

Definizione di una struttura mediante typedef



- E' possibile definire un nuovo tipo struttura utilizzando una **typedef**

```
typedef struct {
    tipo0 nomeVar0;
    tipo1 nomeVar1;
    ...
} nomeTipoStruttura;
```

Nota: non crea variabili!

- Esempio: Definizione di un nuovo tipo struttura con nome `studenteT` (T ci ricorda che è il nome di un tipo):

```
typedef struct {
    char nome[30];
    char cognome[30];
    int voto;
} studenteT;
```

- Oppure (definendo in precedenza `struct studente`):

```
typedef struct studente studenteT;
```

- `studenteT` è sinonimo di `struct studente`
- Dichiarazione di una variabile `stud_ing` di tipo `studenteT`

```
studenteT stud_ing;
```

5

Accesso ai campi di una struttura



- Per accedere ai campi di una struttura si utilizza l'operatore `."` posto dopo il nome della variabile struttura

Esempio

```
typedef struct {
    char nome[30];
    char cognome[30];
    int voto;
} studenteT;
```

```
void main() {
    studenteT stud_ing;
    /* inizializzazione dei campi della struttura da tastiera */
    scanf("%s", stud_ing.nome);
    scanf("%s", stud_ing.cognome);
    scanf("%d", &stud_ing.voto);
    ...
    /* visualizzazione dei campi di una struttura */
    printf("%s", stud_ing.nome);
    printf("%s", stud_ing.cognome);
    printf("%d", stud_ing.voto);
}
```

6

Passaggio di una struttura a funzione (1)



- Le definizioni di tipo devono essere poste dopo le direttive `define` e prima dei prototipi di funzioni in modo da poter essere utilizzate dalle funzioni
- Le strutture vengono passate alle funzioni **per copia**

Esempio

```
#define N 30
typedef struct {
    char nome[N];
    char cognome[N];
    int voto;
} studenteT;
void funz(studentT stud_ing);
main() {
    studenteT stud_ing;
    funz(stud_ing);
}
void funz(studentT stud) {
    printf("%s\n", stud.nome);
    printf("%s\n", stud.cognome);
    printf("%d\n", stud.voto);
}
```

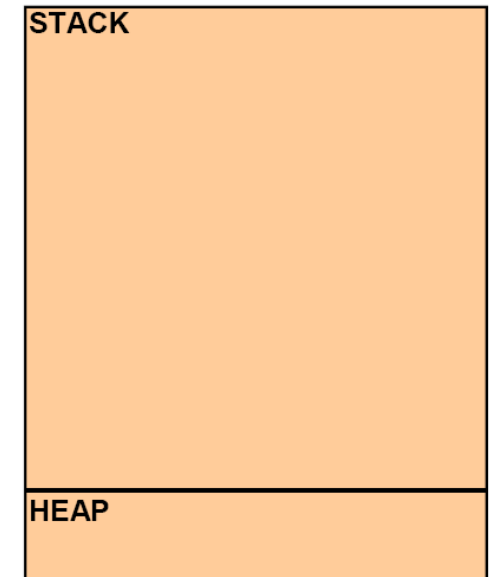
7

Passaggio di una struttura a funzione (2)



Esempio

```
typedef struct {
    char name[30];
    int age;
    double height;
} personT;
void GetPersonData(personT x);
main() {
    personT person1;
    personT person2;
    GetPersonData(person1);
}
void GetPersonData(personT x){
    scanf("%s", x.name);
    scanf("%d", &x.age);
    scanf("%lf", &x.height);
}
```



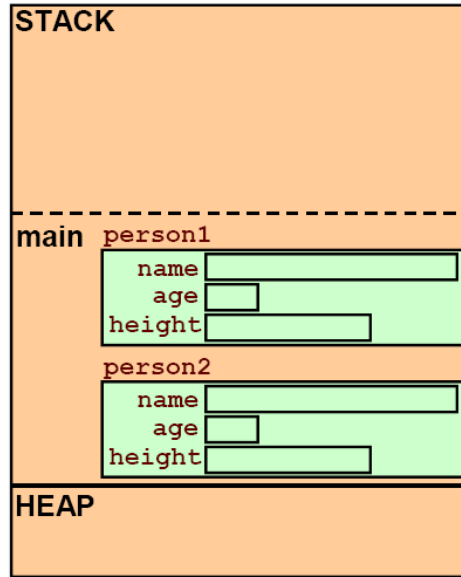
8

Passaggio di una struttura a funzione (3)



Esempio

```
typedef struct {
    char name[30];
    int age;
    double height;
} personT;
void GetPersonData(personT x);
main() {
    personT person1;
    personT person2;
    GetPersonData(person1);
}
void GetPersonData(personT x){
    scanf("%s", x.name);
    scanf("%d", &x.age);
    scanf("%lf", &x.height);
}
```



