Introduzione al C

Lezione 3

Puntatori, array e stringhe

Rossano Venturini

rossano@di.unipi.it

Pagina web del corso

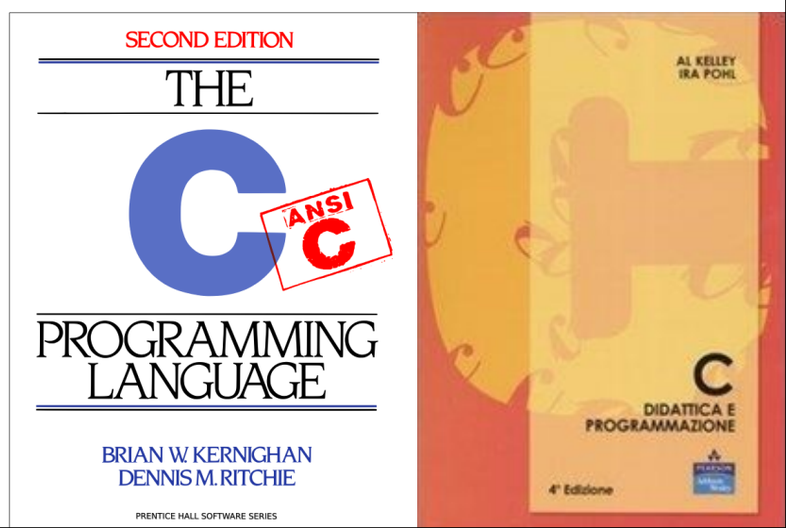
http://didawiki.cli.di.unipi.it/doku.php/informatica/all-b/start

Lezioni di ripasso C

Mercoledì 26	11-13	Aula A-B
Giovedì 27	16-18	Aula A-B

Le successive lezioni di laboratorio saranno

Corso B Giovedì	14-16	Aula H-M
Corso A Giovedì	16-18	Aula H-M



Esercizio 1

Somma dispari

Esercizio

Scrivere una funzione ricorsiva f che, dato un intero N, restituisca la somma dei primi N interi dispari. Scrivere un programma che prenda in input un intero x e stampi il valore di $f(x)$.
L'unica riga dell'input contiene il valore x.
L'unica riga dell'output contiene la somma dei primi x numeri dispari.

Esempio

Input	Output
6	36

Esercizio 1

```
#include <stdio.h>

int odd_sum(int i){
      if (i <= 0 ) return 0;
      return (i*2-1)+odd_sum(i-1);
}

int main(void){
      int number, res;
      scanf("%d", &number);
      res = odd_sum(number);
      printf("%d\n",res);
      return 0;
}
```

Esercizio 1

```
#include <stdio.h>

int odd_sum(int i){
    if (i <= 0 ) return 0;
    return (i*2-1)+odd_sum(i-1);
}

int main(void){
    int number, res;
    scanf("%d", &number);
    res = odd_sum(number);
    printf("%d\n",res);
    return 0;
}
```

Esercizio 1

```
#include <stdio.h>

int odd_sum(int i){
      if (i <= 0 ) return 0;
      return (i*2-1)+odd_sum(i-1);
}

int main(void){
      int number, res;
      scanf("%d", &number);
      res = odd_sum(number);
      printf("%d\n",res);
      return 0;
}
```

Esercizio 4 MinMax

Esercizio

Scrivere una funzione `minmax` avente i seguenti parametri

- un array di interi;
- la lunghezza dell'array;
- un puntatore a una variabile intera `min`;
- un puntatore a una variabile intera `max`.

La funzione scandisce l'array e salva in `min` la posizione in cui si trova l'elemento minimo e in `max` la posizione in cui si trova l'elemento massimo. Si può assumere che l'array contenga valori distinti.

Scrivere poi un programma che

- legga 10 interi da tastiera;
- invochi `minmax` sull'array letto;
- produca in output: la posizione dell'elemento minimo, il valore dell'elemento minimo, la posizione dell'elemento massimo, il valore dell'elemento massimo.

Esercizio 4

```
#include <stdio.h>

#define N (10)

void minmax(int a[], int len, int *min, int *max){
      int i;
      *min = 0; // minimo in posizione 0
      *max = 0; // massimo in posizione 0

      for (i = 0; i < len; i++){
            if ( a[i] < a[*min] ) *min = i;
            if ( a[i] > a[*max] ) *max = i;
      }
}
```

Esercizio 4

```
#include <stdio.h>

#define N (10)

void minmax(int a[], int len, int *min, int *max){
      int i;
      *min = 0; // minimo in posizione 0
      *max = 0; // massimo in posizione 0

      for (i = 0; i < len; i++){
            if ( a[i] < a[*min] ) *min = i;
            if ( a[i] > a[*max] ) *max = i;
      }
}
```

Esercizio 4

```
int main(void) {
    int i = 0, min, max;

    int array[N];
    for (i = 0; i < N; i++){
        scanf("%d",&array[i]);
    }

    minmax(array, N, &min, &max);
    printf("%d\n",min);
    printf("%d\n",array[min]);
    printf("%d\n",max);
    printf("%d\n",array[max]);

    return 0;
}
```

Esercizio 4

```
int main(void) {
    int i = 0, min, max;

    int array[N];
    for (i = 0; i < N; i++){
        scanf("%d",&array[i]);
    }

    minmax(array, N, &min, &max);
    printf("%d\n",min);
    printf("%d\n",array[min]);
    printf("%d\n",max);
    printf("%d\n",array[max]);

    return 0;
}
```

Esercizio 4: Ok così?

```
int main(void) {
    int i = 0, *min, *max;

    int array[N];
    for (i = 0; i < N; i++){
        scanf("%d",&array[i]);
    }

    minmax(array, N, min, max);
    printf("%d\n",min);
    printf("%d\n",array[min]);
    printf("%d\n",max);
    printf("%d\n",array[max]);

    return 0;
}
```

Array e puntatori

```
int a[5];
```

Memoria

nome	*celle*	*indirizzo*
	...	
		0x100
		0x104
		0x108
		0x112
		0x116
		0x120
		0x124
		0x128
	...	

Array e puntatori

```
int a[5];
```

Memoria

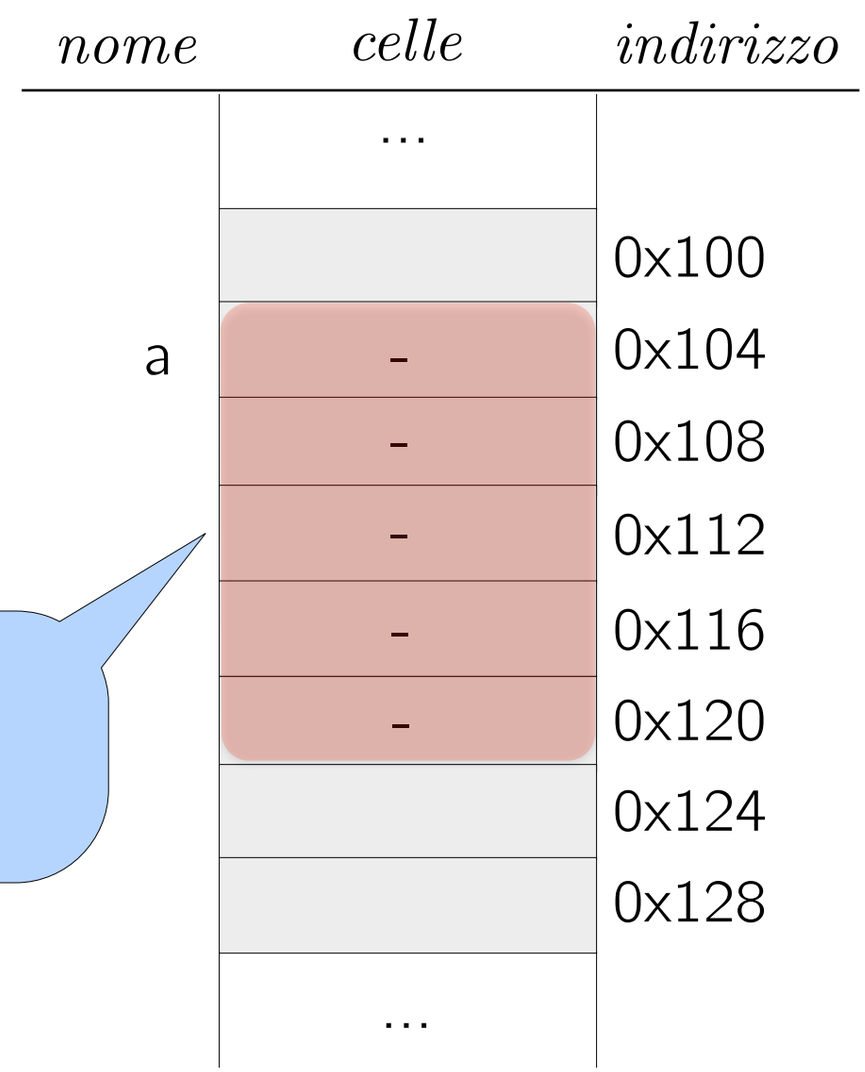
nome	*celle*	*indirizzo*
	...	
		0x100
a	-	0x104
	-	0x108
	-	0x112
	-	0x116
	-	0x120
		0x124
		0x128
	...	

Array e puntatori

```
int a[5];
```

Riservato per contentere
i 5 elementi di a

Memoria



Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
    a[i] = 0;
```

Memoria

nome	*celle*	*indirizzo*
	...	
		0x100
a	-	0x104
	-	0x108
	-	0x112
	-	0x116
	-	0x120
		0x124
		0x128
	...	

Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
    a[i] = 0;
```

Memoria

nome	*celle*	*indirizzo*
	...	
		0x100
a	0	0x104
	0	0x108
	0	0x112
	0	0x116
	0	0x120
		0x124
		0x128
	...	

Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
    a[i] = 0;
```

a è un puntatore **costate** al primo elemento dell'array.

Memoria

nome	*celle*	*indirizzo*
	...	
		0x100
a	0	0x104
	0	0x108
	0	0x112
	0	0x116
	0	0x120
		0x124
		0x128
	...	

Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
      a[i] = 0;

a = &x;   NO! a non può essere modificato.
```

a è un puntatore **costate** al primo elemento dell'array.

Memoria

nome	*celle*	*indirizzo*
	...	
		0x100
a	0	0x104
	0	0x108
	0	0x112
	0	0x116
	0	0x120
		0x124
		0x128
	...	

Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
    a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;      oppure int *p = &a[0];
```

a è un puntatore **costate** al primo elemento dell'array.

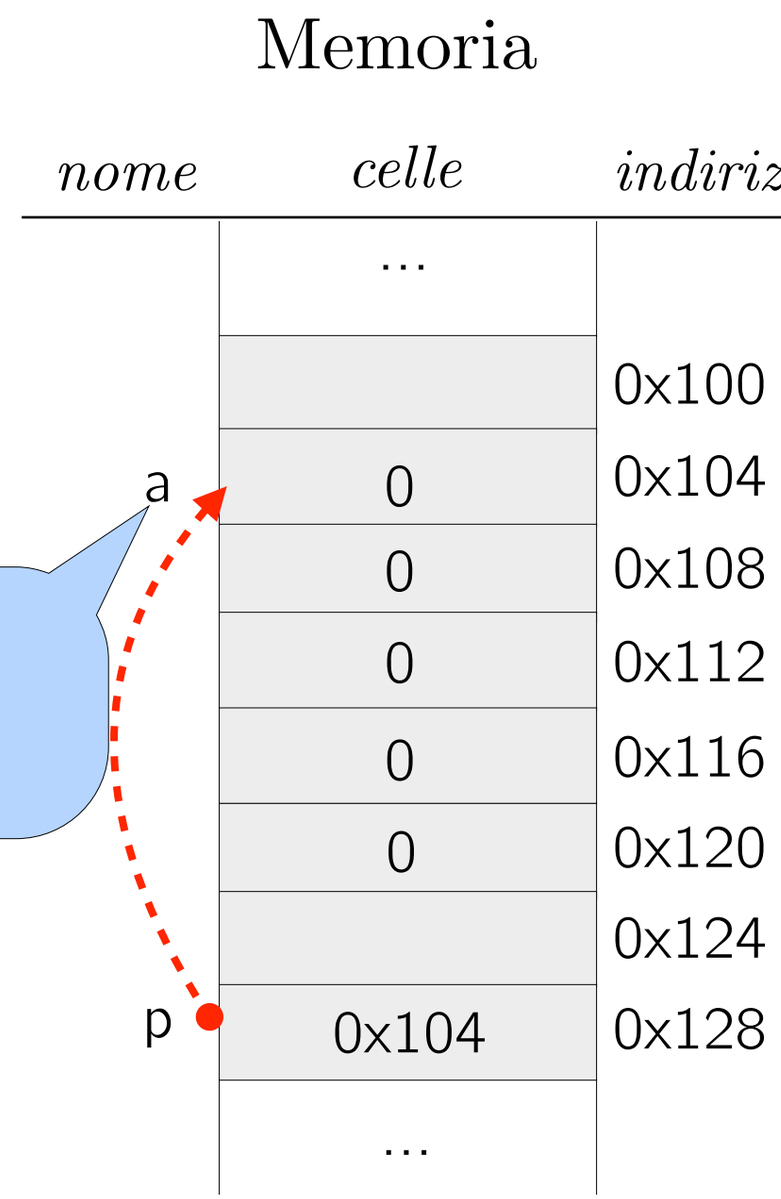
Memoria

nome	*celle*	*indirizzo*
	...	
		0x100
a	0	0x104
	0	0x108
	0	0x112
	0	0x116
	0	0x120
		0x124
		0x128
	...	

Array e puntatori

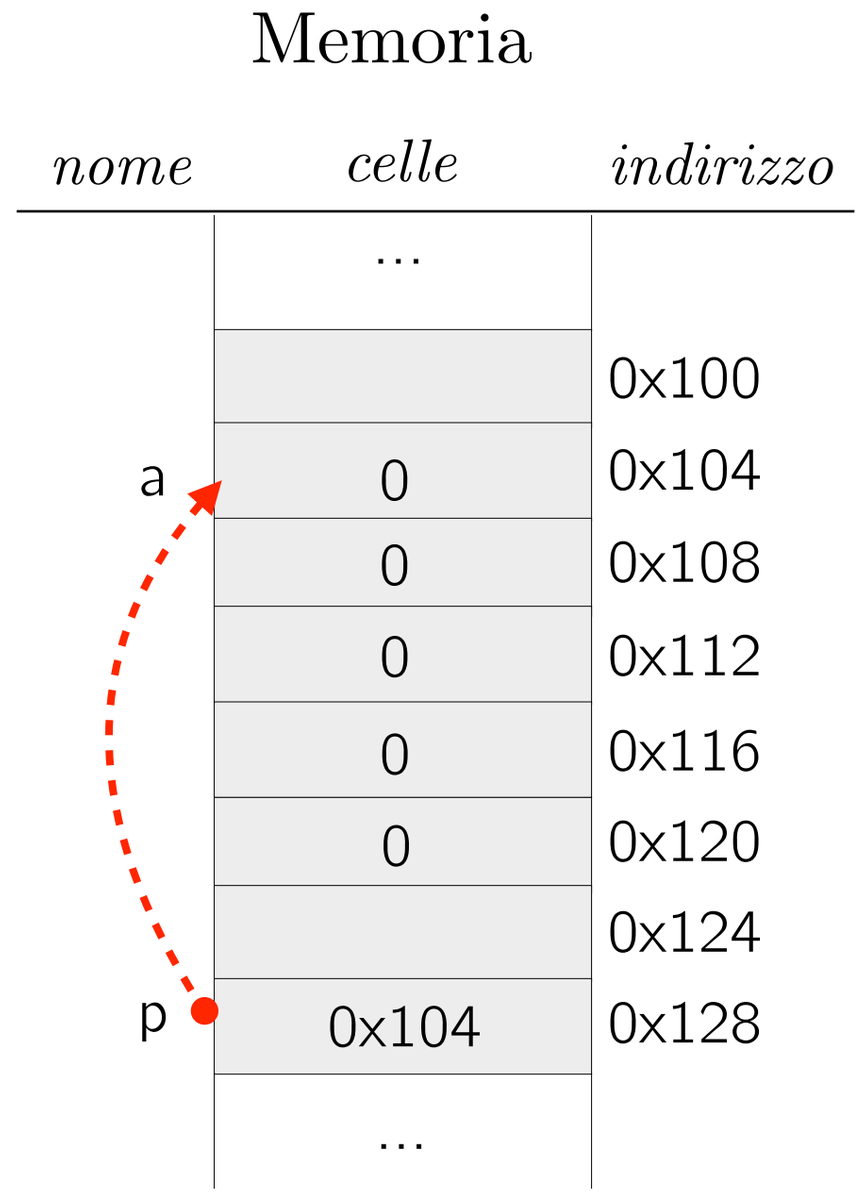
```
int a[5];

for ( i = 0; i < 5; i++)
      a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;      oppure int *p = &a[0];
```

a è un puntatore **costate** al primo elemento dell'array.



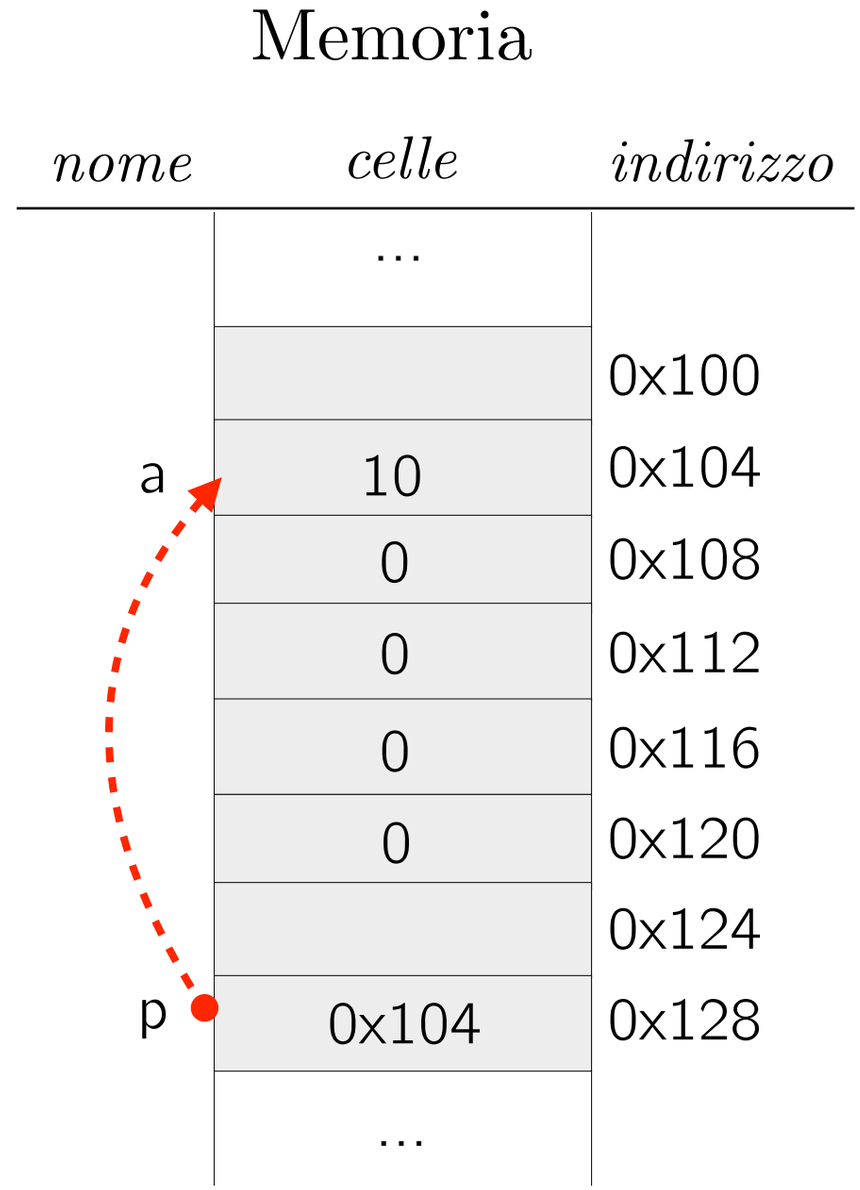
Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
      a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;       oppure int *p = &a[0];

*p = 10;
```



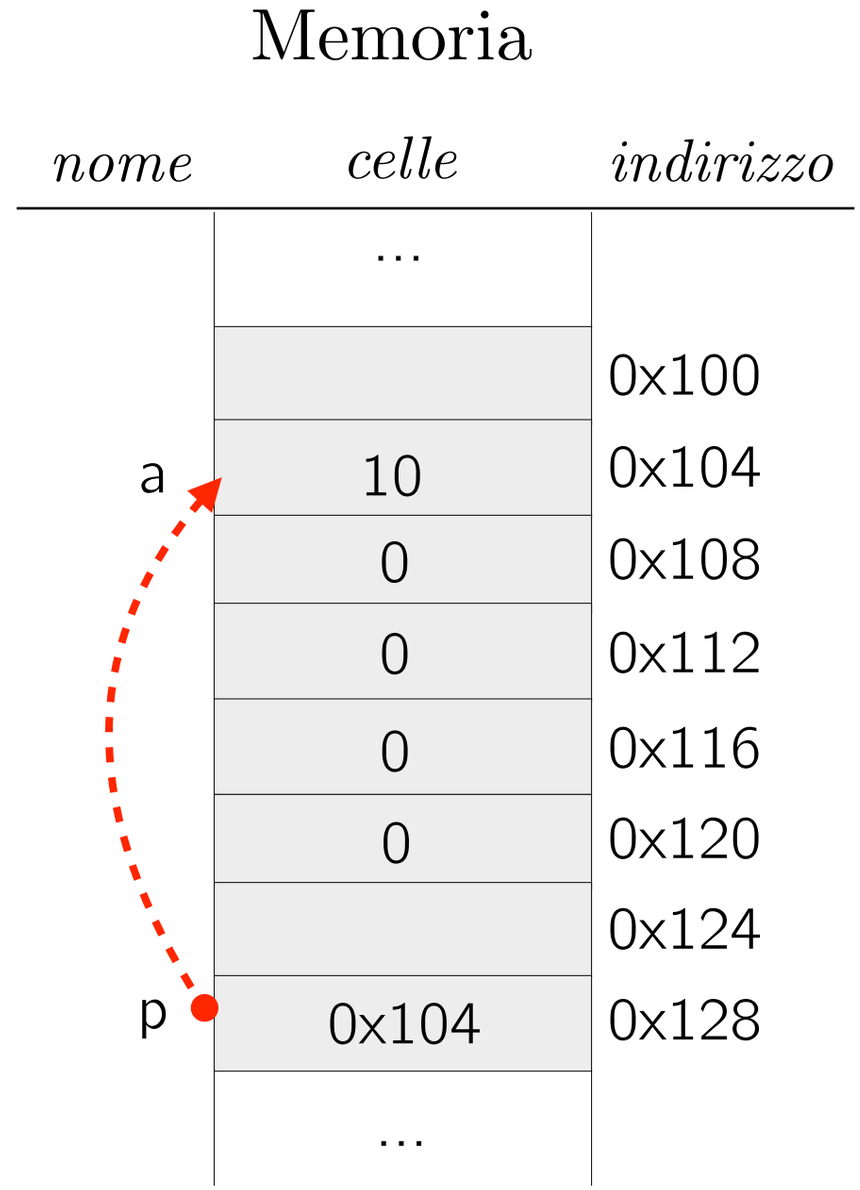
Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
      a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;       oppure int *p = &a[0];

