] < ele) alto = i + 1;
         else basso = i - 1;
}
while(alto <= basso && pos == -1);
return(pos);
}


void visualizzazione( int n, int *vet)
{
int i;
char invio;

for(i = 0; i < n; i++)
  printf("\n%d", vet[i]);
```

```
printf("\n\n Premere Invio per continuare...");
scanf("%c", &invio);
}
```

Capitolo 15

Listato 15.1 *Calcolo del fattoriale mediante una funzione ricorsiva.*

```
/* Calcolo del fattoriale con una funzione ricorsiva */
#include <stdio.h>

int fat(int);

main()
{
int n;
printf("CALCOLO DI n!\n\n");
printf("Inser. n: \t");
scanf("%d", &n);
printf("Il fattoriale di: %d ha valore: %d\n", n, fat(n));
}


int fat(int n)
{
if(n==0)
  return(1);
else
  return(n*fat(n-1));
}
```

Listato 15.2 *Calcolo delle disposizioni semplici.*

```
/* Calcolo delle disposizioni semplici di n oggetti presi k a k */
#include<stdio.h>

int dispo(int, int, int);

main()
{
int n, k;
printf("Disposizioni semplici di k su n oggetti\n");
printf("Inser. n: \t");
scanf("%d", &n);
printf("Inser. k: \t");
scanf("%d", &k);
printf("Le dispos. sempl. di %d su %d sono: %d\n", k, n, dispo(k, n, n));
}


int dispo(int k, int n, int m)
{
if(n==m-k)
  return(1);
else
  return(n*dispo(k, n-1, m));
}
```

Listato 15.3 *Calcolo delle combinazioni semplici; vengono utilizzate le funzioni* `dispo` *e* `fat` *viste precedentemente.*

```
/* Calcolo delle combinazioni semplici di n oggetti presi k a k */

#include <stdio.h>

int comb(int, int);
int dispo(int, int, int);
int fat(int);

main()
{
int n, k;
printf("Combinazioni semplici di k su n oggetti\n");
printf("Inserire n: \t");
scanf("%d", &n);
printf("Inserire k: \t");
scanf("%d", &k);
printf("Le combin. sempl. di %d su %d sono: %d\n",k,n,comb(k,n));
}


comb(int k, int n)
{
return(dispo(k, n, n)/fat(k));
}


int dispo(int k, int n, int m)
{
if(n==m-k)
  return(1);
else
  return(n*dispo(k, n-1, m));
}


fat(int n)
{
if(n==0)
  return(1);
else
  return(n*fat(n-1));
}
```

Listato 15.4 *Calcolo dei termini della successione di Fibonacci.*

```
/* Calcolo dei numeri di Fibonacci */

#include <stdio.h>

long int fibo(int);

main()
{
int n;

printf("Successione di Fibonacci f(0)=1 f(1)=1 f(n)=f(n-1)+f(n-2)");
```

```
printf("\nInserire n: \t");
scanf("%d", &n);
printf("Termine della successione di argomento %d: %d\n", n, fibo(n));
}


long int fibo(int n)
{
if(n==0) return(0);
else if(n==1) return(1);
     else return(fibo(n-1)+fibo(n-2));
}
```

Listato 15.5 *Programma ricorsivo per la torre di Hanoi.*

```
/* Programma ricorsivo per risolvere la torre di Hanoi */
#include <stdio.h>

#define DISCHI 4
int mossa;
void hanoi(int, char, char, char);
void muovi(int, char, char);

main()
{
mossa = 0;
printf("\nMosse da eseguire per spostare %d dischi", DISCHI);
printf("\n--------------------------------------");
hanoi(DISCHI, 'A', 'B', 'C'); /* Chiamata della procedura hanoi */
}


/* Funzione ricorsiva "hanoi" per spostare una torre di n
   dischi da pioloP a pioloA usando aus come piolo di ausilio */
void hanoi(int n, char pioloP, char pioloA, char aus)
{
if(n==1) muovi(1, pioloP, pioloA);
else {
  hanoi(n - 1, pioloP, aus, pioloA);
  muovi(n, pioloP, pioloA);
  hanoi(n - 1, aus, pioloA, pioloP);
}
}

/* Funzione "muovi" per spostare il disco nd
   dal piolo di partenza pP al piolo di arrivo pA */
void muovi(int nd, char pP, char pA)
{
char invio;
mossa = mossa + 1;
printf("\n%3d",mossa);
printf(": muovere disco %d da %c a %c", nd, pP, pA);
scanf("%c", &invio);
}
```

Listato 15.6 *Ordinamento di una sequenza con il metodo quicksort.*

```
/* Ordinamento quicksort di un array di int */

#include <stdio.h>
```

```
#define N 10 /* numero elementi dell'array */
int v[N];    /* array contenente gli interi immessi */

void quick(int, int);
void scambia(int *, int *);

main()
{
int i;
for(i=0; i<N; i++) {
  printf("\nImmettere un intero n.%d: ",i);
  scanf("%d", &v[i]);
}
quick(0,N-1);      /* Chiamata della procedura quick */

for(i=0; i<N; i++) /* Sequenza ordinata */
  printf("\n%d", v[i]);
putchar('\n');
}


/* Procedura ricorsiva "quick" */
void quick(int sin, int des)
{
int i, j, media;
media= (v[sin]+v[des]) / 2;
i = sin;
j = des;

do {
  while(v[i]<media) i = i+1;
  while(media<v[j]) j = j-1;
  if(i<=j) {
    scambia(&v[i], &v[j]);
    i = i+1;
    j = j-1;
  }
}
while (j>=i);

if(sin<j) quick(sin, j); /* Invocazione ricorsiva */
if(i<des) quick(i, des); /* Invocazione ricorsiva */
}


void scambia(int *a, int *b)
{
int temp;
temp = *a;
*a = *b;
*b = temp;
}
```

Listato 15.7 *Programma di soluzione al problema delle otto regine.*

```
/* Soluzione ricorsiva del problema delle otto regine */
#include <stdio.h>

#define DIM 8
int riga[DIM]; /*riga[i] i-esima riga */
```

```
int dia1[2*DIM-1]; /*dia1[i] i-esima diagonale secondaria */
int dia2[2*DIM-1]; /*dia2[i] i-esima diagonale principale */
/* riga[i], dia1[i], dia2[i] possono assumere
   i valori 0 o 1: 0 libera, 1 sotto scacco */

int colonna[DIM]; /*colonna[j]=i regina nella casella (i,j)*/

int nsol; /*numero soluzioni*/
char invio;

void verifica(int);
void visualizzazione(void);
void v(void);

main()
{
int i;
nsol=0;
/*inizializzazione righe e diagonali (libere)*/
for(i=0;i<DIM;riga[i++]=0);
for(i=0;i<2*DIM-1;dia1[i++]=0);
for(i=0;i<2*DIM-1;dia2[i++]=0);
verifica(0);
printf("\nNumero complessivo delle soluzioni = %d\n", nsol);
}


void verifica(int j)
{
int i;
for(i=0;i<DIM;i++) {

  if((riga[i]==0) && (dia1[i+j]==0) && (dia2[i-j+7]==0)) {
    colonna[j] =i; /*regina nella casella (i,j)*/
    riga[i] =1;    /*riga i-esima sotto scacco*/

    dia1[i+j] =1;  /*diag. sec.(i+j)esima sotto scacco*/
    dia2[i-j+7]=1; /*diag. pr.(i-j+7)esima sotto scac.*/

    if(j<DIM-1) verifica(j+1);   /*sostituito j con j+1*/
    else {                       /*trovata una soluzione*/
      nsol=nsol+1;
      visualizzazione();
      scanf("%c",&invio);
    }

    /*inizializzazione righe e diagonali (libere)*/
    riga[i] = 0;
    dia1[i+j] = 0;
    dia2[i-j+7] = 0;
  }
}
}


void visualizzazione(void)
{
int k;
printf("\n----- Soluzione numero %d -----\n", nsol);
for(k=0;k<DIM;k++)
  printf("%d ", colonna[k]);
printf("\n");
}
```

Capitolo 16

Listato 16.1 *La struttura* `automobile`.

```
/* Esempio di definizione di una struttura */

#include <stdio.h>

struct automobile {
  char *marca;
  char *modello;
  int venduto;
};

main()
{
struct automobile a1, a2;

a1.marca = "FERRARI";
a1.modello = "F40";
a1.venduto = 200;

a2.marca = "OPEL";
a2.modello = "ASTRA";
a2.venduto = 1200;

