**Dipartimento di Informatica e Sistemistica**
**Antonio Ruberti**

**“Sapienza” Università di Roma**

# Puntatori
# Passaggio di parametri per indirizzo

## *Corso di Fondamenti di Informatica*

### *Laurea in Ingegneria Informatica*

***(Canale di Ingegneria delle Reti e dei Sistemi Informatici - Polo di Rieti)***

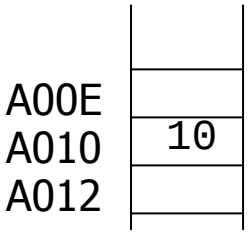
Anno Accademico 2007/2008

**Prof. Paolo Romano**

Si ringrazia il Prof. Alberto Finzi per aver reso
disponibile il materiale didattico sul quale si basano queste slides

# Puntatori

Variabili di tipo puntatore:




Esempio: `int a = 10;`

Proprietà della variabile `a` :

| | |
|---|---|
| Nome: | `a` |
| Tipo: | `int` |
| Valore: | `10` |
| Indirizzo: | `A010` |

Fino ad ora abbiamo utilizzato solo le prime tre proprietà, come utilizzare l'indirizzo??

`&a` è l'operatore indirizzo applicato alla variabile `a`

Gli indirizzi si utilizzano nelle variabili puntatore: puntatori

# Puntatori

Esempio dichiarazione puntatore:

```
int *pi;
```

Proprietà della variabile `pi` :

| | |
|---|---|
| Nome: | `pi` |
| Tipo: | puntatore ad `int` (indirizzo di intero) |
| Valore: | in principio casuale |
| Indirizzo: | fissato |

Sintassi della dichiarazione di variabili puntatore:

*tipo *nome_var*$_1$*, …, *nome_var*$_n$

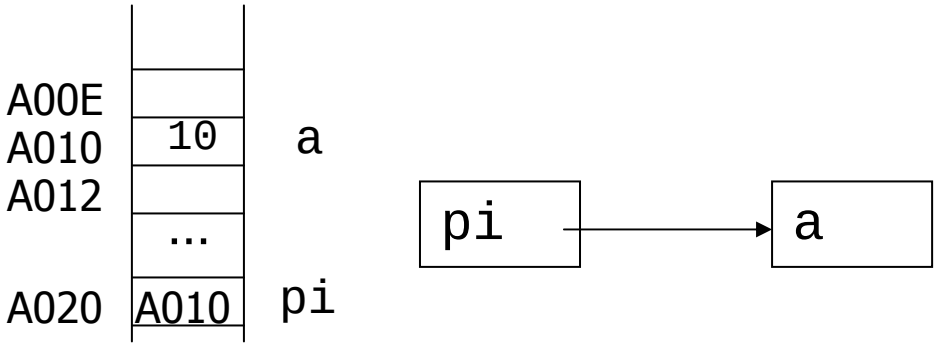
Esempio:

```
int *p1, i, *p2, j;
float *pf, g;
```

# Operatori

Una variabile puntatore può essere inizializzata utilizzando l'operatore di indirizzo:

```
pi = &a;
```




**Operatore di dereferenziamento "*"**

Applicato ad una var. puntatore fa riferimento all'oggetto puntato

Esempio:

```
int *pi, a = 10, b;

pi = &a; /* in pi indirizzo di a*/
b = *pi; /* in b valore contenuto nell'indirizzo di pi */
```

Nota: se `pi` è di tipo `* int` allora `*pi` è `int`

# Operatori

Non confondere:
- **"*" in una dichiarazione**: serve per dichiarare una var. di tipo puntatore (es. `int *pi;`)
- **"*" in una espressione**: è l'operatore di dereferenziamento (es. `a = *pi;`)

Operatori di deref. ed indirizzo:
- priorità più elevata degli operatori binari
- "*" associativo a destra: `**p` equivale a `*(*p)`
- "&" applicabile solo a variabile: es. `&a`
- "*" "&" uno inverso dell'altro:
  a. data `int a;`
     `*&a` equivale ad `a` (due variabili)
  b. data `int *pi;`
     `&*pi` ha stesso valore di `pi`

# Stampa dei puntatori

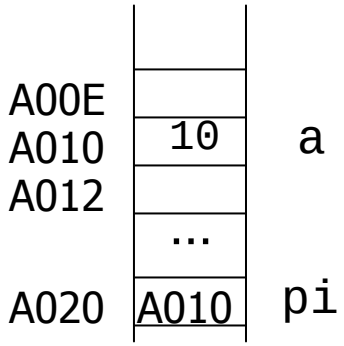
I puntatori si possono stampare con `printf` e specificatore di formato `%p` (stampa in esadecimale)

Esempio

```
int a = 10;
int *pi;

pi = &a;
printf("indirizzo di a:%p",&a);
printf("valore di pi:%p",pi);
printf("valore di &*pi:%p",&*pi);

printf("valore di a:%d",a);
printf("valore di *pi:%d",*pi);
printf("valore di *&a:%d",*&a);
```



# Inizializzazione dei puntatori

Come tutte le variabili i puntatori devono essere inizializzati prima di essere usati.

