Il puntatore

È un tipo scalare, che consente di rappresentare gli **indirizzi** delle variabili allocate in memoria.
Il dominio di una variabile di tipo puntatore è un insieme di indirizzi:

- Il valore di una variabile di tipo puntatore puo` essere l'indirizzo di un'altra variabile (variabile *puntata*).

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 */
```

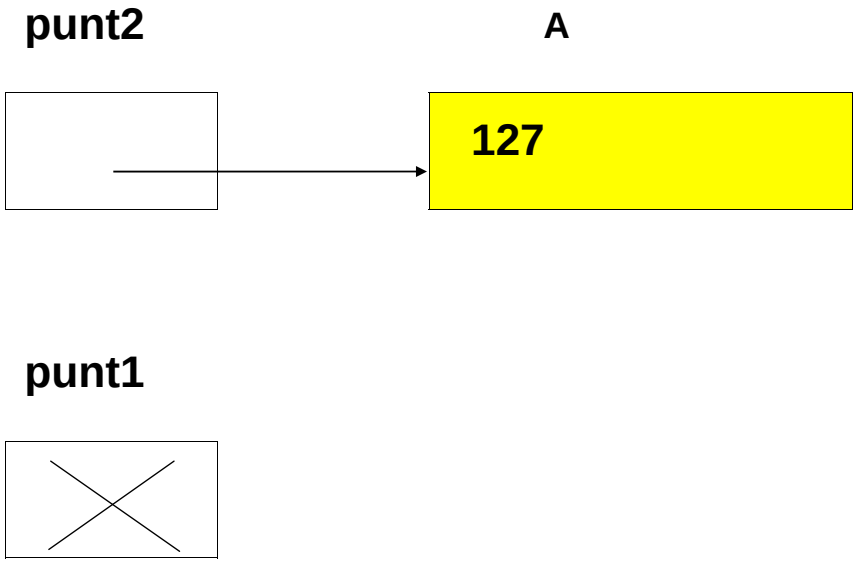

Il puntatore

Operatori:

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

Esempio:

```
int *punt1, *punt2;
int A=0;
punt1=&A;
*punt1=127;
punt2=punt1;
punt1=NULL;
```



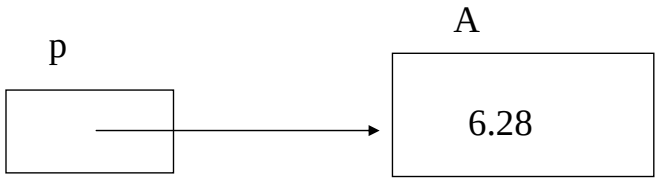
Operatore Indirizzo &:

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

Operatore Dereferencing *:

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

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

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



Puntatore come costruttore di tipo

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;
…
```


Puntatori

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{…}record;

```
int *p, A;
record *q, X;

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.

Variabili Dinamiche

In C si possono definire è possibile classificare le variabili in base al loro tempo di vita; è possibile individuare due categorie:

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

oltre alle variabili **statiche**.

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 puo` riferire.
- Il programmatore non ha la possibilità di influire sul tempo di vita di variabili automatiche.

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 associato ad esse, 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 stabilite dal programmatore).

Variabili Dinamiche

☞ Il C prevede funzioni standard di **allocazione deallocazione** per variabili dinamiche:
- malloc
- free

Non sono definite a livello di linguaggio di programmazione, ma a **livello di sistema operativo**, mediante la libreria standard **<stdlib.h>.**

Variabili Dinamiche

Allocazione di variabili dinamiche:

La memoria dinamica viene allocata con la funzione standard ***malloc***:

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

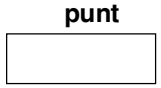
- **tipodato** è il tipo della variabile puntata
- **punt** è una variabile di tipo **tipodato ***
- **sizeof**() è una funzione standard che calcola il numero di bytes che occupa il dato specificato come argomento
- è necessario convertire esplicitamente il tipo del valore ritornato (casting): (tipodato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

Significato:

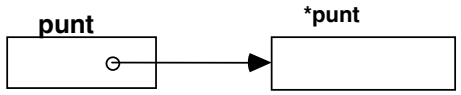
☞ La malloc provoca la creazione di una variabile dinamica nell'***heap*** e restituisce come valore l'indirizzo della variabile creata.

Ad esempio:

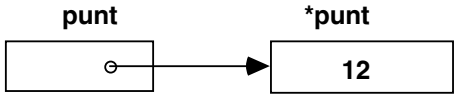
```
#include <stdlib.h>
typedef int *tp;
tp punt;
...
```