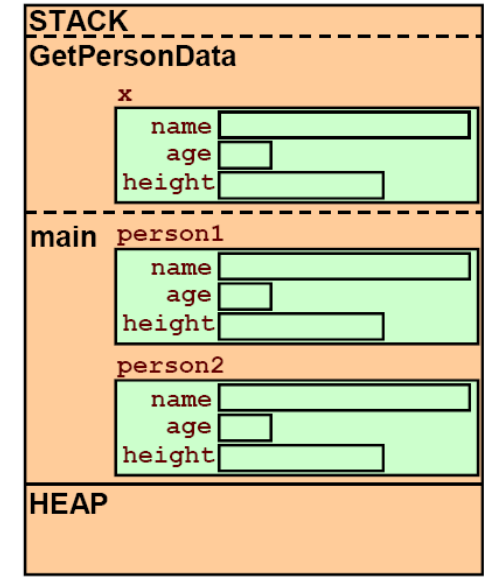
9

Passaggio di una struttura a funzione (4)



Esempio

```
typedef struct {
    char name[30];
    int age;
    double height;
} personT;
void GetPersonData(personT x);
main() {
    personT person1;
    personT person2;
    GetPersonData(person1);
}
void GetPersonData(personT x){
    scanf("%s", x.name);
    scanf("%d", &x.age);
    scanf("%lf", &x.height);
}
```



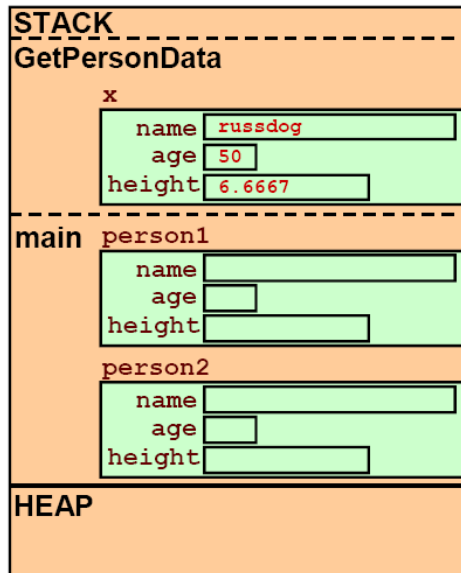
10

Passaggio di una struttura a funzione (5)



Esempio

```
typedef struct {
    char name[30];
    int age;
    double height;
} personT;
void GetPersonData(personT x);
main() {
    personT person1;
    personT person2;
    GetPersonData(person1);
}
void GetPersonData(personT x){
    scanf("%s", x.name);
    scanf("%d", &x.age);
    scanf("%lf", &x.height);
}
```



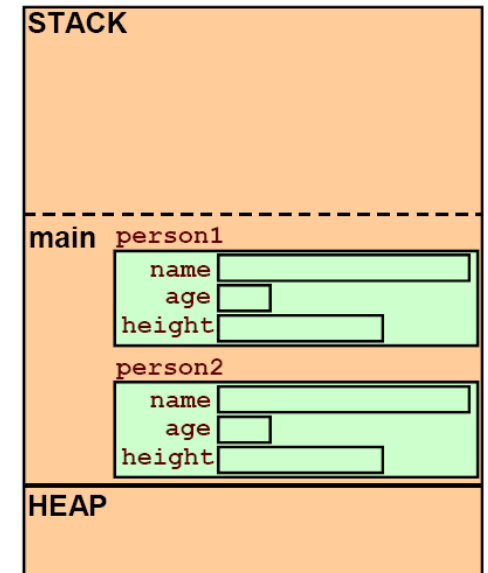
11

Passaggio di una struttura a funzione (6)



```
typedef struct {
    char name[30];
    int age;
    double height;
} personT;
void GetPersonData(personT x);
main() {
    personT person1;
    personT person2;
    GetPersonData(person1);
}
void GetPersonData(personT x){
    scanf("%s", x.name);
    scanf("%d", &x.age);
    scanf("%lf", &x.height);
}
```

- La funzione opera su una struttura locale
- Al termine della funzione tale struttura viene deallocata
- La funzione non modifica l'originale!



12

Puntatori a struttura (1)



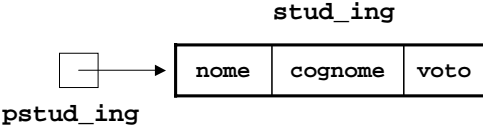
- Per modificare la struttura passata come parametro occorre utilizzare il **passaggio per riferimento**, mediante i puntatori
- Un **puntatore a struttura** è una variabile che contiene l'indirizzo di una variabile struttura

Esempio

```
/* dichiarazione e allocazione statica della struttura
stud_ing */
studenteT stud_ing;
```

```
/* dichiarazione di un puntatore ad una variabile di tipo
studenteT */
studenteT *pstud_ing;
```

```
pstud_ing = &stud_ing;
```



13

Puntatori a struttura (2)



- Accesso ai campi di una struttura tramite puntatori
 - Valgono le stesse regole per l'accesso alle variabili semplici tramite puntatori;

Esempio

```
studenteT stud_ing;
studenteT *pstud_ing;
```



```
pstud_ing = &stud_ing;
```

```
/* Accesso al campo "voto" della struttura "stud_ing" */
```

```
stud_ing.voto ⇔ (*pstud_ing).voto ⇔ pstud_ing->voto
```

Le parentesi sono obbligatorie

Sintassi alternativa e usata moltissimo per l'accesso ai campi di una struttura, riferita mediante un puntatore

14

Puntatori a struttura (3)



