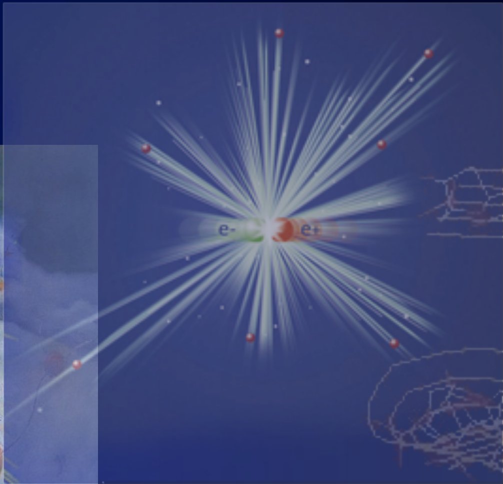
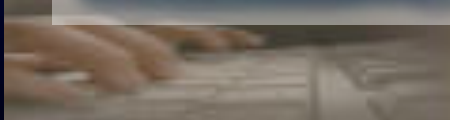
Corso di Informatica A.A. 2009- 2010

Lezione 14



Input/output da file



File buffering

Quando si termina di utilizzare un file è buona norma chiuderlo utilizzando la funzione `fclose()`

```
int fclose(FILE *fp);
```

`fp` rappresenta il puntatore a FILE associato allo stream.

`fclose()` ritorna 0 in caso di successo -1 altrimenti

Quando si chiude un file il buffer associato viene svuotato

In generale quando un programma ha termine tutti gli streams sono svuotati automaticamente e chiusi.

File buffering

Quando si crea uno stream associato ad un file, automaticamente un buffer viene creato ed associato allo stream

Un buffer è un blocco di memoria utilizzato per l'immagazzinamento temporaneo dei dati che devono essere scritti o letti da file

L'utilizzo dei buffers è legato al fatto che le operazioni su disco sono più efficienti quando i dati sono scritti o letti in blocchi di una certa dimensione

Quando un programma invia dei dati ad uno stream i dati sono salvati in un buffer sino a quando questo è pieno e solo allora il suo intero contenuto è scritto sul disco

File buffering

In pratica quindi durante l'esecuzione è possibile che i dati "scritti" dal programma siano ancora nel buffer e non sul disco.
In caso di problemi questi dati vanno quindi perduti.

E' però possibile forzare lo svuotamento di un buffer mediante il comando `fflush()`

```
int fflush(FILE *fp);
```

`fflush()` ritorna 0 in caso di successo altrimenti EOF

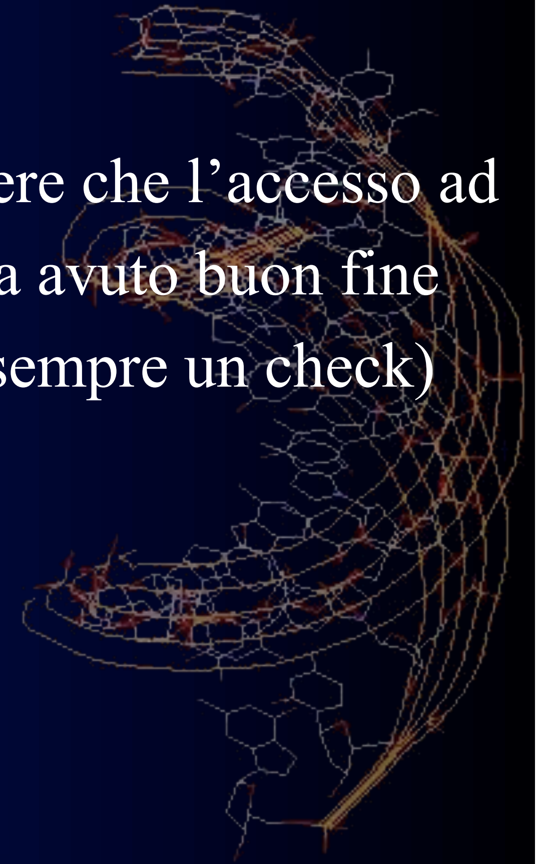
DO & DON'T

Aprire un file prima di effettuarne la lettura o la scrittura

Utilizzare `sizeof()` con le funzioni `fwrite()` e `fread()`

Chiudere tutti i files che sono stati aperti

Non assumere che l'accesso ad un file abbia avuto buon fine (effettuare sempre un check)



Sequenziale vs random

Ad ogni file aperto è associato un indicatore di posizione.
Tale indicatore specifica dove hanno luogo le operazioni di scrittura e/o lettura.
La posizione è espressa in termini di bytes a partire dall'inizio del file.
Quando un file già esistente viene aperto l'indicatore si troverà alla fine dello stesso se è stato aperto con la modalità *append*, in tutti gli altri casi si troverà all'inizio.
Le funzioni di I/O su file utilizzano questo indicatore di posizione, e pertanto la lettura (scrittura) sequenziale su file non richiede particolare cura.

Sequenziale vs random

Per avere però un maggiore controllo è possibile utilizzare delle funzioni che consentono di conoscere e modificare il valore dell'indicatore di posizione.

In questo modo è possibile accedere ad un file in maniera arbitraria (random access).

Posizionamento su file

La funzione `rewind()` posiziona l'indice del file al suo inizio

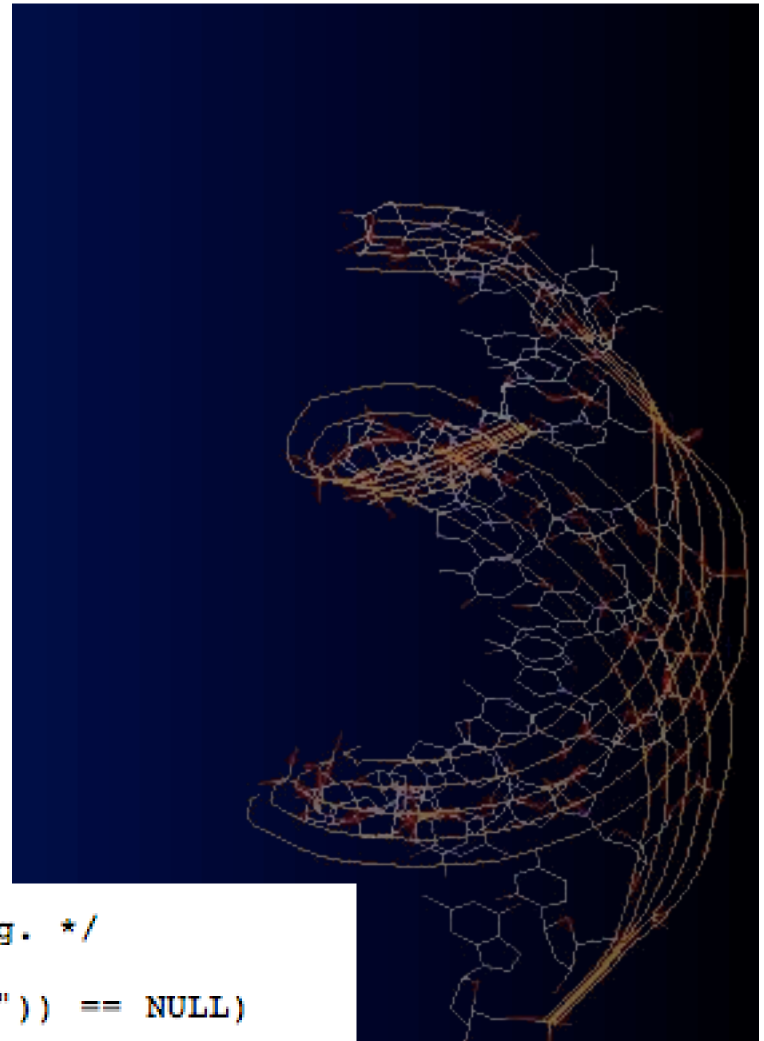
```
void rewind(FILE *fp);
```

La funzione `ftell()` consente invece di conoscere la posizione dell'indicatore all'interno di un file

