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 Salario;

typedef **enum** { FALSE, TRUE} Tboolean; Tboolean fine=FALSE;

typedef **enum** { lunedi, martedi,…,domenica} Tgiorno;
T_giorno giorno;

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];
```

- limite: dimensione fissa definita in compilazione
  Array a lunghezza variabile (C99) FUORI PROGRAMMA

Allocazione memoria (1)

- V: RES 10 (V ≡ indirizzo primo elemento)

```
printf("\nsizeEl=%d",sizeof(V[0])) ;
   printf("\nsize=%d",sizeof(V));

printf("\n%10.10x ", (unsigned int) &V[0]);
printf("\n%10.10x ", (unsigned int) &V[1]);
printf("\n%10.10x ", (unsigned int) &V[9]);
```

- accesso diretto all'elemento in posizione "i"

  0 <= posizione (discreto) <= (DIM –1)

  V[3], V[i] … V[i+1] … V[expression] … V[S[2] +5]

- V = S (NON ammesso);
- print, scanf sull'intero array non ammessi (eccezione stringhe)
- if (V==S) (ammesso sempre FALSE)

Esempio 1:

```
/* Legge sequenza di 10 numeri e la visualizza in ordine inverso*/
#include <stdio.h>
#define DIM 10
   int  i, V[DIM];
main()
{ for (i=0; i < DIM; i++)  scanf("%d", &V[i]);
   for (i=DIM-1; i >=0; i--) printf("%d\n", V[i]);
}
```

Esempio 2

```
/* Scorrimento a destra nel vettore          (2)  */
#include <stdio.h>
#define DIM 100
   int  i, V[DIM] = {1,2,3,4,5,6,7,8,9,10}, temp;
main()
{ temp= V[DIM-1];
   for (i=(DIM-1); i >=1; i--)    V[i]=V[i-1];
   V[0] = temp;
}
```

Problemi: sfondamento confine vettore

Errore facile

int V[DIM];

for (i=0;i<=DIM; i++) … ⇐ 0,1, ..., 9, 10??

for (i=1;i<DIM; i++) … ⇐ 0??, 1, ..., 9

Errore che sfugge

Esempio: slittamento valore più alto in fondo

...

```
for (i=0;i<DIM; i++)
  if(V[i]>=V[i+1]) {temp=V[i]; V[i]=V[i+1]; V[i+1] =temp}
```

Conclusione

- V[-1] o V[10] esistono: sono quelli adiacenti agli estremi del vettore
- Il C non controlla lo sfondamento. Quindi?

Esempio 3

```
/* caricare nel vettore una sequenza di lunghezza variabile di chars
(terminata da #) */
    #define DIM 80
    char V[DIM]; int i; char val;
    int main()
    {
    val = getchar();
    i=0;
    while ((val != '#') && (i<DIM))
      { V[i]=val; i++;      val=getchar(); }
    }
```

Esercizio 4

