

I TIPI COMPOSTI

1. definizione struttura del tipo (tipi semplici + costruttori)
2. definizione e implementazione operazioni (non ora)

La definizione strutturale

typedef old type new type;

typedef int TipoSalario; TipoSalario miosalario, tuosalario;

typedef enum { FALSE, TRUE } boolean; boolean fine=FALSE;

I costruttori di tipo in memoria centrale

costruttore ARRAY:

- aggregato ordinato di elementi dello stesso tipo;
- dimensione fissa; memoria centrale
- accesso agli elementi posizionale.

costruttore RECORD:

- aggregato di elementi che possono essere di tipo diverso;
- dimensione fissa; memoria centrale
- accesso agli elementi per nome .

costruttore RICORSIVO:

- aggregato di elementi dello stesso tipo costruito ammettendo che un tipo possa contenere riferimenti a componenti dello stesso tipo;
- dimensione arbitraria; memoria centrale
- accesso sequenziale agli elementi.

Il costruttore ARRAY (vettori e matrici)

```
#define DIM 10
```

```
int V[DIM] = {1,2,3,4,5,6,7,8,9,10}, S[DIM];
```

Allocazione memoria

- V: RES 10 (V \equiv indirizzo primo elemento)
- limite: dimensione fissa definita in compilazione
- accesso diretto all'elemento in posizione "i"
 $0 \leq \text{posizione (discreto)} \leq (\text{DIM} - 1)$
V[i] ... V[i+1] ... V[S[2] +5] ... V[expression]
- V = S; print, scanf sull'intero array non ammessi
- sfondamento vettore non gestito

Esempio:

```
/* Legge sequenza di 100 numeri e la visualizza in ordine inverso*/  
#include <stdio.h>  
#define MAX 100  
int i, vettore[MAX];  
main()  
{ for (i=0; i < MAX; i++) scanf("%d", &vettore[i]);  
  for (i=MAX-1; i >=0; i--) printf("%d\n", vettore[i]);  
}
```

Problemi:

Attenzione ai confini del vettore

```
int V[DIM];
```

```
for (i=0;i<=DIM; i++) ... ⇐ ultimo
```

```
for (i=1;i<DIM; i++) ... ⇐ primo
```

```
for (i=0;i<DIM; i++)
```

```
    if(V[i]>=V[i-1])...⇐ primo
```

```
for (i=0;i<DIM; i++)
```

```
    if(V[i]>=V[i+1]) ⇐ ultimo.
```

```
for (i=0;i<DIM; i++)
```

```
    if ((V[i]>=V[i-1])&& (i!=0)) associatività
```

⇓

```
    if (i!=0) if (V[i]>=V[i-1])) .
```

```
char d[80]; int i=0; char val;
```

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

```
while (val != '#')
```

```
    { d[i]=val; i++;    scanf("% c", &val); }
```

Dai vettori alle matrici

Bidimensionale

```
typedef int A[10];      A B[ 20];
```

oppure

```
int B [20][10];
```

```
int B[3][2]={ { 1,2},{ 3,4},{ 5,6} }
```

Accesso B[i][j]

Tridimensionale

```
int B [20][10][30]
```

Accesso B[i][j][k]

Memorizzazione

$B \equiv B[0][0][0]$

$B+1 \equiv B[0][0][1]$

....

Esempio

Traduce numero intero in stringa di caratteri (itoa) e visualizza la stringa

```
#include <stdio.h>
int main()

{ int i, c, temp, pos, sign, n; char s[20];

printf ("inserire n: "); scanf("%d", &n);
sign=n;
if (sign<0) n = -n;
pos = 0;
do { /* genera le cifre nell'ordine inverso */
    s[pos] = n % 10 + '0'; pos++; n=n/10;
} while (n > 0);
if (sign < 0) {s[pos] = '-'; pos++;}
s[pos] = '\0';
printf("\n");

// reverse: inverte la stringa (s)
pos--;
for (i = 0; i <= pos; i++, pos--)
    {temp = s[i]; s[i] = s[pos]; s[pos] = temp; }
printf("\n");
i=0; while (s[i] !='\0') {printf("%c", s[i]);i++;}
}
```

RECORD

- 1) typedef struct {int A; int B;} vet; vet V,Z;
- 2) struct vet {int A; int B;}; struct vet V,Z;
- 3) struct {int A; int B;} V,Z;