printf("marca auto = %s\n", a1.marca);
printf("modello auto = %s\n", a1.modello);
printf("vendute = %d\n", a1.venduto);
printf("marca auto = %s\n", a2.marca);
printf("modello auto = %s\n", a2.modello);
printf("vendute = %d\n", a2.venduto);
}
```

Caso di Studio III
Gestione anagrafica

Listato del Caso di studio III *Gestione anagrafica.*

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

#define MAXELE  30
#define DIM     31
#define MENU     0
#define INS      1
#define CAN      2
#define RIC      3
#define VIS      4
#define OUT    100
```

```
/* Semplice struttura che modella una persona */
struct per {
  char cognome[DIM];
  char nome[DIM];
  char ind[DIM];
  int eta;
};

/* Vettore persone */
struct per anag[MAXELE];

/* Numero elementi allocati nel vettore anag */
int index = 0;
int menPer(void);
int insPer(int);
struct per * cerPer(char *, char *, int);
void eliPer(struct per *);
void canPer(void);
void ricPer(void);
void visPer(struct per *);
void visAnagrafe(void);

/* Presenta il menu e lancia la funzione scelta */
int main()
{
int scelta = MENU;
while(scelta!=OUT) {
  switch(scelta) {
    case MENU:
      scelta = menPer();
      if(scelta == MENU)
        scelta = OUT;
      break;
    case INS:
      index = insPer(index);
      scelta = MENU;
      break;
    case CAN:
      canPer();
      scelta = MENU;
      break;
    case RIC:
      ricPer();
      scelta = MENU;
      break;
    case VIS:
      visAnagrafe();
      scelta = MENU;
      break;}
}
}


/* Menu */
int menPer(void)
{
int scelta;
char invio;
int vero = 1;
while(vero){
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  printf("\t\t\t ANAGRAFE");
  printf("\n\n\n\t\t\t 1. Immissione Persona");
```

```
  printf("\n\n\t\t\t 2. Cancellazione Persona");
  printf("\n\n\t\t\t 3. Ricerca Persona");
  printf("\n\n\t\t\t 4. Visualizza anagrafe");
  printf("\n\n\t\t\t 0. Fine");
  printf("\n\n\n\t\t\t\t Scegliere una opzione: ");
  scanf("%d", &scelta);
  scanf("%c", &invio);
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  switch(scelta) {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
      return(scelta);
    default:
      break;
}
}
return(0);
}


/* Inserisce persona nell'anagrafe */
int insPer(int pos)
{
char invio;
if(pos>=MAXELE) {
  printf(" Non si possono inserire altri nomi");
  scanf("%c", &invio);
  return(pos);
}
printf("\n\t\tINSERIMENTO PERSONA");
printf("\n\t\t---------------\n\n");
printf("\nCognome : ");
gets(anag[pos].cognome);
printf("\nNome : ");
gets(anag[pos].nome);
printf("\nIndirizzo : ");
gets(anag[pos].ind);
printf("\nEta' : ");
scanf("%d", &anag[pos].eta);
scanf("%c", &invio);
pos++;
return(pos);
}


/* Cancella persona dall'anagrafe, se presente */
void canPer(void)
{
char pausa;
char cognome[DIM], nome[DIM];
int eta;
struct per *ps;

printf("\n\t\tCANCELLA PERSONA");
printf("\n\t\t------\n\n");
printf("\nCognome : ");
gets(cognome);
printf("\nNome : ");
gets(nome);
printf("\nEta' : ");
```

```
scanf("%d", &eta);
scanf("%c", &pausa);

/* invoca ricerca persona */
ps = cerPer(cognome, nome, eta);
if(ps == NULL) {
  printf("\nPersona non presente in anagrafe");
  scanf("%c", &pausa);
  return;
}

/* Invoca visualizza persona */
visPer(ps);
printf("\nConfermi cancellazione? (S/N) ");
scanf("%c", &pausa);
if(pausa=='S' || pausa=='s') {
  eliPer(ps);
  return;
}
}


/* Elimina persona dall'anagrafe */
void eliPer(struct per *p)
{
strcpy(p->cognome, "");
strcpy(p->nome, "");
strcpy(p->ind, "");
p->eta = 0;
}


/* Ricerca persona se presente nell'anagrafe */
void ricPer(void)
{
char pausa;
char cognome[DIM], nome[DIM];
int eta;
struct per *ps;

/* Inserimento dati persona da ricercare */
printf("\n\t\tRICERCA PERSONA");
printf("\n\t\t------\n\n");
printf("\nCognome : ");
gets(cognome);
printf("\nNome : ");
gets(nome);
printf("\nEta' : ");
scanf("%d", &eta);
scanf("%c", &pausa);

/* invoca ricerca persona */
ps = cerPer(cognome, nome, eta);
if(ps == NULL) {
  printf("\nPersona non presente in anagrafe");
  scanf("%c", &pausa);
  return;
}
visPer(ps);
scanf("%c", &pausa);
}
```

```
/* Scansione sequenziale del vettore anag alla ricerca di una
   persona che abbia determinati cognome, nome ed età         */
struct per *cerPer(char *cg, char *nm, int et)
{
int i;
for(i=0; i<=index; i++) {
  if(strcmp(cg, anag[i].cognome) == 0)
    if(strcmp(nm, anag[i].nome) == 0)
      if(et == anag[i].eta)
        return(&anag[i]);
}
return(NULL);
}


/* Visualizza persona */
void visPer(struct per *p)
{
printf("\n\n--------------------\n");
printf("\n\t\tCognome : %s", p->cognome);
printf("\n\t\tNome : %s", p->nome);
printf("\n\t\tIndirizzo : %s", p->ind);
printf("\n\t\tEta' : %d", p->eta);
printf("\n\n--------------------\n");
}


/* Visualizza l'anagrafe completa */
void visAnagrafe(void)
{
int i; char pausa;
struct per *ps = &anag[0];
for (i=0; i<index; i++) {
  visPer(ps++);
  scanf("%c", &pausa);
}
}
```

Capitolo 17

Listato 17.1 *Conteggio del numero di caratteri nel file* `clienti`.

```
/* Determina il numero di caratteri di un file esistente */

#include <stdio.h>

main()
{
char buf[100];    /* buffer per la lettura */
FILE *fp;         /* file pointer */
long nc;          /* contatore caratteri */
int n;            /* numero caratteri letti con fread() */
int fineFile =0;  /* indica la fine della lettura del file */

fp = fopen("clienti", "r"); /* apertura del file clienti */

if(fp == NULL)
/* Si è verificato un errore: il file non esiste */
  printf("Errore : il file ordini non esiste\n");
```

```
  else {
    nc = 0L; /* inizializza il contatore */
    do {/* ciclo di lettura */
        /* Legge 100 caratteri dal file ordini */
        n = fread(buf, 1, 100, fp);
        if(n==0) /* controllo di fine file */
          fineFile = 1;
        nc += n; /* incremento del contatore */
    }
    while(fineFile==0);

    fclose(fp); /* chiusura del file clienti */
    printf("Il file clienti contiene %ld caratteri\n", nc);
}
}
```

Listato 17.2 *Programma per l'acquisizione di una stringa da tastiera e sua scrittura in un file.*

```
/* Scrittura di una stringa in un file */

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

main()
{
char buf[100];    /* buffer */
FILE *fp;         /* file pointer */
int len;

/* Legge da tastiera il nome del fornitore */
printf("Inserisci un fornitore : ");
scanf("%s",buf);
len = strlen(buf);
fp = fopen("fornitori", "w"); /* crea il file fornitori */

/* Memorizza il nome del fornitore nel file */
fwrite(buf, 1, len, fp);
fclose(fp);                   /* chiude il file */
}
```

Listato 17.3 *Programma per la copia di un file su un altro.*

```
/* Copia di un file su un altro */

#include <stdio.h>

main()
{
FILE *fpin, *fpout;     /* file pointer */
char buf[512];          /* buffer dati */
int n;

fpin = fopen("ordini","r");           /* apre ordini in lettura */
if(fpin!=NULL) {
  fpout = fopen("ordini.bak", "w");   /*crea ordini.bak */
    if(fpout!=NULL) {  /* ordini.bak creato correttamente? */
      for(;;) {              /* copia ordini in ordini.bak */
        n = fread(buf, 1, 512, fpin);    /* legge ordini */
        if(n == 0) break;                /* controllo di fine file */
        fwrite(buf, 1, n, fpout);        /* scrive in ordini.bak */
    }
```

```
    fclose(fpin);      /* chiude il file ordini */
    fclose(fpout);     /* chiude il file ordini.bak */
  }
  else {
    printf("Impossibile aprire il file ordini.bak\n");
    fclose(fpin);      /* chiude il file ordini */
  }
}
else
/* Errore di apertura */
printf("Il file ordini non esiste\n");
}
```

Listato 17.4 *Visualizzazione della dimensione di un file con* `fseek` *e* `ftell`.

```
/* Determinazione del numero di caratteri di un file
   con fseek e ftell                                    */
#include <stdio.h>

main(int argc, char **argv)
{
FILE *fp;
long n;

if(argc < 2)
  printf("File non specificato\n");
else {
  fp = fopen(argv[1], "r");  /* apertura del file */

  if(fp != NULL) {           /* il file esiste? */
    fseek(fp,0L,2);          /* puntatore alla fine del file */
    n = ftell(fp);           /* lettura posizione del puntatore */
    fclose(fp);              /* chiusura del file */
    printf("Dimensione del file %ld\n", n);
  }
  else
    printf("Errore : il file %s non esiste\n", argv[1]);
}
}
```

Listato 17.5 *Programma che conta il numero di righe di un file.*

```
/* Determinazione del numero di linee contenute in un file.
   Ogni linea è definita dal carattere di newline \n        */

#include <stdio.h>

main(int argc, char **argv)
{
char buf[100];
int linee;
FILE *fp;

if( argc < 2 )
  printf("Errato numero di parametri\n");
else {
  fp = fopen(argv[1], "r");        /* apre il file */
  if(fp!= NULL) {                  /* il file esiste? */
    linee = 0;                     /* inizializza contatore di linea */
    for(;;) {                      /* ciclo di lettura da file */
      if( fgets(buf,100,fp) == NULL )
```

```
        break;                      /* fine file */
      linee++;                      /* incrementa contatore linee */
    }
    fclose(fp);                     /* chiude il file */
    printf("Il file contiene %d linee\n", linee);
  }
  else
    printf("Il file %s non esiste\n",argv[1]);
}
}
```

Listato 17.6 *Programma che conta il numero di caratteri numerici contenuti in un file.*

```
/* Determinazione del numero di caratteri numerici
   (cifre decimali) presenti in un file */

#include <stdio.h>

main(int argc, char **argv)
{
FILE *fp;
int c;
int nc;

if(argc < 2)
  printf("Errato numero di parametri\n");
else {
  fp = fopen(argv[1],"r");          /* apre il file */
  if(fp!=NULL) {                    /* il file esiste? */
    nc = 0; /* inizializza il contatore */
    while((c = fgetc(fp)) != EOF)   /* ciclo di lettura */
      if(c>='0' && c<='9') nc++;    /* incrementa il contatore */
    fclose(fp);                     /* chiude il file */
    printf("Numero di caratteri numerici: %d\n", nc);
  }
  else
  printf("Il file %s non esiste\n",argv[1]);
}
}
```

Listato 17.8 *Programma che copia il contenuto di un file in un altro utilizzando i file descriptor.*

```
/* Copia il contenuto di un file in un altro */

#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

main(argc,argv)
int argc;
char **argv;
{
static char buf[BUFSIZ];
int fdin, fdout, n;
if(argc != 3) {
  printf("Devi specificare file sorgente e destinazione\n");
  exit(1);
}
```

```
/* Apre il file sorgente */
fdin = open(argv[1],O_RDONLY);
if(fdin < 0) {
  printf("Non posso aprire %s\n",argv[1]);
  exit(2);
}

/* Apre il file destinazione */
fdout = open(argv[2],O_WRONLY|O_CREAT|O_TRUNC,0600);
if(fdout < 0) {
  printf("Non posso creare %s\n", argv[2]);
  close(fdin);
  exit(3);
}

/* Esegue il ciclo di lettura e scrittura */
for(;;) {
  /* Legge BUFSIZ caratteri da file */
  n = read(fdin, buf, BUFSIZ);

  /* Controlla la fine del file */
  if(n == 0)
    break;

  /* Scrive i caratteri nel file destinazione */
  write(fdout,buf,n);
}

/* Chiude i file */
close(fdin);
close(fdout);
}
```

Listato 17.9 *Programma che memorizza all'interno di un file informazioni su un gruppo di alunni inserite da tastiera.*

```
/* Memorizza in un file le informazioni passate
   dall'utente sugli alunni di una classe          */

#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

/* La struttura alunno contiene nome, cognome ed età di ogni alunno */

struct alunno {
  char nome[100];
  char cognome[100];
  int eta;
};

main()
{
struct alunno alunno;
int nalunni;
int fd;

/* Apre il file alunni */
fd = open("alunni",O_WRONLY|O_CREAT|O_TRUNC,0600);
if(fd < 0) {
  printf("Non posso aprire il file alunni\n");
```

```
  exit(1);
}
printf("Quanti alunni vuoi inserire ? ");
scanf("%d",&nalunni);

while(nalunni-- > 0) {
  printf("Nome : ");
  scanf("%s",alunno.nome);
  printf("Cognome : ");
  scanf("%s",alunno.cognome);
  printf("Eta': ");
  scanf("%d",&alunno.eta);
  write(fd, &alunno, sizeof(struct alunno));
}
close(fd);
}
```

