Il linguaggio C
Puntatori

Moreno Marzolla
Dipartimento di Informatica—Scienza e Ingegneria (DISI)
Università di Bologna
http://www.moreno.marzolla.name/

Ringraziamenti

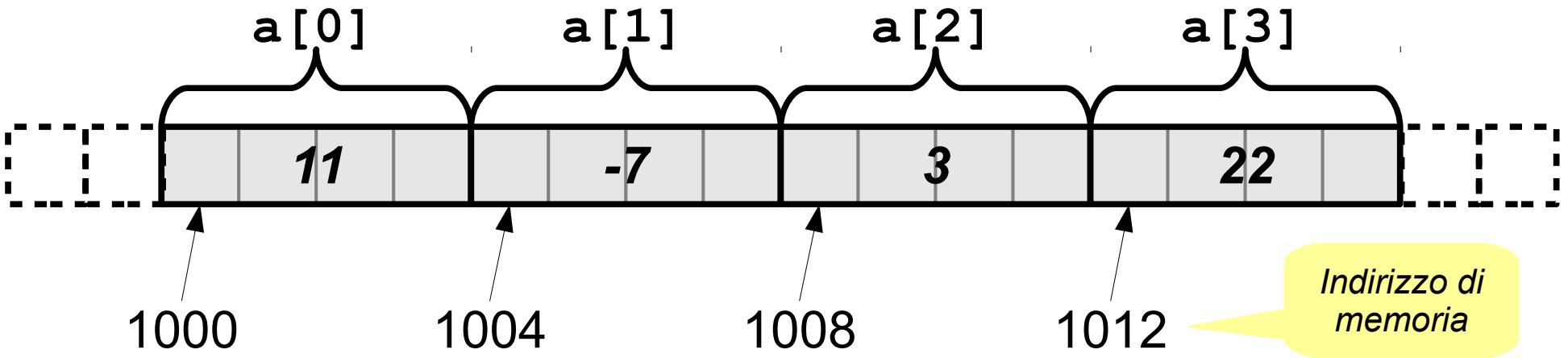
- Questi lucidi si basano su materiale fornito dal prof. Mirko Viroli, Università di Bologna

Dichiarazioni di array e matrici

```
int x[4]={11,12};                    /* OK, {11,12,0,0}          */
int y[4];                            /* OK, {?,?,?,?}            */
int z[] = {3, 5};                    /* OK, lunghezza 2          */
int m[3][2]={{11,12},{21,22},{31,32}}; /* OK                     */
int k[3][2]={{11}, {21, 22}};        /* OK, {{11,0},{21,22},{0,0}} */
int n[3][2];                         /* OK, {{?,?},{?,?},{?,?}} */
int p[][] = {{1,2}, {4,5}};          /* ERRORE                   */
int q[][2] = {{1,2}, {4,5}};         /* OK                       */
```

Come sono rappresentati gli array in C?

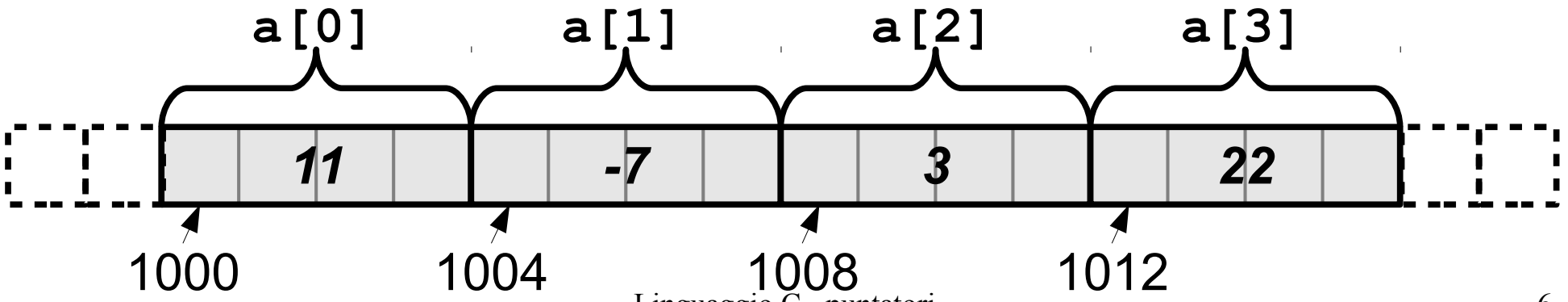
- Consideriamo **`int a[] = {11, -7, 3, 22}`**
- Supponiamo che un intero occupi 4 byte
  - Cioè **`sizeof(int) == 4`**
- Supponiamo che il primo elemento di **`a[]`** sia memorizzato a partire dall'indirizzo di memoria 1000



In generale

- Dato un array **`a[N]`** di valori di tipo *T*
- Supponiamo che **`sizeof(`***T***`)`** = *k*
- Sia Mem(**`a[i]`**) l'indirizzo di memoria a partire dal quale è memorizzato il valore **`a[i]`**
- Allora, per ogni 0 ≤ *i* < *N* si ha:

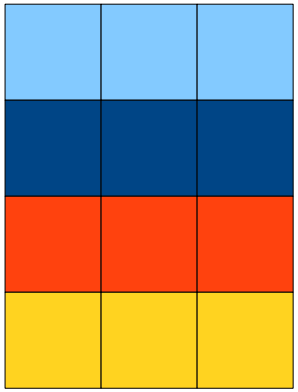
$$\text{Mem}(\texttt{a[i]}) = \text{Mem}(\texttt{a[0]}) + i \times k$$

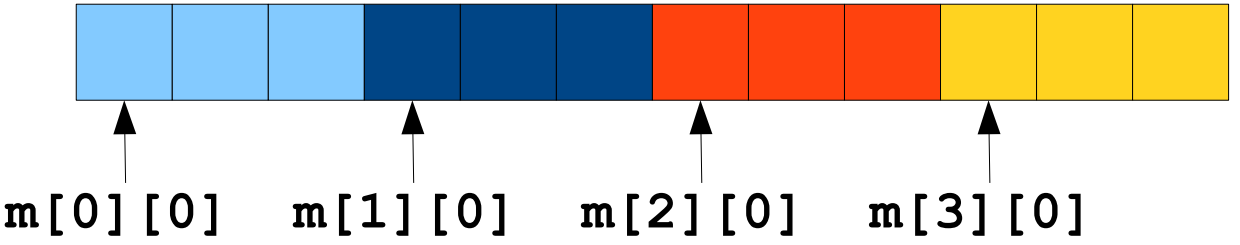


Array multidimensionali

- In C le matrici sono memorizzate "per righe"
  - Cioè le righe della matrice sono memorizzate in posizioni contigue della memoria