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

In caso di errore `ftell()` ritorna -1L

```
/* Demonstrates ftell() and rewind(). */
#include <stdlib.h>
#include <stdio.h>

#define BUFLEN 6

char msg[] = "abcdefghijklmnopqrstuvwxyz";

int main()
 {
     FILE *fp;
     char buf[BUFLEN];

     if ( (fp = fopen("TEXT.TXT", "w")) == NULL)
     {
         fprintf(stderr, "Error opening file.");
         exit(1);
     }

     if (fputs(msg, fp) == EOF)
     {
         fprintf(stderr, "Error writing to file.");
         exit(1);
     }

     fclose(fp);
```



```
/* Now open the file for reading. */

if ( (fp = fopen("TEXT.TXT", "r")) == NULL)

{
    fprintf(stderr, "Error opening file.");
    exit(1);
}
printf("\nImmediately after opening, position = %ld", ftell(fp));

/* Read in 5 characters. */

fgets(buf, BUFLEN, fp);
printf("\nAfter reading in %s, position = %ld", buf, ftell(fp));
```

```c
    /* Read in the next 5 characters. */

    fgets(buf, BUFLEN, fp);
    printf("\n\nThe next 5 characters are %s, and position now = %ld",
            buf, ftell(fp));

    /* Rewind the stream. */

    rewind(fp);

    printf("\n\nAfter rewinding, the position is back at %ld",
            ftell(fp));

    /* Read in 5 characters. */

    fgets(buf, BUFLEN, fp);
    printf("\nand reading starts at the beginning again: %s\n", buf);
    fclose(fp);
    return(0);
}
```

OUTPUT:

```
Immediately after opening, position = 0
After reading in abode, position = 5


The next 5 characters are fghij, and position now = 10


After rewinding, the position is back at 0
and reading starts at the beginning again: abcde
```

Posizionamento su file

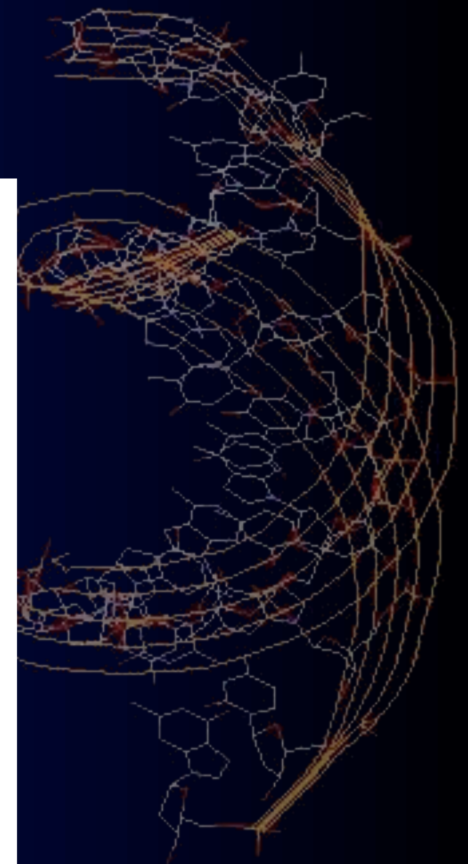
E' possibile ottenere un controllo più preciso dell'indicatore di posizione mediante la funzione `fseek()`

```
int fseek(FILE * fp, long offset, int place);
```

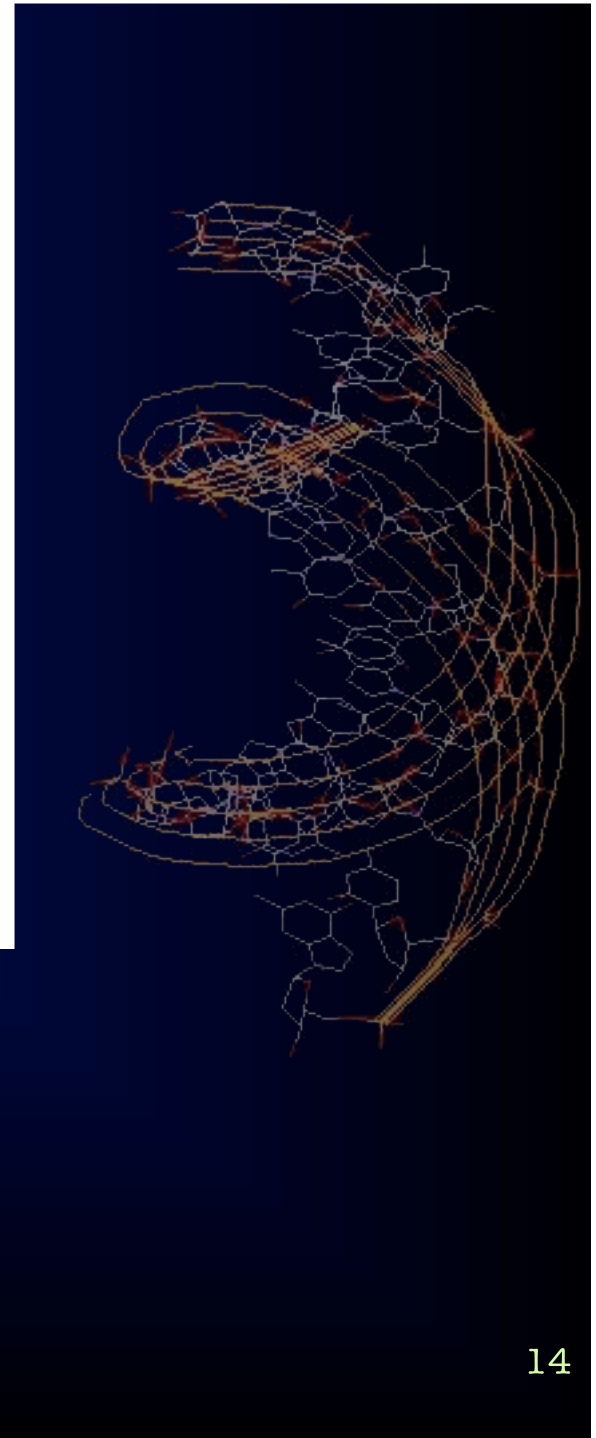
`fseek` posiziona l'indice di posizione del file a *offset* bytes da *place* , che può assumere i valori `SEEK_SET` (inizio file), `SEEK_CUR` (posizione corrente) oppure `SEEK_END` (fine file)

```
rewind(fp) ⇒ fseek(fp, 0L, SEEK_SET)
```

```
/* Random access with fseek(). */

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

#define MAX 50

int main()
{
    FILE *fp;
    int data, count, array[MAX];
    long offset;

    /* Initialize the array. */

    for (count = 0; count < MAX; count++)
        array[count] = count * 10;

    /* Open a binary file for writing. */

    if ( (fp = fopen("RANDOM.DAT", "wb")) == NULL)
    {
        fprintf(stderr, "\nError opening file.");
        exit(1);
    }

    /* Write the array to the file, then close it. */

    if ( (fwrite(array, sizeof(int), MAX, fp)) != MAX)
    {
        fprintf(stderr, "\nError writing data to file.");
        exit(1);
    }

    fclose(fp);
```



```
/* Open the file for reading. */

if ( (fp = fopen("RANDOM.DAT", "rb")) == NULL)
{
    fprintf(stderr, "\nError opening file.");
    exit(1);
}

/* Ask user which element to read. Input the element */
/* and display it, quitting when -1 is entered. */

while(1)
{
    printf("\nEnter element to read, 0-%d, -1 to quit: ",MAX );

    scanf("%ld", &offset);

    if (offset < 0)
        break;
    else if (offset > MAX-1)
        continue;

    /* Move the position indicator to the specified element. */

    if ( (fseek(fp, (offset*sizeof(int)), SEEK_SET)) != 0)

    {
        fprintf(stderr, "\nError using fseek().");
        exit(1);
    }

    /* Read in a single integer. */

    fread(&data, sizeof(int), 1, fp);

    printf("\nElement %ld has value %d.", offset, data);
}

fclose(fp);
return(1);
```



End of file

In alcuni casi si conosce con esattezza la dimensione del file in esame, mentre in altri può risultare necessario localizzare la fine di un file.