Caso di Studio IV
Gestione anagrafica con i file

Listato del Caso di studio IV *Gestione anagrafi ca su file.*

```
#include <stdio.h>
#include <string.h>
#define DIM 31
#define MENU 0
#define INS 1
#define CAN 2
#define RIC 3
#define VIS 4
#define OUT 100

/* Semplice struttura che modella una persona */
struct per {
  char cognome[DIM];
  char nome[DIM];
  char ind[DIM];
  int eta;
};

/* Puntatore al file */
FILE *fp;

/* La variabile di appoggio anag per le operazioni sul file */
struct per anag;

int menPer(void);
void insPer(void);
void ricPer(void);
void canPer(void);
long cerPer(char *, char *, int);
void eliPer(long pos);
void visPer(void);
void visAnagrafe(void);

/* Presenta il menu e lancia la funzione scelta */
int main()
{
int scelta = MENU;
```

```
while(scelta!=OUT) {
  switch(scelta) {
    case MENU:
      scelta = menPer();
      if(scelta == MENU)
        scelta = OUT;
      break;
    case INS:
      insPer();
      scelta = MENU;
      break;
    case CAN:
      canPer();
      scelta = MENU;
      break;
    case RIC:
      ricPer();
      scelta = MENU;
      break;
    case VIS:
      visAnagrafe();
      scelta = MENU;
      break;
  }
}
}


/* Menu */
int menPer(void)
{
int scelta;
char invio;
int vero = 1;

while(vero){
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  printf("\t\t\t ANAGRAFE");
  printf("\n\n\n\t\t\t 1. Immissione Persona");
  printf("\n\n\t\t\t 2. Cancellazione Persona");
  printf("\n\n\t\t\t 3. Ricerca Persona");
  printf("\n\n\t\t\t 4. Visualizza anagrafe");
  printf("\n\n\t\t\t 0. Fine");
  printf("\n\n\n\t\t\t\t Scegliere una opzione: ");
  scanf("%d", &scelta);
  scanf("%c", &invio);
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  switch(scelta) {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
      return(scelta);
    default:
      break;
  }
}
return(0);
}


/* inserisce persona nel file anag.dat */
```

```
void insPer(void)
{
char invio;
/* Se il file non esiste lo crea,
   i dati sono appesi in fondo al file */
fp = fopen("anag.dat","a+");

printf("\n\t\tINSERIMENTO PERSONA");
printf("\n\t\t---------------\n\n");
printf("\nCognome : ");
gets(anag.cognome);
printf("\nNome : ");
gets(anag.nome);
printf("\nIndirizzo : ");

gets(anag.ind);
printf("\nEta' : ");
scanf("%d", &anag.eta);
scanf("%c", &invio);

fwrite(&anag, sizeof(struct per), 1, fp);
fclose(fp);
}


/* Cancella una persona dall'anagrafe, se presente */
void canPer(void)
{
char pausa;
char cognome[DIM], nome[DIM];
int eta;
long pos;
printf("\n\t\tCANCELLA PERSONA");
printf("\n\t\t------\n\n");
printf("\nCognome : ");
gets(cognome);
printf("\nNome : ");
gets(nome);
printf("\nEta' : ");
scanf("%d", &eta);
scanf("%c", &pausa);
/* Invoca ricerca persona */
pos = cerPer(cognome, nome, eta);
if(pos == -1) {
  printf("\nPersona non presente in anagrafe");
  scanf("%c", &pausa);
  return;
}

/* Invoca visualizza persona */
visPer();
printf("\nConfermi cancellazione ? (S/N) ");
scanf("%c", &pausa);
if(pausa=='S' || pausa=='s') {
  eliPer(pos);
  return;
}
}


/* Elimina persona dall'anagrafe */
void eliPer(long pos)
{
```

```
strcpy(anag.cognome, "");
strcpy(anag.nome, "");
strcpy(anag.ind, "");
anag.eta = 0;
fp = fopen("anag.dat","r+");
fseek(fp,pos,0);
fwrite(&anag, sizeof(struct per), 1, fp);
fclose(fp);
}


/* Ricerca persona */
void ricPer(void)
{
char pausa;
char cognome[DIM], nome[DIM];
int eta;
long pos;
/* Inserimento dati persona da ricercare */
printf("\n\t\tRICERCA PERSONA");
printf("\n\t\t------\n\n");
printf("\nCognome : ");
gets(cognome);
printf("\nNome : ");
gets(nome);
printf("\nEta' : ");
scanf("%d", &eta);
scanf("%c", &pausa);
/* Invoca la funzione di scansione sequenziale */
pos = cerPer(cognome, nome, eta);
if(pos == -1) {
  printf("\nPersona non presente in anagrafe");
  scanf("%c", &pausa);
  return;
}
visPer();
scanf("%c", &pausa);
}


/* Effettua una scansione sequenziale del file anag.dat
   alla ricerca di una persona che abbia determinati
   cognome, nome ed età                                  */
long cerPer(char *cg, char *nm, int et)
{
int n;
long pos = 0L;

fp = fopen("anag.dat", "r");
for(;;) {
  n = fread(&anag, sizeof(struct per), 1, fp);
  if(n==0) {
    fclose(fp);
    pos = -1;
    return (pos);
  }
  else
    if(strcmp(cg, anag.cognome) == 0)
      if(strcmp(nm, anag.nome) == 0)
      if(et == anag.eta) {
        pos = ftell(fp);
        fclose(fp);
        return(pos-sizeof(struct per));
```

```
      }
  }
}


/* Visualizza persona */
void visPer(void)
{
printf("\n\n--------------------\n");
printf("\n\t\tCognome : %s", anag.cognome);
printf("\n\t\tNome : %s", anag.nome);
printf("\n\t\tIndirizzo : %s", anag.ind);
printf("\n\t\tEta' : %d", anag.eta);
printf("\n\n--------------------\n");
}


/* Visualizza l'anagrafe completa */
void visAnagrafe(void)
{
int n; char pausa;
fp = fopen("anag.dat", "r");
do {
    n = fread(&anag, sizeof(struct per), 1, fp);
    if(n==0) fclose(fp);
    else {
      visPer();
      scanf("%c", &pausa);
    }
}
while(n!=0);
}
```

Capitolo 18

Listato 18.1 *Visualizzazione della dimensione dei tipi fondamentali.*

```
#include <stdio.h>
main()
{
  int ch, in, sh, lo, fl, dd, ld;
  ch = sizeof(char);
  in = sizeof(int);
  sh = sizeof(short);
  lo = sizeof(long);
  fl = sizeof(float);
  dd = sizeof(double);
  ld = sizeof(long double);

  printf("La dimensione di un char e'           %d\n", ch);
  printf("La dimensione di uno short e'         %d\n", sh);
  printf("La dimensione di un int e'            %d\n", in);
  printf("La dimensione di un long e'           %d\n", lo);
  printf("La dimensione di un float e'          %d\n", fl);
  printf("La dimensione di un double e'         %d\n", dd);
  printf("La dimensione di un long double e'    %d\n", ld);
}
```

Capitolo 19

Listato 19.1 *Copia su stringa.*

```
#include <stdio.h>
int i = 80;
double d = 3.14546;
main()
{
  int numUsc;
  char bufUsc[81];
  numUsc = sprintf(bufUsc, "Il valore di i = %d e \
  il valore di d = %g\n", i, d);

  printf("sprintf() ha scritto %d caratteri e il \
  buffer contiene:\n%s", numUsc, bufUsc);
}
```

Listato 19.2 *Esempio di uso di* `sscanf`.

```
#include <stdio.h>
main()
{
  double valore;
  char buf[31], nome[31];
  printf("Inserire una variabile nel formato\
  \"nome = <valore>\":");
  gets(buf);

  /* Con sscanf() si separano il nome dal valore */
  sscanf(buf, " %[^=] = %lf", nome, &valore);
  printf("La variabile %s vale %f\n", nome, valore);
}
```

Listato 19.3 *Esempio di funzione che accetta un numero di argomenti variabile.*

```
#include <stdio.h>
#include <stdarg.h>

int somma(int, ...);

main()
{
int a=3, b=100, c=321, d=400, e=711;
printf("\n%d %d %d %d %d \n", a, b, c, d, e);
printf("%d ", somma(2, a, b));
printf("%d ", somma(3, a, b, c));
printf("%d ", somma(4, a, b, c, d));
printf("%d ", somma(5, a, b, c, d, e));
}

int somma(int argNum, ...)
{
int totale=0;
va_list ap;
va_start(ap, argNum);
```

```
while(argNum--)
  totale=totale + va_arg(ap, int);
va_end(ap);
return(totale);
}
```

Capitolo 20

Listato 20.1 *Puntatori di funzione per la creazione di un menu.*

```
#include <stdio.h>

void fun1(void), fun2(void), fun3(void);

struct voceMenu {
  char *msg;            /* prompt di voce di menu */
  void (*fun)(void);    /* funzione da innescare */
};

/* Inizializza il vettore di strutture menu
   assegnando il messaggio della voce di menu
   e la relativa funzione                       */

struct voceMenu menu[] = {
  "1. Funzione fun1\n", fun1,
  "2. Funzione fun2\n", fun2,
  "3. Funzione fun3\n", fun3,
  "0. Fine\n", NULL, NULL , NULL
};

void main()
{
int scelta;
struct voceMenu *p;
int loop = 0;

while(loop==0) {
  for(p=menu; p->msg!=NULL; p++) /* presentazione del menu */
    printf("%s", p->msg);
  printf("\n\nScegliere l'opzione desiderata: ");
  scanf("%d", &scelta);
  if(scelta==0)                 /* uscita programma */
    loop = 1;
  else
  /* esecuzione della funzione associata alla scelta */
  (*menu[scelta-1].fun)();
}
}

void fun1(void)
{printf("\n\n Sto eseguendo fun1\n\n\n");}

void fun2(void)
{printf("\n\n Sto eseguendo fun2\n\n\n");}

void fun3(void)
{printf("\n\n Sto eseguendo fun3\n\n\n");}
```

Listato 20.2 *Un array di puntatori scandito da un puntatore di puntatore.*

```
#include <stdio.h>

char *menu[] = {
  "1. Voce di menu 1\n",
  "2. Voce di menu 2\n",
  "3. Voce di menu 3\n",
  "...\n"
  "N. Voce di menu N\n",
  NULL
};

char **ppc = menu;

main()
{
char *pausa;
while(*ppc!=NULL)
  printf("%s", *ppc++);
gets(pausa);
}
```

Listato 20.3 *Variabili locali e globali.*

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

#define DIM 31
#define TAPPO "THE END"

/* Semplice struttura che modella una persona */
struct per {
  char cognome[DIM];
  char nome[DIM];
  char ind[DIM];
  int eta;
};