*p = 10;
```



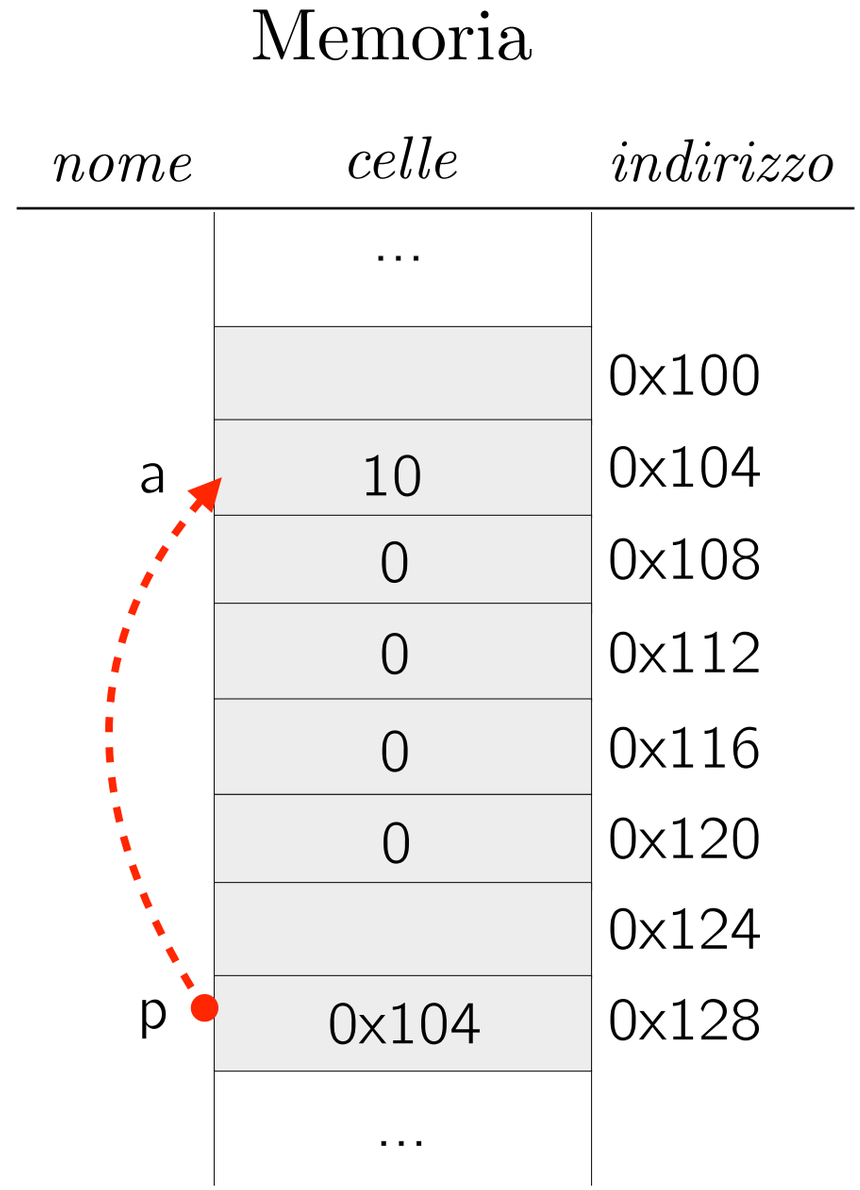
Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
    a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;      oppure int *p = &a[0];

*p = 10;

p[0] = 10;    3 forme equivalenti!

a[0] = 10;
```



Array e puntatori

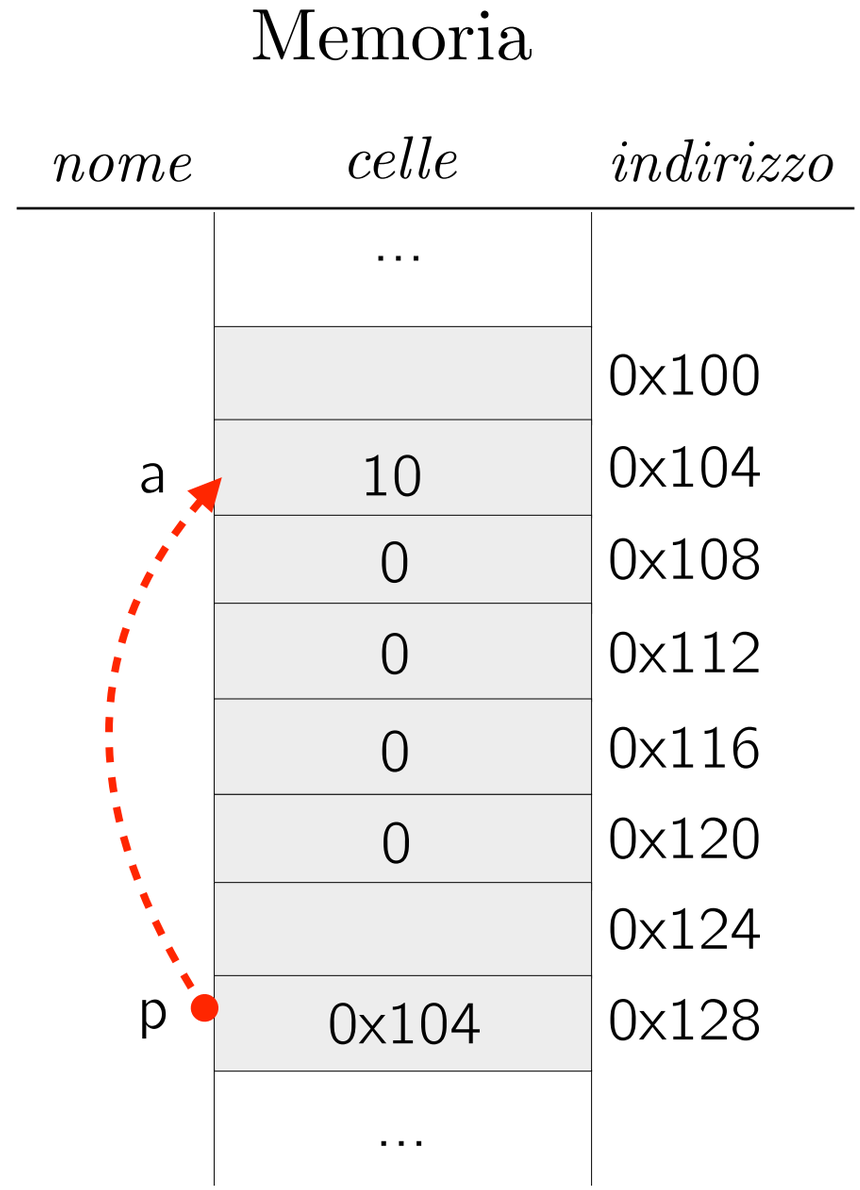
```
int a[5];

for ( i = 0; i < 5; i++)
    a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;      oppure int *p = &a[0];

*p = 10;

p[0] = 10;    3 forme equivalenti!

a[0] = 10;

p+1;
```



Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
    a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;      oppure int *p = &a[0];

*p = 10;

p[0] = 10;    3 forme equivalenti!

a[0] = 10;

p+1;
```

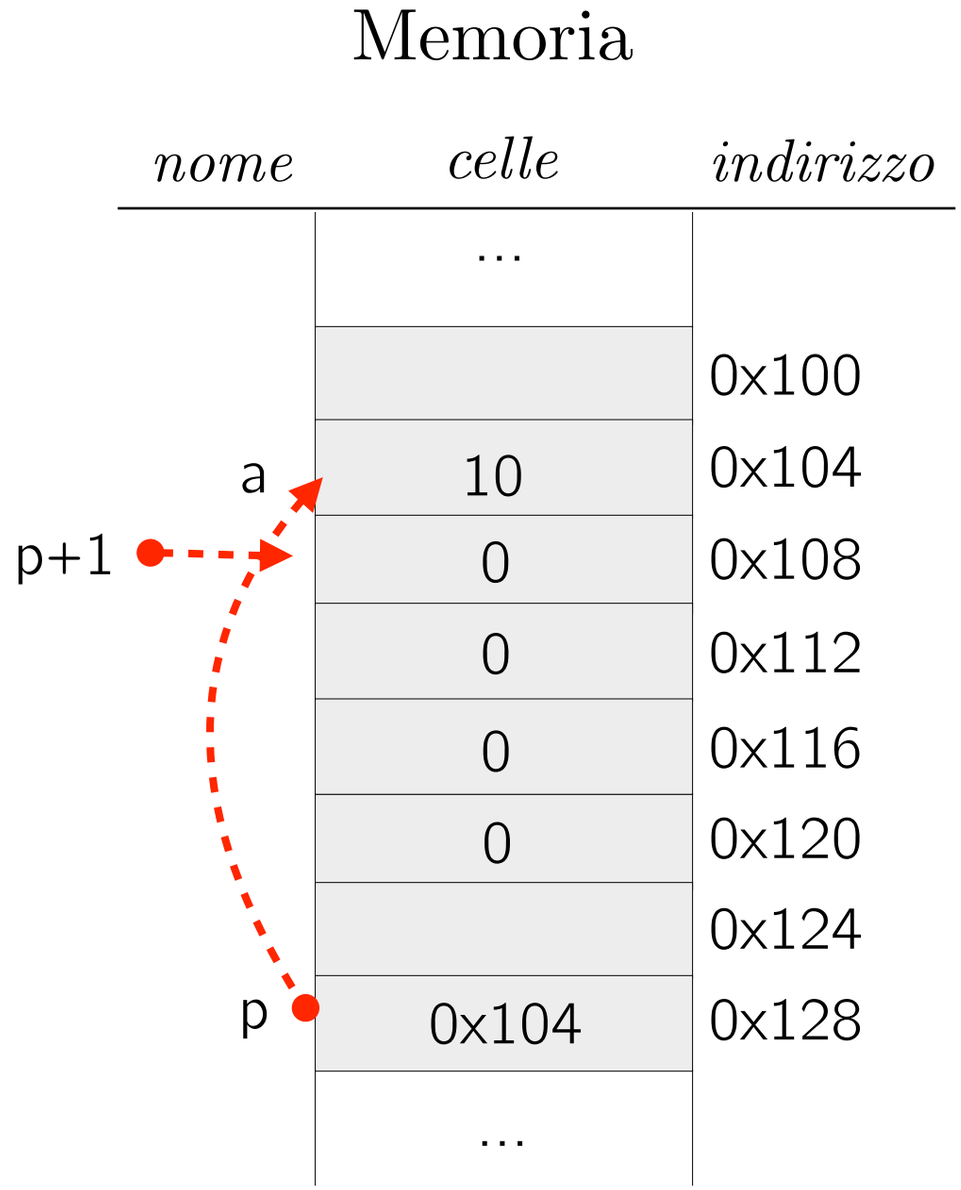
Ora p punta una cella in avanti



Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
    a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;      oppure int *p = &a[0];

*p = 10;

p[0] = 10;    3 forme equivalenti!

a[0] = 10;

p+1;
```

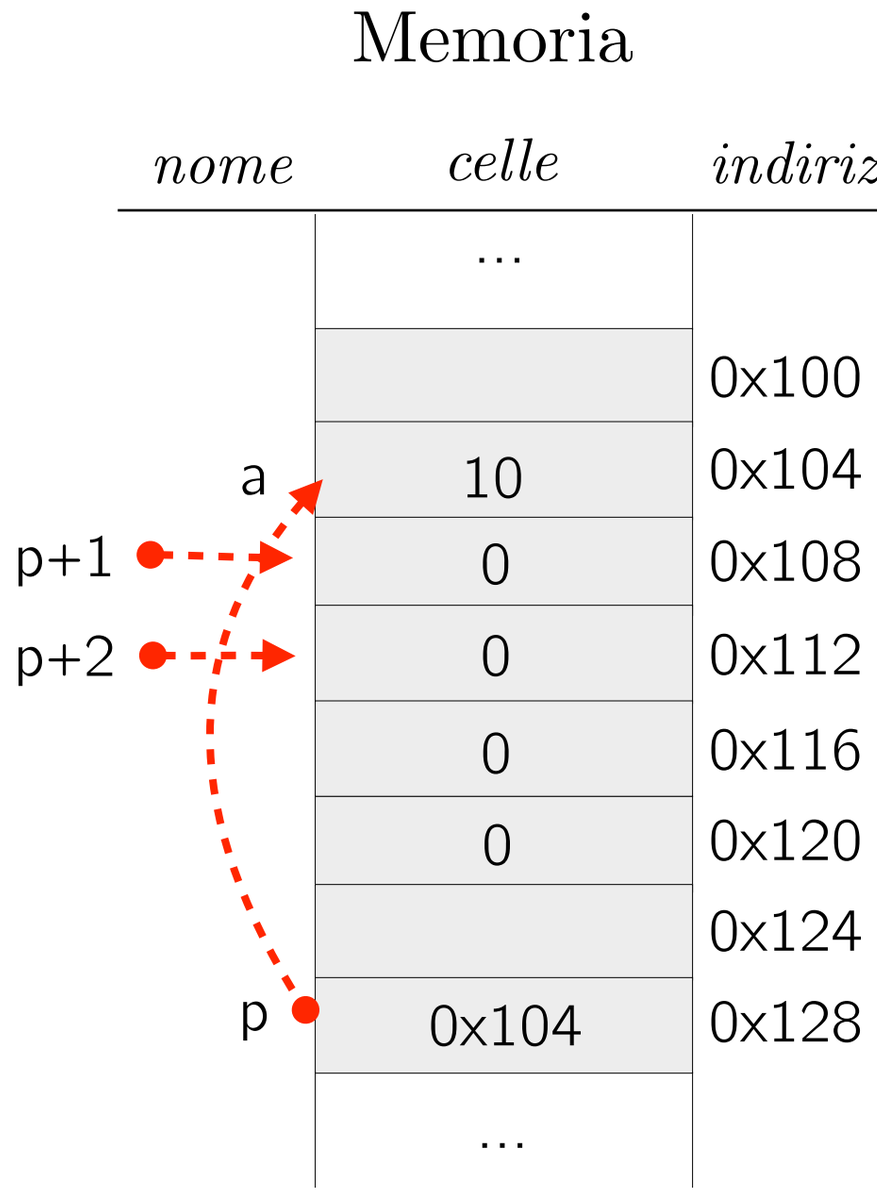
Ora p punta una cella in avanti



Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
    a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;      oppure int *p = &a[0];