```
punt=(tp )malloc(sizeof(int));
```




```
*punt=12
```



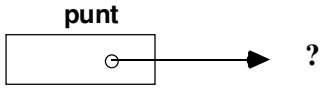
Variabili dinamiche

Deallocazione:

Si rilascia la memoria allocata dinamicamente con:

free (punt);

dove punt è l'indirizzo della variabile da deallocare.



Dopo questa operazione, la cella di memoria occupata da *punt viene deallocata: *punt non esiste piu`.

Esempio:

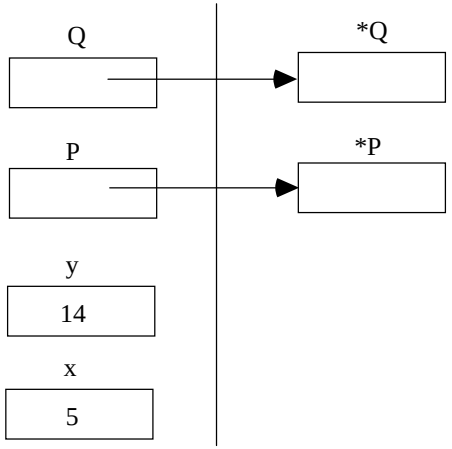
```
main()
{
  char A, *p;

  A='Z';
  p=(char *)malloc(sizeof(char));
  *p=A;
  ...
  <uso di *p>
  ...
  free(p);
  p=NULL;
}
```

Esempio:

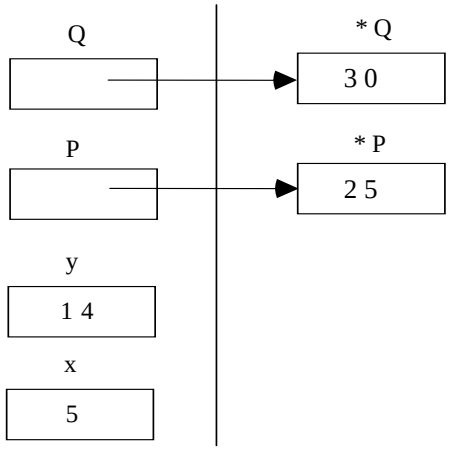
```
#include <stdlib.h>
main()
{
int *p;
  /*definizione del puntatore p
    ad intero;il contenuto di p non è
    ancora definito */

p = (int *) malloc(sizeof (int));
  /*definizione del contenuto di p:
    indirizzo di una cella di memoria
    allocata dinamicamente*/

*p = 55;
   /* assegnamento di un valore alla
      cella *p referenziata da p */

free(p);
   /* deallocazione della cella
      referenziata da p; il contenuto
      di p non è più definito */

p=NULL;      /* si definisce p come
      puntatore a NULL */
}
```


Esempio:

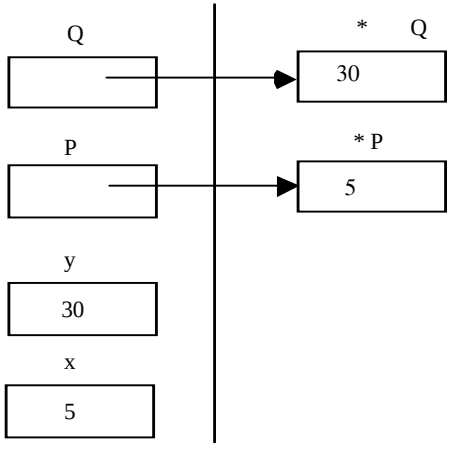
```
main()
{
int *P, *Q, x, y;

x=5;
y=14;
P=(int *)malloc(sizeof(int));
Q=(int *)malloc(sizeof(int));
```



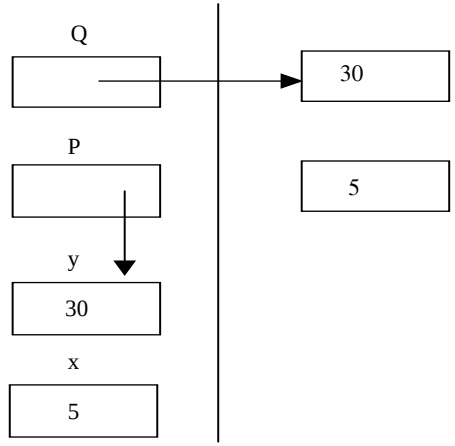
```
*P = 25;
*Q = 30;
```