/* vettore di persone */
struct per anag[] = {
  {"Edison", "Thomas", "Vicolo della Lampadina, 8", 30},
  {"Alighieri", "Dante", "Via del Purgatorio, 13", 21},
  {"More", "Thomas", "Viale Utopia, 48", 39},
  {TAPPO, TAPPO, TAPPO, 0}
};

void visPer(void);

main()
{
  visPer();
}


void visPer(void)
{
char pausa; int i;

for(i=0; strcmp(anag[i].cognome, TAPPO)!=0; i++) {
  printf("\n\n--------------------\n");
```

```
  printf("\n\t\tCognome : %s", anag[i].cognome);
  printf("\n\t\tNome : %s", anag[i].nome);
  printf("\n\t\tIndirizzo : %s", anag[i].ind);
  printf("\n\t\tEta' : %d", anag[i].eta);
  printf("\n\n--------------------\n");
  scanf("%c", &pausa);
}
}
```

Listato 20.4 *File* `MAIN.C`

```
/* Programma principale: MAIN.C */
extern void visPer(void);

main()
{
  visPer();
}
```

Listato 20.5 *File* `VIS_PER.C`

```
/* File delle funzioni: VIS_PER.C */

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

#define DIM 31
#define TAPPO "THE END"

/* Semplice struttura che modella una persona */
struct per {
  char cognome[DIM];
  char nome[DIM];
  char ind[DIM];
  int eta;
};

/* Vettore di persone */
struct per anag[] = {
  {"Edison", "Thomas", "Vicolo della Lampadina, 8", 30},
  {"Alighieri", "Dante", "Via del Purgatorio, 13", 21},
  {"More", "Thomas", "Viale Utopia, 48", 39},
  {TAPPO, TAPPO, TAPPO, 0}
};

void visPer(void)
{
char pausa; int i;

for(i=0; strcmp(anag[i].cognome, TAPPO)!=0; i++) {
  printf("\n\n--------------------\n");
  printf("\n\t\tCognome : %s", anag[i].cognome);
  printf("\n\t\tNome : %s", anag[i].nome);
  printf("\n\t\tIndirizzo : %s", anag[i].ind);
  printf("\n\t\tEta' : %d", anag[i].eta);
  printf("\n\n--------------------\n");

  scanf("%c", &pausa);
}
}
```

Listato 20.6 *Esempio di variabili* `static`

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

#define DIM 31
#define TAPPO "X_Y_Z"

struct per {
  char cognome[DIM];
  char nome[DIM];
  char ind[DIM];
  int eta;
};

/* Vettore di persone */
static struct per anag[] = {
  {"Edison", "Thomas", "Vicolo della Lampadina, 8", 30},
  {"Alighieri", "Dante", "Via del Purgatorio, 13", 21},
  {"More", "Thomas", "Viale Utopia, 48", 39},
  {TAPPO, TAPPO, TAPPO, 0}
};

void visPer(void)
{
char pausa; int i;

for(i=0; strcmp(anag[i].cognome, TAPPO)!=0; i++) {
  printf("\n\n--------------------\n");
  printf("\n\t\tCognome : %s", anag[i].cognome);
  printf("\n\t\tNome : %s", anag[i].nome);
  printf("\n\t\tIndirizzo : %s", anag[i].ind);
  printf("\n\t\tEta' : %d", anag[i].eta);
  printf("\n\n--------------------\n");

  scanf("%c", &pausa);
}
}
```

Listato 20.7 *Un generatore di numeri casuali.*

```
#include <stdio.h>

#define FATTORE 25173
#define MODULO 65535
#define INCREMENTO 13849
#define SEME_INIZIALE 8
#define LOOP 10

unsigned rand(void);

void main()
{
int i;
for(i=0; i<LOOP; i++)
  printf("Il numero casuale %d e' %6u\n\n", i+1, rand());
}

unsigned rand(void)
```

```
{
static unsigned seme = SEME_INIZIALE;
seme = (FATTORE*seme+INCREMENTO) % MODULO;
return(seme);
}
```

Capitolo 21

Listato 21.1 *Creazione e visualizzazione di una lista.*

```
/* Accetta in ingresso una sequenza di interi e li
   memorizza in una lista. Il numero di interi che
   compongono la sequenza è richiesto all'utente.
   La lista creata viene visualizzata                */

#include <stdio.h>
#include <malloc.h>

/* Struttura degli elementi della lista */
struct elemento {
  int inf;
  struct elemento *pun;
};

struct elemento *creaLista();
void visualizzaLista(struct elemento *);

main()
{
struct elemento *puntLista;  /* puntatore alla testa
                                della lista */
puntLista = creaLista();     /* chiamata funzione per
                                creare la lista */
visualizzaLista(puntLista);  /* chiamata funzione per
                                visualizzare la lista */
}


/* Funzione per l'accettazione dei valori immessi
   e la creazione della lista. Restituisce il puntatore
   alla testa                                          */
struct elemento *creaLista()
{
struct elemento *p, *paus;
int i, n;

printf("\n Da quanti elementi e' composta la sequenza? ");

scanf("%d", &n);
if(n==0) p = NULL;            /* lista vuota */
else
{
  /* Creazione del primo elemento */
  p = (struct elemento *)malloc(sizeof(struct elemento));
  printf("\nInserisci la 1a informazione: ");
  scanf("%d", &p->inf);
  paus = p;

  /* Creazione degli elementi successivi */
```

```
  for(i=2; i<=n; i++) {
    paus->pun = (struct elemento *)malloc(sizeof(struct elemento));
    paus = paus->pun;
    printf("\nInserisci la %da informazione: ", i);
    scanf("%d", &paus->inf);
  }
paus->pun = NULL; /* marca di fine lista */
}
return(p);
}


/* Funzione per la visualizzazione della lista.
   Il parametro in ingresso è il puntatore alla testa */
void visualizzaLista(struct elemento *p)
{
printf("\npuntLista---> ");

/* Ciclo di scansione della lista */
while(p!=NULL) {
  printf("%d", p->inf); /* visualizza il campo informazione */
  printf("---> ");
  p = p->pun; /* scorri di un elemento in avanti */
}
printf("NULL\n\n");
}
```

Listato 21.2 *Programma che crea, visualizza la lista e ne determina il maggiore. L'immissione dei valori da parte dell'utente termina con zero.*

```
/* Accetta in ingresso una sequenza di interi e li
   memorizza in una lista. La sequenza termina quando
   viene immesso il valore zero. La lista creata viene
   visualizzata. Determina il maggiore della lista      */

#include <stdio.h>
#include <malloc.h>
#include <limits.h>

struct elemento {
  int inf;
  struct elemento *pun;
};

struct elemento *creaLista2();
void visualizzaLista(struct elemento *);
int maggioreLista(struct elemento *);

main()
{
struct elemento *puntLista;   /* puntatore alla testa
                                 della lista */
puntLista = creaLista2();     /* chiamata funzione per
                                 creare la lista */
visualizzaLista(puntLista);   /* chiamata funzione per
                                 visualizzare la lista */
/* Stampa il valore di ritorno della funzione
maggioreLista() */
printf("\nIl maggiore e': %d\n\n", maggioreLista(puntLista));
}
```

```
/* Accetta in ingresso una sequenza di interi e li
   memorizza in una lista. Il numero di interi che
   compongono la sequenza termina con il valore zero */
struct elemento *creaLista2()
{
struct elemento *p, *paus;
struct elemento x;

printf("\nInserisci un'informazione (0 per fine lista): ");
scanf("%d", &x.inf);

if(x.inf==0) p = NULL; /* lista vuota */
else {
  /* Creazione del primo elemento */
  p = (struct elemento *)malloc(sizeof(struct elemento));

  p->inf = x.inf;
  paus=p;
  while(x.inf!=0) {
    printf("\nInserisci un'informazione (0 per fine lista): ");
    scanf("%d", &x.inf);

    if(x.inf!=0) {

      /* Creazione dell'elemento successivo */
      paus->pun = (struct elemento *)malloc(sizeof(struct elemento));

      paus = paus->pun;              /* attualizzazione di paus */
      paus->inf = x.inf;             /* inserimento dell'informazione
                                        nell'elemento */
    }
    else
      paus->pun = NULL; /* Marca di fine lista */
  }
}
return(p);
}


/* Determina il maggiore della lista.
   Il parametro in ingresso è il puntatore alla testa */
maggioreLista(struct elemento *p)
{
int max = INT_MIN;

/* Ciclo di scansione della lista */
while(p != NULL) {
  if(p->inf > max)
    max = p->inf;
  p = p->pun; /* scorre di un elemento in avanti */
}
return(max);
}


/* Visualizza la lista */
void visualizzaLista(struct elemento *p)
{
printf("\npuntLista---> ");

/* Ciclo di scansione della lista */
while(p!=NULL) {
```

```
  printf("%d", p->inf);       /* visualizza il campo
                                 informazione */
  printf("---> ");
  p = p->pun; /* scorre di un elemento in avanti */
}
printf("NULL\n\n");
}
```

Listato 21.3 *Somma dei valori di due liste.*

```
/* Crea due liste e le visualizza. Addiziona gli elementi
   corrispondenti delle due liste, inserisce il risultato
   in una terza lista e la visualizza                      */

#include <stdio.h>
#include <malloc.h>

struct elemento {
  int inf;
  struct elemento *pun;
};

struct elemento *creaLista2();
void visualizzaLista(struct elemento *);
struct elemento *sommaListe(struct elemento *,struct elemento *);

main()
{
struct elemento *puntLista1, *puntLista2, *puntLista3;

printf("\n PRIMA LISTA \n");
puntLista1 = creaLista2();

printf("\n SECONDA LISTA \n");
puntLista2 = creaLista2();

visualizzaLista(puntLista1);
visualizzaLista(puntLista2);

/* Invocazione della funzione per la somma delle liste */
funpuntLista3 = sommaListe(puntLista1, puntLista2);
/* Visualizzazione della lista somma delle precedenti */
visualizzaLista(puntLista3);
}


/* Addiziona gli elementi corrispondenti di due liste
   e inserisce il risultato in una terza lista          */
struct elemento *sommaListe(struct elemento *p1, struct elemento *p2)
{
struct elemento *p3 = NULL, *p3aus;

if(p1!=NULL && p2!=NULL) {
  /* Creazione primo elemento */
  p3 = (struct elemento *)malloc(sizeof(struct elemento));
  p3->inf = p1->inf + p2->inf; /* somma */
  p3aus = p3; /* p3aus punta III lista */
  p1 = p1->pun; /* scorrimento I lista */
  p2 = p2->pun; /* scorrimento II lista */

  /* Creazione elementi successivi */
```

```
  for(; p1!=NULL && p2!=NULL;) {
    p3aus->pun = (struct elemento *)malloc(sizeof(struct elemento));
    p3aus = p3aus->pun; /* scorrimento III lista */
    p3aus->inf = p1->inf + p2->inf; /* somma */
    p1 = p1->pun; /* scorrimento I lista */
    p2 = p2->pun; /* scorrimento II lista */
  }
p3aus->pun = NULL; /* marca di fine III lista */
}
return(p3); /* ritorno del puntatore alla III lista */
}
/* ATTENZIONE: devono essere aggiunte le definizioni delle
   funzioni per creare – creaLista2() – e visualizzare –
   visualizzaLista() – la lista, presenti nel precedente
   Listato 21.2 */
```