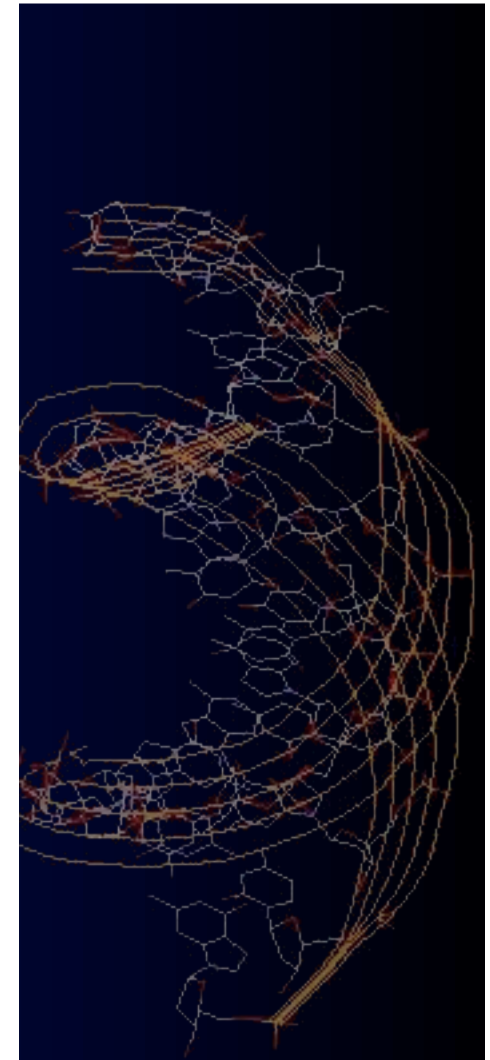
Quando si lavora con un file in modalità testo e si sta leggendo carattere-per-carattere allora è vantaggioso cercare l'EOF

```
while ( (c = fgetc( fp )) != EOF )
```

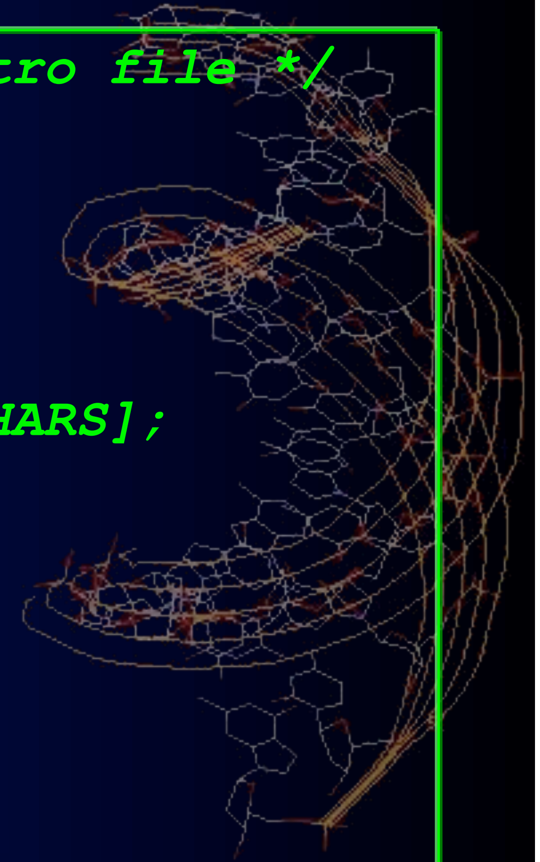
La soluzione precedente non è utilizzabile nel caso di file binari

```
int feof(FILE *fp);
```

```c
/* Detecting end-of-file. */
#include <stdlib.h>
#include <stdio.h>

#define BUFSIZE 100

int main()
{
        char buf[BUFSIZE];
:       char filename[60];
:       FILE *fp;
:
:       puts("Enter name of text file to display: ");
:       gets(filename);
:
:       /* Open the file for reading. */
:       if ( (fp = fopen(filename, "r")) == NULL)
:       {
:           fprintf(stderr, "Error opening file.");
:           exit(1);
:       }
:
:       /* If end of file not reached, read a line and display it. */
:
:       while ( !feof(fp) )
:       {
:           fgets(buf, BUFSIZE, fp);
:           printf("%s",buf);
:       }
:
:       fclose(fp);
:       return(0);
: }
```



Un esempio

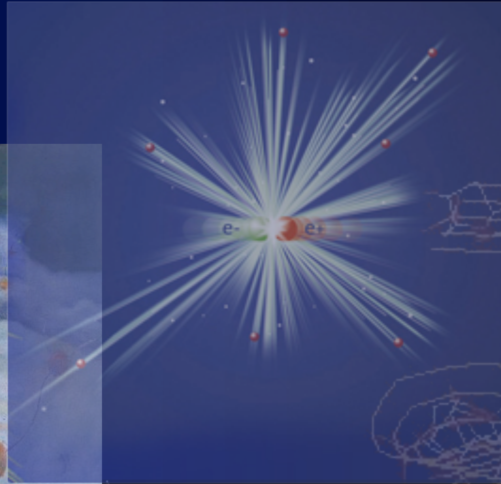


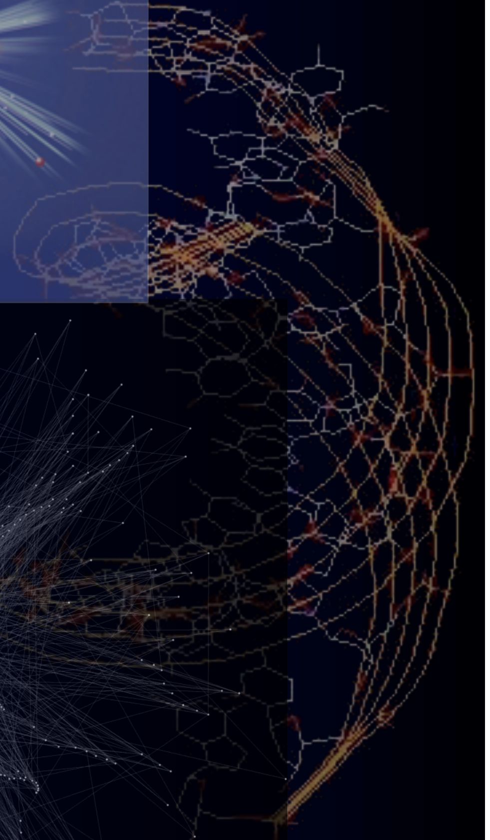
```
/* scrive un file al contrario in un altro file */
#include <stdio.h>
int main(){
 FILE *ifp, *ofp;
 const int MAXCHARS=100;
 char inp_name[MAXCHARS], out_name[MAXCHARS];
 int c;

 printf("\nNome file in ingresso\n");
 scanf("%s",inp_name);
 printf("\nNome file in uscita\n");
 scanf("%s",out_name);
 ifp   = MyFopen(inp_name,"r");
 ofp   = MyFopen(out_name,"w");
/* segue…*/
```

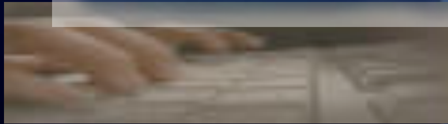
...segue

```
/* …segue */
fseek(ifp,0,SEEK_END); /* si sposta alla
                          fine del file */
fseek(ifp,-1,SEEK_CUR); /* si sposta indietro di un
                          byte = 1 carattere */
while(ftell(ifp)>0){
 c=fgetc(ifp) ; /*legge il carattere e sposta in avanti
                  il puntatore di posizione file */
 fputc(ofp,c); /* stampa il carattere sull'output */
 fseek(ifp,-2,SEEK_CUR); /* torna indietro di due
                          caratteri*/
 }
 fclose(ifp);  /* chiude il file in ingresso */
 fclose(ofp);  /*chiude il file di output */
 return 0;
}
```

Strutture







Struttura

Si definisce struttura una collezione di una o più variabili raggruppate insieme sotto un unico nome.