Errore: dereferenzializzare una variabile puntatore non inizializzata

Esempio:

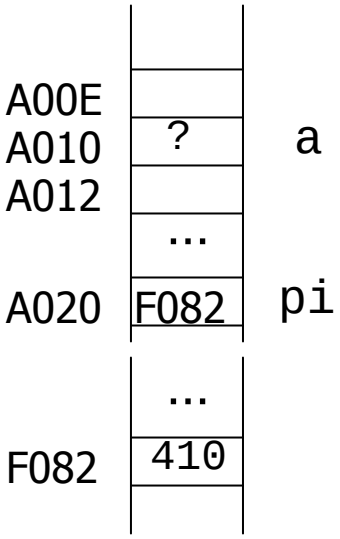
```
int a;

int *pi;

a = *pi;
*pi = 500;
```

Assegna 410 ad a

Assegna 500 nella cella di indirizzo F082



# Tipo di variabile puntatore

Il tipo di una variabile di un puntatore è "puntatore a tipo". Il suo valore è un indirizzo. ***I tipi puntatore sono indirizzi, non interi!!***

Esempio:

```
int a;
int *pi;

a = *pi;
```

Compilando genera un warning:
assignments makes integer from pointer without a cast

Due variabili tipo puntatore che si riferiscono a tipi diversi non sono compatibili tra loro

Esempio:

```
int *pi;
float *pf;
pf = pi;
```

Compilando genera un warning:
assignments from incompatible pointer type

# Tipo di variabile puntatore

Perché il C distingue tra puntatori di tipo diverso?

Per determinare a tempo di compilazione il tipo di *p

Esempio:

```
void *p;
int i; char c; float f;

p=&c;
p=&i;
p=&f;
```

Il tipo di `*p`? Verrebbe a dipendere dall'ultima istruzione!
`i/*p` ? Divisione intera o reale?

In C esiste puntatore a `void`: compatibile con tutti i puntatori; non dereferenziabile a meno di cast espliciti: printf("%d",* (int*) p);

# Funzione sizeof

La funzione sizeof restituisce l'occupazione di memoria in byte di una variabile. Può essere anche applicata ad un tipo.

Tutti i puntatori sono indirizzi ! spazio di mem. di un indirizzo
L'oggetto puntato ha dimensione del tipo puntato

```
char *pc;
int *pi;
double *pd;

printf("%d %d %d  ", sizeof(pc),     sizeof(pi),    sizeof(pd));
printf("%d %d %d\n", sizeof(char *), sizeof(int *), sizeof(double *));
printf("%d %d %d  ", sizeof(*pc),  sizeof(*pi), sizeof(*pd));

printf("%d %d %d\n", sizeof(char), sizeof(int), sizeof(double));
```

Stampa:

```
4 4 4  4 4 4

1 2 8  1 2 8
```

# Passaggio di parametri per indirizzo

Differenza tra copia del valore e copia dell'indirizzo di una variabile:

```
int b, x, *p;
b = 15;                     /* b vale 15 */
x = b;                      /* il valore di b viene copiato in x */
p = &b;                     /* l'indirizzo di b viene messo in p */

x = 23;
printf("b vale %d\n", b);    /* b non e` cambiato */
*p = 47;
printf("b vale %d\n", b);    /* b e` cambiato */
```

Se si ha una **copia dell'indirizzo** di una variabile questa copia può essere usata per **modificare la variabile** (il puntatore dereferenziato è un modo alternativo di denotare la variabile).

Sfruttando questa idea è possibile fare in modo che una **funzione modifichi una variabile della funzione chiamante**.

# Passaggio di parametri per indirizzo

In C i parametri delle funzioni sono passati per valore:

- il parametro formale è una nuova variabile locale alla funzione
- al momento dell'attivazione il valore del parametro attuale viene copiato nel parametro formale

Le modifiche fatte sul parametro formale non si riflettono sul parametro attuale (come b e x dell'esempio precedente)

Però, se passiamo alla funzione **un puntatore ad una variabile**, la funzione può usare il puntatore per modificare la variabile.