Listato 21.4 *Gestione di una pila implementata mediante un array*

```
/* GESTIONE DI UNA PILA
   Operazioni di inserimento, eliminazione
   e visualizzazione. Utilizza un array di strutture
   per implementare la pila                              */

#include <stdio.h>
#include <malloc.h>

#define LUN_PILA 10

void gestionePila(void);

inserimento(int *, int *, int);
eliminazione(int *, int *, int *);
pilaVuota(int);
void visualizzazionePila(int *, int);

main()
{
  gestionePila();
}


void gestionePila(void)
{
int pila[LUN_PILA];
int puntTesta = 0;
int scelta = -1, ele;
char pausa;

while(scelta!=0) {
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  printf("\t\tESEMPIO UTILIZZO STRUTTURA ASTRATTA: PILA");
  printf("\n\n\n\t\t\t 1. Per inserire un elemento");
  printf("\n\n\t\t\t 2. Per eliminare un elemento");
  printf("\n\n\t\t\t 3. Per visualizzare la pila");
  printf("\n\n\t\t\t 0. Per finire");
  printf("\n\n\n\t\t\t\t Scegliere una opzione: ");
  scanf("%d", &scelta);
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

  switch(scelta) {
    case 1:
      if(puntTesta >= LUN_PILA) {
```

```
        printf("Inserimento impossibile: ");
        printf("memoria disponibile terminata");
        printf("\n\n Premere un tasto per continuare...");
        scanf("%c%c", &pausa, &pausa);
      }
      else {
        printf("Inserire un elemento: ");
        scanf("%d", &ele);
        puntTesta = inserimento(pila, &puntTesta, ele);
      }
      break;
    case 2:
      if(pilaVuota(puntTesta)) {
        printf("Eliminazione impossibile: pila vuota");
        printf("\n\n Premere un tasto per continuare...");
        scanf("%c%c", &pausa, &pausa);
      }
      else {
        puntTesta = eliminazione(pila, &puntTesta, &ele);
        printf("Eliminato: %d", ele );
        printf("\n\n Premere un tasto per continuare...");
        scanf("%c%c", &pausa, &pausa);
      }
      break;
    case 3:
      visualizzazionePila(pila, puntTesta);
      printf("\n\n Premere un tasto per continuare...");
      scanf("%c%c", &pausa, &pausa);
      break;
  }
}
}


void visualizzazionePila(int *pila, int p)
{
printf("\n<------ Testa della pila ");
while(p>=1)
  printf("\n%d", pila[--p]);
}


inserimento(int *pila, int *p, int ele)
{
pila[*p] = ele;
++*p;
return(*p);
}


eliminazione(int *pila, int *p, int *ele)
{
--*p;
*ele = pila[*p];
return(*p);
}


int pilaVuota(int p)
{
if(p==0)
  return(1);
else
```

```
  return(0);
}
```

Listato 21.5 *Gestione di una pila implementata mediante una lista.*

```
/* GESTIONE DI UNA PILA
   Operazioni di inserimento, eliminazione
   e visualizzazione. Utilizza una lista lineare
   per implementare la pila                        */

#include <stdio.h>
#include <malloc.h>

struct elemento {
  int inf;
  struct elemento *pun;
};

void gestionePila(void);
struct elemento *inserimento(struct elemento *, int ele);
struct elemento *eliminazione(struct elemento *, int *);
int pilaVuota(struct elemento *);
void visualizzazionePila(struct elemento *);

main()
{
  gestionePila();
}


void gestionePila(void)
{
struct elemento *puntTesta = NULL;
int scelta = -1, ele;
char pausa;
while(scelta!=0) {
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  printf("\t\tESEMPIO UTILIZZO STRUTTURA ASTRATTA: PILA");
  printf("\n\n\n\t\t\t 1. Per inserire un elemento");
  printf("\n\n\t\t\t 2. Per eliminare un elemento");
  printf("\n\n\t\t\t 3. Per visualizzare la pila");
  printf("\n\n\t\t\t 0. Per finire");
  printf("\n\n\n\t\t\t\t Scegliere una opzione: ");
  scanf("%d", &scelta);
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  switch(scelta) {
    case 1:
      printf("Inserire un elemento: ");
      scanf("%d", &ele);
      puntTesta = inserimento(puntTesta, ele);
      if(puntTesta==NULL) {
        printf("Inserimento impossibile: ");
        printf("memoria disponibile terminata");
        printf("\n\n Premere un tasto per continuare...");
        scanf("%c%c", &pausa, &pausa);
      }
      break;
    case 2:
      if(pilaVuota(puntTesta)) {
        printf("Eliminazione impossibile: pila vuota");
        printf("\n\n Premere un tasto per continuare...");
        scanf("%c%c", &pausa, &pausa);
```

```
      }
      else {
        puntTesta = eliminazione(puntTesta, &ele);
        printf("Eliminato: %d", ele );
        printf("\n\n Premere un tasto per continuare...");
        scanf("%c%c", &pausa, &pausa);
      }
      break;
    case 3:
      visualizzazionePila(puntTesta);
      printf("\n\n Premere un tasto per continuare...");
      scanf("%c%c", &pausa, &pausa);
      break;
  }
}
}


void visualizzazionePila(struct elemento *p)
{
struct elemento *paus = p;

printf("\n<------ Testa della pila ");
while(paus!=NULL) {
  printf("\n%d", paus->inf);
  paus = paus->pun;
}
}


struct elemento *inserimento(struct elemento *p, int ele)
{
struct elemento *paus;

paus = (struct elemento *)malloc(sizeof(struct elemento));

if(paus==NULL) return(NULL);

paus->pun = p;
p = paus;
p->inf = ele;
return(p);
}


struct elemento *eliminazione(struct elemento *p, int *ele)
{
struct elemento *paus;

*ele = p->inf;
paus = p;
p = p->pun;
free(paus);
return( p );
}


int pilaVuota(struct elemento *p)
{
if(p==NULL)
  return(1);
else
  return(0);
```

```
}
```

Caso di Studio IV
Gestione anagrafica con i file

Listato Caso di studio V *Gestione di una lista ordinata.*

```
/* GESTIONE DI LISTA ORDINATA
   Operazioni di inserimento, eliminazione e visualizzazione.
   Utilizza una lista lineare                                  */

#include <stdio.h>
#include<malloc.h>

struct elemento {
  int inf;
  struct elemento *pun;
};

void gestioneLista(void);

struct elemento *inserimento(struct elemento *);
struct elemento *eliminazione(struct elemento *);
void visualizzazione(struct elemento *);

main()
{
  gestioneLista();
}


void gestioneLista(void)
{
struct elemento *puntLista = NULL;
int scelta = -1;
char pausa;

while(scelta!=0) {
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  printf("\t\tGESTIONE DI UNA SEQUENZA DI VALORI ORDINATI\n");
  printf("\t\t MEDIANTE UNA STRUTTURA A LISTA");
  printf("\n\n\n\t\t\t 1. Per inserire un elemento");
  printf("\n\n\t\t\t 2. Per eliminare un elemento");
  printf("\n\n\t\t\t 3. Per visualizzare la lista");
  printf("\n\n\t\t\t 0. Per finire");
  printf("\n\n\n\t\t\t\t Scegliere una opzione: ");
  scanf("%d", &scelta);
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

  switch(scelta) {
    case 1:
      puntLista = inserimento(puntLista);
      break;
    case 2:
      puntLista = eliminazione(puntLista);
      break;
    case 3:
```

```
      visualizzazione(puntLista);
      printf("\n\nPremere un tasto per continuare...");
      scanf("%c%c", &pausa, &pausa);
      break;
  }
}
}


/* Visualizzazione della lista */
void visualizzazione(struct elemento *p)
{
struct elemento *paus = p;

printf("\npuntLista---> ");
while(paus!=NULL) {
  printf("%d---> ", paus->inf);
  paus = paus->pun;
}
printf("NULL");
}


/* Inserimento del valore passato dall'utente nella lista
   mantenendo l'ordinamento                                  */
struct elemento *inserimento(struct elemento *p)
{
struct elemento *p0, *p1;
int posizione;

/* Creazione elemento */
p0 = (struct elemento *)malloc(sizeof(struct elemento));

printf("\nInserire l'informazione (un numero intero): ");
scanf("%d", &p0->inf);

if(p==NULL) { /* se la lista è vuota, l'elemento */
  p = p0; /* diventa il primo e unico della lista */
  p->pun = NULL;
}
else {
  if(p->inf > p0->inf) { /* se il valore dell'elemento è */
    p0->pun = p; /* inferiore al primo l'elemento */
    p = p0; /* diventa il primo della lista */
  }
  else { /* ricerca della posizione di inserimento */
    p1 = p;
    posizione = 0;
    while(p1->pun!=NULL && posizione!=1) {
      if(p1->pun->inf < p0->inf)
        p1 = p1->pun; /* scorre in avanti p1 */
      else
        posizione = 1; /* interrompe lo scorrimento */
    }
    p0->pun = p1->pun; /* connessione all'elemento successivo */
    p1->pun = p0; /* connessione dall'elemento precedente */
  }
}
return(p); /* ritorno del puntatore all'inizio della lista */
}


/* Eliminazione dell'elemento richiesto dalla lista ordinata */
```

```
struct elemento *eliminazione(struct elemento *p)
{
struct elemento *p1 = p, *p2;
struct elemento e;
int posizione = 0;
char pausa;

printf("\nInserire l'informazione da eliminare: ");
scanf("%d", &e.inf);

if(p1!=NULL) { /* se la lista è vuota fine */
  if(p1->inf == e.inf) { /* se è il primo da eliminare */
    p2 = p1;
    p = p->pun; /* si modifica il puntatore alla testa */
    free(p2);
    return(p);
  }
  else { /* ricerca dell'elemento da eliminare */
    while(p1->pun!=NULL && posizione!=1) {
      if(p1->pun->inf!=e.inf)
        p1 = p1->pun; /* scorre in avanti p1 */
      else {
        posizione = 1; /* interrompe lo scorrimento */
        p2 = p1->pun;
        p1->pun = p1->pun->pun; /* eliminazione elemento */
        free(p2); /* libera la memoria */
        return( p );
      }
    }
  }
}
if(!posizione) {
  printf("\nElemento non incontrato nella lista ");
  scanf("%c%c", &pausa, &pausa);
}
return(p);
}
```