Le variabili in una struttura possono essere di tipi tra loro differenti

Una struttura può contenere qualsiaisi tipo di variabile, anche arrays e altre strutture

Ogni variabile all'interno di una struttura prende il nome di membro o campo della struttura

`struct`: tipo di dato user-defined

Oltre ai tipi di dati semplici, quali ***`int`***, ***`char`***, ***`float`***..., in C e' possibile creare nuovi tipi aggregando tipi di dati eterogenei sotto un unico nome. Questo nuovo tipo di dato va sotto il nome di *struttura* e si definisce con la parola chiave ***`struct`***:

```
struct [nome-struttura] {
      tipo_1 var_1;
      tipo_2 var_2;
        .
        .
      tipo_N var_N
};
```

nome della struttura (tag)

membri o campi della struttura

Definizione e dichiarazione

```
struct coord {
    int x;
    int y;
};
```

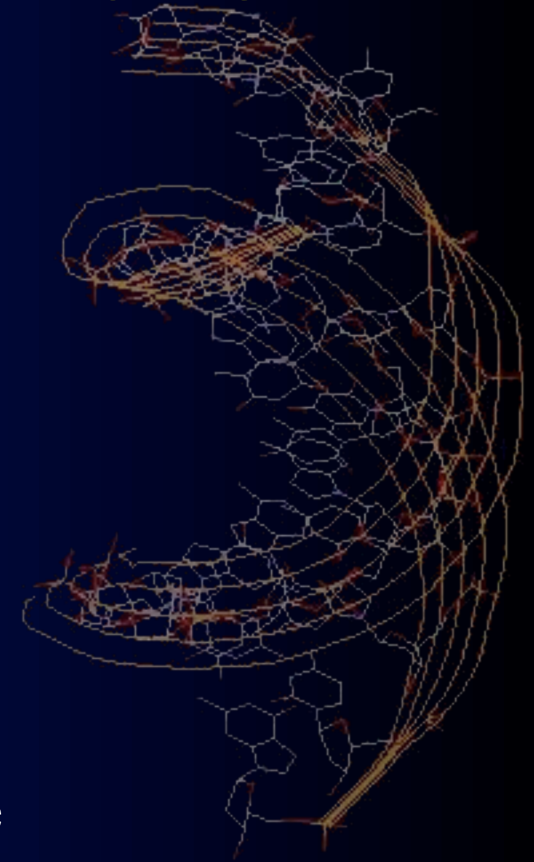
definizione

```
struct coord {
    int x;
    int y;
} first, second;
```

dichiarazione

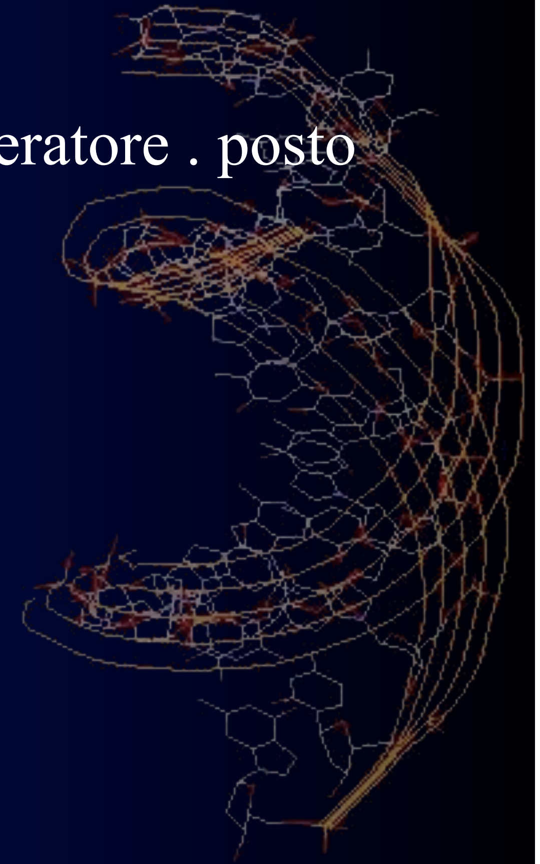
```
struct coord {
    int x;
    int y;
};

/* Additional code may go here */
struct coord first, second;
```

dichiarazione



Strutture

Si accede ai membri di una struttura mediante l’operatore . posto tra il nome della struttura e quello del membro



```
struct coord {
    int x;
    int y;
} first, second;
```

```
first.x = 50;
first.y = 100;
```

Importante

```
first = second;
```

```
first.x = second.x;
first.y = second.y;
```

Example 1

```
/* Declare a structure template called SSN */
struct SSN {
    int first_three;
    char dash1;
    int second_two;
    char dash2;
    int last_four;
}
/* Use the structure template */
struct SSN customer_ssn;
```

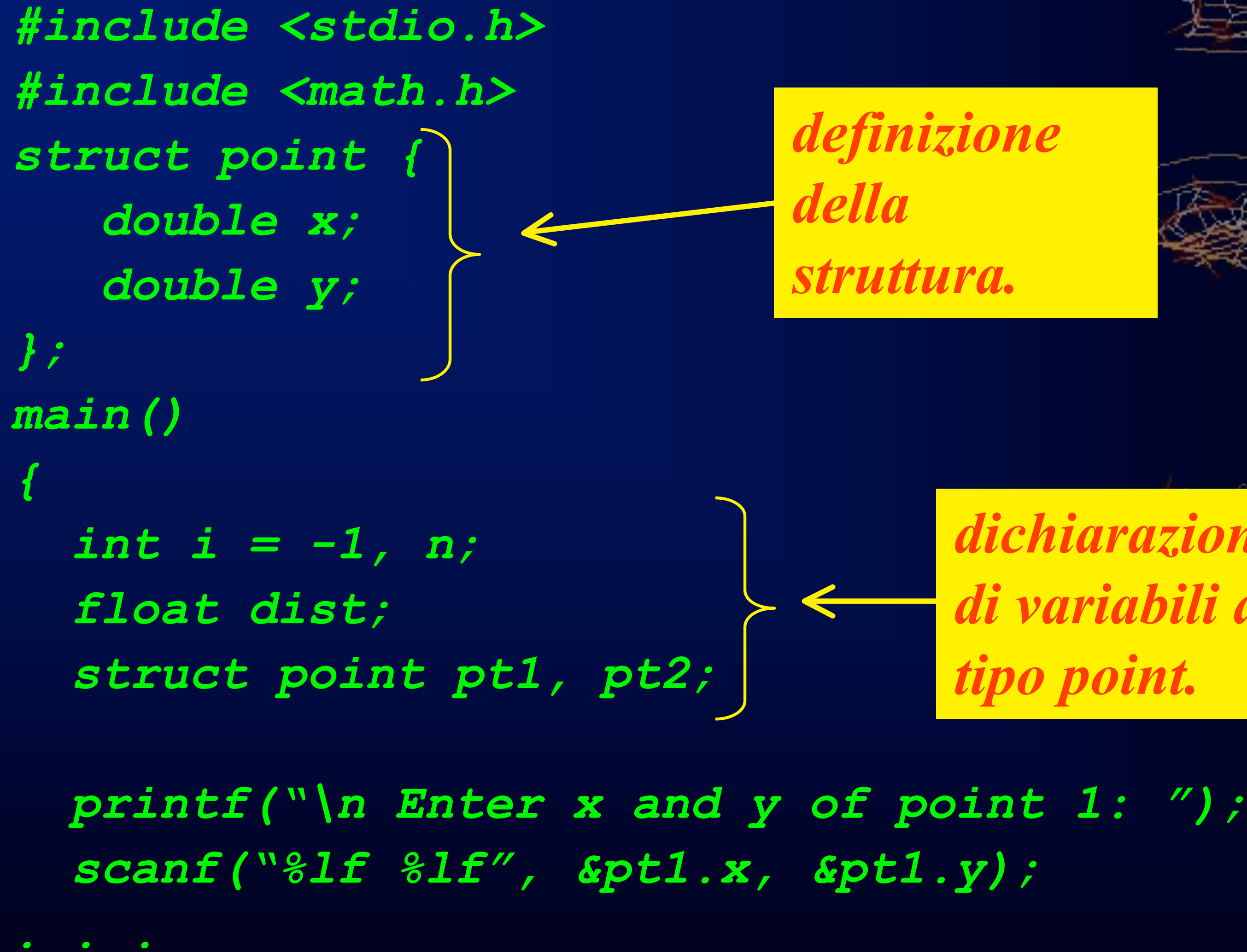
Example 2

```
/* Declare a structure and instance together */
struct date {
    char month[2];
    char day[2];
    char year[4];
} current_date;
```

Example 3

```
/* Declare and initialize a structure */
struct time {
    int hours;
    int minutes;
    int seconds;
} time_of_birth = { 8, 45, 0 };
```

Es: distanza tra due punti




```
#include <stdio.h>
#include <math.h>
struct point {
    double x;
    double y;
};
main()
{
  int i = -1, n;
  float dist;
  struct point pt1, pt2;

  printf("\n Enter x and y of point 1: ");
  scanf("%lf %lf", &pt1.x, &pt1.y);
. . .
```

. . . . continua

```
  printf("\n Enter x and y of point 2: ");
  scanf("%lf %lf", &pt2.x, &pt2.y);
  dist = (pt1.x - pt2.x) * (pt1.x - pt2.x);
  dist = dist + (pt1.y - pt2.y) * (pt1.y - pt2.y);
  dist = (dist > 0 ) ? sqrt(dist): 0.;
  printf("Distance = %lf", dist);
}
```

Definizione vs. Dichiarazione

Nella definizione di una variabile (struttura) viene informato il compilatore sulla composizione della stessa.