Allocazione memoria V.A: RES 1
 V.B: RES 1

Accesso

al singolo elemento

- V.A = 2;
 - scanf e printf solo sul singolo elemento globale
- V = Z; permesso

Esempi:

```
typedef char stringa[20];
typedef enum{ amb, eln, tel, inf,... } tipoCL;
typedef struct {int matricola;
               stringa cognome, nome;
               tipoCL CorsoDiLaurea;
               }
        TipoStudente;
Tipostudente Lista[100];
Tipostudente Studente;
.....
printf("cognome = %s",Studente.cognome);
printf("cognome = %s",Lista[i].cognome);
printf("matricola = %d",Lista[i].matricola);
printf("iniziale cognome = %c",Lista[i].cognome[0]);
```

A proposito di compatibilità dei tipi

- tipi opachi - equivalenza per nome
 - `struct p{int a; int b;}` e `struct q{int a; int b;}` sono diversi
- tipi trasparenti – equivalenza strutturale
 - `struct p{int a; int b;}` e `struct q{int a; int b;}` sono uguali

ma non precisato se struct sia equivalente a:

- `struct q{int b; int a}`
- `struct q{int m; int n}`

C adotta:

- equivalenza per nome per record
- equivalenza strutturale per array e tipi del typedef (eccetto struct)

Java adotta equivalenza per nome eccetto array

La gestione applicativa del vettore

To do

1. domandarsi se sia necessario
 - Leggere sequenza di N valori 0,1 dal terminale e visualizzare il numero di 1 presenti nella sequenza
 - Leggere sequenza di valori 0,1 da terminale e visualizzare la sequenza in complemento a 1; la sequenza è di lunghezza arbitraria con il valore 2 come terminatore.
 - Leggere sequenza di numeri di lunghezza arbitraria e stamparla in ordine inverso.
2. stabilire tipo e dimensione massima (analisi problema)
 - gestione matematica di vettori e matrici (ad es. soluzione di sistemi di equazioni lineari) o sequenza di dati a lunghezza fissa
 - sequenza di dati con dimensione non nota a priori
 - - pila/stack (politica LIFO)
 - - coda (politica FIFO)
 - - sequenza numerica
 - - archivio dati in memoria centrale
3. controllo sfondamento confini del vettore
4. controllo casi limite: vettore pieno in insert o vuoto in delete
5. definire meccanismo di gestione della dinamicità

Gestione della dinamicità

1. Contiguità fisica con terminatore

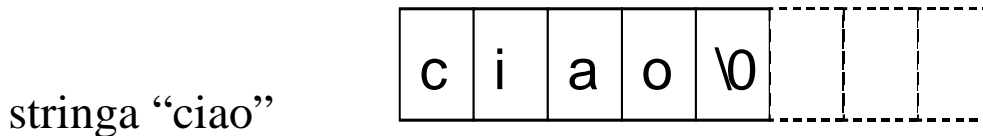
////////////////////////////////////	-9999 ? ? ? ? ? ? ? ? ? ?
--------------------------------------	---------------------------

Problema: conflittualità con valori applicativi

L'esempio della stringa di caratteri

Terminatore convenzionale: carattere NULL speciale \0 (ASCII 0)

Esempio:



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

```
void main()
```

```
{ char v[8]="ciao";
```

```
  oppure
```

```
  char v[]="ciao"; => v[5]
```

```
  oppure
```

```
  char v[8]; e poi v[0]='c'; v[1]='i'; v[2]='a'; v[3]='o';v[4]='\0';
```

```
  attenzione: v="ciao"; non ammessa
```

```
  printf("%s",v);    ammessa
```

```
  - stampa sino a \0 escluso; pericolo sfondamento
```

```
  scanf("%s", v);
```

- non usare &v

- spazio,tab,\n sono considerati terminatore

- aggiunge \0; pericolo sfondamento

```
}
```

Le funzioni di <string.h>

#include <string.h>

- **int** **strcmp(char *s1, char *s2);**
risultato è < 0 s1 alfabeticamente minore di s2
 = 0 s1 alfabeticamente uguale a s2
 > 0 s1 alfabeticamente maggiore di s2
- **char** ***strcpy(char *s1, char *s2);**
s2 copiata in s1 sino a \0 compreso (assume s1 abbastanza capiente) e restituito s1.
- **char** ***strcat(char *s1, char *s2);**
concatena s1 a s2 e pone risultato in s1 (assume s1 abbastanza capiente)
- **unsigned** **strlen(char *s);**
restituisce numero caratteri che precedono \0