Capitolo 22

Listato 22.1 *Creazione e visita in ordine anticipato di un albero binario.*

```
/* Creazione di un albero binario e visita in ordine
   anticipato. L'etichetta dei nodi è un valore intero,
   le occorrenze multiple dello stesso valore non
   vengono memorizzate                                  */

#include <stdio.h>
#include <malloc.h>

struct nodo {
  int inf;
  struct nodo *albSin;
  struct nodo *albDes;
};

struct nodo *albBin(void);
struct nodo *creaNodo(struct nodo *, int);
void anticipato(struct nodo *);
```

```
main()
{
struct nodo *radice; /* puntatore alla radice dell'albero */

radice = albBin(); /* invoca la funzione per la creazione dell'albero binario */
printf("\nVISITA IN ORDINE ANTICIPATO\n");
anticipato(radice);
}


/* Crea l'albero binario. Per ogni etichetta immessa
   dall'utente, invoca la funzione creaNodo.
   Ritorna al chiamante la radice dell'albero         */

struct nodo *albBin(void)
{
struct nodo *p = NULL;
struct nodo x;

do {
  printf("\nInserire una informazione (0 per finire): ");
  scanf("%d", &x.inf);
  if(x.inf!=0)
    p = creaNodo(p, x.inf); /* invoca creaNodo() */
}
while(x.inf!=0);

return(p); /* ritorna la radice */
}


/* Visita ricorsivamente l'albero alla ricerca del punto
   di inserimento. Quando trova la posizione, crea un
   nodo, vi inserisce l'etichetta e ritorna il puntatore
   a tale nodo.
   Parametri in ingresso:
       p e' il puntatore alla radice
       val e' l'etichetta da inserire nel nodo            */

struct nodo *creaNodo(struct nodo *p, int val)
{
if(p==NULL) { /* il punto di inserimento e' stato reperito */
  /* Creazione del nodo */
  p = (struct nodo *) malloc(sizeof(struct nodo));
  p->inf = val; /* inserimento di val in elemento */
  p->albSin = NULL; /* marca di albero sinistro vuoto */
  p->albDes = NULL; /* marca di albero destro vuoto */
}
else { /* ricerca del punto di inserimento */
  if(val > p->inf)
    /* Visita il sottoalbero destro */
    p->albDes = creaNodo(p->albDes, val);
  else
    if(val < p->inf)
    /* Visita il sottoalbero sinistro */
    p->albSin = creaNodo(p->albSin, val);
}
return(p); /* ritorna il puntatore alla radice */
}


/* Visita l'albero binario in ordine anticipato */
```

```
void anticipato(struct nodo *p)
{
if(p!=NULL) {
  printf("%d ", p->inf); /* visita la radice */
  anticipato(p->albSin); /* visita il sottoalbero sinistro */
  anticipato(p->albDes); /* visita il sottoalbero destro */
}
}
```

Listato 22.2 *Visita in ordine simmetrico e ricerca di un valore nell'albero binario.*

```
/* DA AGGIUNGERE ALLE DICHIARAZIONI INIZIALI
   DEL LISTATO 22.1 */

void simmetrico(struct nodo *);
void ricerca(struct nodo *, int, struct nodo **);


/* DA AGGIUNGERE AL MAIN DEL LISTATO 22.1 */

struct nodo *trovato;
int chi;
...
printf("\nVISITA IN ORDINE SIMMETRICO\n");
simmetrico(radice);

printf("\nInserire il valore da ricercare: ");
scanf("%d", &chi);
printf("\nRICERCA COMPLETA");
trovato = NULL;
ricerca(radice, chi, &trovato);
if(trovato != NULL)
  printf("\n Elemento %d presente \n", trovato->inf);
else
  printf("\n Elemento non presente\n");

/* Funzione che visita l'albero binario in ordine simmetrico.
   DA AGGIUNGERE AL LISTATO 22.1 */

void simmetrico(struct nodo *p)
{
if(p!=NULL) {
  simmetrico(p->albSin);
  printf("%d ", p->inf);
  simmetrico(p->albDes);
}
}


/* Funzione che ricerca un'etichetta nell'albero binario.
   DA AGGIUNGERE AL LISTATO 22.1.
   Visita l'albero in ordine anticipato */

void ricerca(struct nodo *p, int val, struct nodo **pEle)
{
if(p!=NULL)
  if(val == p->inf) /* La ricerca ha dato esito positivo */
    *pEle = p; /* pEle è passato per indirizzo per cui l'assegnamento di p
                  avviene sul parametro attuale */
  else {
    ricerca(p->albSin, val, pEle);
```

```
    ricerca(p->albDes, val, pEle);
}
}
```

Listato 22.3 *Ricerca di un valore nell'albero binario*

```
/* Ricerca ottimizzata */
void ricBin(struct nodo *p, int val, struct nodo **pEle)
{
if(p!=NULL)
  if(val == p->inf) {
    printf(" trovato ");
    *pEle = p;
  }
else
  if(val < p->inf) {
    printf(" sinistra");
    ricBin(p->albSin, val, pEle);
  }
else {
  printf(" destra");
  ricBin(p->albDes, val, pEle);
  }
}
```

Listato 22.4 *Creazione di un albero a partire dalla sua rappresentazione parentetica.*

```
/* Creazione di un albero e visita in ordine anticipato.
   L'albero viene immesso dall'utente informa parentetica
   anticipata. L'etichetta dei nodi è un carattere.
   L'albero è implementato con liste multiple            */

#include <stdio.h>
#include<stddef.h>

struct nodo {
  char inf;
  struct nodo *figlio;
  struct nodo *pArco;
};

struct nodo *albero();
struct nodo *creaAlbero(struct nodo *);
void anticipato(struct nodo *);

main()
{
struct nodo *radice;
radice = albero();          /* creazione dell'albero */
printf("\nVISITA IN ORDINE ANTICIPATO\n");
anticipato(radice);
}


/* Legge il primo carattere della rappresentazione
   parentetica e invoca la funzione creaAlbero() */
```

```
struct nodo *albero()
{
struct nodo *p = NULL;
char car;
printf("\nInserire la rappresentazione dell'albero: ");
scanf("%c", &car);
p = creaAlbero(p);
return(p); /* ritorna il puntatore alla radice al chiamante */
}


/* Crea un nodo e vi inserisce la relativa etichetta.
   Per ogni figlio crea un arco. Invoca ricorsivamente
   se stessa                                              */

struct nodo *creaAlbero(struct nodo *p)
{
struct nodo *paus;
char car;
/* Crea il nodo */
p = (struct nodo *) malloc( sizeof(struct nodo) );
scanf("%c", &p->inf); /* inserimento del valore nel nodo */
paus = p;
scanf("%c", &car); /* lettura del carattere successivo */

while(car=='(') { /* per ogni figlio ripeti */
  /* Crea l'arco */
  paus->pArco = (struct nodo *) malloc(sizeof(struct nodo));
  paus = paus->pArco;
  /* Invoca se stessa passando il campo figlio dell'elem. creato */
  paus->figlio = creaAlbero(paus->figlio);
  scanf("%c", &car); /* lettura del carattere successivo */
}

paus->pArco = NULL;
return(p); /* ritorna il puntatore alla radice al chiamante */
}

/* Visita ricorsivamente l'albero in ordine anticipato */
void anticipato(struct nodo *p)
{
printf("(%c", p->inf);
p = p->pArco;
while(p!=NULL) {
  anticipato(p->figlio);
  p = p->pArco;
}
printf(")");
}
```

Listato 22.5 *Ricerca di un nodo e visualizzazione del sottoalbero che si diparte da esso.*

```
/* DA AGGIUNGERE AL LISTATO 22.4 /*

...
struct nodo *trovato;
char sottoAlbero, invio;
...

scanf("%c", &invio);
printf("\nInserire la radice del sottoalbero: ");
```

```
scanf("%c", &sottoAlbero);

trovato = NULL;
ricerca(radice, sottoAlbero, &trovato);
if(trovato!=NULL) {
  printf("\n SOTTOALBERO IN ORDINE ANTICIPATO \n");
  anticipato(trovato);
}
else
  printf("\n Sottoalbero non presente");
}


/* Visita in ordine anticipato, ricercando il sottoalbero
   con radice sa. Se lo reperisce assegna il suo
   indirizzo a *puntSa                                    */

ricerca(struct nodo *p, char sa, struct nodo **puntSa)
{
if(sa == p->inf)
  *puntSa = p;
else {
  p = p->pArco;
  while(p!=NULL) {
    ricerca(p->figlio, sa, puntSa);
      p = p->pArco;
  }
}
}
```

Listato 22.6 *Creazione di un albero binario corrispondente all'albero ordinato (non binario) immesso in forma parentetica dall'utente.*

```
/* Dalla rappresentazione parentetica di un albero ricava
   il corrispondente albero binario, che visita in ordine
   simmetrico, anticipato e differito.
   Per la creazione usa una funzione iterativa (non
   ricorsiva) con l'ausilio di una pila gestita mediante
   una lista lineare il cui campo inf e' un puntatore ai
   nodi dell'albero binario. Per le visite in ordine
   simmetrico, anticipato e differito rimangono valide
   le funzioni ricorsive già esaminate                    */

#include <stdio.h>
#include <stddef.h>

struct nodo { /* nodo dell'albero binario */
  char inf;
  struct nodo *albSin;
  struct nodo *albDes;
};

struct nodo *albBinPar(); /* funzione per la creazione */

void anticipato(struct nodo *); /* visite */
void simmetrico(struct nodo *);
void differito(struct nodo *);

struct elemento { /* elemento della lista lineare */
struct nodo *inf; /* con cui e' implementata la pila */
struct elemento *pun;
};
```

```
/* Funzioni per la gestione della pila */
struct elemento *inserimento(struct elemento *, struct nodo *);
struct elemento *eliminazione(struct elemento *, struct nodo **);
int pilaVuota(struct elemento *);

main()
{
struct nodo *radice;

radice = albBinPar();

printf("\nVISITA IN ORDINE SIMMETRICO\n");
simmetrico(radice);
printf("\nVISITA IN ORDINE ANTICIPATO\n");
anticipato(radice);
printf("\nVISITA IN ORDINE DIFFERITO\n");
differito(radice);
}


/* Dalla rappresentazione parentetica di un albero crea
   il corrispondente albero binario */

struct nodo *albBinPar()
{
struct nodo *p;
struct nodo *paus, *pp;
char car;
int logica = 1;

struct elemento *puntTesta = NULL; /* inizializzazione pila */

printf("\nInserire la rappresentazione dell'albero: ");
scanf("%c", &car);

/* Creazione della radice */
p = (struct nodo *) malloc(sizeof(struct nodo));

scanf("%c", &p->inf);
p->albSin = NULL; /* inizializzazione dei puntatori */
p->albDes = NULL; /* ai sottoalberi */

paus = p; logica = 1;

do {
  scanf("%c", &car);
  if(car=='(') {
    pp = (struct nodo *) malloc(sizeof(struct nodo));
    scanf("%c", &pp->inf);
    pp->albSin = NULL;
    pp->albDes = NULL;
    if(logica) {
      paus->albSin = pp;
      /* Inserimento in pila */
      puntTesta = inserimento(puntTesta, paus);
    }
    else {
      paus->albDes = pp;
      logica = 1;
    }
    paus = pp;
  }
```

```
  else
  if(logica)
    logica = 0;
  else
    /* Eliminazione dalla pila */
    puntTesta = eliminazione(puntTesta, &paus);
}
while(!pilaVuota(puntTesta));

return(p);
}


/* Funzioni per la gestione della pila */

struct elemento *inserimento(struct elemento *p, struct nodo *ele)
{
struct elemento *paus;

paus = (struct elemento *)malloc(sizeof(struct elemento));
if(paus==NULL) return(NULL);

paus->pun = p;
p = paus;
p->inf = ele;
return(p);
}


struct elemento *eliminazione(struct elemento *p, struct nodo **ele)
{
struct elemento *paus;

*ele = p->inf;
paus = p;
p = p->pun;
free(paus);

return(p);
}


int pilaVuota(struct elemento *p)
{
if(p ==NULL)
  return(1);
else
  return(0);
}


/* DI SEGUITO SONO INCLUSE LE FUNZIONI RICORSIVE DI VISITA
   DELL'ALBERO BINARIO GIÀ ESAMINATE, DOVE IL VALORE
   VISUALIZZATO È DI TIPO CHAR /*
...
```