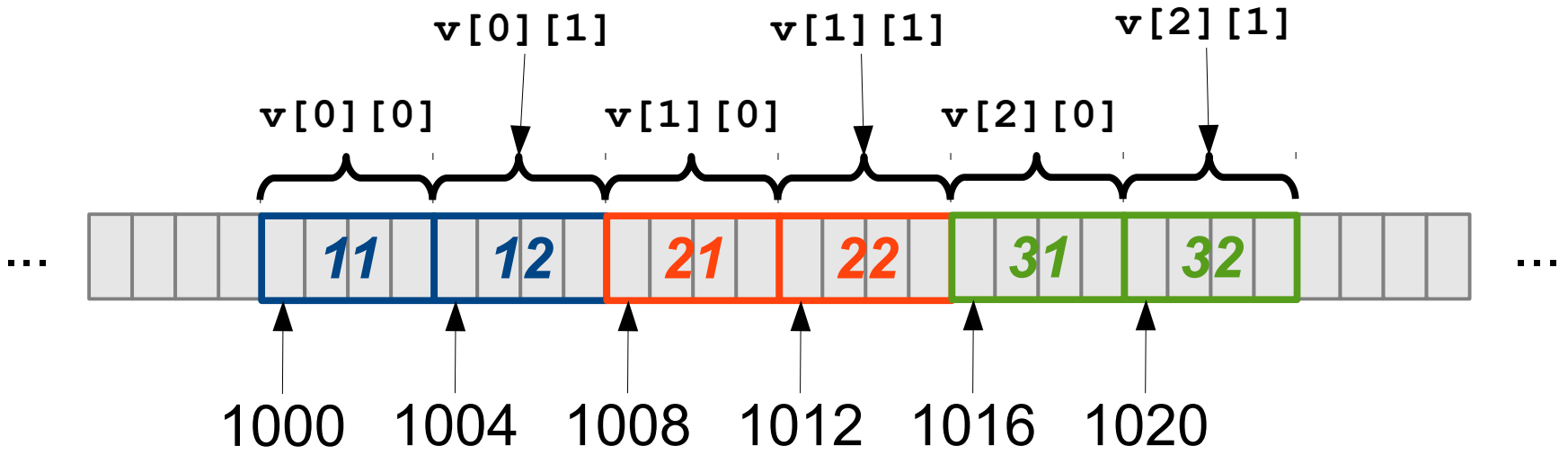
```
int m[4][3]
```





Array multidimensionali

- Consideriamo
  `int v[3][2]={{11,12},{21,22},{31,32}}`
- Supponiamo che `v[0][0]` sia memorizzato a partire dall'indirizzo di memoria 1000
- Supponiamo che `sizeof(int) == 4`



In generale

- Data una matrice `m[R][C]` di valori di tipo *T*
- Supponiamo che `sizeof(`*T*`)` = *k*
- Sia Mem(`m[i][j]`) l'indirizzo di memoria a partire dal quale è memorizzato il valore `m[i][j]`
- Allora, per ogni $0 \le i < R$, $0 \le j < C$ si ha:

$$\text{Mem}(\texttt{m[i][j]}) = \text{Mem}(\texttt{m[0][0]}) + (i \times C + j) \times k$$

Domanda

- Supponiamo che **`sizeof(double) == 8`**
- Dato un array **`double a[10]`**
  - Supponiamo che **`a[0]`** sia memorizzato a partire dall'indirizzo di memoria 2048
  - A partire da quale indirizzo di memoria si trova **`a[5]`**?
- Dato un array **`double m[3][7]`**
  - Supponiamo che **`m[0][0]`** sia memorizzato a partire dall'indirizzo di memoria 4096
  - A partire da quale indirizzo di memoria si trova **`m[2][4]`** ?

Stringhe (sequenze di caratteri) in C

- La notazione **`"prova di stringa"`** è convertita dal compilatore in un array di **`char`** che termina col carattere speciale **`'\0'`**, il cui valore è 0
  - Quindi **`"prova di stringa"`** occupa 17 byte di memoria: 16 per i caratteri + 1 per il terminatore
- È possibile calcolare la lunghezza di una stringa contando quanti caratteri ci sono prima del **`'\0'`**

Esempio stringhe

```
/* es-stringhe.c */
#include <stdio.h>

int main( void )
{
    char s1[6] = "Prova"; /* Ok, il compilatore alloca 6 elementi */
    char s2[] = "Prova";  /* Ok, il compilatore alloca 6 elementi */
    char s3[] = {'P','r','o','v','a','\0'};
    printf("Stampa diretta di s2 e s3 : %s, %s\n", s2, s3);
    printf("Elementi di s2: %c %c %c %c %c %c\n",
            s2[0], s2[1], s2[2], s2[3], s2[4], s2[5]);
    printf("Codici ASCII di s3: %d %d %d %d %d %d\n",
            s3[0], s3[1], s3[2], s3[3], s3[4], s3[5]);
    return 0;
}
```

```
Stampa diretta di s2 e s3 : Prova, Prova
Elementi di s2: P r o v a
Codici ASCII di s3: 80 114 111 118 97 0
```

Lunghezza di una stringa

```c
/* lunghezza-stringa.c */
#include <stdio.h>

int lunghezza(char c[])
{
    int i;
    for (i=0; c[i]!=0; i++) ; /* Corpo vuoto */
    return i;
}

int main(void)
{
    char s1[] = "Prova";
    char s2[] = {'P','r','o','v','a','\0'};
    printf("Lunghezza di s1: %d\n", lunghezza(s1));
    printf("Lunghezza di s2: %d\n", lunghezza(s2));
    return 0;
}
```

```
Lunghezza di s1: 5
Lunghezza di s2: 5
```

Lunghezza di una stringa

- Esiste una funzione predefinita **`strlen(s)`** che restituisce la lunghezza di **`s`**
  - È necessario includere **`string.h`**
  - Restituisce un **`size_t`**, che va convertito in **`int`** per poter essere stampato con la stringa di formato **`%d`**

```
/* es-strlen.c */
#include <stdio.h>
#include <string.h>

int main(void)
{
   char s[] = "Prova";
   printf("Lunghezza di s: %d\n", (int)strlen(s));
   return 0;
}
```

