Array

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Array

- Un array è un gruppo di variabili dello stesso tipo che hanno locazioni di memoria contigue.
- Dichiarazione: tipo nome[dimensione];
- Es. int c[12]; // dichiara un array (una sequenza) di 12 variabili intere
- c[i] si comporta come una variabile di tipo intero
- c[i] accede all'i-esimo elemento dell'array (si usa la parola indice per riferirsi al numero tra []):
  - Il primo elemento ha indice 0
  - l'ultimo ha indice dimensione-1 (11 nel nostro esempio)
- Es. printf("secondo elemento di c=%d", c[1])
- c[2] = 1; // il valore del terzo elemento dell'array q destra passa da 0 a 1

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Array: Esempi

- int n[5];
  for (int i = 0; i < 5; i=i+1) { // inizializza a zero gli elementi dell'array
  n[i] = 0;
  }

  for (int i = 0; i < 5; i=i+1) { // stampa gli elementi dell'array
  printf("%d = %d\n", i, n[i]);
  }

- int n[5] = {32, 27, 64, 18, 95}; // dichiara ed inizializza l'array
  int n[] = {32, 27, 64, 18, 95}; // dichiarazione equivalente

Confini di un Array

- Il C non ha meccanismi di controllo dei confini di un array
- Un programma può fare riferimento a un elemento che non esiste e non ricevere un errore!

- int n[5] = {32, 27, 64, 18, 95};

```
for (int i = 0; i < 6; i=i+1) {
    printf("%d = %d\n", i, n[i]);
}
```

- Nell'ultima iterazione si stampa il contenuto della cella di memoria seguente all'array; se siamo fortunati questo genera un errore, altrimenti viene stampato un valore indefinito (859 nell'esempio)

n[0]	32
n[1]	27
n[2]	64
n[3]	18
n[4]	95
	859

Array a Dimensione Fissata

- In C89 la dimensione di un array deve essere una costante intera positiva (no variabile const int) nota a tempo di compilazione (non una variabile, letta da tastiera o meno)
- Di norma la costante si indica tramite #define
  - #define DIM_ARRAY 3
  - int x[DIM_ARRAY] = {1,2,3} //OK
- Dal C99 è possibile dichiarare array la cui dimensione non è fissata a tempo di compilazione (si chiamano VLA, variable length arrays)
- Tali array sono implementati internamente in modo diverso dal C. Una delle differenze è che non possono essere inizializzati
  - int n = 3; int x[n] = {1,2,3}; //NO
  - x[0]=1; x[1]=2; x[2]=3; //OK

Stringhe (Sequence di Caratteri)

- Una sequenza di caratteri che termina con il carattere ‘\0’ (tra apici singoli, non doppi) viene chiamata stringa. Es. char s[] = {‘c’, ‘i’, ‘a’, ‘o’,’\0’} è una stringa, ma anche char s2[] = {‘c’, ‘i’, ‘a’, ‘o’, ’\0’, ’a’, ‘a’} indica la stessa stringa.

- char string1[] = ”ciao mondo"; // il carattere \0 viene aggiunto
  // automaticamente (“” indicano una stringa)

printf(“la stringa inizia per %c\n”, string1[0]); // stampa c

printf("%s\n", string1); // stampa ciao mondo, %s sta per stringa, assume che l’array di caratteri termini con \0

printf("%s\n", s2); // stampa ciao

Array di Caratteri e Puntatori a Stringhe

Una stringa può essere rappresentata da

- un array di caratteri, se si aggiunge '\0' alla fine.
  - La dichiarazione riserva un certo numero di celle di memoria. Quindi Si riesce a modificare un singolo carattere (s[0]='K')
- un puntatore a char, a cui può essere assegnata una stringa costante, della quale non si possono modificare i caratteri. Si può riassegnare una seconda stringa al puntatore.

```
char s[8] = "Hearts";
char *ps = "Hearts";
printf("%s - %s\n\n", s, ps);
s[0] = 'K';
ps = "Ke";
printf("%s - %s\n", s, ps);
--Output--
Hearts - Hearts

Kearts - Ke
```

Passaggio di Parametri in C: Array

- I due prototipi seguenti sono equivalenti ed intercambiabili:

```
void f(int *array);
void f(int array[]);
```

- int x[5];
- x == &x[0] // puntatore al primo elemento dell'array
- int *p = x; // corretto, equivale a &x[0]
- f(x); f(p); // sono corrette per entrambi i prototipi
- Gli elementi di un array vengono sempre modificati all'interno di una funzione! Perché?