*p = 10;

p[0] = 10;    3 forme equivalenti!

a[0] = 10;

p+1;
```

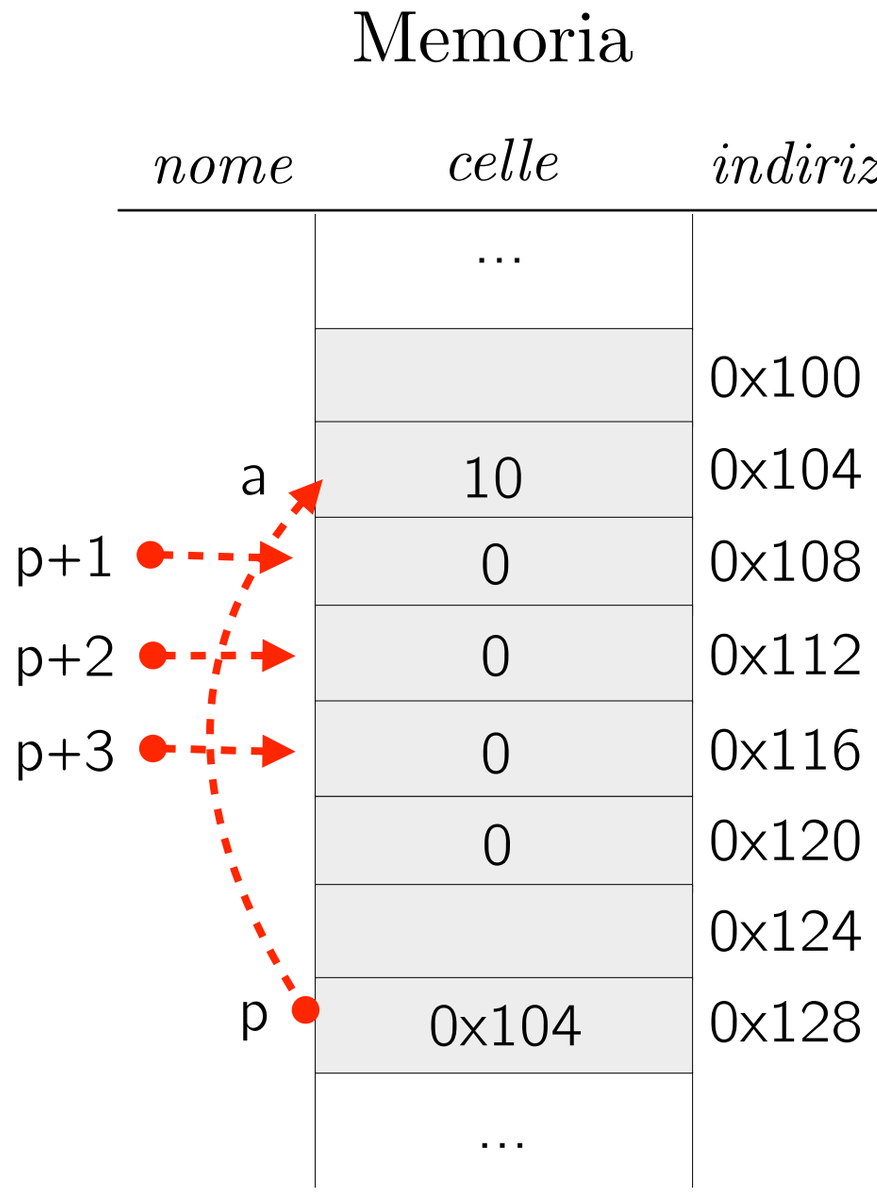
Ora p punta una cella in avanti



Array e puntatori

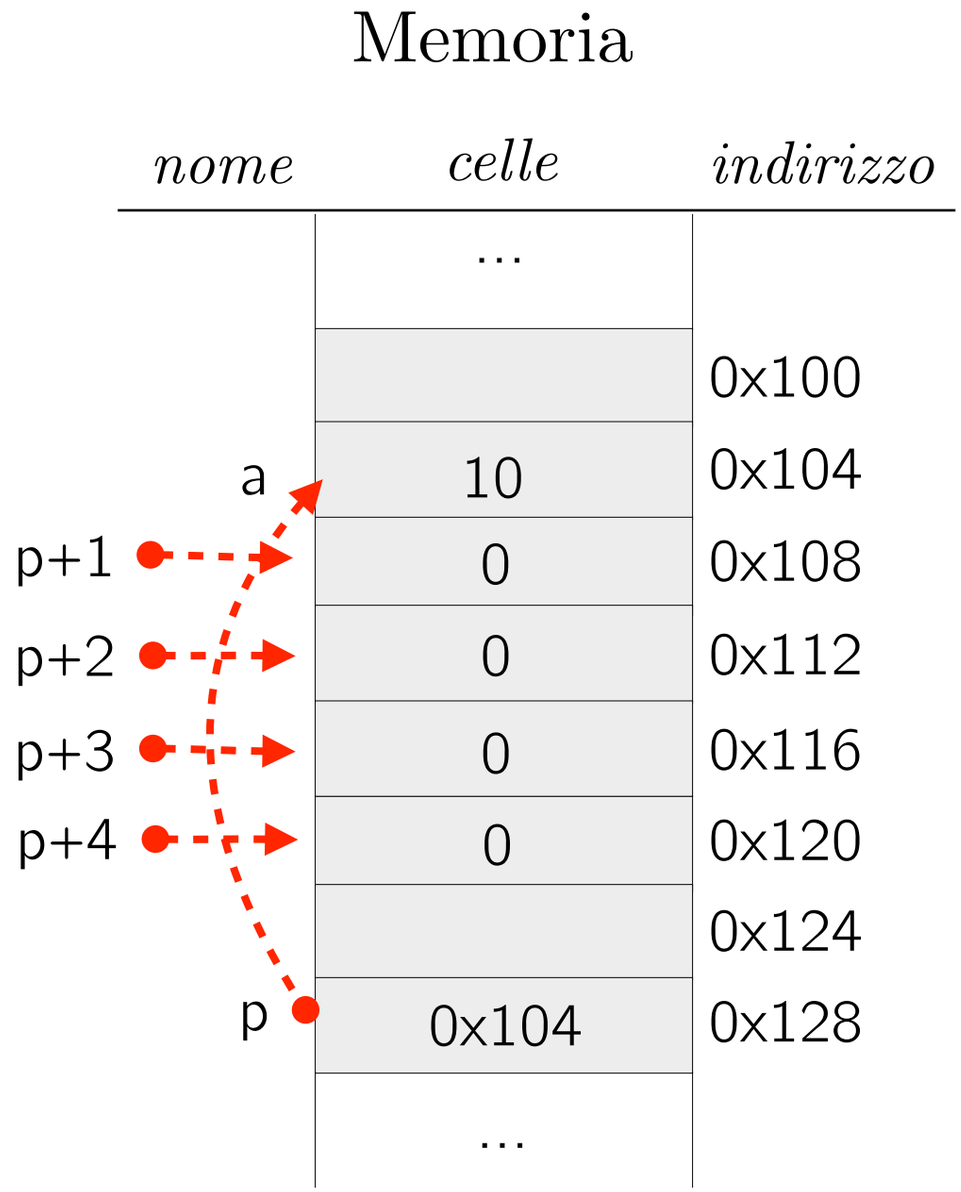
```
int a[5];

for ( i = 0; i < 5; i++)
    a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;      oppure int *p = &a[0];

*p = 10;

p[0] = 10;    3 forme equivalenti!

a[0] = 10;

p+1;
```

Ora p punta una cella in avanti



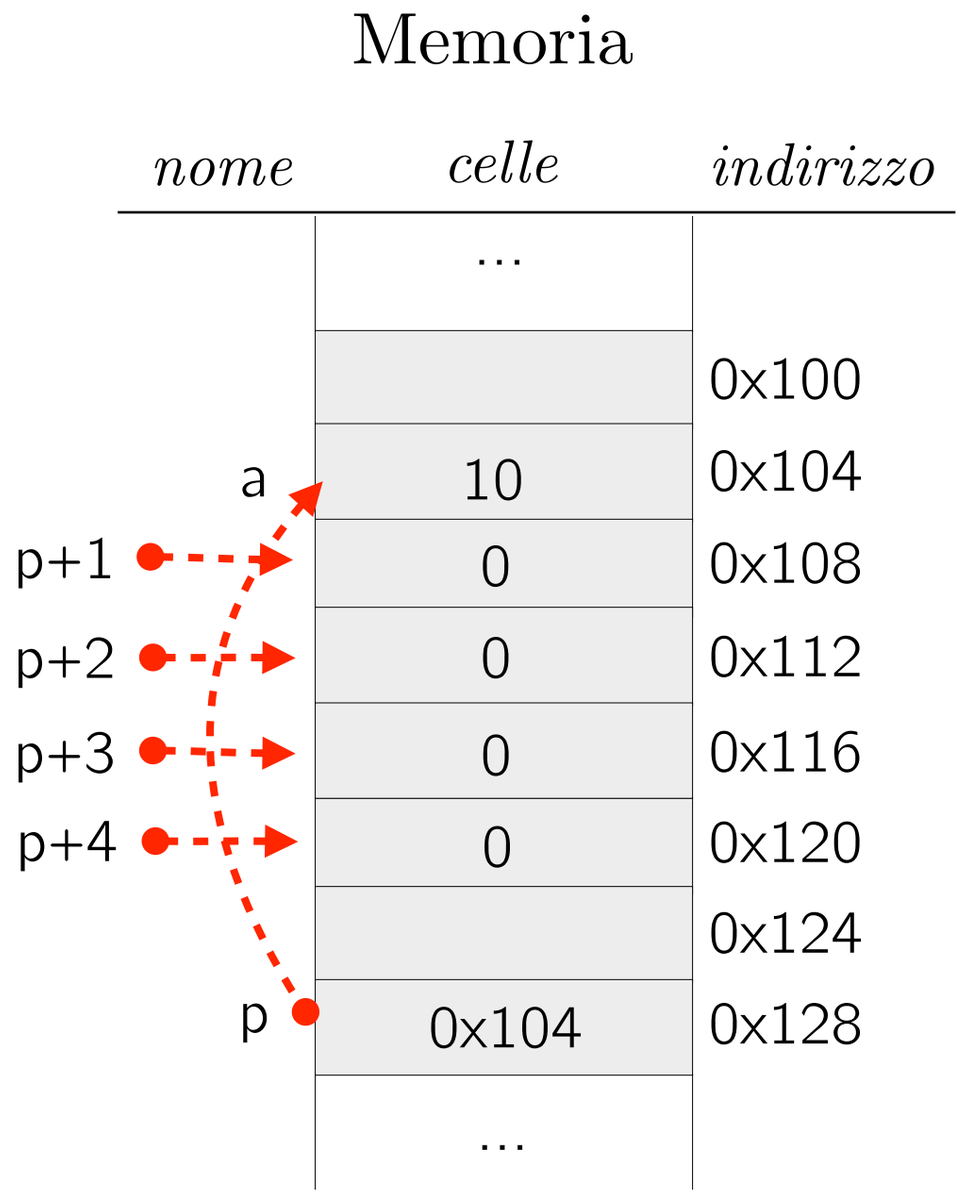
Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
    a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;      oppure int *p = &a[0];

*p = 10;

p[0] = 10;    3 forme equivalenti!

a[0] = 10;

p+1;
```