- Esempi di accesso ai campi di una struttura tramite puntatori

```
studenteT stud_ing;
studenteT *pstud_ing;
```



```
pstud_ing = &stud_ing;
```

```
/* Acquisizione dei valori dei campi cognome e voto */
scanf("%s %d\n", stud_ing.cognome, &stud_ing.voto);
/* oppure */
scanf("%s %d\n", (*pstud_ing).cognome, &(*pstud_ing).voto);
/* oppure */
scanf("%s %d\n", pstud_ing->cognome, &pstud_ing->voto);
```

```
/* Visualizzazione del contenuto del campo "cognome" e "voto" */
printf("%s %d\n", stud_ing.cognome, stud_ing.voto);
/* oppure */
printf("%s %d\n", (*pstud_ing).cognome, (*pstud_ing).voto);
/* oppure */
printf("%s %d\n", pstud_ing->cognome, pstud_ing->voto);
```

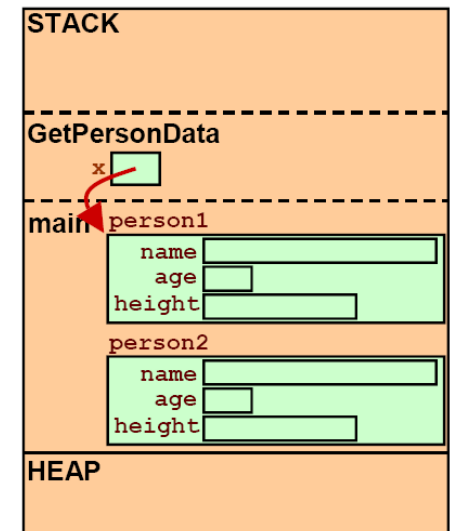
15

Passaggio di una struttura a funzione (7)



```
typedef struct {
char name[30];
int age;
double height;
} personT;
void GetPersonData(personT *x);
main() {
personT person1;
personT person2;
GetPersonData(&person1);
}
void GetPersonData(personT *x){
scanf("%s", (*x).name);
scanf("%d", &(*x).age);
scanf("%lf", &(*x).height);
}
```

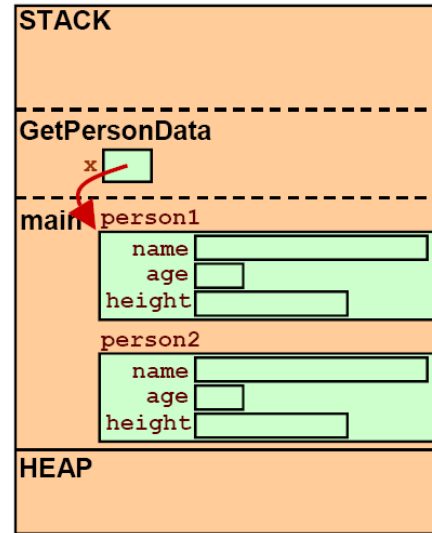
- Al posto di x occorre sostituire (*x)



Passaggio di una struttura a funzione (8)



```
typedef struct {
    char name[30];
    int age;
    double height;
} personT;
void GetPersonData(personT *x);
main() {
    personT person1;
    personT person2;
    GetPersonData(&person1);
}
void GetPersonData(personT *x){
    scanf("%s", x->name);
    scanf("%d", &x->age);
    scanf("%lf", &x->height);
}
```

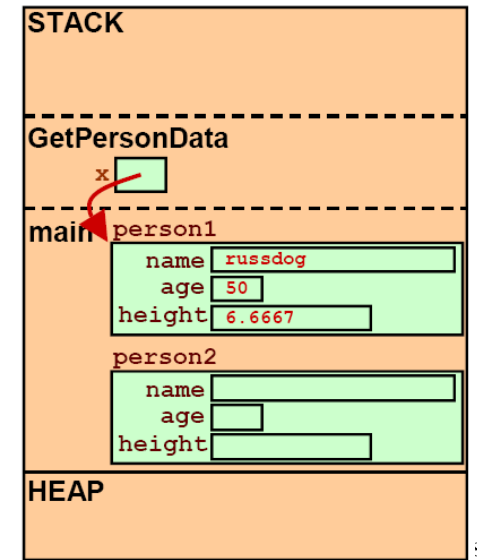


- Al posto di (*x). sostituiamo ->

Passaggio di una struttura a funzione (9)



```
typedef struct {
    char name[30];
    int age;
    double height;
} personT;
void GetPersonData(personT *x);
main() {
    personT person1;
    personT person2;
    GetPersonData(&person1);
}
void GetPersonData(personT *x){
    scanf("%s", x->name);
    scanf("%d", &x->age);
    scanf("%lf", &x->height);
}
```