Puntatori: dichiarazione

- Un puntatore è una variabile che contiene un indirizzo di memoria in cui è presente un valore di un certo tipo
- Dichiarazione:
  - **`int *x;`** dichiara una variabile **`x`** contenente l'indirizzo di memoria di un valore di tipo **`int`**
  - Il tipo di **`x`** è "puntatore a intero"
- Attenzione:
  - **`int *x, *y;`**
    dichiara due variabili **`x`** e **`y`** di tipo "puntatore a intero"
  - **`int *x, y;`**
    dichiara una variabile **`x`** di tipo "puntatore a intero", e una variabile **`y`** di tipo "intero" (NON puntatore)

Operatore `&` (riferimento)

- Operatore unario applicabile a variabili
  - restituisce l'indirizzo della cella di memoria in cui è memorizzata quella variabile
- Se **`v`** è una variabile di tipo ***`T`***, **`&v`** ha tipo "puntatore a *T*"

```
/* punt.c */
#include <stdio.h>
int main(void)
{
    int i=5;  double d=1.5;  int a[5];
    int *pi = &i;        /* E' un puntatore alla variabile i          */
    double *pd = &d;     /* E' un puntatore alla variabile d          */
    int *pa4 = &a[4];    /* E' un puntatore al quinto elemento di a[] */
    printf("%p\n", (void*)pi);
    return 0;
}
```

```
0x7ffc68e726d8
```

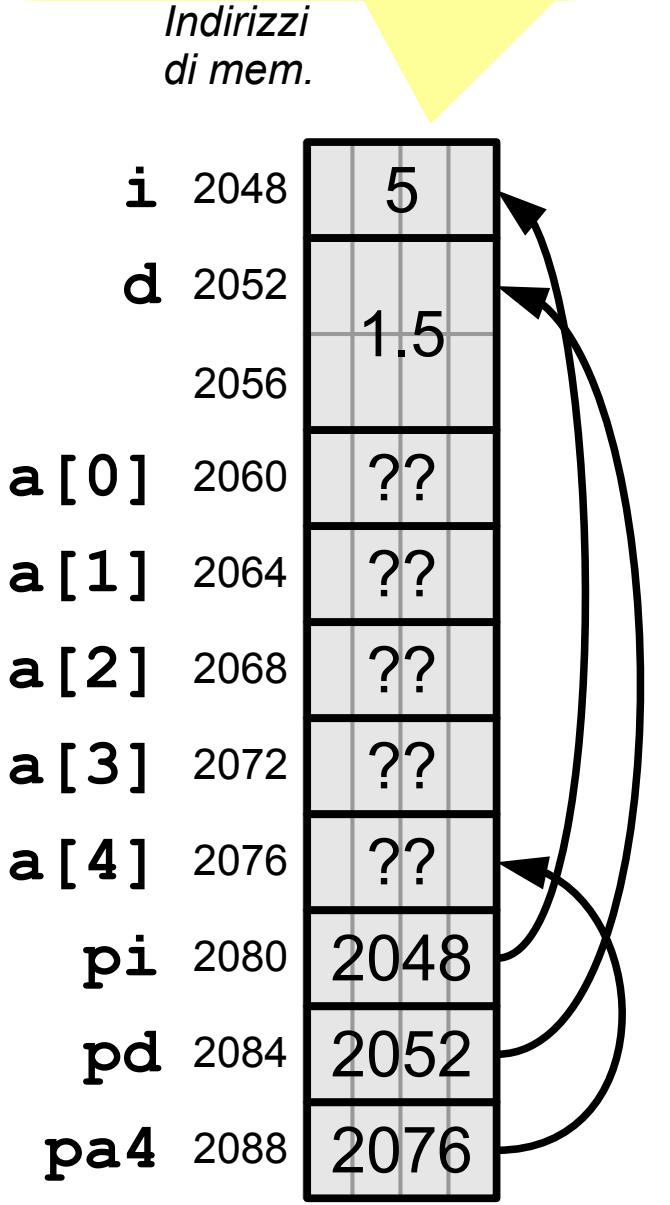
Esempio

```
/* punt.c */
#include <stdio.h>
int main(void)
{
    int i=5;  double d=1.5;  int a[5];
    int *pi = &i;
    double *pd = &d;
    int *pa4 = &a[4];
    printf("%p\n", (void*)pi);
    return 0;
}
```

Assumiamo che:
- un **`int`** occupi 4 byte
- un **`double`** occupi 8 byte
- un puntatore occupi 4 byte

Domanda: quale è la dimensione massima della memoria utilizzabile?

*Non necessariamente le variabili i, d, pi, pd, pa4 e a[] sono adiacenti in memoria; gli elementi di a[] sono invece **garantiti** essere adiacenti*



Operatore di *deriferimento* *

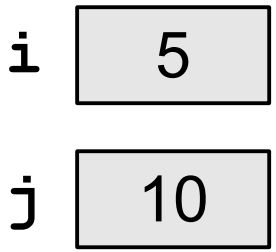
- Detto anche operatore di *indirezione*
  - dato un puntatore, l'operatore unario * restituisce il valore della variabile "puntata" dal puntatore

```
int main(void)
{
  int v = 5;
  int *p = &v; /* p punta alla variabile v     */
  int w = *p;  /* w ha valore 5               */
  *p = *p + 1; /* v diventa 6                 */
  return 0;
}
```

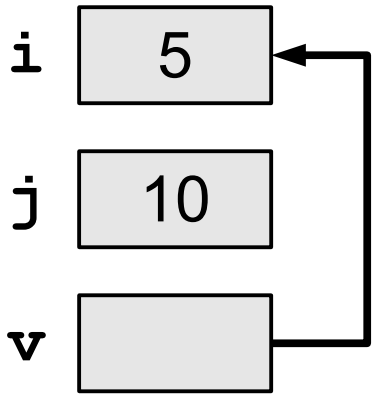
Operatore di *deriferimento* *

```
/* punt2.c */
#include <stdio.h>

int main(void)
{
  int i = 5, j = 10;
  int *v = &i; /* v punta alla variabile i */
  int *w;      /* w ha contenuto indefinito */
  w = &j;      /* w punta alla variabile j */
  *w = 4;      /* j viene modificato in 4 */
  *w = *v;     /* j viene modificato in 5, come i */
  w = v;       /* w punta a dove punta v, ossia i */
  *w = 7;      /* ora i vale 7 */
  printf("i=%d, j=%d, *w=%d, *v=%d\n", i, j, *w, *v);
  return 0;
}
```