Noto il tipo di variabile (struttura) è possibile valutarne la sua dimensione in termini di bytes.

La dichiarazione riguarda l' esecuzione del programma: la dichiarazione è trasformata in istruzioni che allocano fisicamente la memoria necessaria alla vita della variabile (struttura). Una zona di memoria viene quindi riservata alla variabile quando questa è dichiarata: la variabile (struttura) esiste dal momento in cui viene dichiarata.

Strutture

Immaginiamo di dover scrivere un programma grafico che ha a che fare con dei rettangoli

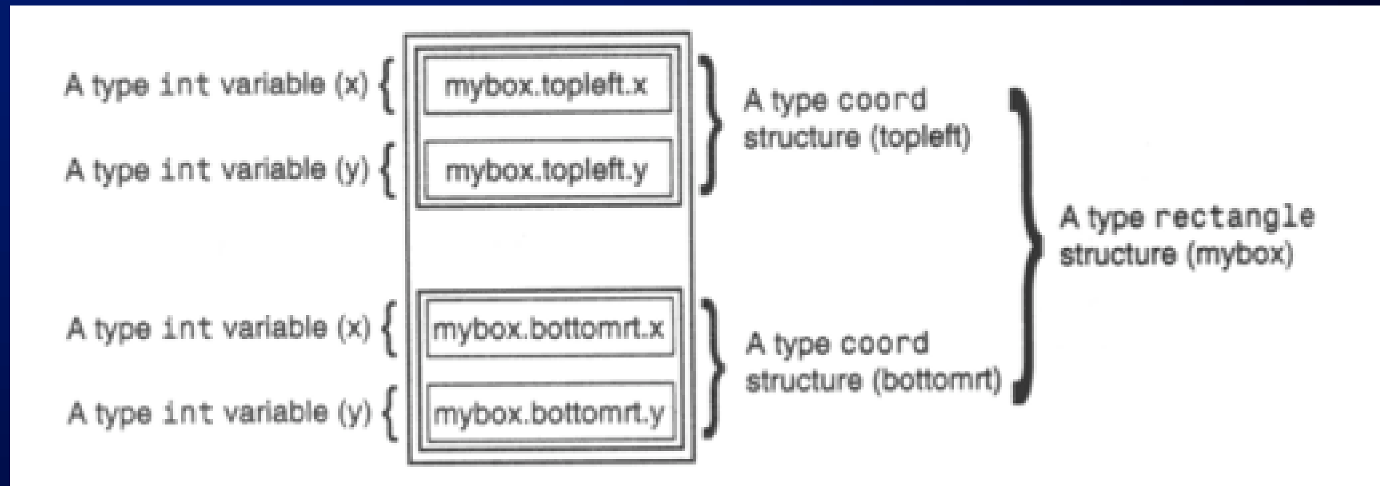
```
struct coord {
     int x;
     int y;
};
```

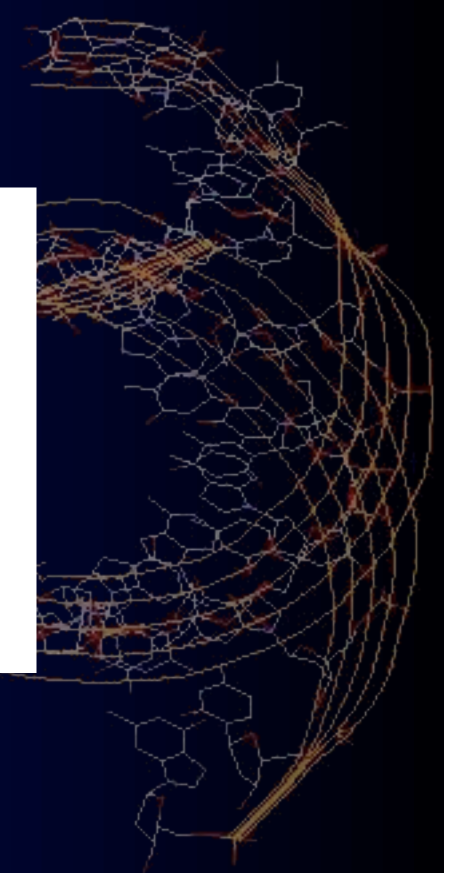
```
struct rectangle {
     struct coord topleft;
     struct coord bottomrt;
} mybox;
```

Per accedere ai membri bisogna utilizzare l'operatore . due volte

```
mybox.topleft.x = 0;
mybox.topleft.y = 10;
mybox.bottomrt.x = 100;
mybox.bottomrt.y = 200;
```

Strutture





```
/* Demonstrates structures that contain other structures. */

#include <stdio.h>

int length, width;
long area;

struct coord{
    int x;
    int y;
};

struct rectangle{
    struct coord topleft;
    struct coord bottomrt;
} mybox;

int main()
{
    /* Input the coordinates */

    printf("\nEnter the top left x coordinate: ");
    scanf("%d", &mybox.topleft.x);

    printf("\nEnter the top left y coordinate: ");
    scanf("%d", &mybox.topleft.y);

    printf("\nEnter the bottom right x coordinate: ");
    scanf("%d", &mybox.bottomrt.x);

    printf("\nEnter the bottom right y coordinate: ");
    scanf("%d", &mybox.bottomrt.y);

    /* Calculate the length and width */

    width = mybox.bottomrt.x - mybox.topleft.x;
    length = mybox.bottomrt.y - mybox.topleft.y;

    /* Calculate and display the area */

    area = width * length;
    printf("\nThe area is %ld units.\n", area);

    return 0;
}
```

Strutture

Si possono definire strutture contenenti uno o più arrays

```
struct data{
    int x[4];
    char y[10];
};
```

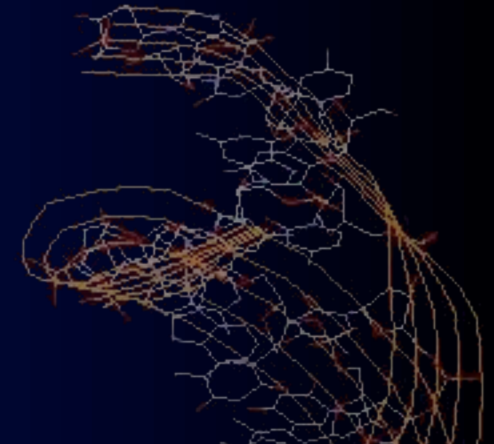
Una volta dichiarata una struttura di tipo `data` di nome `record`

```
struct data record;
```

È possibile accedere ai singoli elementi con istruzioni del tipo

```
record.x[2] = 100;
record.y[1] = 'x';
```

```c
/* Demonstrates a structure that has array members. */

#include <stdio.h>

/* Define and declare a structure to hold the data. */
/* It contains one float variable and two char arrays. */

struct data{
     float amount;
     char fname[30];
     char lname[30];
} rec;

int main()
{
     /* Input the data from the keyboard. */

    printf("Enter the donor's first and last names,\n");
    printf("separated by a space: ");
    scanf("%s %s", rec.fname, rec.lname);

    printf("\nEnter the donation amount: ");
    scanf("%f", &rec.amount);

    /* Display the information. */
    /* Note: %.2f specifies a floating-point value */
    /* to be displayed with two digits to the right */
    /* of the decimal point. */

    /* Display the data on the screen. */

    printf("\nDonor %s %s gave $%.2f.\n", rec.fname, rec.lname,
            rec.amount);

    return 0;
}
```