# Esempio passaggio di parametri per indirizzo

```
void fun(int x, int *p) {
  x = 23;
  *p = 45;
}

int main(void) {
  int a, b;
  int *q;
  a = 1;
  b = 15;
  printf("prima: a = %d,  b = %d\n", a, b);
  fun(a, &b);
  printf("dopo:  a = %d,  b = %d\n", a, b);
  a = 1;
  b = 15;
  q = &b;
  printf("prima: a = %d,  b = %d,  q = %p\n", a, b, q);
  fun(a, q);
  printf("dopo:  a = %d,  b = %d,  q = %p\n", a, b, q);
  return 0;
}
```

# Esempio passaggio di parametri per indirizzo

Output:

```
prima: a = 1,  b = 15,  q = 0xbffff820
dopo:  a = 1,  b = 45,  q = 0xbffff820
prima: a = 1,  b = 15
dopo:  a = 1,  b = 45
```

Il precedente è un esempio di un *passaggio per indirizzo*:

- la funzione chiamante passa l'indirizzo della variabile come parametro attuale
- la funzione chiamata usa l'operatore "*" per riferirsi alla variabile passata

(simulato il **passaggio per riferimento** che esiste in molti linguaggi)

# Esempi passaggio di parametri per indirizzo

**Esempio:**
Per passare un intero `i` per indirizzo:
- il parametro formale si dichiara come: `int *pi` (di tipo: `int*`)
- il parametro attuale è l'indirizzo di `i`: `&I`
- nel corpo della funzione si usa: `&*pi`

Il passaggio per indirizzo viene usato ogni volta che una funzione deve restituire **più di un valore alla funzione chiamante**.

**Esempio:** funzione scambio tra variabili

```
void scambia(int *i, int *j) {
  int temp;
  temp = *i;
  *i = *j;
  *j = temp;
}  /* scambia */
```

```
int main(void) {
  int p, s;
  scanf("%d%d", &p, &s);
  scambia(&p, &s);
  return 0;
}
```

# Esempi passaggio di parametri per indirizzo

Esempio ordinamento:

```
void ordina (int *pa, int *pb){
  int aux;
  if (*pa > *pb) {
    aux = *pa;
    *pa = *pb;
    *pb = aux;
  }
}
int main (void){
  int a, b;
  printf("Introduci due interi da ordinare: ");
  scanf("%d%d", &a, &b);
  ordina(&a, &b);
  printf("Numeri ordinatati: %d %d\n", a, b);
  return 0;
}
```

# Esempi passaggio di parametri per indirizzo

Scrivere una funzione che riceve come parametri giorno, mese, ed anno di una data, e li aggiorna ai valori per la data del giorno dopo:

```
void avanzaUnGiorno(int *giorno, int *mese, int *anno)
{
  if (*giorno == giorniDelMese(*mese, *anno)) {
    *giorno = 1;
    if (*mese == 12) {
      *mese = 1;
      (*anno)++;
    }
    else
      (*mese)++;
  }
  else
    (*giorno)++;
}  /* avanzaUnGiorno */
```

# Aritmetica dei puntatori

Sui puntatori si possono effettuare diverse *operazioni*:

- dereferenziamento:

```
int *p, i;
i = *p;
```

- assegnamento:

```
int *p, *q;
p = q;
```

N.B. p e q devono essere dello stesso tipo (altrimenti bisogna usare l'operatore di cast)

- confronto:

```
if (p == q)…
if (p > q)…
```

Che senso ha?

- aritmetiche:

```
incremento (++) o decremento (--)
somma (+=) o sottrazione (-=) di un intero
sottrazione di un puntatore da un altro
```

# Aritmetica dei puntatori

Il *numero di byte* di cui viene modificato il puntatore dipende dal suo tipo.

```
int *pi;
*pi = 15; ! punta al prossimo int (4 byte dopo)
*pi = 20;

double *pd;
*pd = 12.2;
pd += 3; ! punta 3 double dopo (24 byte dopo)

char *pc;
*pc = 'A';
pc -= 5; ! punta 5 char prima (5 byte prima)
```

# Relazione tra vettori e puntatori

Attenzione: in generale non sappiamo cosa contengono le celle di memoria adiacenti ad una data cella.

L'unico caso in cui sappiamo quali sono le locazioni di memoria successive e cosa contengono è quando utilizziamo dei vettori.