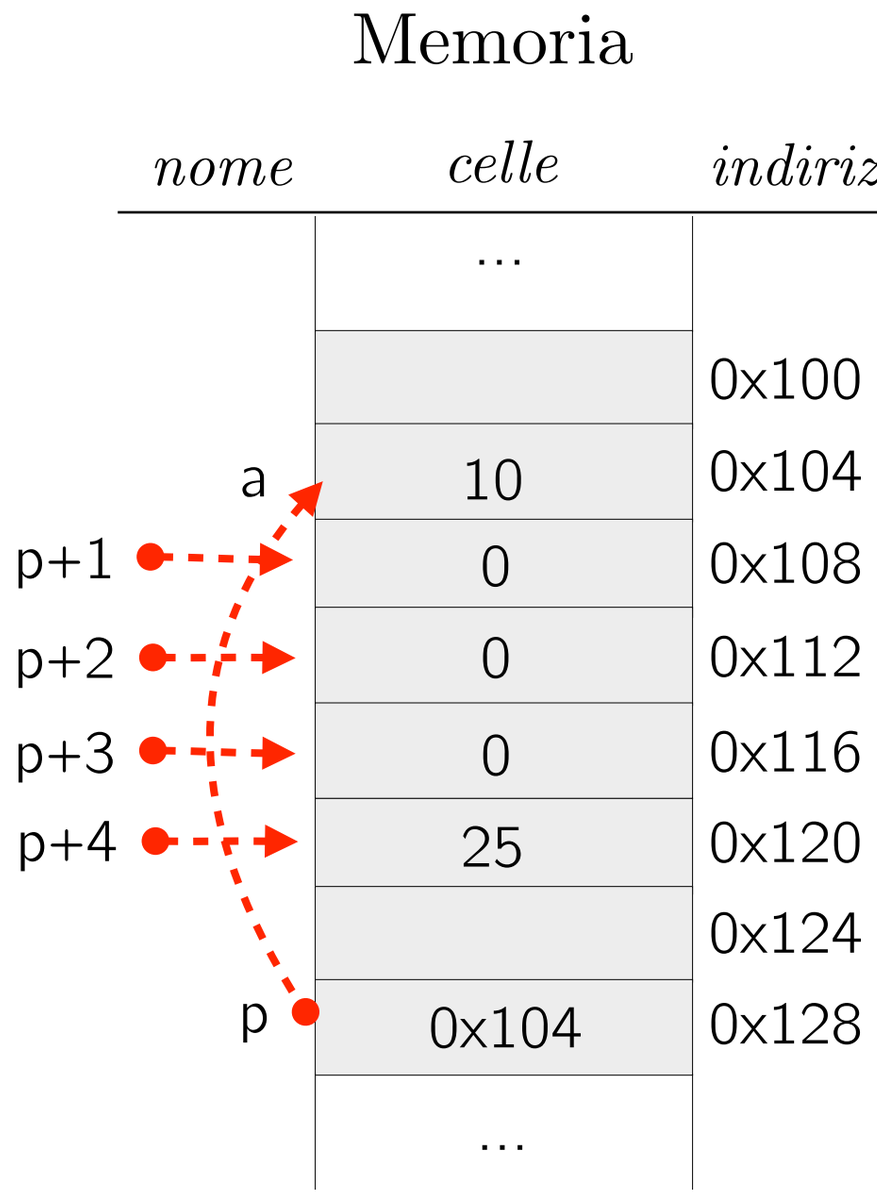
Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
     a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;      oppure int *p = &a[0];

*p = 10;

p[0] = 10;    3 forme equivalenti!

a[0] = 10;

p+1;

*(p+4)= 25;
```



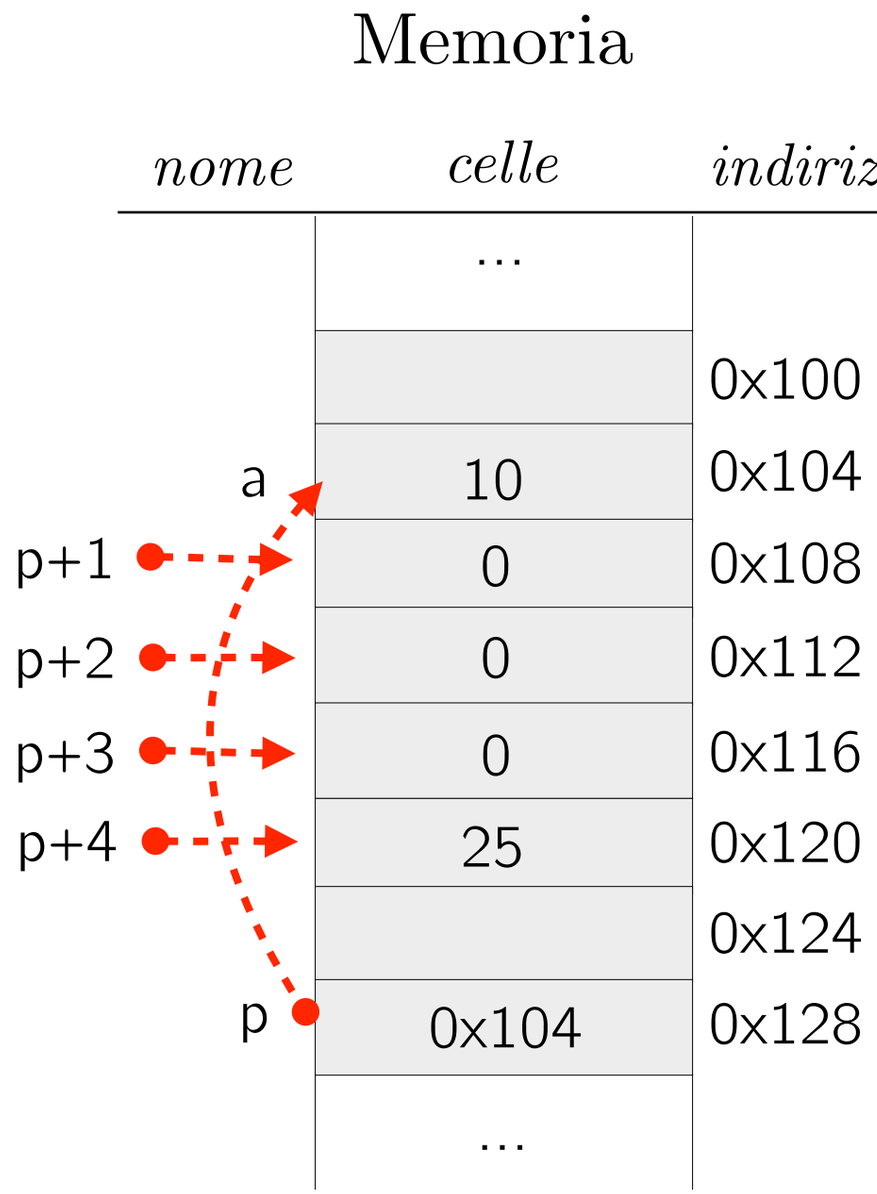
Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
     a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;      oppure int *p = &a[0];

*p = 10;

p[0] = 10;    3 forme equivalenti!

a[0] = 10;

p+1;

*(p+4)= 25;
```



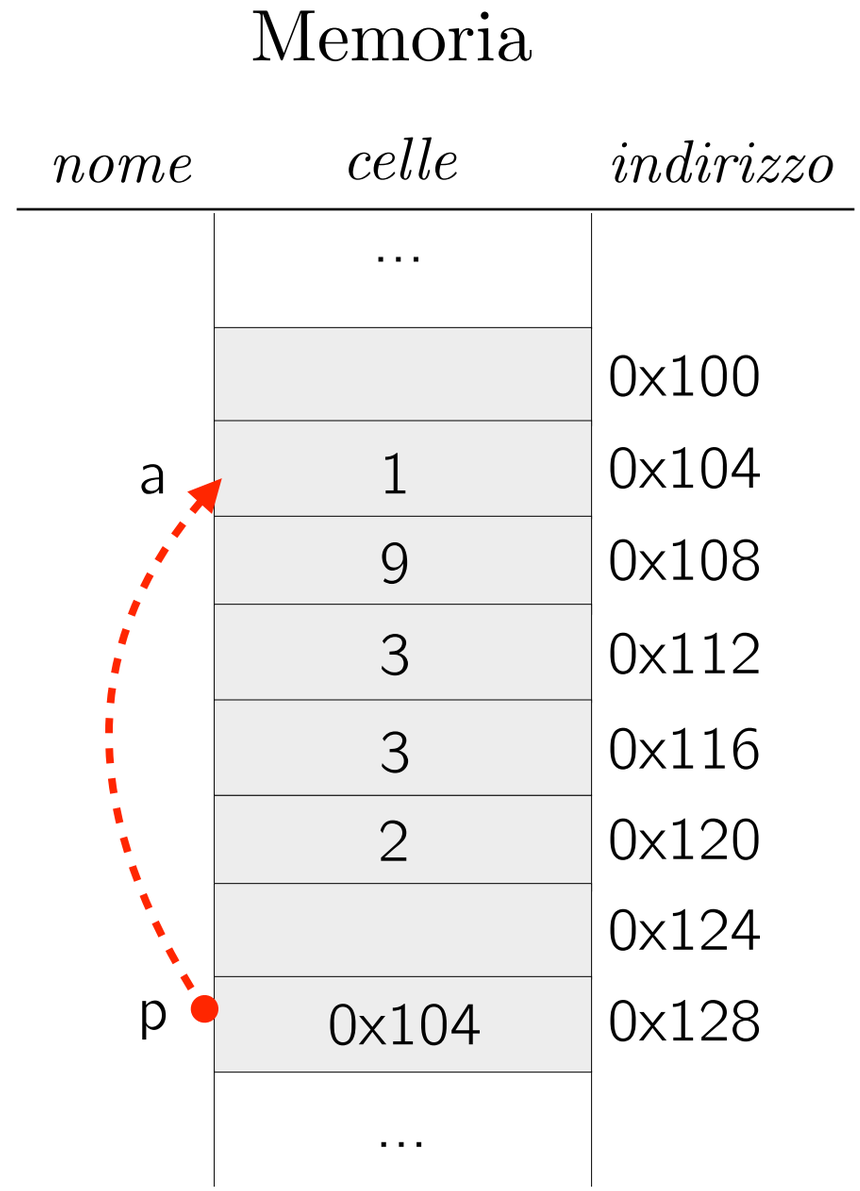
Array e puntatori

```
int a[5];

for ( i = 0; i < 5; i++)
     a[i] = 0;

a = &x;   NO! a non può essere modificato.

int *p = a;      oppure int *p = &a[0];

*p = 10;

p[0] = 10;    3 forme equivalenti!

a[0] = 10;

p+1;

*(p+4)= 25;

p[4] = 25;    3 forme equivalenti!

a[4] = 25;
```