Esempio:

```
/* Programma Concatenazione di stringhe */
#include <stdio.h>
#include <string.h>
#define dim 50
main()
{ char Stringa1[dim], Stringa2[dim], StringaConc[2 * dim];
  int LunghezzaConc;
  scanf("%s", Stringa1); scanf("%s", Stringa2);
  { strcpy(StringaConc, Stringa1); strcat(StringaConc, Stringa2);}
  LunghezzaConc = strlen(StringaConc);
  printf("la stringa concatenata %s.\n è lunga %d caratteri\n",
        StringaConc, LunghezzaConc);
}
```

2: Contiguità fisica con variabile di supporto. N



ultimo ↑

0

N

Esempio:

int Lista[100]; int ultimo; (vettore vuoto)

oppure

struct {int ultimo; int vettore[100]; } Lista;

Inizializzazione ultimo

3. Mappa degli occupati

//	?	?	//	//	//	?	//	?	//	//
T	F	F	T	T	T	F	T	F	T	T

↑ occupato

↑ libero

Esempio

typedef enum {TRUE, FALSE} boolean;

typedef struct {int valore;

boolean occupato;

} Tipoelemento;

Tipoelemento Lista[100];

Inizializzazione del campo occupato

Osservazioni:

- le strutture di supporto devono essere mantenute congruenti nelle operazioni di aggiornamento della struttura dati;
- efficacia con insert, delete, update: discussione

PUNTATORI Accesso a variabili via nome e via indirizzo

```
int Norm;
```

```
int *P1=NULL, *P2;
```


```
float *P3;
```

	adr	
Norm	1	??
P1	2	NULL
P2	3	??
P3	4	??

```
Norm = 4;
```

```
P1 = &Norm;
```

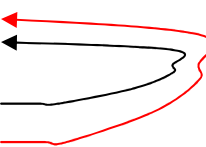
	adr	
Norm	1	4
P1	2	1
P2	3	??
P3	4	??



```
P2 = P1;
```

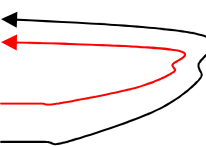
```
if (p1 == p2)  $\Rightarrow$  vero
```

	adr	
Norm	1	4
P1	2	1
P2	3	1
P3	4	??



```
*P1 = 5
```

	adr	
Norm	1	5
P1	2	1
P2	3	1
P3	4	??



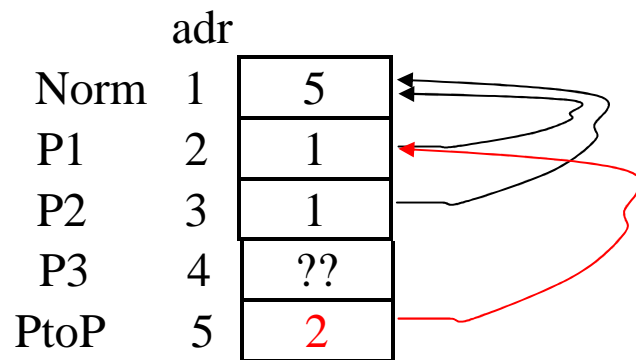
```
printf("%d %d %d", Norm, *P1, *P2);  $\Rightarrow$  5 5 5
```

```
P3 = P1;     warning a compile time
```

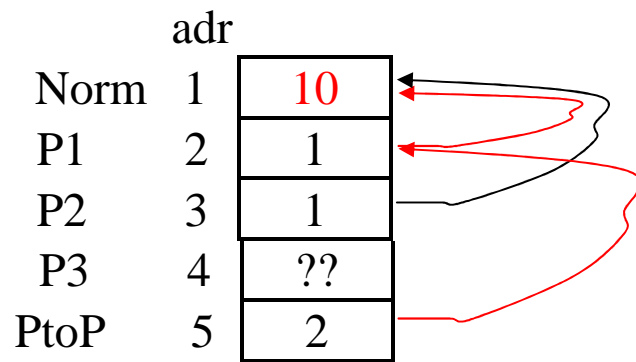
Catene di puntatori

```
int **PToP;
```

