

Tipi di dato strutturato

Adrian	Allincai	2900	21.0
Marco	Albergamo	31000	27.0
Leonardo	Amico	8000	21.0
Massimiliano	Arpino	1500	19.2
Valerio	Zerillo	45000	34.5

Fondamenti di Informatica L-A

Tipi strutturati: il Record

Un record è un insieme finito di elementi, in generale non omogeneo:

- il numero degli elementi è rigidamente fissato a priori.
- gli elementi possono essere di tipo diverso.
- il tipo di ciascun elemento componente (campo) è prefissato.

Ad esempio:

NOME	COGNOME	REDDITO	ALIQUOTA
------	---------	---------	----------

Formalmente:

Il record è un tipo strutturato il cui dominio si ottiene mediante prodotto cartesiano:

dati n insiemi, $A_{c1}, A_{c2}, \dots, A_{cn}$, il prodotto cartesiano tra essi:

$$A_{c1} \times A_{c2} \times \dots \times A_{cn}$$

consente di definire un tipo di dato strutturato (il record) i cui elementi sono n -ple ordinate:

$$(a_{c1}, a_{c2}, \dots, a_{cn})$$

dove $a_{ci} \in A_{ci}$.

Ad esempio:

Il numero complesso può essere definito attraverso il prodotto cartesiano $\mathbb{R} \times \mathbb{R}$.

Fondamenti di Informatica L-A

Il tipo struct

Accesso ai campi:

Per accedere (e manipolare) i singoli campi di un record si usa la notazione **postfissa**:

<id-variabile>.<componente>

indica il valore del campo **<componente>** della variabile **<id-variabile>**.

→ I singoli campi possono essere manipolati con gli operatori previsti per il tipo ad essi associato.

```
struct {
    char Nome[20],
          Cognome[20];
    int Reddito;
    float Aliquota;
} c1;
```

c1.Nome

```
strcpy(c1.Nome, "Valentina");
strcpy(c1.Cognome, "Florio");
c1.Reddito = 7000+18000;
c1.Aliquota = 23.0;
```

Fondamenti di Informatica L-A

Il Record

Esempio:

Si vuole rappresentare adeguatamente l'astrazione "contribuente", caratterizzata dai seguenti attributi:

- Nome,
- Cognome,
- Reddito,
- Aliquota

Con gli strumenti visti finora, per ogni contribuente è necessario introdurre 4 variabili:

```
char Nome[20], Cognome[20];
int Reddito;
float Aliquota;
```

È una soluzione scomoda e non "astratta": le quattro variabili sono indipendenti tra di loro.

→ È necessario un costrutto che consenta l'aggregazione dei 4 attributi nell'astrazione "contribuente": il record.

Fondamenti di Informatica L-A

Il Record in C: struct

Collezioni con un numero finito di campi (anche disomogenei) sono realizzabili in C mediante il costruttore di tipo strutturato **struct**.

Definizione di **variabile di tipo struct**:

```
struct {
    <lista def. campi>
} <id-variabile>;
```

```
struct {
    char Nome[20],
          Cognome[20];
    int Reddito;
    float Aliquota;
} c1;
```

dove:

- <lista def. campi>** è l'insieme delle definizioni dei campi componenti, costruita usando le stesse regole sintattiche della definizione di variabili:

```
<tipo1> <campo1>;
<tipo2> <campo2>;
...
<tipoN> <campoN>;
```

```
char Nome[20],
      Cognome[20];
int Reddito;
float Aliquota;
```

- <id-variabile>** è l'identificatore della variabile di tipo record così definita.

Fondamenti di Informatica L-A

Il tipo struct: assegnamento

```
struct {
    char Nome[20],
          Cognome[20];
    int Reddito;
    float Aliquota;
} c1, c2;
```

```
strcpy(c1.Nome, "Valentina");
strcpy(c1.Cognome, "Florio");
c1.Reddito = 7000+18000;
c1.Aliquota = 23.0;
```

c2=c1;

Operatori:

L'unico operatore previsto per dati di tipo struct è l'operatore di **assegnamento (=)**:

→ è possibile l'assegnamento diretto tra record di tipo equivalente.

Fondamenti di Informatica L-A

Inizializzazione di record:

È possibile inizializzare i record in fase di definizione.

Ad esempio:

```
struct { char Nome[20],
        Cognome[20];
        int Reddito;
        float Aliquota;
} c1 = {"Valentina", "Florio", 7000+18000, 23.0};
```

Fondamenti di Informatica L-A

typedef struct

Il costruttore struct può essere utilizzato per dichiarare tipi non primitivi basati sul record:

```
Dichiarazione di tipo strutturato record:
typedef struct {
    <lista dichiarazioni campi>
} <id-tipo>;
```

dove:

- <lista definizioni campi> è l'insieme delle definizioni dei campi componenti;
- <id-tipo> è l'identificatore del nuovo tipo.