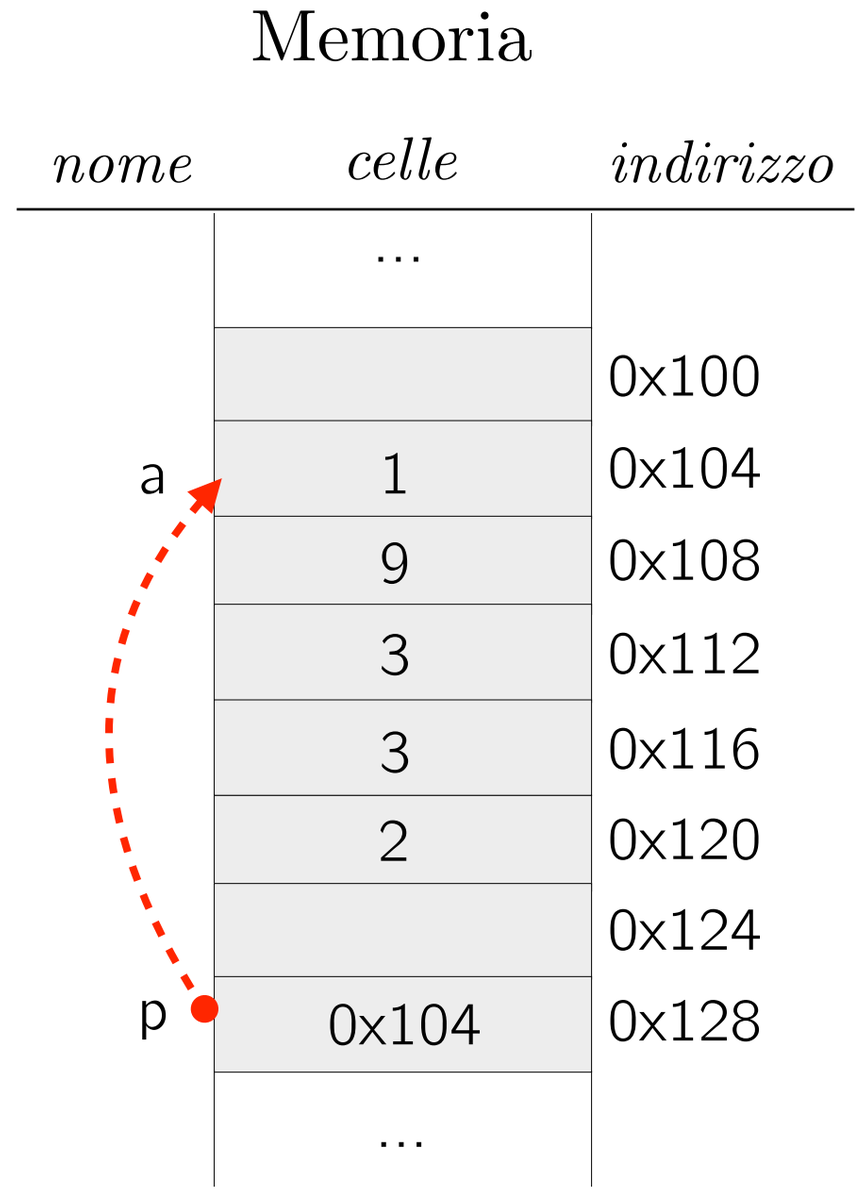
Cinque frammenti equivalenti

```
int a[5] = { 1, 9, 3, 3, 2 };
int i, sum = 0;
int *p = a;
```



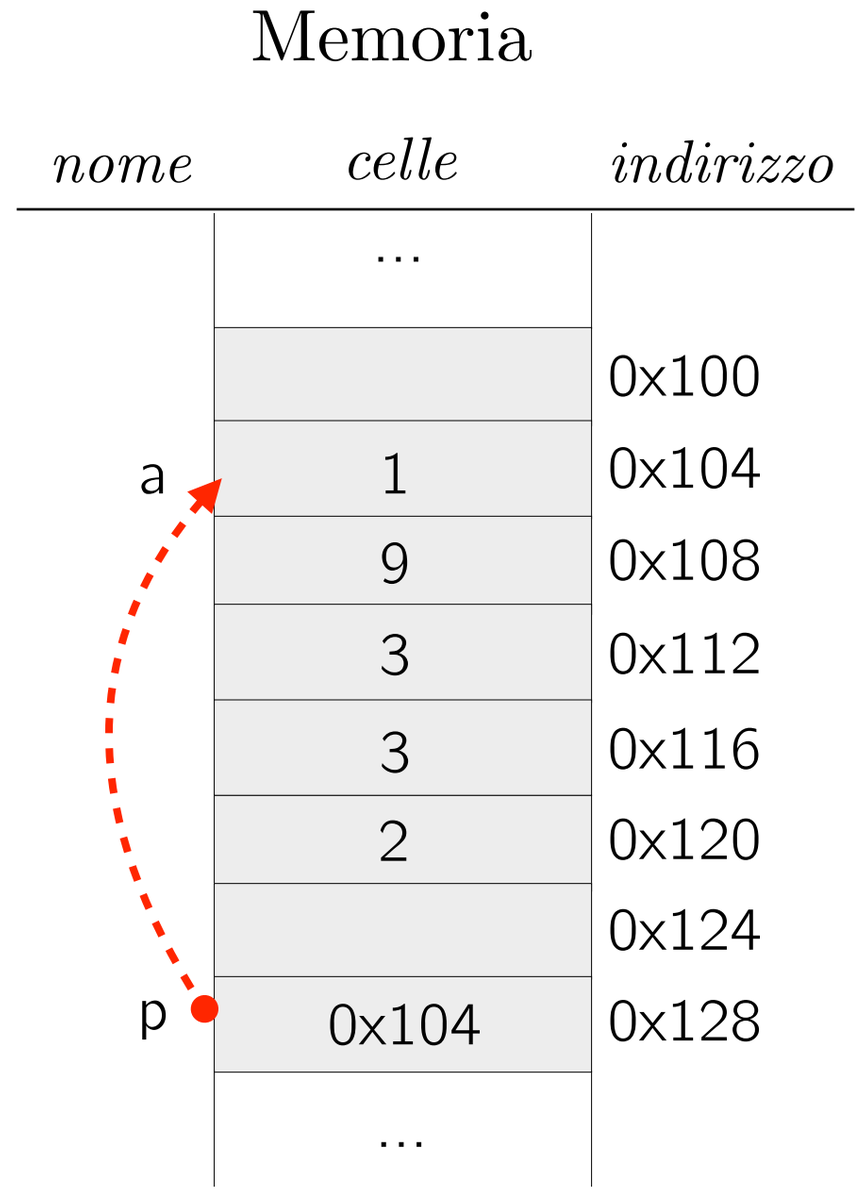
Cinque frammenti equivalenti

```
int a[5] = { 1, 9, 3, 3, 2 };
int i, sum = 0;
int *p = a;

for ( i = 0; i < 5; i++)
    sum += a[i];
```



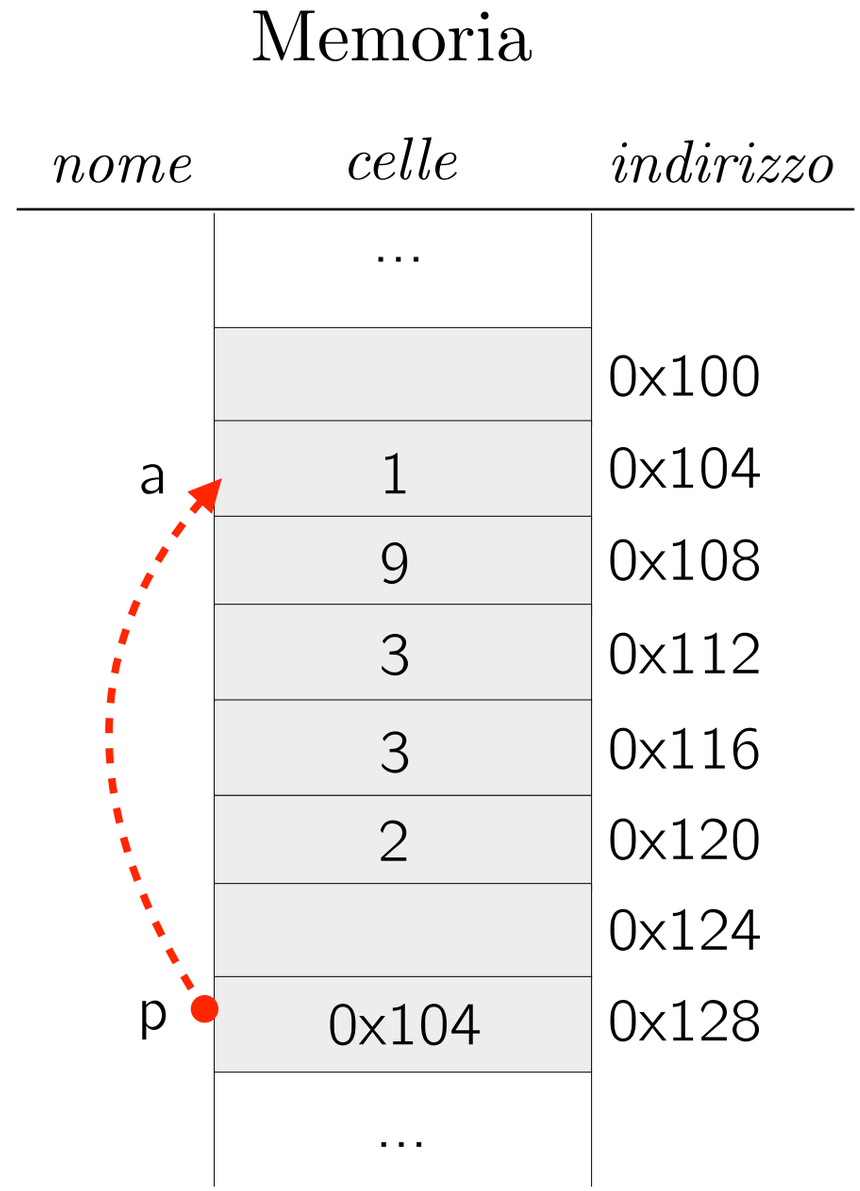
Cinque frammenti equivalenti

```
int a[5] = { 1, 9, 3, 3, 2 };
int i, sum = 0;
int *p = a;

for ( i = 0; i < 5; i++)
    sum += a[i];

for ( i = 0; i < 5; i++)
    sum += *(a+i);
```



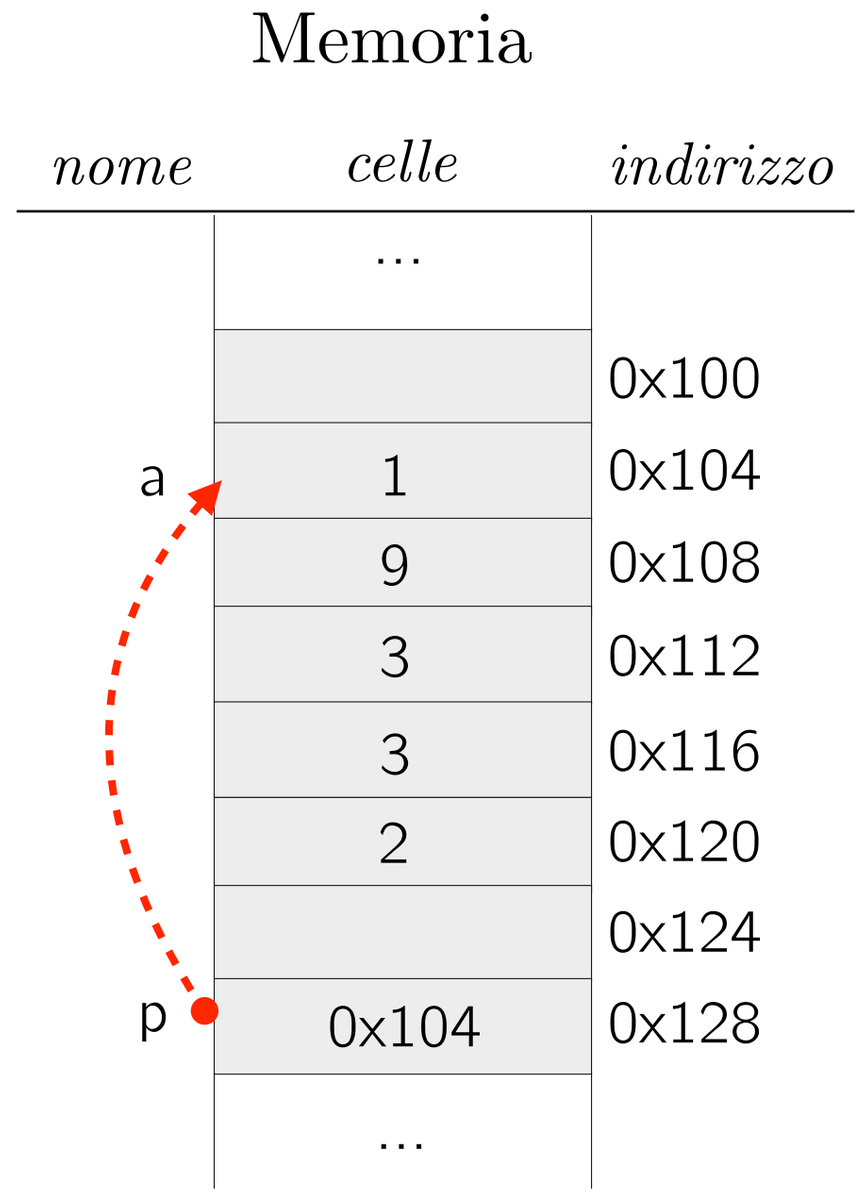
Cinque frammenti equivalenti

```
int a[5] = { 1, 9, 3, 3, 2 };
int i, sum = 0;
int *p = a;

for ( i = 0; i < 5; i++)
    sum += a[i];

for ( i = 0; i < 5; i++)
    sum += *(a+i);

for ( i = 0; i < 5; i++)
    sum += p[i];
```