```
/* Se vettori uguali ok else notok e interruzione ciclo        */
    #define DIM 3
    int V[DIM], S[DIM], i, diversi;
    int main()
    {…//caricamento vettori
     i=0; diversi=0; //falso
    while ((diversi= =0) && (i<DIM))
      { if (V[i] != S[i]) diversi=1; }
        i++;
       }
     if (diversi = = 1)   notok  else ok
     oppure
     if (i = = DIM)       ok      else ok
   }
```

Testing

Cosa serve ancora?

- Init parziali
  int V[5] = {1,2}; da non usare

  int V[5] = {0}; si può usare

- Derivazione statica della dimensione
  int V[] = {1,2,3}; sizeof (V) = 12;

- Dai vettori alle matrici
  Bidimensionale
  typedef int A[2];
  A B[3];

  oppure

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

  Accesso B[i][j]=5;

  Memorizzazione per righe
  int B[3][2] = {1,2,3,4,5,6};

  Tridimensionale
  int B [20][10][30] Accesso B[i][j][k]

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 se compatibili
- V = = Z invalid error compile time

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

Suggerimento: adottare equivalenza per nome dove possibile

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.

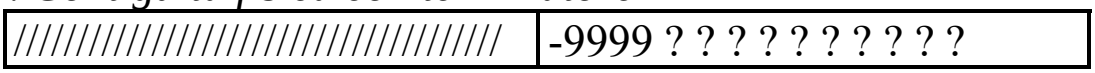
2. stabilire tipo e dimensione massima (analisi problema)
   - gestione matematica di vettori e matrici (ad es. soluzione di sistemi di equazioni lineari) o sequenza di dati a lunghezza fissa
   - sequenza di dati con dimensione non nota a priori
   - - pila/stack (politica LIFO)
   - - coda (politica 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

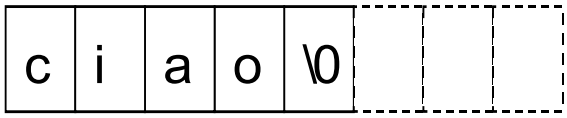
1. Contiguità fisica con terminatore



Problema: conflittualità con valori applicativi

L'esempio della stringa di caratteri

Terminatore convenzionale: carattere \0 (ASCII 0)



char v[8]="ciao";
char v[8]={'c','i','a','o'};
char v[8]; v[0]='c'; v[1]='i'; v[2]='a'; v[3]='o'; v[4]='\0';
char v[8]="ciao Ro";
char v[8]="ciao Rossi"; too long (compiler) troncato NO
char v[]="ciao"; ⇒ 5 bytes
char v[8]; v="ciao"; aggiornamento vettore NO
v[i]='s'; ammessa

printf("%s",v); 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>

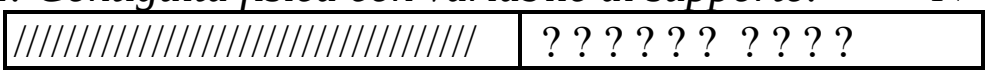
#include <string.h>

- **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:

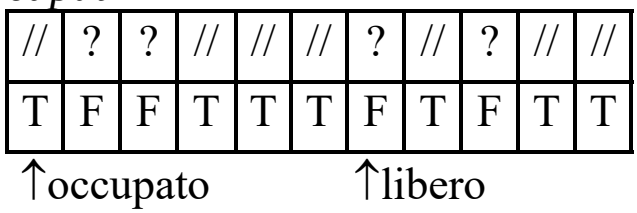
int Lista[100]; int ultimo; (vettore vuoto)

oppure

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

Inizializzazione ultimo

3. Mappa degli occupati



Esempio

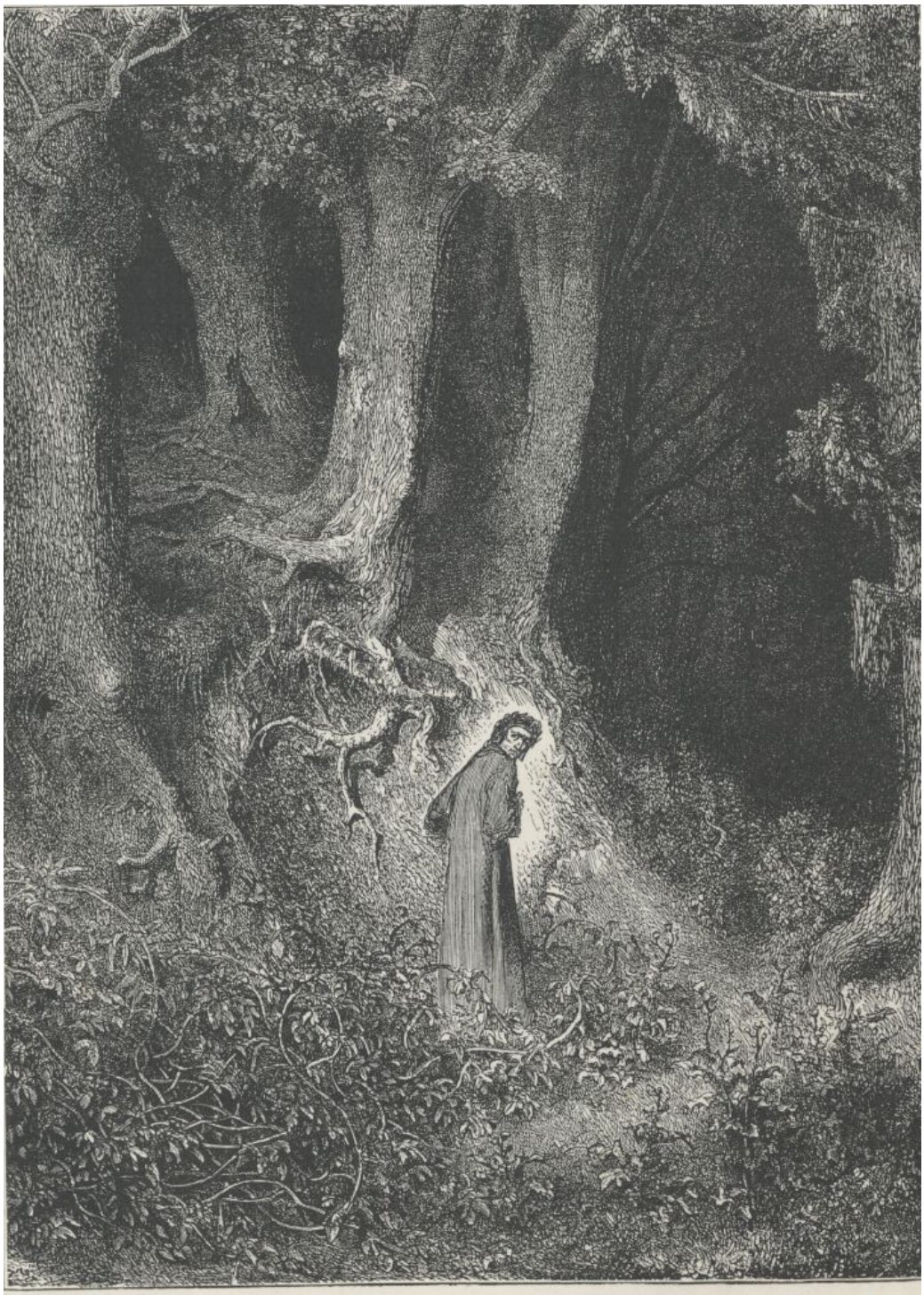
```
typedef enum{TRUE, FALSE} boolean;
typedef struct {int valore;
                boolean occupato;
                } Tipoelemento;
Tipoelemento Lista[100];
```

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

Nel mezzo del cammin di nostra vita
Mi ritrovai per una selva oscura,
ché la diritta via era smarrita …



In the midway of this our mortal life,
I found me in a gloomy wood, astray.
Canto I., lines 1, 2.

PUNTATORI Accesso a variabili via nome e via indirizzo