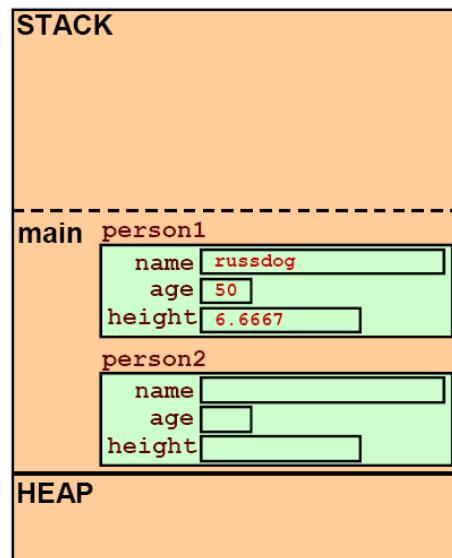


- La funzione ora modifica la struttura person1 contenuta nel main

Passaggio di una struttura a funzione (10)



```
typedef struct {
    char name[30];
    int age;
    double height;
} personT;
void GetPersonData(personT *x);
main() {
    personT person1;
    personT person2;
    GetPersonData(&person1);
}
void GetPersonData(personT *x){
    scanf("%s", x->name);
    scanf("%d", &x->age);
    scanf("%lf", &x->height);
}
```

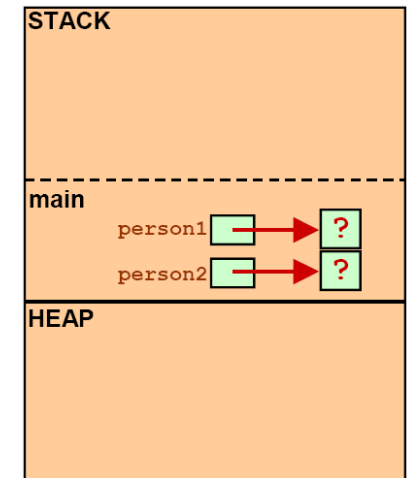


Allocazione dinamica di strutture (1)



- Possiamo definire strutture dinamiche, in maniera analoga agli array
- Utilizziamo un puntatore a struttura e la funzione di libreria malloc
- Esempio: utilizziamo una funzione **GetPersonData** per creare la struttura e per inizializzare i campi da tastiera

```
personT* GetPersonData();
main() {
    personT *person1;
    personT *person2;
    person1 = GetPersonData();
    printf("%s\n", person1->name);
    printf("%d\n", person1->age);
    printf("%.2f\n", person1->height);
}
personT* GetPersonData(){
    personT *x;
    x = (personT *) malloc(sizeof(personT));
    scanf("%s", x->name);
    scanf("%d", &x->age);
    scanf("%lf", &x->height);
    return x;
}
```



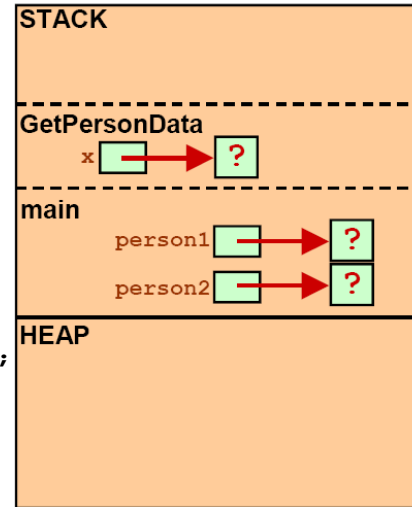


Allocazione dinamica di strutture (2)

```

personT* GetPersonData();
main() {
    personT *person1;
    personT *person2;
    person1 = GetPersonData();
    printf("%s\n", person1->name);
    printf("%d\n", person1->age);
    printf("%.2f\n", person1->height);
}

personT* GetPersonData(){
    personT *x;
    x = (personT *) malloc(sizeof(personT));
    scanf("%s", x->name);
    scanf("%d", &x->age);
    scanf("%lf", &x->height);
    return x;
}
    
```



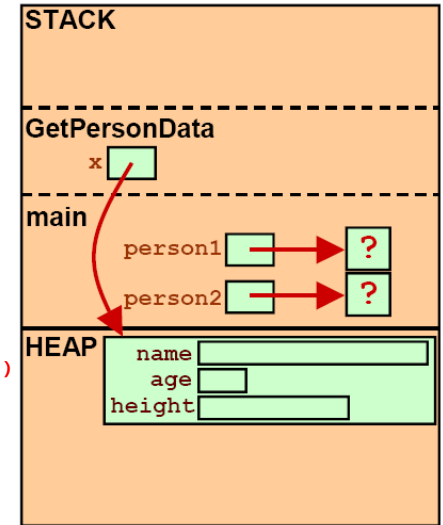
Allocazione dinamica di strutture (3)

- La struttura viene creata nell'heap mediante la funzione di libreria malloc

```

personT* GetPersonData();
main() {
    personT *person1;
    personT *person2;
    person1 = GetPersonData();
    printf("%s\n", person1->name);
    printf("%d\n", person1->age);
    printf("%.2f\n", person1->height);
}

personT* GetPersonData(){
    personT *x;
    x = (personT *) malloc(sizeof(personT));
    scanf("%s", x->name);
    scanf("%d", &x->age);
    scanf("%lf", &x->height);
    return x;
}
    
```