Ad esempio:

```
typedef struct { char Nome[20],
                Cognome[20];
                int Reddito;
                float Aliquota;
            } contribuente;

contribuente c1, c2, c3;
strcpy(c1.Nome, "Valentina");
...
```

< lista dichiarazioni campi >

< id-tipo >

Fondamenti di Informatica L-A

typedef struct

Nota: non è consentito definire due tipi diversi con lo stesso nome. Però è possibile che un campo del record abbia lo stesso nome di una variabile. Come Reddito, nell'esempio:

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

contribuente c1, c2, c3;
```

```
unsigned int Reddito = 7000;
```

```
c1.Reddito = Reddito + 18000;
```

variabile.campo variabile
(variabile strutturata) (intero senza segno)

Fondamenti di Informatica L-A

Il tipo struct in sintesi

Riassumendo, la sintassi da adottare è:

```
[typedef] struct {
    <tipo-1> <nome_campo-1>;
    <tipo-2> <nome_campo-2>;
    ...
    <tipo-N> <nome_campo-N>;
} <nome>;
```

Vincoli:

- <nome-campo-i> è un identificatore stabilito che individua il campo i-esimo;
- <tipo-i> è un qualsiasi tipo, semplice o strutturato.
- <nome> è l'identificatore della struttura (o del tipo, se si usa typedef)

Uso:

- la struttura è una collezione di un numero fissato di elementi di vario tipo (<tipo-campo-i>);
- il singolo campo <nome-campo-i> di un record R è individuato mediante la notazione: R.<nome-campo-i>;
- se due strutture di dati di tipo struct hanno lo stesso tipo, allora è possibile l'assegnamento diretto.

Fondamenti di Informatica L-A

Esercizio sui record

Realizzare un programma che, lette da input le coordinate di un punto P del piano, sia in grado di applicare a P alcune trasformazioni geometriche (traslazione, e proiezioni sui due assi).

→ Rappresentiamo il punto del piano cartesiano mediante una struct di due campi (float), ciascuno associato a una particolare coordinata:

```
typedef struct {
    float x;
    float y;
} punto;

punto P; /* P è una variabile di tipo punto */
```

P

x
y

Fondamenti di Informatica L-A

Esercizio sui record

```
#include <stdio.h>

main()
{ typedef struct{float x,y;} punto;
  punto P;
  unsigned int op;
  float Dx, Dy;
  /* lettura delle coordinate da input in P: */
  printf("ascissa? ");
  scanf("%f", &P.x);
  printf("ordinata? ");
  scanf("%f", &P.y);

  /* lettura dell'operazione richiesta assumendo le convenzioni:
  0: termina
  1: proietta sull'asse x
  2: proietta sull'asse y
  3: trasla di Dx, Dy */
  scanf("%d", &op);
  switch(op)
  {
    case 0: break;
    case 1: P.x = P.x + Dx; break;
    case 2: P.y = P.y + Dy; break;
    case 3: P.x = P.x + Dx; P.y = P.y + Dy; break;
  }
}
```

P

x
y

Fondamenti di Informatica L-A

Esercizio sui record

```

switch (op)
{
  case 1:   proietta sull'asse x   P x
  case 2:   proietta sull'asse y
  case 3:   printf("%s", "Traslazione?") y
            scanf("%f%f", &Dx, &Dy);
            calcola
            nuove
            coordinate

  default: printf("errore!");
}
printf( "Nuove coordinate:
        %f\t%f\n" coordinate );
}

```

Fondamenti di Informatica L-A

Vettori e record

Non ci sono vincoli riguardo al tipo degli elementi di un vettore: si possono realizzare anche vettori di record (strutture tabellari).

Ad esempio:

```

typedef struct {
  char  Nome[20];
  char  Cognome[20];
  int   Reddito;
  float Aliquota;
} Contribuente;

contribuente archivio[1000];

```

→ archivio è un vettore di 1000 elementi, ciascuno dei quali è di tipo contribuente

→ abbiamo realizzato un vettore di record, o *struttura tabellare*.

	Nome	Cognome	Reddito	Aliquota
0				
...				
999				

Fondamenti di Informatica L-A

Vettori e Record

Allo stesso modo, si possono fare **record di vettori** e **record di record**.

Ad esempio:

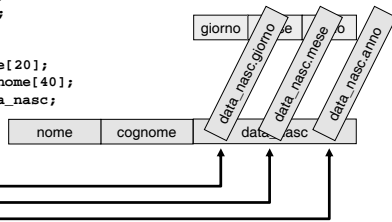
```

typedef struct{
  int giorno;
  int mese;
  int anno;
} data;

typedef struct{
  char nome[20];
  char cognome[40];
  data data_nasc;
} persona;

persona P;
...
P.data_nasc.giorno=25;
P.data_nasc.mese=3;
P.data_nasc.anno=1992;
...

```



Fondamenti di Informatica L-A

Esercizio

Scrivere un programma che acquisisca i dati relativi agli studenti di una classe:

- nome
- cognome
- voti: rappresenta i voti dello studente in 3 materie (italiano, matematica, inglese);

Il programma deve successivamente **calcolare** e **stampare**, per ogni studente, la **media dei voti ottenuti nelle 3 materie**.

Introduciamo un tipo di dato per rappresentare il generico studente:

```

typedef struct {
  ...
} studente;

```

La classe è rappresentata da un vettore di studenti:

```

studente classe[20];

```

Fondamenti di Informatica L-A

```

#include <stdio.h>
#define ita 0
#define mat 1
#define ing 2
#define N 20
typedef struct{
  ...
} studente;

main()
{
  studente classe[N];
  float m;
  int i, int j;
  /* lettura dati */
  for(i=0; i<N; i++)
  {
    fflush(stdin);
    gets( nome i-mo studente );
    gets( cognome i-mo studente );
    for(j=ita; j<=ing; j++)
      scanf( voto j-ma materia i-mo studente );
  }
}

```

Fondamenti di Informatica L-A

```

/* continua.. stampa delle medie */
for( i=0; i<N; i++ ) // N studenti
{
  for( m=0, j=ita; j<=ing; j++ ) // ita=0; ing=2
    aggiorna il totale dei voti... // m accumulatore
    printf( "media di %s %s: %f\n",
            media di <nome> <cognome>: <media> );
}
}

```

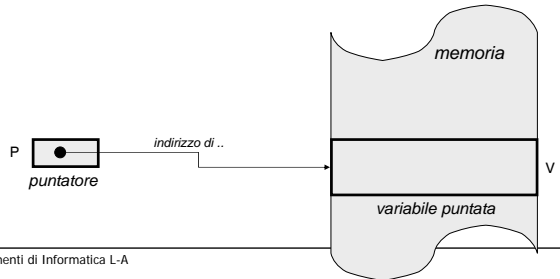
Fondamenti di Informatica L-A

Il puntatore

È un tipo di dato scalare, che consente di rappresentare gli **indirizzi** delle variabili allocate in memoria.

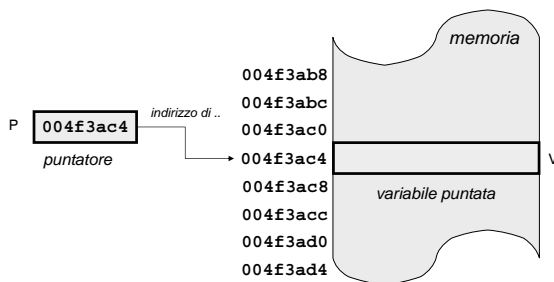
Dominio:

Il dominio di una variabile di tipo puntatore è un insieme di indirizzi: il valore di una variabile P di tipo puntatore può essere l'**indirizzo** di un'altra variabile (variabile **puntata**).



Puntatori

Esempio



Il puntatore in C

In C i puntatori si definiscono mediante il costruttore *.

Definizione di una variabile puntatore:

```
<TipoElementoPuntato> * <NomePuntatore>;
```

dove:

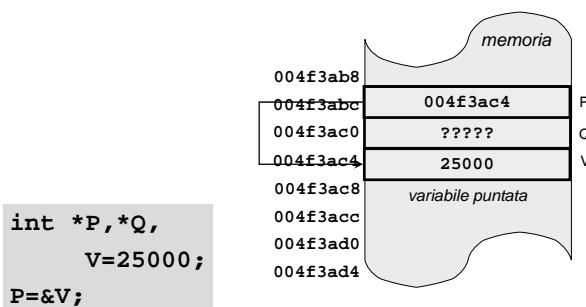
- <TipoElementoPuntato> è il tipo della variabile puntata
- <NomePuntatore> è il nome della variabile di tipo puntatore
- il simbolo * è il costruttore del tipo puntatore.

Ad esempio:

```
int *P; /*P è un puntatore a intero */
```

Fondamenti di Informatica L-A

Esempio



Il puntatore in C

Operatori:

- **Assegnamento (=)**: è possibile l'assegnamento tra puntatori (dello stesso tipo). È disponibile la costante NULL, per indicare l'indirizzo nullo (0).
- **Dereferencing (*)**: è un operatore unario. Si applica a un puntatore e restituisce il valore contenuto nella cella puntata => serve per accedere alla variabile puntata.
- **Operatori aritmetici** (vedi vettori e puntatori).
- **Operatori relazionali**: >, <, ==, !=
- **Operatore indirizzo &**: si applica ad una variabile e restituisce l'indirizzo della cella di memoria nella quale è allocata la variabile.

Ad esempio:

```
int *p1, *p2;
int A;
p1 = &A;
*p1 = 127;
p2 = p1;
p1 = NULL; /* NULL è la costante che vale 0 e
denota il puntatore "nullo" */
```

Fondamenti di Informatica L-A

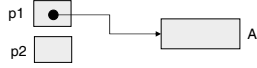
Il puntatore in C

Operatori:

- **Assegnamento (=)**: è possibile l'assegnamento tra puntatori (dello stesso tipo). È disponibile la costante NULL, per indicare l'indirizzo nullo (0).
- **Dereferencing (*)**: è un operatore unario. Si applica a un puntatore e restituisce il valore contenuto nella cella puntata => serve per accedere alla variabile puntata.
- **Operatori aritmetici** (vedi vettori e puntatori).
- **Operatori relazionali**: >, <, ==, !=
- **Operatore indirizzo (&)**: si applica ad una variabile e restituisce l'indirizzo della cella di memoria nella quale è allocata la variabile.

Ad esempio:

```
int *p1, *p2;
int A;
p1 = &A;
*p1 = 127;
p2 = p1;
p1 = NULL;
```



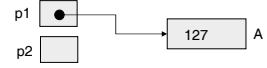
Il puntatore in C

Operatori:

- **Assegnamento (=)**: è possibile l'assegnamento tra puntatori (dello stesso tipo). È disponibile la costante NULL, per indicare l'indirizzo nullo (0).
- **Dereferencing (*)**: è un operatore unario. Si applica a un puntatore e restituisce il valore contenuto nella cella puntata => serve per accedere alla variabile puntata.
- **Operatori aritmetici** (vedi vettori e puntatori).
- **Operatori relazionali**: >, <, ==, !=
- **Operatore indirizzo (&)**: si applica ad una variabile e restituisce l'indirizzo della cella di memoria nella quale è allocata la variabile.

Ad esempio:

```
int *p1, *p2;
int A;
p1 = &A;
*p1 = 127;
p2 = p1;
p1 = NULL;
```



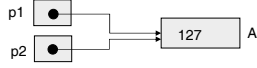
Il puntatore in C

Operatori:

- **Assegnamento (=)**: è possibile l'assegnamento tra puntatori (dello stesso tipo). È disponibile la costante NULL, per indicare l'indirizzo nullo (0).
- **Dereferencing (*)**: è un operatore unario. Si applica a un puntatore e restituisce il valore contenuto nella cella puntata => serve per accedere alla variabile puntata.
- **Operatori aritmetici** (vedi vettori e puntatori).
- **Operatori relazionali**: >, <, ==, !=
- **Operatore indirizzo (&)**: si applica ad una variabile e restituisce l'indirizzo della cella di memoria nella quale è allocata la variabile.

Ad esempio:

```
int *p1, *p2;
int A;
p1 = &A;
*p1 = 127;
p2 = p1;
p1 = NULL;
```



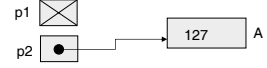
Il puntatore in C

Operatori:

- **Assegnamento (=)**: è possibile l'assegnamento tra puntatori (dello stesso tipo). È disponibile la costante NULL, per indicare l'indirizzo nullo (0).
- **Dereferencing (*)**: è un operatore unario. Si applica a un puntatore e restituisce il valore contenuto nella cella puntata => serve per accedere alla variabile puntata.
- **Operatori aritmetici** (vedi vettori e puntatori).
- **Operatori relazionali**: >, <, ==, !=
- **Operatore indirizzo (&)**: si applica ad una variabile e restituisce l'indirizzo della cella di memoria nella quale è allocata la variabile.

Ad esempio:

```
int *p1, *p2;
int A;
p1 = &A;
*p1 = 127;
p2 = p1;
p1 = NULL;
```



Il puntatore in C: * e &

Operatore Indirizzo &:

- & si applica solo ad oggetti che esistono in memoria (quindi, già definiti).
- & non è applicabile ad espressioni.

Operatore Dereferencing *:

- consente di accedere ad una variabile specificandone l'indirizzo
- l'indirizzo rappresenta un modo alternativo al nome (*alias*) per accedere e manipolare la variabile:

Ad esempio:

```
float *p;
float R, A;

p=&A; /* *p è un alias di A*/
R=2;
*p=3.14*R; /* A è modificato */
```



Il puntatore in C: * e &

Operatore Indirizzo &:

- & si applica solo ad oggetti che esistono in memoria (quindi, già definiti).
- & non è applicabile ad espressioni.

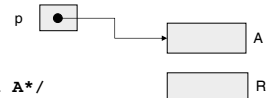
Operatore Dereferencing *:

- consente di accedere ad una variabile specificandone l'indirizzo
- l'indirizzo rappresenta un modo alternativo al nome (*alias*) per accedere e manipolare la variabile:

Ad esempio:

```
float *p;
float R, A;

p=&A; /* *p è un alias di A*/
R=2;
*p=3.14*R; /* A è modificato */
```



Il puntatore in C: * e &

Operatore Indirizzo &:

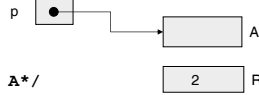
- & si applica solo ad oggetti che esistono in memoria (quindi, già definiti).
- & non è applicabile ad espressioni.

Operatore Dereferencing *:

- consente di accedere ad una variabile specificandone l'indirizzo
- l'indirizzo rappresenta un modo alternativo al nome (*alias*) per accedere e manipolare la variabile:

Ad esempio:

```
float *p;
float R, A;
```



```
p=&A; /* *p è un alias di A*/
R=2;
*p=3.14*R; /* A è modificato */
```

Fondamenti di Informatica L-A

Il puntatore in C: * e &

Operatore Indirizzo &:

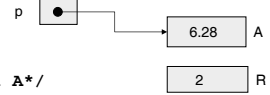
- & si applica solo ad oggetti che esistono in memoria (quindi, già definiti).
- & non è applicabile ad espressioni.

Operatore Dereferencing *:

- consente di accedere ad una variabile specificandone l'indirizzo
- l'indirizzo rappresenta un modo alternativo al nome (*alias*) per accedere e manipolare la variabile:

Ad esempio:

```
float *p;
float R, A;
```



```
p=&A; /* *p è un alias di A*/
R=2;
*p=3.14*R; /* A è modificato */
```

Fondamenti di Informatica L-A

Puntatore come costruttore di tipo

Il costruttore di tipo "*" può essere anche usato per dichiarare tipi non primitivi basati sul puntatore.

Dichiarazione di un tipo puntatore:

```
typedef <TipoElementoPuntato> *<NomeTipo>;
```

- <TipoElementoPuntato> è il tipo della variabile puntata
- <NomePuntatore> è il nome del tipo dichiarato.

Ad esempio:

```
typedef float *tpf;
tpf p;
float f;
p=&f;
*p=0.56;
```

Fondamenti di Informatica L-A

Puntatori: controlli di tipo

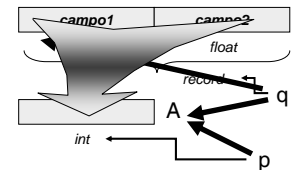
Nella definizione di un puntatore è **necessario** indicare il tipo della variabile puntata.

→ il compilatore **può** effettuare controlli statici sull'uso dei puntatori.

Esempio:

```
typedef struct{int campo1; float campo2;}record;
int A, *p;
record X, *q;

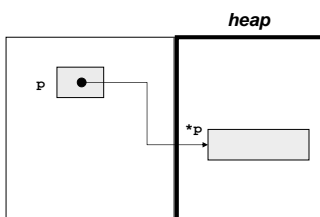
p = &A;
q = p;
/* warning! */
q = &X;
*p = *q;
/* errore! */
```



→ Viene segnalato dal compilatore (**warning**) il tentativo di utilizzo congiunto di puntatori a tipi differenti.

Fondamenti di Informatica L-A

Variabili dinamiche



Fondamenti di Informatica L-A

Variabili automatiche e dinamiche

In C è possibile classificare le variabili in base al loro tempo di vita.

Due categorie:

- variabili **automatiche**
- variabili **dinamiche**

Variabili automatiche:

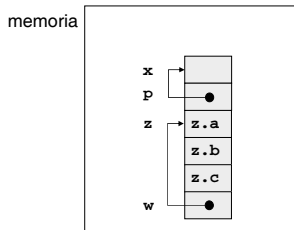
- L'allocazione e la deallocazione di variabili automatiche è effettuata **automaticamente** dal sistema (senza l'intervento del programmatore).
- Ogni variabile automatica ha un **nome**, attraverso il quale la si può riferire.
- Il programmatore non ha la possibilità di influire sul tempo di vita di variabili automatiche.

→ tutte le variabili viste finora rientrano nella categoria delle **variabili automatiche**.

Fondamenti di Informatica L-A

Variabili automatiche

```
int *p, x;
struct {int a, b, c;} z, *w;
```



Fondamenti di Informatica L-A

Variabili dinamiche

Variabili dinamiche:

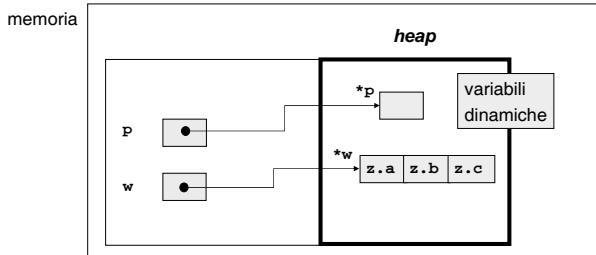
- Le variabili dinamiche devono essere allocate e deallocate **esplicitamente** dal programmatore.
- L'area di memoria in cui vengono allocate le variabili dinamiche si chiama **heap**.
- Le variabili dinamiche non hanno un **identificatore**, ma possono essere riferite soltanto attraverso il loro indirizzo (mediante i **puntatori**).
- Il tempo di vita delle variabili dinamiche è l'intervallo di tempo che intercorre tra l'allocazione e la deallocazione (che sono impartite esplicitamente dal programmatore).

Fondamenti di Informatica L-A

Variabili dinamiche

```
int *p;
struct {int a, b, c;} *w;
```

variabili automatiche



Fondamenti di Informatica L-A

Variabili Dinamiche in C

Il C prevede funzioni standard di allocazione e deallocazione per variabili dinamiche:

- Allocazione:** `malloc`
- Deallocazione:** `free`

`malloc` e `free` sono definite a livello di **sistema operativo**, mediante la libreria standard `<stdlib.h>` (da includere nei programmi che le usano).

Fondamenti di Informatica L-A

Variabili Dinamiche

Allocazione di variabili dinamiche:

La memoria dinamica viene allocata con la funzione standard `malloc`. La **sintassi** da usare è:

```
punt = (tipodato *)malloc(sizeof(tipodato));
```

dove:

- `tipodato` è il tipo della variabile puntata
- `punt` è una variabile di tipo `tipodato *`
- `sizeof()` è una funzione standard che calcola il numero di byte che occupa il dato specificato come argomento
- è necessario convertire esplicitamente il tipo del valore ritornato (*casting*):
`(tipodato *) malloc(..)`

Significato:

La `malloc`

- provoca la creazione di una variabile dinamica nell'**heap** e
- restituisce l'**indirizzo** della variabile creata.

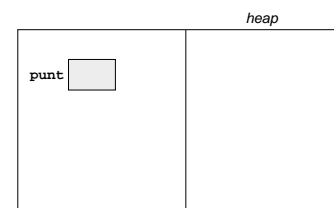
Fondamenti di Informatica L-A

Variabili Dinamiche

Esempio:

```
#include <stdlib.h>
typedef int *tp;
tp punt;
...
punt=(tp)malloc(sizeof(int));
```

```
*punt=12;
```

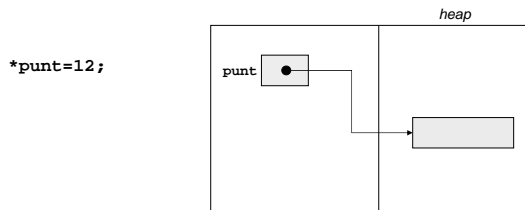


Fondamenti di Informatica L-A

Variabili Dinamiche

Esempio:

```
#include <stdlib.h>
typedef int *tp;
tp punt;
...
➡ punt=(tp )malloc(sizeof(int));
```

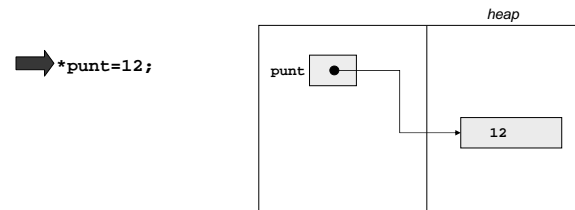


Fondamenti di Informatica L-A

Variabili Dinamiche

Esempio:

```
#include <stdlib.h>
typedef int *tp;
tp punt;
...
➡ punt=(tp )malloc(sizeof(int));
```



Fondamenti di Informatica L-A

Variabili dinamiche

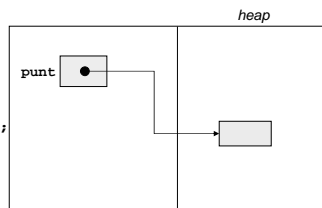
Deallocazione:

Si rilascia la memoria allocata dinamicamente con:
free(punt);

dove punt e' l'indirizzo della variabile da deallocare.

➔ Dopo questa operazione, la cella di memoria occupata da *punt viene liberata: *punt non esiste piu'.

```
Esempio:
#include <stdlib.h>
typedef int *tp;
tp punt;
...
punt=(tp )malloc(sizeof(int));
*punt=12;
...<uso di punt>...
➡ free(punt);
```

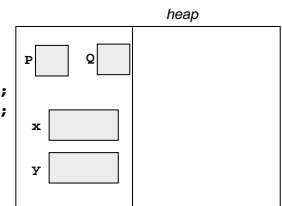


Fondamenti di Informatica L-A

Puntatori e variabili dinamiche: esempio

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

```
main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

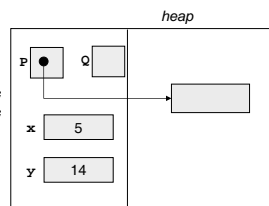


Fondamenti di Informatica L-A

Puntatori e variabili dinamiche: esempio

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

```
main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  ➡ P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

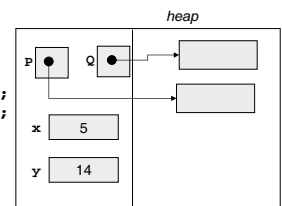


Fondamenti di Informatica L-A

Puntatori e variabili dinamiche: esempio

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

```
main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  ➡ P=(int *)malloc(sizeof(int));
  ➡ Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

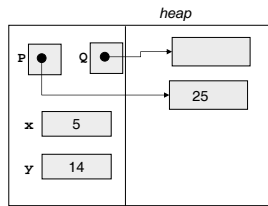


Fondamenti di Informatica L-A

Puntatori e variabili dinamiche: esempio

```
#include <stdio.h>

main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

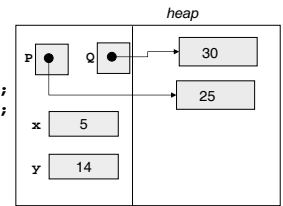


Fondamenti di Informatica L-A

Puntatori e variabili dinamiche: esempio

```
#include <stdio.h>

main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

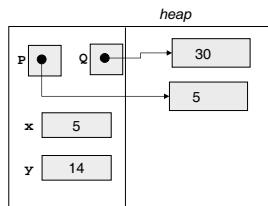


Fondamenti di Informatica L-A

Puntatori e variabili dinamiche: esempio

```
#include <stdio.h>

main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

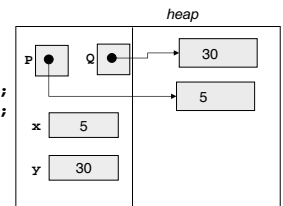


Fondamenti di Informatica L-A

Puntatori e variabili dinamiche: esempio