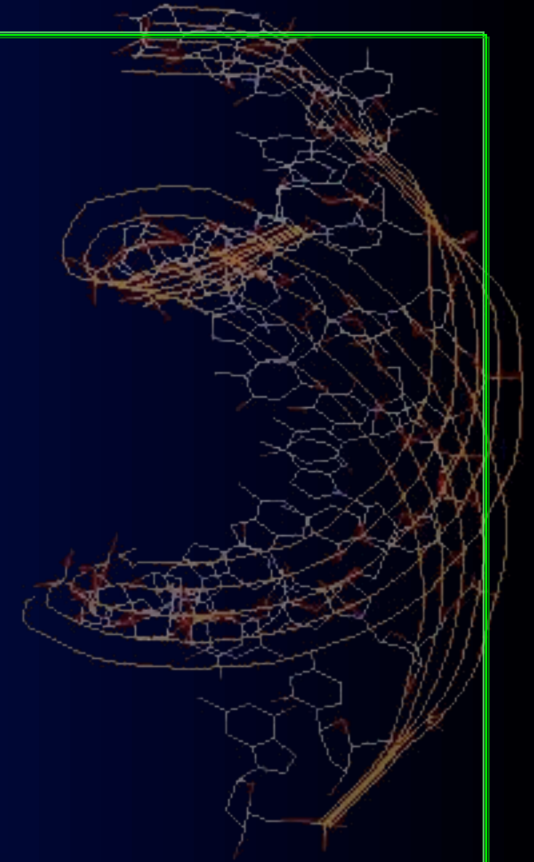


INPUT/OUTPUT:

```
Enter the donor's first and last names,
separated by a space: Bradley Jones


Enter the donation amount: 1000.00


Donor Bradley Jones gave $1000.00.
```

Array di strutture

```
#include<stdio.h>
struct Circle {
   float raggio;
   float area;
};
float CircleArea(float Radius);

main()
{
  int i = -1, n;
  struct Circle circle[100];
  printf("\n Enter 0 to QUIT.");
. . . .
```



```
  do{
        i++;
        printf("\n Radius ? ");
        scanf("%f", &circle[i].raggio);
        circle[i].area = CircleArea(circle[i].raggio);
  } while(circle[i].raggio != 0)
  n = --i;
  for(i = 0; i < n; i++){
        printf("\n Radius =%f, Area = %f",
                          circle[i].raggio,
                          circle[i].area);
  }
}
```

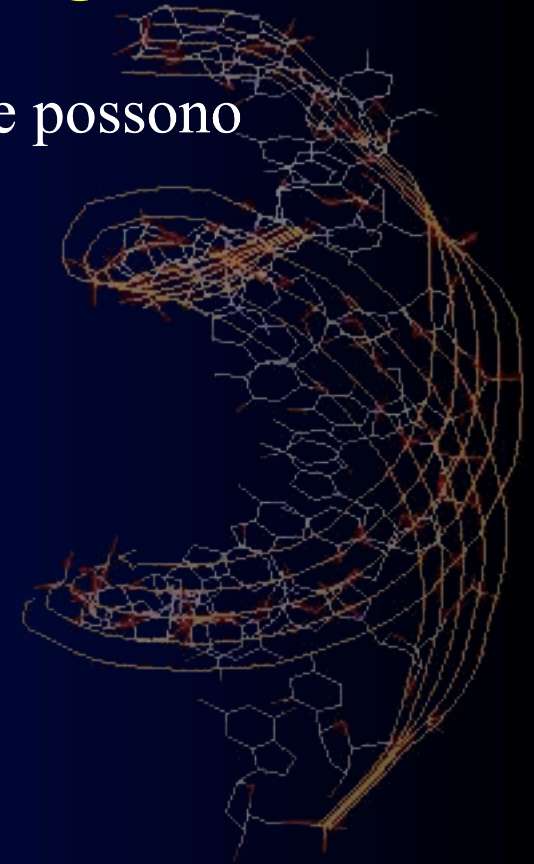
```c
/* Demonstrates using arrays of structures. */

#include <stdio.h>

/* Define a structure to hold entries. */

struct entry {
     char fname[20];
     char lname[20];
     char phone[10];
};

/* Declare an array of structures. */

struct entry list[4];

int i;

int main()
{
    /* Loop to input data for four people. */

    for (i = 0; i < 4; i++)
    {
        printf("\nEnter first name: ");
        scanf("%s", list[i].fname);
        printf("Enter last name: ");
        scanf("%s", list[i].lname);
        printf("Enter phone in 123-4567 format: ");
        scanf("%s", list[i].phone);
    }

    /* Print two blank lines. */

    printf("\n\n");

    /* Loop to display data. */

    for (i = 0; i < 4; i++)
    {
        printf("Name: %s %s", list[i].fname, list[i].lname);
        printf("\t\tPhone: %s\n", list[i].phone);
    }

    return 0;
}
```

Inizializzare strutture

Come tutte le altre variabili in C anche le strutture possono essere inizializzate alla loro dichiarazione

```
struct sale {
    char customer[20];
    char item[20];
    float amount;
} mysale = { "Acme Industries",
             "Left-handed widget",
              1000.00
           };
```

```
struct customer {
    char firm[20];
    char contact[25];
}

struct sale {
   struct customer buyer;
   char item[20];
   float amount;
} mysale = { { "Acme Industries", "George Adams"},
               "Left-handed widget",
                1000.00
           };
```



Puntatori & Strutture

Una struttura può contare tra i suoi campi puntatori a qualsiasi tipo di dato.

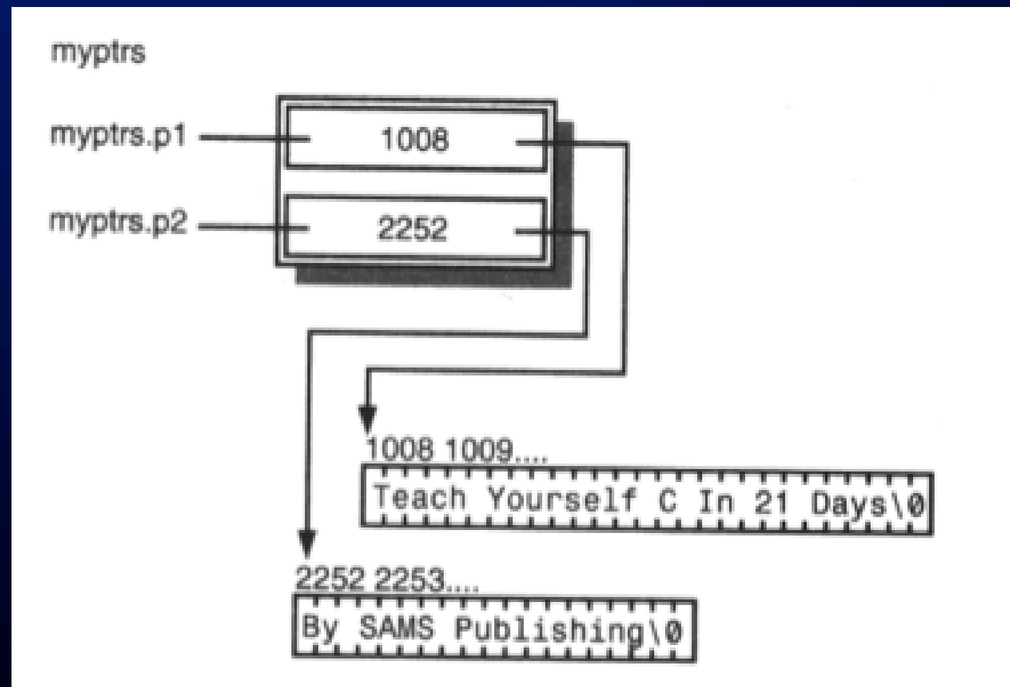
```
struct data {
    int *value;
    int *rate;
} first;
```