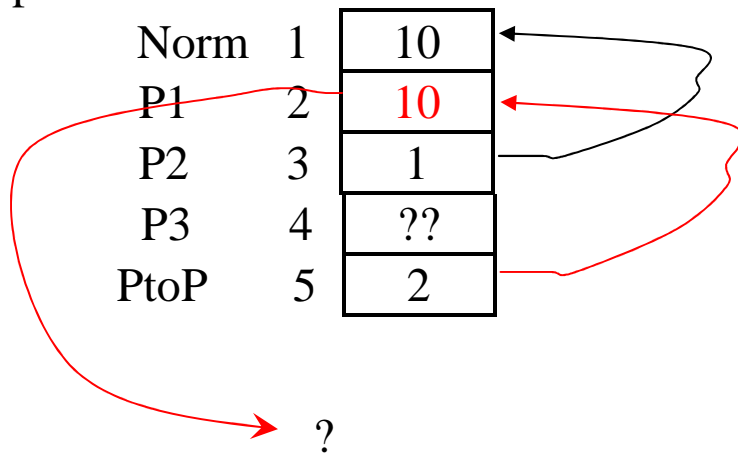
```
PToP = &P1;
```



```
**PToP = 10;
```



```
*PTop = 10
```



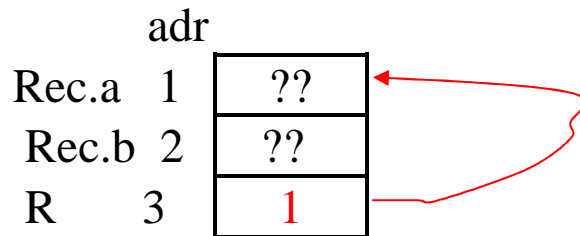
Accesso ad un record

Esempio:

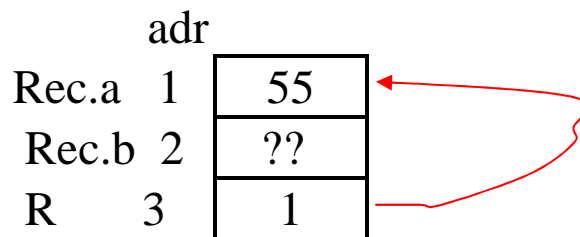
```
typedef struct {int a;int b;} miotipo;
```

```
miotipo rec, *R;
```

```
R = &rec; indirizzo del record
```



```
(*R).a = 55;    ≡    R ->a = 55;
```



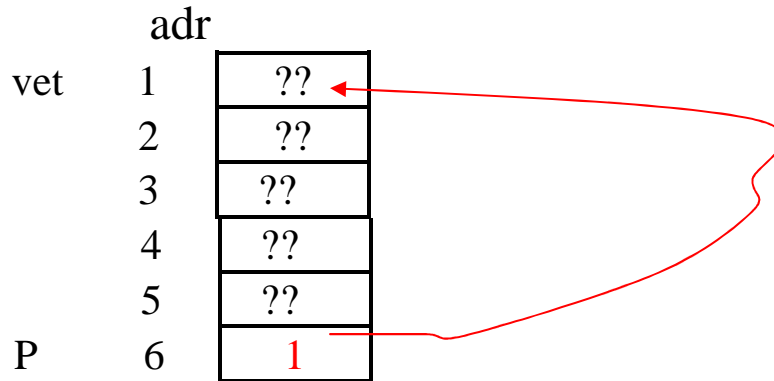
Accesso ad un vettore

```
typedef int A[5];
```

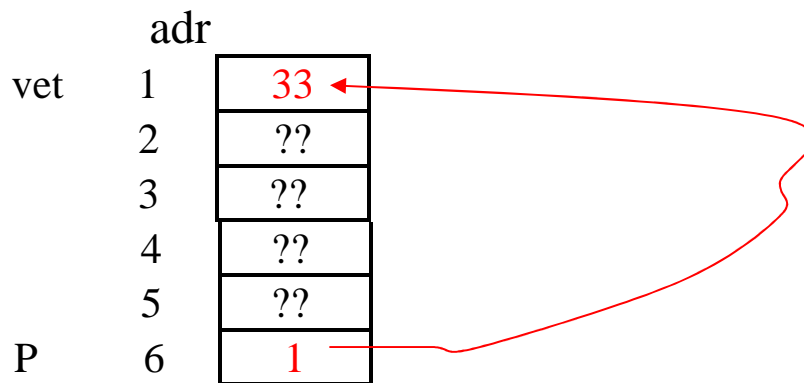
```
A vet;    vet  $\equiv$  indirizzo del primo elemento del vettore
```

```
int *P;    indirizzo del singolo elemento
```