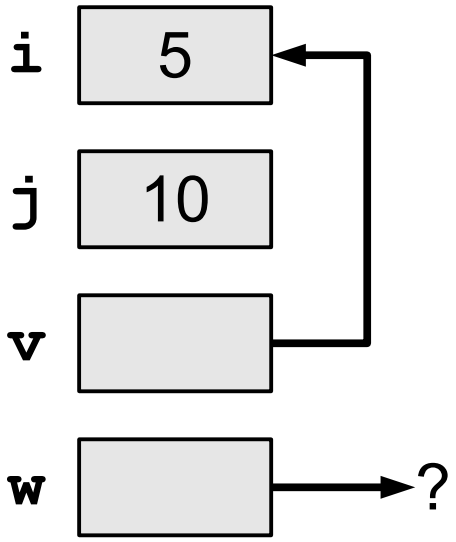
```c
/* punt2.c */
#include <stdio.h>

int main(void)
{
  int i = 5, j = 10;
  int *v = &i; /* v punta alla variabile i */
  int *w;      /* w ha contenuto indefinito */
  w = &j;      /* w punta alla variabile j */
  *w = 4;      /* j viene modificato in 4 */
  *w = *v;     /* j viene modificato in 5, come i */
  w = v;       /* w punta a dove punta v, ossia i */
  *w = 7;      /* ora i vale 7 */
  printf("i=%d, j=%d, *w=%d, *v=%d\n", i, j, *w, *v);
  return 0;
}
```



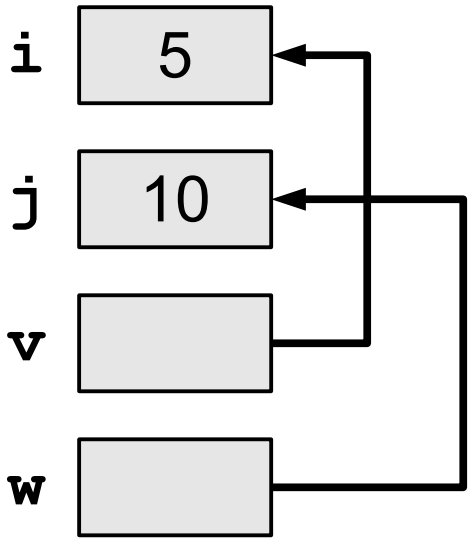
```c
/* punt2.c */
#include <stdio.h>

int main(void)
{
  int i = 5, j = 10;
  int *v = &i; /* v punta alla variabile i */
  int *w;      /* w ha contenuto indefinito */
  w = &j;      /* w punta alla variabile j */
  *w = 4;      /* j viene modificato in 4 */
  *w = *v;     /* j viene modificato in 5, come i */
  w = v;       /* w punta a dove punta v, ossia i */
  *w = 7;      /* ora i vale 7 */
  printf("i=%d, j=%d, *w=%d, *v=%d\n", i, j, *w, *v);
  return 0;
}
```



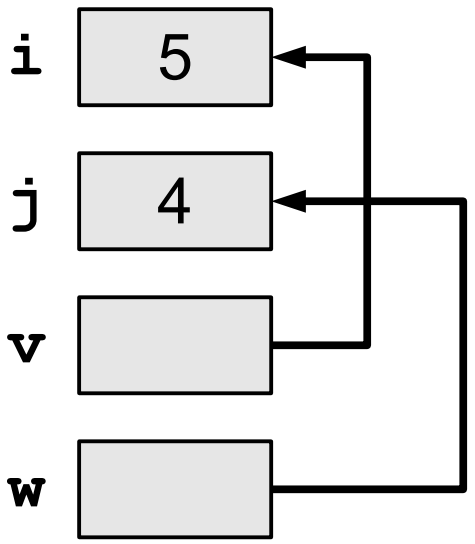
```c
/* punt2.c */
#include <stdio.h>

int main(void)
{
  int i = 5, j = 10;
  int *v = &i; /* v punta alla variabile i */
  int *w;      /* w ha contenuto indefinito */
  w = &j;      /* w punta alla variabile j */
  *w = 4;      /* j viene modificato in 4 */
  *w = *v;     /* j viene modificato in 5, come i */
  w = v;       /* w punta a dove punta v, ossia i */
  *w = 7;      /* ora i vale 7 */
  printf("i=%d, j=%d, *w=%d, *v=%d\n", i, j, *w, *v);
  return 0;
}
```



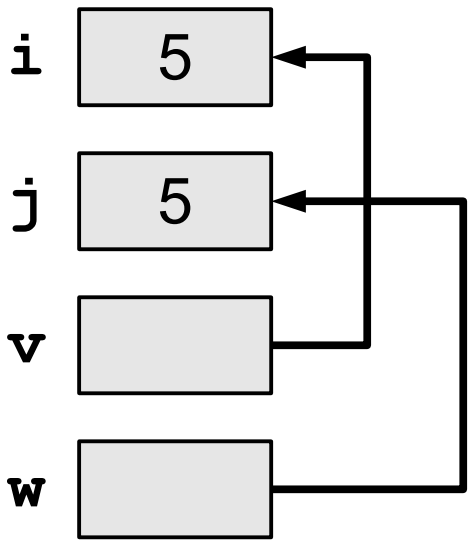
```c
/* punt2.c */
#include <stdio.h>

int main(void)
{
  int i = 5, j = 10;
  int *v = &i; /* v punta alla variabile i */
  int *w;      /* w ha contenuto indefinito */
  w = &j;      /* w punta alla variabile j */
  *w = 4;      /* j viene modificato in 4 */
  *w = *v;     /* j viene modificato in 5, come i */
  w = v;       /* w punta a dove punta v, ossia i */
  *w = 7;      /* ora i vale 7 */
  printf("i=%d, j=%d, *w=%d, *v=%d\n", i, j, *w, *v);
  return 0;
}
```



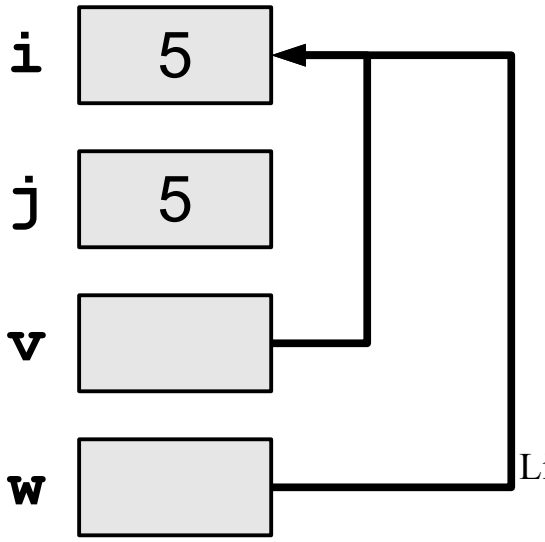
```c
/* punt2.c */
#include <stdio.h>

int main(void)
{
  int i = 5, j = 10;
  int *v = &i; /* v punta alla variabile i */
  int *w;      /* w ha contenuto indefinito */
  w = &j;      /* w punta alla variabile j */
  *w = 4;      /* j viene modificato in 4 */
  *w = *v;     /* j viene modificato in 5, come i */
  w = v;       /* w punta a dove punta v, ossia i */
  *w = 7;      /* ora i vale 7 */
  printf("i=%d, j=%d, *w=%d, *v=%d\n", i, j, *w, *v);
  return 0;
}
```