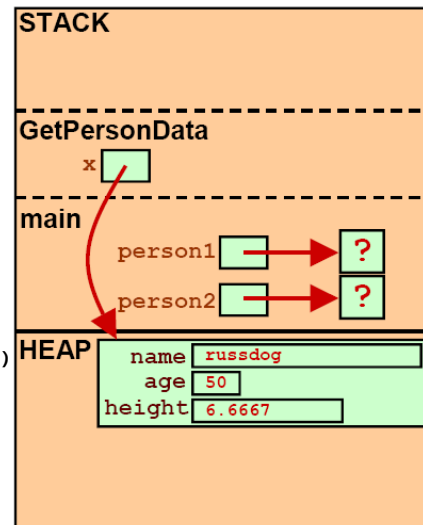


Allocazione dinamica di strutture (4)

```

personT* GetPersonData();
main() {
    personT *person1;
    personT *person2;
    person1 = GetPersonData();
    printf("%s\n", person1->name);
    printf("%d\n", person1->age);
    printf("%.2f\n", person1->height);
}

personT* GetPersonData(){
    personT *x;
    x = (personT *) malloc(sizeof(personT));
    scanf("%s", x->name);
    scanf("%d", &x->age);
    scanf("%lf", &x->height);
    return x;
}
    
```

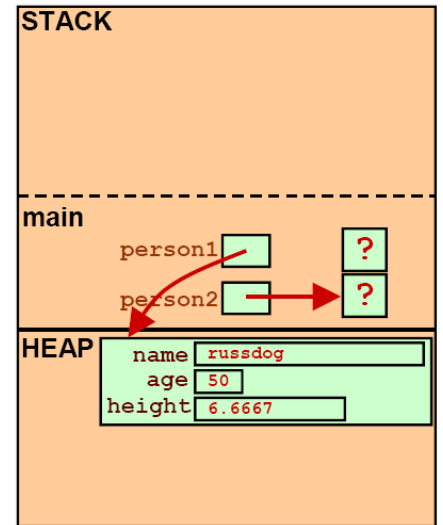


Allocazione dinamica di strutture (5)

```

personT* GetPersonData();
main() {
    personT *person1;
    personT *person2;
    person1 = GetPersonData();
    printf("%s\n", person1->name);
    printf("%d\n", person1->age);
    printf("%.2f\n", person1->height);
}

personT* GetPersonData(){
    personT *x;
    x = (personT *) malloc(sizeof(personT));
    scanf("%s", x->name);
    scanf("%d", &x->age);
    scanf("%lf", &x->height);
    return x;
}
    
```





- Sono vettori i cui elementi sono strutture di un determinato tipo

Esempio

```
typedef struct {
    char nome[30];
    char cognome[30];
    int voto;
} studenteT;

studenteT studenti[100];
```

- Realizza l'allocazione statica del vettore studenti di 100 strutture di tipo studenteT

25



Esempio

```
/* Acquisizione dei dati */
for (i = 0; i < 100; i++) {
    scanf("%s", stud_ing[i].nome);
    scanf("%s", stud_ing[i].cognome);
    scanf("%d", &(stud_ing[i].voto));
}

/* Visualizzazione dei dati */
for (i = 0; i < 100; i++) {
    printf("%s", stud_ing[i].nome);
    printf("%s", stud_ing[i].cognome);
    printf("%d", stud_ing[i].voto);
}
```

26



Esempio

```
studenteT *studenti; /* puntatore al tipo derivato
                    studenteT */

int dim;

scanf("%d", &dim); /* Acquisizione da tastiera della
                  dimensione della struttura dati
                  da allocare */

/* Allocazione dinamica del vettore studenti di dim
   elementi di tipo studenteT */
studenti = (studenteT*)malloc(sizeof(studenteT)*dim);
```

27



Esempio

```
void visualizza_vect(studenteT *pstud, int dim);
main() {
    int i, dim;
    studenteT *pstud_ing;
    scanf("%d", &dim);
    pstud_ing = (studenteT*)malloc(sizeof(studenteT)*dim);
    for (i = 0; i < dim; i++) {
        scanf("%s", pstud_ing[i].nome);
        scanf("%s", pstud_ing[i].cognome);
        scanf("%d", &pstud_ing[i].voto);
    }
    visualizza_vect(pstud_ing, dim);
}

void visualizza_vect(studenteT *pstud, int dim) {
    int i;
    for (i = 0; i < dim; i++) {
        printf("%s %s %d\n", pstud[i].nome, pstud[i].cognome,
              pstud[i].voto);
    }
}
```

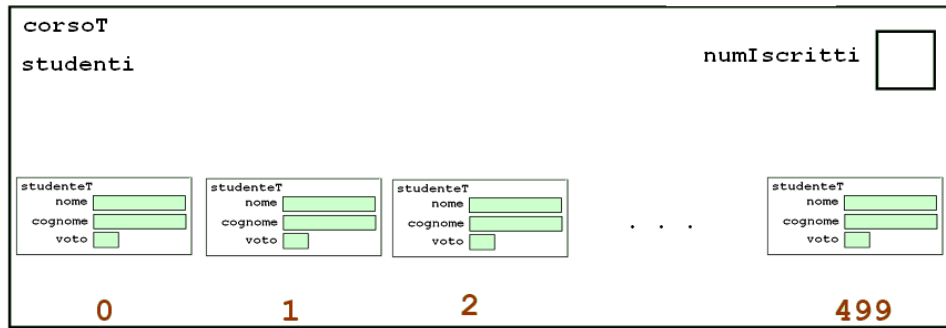
28

Esempi di strutture (1)