```
#include <stdio.h>
int N;    int *P1=NULL, *P2;      float *P3;
int main(){
printf("\n%10.10x ", (unsigned int) &N);  24
printf("\n%10.10x ", (unsigned int) &p1);       1c
printf("\n%10.10x", (unsigned int) &p2);        20
printf("\n%10.10x", (unsigned int) &p3);        28
}
```

int, float, adr= ?bytes

			valore	ind.finale
P1	(804a0)1c		NULL	
P2		20	??	
N		24	??	
P3		28	??	

Norm = 4; P1 = &N;
printf("\n%10.10x ", (unsigned int) p1); 24

P1	(804a0)1c	24	
P2	20	??	
N	24	4	
P3	28	??	

P2 = P1;
if (p1 == p2)

P1	(804a0)1c	24	
P2	20	24	
N	24	4	
P3	28	??	

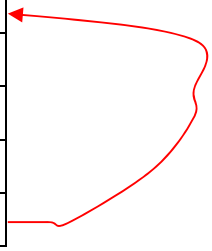
*P1 = 5

P1	(804a0)1c	24	
P2	20	24	
N	24	5	
P3	28	??	

printf("%d %d %d", N, *P1, *P2); ⇒ 5 5 5
P3 = P1; warning a compile time
Catene di puntatori

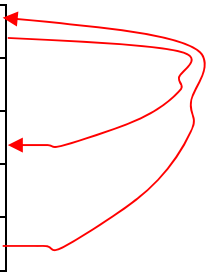
int **PToP;
PToP = &P1;

P1	(804a0)1c	24
P2	20	24
N	24	5
P3	28	??
PToP	2C	1C



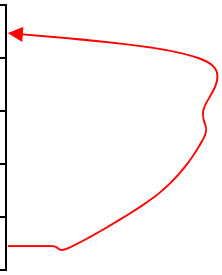
**PToP = 10;

P1	(804a0)1c	24
P2	20	24
N	24	10
P3	28	??
PToP	2C	1C



*PTop = 10 Warning → *PToP = (*int)10; OK

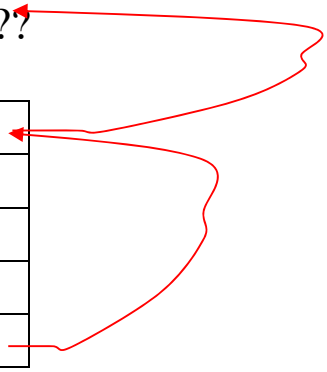
P1	(804a0)1c	10
P2	20	24
N	24	10
P3	28	??
PToP	2C	1C



*p1 =5; oppure
**PtoP=5;

??

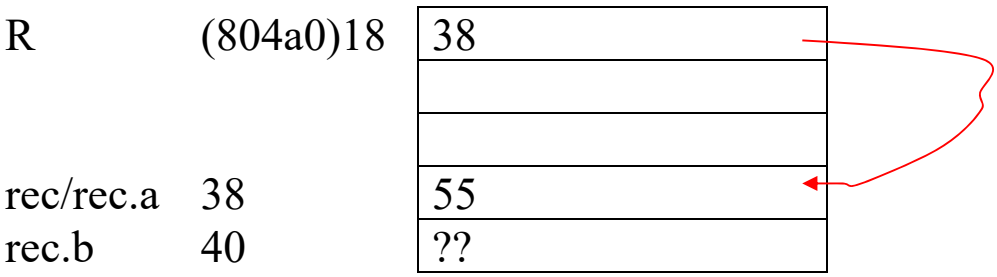
P1	(804a0)1c	10
P2	20	24
N	24	10
P3	28	??
PToP	2C	1C



Segmentation fault

Accesso ad un record

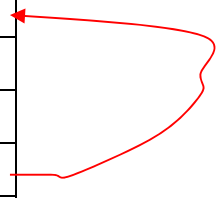
```
...
typedef struct {double a; int b;} miotipo;
miotipo rec, *R= &rec;
int main(){
(*R).a = 55;     ≡   R ->a = 55;
printf("\n%10.10x", (unsigned int) &rec);      ….38
printf("\n%10.10x", (unsigned int) &rec.a);   38
printf("\n%10.10x", (unsigned int) &rec.b);   40
printf("\n%f", rec.a);                                  55
printf("\n%10.10x", (unsigned int) &R);        18
printf("\n%10.10x", (unsigned int) R);                  38
printf("\n%10.10x", (unsigned int) &(r->a));           38
printf("\n%10.10x", (unsigned int) &(r->b));           40