Il tipo di puntatore più frequentemente utilizzato come campo di una struttura è il puntatore a `char`

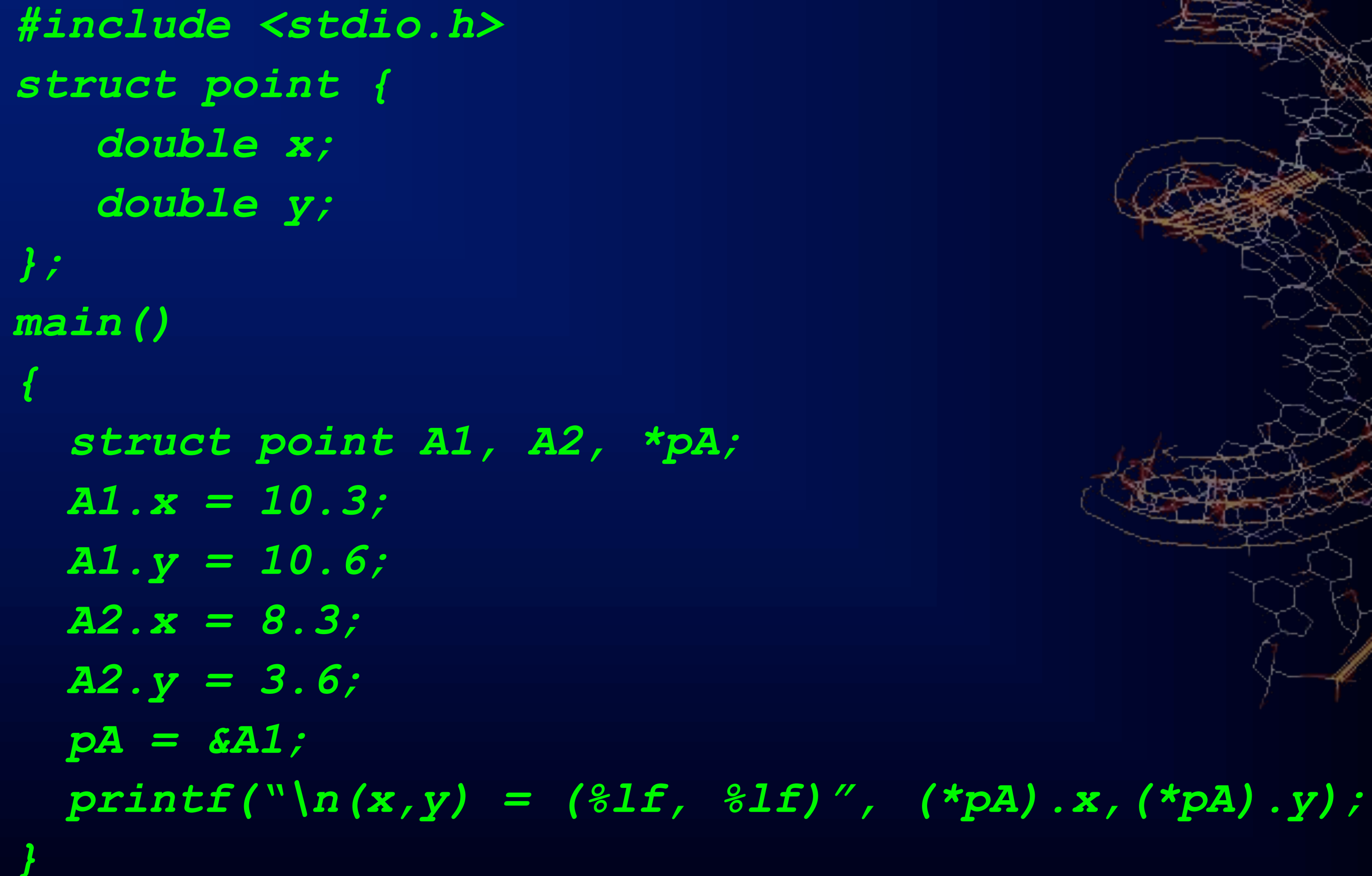
```
struct msg {
    char *p1;
    char *p2;
} myptrs;
myptrs.p1 = "Teach Yourself C In 21 Days";
myptrs.p2 = "By SAMS Publishing";
```

Puntatori & Strutture

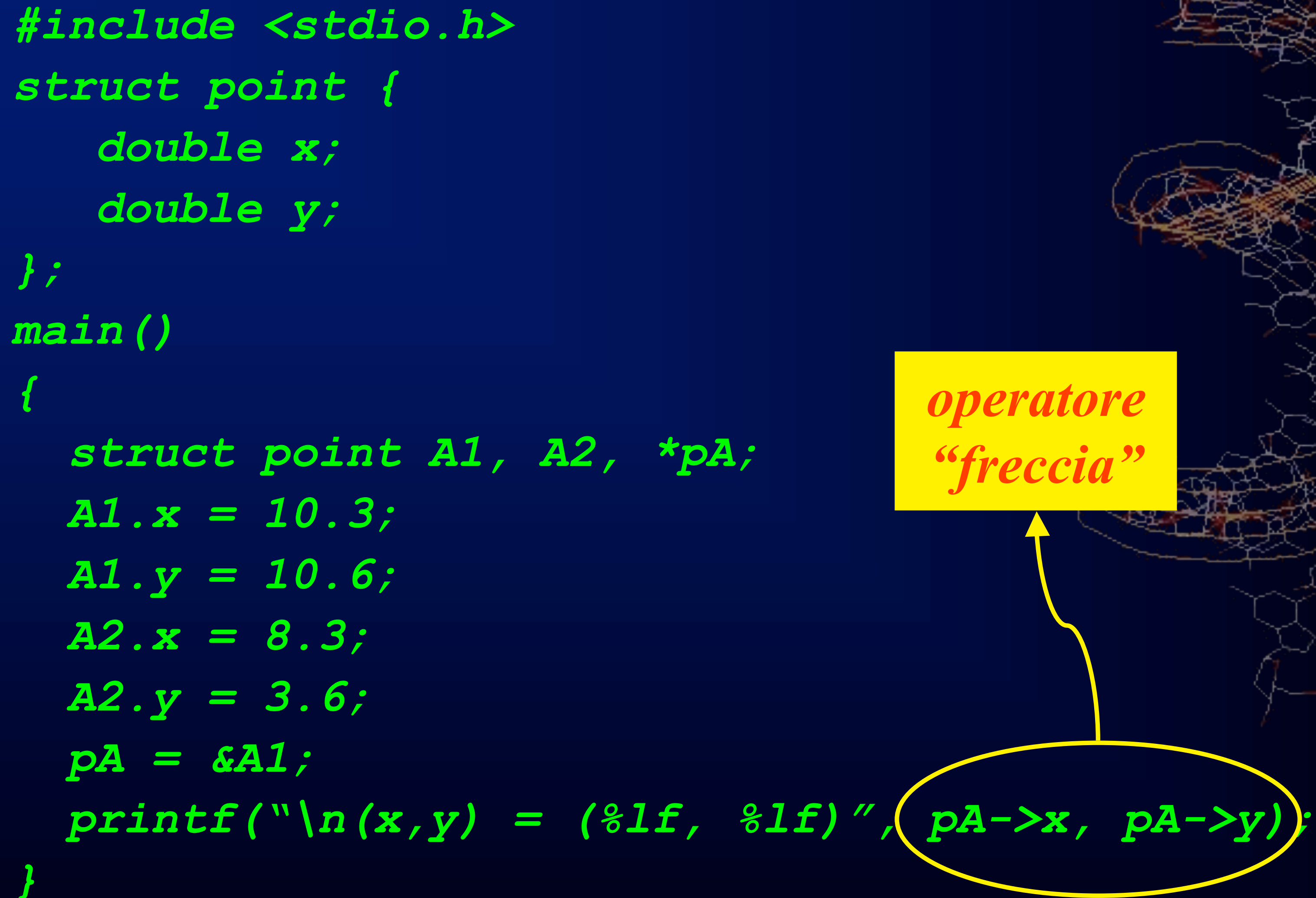
```
struct msg {
    char *p1;
    char *p2;
} myptrs;
myptrs.p1 = "Teach Yourself C In 21 Days";
myptrs.p2 = "By SAMS Publishing";
```