- Una struttura che rappresenta un corso contenente
 - Un array di studenti (struttura studenteT)
 - Il numero di studenti iscritti al corso (intero num)

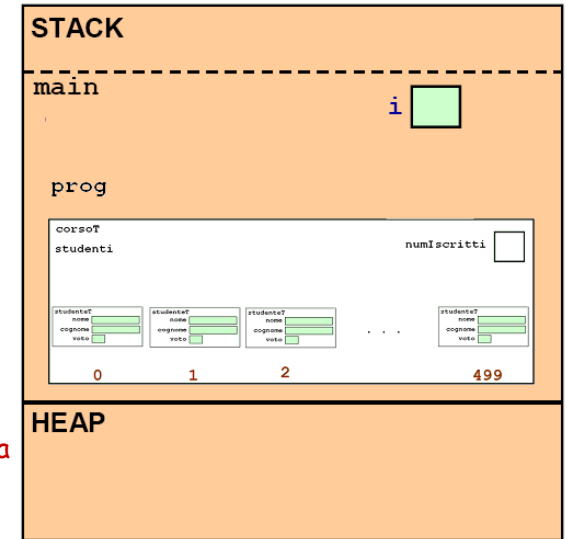
```
#define MAX 500
typedef struct {
    studenteT studenti[MAX];
    int numIscritti;
} corsoT;
```



Esempi di strutture (2)



```
main() {
    corsoT prog;
    int i;
    prog.numIscritti = 0;
    for(i = 0; i < 500; i++) {
        prog.studenti[i] =
            GetStudentData();
        prog.numIscritti++;
    }
}
```



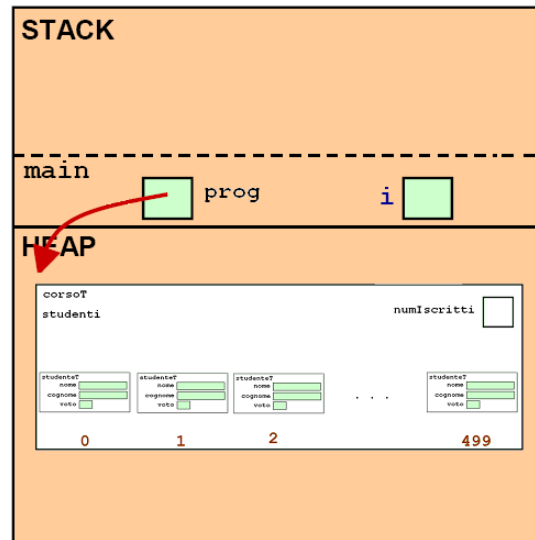
La memoria per la struttura prog è allocata sullo stack

Esempi di strutture (3)



```
main() {
    corsoT *prog;
    int i;
    prog = (corsoT *)
        malloc(sizeof(corsoT));
    prog->numIscritti = 0;
    for(i = 0; i < 500; i++) {
        prog->studenti[i] =
            GetStudentData();
        prog->numIscritti++;
    }
}
```

La memoria per la struttura prog è allocata sull'heap



Esempi di strutture (4)

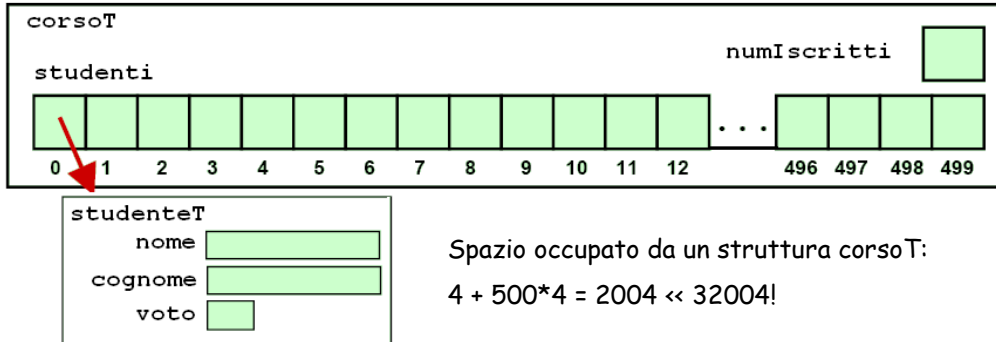


- Quanti byte occupa una struttura di tipo corsoT?
- Il campo numIscritti occupa 4 byte
- L'array di strutture di tipo studenteT occupa i byte necessari per una variabile di tipo studenteT moltiplicato per 500
 - Una variabile di tipo studenteT occupa:
 - 30 byte per il campo nome
 - 30 byte per il campo cognome
 - 4 byte per il campo voto
 - $30+30+4 = 64$ byte
- $= 4+500*64 = 32004!$
- Un modo per risparmiare memoria nel caso in cui solo una parte dell'array venga utilizzato, è quello di dichiarare come campo un array a puntatori alla struttura studenteT
 - In questo modo una struttura studenteT verrà allocata solo quando occorre effettivamente

Esempi di strutture (5)



```
#define MAX 500
typedef struct {
    studenteT *studenti[MAX];
    int numIscritti;
} corsoT;
```