Aritmetica dei Puntatori

- Si può sommare o sottrarre un intero ad un puntatore
  - int x, *p = &x;
  - p+2 equivale a p+2*sizeof(int)
- Si può sottrarre due puntatori (ha senso se si riferiscono allo stesso array)
  - p1-p2
- Si possono confrontare due puntatori:
  - p1>p2, p1==p2, p1 != p2
- Non è ammesso utilizzare puntatori in moltiplicazioni o divisioni
  - p1/3, p1/p2, p1*p2
- Non è ammesso sommare due puntatori
  - p1+p2

Array e Puntatori

UNIVERSITÀ DEGLI STUDI DI PADOVA

- int x[] = {1,2,3,4,5}
- quando usato in un'espressione, x è un puntatore **costante** al primo elemento dell'array:
  - x == &x[0]
- int *p = x; // corretto, equivale a &x[0]

//assumendo che L-valore di x sia 1022

&x[0] == p == 1022

&x[1] == p+1 == 1022 + 1*sizeof(int) == 1026

&x[2] == p+2 == 1022 + 2*sizeof(int) == 1030

&x[3] == p+3 == 1022 + 3*sizeof(int) == 1034

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Esempio: Stampa Array


```
int a[] = {1,2,3,4};
int i = 0, *p = a;
    while(i<4) {
        printf("%d\n", *p);
        i+=1;
        p = p+1;
    }
```

Stampa:

1

2

3

4

Esempio: Stampa Array (versioni 2 - 3)


```
int a[] = {1,2,3,4};
int i = 0, *p = a;
while(i<4) {
        printf("%d\n", *(p+i));
        i+=1;
}
// Oppure
while(i<4) {
        printf("%d\n", *(a+i));
        i+=1;
} //ma a+=i sarebbe sbagliato!
```

Entrambi Stampano:

1

2

3

4

Array Come Parametri di Funzioni

- Quando passiamo un array ad una funzione, in realtà passiamo per valore il puntatore al primo elemento dell'array
- È quindi possibile modificare gli elementi di un array passato come parametro ad una funzione

```
void successivo(int ar[], int N) {
    for(int i=0; i<N; i+=1) {
        ar[i] += 1;
    }
}
```

```
int main (void) {
        int a[]={1,2,3};
        successivo(a,3);
        //a={2,3,4}
}
```

Array Come Parametri di Funzioni

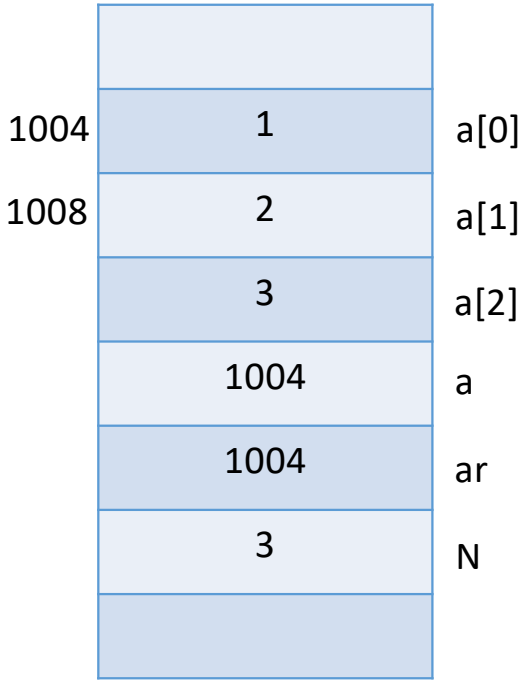

```
void successivo(int ar[], int N) {
  //ar=a; N=3;
    for(int i=0; i<N; i+=1) {
        ar[i] += 1;
    }
}
```

```
int main (void) {
        int a[]={1,2,3};
        successivo(a,3);
        //a={2,3,4}
}
```

- successivo(a,3);