```
#include <stdio.h>

main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

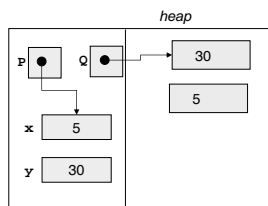


Fondamenti di Informatica L-A

Puntatori e variabili dinamiche: esempio

```
#include <stdio.h>

main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```



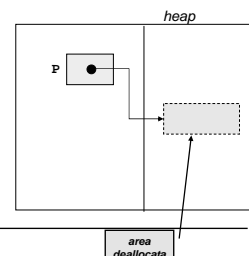
→ l'ultimo assegnamento ha come effetto collaterale la **perdita dell'indirizzo** di una variabile dinamica (quella precedentemente referenziata da P) che rimane allocata ma **non è più utilizzabile!**

Fondamenti di Informatica L-A

Problemi legati all'uso dei Puntatori

- 1. Aree inutilizzabili:**
Possibilità di perdere l'indirizzo di aree di memoria allocate al programma che quindi non sono più accessibili. (v. esempio precedente).
- 2. Riferimenti pendenti (dangling references):**
Possibilità di fare riferimento ad aree di memoria non più allocate.

```
Ad esempio:
int *P;
P = (int *) malloc(sizeof(int));
...
free(P);
*P = 100; /* Da non fare! */
```



Fondamenti di Informatica L-A

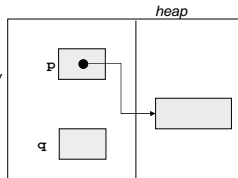
Problemi legati all'uso dei Puntatori

3. Aliasing:

Possibilita' di riferire la stessa variabile con puntatori diversi.

Ad esempio:

```
int *p, *q;
p=(int *)malloc(sizeof(int));
*p=3;
q=p; /*p e q puntano alla stessa
      variabile */
*q = 10; /*anche *p e` cambiato! */
```



Fondamenti di Informatica L-A

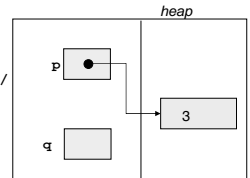
Problemi legati all'uso dei Puntatori

3. Aliasing:

Possibilita' di riferire la stessa variabile con puntatori diversi.

Ad esempio:

```
int *p, *q;
p=(int *)malloc(sizeof(int));
*p=3;
q=p; /*p e q puntano alla stessa
      variabile */
*q = 10; /*anche *p e` cambiato! */
```



Fondamenti di Informatica L-A

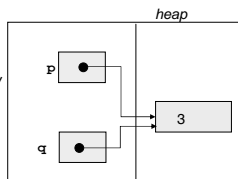
Problemi legati all'uso dei Puntatori

3. Aliasing:

Possibilita' di riferire la stessa variabile con puntatori diversi.

Ad esempio:

```
int *p, *q;
p=(int *)malloc(sizeof(int));
*p=3;
q=p; /*p e q puntano alla stessa
      variabile */
*q = 10; /*anche *p e` cambiato! */
```



Fondamenti di Informatica L-A

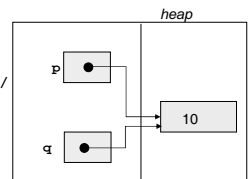
Problemi legati all'uso dei Puntatori

3. Aliasing:

Possibilita' di riferire la stessa variabile con puntatori diversi.

Ad esempio:

```
int *p, *q;
p=(int *)malloc(sizeof(int));
*p=3;
q=p; /*p e q puntano alla stessa
      variabile */
*q = 10; /*anche *p e` cambiato! */
```



Fondamenti di Informatica L-A

Puntatori a puntatori

Un puntatore può *puntare* a variabili di tipo qualunque (semplici o strutturate):

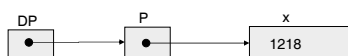
→ può *puntare* anche a un puntatore:

```
[ typedef ] TipoDato    **TipoPunt;
```

Ad esempio:

```
int x, *P, **DP;
P = &x;
DP = &P;
```

→ **DP=1218;



→ DP è un **doppio puntatore** (o *handle*): *dereferenziando 2 volte* DP, si accede alla variabile puntata dalla "catena" di riferimenti.

Fondamenti di Informatica L-A

Vettori & Puntatori

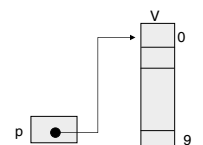
Nel linguaggio C, i vettori sono rappresentati mediante puntatori:

→ il **nome** di una variabile di tipo vettore denota l'**indirizzo del primo elemento** del vettore.

Ad esempio:

```
float V[10]; /*V è una costante di tipo puntatore:
             V equivale a &V[0];
             V e` un puntatore (costante!) a float
             */
```

```
float *p;
p=V; /* p punta a V[0] */
*p=0.15; /* equivale a V[0]=0.15 */
```



Fondamenti di Informatica L-A

Vettori & Puntatori

Nel linguaggio C, i vettori sono rappresentati mediante puntatori:

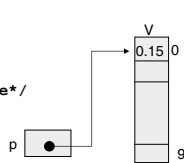
il nome di una variabile di tipo vettore denota l'indirizzo del primo elemento del vettore.

Ad esempio:

```
float V[10]; /*V è una costante di tipo puntatore:
             V equivale a &V[0];
             V è un puntatore (costante!) a float
             */
```