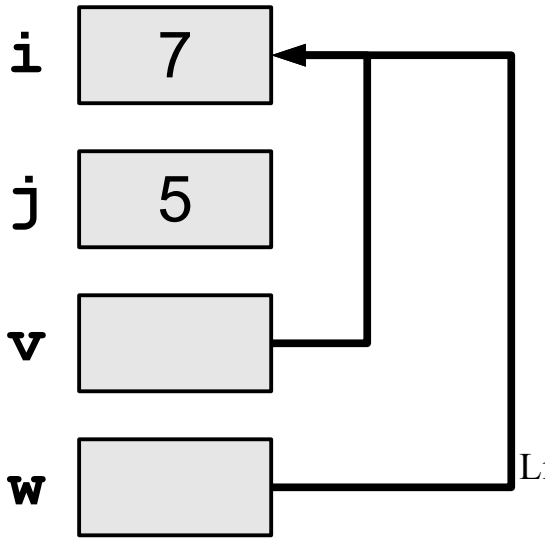
```c
/* punt2.c */
#include <stdio.h>

int main(void)
{
  int i = 5, j = 10;
  int *v = &i; /* v punta alla variabile i */
  int *w;      /* w ha contenuto indefinito */
  w = &j;      /* w punta alla variabile j */
  *w = 4;      /* j viene modificato in 4 */
  *w = *v;     /* j viene modificato in 5, come i */
  w = v;       /* w punta a dove punta v, ossia i */
  *w = 7;      /* ora i vale 7 */
  printf("i=%d, j=%d, *w=%d, *v=%d\n", i, j, *w, *v);
  return 0;
}
```



```c
/* punt2.c */
#include <stdio.h>

int main(void)
{
  int i = 5, j = 10;
  int *v = &i; /* v punta alla variabile i */
  int *w;      /* w ha contenuto indefinito */
  w = &j;      /* w punta alla variabile j */
  *w = 4;      /* j viene modificato in 4 */
  *w = *v;     /* j viene modificato in 5, come i */
  w = v;       /* w punta a dove punta v, ossia i */
  *w = 7;      /* ora i vale 7 */
  printf("i=%d, j=%d, *w=%d, *v=%d\n", i, j, *w, *v);
  return 0;
}
```



```c
/* punt2.c */
#include <stdio.h>

int main(void)
{
  int i = 5, j = 10;
  int *v = &i; /* v punta alla variabile i */
  int *w;      /* w ha contenuto indefinito */
  w = &j;      /* w punta alla variabile j */
  *w = 4;      /* j viene modificato in 4 */
  *w = *v;     /* j viene modificato in 5, come i */
  w = v;       /* w punta a dove punta v, ossia i */
  *w = 7;      /* ora i vale 7 */
  printf("i=%d, j=%d, *w=%d, *v=%d\n", i, j, *w, *v);
  return 0;
}
```

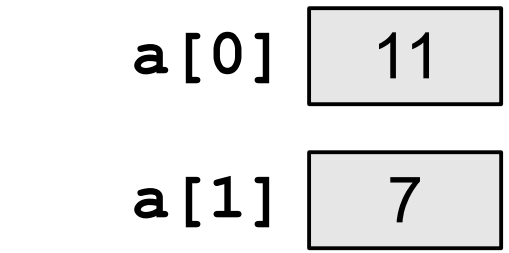


Esempio con gli array

```
int main(void)
{
  int a[] = {11, 7};
  int *w = &a[1]; /* w punta al secondo elemento dell'array  */
  int i = *w;  /* i viene assegnato a 7 */
  *w = 0; /* il secondo elem. dell'array viene posto a 0 */
  return 0;
}
```

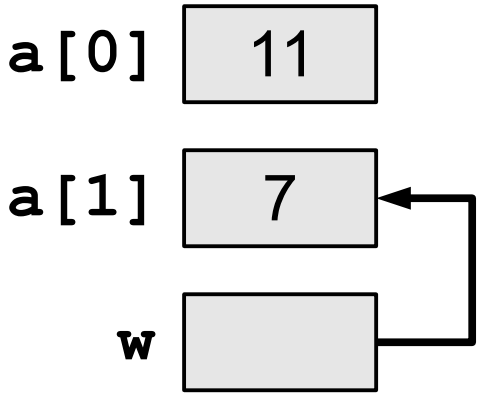
Esempio con gli array

```
int main(void)
{
  int a[] = {11, 7};
  int *w = &a[1]; /* w punta al secondo elemento dell'array  */
  int i = *w;  /* i viene assegnato a 7 */
  *w = 0; /* il secondo elem. dell'array viene posto a 0 */
  return 0;
}
```



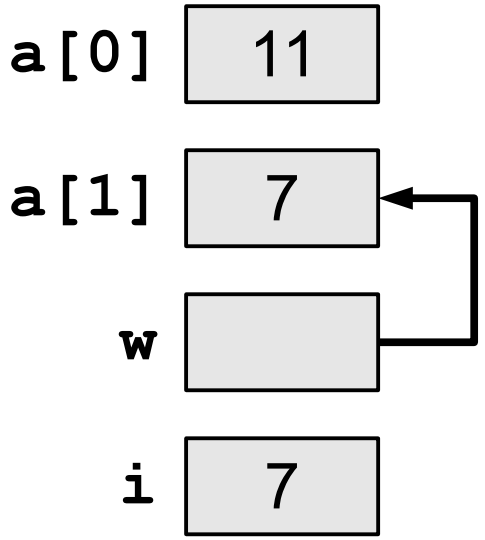
Esempio con gli array

```
int main(void)
{
  int a[] = {11, 7};
  int *w = &a[1]; /* w punta al secondo elemento dell'array  */
  int i = *w;  /* i viene assegnato a 7 */
  *w = 0; /* il secondo elem. dell'array viene posto a 0 */
  return 0;
}
```