```
*P = x;
y = *Q;
```




```
P = &y;
```




```
...
}
```

☞ 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 é più utilizzabile!

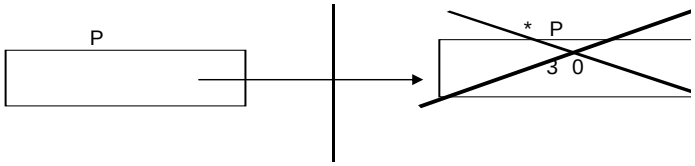
Problemi legati all'uso dei Puntatori

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!  */
```



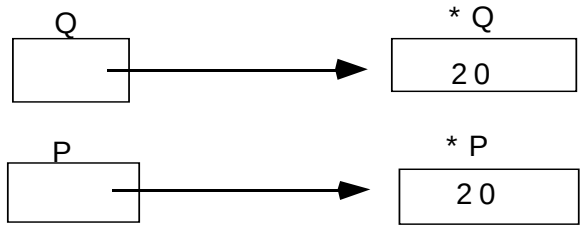
Problemi legati all'uso dei Puntatori

Aree inutilizzabili:

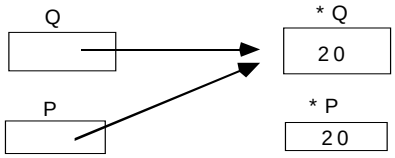
Possibilità di perdere il riferimento ad aree di memoria allocate al programma (non più riusabili).

Ad esempio:

```
int *P,*Q;
  P = (int *) malloc ( sizeof (int));
  Q = (int *) malloc ( sizeof (int));
  *P = 30;
  *Q = 20;
*P = *Q;
```



P = Q;



L’area che era puntata da P non è più raggiungibile, ma rimane allocata al programma!

Prima dell’assegnamento occorre (da programma) liberare l’area referenziata da P:

free(P);
P = Q;

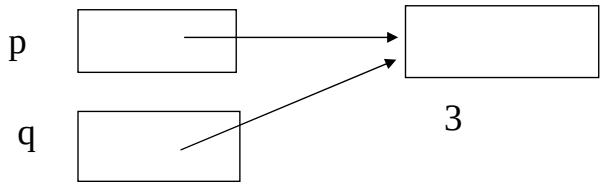
Problemi legati all'uso dei Puntatori

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 è cambiato! */
```

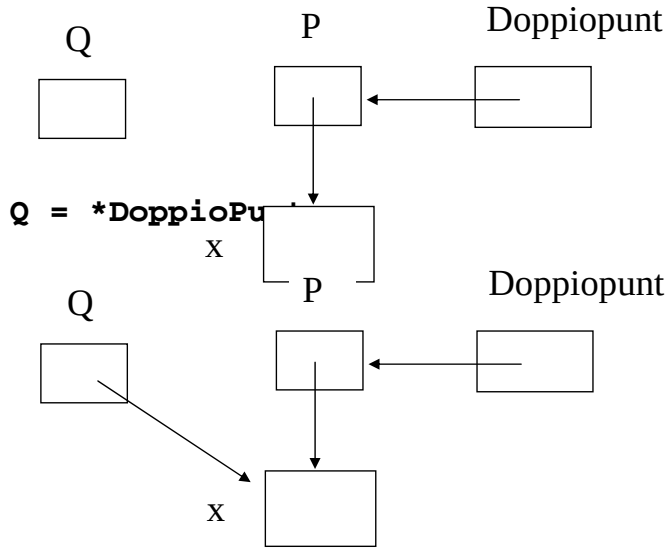
Puntatori a puntatori (handle)

Un puntatore pu` puntare a variabili di tipo qualunque (semplici o strutturate); puo` puntare anche a un puntatore:

[typedef] TipoDato **TipoPunt;

Ad esempio:

```
int x, *P, *Q, **DoppioPunt;
  P = &x;
  DoppioPunt = &P;
```



Vettori & Puntatori

Vettori:

- in C, i vettori vengono allocati in memoria in **parole consecutive** (cioè parole fisicamente adiacenti), la cui ***dimensione*** dipende dal tipo degli elementi del vettore.

- Il ***nome*** di una variabile di tipo vettore viene considerato dal C come *l'**indirizzo*** del primo elemento del vettore.

Ad esempio:

```
int V[10];
```

☞ V è una **costante**:
- V equivale a **&**V[0]
- come tipo è un puntatore ad intero:

```
int *p, V[10];
p=V;
/* p punta a V[0] */
V = p;
/*NO! V è un puntatore costante*/
```

Vettori & Puntatori

Il C consente di eseguire operazioni di somma e sottrazione sui puntatori (a vettori).

Operatori aritmetici 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 indicato da V;
(V-W):restituisce l'intero che rappresenta il numero di elementi compresi tra V e W.

Ad esempio:

```
float V[100], *p, *q;
int k;
p=V+7;          /* p punta a V[7] */
q=V+2;          /* q punta a V[2] */
k=p-q;               /* k vale 5 */
...
```


Vettori e Puntatori

☞ In C, ogni riferimento ad un elemento di un vettore è espanso come un *puntatore dereferenziato*:

V[0]	equivale a	*(V)
V[1]	equivale a	*(V + 1)
V[i]	equivale a	*(V+i)
V[expr]	equivale a	*(V + expr)

Ad esempio:

```
main ()
{
char a[] = "0123456789";
/* a vettore di caratteri */
int i = 5;

printf("%c%c%c%c\n",a[i],a[5],i[a],5[a]);
}
```

Stampa:

5 5 5 5

Per il compilatore V[i] e i[V] sono lo stesso elemento, perché viene sempre eseguita la conversione:

V[i] => *(V+i)

senza eseguire alcun controllo né su V né su i.

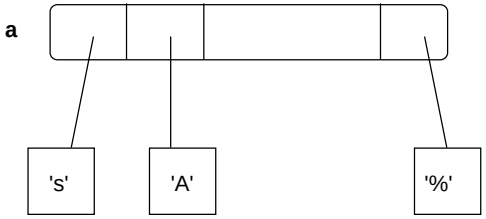
Vettori & Puntatori

☞ **[ ]** ha precedenza rispetto a *

Quindi:

char *a[10]; => equivale a **char** *(a[10]);

a è un **vettore di puntatori** a carattere.



☞ Per un puntatore ad un vettore di caratteri è necessario forzare la precedenza (con le parentesi)

char (* a) [10];

oppure:

typedef char Vetchar[10];
Vetchar *a;

Puntatori a strutture:

E' possibile utilizzare i puntatori per accedere a variabili di tipo struct.

Ad esempio:

```
typedef struct
     {int Campo_1,Campo_2;} TipoDato;

TipoDato S, *P;

P = &S;
```

Il punto della notazione postfissa ha precedenza sull'operatore di dereferencing *; per accedere alle componenti della struttura referenziata da P è necessario utilizzare le parentesi tonde:

```
(*P).Campo_1=75;
```

Operatore ->:

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

```
P->Campo_1=75;
```

Esempio

Si vuole realizzare un programma che data da input una sequenza di N parole (ciascuna, al massimo, di 20 caratteri ciascuna), una per riga, stampi in oredine inverso le parole date, ognuna "ribaltata" (cioè, stampando i caratteri in ordine inverso: dall'ultimo al primo). Utilizzare una struttura dinamica.

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

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

  printf("Quante parole? ");
  scanf("%d", &N);
  /* allocazione del vettore */
  p=(parola *)malloc(N*sizeof(parola));
  /* lettura della sequenza */
  for(i=0; i<N; i++)
    gets(&p[i]);

/* stampa */
  for(i=N-1; i>=0; i--)
  { j=20;
    do
       j--;
    while(p[i][j]!='\0');
    for(j--;j>=0; j--) putchar(p[i][j]);
    printf("\n");
  }
free(p); /* deallocazione */ }
```