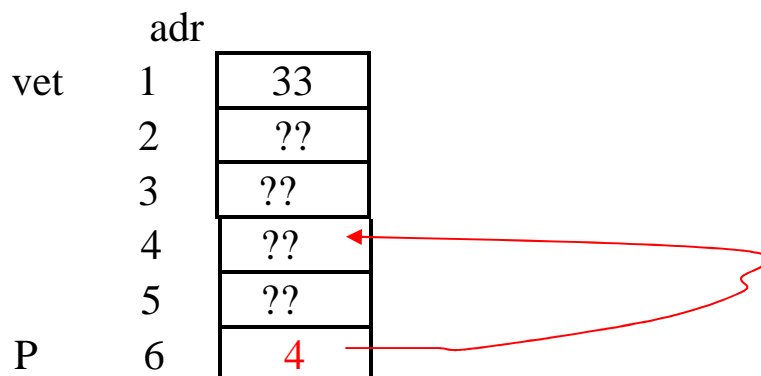
```
P = vet;  $\equiv$  P = &vet[0];
```



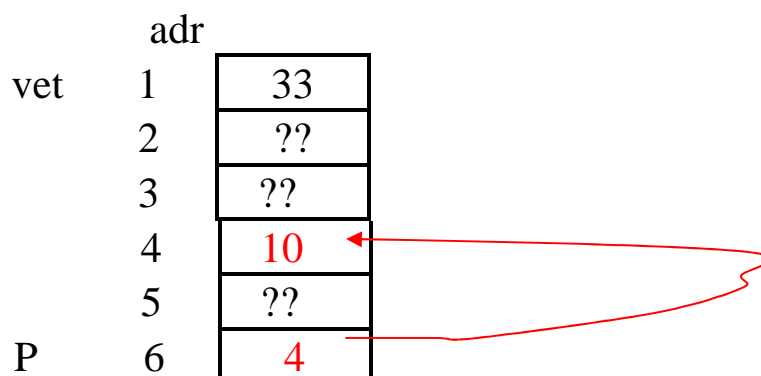
```
*P = 33;  $\equiv$  vet[0] = 33;
```



```
P = &vet[3]
```



```
*P = 10;  $\equiv$  vet[3] = 10;
```



Esempio:

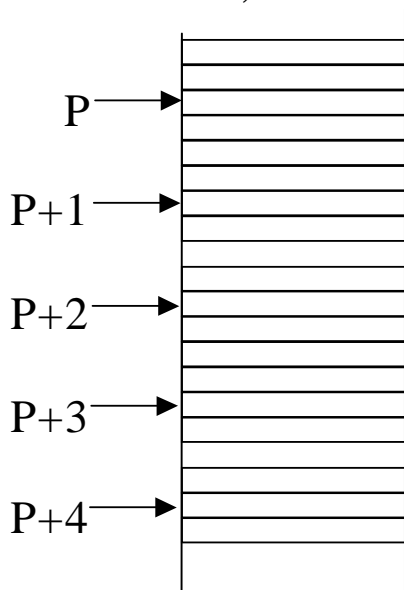
vettore di 100 interi diversi tra loro; assegna a due puntatori l'indirizzo degli elementi con valore minimo e massimo

```
/* Programma */
#define MAX100
int i, Vettore[MAX], *PuntaAMinore, *PuntaAMaggiore;
main()
{.....
  PuntaAMinore = &Vettore[0];
  PuntaAMaggiore = &Vettore[0];
  for (i=1; i < MAX; i++)
    {if (Vettore[i] < *PuntaAMinore) PuntaAMinore = &Vettore[i];
     if (Vettore[i] > *PuntaAMaggiore) PuntaAMaggiore = &Vettore[i];
    }
}
```

L'aritmetica dei puntatori e gli array

```
typedef struct {int a; int b;} el;
el vet[100];  el *p=vet;
```

$P = P + 1 \equiv P++; \Rightarrow \text{indirizzo}(P) + 1 * \text{sizeof}(\text{el}) \equiv \&\text{vet}[1]$



P=vect;
(P+i) ≡ &vet[i] ≡ (vet+i)
e
*(P + i) ≡ vet[i]

Esempio:

```
typedef int A[3];  
A vet;  int *P; int i;  
void main()  
{ P = vet;  
  for (i=0; i<=2; i++)  
    { vet[i] = i*10; printf("\n%d",*(P+i)); }  
}
```

stampa:

0
10
20

Attenzione

typedef int A[10]; A vet; A *P; ⇒ p++ sizeof intero vettore

Osservazione:

Priorità

[]()

*

int *p[5] ⇒ array di 5 pointer to int

int (*p) [5] ⇒ pointer to array di 5 int