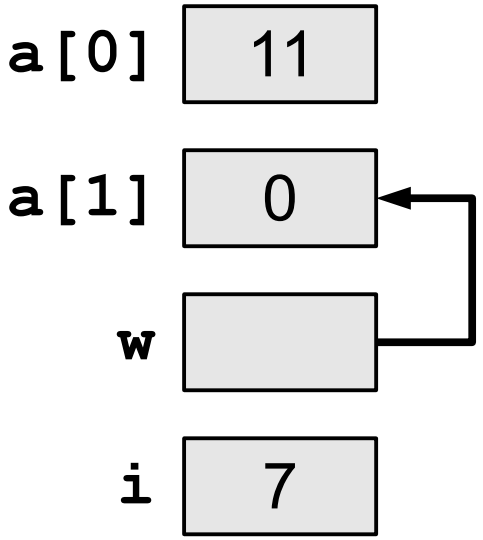
Esempio con gli array

```
int main(void)
{
  int a[] = {11, 7};
  int *w = &a[1]; /* w punta al secondo elemento dell'array  */
  int i = *w;  /* i viene assegnato a 7 */
  *w = 0; /* il secondo elem. dell'array viene posto a 0 */
  return 0;
}
```



Esempio con gli array

```
int main(void)
{
  int a[] = {11, 7};
  int *w = &a[1]; /* w punta al secondo elemento dell'array  */
  int i = *w;  /* i viene assegnato a 7 */
  *w = 0; /* il secondo elem. dell'array viene posto a 0 */
  return 0;
}
```



Puntatori: inizializzazione

- L'header file **`stddef.h`** definisce il simbolo **`NULL`** che può essere usato per denotare un indirizzo di memoria "non valido"
  - Può quindi essere usato come valore iniziale di un puntatore, se necessario
- **`stddef.h`** viene incluso indirettamente anche da altri header (es., **`stdio.h`**), quindi potrebbe non essere necessario includerlo esplicitamente

Esempio

```
/* null-ptr.c : esempio di uso di NULL */
#include <stdio.h>
#include <stddef.h>

int main( void )
{
    int a = 10, *ptr_a = NULL;
    ptr_a = &a;
    if ( ptr_a == NULL ) {
        printf("ptr_a non valido\n");
    } else {
        printf("ptr_a punta a %p che contiene il valore %d\n",
                (void*)ptr_a, *ptr_a);
    }
    return 0;
}
```

Può essere omesso, perché in gcc viene già incluso indirettamente da stdio.h

```
ptr_a punta a 0x7ffe4439c7d4 che contiene il valore 10
```

Input da tastiera

- La funzione **`scanf`**, duale a **`printf`**, viene usata per ricevere input da tastiera e assegnarne il valore ad una variabile
  - bisogna passargli il puntatore alla variabile dove si vuole venga memorizzato il risultato

```
#include <stdio.h>

int main(void)
{
  int x;
  float f;
  printf("Inserisci un intero: "); scanf("%d",&x);
  printf("Inserisci un float: ");  scanf("%f",&f);
  printf("x=%d, f=%f, prodotto=%f\n", x, f, x*f);
  return 0;
}
```

Input di stringhe

- Le stringhe andranno depositate in un array di caratteri
  - scanf vuole l'indicazione **`%s`** o **`%Ns`**
  - come secondo argomento mettere l'array

```
#include <stdio.h>

int main(void)
{
  char str1[11], str2[11];
  printf("Inserisci una stringa: ");
  scanf("%s", str1); /* non usare &str1 !!! */
  printf("Inserisci una stringa di non più di 10 caratteri: ");
  scanf("%10s", str2); /* nota: char str2[11] per includere il \0 finale */
  printf("str1: %s\n", str1);
  printf("str2: %s\n", str2);
  return 0;
}
```

Array e puntatori

- Un array di *T* si comporta come un puntatore
  - Se **`a`** è una variabile dichiarata come ***`T a[N]`*** (array di *T*)...
  - ...l'espressione **`a`** (senza indici) ha tipo **`T*`** (puntatore a *T*) e ha come valore l'indirizzo di memoria di **`a[0]`**
- Un puntatore può essere "indicizzato" come se fosse un array
  - Se **`v`** è dichiarata come ***`T *v`*** (puntatore a *T*) e valore un indirizzo di memoria *M*...
  - ...allora l'espressione **`v+k`** (*k* intero) ha tipo "puntatore a *T*" e contiene l'indirizzo di memoria

$$M + k \times \texttt{sizeof}(T)$$

  - ...e l'espressione **`v[k]`** (*k* intero) ha tipo *T* e valore pari a quello memorizzato a partire dall'indirizzo $M + k \times \texttt{sizeof}(T)$

Aritmetica dei puntatori

```
/* arit-punt.c */
#include <stdio.h>

int main(void)
{
  int a[] = {15, 16, 17};
  int *p, *q;
  p = a;       /* Ok, e' come scrivere p = &a[0]     */
  *p = -1;     /* ora l'array a[] vale {-1, 16, 17} */
  q = p + 1;   /* q contiene l'indirizzo di a[1]     */
  q++;         /* q contiene l'indirizzo di a[2]     */
  *q = -3;     /* ora l'array a[] vale {-1, 16, -3} */
  *(p+1) = 2; /* ora l'array a[] vale {-1, 2, -3}   */
  printf("%d %d %d\n", a[0], a[1], a[2]);
  printf("%d\n", *(p+2));
  return 0;
}
```

Attenzione

- Se **`p`** è una variabile di tipo puntatore, allora in genere

`*(p+1)` ≠ `*p + 1`

- Esempio: cosa stampa il programma seguente?

```
/* arit-punt2.c */
#include <stdio.h>

int main(void)
{
  int a[] = {1, 21, -3};
  int *p = a;
  printf("%d\n", *(p+1));
  printf("%d\n", *p + 1);
  return 0;
}
```

Attenzione

- Da quanto detto in precedenza dovrebbe essere chiaro perché non sia possibile controllare se due array **`a[]`** e **`b[]`** contengono gli stessi valori scrivendo **`(a == b)`**
  - Il programma seguente stampa "falso": perché?

```
/* confronto.c */
#include <stdio.h>

int main(void)
{
    int a[] = {1, 21, -3};
    int b[] = {1, 21, -3};
    printf("%s\n", (a == b ? "vero" : "falso"));
    return 0;
}
```

Metodo corretto per confrontare il contenuto di due array

```
/* arrays_equal.c */
#include <stdio.h>

/* restituisce 1 se e solo se a[] e b[] hanno lo stesso contenuto */
int arrays_equal(int a[], int b[], int n)
{
    int i;
    for (i=0; i<n; i++) {
        if (a[i] != b[i]) {
            return 0;
        }
    }
    return 1;
}

int main(void)
{
    int a[] = {1, 21, -3};
    int b[] = {1, 21, -3};
    printf("%s\n", (arrays_equal(a, b, 3) ? "vero" : "falso"));
    return 0;
}
```