```
float *p;
p=V; /* p punta a V[0] */
/* *p=0.15; /* equivale a V[0]=0.15 */
```

V = p; /*ERRORE! V è un puntatore costante*/



Fondamenti di Informatica L-A

Operatori aritmetici su puntatori a vettori

Nel linguaggio C, gli elementi di un vettore vengono allocati in memoria in parole consecutive (cioè, in celle fisicamente adiacenti), la cui dimensione dipende dal tipo dell'elemento.

→ Conoscendo l'indirizzo del primo elemento e la dimensione dell'elemento, è possibile calcolare l'indirizzo di qualunque elemento del vettore:

Operatori aritmetici (somma e sottrazione) su puntatori a vettori:

Se V e W sono puntatori ad elementi di vettori ed i è un intero:

- (V+i) restituisce l'indirizzo dell'elemento spostato di i posizioni in avanti rispetto a quello puntato da V;
- (V-i) restituisce l'indirizzo dell'elemento spostato di i posizioni all'indietro rispetto a quello puntato da V;
- (V-W) restituisce l'intero che rappresenta il numero di elementi compresi tra V e W.

Fondamenti di Informatica L-A

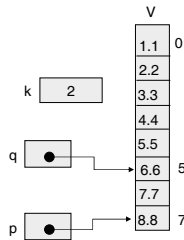
Operatori aritmetici su puntatori a vettori

Esempio:

```
#include <stdio.h>

main()
{ float V[8]={1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8};
  int k;
  float *p, *q;
  p=V+7;
  q=p-2;
  k=p-q;
  printf("%f, %f, %d\n", *p, *q, k);
}
```

→ Stampa:



Fondamenti di Informatica L-A

Vettori e Puntatori

Durante l'esecuzione di ogni programma C, ogni riferimento ad un elemento di un vettore è tradotto in un puntatore dereferenziato; per esempio:

V[0]	viene tradotto in	*(V)
V[1]	viene tradotto in	*(V + 1)
V[i]	viene tradotto in	*(V + i)
V[expr]	viene tradotto in	*(V + expr)

Esempio:

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

```
main ()
{ char a[ ] = "0123456789"; /* a è un vettore di 10 char */
  int i = 5;
  printf("%c%c%c%c%c\n", a[i], a[5], i[a], 5[a], (i-1)[a]); /* !!! */
}
```

→ Stampa:

NB: Per il compilatore a[i] e i[a] sono lo stesso elemento, perché viene sempre eseguita la conversione: a[i] => *(a+i) (senza eseguire alcun controllo né su a, né su i).

Fondamenti di Informatica L-A

Complementi sui puntatori

Vettori di puntatori:

Il costruttore [] ha precedenza rispetto al costruttore *. Quindi:

```
char *a[10]; equivale a char *(a[10]);
```

→ a è un vettore di 10 puntatori a carattere.

NB: Per un puntatore ad un vettore di caratteri è necessario forzare la precedenza (con le parentesi): char (* a) [10];

Puntatori a strutture:

È possibile utilizzare i puntatori per accedere a variabili di tipo struct, tenendo conto che il punto della notazione postfissa ha la precedenza sull'operatore di dereferencing *.

Esempio:

```
typedef struct{ int Campo_1, Campo_2; } TipoDato;
TipoDato S, *P;
P = &S;
```

```
(*P).Campo1=75; /* assegnamento della costante 75 al Campo1 della
                 struct puntata da P* (e' necessario usare le
                 parentesi) */
```

Operatore ->:

L'operatore -> consente di accedere ad un campo di una struttura referenziata da un puntatore in modo più sintetico:

```
P->Campo1=75;
```

Fondamenti di Informatica L-A

Esercizio

Si vuole realizzare un programma che, data da input una sequenza di N parole (ognuna, al massimo, di 20 caratteri), stampi in ordine inverso le parole date, ognuna "ribaltata" (cioè, stampando i caratteri in ordine inverso: dall'ultimo al primo). Si supponga che N non sia noto a priori, ma venga fornito da input. Utilizzare una struttura dinamica.

Progetto dei dati.

Memorizziamo le parole in un vettore di N. stringhe che verra' allocato dinamicamente.

```
typedef char parola[21];
/* tipo associato alla singola parola */
parola *p;
/* puntatore per l'accesso
   al vettore delle parole */
```

Fondamenti di Informatica L-A

Soluzione

```
#include <stdio.h>
#include <stdlib.h>
typedef char parola[21];

main()
{ parola w, *p;
  int i, j, N;

  printf("Quante parole? ");
  scanf("%d", &N);
  fflush(stdin);
  /* allocazione del vettore */
  = malloc ;
  /* lettura della sequenza */
  for(i=0; i<N; i++)
    gets( );
```

Fondamenti di Informatica L-A

Esercizio

```
/* ..Continua: stampa */
for(i=N-1; i>=0; i--)
{ j=20;
  while
    posizionati sull'ultimo carattere della parola
  }
  for(
    stampa in ordine inverso
  )
  printf("\n");
}
/* deallocazione del vettore*/
}
```

Fondamenti di Informatica L-A