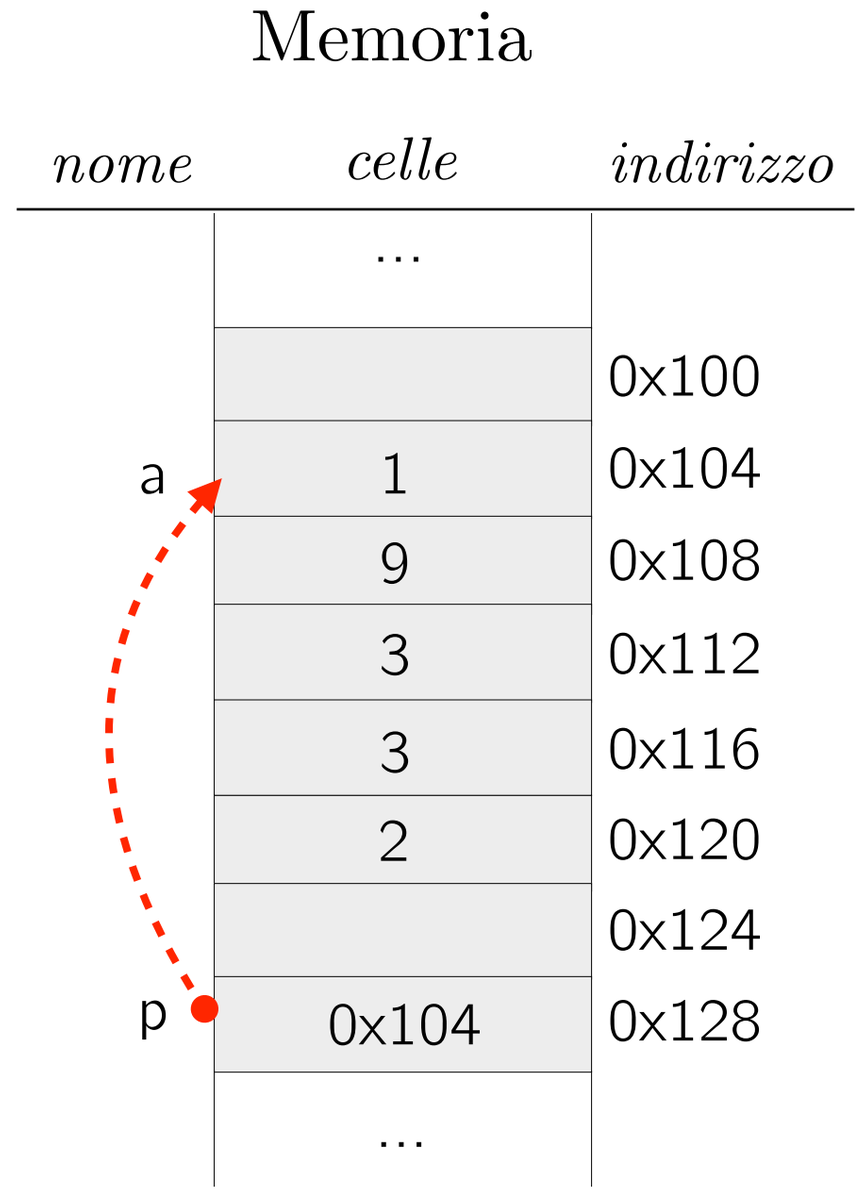
Cinque frammenti equivalenti

```
int a[5] = { 1, 9, 3, 3, 2 };
int i, sum = 0;
int *p = a;

for ( i = 0; i < 5; i++)
    sum += a[i];

for ( i = 0; i < 5; i++)
    sum += *(a+i);

for ( i = 0; i < 5; i++)
    sum += p[i];

for ( i = 0; i < 5; i++)
    sum += *(p+i);
```



Cinque frammenti equivalenti

```
int a[5] = { 1, 9, 3, 3, 2 };
int i, sum = 0;
int *p = a;

for ( i = 0; i < 5; i++)
    sum += a[i];

for ( i = 0; i < 5; i++)
    sum += *(a+i);

for ( i = 0; i < 5; i++)
    sum += p[i];


for ( i = 0; i < 5; i++)
    sum += *(p+i);


for ( p = a; p < a + 5; p++)
    sum += *p;
```



Cinque frammenti equivalenti

```
int a[5] = { 1, 9, 3, 3, 2 };
int i, sum = 0;
int *p = a;

for ( i = 0; i < 5; i++)
    sum += a[i];

for ( i = 0; i < 5; i++)
    sum += *(a+i);

for ( i = 0; i < 5; i++)
    sum += p[i];

for ( i = 0; i < 5; i++)
    sum += *(p+i);

for ( p = a; p < a + 5; p++)
    sum += *p;
```

Provateli nel vostro codice!

Memoria

nome	*celle*	*indirizzo*
	...	
		0x100
a	1	0x104
	9	0x108
	3	0x112
	3	0x116
	2	0x120
		0x124
p	0x104	0x128
	...	

Passaggio di array a funzioni

Gli array sono sempre passati per riferimento.

Passaggio di array a funzioni

Gli array sono sempre passati per riferimento.

Ciò che viene passato (e copiato) è il puntare al primo elemento.

Passaggio di array a funzioni

Gli array sono sempre passati per riferimento.

Ciò che viene passato (e copiato) è il puntare al primo elemento.

Esempio

```
int inizializza(int a[], int len) {
    int i;
    for( i = 0; i < len; i++ )
        a[i] = 0;
}

int main() {
    int a[5];
    inizializza(a, 5);
    /* da qui tutti gli elementi di a sono a 0 */
    ...
```

Passaggio di array a funzioni

Gli array sono sempre passati per riferimento.

Ciò che viene passato (e copiato) è il puntare al primo elemento.

Esempio

```
int inizializza(int a[], int len) {
    int i;
    for( i = 0; i < len; i++ )
        a[i] = 0;
}

int main() {
    int a[5];
    inizializza(a, 5);
    /* da qui tutti gli elementi di a sono a 0 */
    ...
```

Passare sempre anche la lunghezza.

Passaggio di array a funzioni

Gli array sono sempre passati per riferimento.

Ciò che viene passato (e copiato) è il puntare al primo elemento.

Altro Esempio

```
int inizializza(int a[], int len) {
    int i;
    for( i = 0; i < len; i++ )
        a[i] = 0;
}

int main() {
    int a[5];
    inizializza(a+1, 4);
    /* da qui tutti gli elementi di a (escluso il primo) sono a 0 */
    ...
```

Passaggio di array a funzioni

Gli array sono sempre passati per riferimento.

Ciò che viene passato (e copiato) è il puntare al primo elemento.

Altro Esempio

```
int inizializza(int a[], int len) {
    int i;
    for( i = 0; i < len; i++ )
        a[i] = 0;
}

int main() {
    int a[5];
    inizializza(a+1, 4);
    /* da qui tutti gli elementi di a (escluso il primo) sono a 0 */
    ...
```

Sottoarray che inizia dalla seconda posizione di a.

Stringhe (1)

Stringhe (1)

Una stringa è una sequenza di caratteri, ad esempio una parola o un testo.

Stringhe (1)

Una stringa è una sequenza di caratteri, ad esempio una parola o un testo.

In C non è previsto un tipo per le stringhe.

Stringhe (1)

Una stringa è una sequenza di caratteri, ad esempio una parola o un testo.

In C non è previsto un tipo per le stringhe.

Una stringa è vista come un array di caratteri che, per convenzione, termina con il carattere speciale ‘\0’.

Stringhe (1)

Una stringa è una sequenza di caratteri, ad esempio una parola o un testo.

In C non è previsto un tipo per le stringhe.

Una stringa è vista come un array di caratteri che, per convenzione, termina con il carattere speciale '\0'.

Quindi si usa

```
char s[N+1];
```

per memorizzare una stringa di N caratteri.

Stringhe (2)

Stringhe (2)

Le costanti stringa sono specificate tra virgolette.

Stringhe (2)

Le costanti stringa sono specificate tra virgolette.

Ad esempio, “ciao” è un array di 5 caratteri.

Stringhe (2)

Le costanti stringa sono specificate tra virgolette.

Ad esempio, “ciao” è un array di 5 caratteri.

Una costante stringa viene trattata come il puntatore al suo primo carattere.

Stringhe (2)

Le costanti stringa sono specificate tra virgolette.

Ad esempio, “ciao” è un array di 5 caratteri.

Una costante stringa viene trattata come il puntatore al suo primo carattere.

Esempio

```
int main () {
    char *s = “ciao”;
```

Stringhe (2)

Le costanti stringa sono specificate tra virgolette.

Ad esempio, “ciao” è un array di 5 caratteri.

Una costante stringa viene trattata come il puntatore al suo primo carattere.

Esempio

```
int main () {
    char *s = “ciao”;
    printf(“%s\n”, s);
```

Stringhe (2)

Le costanti stringa sono specificate tra virgolette.

Ad esempio, “ciao” è un array di 5 caratteri.

Una costante stringa viene trattata come il puntatore al suo primo carattere.

Esempio

```
int main () {
    char *s = “ciao”;
    printf(“%s\n”, s);
```

```
ciao
```

Stringhe (2)

Le costanti stringa sono specificate tra virgolette.

Ad esempio, “ciao” è un array di 5 caratteri.

Una costante stringa viene trattata come il puntatore al suo primo carattere.

Esempio

```
int main () {
    char *s = “ciao”;
    printf(“%s\n”, s);
    printf(“%s\n”, s+1);
```

```
ciao
```

Stringhe (2)

Le costanti stringa sono specificate tra virgolette.

Ad esempio, “ciao” è un array di 5 caratteri.

Una costante stringa viene trattata come il puntatore al suo primo carattere.

Esempio

```
int main () {
    char *s = “ciao”;
    printf(“%s\n”, s);
    printf(“%s\n”, s+1);
```

```
ciao
iac
```

Stringhe (2)

Le costanti stringa sono specificate tra virgolette.

Ad esempio, “ciao” è un array di 5 caratteri.

Una costante stringa viene trattata come il puntatore al suo primo carattere.

Esempio

```
int main () {
    char *s = “ciao”;
    printf(“%s\n”, s);
    printf(“%s\n”, s+1);
    printf(“%c\n”, *s);
```

```
ciao
iac
```

Stringhe (2)

Le costanti stringa sono specificate tra virgolette.

Ad esempio, “ciao” è un array di 5 caratteri.

Una costante stringa viene trattata come il puntatore al suo primo carattere.

Esempio

```
int main () {
    char *s = “ciao”;
    printf(“%s\n”, s);
    printf(“%s\n”, s+1);
    printf(“%c\n”, *s);
```

```
ciao
iac
c
```

Stringhe (2)

Le costanti stringa sono specificate tra virgolette.

Ad esempio, “ciao” è un array di 5 caratteri.

Una costante stringa viene trattata come il puntatore al suo primo carattere.

Esempio

```
int main () {
    char *s = “ciao”;
    printf(“%s\n”, s);
    printf(“%s\n”, s+1);
    printf(“%c\n”, *s);
    printf(“%c\n”, *(s+1));
    return 0;
}
```

```
ciao
iac
c
```

Stringhe (2)

Le costanti stringa sono specificate tra virgolette.