//ar=a

ar[0]+=1 /* modifica la variabile all'indirizzo 1008*/

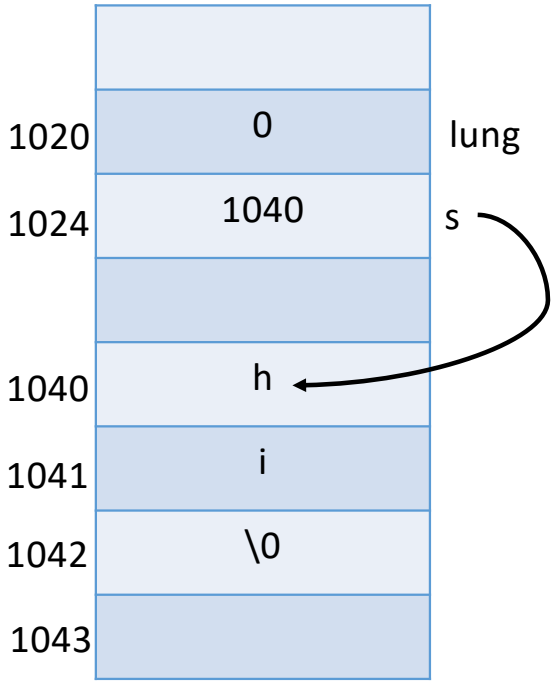


Esempio: Lunghezza Stringa

- Calcolare la lunghezza di una stringa:

```
char *s="hi";
```

Idea: spostare il puntatore s fino alla fine della stringa, un carattere alla volta, contando il numero di spostamenti effettuati.

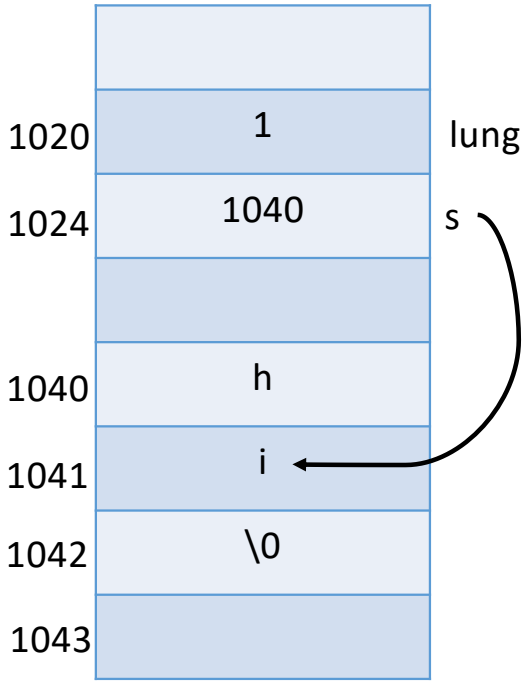


Esempio: Lunghezza Stringa

- Calcolare la lunghezza di una stringa:

```
char *s="hi";
```

Idea: spostare il puntatore s fino alla fine della stringa, un carattere alla volta, contando il numero di spostamenti effettuati.

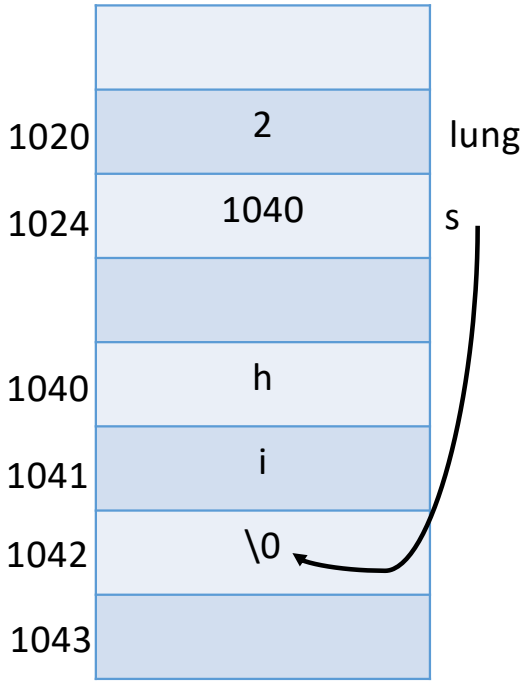


Esempio: Lunghezza Stringa

- Calcolare la lunghezza di una stringa:

```
char *s="hi";
```

Idea: spostare il puntatore s fino alla fine della stringa, un carattere alla volta, contando il numero di spostamenti effettuati.