Array e funzioni

- Le funzioni seguenti sono equivalenti e si invocano tutte allo stesso modo

```
int sum1(int a[], int n)
{
    int i, s=0;
    for (i=0; i<n; i++) {
        s += a[i];
    }
    return s;
}
```

```
int sum2(int *a, int n)
{
    int i, s=0;
    for (i=0; i<n; i++) {
        s += a[i];
    }
    return s;
}
```

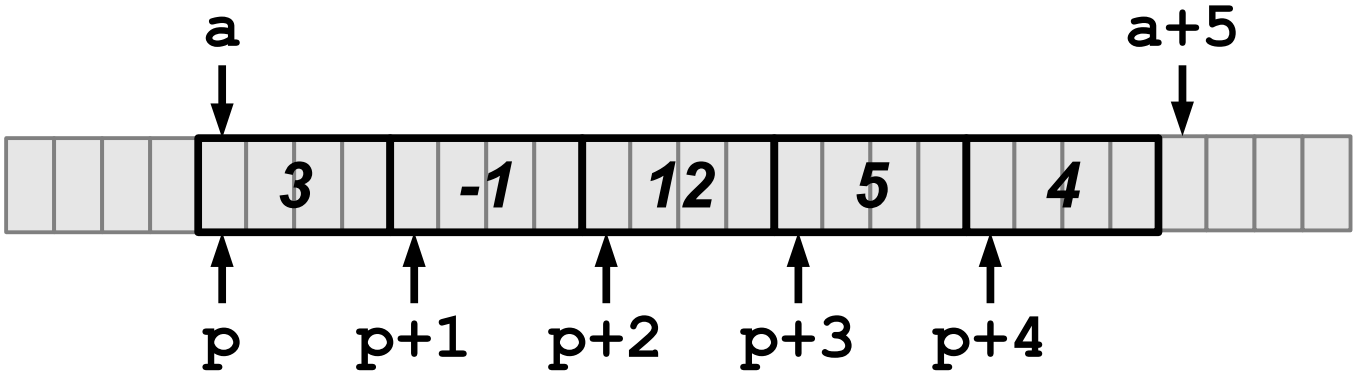
```
int sum3(int *a, int n)
{
    int i, s=0;
    for (i=0; i<n; i++) {
        s += *(a+i);
    }
    return s;
}
```

```
int sum4(int *a, int n)
{
    int *p, s=0;
    for( p=a; p<a+n; p++) {
        s += *p;
    }
    return s;
}
```

Spiegazione di `sum4()`

- Supponiamo
  - `sizeof(int) == 4`
  - `a[] = {3, -1, 12, 5, 4}`

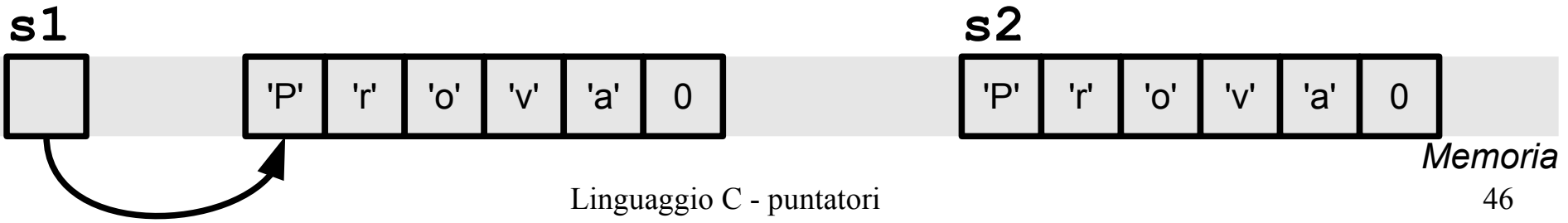
```
int sum4(int *a, int n)
{
    int *p, s=0;
    for( p=a; p<a+n; p++) {
        s += *p;
    }
    return s;
}
```



Puntatori a carattere

- L'"equivalenza" tra array e puntatori consente di manipolare le stringhe (sequenze di caratteri) sia come array di **`char`**, sia come puntatori a **`char`**

```
/* lunghezza2.c */
#include <stdio.h>

int lunghezza( char* s ) {
    int n;
    for (n=0; *s; s++) { n++; }
    return n;
}

int main( void )
{
    char *s1 = "Prova";
    char s2[] = "Prova";
    printf("%d %d\n",
           lunghezza(s1),
           lunghezza(s2));
    return 0;
}
```

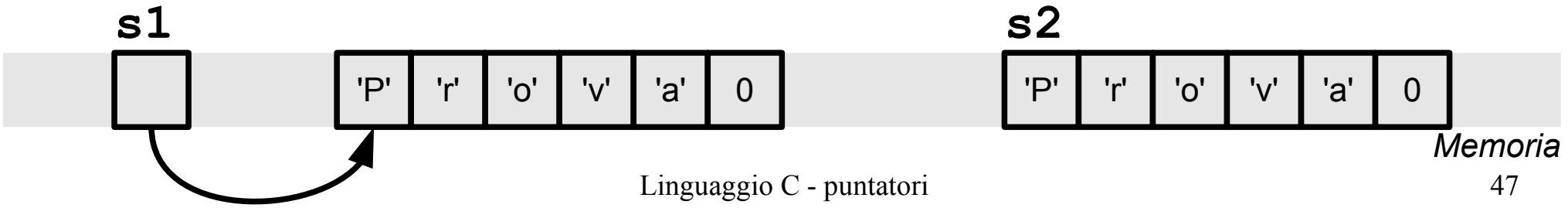


Attenzione

- C'è una differenza importante tra **`s1`** e **`s2`**
  - Modificare il contenuto di **`s1`** causa un comportamento indefinito
  - Modificare il contenuto di **`s2`** è ammesso

Solo in questo caso, i caratteri del letterale "Prova" vengono memorizzati in un'area di memoria che si deve immaginare "a sola lettura"

```
/* indefinito.c */
#include <stdio.h>

int main( void )
{
    char *s1 = "Prova";
    char s2[] = "Prova";
    *s1 = 'p'; /* INDEFINITO */
    s2[0] = 'p'; /* Ok */
    printf("%s %s\n", s1, s2);
    return 0;
}
```