Ad esempio, “ciao” è un array di 5 caratteri.

Una costante stringa viene trattata come il puntatore al suo primo carattere.

Esempio

```
int main () {
    char *s = “ciao”;
    printf(“%s\n”, s);
    printf(“%s\n”, s+1);
    printf(“%c\n”, *s);
    printf(“%c\n”, *(s+1));
    return 0;
}
```

```
ciao
iac
c
i
```

Stringhe (2)

Le costanti stringa sono specificate tra virgolette.

Ad esempio, “ciao” è un array di 5 caratteri.

Una costante stringa viene trattata come il puntatore al suo primo carattere.

Esempio

```
int main () {
    char *s = “ciao”;
    printf(“%s\n”, s);
    printf(“%s\n”, s+1);
    printf(“%c\n”, *s);
    printf(“%c\n”, *(s+1));
    return 0;
}
```

```
ciao
iac
c
i
```

La libreria string.h contiene utili funzioni per gestire le stringhe

Stringhe (3)

Esempio

```
#include <stdio.h>


void my_printf(char *s) {
    int i = 0;
    while(s[i]) { // s[i] != '\0'
        printf("%c", s[i++]);
    }
}

int main () {
    char s[101]; // stringhe fino a 100 caratteri
    scanf("%s", s);
    my_printf(s);
    return 0;
}
```

Stringhe (3)

Esempio

```
#include <stdio.h>


void my_printf(char *s) {
    int i = 0;
    while(s[i]) { // s[i] != '\0'
        printf("%c", s[i++]);
    }
}

int main () {
    char s[101]; // stringhe fino a 100 caratteri
    scanf("%s", s);
    my_printf(s);
    return 0;
}
```

Stringhe (3)

Esempio

```
#include <stdio.h>

void my_printf(char *s) {
    int i = 0;
    while(s[i]) { // s[i] != '\0'
        printf("%c", s[i++]);
    }
}

int main () {
    char s[101]; // stringhe fino a 100 caratteri
    scanf("%s", s);
    my_printf(s);
    return 0;
}
```

Senza & perché?

Stringhe (3)

Esempio

```
#include <stdio.h>


void my_printf(char *s) {
    int i = 0;
    while(s[i]){ // s[i] != '\0'
        printf("%c", s[i++]);
    }
}

int main () {
    char s[101]; // stringhe fino a 100 caratteri
    scanf("%s", s);
    my_printf(s);
    return 0;
}
```

Stringhe (3)

Esempio

```
#include <stdio.h>


void my_printf(char *s) {
    int i = 0;
    while(s[i]) { // s[i] != '\0'
        printf("%c", s[i++]);
    }
}

int main () {
    char s[101]; // stringhe fino a 100 caratteri
    scanf("%s", s);
    my_printf(s);
    return 0;
}
```

Stringhe (4)

Esempio: versione alternativa

```
#include <stdio.h>

void my_printf(char *s) {
    while(*s) {
        printf("%c", *s++);  // è s ad essere incrementato
    }
}

int main () {
    char s[101]; // stringhe fino a 100 caratteri
    scanf("%s", s);
    my_printf(s);
    return 0;
}
```

Stringhe (4)

Esempio: versione alternativa

```
#include <stdio.h>


void my_printf(char *s) {
    while(*s) {
        printf("%c", *s++); // è s ad essere incrementato
    }
}


int main () {
    char s[101]; // stringhe fino a 100 caratteri
    scanf("%s", s);
    my_printf(s);
    return 0;
}
```

Stringhe (4)

Esempio: versione alternativa

```
#include <stdio.h>


void my_printf(char *s) {
    while(*s) {
        printf("%c", *s++); // è s ad essere incrementato
    }
}


int main () {
    char s[101]; // stringhe fino a 100 caratteri
    scanf("%s", s);
    my_printf(s);
    return 0;
}
```

s dove punta ora?

Stringhe (4)

Esempio: versione alternativa

```
#include <stdio.h>


void my_printf(char *s) {
    while(*s) {
        printf("%c", *s++); // è s ad essere incrementato
    }
}


int main () {
    char s[101]; // stringhe fino a 100 caratteri
    scanf("%s", s);
    my_printf(s);
    return 0;
}
```

s dove punta ora?

Ancora all'inizio della stringa.
my_printf modifica una copia di s!

Valgrind

Valgrind

Strumento **molto** utile per dare la "caccia" ai bug, specialmente per problemi legati alla gestione della memoria. Installabile su qualunque distribuzione Linux o Mac OS X.

Valgrind

Strumento **molto** utile per dare la "caccia" ai bug, specialmente per problemi legati alla gestione della memoria. Installabile su qualunque distribuzione Linux o Mac OS X.

```
int main () {
    int x;
    if ( x >= 0 ) {
        printf("positivo");
    } else {
        printf("negativo");
    }
}
```

Valgrind

Strumento **molto** utile per dare la "caccia" ai bug, specialmente per problemi legati alla gestione della memoria. Installabile su qualunque distribuzione Linux o Mac OS X.

```
int main () {
    int x;
    if ( x >= 0 ) {
        printf("positivo");
    } else {
        printf("negativo");
    }
}
```

```
$ gcc -g -o prog prog.c
```

Valgrind

Strumento **molto** utile per dare la "caccia" ai bug, specialmente per problemi legati alla gestione della memoria. Installabile su qualunque distribuzione Linux o Mac OS X.

```
int main () {
    int x;
    if ( x >= 0 ) {
        printf("positivo");
    } else {
        printf("negativo");
    }
}
```

```
$ gcc -g -o prog prog.c
$ valgrind ./prog
```

Valgrind

Strumento **molto** utile per dare la "caccia" ai bug, specialmente per problemi legati alla gestione della memoria. Installabile su qualunque distribuzione Linux o Mac OS X.

```
int main () {
    int x;
    if ( x >= 0 ) {
        printf("positivo");
    } else {
        printf("negativo");
    }
}
```

```
$ gcc -g -o prog prog.c
$ valgrind ./prog
...
==1426==
==1426== Conditional jump or move depends on uninitialised value(s)
==1426==    at 0x100000F36: main (prog.c:4)
```

Valgrind

Strumento **molto** utile per dare la "caccia" ai bug, specialmente per problemi legati alla gestione della memoria. Installabile su qualunque distribuzione Linux o Mac OS X.

```
int main () {
    int x;
    if ( x >= 0 ) {
        printf("positivo");
    } else {
        printf("negativo");
    }
}
```

```
$ gcc -g -o prog prog.c
$ valgrind ./prog
...
==1426==
==1426== Conditional jump or move depends on uninitialised value(s)
==1426==    at 0x100000F36: main (prog.c:4)
```

Valgrind

Strumento **molto** utile per dare la "caccia" ai bug, specialmente per problemi legati alla gestione della memoria. Installabile su qualunque distribuzione Linux o Mac OS X.

```
int main () {
      int a[10], i;
      for( i = 0; i < 100; i++ )
            a[i] = 0;
}
```

```
$ gcc -g -o prog prog.c
$ valgrind ./prog
```

Valgrind

Strumento **molto** utile per dare la "caccia" ai bug, specialmente per problemi legati alla gestione della memoria. Installabile su qualunque distribuzione Linux o Mac OS X.

```
int main () {
      int a[10], i;
      for( i = 0; i < 100; i++ )
            a[i] = 0;
}
```

```
$ gcc -g -o prog prog.c
$ valgrind ./prog
```

Valgrind

Strumento **molto** utile per dare la "caccia" ai bug, specialmente per problemi legati alla gestione della memoria. Installabile su qualunque distribuzione Linux o Mac OS X.

```
int main () {
      int a[10], i;
      for( i = 0; i < 100; i++ )
            a[i] = 0;
}
```

```
$ gcc -g -o prog prog.c
$ valgrind ./prog
...
==1487== Invalid write of size 4
==1487==    at 0x100000F35: main (prog.c:5)
==1487==  Address 0x104803000 is not stack'd, malloc'd or
(recently) free'd
```

Valgrind

Strumento **molto** utile per dare la "caccia" ai bug, specialmente per problemi legati alla gestione della memoria. Installabile su qualunque distribuzione Linux o Mac OS X.

```
int main () {
    int a[10], i;
    for( i = 0; i < 100; i++ )
        printf("%d", a[i]);
}
```

```
$ gcc -g -o prog prog.c
$ valgrind ./prog
...
==1487== Invalid write of size 4
==1487==    at 0x100000F35: main (prog.c:5)
==1487==  Address 0x104803000 is not stack'd, malloc'd or
(recently) free'd
```

Valgrind

Strumento **molto** utile per dare la "caccia" ai bug, specialmente per problemi legati alla gestione della memoria. Installabile su qualunque distribuzione Linux o Mac OS X.

```
int main () {
    int a[10], i;
    for( i = 0; i < 100; i++ )
        printf("%d", a[i]);
}
```

```
$ gcc -g -o prog prog.c
$ valgrind ./prog
...
==1519== Invalid read of size 4
==1519==    at 0x100000F1C: main (prog.c:5)
==1519==  Address 0x104803000 is not stack'd, malloc'd or
(recently) free'd
```

Esercizio 1

FindVal

Esercizio

Scrivere una funzione

```
int* FindVal(int a[], int len, int val)
```

che, dato un array *a* e la sua lunghezza *len*, cerchi il valore *val* all'interno di *a* e restituisca un puntatore alla cella che lo contiene, o la costante predefinita `NULL` se *val* non è contenuto in *a*.

Scrivere poi un programma che legga da input un array di 10 interi e un intero val e stampi `trovato` se l'intero val si trova nell'array, `non trovato` altrimenti.

L'input è formato da dieci righe contenenti gli elementi dell'array, seguite dall'intero val da cercare.
L'unica riga dell'output contiene la stringa

```
trovato
```

se l'intero val si trova nell'array,

```
non trovato
```

altrimenti.

Esercizio 2

Contatore

Esercizio

Scrivere una programma che data una sequenza di interi tenga traccia delle frequenze degli interi compresi tra 0 e 9 (estremi inclusi). La sequenza termina quando viene letto il valore -1. Il programma deve stampare in output le frequenze dei valori compresi tra 0 e 9.
Le frequenze saranno mantenute in un array di contatori di lunghezza 10 che sarà inizializzato a 0.

Implementare queste due funzioni:

- `void reset(int array[], int len)`: inizializza l'array dei contatori a 0;
- `void add(int array[], int len, int val)`: incrementa il contatore `array[val]` se `val` è tra 0 e `len`-1.

L'input è formato da una sequenza di interi terminata dall'intero -1.
L'output è costituito dalle frequenze (una per riga) degli interi tra 0 e 10 nella sequenza letta in input.

Esercizio 3

My strlen

Esercizio

Scrivere una funzione

```
int my_strlen(char *s)
```

che restituisce il numero di caratteri della stringa *s*.

Scrivere un programma che provi questa funzione leggendo una stringa da tastiera. Si può assumere che la stringa in input contenga non più di 1000 caratteri.

L'input è costituito da una stringa di lunghezza non maggiore di 1000 caratteri.
L'unica riga dell'output contiene la lunghezza della stringa.

Esempio

Input	Output
`ciao!`	`5`

Esercizio 4 Anagramma

Esercizio

Scrivere la funzione

`int anagramma(unsigned char *s1, unsigned char *s2)`

che restituisca 1 se le stringhe puntate da $s1$ e $s2$ sono una l'anagramma dell'altro e 0 altrimenti.
Esempio: `anagramma(''pizza'', ''pazzi'') == 1`

Strivere quindi un programma che legga da input due stringhe $s1$ e $s2$ e utilizzi questa funzione per stabilire se una è l'anagramma dell'altra. Nota: utilizzare il tipo `unsigned char *` per le stringhe.

Hint. Data una stringa S, costruire un array $aS[256]$ tale che $aS[i]$ memorizzi il numero di occorrenze del carattere i in S. Come sono gli array aS e aZ di due stringhe S e Z che sono una l'anagramma dell'altra?

L'input è formato da due stringhe $s1$ e $s2$.
L'output è 1 se $s1$ è l'anagramma di $s2$, 0 altrimenti.

Esempi

Input	Output
`aeiou` `uoaei`	`1`