Listato 22.7 *Rappresentazione di grafi .*

```
/* Trasformazione della rappresentazione di un grafo da
   una matrice di adiacenze a una lista di successori   */
```

```
#include <stdio.h>
#include <malloc.h>

struct nodo { /* struttura di un nodo */
  char inf;
  struct successore *pun;
};

struct successore { /* elemento della lista di successori */
  int inf;
  struct successore *pun;
};

int a[10][10]; /* matrice di adiacenze */

struct nodo s[10]; /* array di nodi */
int n; /* numero di nodi */

void matAdiacenze(void);
void visMatAdiacenze(void);
void successori(void);
void creaSucc(int, int);
void visita(void);

main()
{
matAdiacenze();     /* creazione della matrice di adiacenze */
visMatAdiacenze();  /* visualizzazione della matrice */
successori();       /* creazione delle liste di successori */
visita();           /* visual. dei successori di ogni nodo */
}


/* Crea la matrice di adiacenze */

void matAdiacenze(void)
{
int i, j;
char invio;

printf("\nNumero di nodi: ");
scanf("%d", &n);
scanf("%c", &invio);

for(i=0; i<n; i++) { /* richiesta etichette dei nodi */
  printf("\nEtichetta del nodo: ");
  scanf("%c", &s[i].inf);
  scanf("%c", &invio);
  s[i].pun = NULL;
}

for(i=0; i<n; i++) /* richiesta archi orientati */
  for(j=0; j<n; j++) {
    printf("\nArco orientato da [%c] a [%c] (0 no, 1 si) ? ",
    s[i].inf, s[j].inf);
    scanf("%d", &a[i][j]);
  }
}


/* Visualizza la matrice di adiacenze */

void visMatAdiacenze(void)
```

```
{
int i, j;

printf("\nMATRICE DI ADIACENZE\n");
for(i=0; i<n; i++) /* visualizza i nodi (colonne) */
  printf(" %c", s[i].inf);
for(i=0; i<n; i++) {
  printf("\n%c ", s[i].inf); /* visualizza i nodi (righe) */
  for(j=0; j<n; j++)
    printf("%d ", a[i][j]); /* visualizza gli archi */
}
}


/* Crea le liste di successori. Per ogni arco rappresentato
   nella matrice di adiacenze chiama creaSucc */

void successori(void)
{
int i, j;

for(i=0; i<n; i++)
  for(j=0; j<n; j++) {
    if(a[i][j]==1)
      creaSucc(i, j);
}
}


/* Dato un certo arco nella matrice di adiacenze, crea
   il rispettivo elemento di lista */

void creaSucc( int i, int j )
{
struct successore *p;

if(s[i].pun==NULL) { /* non esiste la lista dei successori */
  s[i].pun = (struct successore *)(malloc(sizeof(struct successore)));
  s[i].pun->inf = j;
  s[i].pun->pun = NULL;
}
else { /* esiste la lista dei successori */
  p = s[i].pun;
  while(p->pun!=NULL)
    p = p->pun;
  p->pun = (struct successore *)(malloc(sizeof(struct successore)));
  p = p->pun;
  p->inf = j;
  p->pun = NULL;
}
}


/* Per ogni nodo del grafo restituisce i suoi successori.
   Lavora sulle liste di successori                        */

void visita(void)
{
int i;
struct successore *p;

printf("\n");
```

```
for(i=0; i<n; i++) {
  printf("\n[%c] ha come successori: ", s[i].inf);
  p = s[i].pun;
  while(p!=NULL) {
    printf(" %c", s[p->inf].inf);
    p = p->pun;
  }
}
}
```

Capitolo 23

Listato 23.1 *Utilizzo di una macro con argomenti di tipo diverso.*

```
#include <stdio.h>

#define MAGGIORE(a, b) ((a) > (b) ? (a) : (b))

int i, j;
double x, y;
char c1, c2;

main()
{
i=88; j=55;
x=83.54; y=96.2;
c1='c'; c2='z';
printf("Maggiore: %d\n", MAGGIORE(i, j));
printf("Maggiore: %f\n", MAGGIORE(x, y));
printf("Maggiore: %c\n", MAGGIORE(c1, c2));
}
```

Listato 23.2 *Utilizzo delle direttive per la compilazione condizionale.*

```
#include <stdio.h>
#define EU 1
#define UK 2
#define US 3

#define AREA EU

main()
{
#if AREA==US
  char moneta[]="dollaro";
#elif AREA==EU
  char moneta[]="euro";
#else
  char moneta[]="sterlina";
#endif
printf("Moneta corrente: %s\n", moneta);
}
```

Listato 23.3 *Esempio di utilizzo di* `#if` *annidate.*

```
#include <stdio.h>
```

```
#define EU 1
#define US 2

#define ITALIA 10
#define UK 20

#define AREA EU
#define PAESE ITALIA

main()
{
#if AREA==EU
  #if PAESE == ITALIA
    char moneta[]="euro";
  #elif PAESE == UK
    char moneta[]="sterlina";
  #endif
#else
  char moneta[]="dollaro";
#endif
printf("Moneta corrente: %s\n", moneta);
}
```

Listato 23.4 *Esempio di uso dell'operatore del preprocessore* ##.

```
#include <stdio.h>
#define GARE(a) gara##a
main()
{
  char garaPrima[] = "Stile libero";
  char garaSeconda[] = "Rana";
  char garaTerza[] = "Dorso";

  printf("%s\n", GARE(Prima));
  printf("%s\n", GARE(Seconda));
  printf("%s\n", GARE(Terza));
}
```

Capitolo 24

Listato 24.1 *Nel programma sono evidenziati i punti di separazione.*

```
/* M.C.D. Euclideo */

#include <stdio.h>

int main ()
{
  int x;
  int y;
  // Stato 0

  scanf("%i%i", &x,&y);
  // Stato Iniziale

  while(x != y)
```

```
  {
    // Stato Transitorio
    if(x > y)
    {
      x = x - y;
    }
    else
    {
      y = y - x;
    }
  }

// Stato Finale
printf("%d %d", x, y);
}
```

Listato 24.2 *Porzione di programma C su cui applicheremo il metodo della specifi ca.*

```
...
int x, y, max;
x = X;
y = Y;
max = 0;
/* Supponiamo che X e Y siano
   definiti da qualche #define
*/

if(x>y)
{
  max = x;
}
else
{
  max = y;
}
...
```

Listato 24.3 *Il metodo della specifi ca applicata al programma del Listato 24.2.*

```
...
int x, y, max;

x = X;
<x' = X ∧ y'=y ∧ max' = max>
y = Y;
<x' = x ∧ y'= Y ∧ max' = max
ovvero
x' = X ∧ y'= Y ∧ max' = max>
max = 0;
<x' = x ∧ y'= y ∧ max' = 0
ovvero
x' = X ∧ y'= Y ∧ max' = 0>

/* Supponiamo che X e Y siano
   definiti da qualche #define
*/

if(x>y)
```

```
{
  max = x;
}
else
{
max = y;
}
< max = x ovvero X, x > y,
max = y ovvero Y, x ≤ y>
...
```