}
```



Accesso ad un vettore

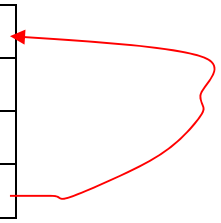
```
#include <stdio.h>
double vet[3]={1,2,3};   double *p=vet; // p=&vet[0];
int main(){
 printf("\n%10.10x ", (unsigned int) vet);          18
 printf("\n%10.10x ", (unsigned int) &vet[-1]);   10
 printf("\n%10.10x", (unsigned int) &vet[0]);      18
 printf("\n%10.10x", (unsigned int) &vet[1]);      20
 printf("\n%10.10x", (unsigned int) &vet[2]);      28
}
```

	vet[-1]	(804a0)10	??
vet	vet[0]	18	1
	vet[1]	20	2
	vet[2]	28	3
	p	30	18



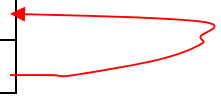
*p =33; ≡ vet[0] = 33;

vet[0]	18	33
vet[1]	20	2
vet[2]	28	3
p	30	18



p = &vet[2]; *p = 10; ≡ vet[3] = 10;

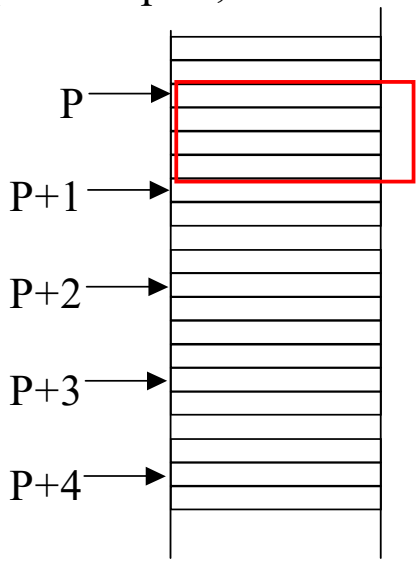
vet[0]	18	33
vet[1]	20	2
vet[2]	28	10
p	30	28



L'aritmetica dei puntatori e gli array

typedef struct {int a; int b;} el; el vet[10];
el *p=vet;

p = p + 1 ≡ p++;⇒ indirizzo(p) + 1*sizeof(struct el) ≡ &vet[1]



ossia (p+i) ≡ &vet[i] ≡ (vet+i) indirizzi
e
*(p + i) ≡ vet[i] contenuto

// numero di 'a' presenti in una stringa

```
#include <stdio.h>
#define DIM 100
char vet[DIM]="stringa di prova", *p=vet; int somma, i;
int main () {somma=0;
   for (i=0;*(p+i)!='\0';i++) if (*(p+i)=='a') somma++
   printf("\n%d\n",somma);
}
```

equivale a

```
   for (i=0;vet[i]!='\0';i++) if (vet[i]=='a') somma ++;
```

Osservazioni

- int i,j;
  j=*&i; equivale a j = i;

- int i,*p=&i;
  scanf("%d", p); valore nella variabile i;
  scanf("%d", &p); valore nella variabile p

- int *p, *q; ...
  p+i => p + sizeof(int)
  p-i => p - sizeof(int)
  p-q => distanza in numero di elementi (p-q)/sizeof(int)

- int *p[5] => array di 5 pointer to int
  int (*p) [5] => pointer to array di 5 int
  Priorità
  []()
  * (deref)