In C il nome di un vettore è **l'indirizzo dell'elemento di indice 0**

Esempio:

```
int vet[10];
```

`vet` e `&vet[0]` hanno lo stesso valore.

```
printf("%p %p", vet, &vet[0]);
```

stampa 2 volte lo stesso indirizzo.

# Accesso agli elementi di un vettore

Possiamo far puntare un puntatore al primo elemento di un vettore.

Esempio:

```
int vet[5];
int *pi;
pi = vet;
```

è equivalente a `pi = &vet[0];`

Esempio:

```
int vet[5];
int *pi = vet;
*(pi + 3) = 28;
```

`pi+3` punta all'elemento di indice `3` del vettore. `3` viene detto *offset* (o scostamento) del puntatore.

# Accesso agli elementi di un vettore

**Osservazione:**
`&vet[3]` equivale a `pi+3` equivale a `vet+3`
`*&vet[3]` equivale a `*(pi+3)` equivale a `*(vet+3)`
inoltre, `*&vet[3]` equivale a `vet[3]`.

Qundi: in C, `vet[3]` è semplicemente un modo alternativo di scrivere `*(vet+3)`

**Notazioni per gli elementi di un vettore:**
`vet[3]` : notazione con *puntatore e indice*
`*(vet+3)` : notazione con *puntatore e offset*

# Accesso agli elementi di un vettore

Riassumendo, si può accedere agli elementi di un vettore al seguente modo:

Esempio:

```
int vet[5] = {11, 22, 33, 44, 55};
int *pi = vet;
int offset = 3;
vet[offset] = 88;
*(vet + offset) = 88;
pi[offset] = 88;
*(pi + offset) = 88;
```

Esempio: Sono corrette le istruzioni nel seguente frammento di codice?

```
int vet[10];
int *pi;
vet = pi;
vet++;
vet += 2;
```

No, perchè `vet` è un *puntatore costante*, e quindi non può essere modificato.

# Passaggio di parametri di tipo vettore

Quando si passa un vettore come parametro ad una funzione, in realtà si sta passando l'indirizzo dell'elemento di indice 0.

Il parametro formale deve essere di tipo puntatore al tipo degli elementi del vettore. Di solito si passa la dimensione del vettore in un ulteriore parametro.

Esempio:

```
void stampa(double *v, int dim){
  int i;
  for (i = 0; i < dim; i++)
    printf("%d: %g\n", i, v[i]);
}

int main(void) { ...
  double vet[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
  stampa(vet, 5);
  ...
}
```

# Passaggio di parametri di tipo vettore

Per evidenziare che il parametro formale è in realta un vettore (ovvero l'indirizzo dell'elemento di indice 0), di solito si usa la notazione `nome-parametro[]` invece di `*nome-parametro`.

Esempio: `stampa(int v[], int dim) { ... }`

Si può anche specificare la dimensione nel parametro, ma viene ignorata.
Esempio: `stampa(int v[5], int dim) { ... }`

Nel prototipo della funzione può anche mancare il nome del vettore.
Esempio: `void stampa(int [], int);`

Il passaggio di un vettore è in realtà un *passaggio per indirizzo*
! La funzione può modificare gli elementi del vettore passato.

# Passaggio di parametri di tipo vettore

Esempio: Funzioni per la lettura e stampa di un vettore.

```
#define LUNG 5
void leggiVettore(double v[], int dim){
  int i;
  for (i = 0; i < dim; i++) {
    printf("immetti l'elemento di indice %d: ", i);
    scanf("%lg", &v[i]);
  }
}
void stampaVettore(double v[], int dim) {
  int i;
  printf("Indice Elemento\n");
  for (i = 0; i < dim; i++)
    printf("%6d %8g\n", i, v[i]);
}
int main(void) {
  double vet[LUNG]; leggiVettore(vet, LUNG);
  stampaVettore(vet, LUNG);
  return 0;
}
```

# Passaggio di parametri di tipo vettore

Esempio: Programma che legge un vettore di dimensione N e lo inverte.

```
void invertiVettore(int v[], int dim){
/* Inverte il vettore v di dimensione dim. Versione iterativa.*/
  int temp;  /* usata per scambiare due elementi del vettore */
  int i;
  for (i = 0; i < dim/2; i++){/* scambia vett[i] con vett[dim-1-i]*/
   temp = v[i];
   v[i] = v[dim-1-i];
   v[dim-1-i] = temp;
  }
}
int main() {
  int vett[LUNG];
  leggiVettore(vett, LUNG);
  invertiVettore(vett, LUNG);
  stampaVettore(vett, LUNG);
  invertiVettoreRic(vett, LUNG);
  stampaVettore(vett, LUNG);
  return 0;
}
```