Puntatori & Strutture

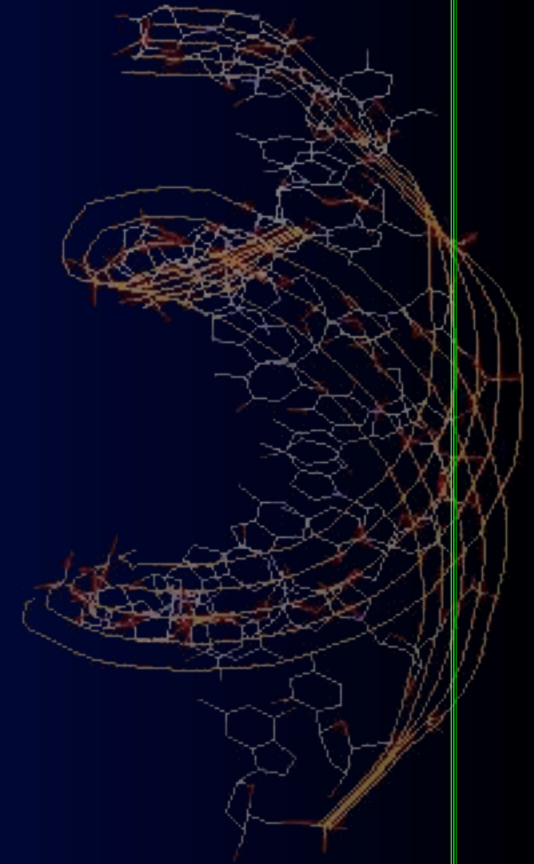


Puntatori & Strutture



Puntatori & Strutture

```
#include <stdio.h>
struct point {
    double x;
    double y;
};
main()
{
  struct point A1, A2, *pA;
  pA = &A1;
  pA->x = 10.3;
  pA->y = 10.6;
  pA = &A2;
  pA->x = 8.3;
  pA->y = 3.6;
  printf("\n(x,y) = (%lf, %lf)", pA->x, pA->y);
}
```



Puntatori & Strutture

Abbiamo pertanto tre modi di accedere ad un campo di una struttura

- mediante il nome della struttura `str.memb`
- mediante un puntatore e l'op. * `(*p_str).memb`
- mediante un puntatore e l'op. freccia `p_str->memb`

Puntatori & Strutture

Si possono utilizzare dei puntatori per accedere a strutture che sono elementi di un array

Esempio

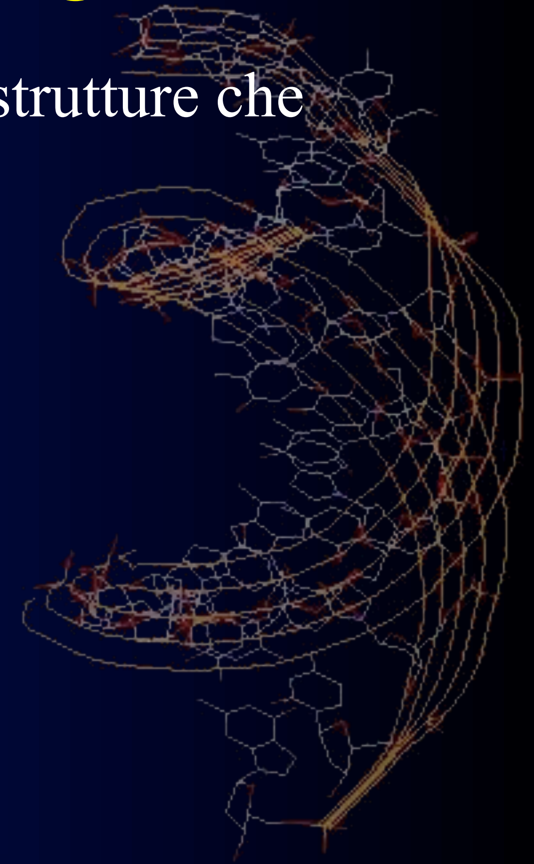
```
struct part {
     int number;
     char name[10];
};
```

```
struct part data[100];
```

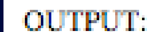
```
struct part *p_part;
p_part = &data[0];
```

```
p_part = data;
```

```
printf("%d %s", p_part->number, p_part->name);
```



```c
/* Demonstrates stepping through an array of structures */
/* using pointer notation. */

#include <stdio.h>

#define MAX 4

/* Define a structure, then declare and initialize */
/* an array of 4 structures. */

struct part {
    int number;
    char name[10];
} data[MAX] = {1, "Smith",
               2, "Jones",
               3, "Adams",
               4, "Wilson"
               };

/* Declare a pointer to type part, and a counter variable. */

struct part *p_part;
int count;

int main()
{
    /* Initialize the pointer to the first array element. */

    p_part = data;

    /* Loop through the array, incrementing the pointer */
    /* with each iteration. */

    for (count = 0; count < MAX; count++)
    {
        printf("At address %d: %d %s\n", p_part, p_part->number,
                p_part->name);
        p_part++;
    }

    return 0;
}
```



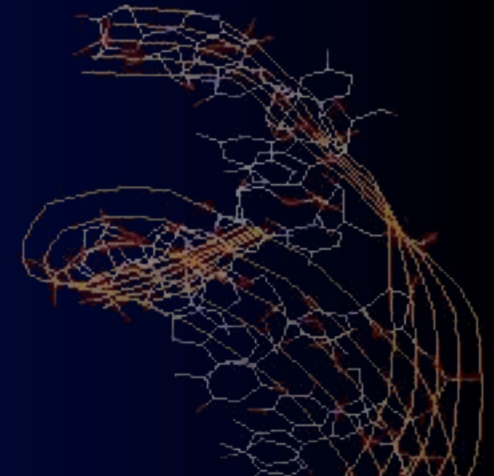
OUTPUT:

```
At address 96: 1 Smith
At address 108: 2 Jones
At address 120: 3 Adams
At address 132: 4 Wilson
```

Strutture e funzioni

Come le altre variabili una struttura può essere passata come argomento ad una funzione

```c
/* Demonstrates passing a structure to a function. */

#include <stdio.h>

/* Declare and define a structure to hold the data. */

struct data {
    float amount;
    char fname[30];
    char lname[30];
} rec;

/* The function prototype. The function has no return value,
/* and it takes a structure of type data as its one argument

void print_rec(struct data x);

int main()
{
    /* Input the data from the keyboard. */

    printf("Enter the donor's first and last names,\n");
    printf("separated by a space: ");
    scanf("%s %s", rec.fname, rec.lname);

    printf("\nEnter the donation amount: ");
    scanf("%f", &rec.amount);

    /* Call the display function. */
    print_rec( rec );

    return 0;
}
void print_rec(struct data x)
{
    printf("\nDonor %s %s gave $%.2f.\n", x.fname, x.lname,
            x.amount);
}
```



INPUT/OUTPUT:

```
Enter the donor's first and last names,
separated by a space: Bradley Jones


Enter the donation amount: 1000.00


Donor Bradley Jones gave $1000.00.
```

Puntatori & Strutture

Una struttura può contare tra i suoi campi puntatori a qualsiasi tipo di dato. Sono perciò ammessi anche puntatori a struttura, persino *puntatori* a *struttura identificata dal medesimo tag*.

```
#include <stdio.h>
struct point {
    double x;
    double y;
    struct point *next;
};
struct rect {
    struct point *ListPt;
    int FillColor;
};
main()
{
. . . . . .
```

Utilizzo di typedef

Il linguaggio C fornisce il meccanismo *typedef* che permette al programmatore di associare un tipo ad un identificatore

```
typedef char uppercase;
typedef int FEET;
typedef unsigned int site_t;  /* stddef.h */
…..
uppercase u;
FEET length, width;
```

Esempio

```
struct complex {
        double re;
        double im;
};
typedef struct complex complex;
void add(complex *a, complex *b, complex *c){
*a.re = *b.re + *c.re;
*a.im = *b.im + *c.im;
}
```