Caso di Studio VI
Progetto per la gestione aziendale della fatturazione

```
typedef struct
{
  char PIVA[16];
  stringa RagioneSociale;
  stringa Indirizzo;
  stringa Citta;
  char Provincia[3];
} Cliente;

typedef struct
{
  int CodArt;
  stringa Descrizione;
  stringa Tipologia;
} Articolo;

typedef struct
{
  int CodFatt;
  char Data[11];
  stringa TipoPagamento;
  int IVA;
  char PIVA[16];
} Fattura;

typedef struct
{
  int CodFatt;
  int NRiga;
  int CodArt;
  int Quantita;
  float PrezzoUnitario;
} RigaFattura;



void main()
{
int scelta;
char invio;
int numfatture;
int numarticoli;
boolean risultato = vero;
```

```
do {
  numfatture = ContaFatture();
  numarticoli = ContaArticoli();
  VisualizzaMenu();
  // acquisizione scelta utente
  scanf("%d", &scelta);
  scanf("%c", &invio);
  // interpretazione scelta ed esecuzione della
  // relativa operazione
  switch (scelta) {

    case 0:
      printf("Uscita dal programma\n");
      break;

    case 1:
      risultato = InserisciArticolo(++numarticoli);
      break;

    case 2:
      risultato = LeggiArticolo();
      break;

    case 3:
      risultato = InserisciCliente();
      break;

    case 4:
      risultato = EsportaClienti();
      break;

    case 5:
      risultato = InserisciFattura(++numfatture, numarticoli);
      break;

    case 6:
      risultato = LeggiFattura();
      break;

    default:
      printf("la scelta non e' corretta,
    inserirne un'altra\n");
  }
} while(scelta != 0 && risultato);
}


void VisualizzaMenu(void)
{
  printf("\nMenu' Principale\nScegli l'operazione da eseguire: \n");
  printf("1. Inserisci articolo\n");
  printf("2. Visualizza tutti gli articoli\n");
  printf("3. Inserisci cliente\n");
  printf("4. Salva clienti su file txt\n");
  printf("5. Inserisci fattura\n");
  printf("6. Visualizza fatture di un cliente\n");
  printf("0. Termina il programma\n");
}



int ContaArticoli(void)
{
```

```
  int nArt;
  FILE *fArt;

  fArt = fopen(ARTICOLI_NAME, "rb");
  if (fArt == NULL) {
    return 0;
  }
  else {
    // posizionamento del cursore a fine file
    fseek(fArt, 0, SEEK_END);
    // calcolo del numero di articoli presenti nel file
    nArt = ftell(fArt)/sizeof(Articolo);
    fclose(fArt);
    return nArt;
  }
}



int InserisciArticolo(int codArt)
{
  Articolo art;
  FILE *fArt;
  char invio;

  printf("\n\nInserimento nuovo articolo\n\n");

  art.CodArt = codArt;
  printf("CODICE ARTICOLO: %d\n", art.CodArt);
  printf("DESCRIZIONE (max 20 char): ");
  gets(art.Descrizione);

  printf("TIPOLOGIA (max 20 char): ");
  gets(art.Tipologia);

  fArt = fopen(ARTICOLI_NAME, "ab+");
  if (fArt == NULL) {
    printf("Errore di apertura file");
    return falso;
  }
  // inserimento di un nuovo record Articolo in coda al file
  fwrite(&art, sizeof(Articolo), 1, fArt);

  fclose(fArt);
  return vero;
}



int LeggiArticolo(void)
{
  Articolo art;
  FILE *fArt;
  printf("\n\nELENCO ARTICOLI\n\n");
  printf("CODICE \tDESCRIZIONE\tTIPOLOGIA\n");

  fArt = fopen(ARTICOLI_NAME, "rb");
  if (fArt == NULL) {
    printf("Errore di apertura file");
    return falso;
  }
  // lettura sequenziale del file degli articoli
  // un record per volta
```

```
  while (fread(&art, sizeof(Articolo), 1, fArt)>0)
  {
    printf("%5d %20s %20s\n", art.CodArt, art.Descrizione,
    art.Tipologia);
  }
  fclose(fArt);
  printf("Premi ENTER per tornare al Menu' Principale");
  getchar();
  return vero;
}



int InserisciCliente(void)
{
  Cliente cli;
  FILE *fCli;

  printf("\n\nInserimento nuovo cliente\n\n");

  printf("PARTITA IVA: ");
  gets(cli.PIVA);

  printf("RAGIONE SOCIALE (max 20 char): ");
  gets(cli.RagioneSociale);

  printf("INDIRIZZO (max 20 char): ");
  gets(cli.Indirizzo);

  printf("CITTA' (max 20 char): ");
  gets(cli.Citta);

  printf("PROVINCIA (2 char): ");
  gets(cli.Provincia);

  fCli = fopen(CLIENTI_NAME, "ab+");
  if (fCli == NULL) {
    printf("Errore di apertura file");
    return falso;
  }
// inserimento di un nuovo record Cliente in coda al file
fwrite(&cli, sizeof(Cliente), 1, fCli);

fclose(fCli);
return vero;
}



int EsportaClienti(void)
{
  Cliente cli;
  FILE *fCli;
  FILE *fOut;
  stringa fName;
  char pr[3];
  int i= 0;

  printf("\n\nEsporta clienti su file di testo\n\n");

  printf("Nome file di output (max 20 char): ");
  gets(fName);
```

```
  printf("Scegli la PROVINCIA (2 char, * per tutte): ");
  gets(pr);

  fCli = fopen(CLIENTI_NAME, "rb");
  if (fCli == NULL) {
    printf("Errore di apertura file");
    return falso;
  }

  fOut = fopen(fName, "wt");
  if (fOut == NULL) {
    printf("Errore di apertura file");
    return falso;
  }

  // lettura sequenziale del file degli clienti un record per volta
  while (fread(&cli, sizeof(Cliente), 1, fCli)>0)
  {
    // verifica della congruita' tra provincia corrente
    // e quella richiesta dall'utente
    if (strcmp(cli.Provincia, pr) == 0 || strcmp("*", pr) == 0)
    {
      fprintf(fOut, "PARTITA IVA: %s\n", cli.PIVA);
      fprintf(fOut, "RAGIONE SOCIALE: %s\n",
      cli.RagioneSociale);
      fprintf(fOut, "INDIRIZZO: %s\n", cli.Indirizzo);
      fprintf(fOut, "CITTA': %s\n", cli.Citta);
      fprintf(fOut, "PROVINCIA: %s\n\n\n", cli.Provincia);
      // incremento del contatore del numero di clienti
      // esportati
      i++;
    }
  }
  printf("Clienti esportati su file: %d\n", i);
  printf("Premi ENTER per tornare al Menu' Principale");
  getchar();
  fclose(fCli);

fclose(fOut);
return vero;
}



int ContaFatture(void)
{
  int nFat;
  FILE *fFat;
  fFat = fopen(FATTURE_NAME, "rb");
  if (fFat == NULL) {
    return 0;
  }
  else {
    // posizionamento del cursore a fine file
    fseek(fFat, 0, SEEK_END);
    // calcolo del numero di articoli presenti nel file
    nFat = ftell(fFat)/sizeof(Fattura);
    fclose(fFat);
    return nFat;
  }
}
```

```
int InserisciFattura(int codeFat, int numart)
{
  Fattura fat;
  RigaFattura rfat;
  char continua;
  int nriga = 1;

  FILE *fFat;
  FILE *fRFat;

  char invio;

  printf("\n\nInserimento intestazione nuova fattura\n\n");

  fFat = fopen(FATTURE_NAME, "ab+");
  if (fFat == NULL) {
    printf("Errore di apertura file");
    return falso;
  }

  fat.CodFatt = codeFat;

  printf("CODICE FATTURA (progressivo): %d\n", fat.CodFatt);

  printf("DATA FATTURA (max 10 char): ");
  gets(fat.Data);

  printf("TIPO DI PAGAMENTO (max 20 char): ");
  gets(fat.TipoPagamento);

  printf("PARTITA IVA CLIENTE (max 16 char): ");
  gets(fat.PIVA);

  printf("VALORE IVA: ");
  scanf("%d", &fat.IVA);
  scanf("%c", &invio);

  // inserimento di un nuovo record Fattura in coda al file
  fwrite(&fat, sizeof(Fattura), 1, fFat);

  fclose(fFat);

  fRFat = fopen(RIGHEFATT_NAME, "ab+");
  if (fRFat == NULL) {
    printf("Errore di apertura file");
    return falso;
  }

  do {
    printf("\nInserisci una nuova riga di fattura:\n");
    rfat.CodFatt = fat.CodFatt;
    rfat.NRiga = nriga;
    do {
      printf("CODICE ARTICOLO (da 1 a %d): ", numart);
      scanf("%d", &rfat.CodArt);
      scanf("%c", &invio);
    } while (rfat.CodArt > numart && rfat.CodArt < 0);
    // verifica della presenza nell'archivio
    // del codice articolo immesso

    printf("QUANTITA': ");
    scanf("%d", &rfat.Quantita);
```

```
    scanf("%c", &invio);

    printf("PREZZO UNITARIO: ");
    scanf("%f", &rfat.PrezzoUnitario);
    scanf("%c", &invio);

    // inserimento di un nuovo record RigaFattura
    // in coda al file
    fwrite(&rfat, sizeof(RigaFattura), 1, fRFat);
    // incremento del contatore di riga
    nriga++;

    printf("\nVuoi inserire una nuova riga di fattura ('c' per continuare)? ");
    scanf("%c", &continua);
  } while (continua == 'c');

  fclose(fRFat);
  return vero;
}



Articolo TrovaArticolo(int numArt)
{
  Articolo art;
  FILE *fArt;
  fArt = fopen(ARTICOLI_NAME, "rb");
  if (fArt == NULL) {
    printf("Errore di apertura file");
  }
  // posizionamento del cursone all'inizio del record
  // in posizione numArt
  fseek(fArt, (numArt-1) * sizeof(Articolo), SEEK_SET);
  fread(&art, sizeof(Articolo), 1, fArt);
  return art;
}



int LeggiFattura(void)
{
  Cliente cli;
  Fattura fat;
  RigaFattura rfat;
  Articolo art;

  FILE *fCli;
  FILE *fFat;
  FILE *fRFat;

  stringa RagSoc;
  float imponibile = 0;
  int contaFatture = 0;

  boolean trovato = falso;

  printf("\n\nRicerca fatture cliente\n\n");

  printf("Inserire RAGIONE SOCIALE (max 20 char): ");
  gets(RagSoc);

  fCli = fopen(CLIENTI_NAME, "rb");
  if (fCli == NULL) {
```

```
    printf("Errore di apertura file");
    return falso;
  }

  // ricerca della presenza del cliente indicato dall'utente
  while (fread(&cli, sizeof(Cliente), 1, fCli)>0)
  {
    if (strcmp(cli.RagioneSociale, RagSoc) == 0)
    {
      trovato = vero;
      break;
    }
  }

  fclose(fCli);

  if (!trovato) {
    printf("Il cliente non e' presente\n");
    printf("Premi ENTER per tornare al Menu' Principale");
    getchar();
    return vero;
  }

  fFat = fopen(FATTURE_NAME, "rb");
  if (fFat == NULL) {
    printf("Errore di apertura file");
    return falso;
  }

  printf("Elenco Fatture\n\n");
  // scansione sequenziale del file delle fatture
  while (fread(&fat, sizeof(Fattura), 1, fFat)>0)
  {
    // verifica che la fattura appartenga al cliente
    // indicato dall'utente
    if (strcmp(cli.PIVA, fat.PIVA) == 0)
    {
      contaFatture++;
      if (contaFatture > 1) {
        printf("\n\nPremi ENTER per continuare");
        getchar();
      }
      // visualizzazione intestazione fattura
      printf("\n\nFATTURA (%d)\n", contaFatture);
      printf("Numero:\t %d\t\t\t\t", fat.CodFatt);
      printf("Data: %10s\n\n", fat.Data);
      printf("Cliente\t %s\n", cli.RagioneSociale);
      printf("\t %s\n", cli.Indirizzo);
      printf("\t %s (%s)\n", cli.Citta, cli.Provincia);
      printf("P. IVA\t %s\n\n", fat.PIVA);
      fRFat = fopen(RIGHEFATT_NAME, "rb");

      if (fRFat == NULL) {
        printf("Errore di apertura file");
        return falso;
      }
      printf("\nCod. e Desc. Art.\t\tQuantita'\t Pezzo Unitario\tPrezzo
Totale\n");
      // scansione sequenziale del file delle righe di fattura
      while (fread(&rfat, sizeof(RigaFattura), 1, fRFat)>0)
      {
        // verifica che la riga fattura appartenga alla
        // fattura corrente
```

```
        if (fat.CodFatt == rfat.CodFatt)
        {
          // visualizzazione singola riga di fattura
          printf("%5d ", rfat.CodArt);
          art = TrovaArticolo(rfat.CodArt);
          printf("%20s\t", art.Descrizione);
          printf("%5d\t\t", rfat.Quantita);
          printf("%6.2f\t\t", rfat.PrezzoUnitario);
          printf("%6.2f\n",
          rfat.Quantita * rfat.PrezzoUnitario);
          imponibile = imponibile + rfat.Quantita * rfat.PrezzoUnitario;
        }
        else {
          // interruzione della lettura quanto una riga
          // fattura e' riferita ad una fattura con
          // codice superiore alla fattura corrente
          if (rfat.CodFatt > fat.CodFatt)
            break;
        }
      }
      fclose(fRFat);

      printf("\n\t\t\t\t\t\tImponibile\t%6.2f\n", imponibile);
      printf("\t\t\t\t\t\tIVA (%d%%)\t\t%6.2f\n", fat.IVA, imponibile * fat.IVA
/ 100);
      printf("\t\t\t\t\t\tTOTALE\t\t%6.2f\n", imponibile + imponibile * fat.IVA
/ 100);
    }
  }
  fclose(fFat);
  printf("\n\nPremi ENTER per tornare al Menu' Principale");
  getchar();
  return vero;
}
```