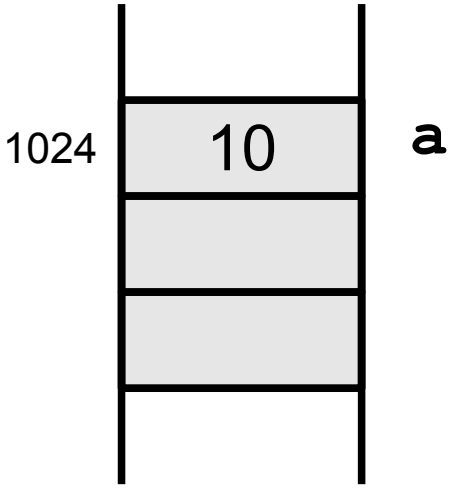
Passaggio per puntatore

- Una applicazione importante dei puntatori è per "modificare" il parametro attuale
  - non si passa il “cosa” ma il “dove”

```
/* punt-parametro.c */
#include <stdio.h>

void incrementa(int *x)
{
    *x = *x + 1;
}

int main( void )
{
    int a = 10;
    incrementa(&a);
    printf("a = %d\n", a);
    return 0;
}
```

Passaggio per puntatore

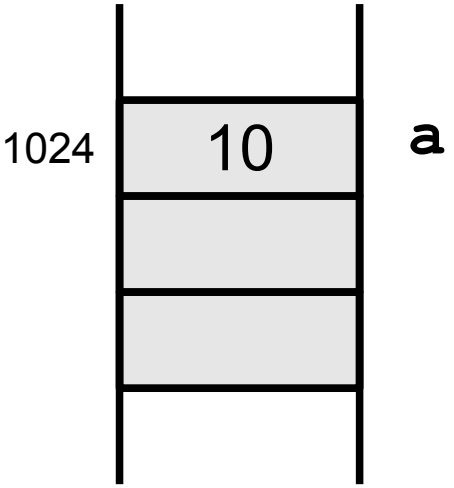
```
/* punt-parametro.c */
#include <stdio.h>

void incrementa(int *x)
{
    *x = *x + 1;
}

int main( void )
{
    int a = 10;
    incrementa(&a);
    printf("a = %d\n", a);
    return 0;
}
```



Passaggio per puntatore

```
/* punt-parametro.c */
#include <stdio.h>

void incrementa(int *x)
{
    *x = *x + 1;
}

int main( void )
{
    int a = 10;
    incrementa(&a);
    printf("a = %d\n", a);
    return 0;
}
```

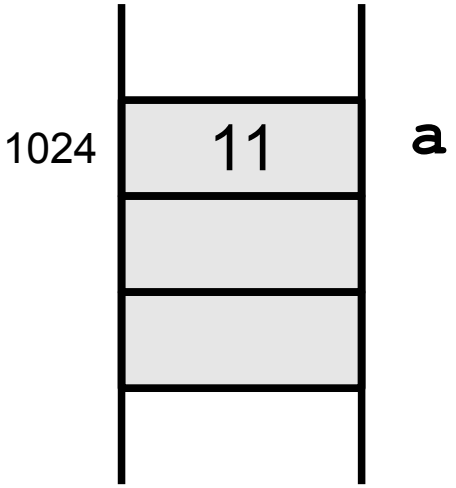
```
x = 1024

void incrementa(int *x)
{
    *x = *x + 1;
}
```



Passaggio per puntatore

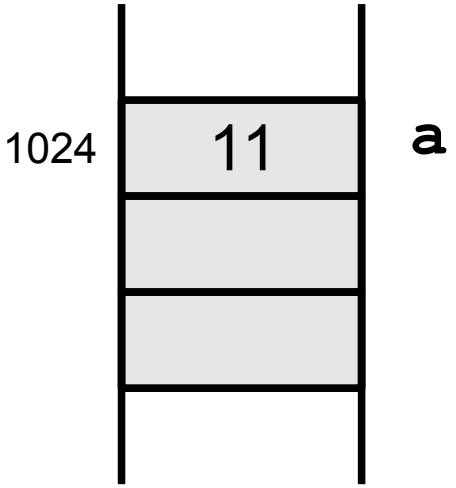
```
/* punt-parametro.c */
#include <stdio.h>

void incrementa(int *x)
{
    *x = *x + 1;
}

int main( void )
{
    int a = 10;
    incrementa(&a);
    printf("a = %d\n", a);
    return 0;
}
```

```
x = 1024

void incrementa(int *x)
{
    *x = *x + 1;
}
```



Passaggio per puntatore

```
/* punt-parametro.c */
#include <stdio.h>

void incrementa(int *x)
{
    *x = *x + 1;
}

int main( void )
{
    int a = 10;
    incrementa(&a);
    printf("a = %d\n", a);
    return 0;
}
```

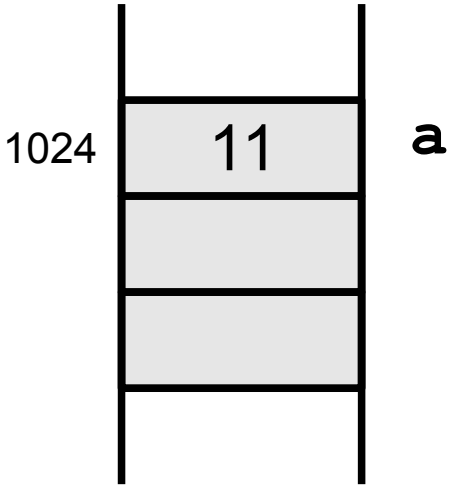


x = 1024

```
void incrementa(int *x)
{
    *x = *x + 1;
}
```

Passaggio per puntatore

```
/* punt-parametro.c */
#include <stdio.h>

void incrementa(int *x)
{
    *x = *x + 1;
}

int main( void )
{
    int a = 10;
    incrementa(&a);
    printf("a = %d\n", a);
    return 0;
}
```

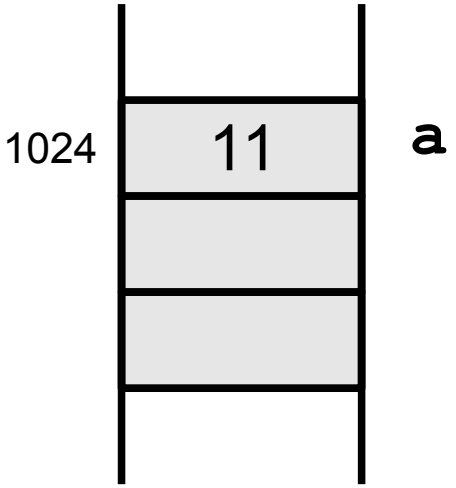


Output

```
a = 11
```

Passaggio per puntatore

```
/* punt-parametro.c */
#include <stdio.h>

void incrementa(int *x)
{
    *x = *x + 1;
}

int main( void )
{
    int a = 10;
    incrementa(&a);
    printf("a = %d\n", a);
    return 0;
}
```



Output

```
a = 11
```