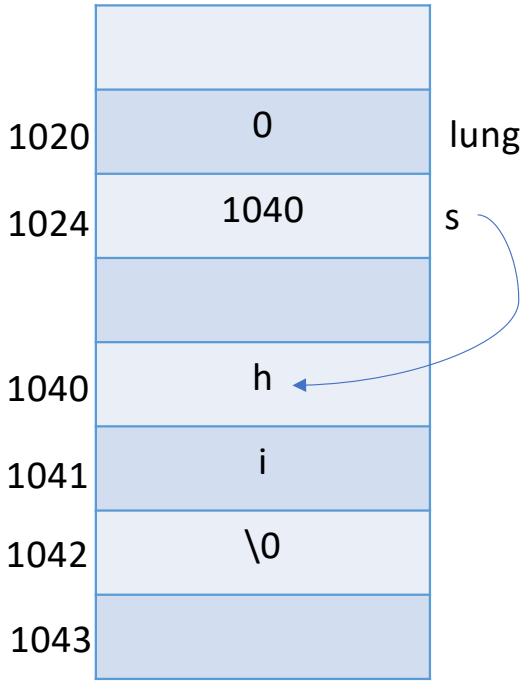


Esempio: Lunghezza Stringa

- Calcolare la lunghezza di una stringa:

char *s="hi";

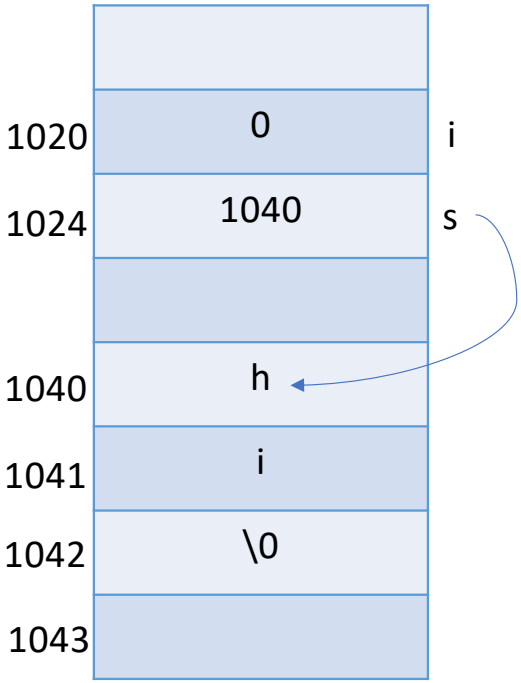
```
int l;
for(l=0; *s!='\0'; s+=1)
        l+=1;
printf("lunghezza stringa: %d\n", l);
```



Esempio: Lunghezza Stringa

- Calcolare la lunghezza di una stringa:

```
char *s="hi";
```



Esercizi

Cosa Stampa?

```c
#include<stdio.h>

int main() {
   int x;
   int y=2;
   int *p, *q = &y;
   int **qq =&p;
   **qq = 3;
   printf("%d\n",**qq);
}
```

Esercizi

Cosa Stampa?

```c
#include<stdio.h>

int main() {
   int y=2;
   int **qq =&&y;
   printf("%d\n", **qq);
}
```

Esercizi

- Vero – Falso: una variabile puntatore ad intero occupa la stessa quantità di memoria di una variabile di tipo puntatore a double.

Esercizi

- Vero – Falso: una variabile puntatore ad intero occupa la stessa quantità di memoria di una variabile di tipo puntatore a double.

Vero: la variabile puntatore ha come R-valore un L-valore, cioè l' indirizzo di memoria della cella dove inizia la rappresentazione del dato puntato, che è indipendente dal numero di celle che serve a rappresentare il dato puntato.

Esercizi

- Cosa stampa il codice seguente?

```
void fun(int* a){
   a[1]=a[1]*2;
   a[2]=a[2]*2;
}
int main(void) {
   int x[]={0,1,2,3,4};
   fun(x+2);
   for(int i=0; i<5; i+=1) {
      printf(" %d", x[i]);
   }
   printf("\n");
}
```

Esercizi

- Cosa stampa il codice seguente?

```
#include <stdio.h>
void quadrato(int x) {
   x = x*x;
}

int main (void) {
   int x[3] = {1,2,3};
   for(int i=0; i<3; i+=1) {
      quadrato(x[i]);
      printf(" %d", x[i]);
   }
}
```

Dichiarazioni di Array

- Le dichiarazioni in C possono essere piuttosto convolute
- Per “risolvere” una dichiarazione si parte dal nome della variabile e ci si sposta a destra/sinistra a seconda della priorità degli operatori
- int a[5]; //a è un array di 5 interi (a è un array di 5 elementi, ciascuno di questi è un intero)

- int *p[4] ?
- int (*p)[4] ?

Dichiarazioni di Array

- int *p[4] ?
- poiché [] ha priorità su *, int *p[4] si legge: p è un array di 4 elementi (ciascuno) int *, ovvero un array di 4 puntatori ad intero

- int (*p)[4] ?
- int (*p)[4] si legge: p è un puntatore ad un array di 4 interi