Esercizio



- Definire una struttura che rappresenta un punto su un piano definito dalle coordinate cartesiane
- ```
typedef struct {
 int x;
 int y;
} puntoT;
```
- Definire una funzione che legga da tastiera i valori da assegnare ai campi di una struttura di tipo `puntoT`
  - Versione che non utilizza il passaggio per riferimento
- ```
puntoT inputPunto();
```
- In questo caso si crea e si modifica una struttura locale alla funzione. Ad termine della funzione, mediante l'istruzione `return`, il contenuto della struttura viene copiato nella struttura della funzione chiamante
 - Richiede **copia** in memoria, quindi è inefficiente soprattutto se la struttura è grande
 - Non si preferisce questa versione!
- ```
void inputPunto(puntoT *p); /*pass. per riferimento */
```
- In questo caso la funzione modifica direttamente la variabile della funzione chiamante (`main`)
  - Non richiede copie, viene passato solo il puntatore alla struttura

34

## Esercizio



- Definire una funzione che stampa a video le coordinate cartesiane di un punto definito mediante una struttura di tipo `puntoT`
- ```
void stampa(puntoT p);
```
- Definire una funzione che calcola la distanza tra due punti rappresentati da due strutture di tipo `puntoT`
- ```
float calcola_distanza(puntoT p1, puntoT p2);
```
- Scrivere una funzione che calcola il punto medio tra due punti
- ```
void punto_medio (puntoT p1, puntoT p2, puntoT *p_medio);
```
- Scrivere un programma che legga da tastiera due punti e visualizzi a video le coordinate dei due punti letti, del punto medio e la distanza tra loro

35

Soluzione



```
#include<math.h>
#include<stdio.h>
typedef struct {
    float x;
    float y;
} puntoT;

void inputPunto(puntoT *p);
void stampa(puntoT p);
float calcola_distanza(puntoT p1, puntoT p2);
void punto_medio (puntoT p1, puntoT p2, puntoT *p_medio);

main() {
    puntoT p1,p2,p3;
    float dist;
    inputPunto(&p1);
    inputPunto(&p2);
    printf("Stampa del punto 1\n");
    stampa(p1);
```

36

Soluzione



```
printf("Stampa del punto 2\n");
stampa(p2);
dist = calcola_distanza(p1, p2);
printf("Distanza: %.2f\n",dist);
punto_medio(p1,p2, &p3);
printf("Stampa del punto medio\n");
stampa(p3);
}
void inputPunto(puntoT *p){
printf("Inserire la coordinata x\n");
scanf("%f", &p->x);
printf("Inserire la coordinata y\n");
scanf("%f", &p->y);
}
```

37

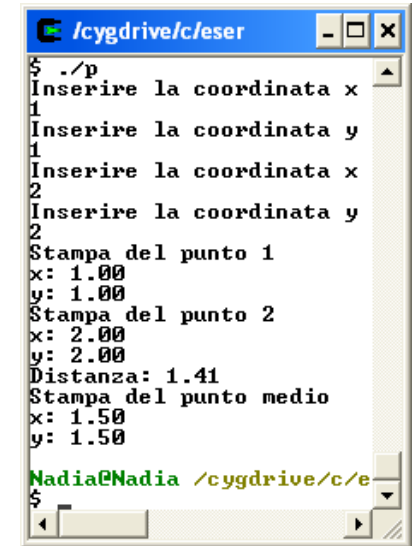
Soluzione



```
void stampa(puntoT p){
printf("x: %.2f\n", p.x);
printf("y: %.2f\n", p.y);
}

float calcola_distanza(puntoT p1,
puntoT p2){
float dist;
dist = sqrt((p1.x-p2.x)*(p1.x-
p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
return dist;
}

void punto_medio (puntoT p1, puntoT
p2, puntoT *p_medio) {
p_medio->x = (p1.x+p2.x)/2;
p_medio->y = (p1.y+p2.y)/2;
}
```



```

/cygdrive/c/leser
$ ./p
Inserire la coordinata x
1
Inserire la coordinata y
1
Inserire la coordinata x
2
Inserire la coordinata y
2
Stampa del punto 1
x: 1.00
y: 1.00
Stampa del punto 2
x: 2.00
y: 2.00
Distanza: 1.41
Stampa del punto medio
x: 1.50
y: 1.50
Nadia@Nadia /cygdrive/c/e
$
```

Esercizio



- Definire una struttura che rappresenta una data, contenente tre campi interi che rappresentano giorno, mese, anno

```
typedef struct {
int giorno, mese, anno;
} dataT;
```

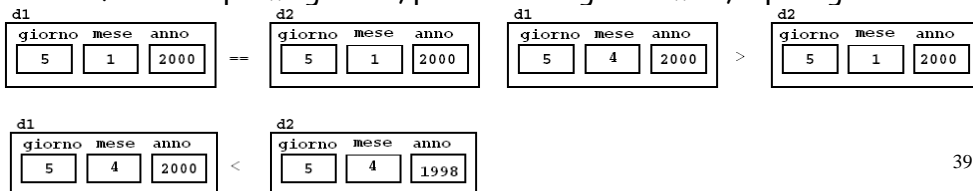
- Scrivere una funzione

```
int confronta_date (dataT d1, dataT d2);
```

che confronta due date passate come parametro e restituisce:

- +1 se la prima è più grande
- 0 se sono uguali
- 1 se la prima è più piccola

- Si confrontano prima gli anni, poi se sono uguali i mesi, e poi i giorni



39

Soluzione



```
typedef struct {
int giorno;
int mese;
int anno;
} dataT;
void inputData(dataT *d);
void stampa(dataT d);
int confronta_date(dataT d, dataT d2);
main() {
dataT d1,d2;
int risultato;
inputData(&d1);
inputData(&d2);
printf("Stampa della data 1\n");
stampa(d1);
printf("Stampa della data 2\n");
stampa(d2);
risultato = confronta_date(d1, d2);
printf("Confronto: %d\n",risultato);
}
-
```

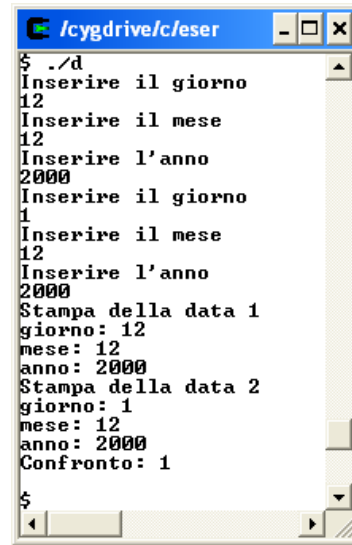
40



```
void inputData(dataT *d){
    printf("Inserire il giorno\n");
    scanf("%d", &d->giorno);
    printf("Inserire il mese\n");
    scanf("%d", &d->mese);
    printf("Inserire l'anno\n");
    scanf("%d", &d->anno);
}

void stampa(dataT d){
    printf("giorno: %d\n",d.giorno);
    printf("mese: %d\n",d.mese);
    printf("anno: %d\n",d.anno);
}

int confronta_date(dataT d1, dataT d2){
    if (d1.anno<d2.anno) return (-1);
    if (d1.anno>d2.anno) return (+1);
    if (d1.mese < d2.mese) return (-1);
    if (d1.mese > d2.mese) return (+1);
    if (d1.giorno < d2.giorno) return (-1);
    if (d1.giorno > d2.giorno) return (+1);
    return (0);
}
```



```
lcygdrive/c/leser
$ ./d
Inserire il giorno
12
Inserire il mese
12
Inserire l'anno
2000
Inserire il giorno
1
Inserire il mese
12
Inserire l'anno
2000
Stampa della data 1
giorno: 12
mese: 12
anno: 2000
Stampa della data 2
giorno: 1
mese: 12
anno: 2000
Confronto: 1
$
```



- Si scriva una funzione baricentro che calcola il baricentro di un dato insieme di punti in uno spazio a dimensione tre.
- L'insieme di punti deve essere memorizzato in un array di dimensione N (il numero di punti puo' essere inferiore a N)
- Si definiscano prima di tutto:
 - la costante N con la direttiva #define (N sia uguale a 10)
 - un tipo di dati strutturato (struct), di nome PuntoT, per rappresentare un punto, composto da tre campi reali (x, y e z)
 - un array di N elementi di tipo PuntoT, di nome NPunti
 - un tipo di dati strutturato (struct), di nome PuntiT, con due campi:
 - un campo P di tipo NPunti
 - un intero n che contiene il numero di punti memorizzati in P



- Definire quindi la funzione baricentro:
 - valore restituito: un dato di tipo PuntoT
 - parametri: una variabile "punti" di tipo PuntiT
 - algoritmo: scandire con un'istruzione iterativa l'array nel campo P di punti, e memorizzare in una variabile b di tipo PuntoT il baricentro

```
#include <stdio.h>
#define N 10
typedef struct {
    float x;
    float y;
    float z;
} PuntoT;
typedef PuntoT NPunti[N];
typedef struct {
    NPunti P;
    int n;
} PuntiT;
PuntoT baricentro(PuntiT);
```



- Esempio di main che consente di verificare la correttezza della funzione baricentro

```
void main(){
    PuntiT punti;
    PuntoT b;
    int i, n_punti;
    float numero;
    printf("Inserisci il numero di punti (max %d):", N);
    scanf("%d", &n_punti);
```



```
printf("\nInserisci le coordinate dei punti:\n\n");
for (i = 0; i < n_punti; i++){
    printf("Punto %d:\n", i+1);
    printf("x: ");
    scanf("%f", &numero);
    punti.P[i].x = numero;
    printf("y: ");
    scanf("%f", &numero);
    punti.P[i].y = numero;
    printf("z: ");
    scanf("%f", &numero);
    punti.P[i].z = numero;
}
punti.n = n_punti;
b = baricentro(punti);
printf("\nIl baricentro dei punti e'");
printf("(%.3f; %.3f; %.3f)\n\n", b.x, b.y, b.z);
}
```

45



```
PuntoT baricentro(PuntiT punti) {
    int i;
    PuntoT b;
    b.x = 0;
    b.y = 0;
    b.z = 0;
    for (i = 0; i < punti.n; i++) {
        b.x = b.x + punti.P[i].x;
        b.y = b.y + punti.P[i].y;
        b.z = b.z + punti.P[i].z;
    }
    b.x = b.x / punti.n;
    b.y = b.y / punti.n;
    b.z = b.z / punti.n;
    return b;
}
```

46